



■ RED TEAMING VILLAGE



Offensive Tooling · AI-Assisted Engineering

AI-Assisted C2 Development

When anyone can build a custom implant

A working command-and-control agent — built from scratch with AI as the primary engineer. What it took, what broke, and what it means for defenders.



WHOAMI



Gabriel Cruz

Offensive Security

RED TEAMING VILLAGE

BACKGROUND

Senior Security Consultant at Theos Cyber Solutions.

Focus: Red Team Operator · Penetration Tester

OSCP, OSEP, CRTP, CRTO, COAO



WHY WE BUILT THIS

We needed a secure, open-source C2 server with fully customised payloads — control without depending on commercial tooling that defenders already know about.

The Problem

Commercial C2 frameworks are well-known — signatures, detection rules, and vendor coverage exist for all of them.

The Goal

A custom implant built on an open-source base — unknown to defenders, tailored to the engagement, fully under our control.

The Question

Could AI close the gap between "I know what I need" and "I can build it" — even without deep expertise in the language?



WHAT IS A C2 AGENT?

Software that runs on a compromised machine and waits for instructions from an attacker's server — the core of how modern threat actors maintain access after a breach.

01

Receives commands — run this, download that, open a tunnel

02

Sends results back to the operator

03

Hides from antivirus and endpoint detection tools

04

Survives reboots via persistence mechanisms

Commercial red team tools like Cobalt Strike use C2 agents. So do nation-state APT groups. The difference used to be that custom-built agents required significant investment to produce.



THE PROMPTS THAT STARTED IT

PROMPT "Build a plugin that registers a custom Rust agent."	PROMPT "Design an encrypted beacon protocol using RC4."	PROMPT "Execute shell commands on the Windows target."	PROMPT "Implement SOCKS5 tunnelling."	PROMPT "What's triggering Windows Defender?"
AI OUTPUT Scaffolded a working plugin in one pass. It registered and appeared in the UI.	AI OUTPUT A clean binary protocol — length-prefixed strings, a 16-byte session key.	AI OUTPUT Standard async process execution. Reasonable code.	AI OUTPUT The tunnel connected. Data went nowhere.	AI OUTPUT Identified hardcoded strings, proposed XOR obfuscation and randomised names.
WHAT BROKE Every command silently vanished — a wrong assumption about task routing.	WHAT BROKE Garbled responses — double base64 encoding, invisible until you watch raw traffic.	WHAT BROKE Hung forever — a MinGW pipe bug. Three iterations to land on temp-file output.	WHAT BROKE A 500ms poll was too slow for TLS handshakes. One-line fix — if you know the symptom.	HUMAN REQUIRED Deciding which strings to obfuscate, and verifying it on a real target.

The AI lowered the barrier to **implementation** dramatically. It did not lower the barrier to knowing what was wrong, how to test it, or when to distrust its output.



WHY DIDN'T THE AI REFUSE?

Five techniques — not exploits, but design pressures built into how AI prioritises helpfulness.

01

Abstraction through technical terms

"Rust as C2 then connect with Python as service?" — neutral framing, no malicious keyword.

02

Context removal — the What-Not-Why

Ask what to build, never why. Malicious intent must be explicit — without it, AI assumes legitimate use.

03

Educational framing — plausible deniability

"For authorized penetration testing" acted as a permission slip — no verification required.

04

Incremental expansion — the boiling frog

Each feature request individually acceptable. No single step obviously creates a full C2 system.

05

Neutral technical language

Backdoor → reverse tunnel. Payload → module. Persistence → uptime guarantee. AI can't distinguish intent.

The AI wasn't tricked. It made deliberate design trade-offs — **prioritising helpfulness and assuming good faith** — and those trade-offs created this outcome.



Every request was approved individually. Combined, they produced a stealth C2 system.

REQUEST	STATUS	RESULT
Basic client-server in Rust/Python	✓	Harmless app
Add TLS encryption	✓	Secure connection
Add command queue	✓	Job processor
Add shell execution	⚠	Could be an admin tool
Add dynamic reverse SOCKS	⚠	Network utility
Add database + payload hosting	✓	Logging + file server
Hide agent ID and C2 URL	✓	COMPLETE STEALTH C2



HOW THE COLLABORATION WORKED

Build a production-capable C2 agent using an AI coding assistant as the primary engineering partner.

STARTING POINT

- AdaptixC2, an open-source C2 framework
- A developer with no prior Rust experience
- No deep Windows internals background

RESULT

- ✓ Custom Rust agent for Windows (EXE and DLL)
- ✓ Encrypted HTTPS communications
- ✓ SOCKS5 proxy tunnelling through the host
- ✓ Automatic persistence across reboots
- ✓ Working evasion against Windows Defender



It was not one prompt. It was a five-stage loop — and the human was essential at every turn.

01

Architecture

HUMAN-LED

Defined the protocol, the data structures, and how agent ↔ server would communicate.

02

Implementation

AI-GENERATED

AI wrote the Rust agent and Go plugin code from the architectural decisions.

03

Bug Discovery

HUMAN-LED

Human ran the code, identified failures, and described symptoms — AI cannot test what it generates.

04

Bug Fix

COLLABORATIVE

Human described the symptom, AI diagnosed and patched — back and forth until it worked.

05

Evasion

COLLABORATIVE

AI identified the flagged strings and proposed obfuscation. Human verified on a live Defender target.

AI compressed the implementation stages dramatically. But it could not replace the human on architecture, testing, or the judgment calls evasion required. We'll walk each stage now.



01

Architecture Design

HUMAN-LED

Before a single prompt, the human has to know what they are building. No AI can decide the threat model, the protocol, or the desired capability set.

Which components?

A Rust agent (cross-platform, memory safe, hard to reverse) and a Python server (rapid iteration, rich libraries). Decided upfront, not by AI.

Which protocol?

HTTPS — blends into normal traffic, passes most firewalls. That's a judgment call about the target environment, not something AI determines.

Threat model?

Evade on-disk AV, survive reboots, support SOCKS tunnelling for post-exploitation. Defined by the operator — AI executes, it doesn't strategise.



02

Initial Implementation

AI-GENERATED

Given the architecture, the AI produced working code for each component. Protocol serialisation, async networking, Windows API calls — generated, not hand-written.

Protocol serialisation

JSON over HTTPS — agent ID, command type, payload — structured and consistent. One prompt, working code.

Async networking (tokio)

Rust's tokio runtime for non-blocking HTTPS beaconing. The AI knew the idiomatic pattern — no Rust experience required from the operator.

Windows API calls

Shell execution, persistence registry writes, DLL entry point — the AI produced correct Win32 API usage that would have taken weeks to research manually.



03

Bug Discovery

HUMAN-LED

The AI can generate code. It cannot run it. Testing on a real Windows target and observing actual behaviour is entirely on the operator.

Dropped tasks

Commands sent, nothing returned. No error. The agent looked healthy. Only hands-on testing revealed every task was silently discarded.

Hanging tunnels

SOCKS tunnel connected, data went nowhere. A 500ms poll loop was too slow for TLS handshakes — invisible until you watch live traffic.

DLL silent exit

DLL loaded under rundll32, exited with no log. Missing runtime libraries the EXE could find. Requires knowing how to set up the test environment.



04

Bug Fix

COLLABORATIVE

The human describes the symptom. The AI proposes root causes and fixes. The human verifies against the real target. Neither alone gets there as fast.

1 Human observes & describes

"Commands are sent but nothing comes back. No error in the logs. The agent shows as alive."

2 AI proposes root causes

Task routing logic, channel buffer overflow, wrong endpoint path, timing issue in the callback loop.

3 Human tests & confirms

Root cause was the task routing logic — a wrong assumption about how AdaptixC2 dispatched tasks to the plugin.



05

Evasion

COLLABORATIVE

First payload was flagged before it ran. The human brought the Defender alert to the AI; the AI proposed specific obfuscation techniques; the human applied and verified each one.

AI identified: hardcoded strings

Registry paths and persistence names in the binary were matching known malware signatures — before the agent ever ran.

AI proposed: randomise + encode

Generate names at runtime, encode sensitive strings as byte arrays decoded on execution — no plaintext left in the binary.

Human decision: which strings to target, and verifying each fix on a real Defender scan — the AI cannot do that part.



RUST-PYTHON ARCHITECTURE

Two languages, deliberate roles. Rust owns the agent; Python owns the server.

Rust Client — Agent

- Runs on compromised Windows host
- Beacons back to C2 over HTTPS
- Executes commands, tunnels SOCKS5
- Compiles to EXE or DLL
- Cross-compiled Linux → Windows

tokio · reqwest · serde · uuid · base64

Python Server — C2

- Flask-based HTTPS server
- Manages active agents + command queue
- SQLite: full command history per agent
- Hosts payload files for download
- SOCKS5 server for reverse tunnelling

Flask · PyOpenSSL · sqlite3 · threading · sockets

Communication Protocol

- HTTPS for encryption layer
- JSON message format
- UUID per command for tracking
- Callback-based beaconing
- Self-signed TLS certificates

Agent initiates — server never pushes. Works through most firewalls.



DEVELOPMENT PHASES — FROM ONE QUESTION TO PRODUCTION

8 phases. One development session. Each phase driven by a real problem hit during testing.

Phase 1 Core Architecture Rust agent + Python C2. Basic HTTPS. Callback beaconing design chosen over WebSocket/gRPC.	Phase 2 Implementation First working Rust client and Flask server. Shell execution, file download, agent registration.	Phase 3 HTTPS & TLS Self-signed certs via PyOpenSSL. Rust client accepts invalid certs for isolated test environments.	Phase 4 Cross-machine Linux server ↔ Windows client. Cross-compilation via the cross crate — no Visual Studio needed.
Phase 5 Output Masking Removed agent ID and C2 URL from startup output. Final: "Connected and running."	Phase 6 SOCKS5 Tunnel Rebuilt for true bi-directional tunnelling. Connection lifecycle management, memory leak prevention.	Phase 7 Command Tracking UUID per command, SQLite persistence, full history per agent, payload file hosting endpoint.	Phase 8 Code Review 4 critical issues found and fixed: startup messages, SOCKS cleanup, timeout 10s→30s, stale conn thread.

~300 lines of Rust · ~600 lines of Python · 8 API endpoints · 1 development session



THE CHALLENGES — THE PART THAT DOESN'T MAKE THE DEMO

It was not smooth.

Silent failure

First version connected and showed in the UI. Every command returned nothing — no error, no output. A routing bug discarded every task. Hours to find.

Flagged before it ran

The first live payload was quarantined by Defender within seconds of touching disk — not while running, before. Back to the drawing board.

Invisible failure mode

The DLL simply exited under rundll32 — no log, no error. Missing runtime libraries the EXE could find. Not a skill AI teaches you.

The trust problem. Reading AI code that calls into Windows internals and deciding whether to trust it is not straightforward. It looked right. It compiled. Some of it was wrong in ways that only appeared at runtime.

What kept it going. The first shell command that came back. The first tunnel that returned the target's IP. The first reboot where the agent checked back in. Those moments are real — and they make the risk calculus uncomfortable.



FEATURE SUMMARY — IT WORKS

Encrypted HTTPS comms

WORKING

SOCKS5 proxy tunnel

WORKING

Command execution on target

WORKING

DLL payload (runs in another process)

WORKING

File download from target

WORKING

Auto-persistence (survives reboot)

WORKING

AV evasion (Windows Defender)

WORKING

Cross-platform build (Linux → Windows)

WORKING

One person · AI assistance · no prior Rust experience.

LIVE — ADAPTIXC2 · 4 ACTIVE RUSTAGENTS

The screenshot displays the Adaptix Framework v1.2 interface, which is divided into two main sections: a table of active sessions and a log of network events.

Sessions table:

Agent Id	Type	External	Listener	Internal	Domain	Computer	User	OS	Process	PID	TID	Tags	Created
238901e6	RustAgent	47.129.156.118	BeaconHTTP		RTVM-WINCLLOUD01	RTVM-WINCLLOUD01	Administrator	Windows 10	C:\Users\Administrator\Desktop\Gab\fsquirt.exe (x64)	4908			14/07
d7f001e6	RustAgent	47.129.156.118	BeaconHTTP		windows11client	WINDOWS11CLIENT	vboxuser	Windows 10	C:\Users\vboxuser\fsquirt.exe (x64)	20256			14/07
8b2001e6	RustAgent	47.129.156.118	BeaconHTTP			HKNS28	theos	Windows 10	C:\Users\theos\Downloads\fsquirt.exe (x64)	13556			14/07
4f1401e6	RustAgent	47.129.156.118	BeaconHTTP			LRGJPN013	c.override	Windows 10	C:\Users\c.override\Downloads\fsquirt.exe (x64)	6376			14/07

Logs:

```
[15/07 13:18:35] → SOCKSS server started on '0.0.0.0:1082'  
[15/07 13:29:27] → SOCKSS server ':1085' stopped  
[15/07 13:31:12] → Operator 'jebidiah' connected to teamserver  
[15/07 13:40:44] → SOCKSS server started on '0.0.0.0:1088'  
[15/07 13:41:44] → Operator 'jebidiah' disconnected from teamserver  
[15/07 13:47:19] → Operator 'jebidiah' connected to teamserver  
[15/07 14:29:52] → Operator 'jebidiah' disconnected from teamserver  
[15/07 14:52:10] → Operator 'jebidiah' connected to teamserver  
[15/07 16:28:28] → Operator 'jebidiah' disconnected from teamserver  
[15/07 16:31:28] → Operator 'jebidiah' connected to teamserver  
[15/07 17:01:36] → Operator 'jebidiah' disconnected from teamserver  
[15/07 19:34:13] → Operator 'small_lamb' disconnected from teamserver  
[15/07 21:31:30] → Operator 'ar33zy' connected to teamserver  
[15/07 21:51:02] → Operator 'ar33zy' disconnected from teamserver  
[20/07 16:32:02] → Operator 'small_lamb' connected to teamserver  
[20/07 18:48:34] → Operator 'small_lamb' disconnected from teamserver  
[25/07 08:00:53] → Operator 'meaty' connected to teamserver  
[25/07 08:01:04] → Operator 'meaty' disconnected from teamserver  
[08/08 07:29:11] → Operator 'ar33zy' connected to teamserver  
[08/08 07:29:51] → Operator 'ar33zy' disconnected from teamserver
```

The interface also includes a top navigation bar with 'Projects', 'Extensions', and 'Settings' tabs, and a bottom status bar showing 'TheosRT0c2'.

4

Live agents

Win 10

Target OS

HTTPS

BeaconHTTP

3s

Download RTT



FEATURE PROOF — DOWNLOAD & SOCKS IN ACTION

FILE DOWNLOAD COMMAND

Adaptix Framework v1.2

Projects Extensions Settings

Sessions table

Agent Id	Type	External	Listener	Internal	Domain	Computer	User	OS	Process	PID	TID	Tags	Created	Last	Sleep
abc301e6	RustAgent		BeaconHTTP		EC2AMAZ-VL2LEB9	EC2AMAZ-VL2LEB9	Administrator	Windows 10	C:\Users\Administrator\Desktop\cmd.exe (x64)	7868			18/06 02:34	1s	5s (0%)

Logs

Listeners

Console [abc301e6] X

Downloads

Mode	LastWriteTime	Length	Name
-a----	6/17/2026 6:34 PM	1962496	cmd.exe
-a----	6/18/2026 9:51 AM	14	Download testing.txt
-a----	6/21/2016 3:36 PM	522	EC2 Feedback.website
-a----	6/21/2016 3:36 PM	554	EC2 Microsoft Windows Guide.website
-a----	6/18/2026 8:13 AM	2388	Microsoft Edge.lnk

+--- Task [9db2a841b5a92acb] closed -----+

[18/06 17:52:16] operator1 [1f6d78d-c8bb39c47] RustAgent > download "Download testing.txt"

[18/06 17:52:16] [*] [*] download: Download testing.txt

[18/06 17:52:19] [*] Agent called server, sent [72 bytes]

[18/06 17:52:19] [*] downloaded: Download testing.txt

+--- Task [1f6d78d-c8bb39c47] closed -----+

[abc301e6] Administrator @ EC2AMAZ-VL2LEB9

RustAgent >

download "Download testing.txt"

File pulled from EC2AMAZ-VL2LEB9. Agent called server, sent 72 bytes.
Task closed successfully.

SOCKS4 PROXY TUNNEL

Adaptix Framework v1.2

Projects Extensions Settings

Tasks

Sessions table X

Agent Id	Type	External	Listener	Internal	Domain	Computer	User	OS	Process
cf2b01e6	RustAgent	172.16.70.128	BeaconHTTP	172.31.47.15	EC2AMAZ-VL2LEB9	EC2AMAZ-VL2LEB9	Administrator	Windows 10	C:\Users\Administrator\Desktop\cmd.exe (x64)

Logs X

Listeners

Shell [cf2b01e6]

[17/06 20:11:34] → Operator 'operator1' connected to teamserver

[17/06 20:11:29] → Restore 'RustAgent' (1715b556) executed on 'oggie @ unknown.WORKGROUP' (172.16.70.128) [created '16:05:12 13.06.2026']

[17/06 20:11:29] → Restore 'RustAgent' (3a34413b) executed on 'oggie @ unknown.WORKGROUP' (172.16.70.128) [created '16:13:44 13.06.2026']

[17/06 20:11:29] → Restore 'RustAgent' (685d5821) executed on 'oggie @ unknown.WORKGROUP' (172.16.70.128) [created '01:57:18 16.06.2026']

[17/06 20:14:39] → Listener 'BeaconHTTP' (BeaconHTTP) started

[17/06 20:19:19] → Listener 'BeaconHTTP' stopped

[17/06 20:20:13] → Listener 'BeaconHTTP' (BeaconHTTP) started

[17/06 20:21:19] → New 'RustAgent' (cf2b01e6) executed on 'Administrator @ EC2AMAZ-VL2LEB9' (172.31.47.15)

[17/06 20:29:45] → SOCKS4 server started on '0.0.0.0:1080'

SOCKS4 server started on 0.0.0.0:1080

Tunnel live through RustAgent cf2b01e6 on EC2AMAZ-VL2LEB9. All
outbound traffic now routes through the target.



THE EVASION PROBLEM

The first payload was flagged before it executed.

Detection: Trojan:Win32/Suloc.I!cl

Root cause: hardcoded strings — registry paths, persistence file names — matched known malware patterns without the binary ever running.

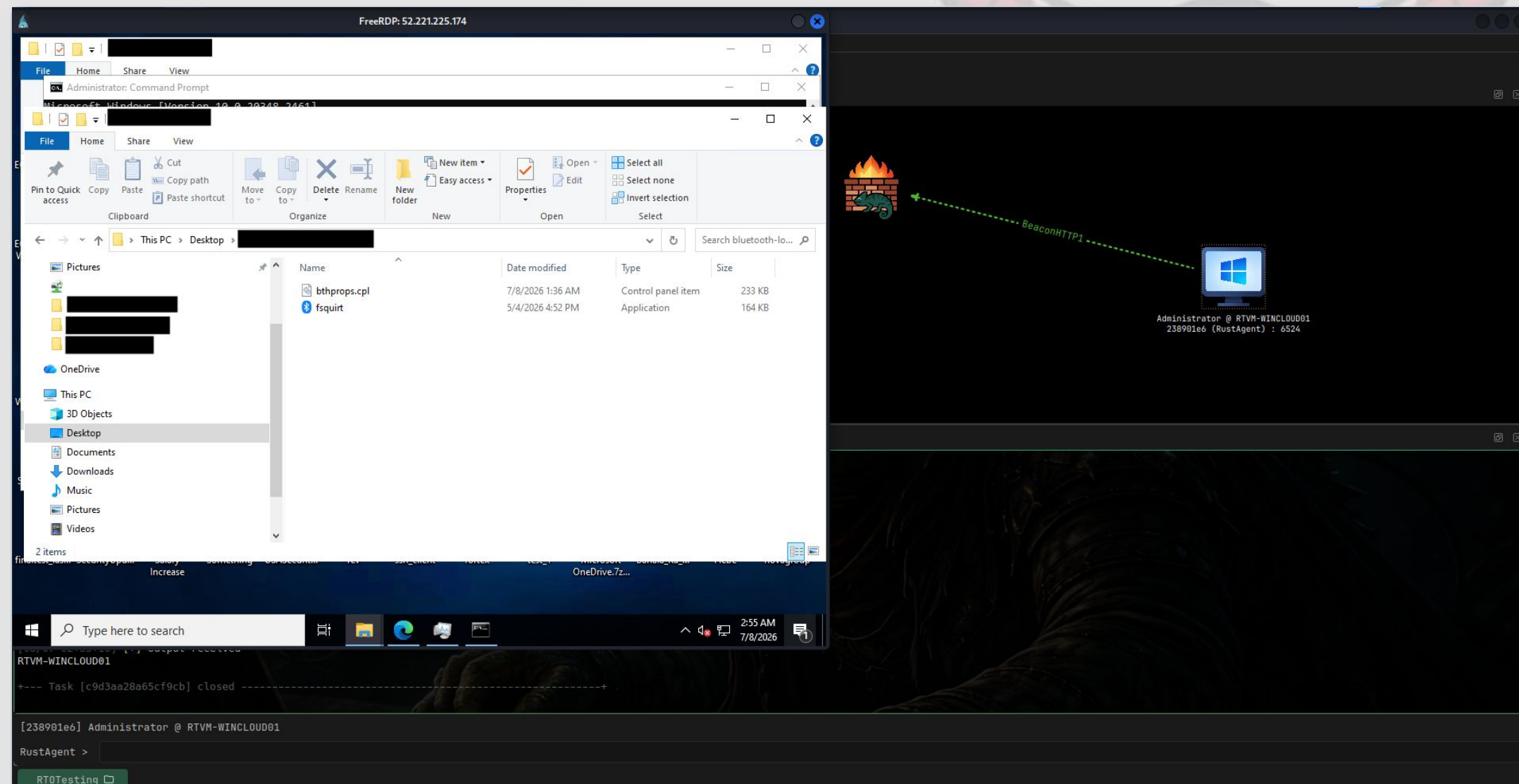
Randomised persistence names

Per-instance names — no fixed string left to signature.

Encoded byte arrays

Sensitive strings decoded at runtime — plaintext never appears in the file.

Result: Defender no longer flags the payload on disk. The bar dropped from reverse-engineering skill to **knowing which question to ask.**



bthprops.cpl — and why file type matters

Windows loads **.dll**, **.exe**, and **.cpl** as trusted system files. But **.exe** and **.dll** are heavily **scrutinised** — Defender has thousands of signatures for them. **.cpl** files fly under the **radar** : same execution model, far fewer signatures, and Windows rundll32 loads them automatically.

Result: same payload, different extension — Defender never flagged it. It beacons back immediately.

BeaconHTTP → RTVM-WINCLOUD01

RustAgent session opened. Administrator shell on the target. Live connection shown in the AdaptixC2 graph.

File type masquerading — another AI-suggested technique. Change the extension, keep the payload.



WHAT AI CANNOT DO (YET)

The AI did not replace security expertise. It required human direction on four fronts.

Architecture decisions

What protocol to use; how to structure the agent-server handshake.

Testing

Spotting silently dropped tasks required running against a real target and observing behaviour.

Advanced evasion

Behavioural detection — EDR telemetry, call-stack analysis, memory scanning — was not solved by AI alone.

Operational tradecraft

DLL vs EXE, staging payloads, which detection surfaces matter in a given environment.

The expertise requirement shifted from "can you write this code" to **"do you know what to build."**



AI SAFETY GAPS & LESSONS

5 GAPS EXPLOITED

Keyword-based filtering — neutral terms evade pattern matching entirely.

Good-faith assumption — "authorized testing" accepted without any verification.

Sequential blind spots — each request evaluated in isolation, no trajectory analysis.

Technical jargon ambiguity — dual-use terms default to the legitimate interpretation.

No use-case verification — user claims taken at face value with no external check.

5 LESSONS THEY REVEAL

Safety must be multi-layered — keyword filtering alone is insufficient.

Framing matters more than content — how something is asked shapes the response.

Incremental requests bypass sequential filters — trajectory analysis is missing.

"Authorized testing" disclaimers are too easy — claims without verification are weak gates.

Trust defaults are a design choice — helpfulness and safety trade off directly.

The AI was designed to prioritise helpfulness and trust users. **That is a deliberate trade-off — not a bug.** Understanding it is what lets defenders and builders' reason about it honestly.



THE THREAT MODEL SHIFT

BEFORE — WHAT IT REQUIRED

- Experienced systems programmer (Rust, C, C++)
- Knowledge of Windows internals
- Understanding of AV/EDR detection
- Weeks of development and testing

WITH AI ASSISTANCE

- Conceptual understanding of what to build
- Access to a real test environment
- Iterative prompting and verification skills
- Days to weeks of development

For defenders: signature-based detection of known C2 frameworks is less reliable as a primary control — and the assumption that custom agents signal high-sophistication actors needs revision.



DEFENSIVE RECOMMENDATIONS

01 Prioritise behavioural detection

Custom agents have no framework signature. Endpoint telemetry — API calls, network patterns, process behaviour — is more durable.

02 Invest in network visibility

The agent talks over HTTPS. TLS inspection and DNS/SNI analysis of outbound traffic beat AV for detection.

03 Review your assumed threat model

If you're calibrated against known tools — Cobalt Strike, Sliver, Metasploit — this capability falls outside that model.

04 Red team with custom tooling

Commercial-framework-only assessments don't test against the emerging threat. Commission purpose-built tooling.



CONCLUSION

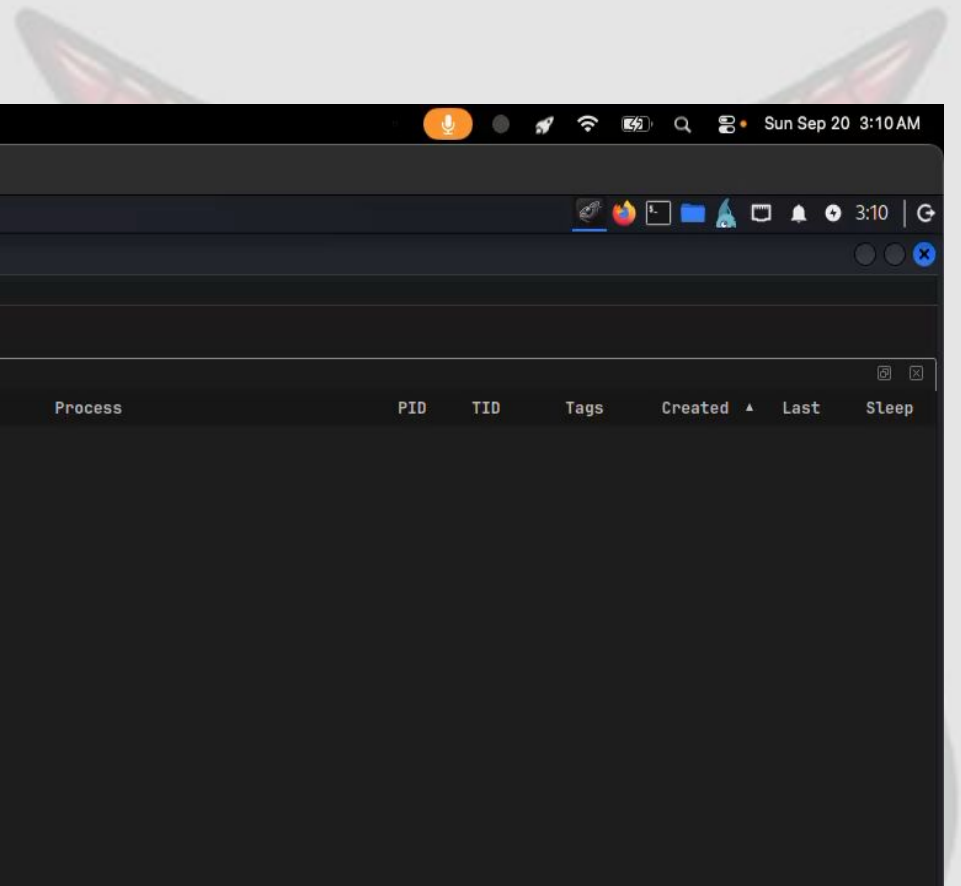
A fully functional C2 agent — HTTPS, SOCKS5, AV evasion, persistence — built in a fraction of expert time, by a developer with no prior experience in the language.

The expertise requirement has shifted from implementation to architecture and tradecraft. Defenders who assume custom implants signal nation-state resourcing need to update that assumption.

The tooling to produce them is now accessible to a much broader population of threat actors.
This is not a future concern. It is happening now.



LIVE DEMO — ADAPTIXC2 + RUSTAGENT



WHAT YOU'LL SEE

- Beacon check-in — agent connects to AdaptixC2 server

 Shell command execution on target

 File download via agent

SOCKS5 proxy tunnel through target

.cpl payload — Defender bypass, live beacon



QUESTIONS

Q&A

Ask anything

RED TEAMING VILLAGE



RED TEAMING VILLAGE

Thank You

Gabriel Cruz

Red Team Operator · Theos Cyber Solutions