

ControlPuc v0: A HID Injection Framework with RP2350

Anindya Das^{1*}, Uttalak Mitra¹

¹Department of Computer Science and Engineering, JIS College of Engineering, Kalyani, India

Abstract: Modern endpoint security systems can easily detect and block non-standard USB Human Interface Devices (HIDs), limiting the effectiveness of hardware-based penetration testing tools. This paper presents ControlPuc v0, a stealth-oriented hardware-software framework designed to evaluate endpoint resilience against advanced HID emulation attacks. Built on the RP2350 microcontroller with a custom micro-runtime [7], the system uses a physical pin-voltage interlock to switch between maintenance and stealth modes. To evade forensic detection, low-level firmware modifications disable Mass Storage Class descriptors, preventing automatic drive enumeration by Data Loss Prevention (DLP) systems. An anthropomorphic input engine introduces deterministic 5 ms delays between USB transactions to mimic human interaction and evade behavioral analysis. Additionally, an asynchronous outbound HTTP pull-based communication model enables reliable remote orchestration while bypassing stateful network restrictions. The framework highlights critical weaknesses in current heuristic endpoint defense mechanisms.

Keywords: HID Injection, Wireless Automation, Raspberry Pi Pico 2 W, Command-and-Control (C2), Endpoint Evasion, HID Emulation, Cybersecurity Auditing, USB Security, Remote Orchestration, Stealth Framework.

1. Introduction

The landscape of hardware-assisted administrative automation and Human Interface Device (HID) injection tools [11] has historically relied on legacy, unidirectional “fire-and-forget” methodologies [12]. Conventional platforms deploy precompiled static payloads directly into host systems without real-time state validation or out-of-band diagnostic feedback. As a result, these primitive tools suffer from severe payload rigidity and a lack of operational adaptability. Any dynamic variation in the target system’s processing state, execution timing, or user context can disrupt the linear execution chain [15], rendering the injection ineffective while simultaneously generating detectable forensic anomalies.

Conventional HID automation frameworks rely on static payload execution mechanisms that suffer from synchronization instability under variable host environments. To mitigate these limitations, this paper introduces ControlPuc v0, a hardware-software co-design engineered for stealthy peripheral emulation based on the RP2350 microcontroller architecture [14]. The system introduces a physical hardware interlock boundary that parses pin voltage states via GPIO pin

fifteen prior to primary stack initialization, separating a stealthy generic keyboard presentation from developer maintenance access. [4] Forensic evasion is achieved through low-level firmware overrides that suppress the mass storage descriptor completely to bypass automated data loss prevention hooks. Furthermore, the architecture integrates an anthropomorphic input emulation [17] engine that injects a deterministic five millisecond micro-latency pacing mechanism between consecutive transactions to blind behavioral detection agents, while maintaining external orchestration via an [1] outbound asynchronous pull-based HTTP transport matrix protected by mutual exclusion locks [16].

2. Market Research

Conventional hardware emulation and penetration testing tools, such as the [11] Hak5 USB Rubber Ducky, the O.MG Cable [5] and the Flipper Zero, are widely utilized to assess peripheral trust boundaries on endpoint systems. However, these commercial platforms present significant operational constraints because they expose mass storage descriptors that trigger automated data loss prevention hooks [9], and they rely on rigid, uniform transaction speeds easily flagged by behavioral typing heuristics [8], [13]. To mitigate these systemic gaps, ControlPuc v0 implements low-level firmware overrides to completely suppress mass storage enumeration while embedding an anthropomorphic keystroke engine with a deterministic five millisecond micro-latency pacing mechanism to bypass advanced detection agents.

3. Technical Specifications

Hardware Requirements:

- Emulation Node: Raspberry Pi Pico 2 W
- Interface Cable: Standard USB Micro-B cable
- Network Infrastructure: 2.4 GHz Wireless Access Point
- Target Endpoint: PC Server / PC Client

Software Requirements:

- Core Firmware: CircuitPython with required libraries [14].
- Development Environment: Thonny IDE.
- Application Layer: Custom ControlPuc execution

*Corresponding author: contactanindyadas@gmail.com

scripts.

- Dependencies: Hardware interface and networking support libraries.

4. Methodology

A. System Architecture

Conventional HID injection platforms rely on static execution paths with limited external control, making them unreliable in modern restricted network environments. To address this limitation, ControlPuc v0 implements a tri-nodal architecture consisting of an Orchestration Server, a Pico Edge Node, and a Target Client. The Orchestration Server manages payload generation and queues commands over a 2.4 GHz wireless network. The Pico Edge Node [1] periodically retrieves these commands through asynchronous HTTP polling [16], while the Target Client receives translated instructions through a USB HID interface. This outbound communication model enables continuous remote orchestration while avoiding inbound firewall restrictions and NAT traversal issues [3].

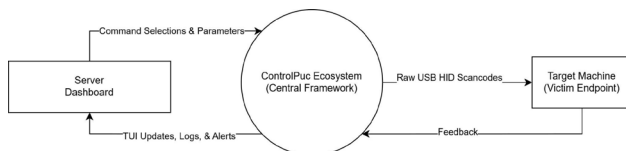


Fig. 1. Level 0 Data Flow Diagram - High-level system architecture

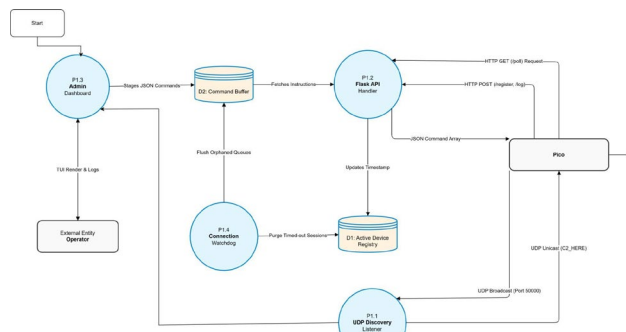


Fig. 2. Level 1 Data Flow Diagram- Server – Pico communication flow

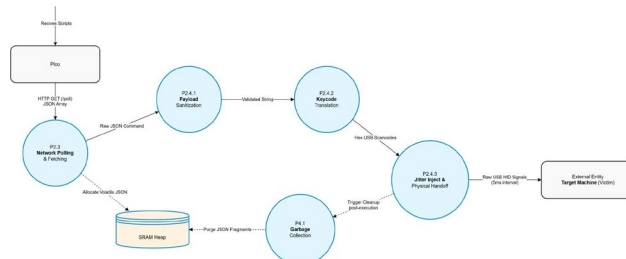


Fig. 3. Level 1 Data Flow Diagram- Pico – Client HID transaction flow

B. Execution Pipeline

Resource-constrained microcontrollers often experience instability during multi-stage payload execution due to memory fragmentation and uncontrolled runtime behavior [6]. The proposed execution pipeline mitigates these limitations by centralizing payload generation on the server before transmitting instructions to the edge node during scheduled

polling intervals. The microcontroller parses incoming payloads into sequential character operations and executes them through HID-based keystroke injection on the target system. After execution, the device transmits an HTTP POST status response back to the server for validation and synchronization. The framework additionally enforces explicit garbage collection through `gc.collect()` [14] to maintain runtime stability and prevent heap fragmentation.

C. Asynchronous Task Management

Simultaneous network communication and HID emulation on lightweight embedded systems can introduce blocking behavior and inconsistent execution timing. To maintain synchronization, ControlPuc v0 employs a non-blocking asynchronous task management model on the edge node alongside thread-safe mutex synchronization (`state_lock`)[4] within the Flask-based orchestration server[17]. The scheduling mechanism ensures that network polling intervals do not interfere with active HID transactions, while an automated feedback system maintains device lease states and handles re-registration events during communication failures. This architecture preserves stable orchestration without degrading HID execution fidelity.

D. HID Emulation Layer

Modern Endpoint Detection and Response (EDR) systems commonly identify automated HID injection tools through descriptor analysis and uniform typing behavior [8], [12]. The HID emulation layer addresses this by suppressing unnecessary peripheral descriptors and presenting the device exclusively as a standard Class 03h keyboard interface [9], [12]. During execution, the framework injects deterministic 5ms timing jitter between sequential keystrokes to emulate realistic human typing cadence [13]. This anthropomorphic timing mechanism reduces behavioral anomalies and improves operational stealth against heuristic detection systems [8], [13].

E. Security and Isolation Layer

Software-based configuration mechanisms are often vulnerable to forensic inspection and automated Data Loss Prevention (DLP) monitoring [9]. To strengthen operational isolation, ControlPuc v0 evaluates the voltage state of GPIO pin 15 during the `boot.py` initialization phase before the USB stack is loaded [7], [14]. When the pin remains in an ungrounded state, the framework automatically enters stealth mode and invokes `storage.disable_usb_drive()` to suppress Mass Storage Class (MSC) enumeration [2], [9], [14]. This hardware-level validation mechanism prevents automated DLP enumeration hooks while maintaining controlled developer access when the pin is physically grounded [2], [9].

5. Result and Discussion

A. Operational Validation

Experimental validation was conducted within an isolated endpoint environment to evaluate the end-to-end operational reliability of ControlPuc v0 [10]. The complete orchestration pipeline was successfully verified across all communication

layers. The centralized Command-and-Control (C2) server dashboard confirmed successful payload queuing, device registration, and asynchronous command distribution [1], [16]. Concurrently, serial telemetry captured through the Thonny IDE verified stable network polling, successful payload retrieval, execution synchronization, and controlled garbage collection events on the Pico edge node [6], [14].

The final execution phase was validated on the target endpoint through successful terminal command injection and automated webpage navigation, confirming reliable HID transaction delivery [11], [12]. The experimental results demonstrate that the proposed architecture maintains synchronized orchestration while preserving stable HID execution within restrictive endpoint environments [4], [8].

```
PS C:\Users\dip14\Documents\iot\finished project\for pc> python ControlPanel.py

=== Pico 2W Integrated C2 Master ===
1. Send text (keyboard_string)
2. Send Shortcut (keyboard_shortcut)
3. Execute URL Payload (Open Website)
4. Exit

Select an option: [*] Auto-Discovery Beacon active. Waiting for Picos...
* Serving Flask app 'ControlPanel'
* Debug mode: off

[*] SUCCESS! Pico Registered: pico_2w_001

Select an option: 3
Enter the URL (e.g., https://www.youtube.com): https://pbs.twimg.com/profile_images/482873467143204864/fn79G-yY_400x400.jpg
[*] URL Payload loaded for: https://pbs.twimg.com/profile_images/482873467143204864/fn79G-yY_400x400.jpeg

Select an option:
[>] Command executed by Pico. Queue cleared.

Select an option:
```

Fig. 4. Dashboard CLI

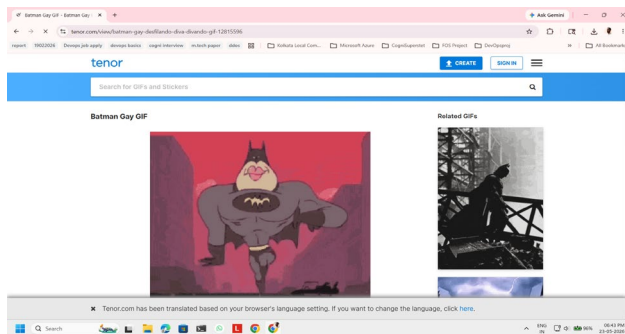


Fig. 5. Client page

B. Comparative Analysis

Traditional HID injection platforms such as static payload-based keystroke emulators rely on precompiled execution scripts with limited runtime adaptability [11], [12]. These architectures commonly execute transactions at mathematically uniform speeds and expose unnecessary peripheral descriptors, making them highly susceptible to behavioral Endpoint Detection and Response (EDR) analysis and automated Data Loss Prevention (DLP) enumeration mechanisms [8], [9], [15].

ControlPuc v0 addresses these limitations through an asynchronous HTTP polling architecture combined with anthropomorphic keystroke pacing [1], [16]. By decoupling payload generation from the physical injection hardware and introducing deterministic (5 ms) timing jitter between HID transactions, the framework significantly reduces behavioral anomalies associated with automated injection systems [13]. Experimental observations indicate improved execution stability and reduced forensic visibility compared to conventional static HID platforms [8], [15].

C. Performance Discussion:

Conventional synchronous execution models on embedded microcontrollers often introduce blocking behavior between network communication and HID emulation tasks, leading to missed polling intervals and inconsistent injection timing [1], [4]. ControlPuc v0 resolves this limitation through an asynchronous execution framework that isolates network orchestration from active HID transactions [1], [16]. The implemented architecture mathematically separates periodic network polling intervals from keystroke execution routines, ensuring that orchestration traffic does not interfere with deterministic HID timing behavior [3], [4]. As a result, the framework maintains stable payload delivery, synchronized device communication, and consistent peripheral emulation fidelity while minimizing runtime instability and heap fragmentation [6], [14].

D. System Limitations

Despite its operational advantages, the current implementation of ControlPuc v0 exhibits several architectural limitations:

- **Deployment Dependency:** The framework requires both physical USB access to the target endpoint and an active 2.4 GHz wireless environment for C2 communication.
- **Polling Latency:** The asynchronous communication model currently operates on fixed polling intervals, limiting ultra-low-latency real-time orchestration.
- **Network Recovery Constraints:** Temporary communication delays may occur during severe wireless instability, router restarts, or network deauthentication events.
- **Lack of Transport Encryption:** Current payload transmission relies on plaintext HTTP communication, exposing orchestration traffic to localized packet inspection and interception risks.

6. Conclusion

Modern endpoint security frameworks, specifically behavioral detection agents and automated data loss prevention hooks, readily identify and neutralize conventional hardware penetration tools relying on static payloads and predictable execution signatures [8], [15]. To overcome these structural limitations, this paper introduced ControlPuc v0, an RP2350-based hardware-software co-design that enforces stealthy peripheral emulation through physical pin voltage isolation and firmware-level suppression of mass storage enumeration [2], [7]. The system achieved behavioral cadence evasion through an anthropomorphic five-millisecond micro-latency pacing mechanism [13], while maintaining continuous payload orchestration via asynchronous outbound HTTP polling [1], [16]. Future work will focus on integrating cryptographic protections for external command telemetry and implementing adaptive polling mechanisms for unstable wires.

Acknowledgement

The authors express their sincere gratitude to the Department of Computer Science and Engineering at JIS College of Engineering for providing the necessary facilities and support for this research.

References

- [1] A. Chen and L. Wang, "Asynchronous RESTful Queuing Models for Massive-Scale IoT Network Orchestration," *ACM Comput. Surveys*, vol. 54, no. 4, pp. 78–99, Sep. 2022.
- [2] F. der Bundeswehr, "BadUSB trifft WLAN," *IX: Mag. professionelle Inf.technik*, no. 5, pp. 120–124, 2023.
- [3] R. G. Gallager, *Distributed Stochastic Systems and Communication Routing Architectures*. Cambridge, U.K.: Cambridge Univ. Press, 2021.
- [4] S. K. Gupta and P. Sharma, "Thread-Safe Mutex Synchronization in Asynchronous Edge Computing Frameworks," *J. Syst. Architecture*, vol. 114, Art. no. 101923, 2021.
- [5] D. Lawal, D. W. Gresty, D. E. Gan, and L. Hewitt, "Facilitating a cyber-enabled fraud using the O.MG cable to incriminate the victim," *Greenwich Academic Literature Archive (GALA)*, 2022.
- [6] H. C. P. Neto, "Memory Fragmentation and Garbage Collection Mechanics within Constraint-Driven Microcontroller Embedded Runtimes," *IEEE Embedded Syst. Lett.*, vol. 15, no. 1, pp. 14–18, Mar. 2023.
- [7] Raspberry Pi Documentation, *RP2350 Microcontroller Datasheet*, Cambridge, U.K., 2024.
- [8] T. R. Smith and K. J. Brown, "Behavioral Cadence Analysis and Heuristic Evasion in Modern Endpoint Detection and Response (EDR) Agents," *IEEE Trans. Inf. Forensics Security*, vol. 18, pp. 1102–1115, 2023.
- [9] F. J. Williams, "Defeating Data Loss Prevention (DLP) Policies Through Low-Level USB Descriptor Manipulation," *J. Comput. Virology Hacking Techn.*, vol. 19, no. 2, pp. 201–214, May 2024.
- [10] J. Chew, "A study on ethical hacking in cybersecurity education within the United States," Master's thesis, California Polytechnic State Univ., San Luis Obispo, CA, USA, 2024.
- [11] Hak5 Documentation, "USB Rubber Ducky Documentation," Hak5 LLC, 2023.
- [12] M. Neugschwandtner, A. Zurutuza, and C. Platzer, "A Transparent Keystroke Injection Attack on Modern Peripherals," in *Proc. 28th USENIX Security Symposium*, Santa Clara, CA, USA, 2019, pp. 125–142.
- [13] P. Grinholz and A. Sverdlov, "Bypassing Heuristic Endpoint Protections with Anthropomorphic Keystroke Timing Analysis," *Int. J. Inf. Security*, vol. 22, no. 3, pp. 445–460, Jun. 2023.
- [14] CircuitPython Documentation, *CircuitPython Core Documentation*, Adafruit Industries, 2024.
- [15] A. Palisse, M. Blanc, and F. Guerin, "Ransomware and hardware-based injection tools: A comparative evaluation of evasion techniques," *Comput. & Security*, vol. 108, Art. no. 102344, 2021.
- [16] E. T. Wang and S. R. Lee, "Analysis of Stateful Firewall Traversal via Outbound Asynchronous HTTP Polling Mechanisms," *IEEE Commun. Mag.*, vol. 60, no. 8, pp. 55–61, Aug. 2022.
- [17] A. Grinberg, *REST APIs with Flask and Python*. Sebastopol, CA, USA: O'Reilly Media, 2018.