

PHANTOMPULSE: anatomy of a hijackable blockchain-C2 RAT

 elastic.co/security-labs/blockchain-c2-phantompulse-rat-sinkhole

Salim Bitam

May 22, 2026



AI generated strings in the binary

Elastic Security Labs presents a detailed reverse-engineering analysis of PHANTOMPULSE, the long-lived RAT delivered to crypto-sector victims through the REF6598 intrusion set.

 18 min read  [Malware Analysis, Threat Intelligence](#)

Elastic Security Labs's prior coverage of [REF6598](#) documented an intrusion set whose Windows toolchain landed via Obsidian plugin abuse, escalated via an in-memory PE loader (PHANTOMPULL), and finished with a RAT (PHANTOMPULSE). That post focused on delivery. This post analyzes the final stage: PHANTOMPULSE, an implant that ships three process-injection techniques, resolves its C2 through Ethereum/Base/Optimism transaction inputs, and bypasses UAC via the public schuac technique. The analysis surfaces a sinkhole-able blockchain C2 channel, a unified hardware-breakpoint primitive that disables AMSI / WLDP / ETW and pervasive AI-assisted-development fingerprints in the implant's debug strings.

Key takeaways

- PHANTOMPULSE implements three injection techniques adapted from recent public offensive-security PoCs.
- AMSI, WLDP, and ETW are bypassed via a single shared HWBP primitive
- The blockchain C2 resolver has **no sender verification**, allowing a defender to override the C2 URL for every implant by posting a single transaction
- Strong AI-assisted-development indicators present in the binary

A note on AI-assisted development

PHANTOMPULSE bears strong fingerprints of AI coding assistance, visible throughout the debug strings.

The clearest tells:

- **Structured step numbering** in operational logs: [STEP 1] Staged mode – payload downloaded from C2 at runtime, [STEP 1/3] Scheduled Task (DotNetSvcUpdateTask, logon + every 3 min), [STEP 2/3] Boot Task (DotNetSvcCoreTask, INTERACTIVE_TOKEN + BootTrigger), [UNINSTALL 4/6] Removing persist_loader DLL + registry PE data..., [REPAIR] Reinstalling boot task (INTERACTIVE_TOKEN)....
- **ENTER/DONE function tracing**: "[HEIS] encrypt_text_only ENTER" / "[HEIS] encrypt_text_only DONE", "KeylogResolveAPIs: ENTER". The diagnostic style LLMs default to when generating new functions.
- **Verbose diagnostics**: "FindHostProcessEx: scan stats: total=%lu sessSkip=%lu openFail=%lu native=%lu wow64=%lu mapReject=%lu dbgReject=%lu sess=%lu", "ManualMap: thread hijacked and resumed – DLL injection via thread hijack complete". Self-explanatory output, unusually talkative for malware.
- **Em dashes in C strings**: "elevate: FAIL – no deployed DLL path", ">>> .elevate: NOT proxy – spawning trusted host to handle elevation".

 AI generated strings in the binary

AI generated strings in the binary

Execution chain

MainEntryLogic is the orchestration function that runs the full initialization sequence before entering the C2 loop:

start

└─ WinMain

```

└─ MainEntryLogic

    1. DynInit          // Bootstrap API resolution

    2. ElevationStateCheck // ".elevate" marker detection, routes by token elevation state

    3. SingleInstanceCheck // XOR-decrypted mutex, exit if already running

    4. EvasionInit      // Direct syscalls + ETW HWBP

    5. SyscallResolverInit // CPUID + hash-based kernel32 resolution

    6. SleepMaskInit    // Sleep obfuscation setup

    7. ComputeMachineID // DJB2(module name) ^ volume serial

    8. IsRunningHollowed // Process hollowing self-check

    9. CollectSysInfo   // CPU, GPU, RAM, OS, AV, apps

    10. FilelessPersist // Drop stub DLL, registry artifact

    11. InstallPersistence // Three scheduled tasks via COM ITaskService

    12. C2Loop_Init → C2Loop_Main

```

At startup, the implant DJB2-hashes the user name and computer name and looks each up in a precomputed table. A match exits the process. Brute-forcing the table against public anti-sandbox wordlists recovered 20 of the 61 entries: WDAGUtilityAccount (Windows Defender Application Guard), several DESKTOP-XXXXXXX default-VM names, and the [Joe Sandbox personas](#) (abby, patex, george, john, lisa, frank, RDhJ0CNFevzX).

Defense evasion

Direct syscalls and API wrappers

PHANTOMPULSE resolves ntdll functions by walking PEB→Ldr with DJB2 hashes, extracts System Service Numbers (SSNs) from each NT function's prologue, and builds private syscall stubs. These stubs are wrapped in higher-level helpers used throughout the rest of the implant:

- NtCreateFile_Wrap
- NtWriteFile_Wrap
- NtClose_Wrap
- NtCreateSection_Wrap
- NtMapViewOfSection_Wrap
- NtProtectVirtualMemory_Wrap
- NtWriteVirtualMemory_Wrap

The rest of the implant calls these wrappers instead of kernel32/ntdll exports, defeating user-mode ntdll hooks (IAT replacements, inline detours, or trampoline patches) that EDR products inject into the documented API surface.

A single helper function routes every disk write through NtCreateFile + NtWriteFile directly, with delete-and-retry on access errors.

String and config obfuscation

PHANTOMPULSE uses four XOR layers for different artifacts:

What	Key	Where the key lives
C2 fallback URL, mutex, drop-path filenames	16-byte: F7 7C 8E 40 DF C1 7B E5 E7 4D 86 79 D5 B3 53 41	Embedded in .rdata

What	Key	Where the key lives
Blockchain provider hostnames (UTF-16 LE)	8-byte: 5A 3C 7E 1D 9F 2B 4E 8A	Embedded in .rdata
COM Elevation Moniker, keylog file payload	0xE95CA237, computed at runtime to keep the constant out of .rdata	Computed, not stored
C2 URL pulled from blockchain transaction input	The resolver wallet address itself	Reused from the public lookup key

 String dexoring stub
String dexoring stub

AMSI, WLDP, and ETW bypass via hardware breakpoints

PHANTOMPULSE disables AMSI, the Windows Lockdown Policy code-trust check, and ETW telemetry through a single shared primitive: a hardware breakpoint planted on each API entry, intercepted by a vectored exception handler that fakes the return value without inline patching.

Slot	Target API	Spoofed return (RAX)
DR0	WldpQueryDynamicCodeTrust	0 (S_OK)
DR1	AmsiScanBuffer	0x80070057 (E_INVALIDARG)
DR2	EtwEventWrite	0 (STATUS_SUCCESS)

 Hardware breakpoint setup function
Hardware breakpoint setup function

The mechanism, step by step:

1. The implant resolves the target API. AMSI and WLDP go through LoadLibraryA + hash-based export lookup; ETW uses a PEB→Ldr walk since ntdll is already loaded.
2. The HWBP descriptor (target API address, mode, spoofed return value) is written into one of four 40-byte slots in a global slot table.
3. A helper thread suspends the target thread, calls NtGetContextThread / NtSetContextThread to write DR0–DR3 + DR7, then resumes. (If the implant's vectored exception handler is already installed, an in-process STATUS_BREAKPOINT is raised instead, letting the VEH read the slot table and program the DRs without a helper thread.)
4. When the protected API is called, the CPU raises Debug Exception on the function's first instruction.
5. The implant's vectored exception handler intercepts the Debug Exception, walks its 4-slot table to find the firing address, and modifies the thread context: CONTEXT.Rax is set to the per-slot spoofed return value, CONTEXT.Rip is redirected to a pre-stored "skip" thunk that returns to the caller.
6. The handler returns EXCEPTION_CONTINUE_EXECUTION. The caller sees the spoofed RAX as if the API had run.

 Exception handler: BREAKPOINT STATUS
Exception handler: BREAKPOINT STATUS

The dispatcher serves two paths. One handler (VEH_Dispatcher) processes both the implant's own RaiseException(STATUS_BREAKPOINT) calls (used to seed and re-program the DR registers from the slot table) and the STATUS_SINGLE_STEP exceptions that fire when a protected API is called. The exception code drives the branch: STATUS_BREAKPOINT triggers DR programming, STATUS_SINGLE_STEP triggers the spoof.

The handler is also not registered directly. AddVectoredExceptionHandler receives a tiny JMP thunk allocated at runtime in a fresh MEM_PRIVATE page (VirtualAlloc + VirtualProtect to PAGE_EXECUTE_READ). The thunk is a JMP [RIP-relative] indirect

jump (6-byte opcode FF 25 00 00 00 00) followed inline by the dispatcher's address. Because no bytes are ever written to AmsiScanBuffer, WldpQueryDynamicCodeTrust, or EtwEventWrite, signature-based detection that scans for prologue patches misses this entirely.

Build variant: active and dormant subsystems

Several subsystems exist in the binary as code or strings, but are not active in this build. This is a stripped-down build of a larger codebase.

- **NTDLL unhooking**: Debug strings for an unhooking subsystem live in .rdata (UnhookNtdll: ntdll base = %p, applied %d relocation fixups to .text), but nothing references them. Dead in this variant.
- **Registry-resident PE blob loader**: earlier builds stored the next-stage PE inside the registry. This build does not, but the uninstall routine still cleans up the legacy registry blob.
- **COM hijack persistence**: never installed by this build. Cleanup logic for it remains in the uninstall routine.
- **Print monitor persistence**: same pattern as COM hijack; install path absent, uninstall path retained.

The last three (registry blob loader, COM hijack, print monitor) show the opposite pattern: cleanup logic with no install logic, retained for backward compatibility against older deployments.

Feature	payloads build	Evidence
Direct syscalls (SSN extraction)	Active	SSN extraction + stub generation confirmed
AMSI / WLDAP / ETW HWBP bypass	Active	DR0/DR1/DR2 via shared helper-thread primitive
Three-way process injection	Active	PhantomInject, DbgNexum, ManualMap are all functional
Blockchain C2 resolution	Active	Three Blockscout providers queried
NTDLL unhooking	Dead code	Strings present, zero code references
HEIS encryption	Disabled	Code encrypt/decrypt stubbed
Registry-resident PE blob loader	Legacy only	Only cleaned during uninstall
COM hijack persistence	Legacy only	Cleaned during uninstall, never installed
Print monitor persistence	Legacy only	Cleaned during uninstall, never installed
Decoy strings	Active	4 unreferenced decoy strings in .rdata

Command and control

Blockchain C2 resolution

PHANTOMPULSE decentralizes C2 lookup through three Blockscout providers:

- eth.blockscout[.]com (Ethereum L1)
- base.blockscout[.]com (Base L2)
- optimism.blockscout[.]com (Optimism L2)

The wallet address 0xc117688c530b660e15085bF3A2B664117d8672aA is XOR-decrypted from storage with a 16-byte key. For each provider, the implant issues an HTTPS GET (port 443, SSL cert errors ignored), pulls the input field of the latest transaction, hex-decodes it, XOR-decrypts with the wallet address bytes as the key, and validates that the result begins with http. On total failure, it falls back to the hardcoded URL https://panel.fefea22134[.]net.

The resolver does not verify the sender of the transaction. It only checks that the latest decoded input starts with http. Anyone can submit a transaction to that wallet with their own URL XOR-encoded under the wallet bytes, and every PHANTOMPULSE instance of that campaign that polls afterward resolves to that URL. For network defenders, this is a viable sinkhole at the cost of one transaction.

Endpoints and heartbeat

Five API paths are constructed at runtime, re-encrypted in memory with a per-session key:

Path	Method	Content-Type	Purpose
/v1/telemetry/report	POST	application/json	Heartbeat with full system telemetry
/v1/telemetry/tasks/<machine_id>	GET		Command fetch
/v1/telemetry/upload/	POST	image/bmp	Screenshot / file upload
/v1/telemetry/result	POST	application/json	Command result delivery
/v1/telemetry/keylog/	POST	text/plain	Keylog data upload

The heartbeat sends a full system profile as JSON:

```
{
  "machine_id": "<uint32>",
  "status": "online",
  "cpu": "<model>",
  "gpu": "<description>",
  "ram_mb": "<uint32>",
  "os": "<version>",
  "username": "<user>",
  "computer_name": "<name>",
  "cores": "<uint32>",
  "screen_w": "<int>",
  "screen_h": "<int>",
  "privilege": "<user|admin|admin_nouac|system>",
  "build": "payloads",
  "public_ip": "<ip>",
  "av_list": ["<av1>", "<av2>"],
  "apps": ["<app1>", "<app2>"],
  "last_cmd": "<cmd>",
  "last_cmd_result": "<result>"
}
```



Building heartbeat JSON document

Two response fields are parsed: "status": "deleted" triggers full uninstall; "ip": "<value>" populates the public-IP cache, but only if local discovery (ipiff[.]org / icanhazip[.]com/ checkip.amazonaws[.]com) hasn't already filled it.



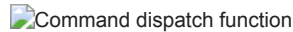
Parsing heartbeat response

Loop cadence and resilience

- **Sleep**: uniform random in [20, 40] seconds
- **Self-healing**: runs at iteration 2, then every 10th iteration
- **Health-monitor tick**: first call after the implant comes online, then every 5th iteration thereafter. Populates a local system-info struct (CPU%, RAM, OS version, uptime, computer name).
- **Failure threshold**: 10 consecutive heartbeat failures trigger a self-restart for stuck SSL/TLS recovery
- **Re-resolution**: on failure, blockchain re-resolution runs; if the resolved URL changes, the failure counter resets
- **Public IP**: api4.ipify[.]org → ipv4.icanhazip[.]com → checkip.amazonaws[.]com
- **Connectivity check**: probes microsoft[.]com, google[.]com, cloudflare[.]com, github[.]com

Command dispatch

The command dispatcher routes commands by DJB2 hash. Eight commands total:



Command dispatch function

Hash	Command	Behavior
0x04CF1142	inject	Inject shellcode/DLL/EXE. Routes by type: shellcode → PhantomInject; DLL → ManualMap; EXE → DbgNexum. The AMSI and WLDP HWBP bypasses are installed on the first inject call (the ETW HWBP is already in place from EvasionInit).
0x7C95D91A	drop	Drop the file to the disk and execute. Supports DLL, EXE, shellcode (APC injection), and MSI payloads.
0x9A37F083	screenshot	GDI capture, downscale to 960px wide, upload as BMP.
0x08DEDEF0	keylog	Start or stop the inline keylogger.
0x4EE251FF	uninstall	6-step cleanup and termination.
0x65CCC50B	elevate	UAC bypass via the schuac technique (IElevatedFactoryServer::ServerCreateElevatedObject(CLSID_TaskScheduler)); registers a transient elevated task that relaunches the implant.
0xB3B5B880	downgrade	SYSTEM → elevated admin transition.
0x20CE3BC8	(self-restart)	Cascading self-terminate: NtTerminateProcess(-1, 0) direct syscall first; if that fails to resolve, falls back to ExitProcess(0). Persistence relaunches the implant on the next scheduled task tick. Operationally equivalent to a soft restart.

The eighth handler has no debug log naming it. It self-terminates; the scheduled task relaunches the implant on the next tick. The lack of LLM-style scaffolding (debug strings) makes this one of the few handlers in the binary that appears to be added by the human author rather than LLM-generated.

Injection techniques

PHANTOMPULSE ships three injection techniques, one per payload type. The inject C2 command routes shellcode to PhantomInject, DLLs to ManualMap, and EXEs to DbgNexum.



Inject command switch case

The AMSI/WLDP hardware-breakpoint bypasses are installed on the first inject call, before any injector runs.

Payload type	Injector	Strategy
Shellcode	PhantomInject	Module stomping in dbghelp.dll via SEC_IMAGE
EXE	DbgNexum	Debug-API state machine
DLL	ManualMap	Full PE manual mapping

PhantomInject: module stomping into dbghelp.dll

Module stomping avoids MEM_PRIVATE allocation by mapping a legitimate Windows DLL as SEC_IMAGE and overwriting .text:

1. Acquires SeDebugPrivilege (via OpenProcessToken / LookupPrivilegeValueW / AdjustTokenPrivileges), then walks the process snapshot for one of seven host-process candidates (case-insensitive match). Tried in priority order: sihost.exe, taskhostw.exe, backgroundTaskHost.exe, RuntimeBroker.exe, dllhost.exe, ctfmon.exe, explorer.exe.
 2. Opens dbghelp.dll via NtOpenFile, creates SEC_IMAGE section, maps into target via NtMapViewOfSection
- 
- A screenshot of a debugger window showing the memory map of a process. The 'dbghelp.dll' section is highlighted, indicating it has been mapped as SEC_IMAGE.
- PhantomInject dll stomping
3. Parses the local copy for .text RVA and size, then frees it
 4. Selects and suspends a thread, captures context
 5. Builds an [82-byte save-call-restore trampoline](#)
 6. Writes shellcode + trampoline into .text of the mapped DLL
 7. Flips protection to PAGE_EXECUTE_READ
 8. Repoints RIP to the trampoline, resumes thread

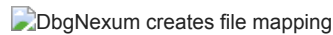
To a memory scanner, the result looks like a thread executing inside legitimate dbghelp.dll, a file-backed image region with the right file path, section name, and first-page hash.

DbgNexum: debug API as an execution controller

DbgNexum handles EXE payloads. Rather than writing executable code into the target up front, it uses the Windows debug API to drive execution one exception at a time: a ROP chain whose gadgets are full Windows APIs in the target.

The technique is not original to PHANTOMPULSE. It is a verbatim lift of [dis0rder0x00/DbgNexum](#), a public proof of concept published on GitHub on 2026-01-04. The operator kept the published technique name ("DbgNexum") in the implant's debug strings unchanged, and the inner state machine is a 1:1 match: the same bait API, section name, gadget chain, and constants. PHANTOMPULSE wraps the lifted x64 core with operational scaffolding the PoC does not have: host-process selection (FindHostProcessEx), a fallback that spawns a fresh SysWow64\cmd.exe / rundll32.exe / notepad.exe, a custom PE-loading bootstrap (so it can carry full EXEs instead of raw shellcode), and a separate WoW64 cross-architecture variant.

For native x64 payloads, the implant pre-stages the PE, the bootstrap stub, and the trampoline config inside a named file-mapping section. The section name is the literal two-byte string "MZ", the implant attaches with `DebugActiveProcess` and creates a remote thread on `FileTimeToSystemTime` with a hardware breakpoint on DR0.



DbgNexum creates file mapping

When the bait hits the breakpoint, a state machine drives the target through this API chain:

1. Redirect RIP to `DbgBreakPoint+1` with the trap flag set; the resulting single-step exception bridges into the rest of the chain.
2. `LocalAlloc(LMEM_ZEROINIT, 3)`, allocate the 3-byte name buffer.
3. `memcpy(buf, kernel32_base, 2)`, copy "MZ" from `kernel32.dll`'s DOS header into the buffer.
4. `memset(stack+40, 0, 8)`: zero a stack arg slot.
5. `OpenFileMappingA(0x1F, FALSE, "MZ")`, open the prepared section with full section-mapping access.
6. `MapViewOfFile(...)`, map it into the target.
7. Redirect RIP to `mapped_base + 0x400`, the bootstrap stub. This is the only stage logged directly: `DbgNexumLoop64: stage 6 -> stub at %llx, base=%llx`. (The PoC redirects to `mapped_base + 0` for raw shellcode; PHANTOMPULSE adds the `+0x400` offset to land on its custom PE loader.)

Each transition intercepts the next debug event, restores RSP, clears the trap flag, modifies RIP and the argument registers (RCX, RDX, R8, R9) for the next call, and continues the debuggee. DR0 is reused as an execute hardware breakpoint on each saved return address, so no inline patches or `WriteProcessMemory` against the target are needed. From the kernel's view, all that happened was a thread inside `kernel32.dll` calling `LocalAlloc`, `OpenFileMappingA`, and `MapViewOfFile`.

The cross-architecture path (PE32 from a 64-bit implant) is a PHANTOMPULSE-only variant the public PoC does not have. It takes a shortcut: the implant walks the target's PEB.Ldr via `NtReadVirtualMemory` (using `ProcessWow64Information` to pick the 32-bit or 64-bit PEB layout) to find a DLL with a callable entry point, then pre-maps a section containing a 32-bit loader stub and trampoline into both processes via `NtCreateSection + NtMapViewOfSection`. The redirect is just two stages: bait at `RtlExitUserThread`, then a `DbgBreakPoint`-mediated single step jumps RIP to the trampoline. There is no API chain on this path because the section is already mapped on both sides, so `OpenFileMappingA/MapViewOfFile` aren't needed.

Cross-arch host selection runs through `FindHostProcessEx`, which excludes critical system processes (`csrss.exe`, `lsass.exe`, `smss.exe`, `winlogon.exe`, `services.exe`, `wininit.exe`, `svchost.exe`, `MsMpEng.exe`) and falls back to spawning a fresh `SysWow64\cmd.exe / rundll32.exe / notepad.exe` if no usable WoW64 host is found.

ManualMap: full PE mapper

ManualMap handles DLL payloads with a complete PE manual mapping implementation:

1. Validates MZ/PE signature; rejects PE32 in the x64 host path (debug-log: "PE32 DLL in x64 host is impossible")
2. Allocates `SizeOfImage` in the target via `NtAllocateVirtualMemory`
3. Copies headers and sections to a local staging buffer
4. Applies base relocations (`IMAGE_REL_BASED_DIR64`, `IMAGE_REL_BASED_HIGHLOW`)
5. Resolves imports via `LoadLibraryA + GetProcAddress`
6. Wipes PE headers (zeros `SizeOfHeaders` bytes)



ManualMap PE headers wiping

7. Writes the staged image into the remote allocation
8. Sets per-section memory protection

9. Builds a **137-byte trampoline** in a separate 0x2000-byte remote allocation (set to PAGE_EXECUTE_READ):

The [full trampoline shellcode gist](#) contains the complete bytes.



ManualMap trampoline write

10. Hijacks a thread via suspend / get-context / set-context

Privilege escalation

The elevate command is a UAC bypass via the **schuac technique**

(IElevatedFactoryServer::ServerCreateElevatedObject(CLSID_TaskScheduler)), published as [UACME issue #129](#) by zcgovh, [currently under the ID 74](#).



screenshot showing the ID of the privilege escalation technique under UACME

Mechanism

MaintenanceUI.dll's CMaintenanceUIVirtualFactory (CLSID {A6BFEA43-501F-456F-A845-983D3AD7B8F0}) is registered with an Elevation registry key, so the OS hands non-admin callers an elevated instance. Its IElevatedFactoryServer interface exposes ServerCreateElevatedObject(rclsid, riid, ppv), which the elevated server uses to instantiate **any other CLSID** under its elevated context. PHANTOMPULSE feeds it CLSID_TaskScheduler, gets back an elevated ITaskService, and uses that service to register a HighestAvailable-RunLevel task that re-launches the implant.

The elevate C2 command

Inside ProcessCommands, the elevate handler:

1. Builds the task action. The command is <system_dir>\rundll32.exe with arguments \"
<deployed_dll>\",DllRegisterServer. The user identifier is COMPUTERNAME\USERNAME from GetEnvironmentVariableW. !
[Elevate command call]/assets/images/blockchain-c2-phantompulse-rat-sinkhole/image17.png]
2. Writes the .elevate marker as a single byte ("1", 0x31), not encoded parameters. The write goes through NtCreateFile + NtWriteFile to bypass user-mode hooks. The marker is just a presence flag; the elevation parameters travel inside the task definition the implant is about to register.
3. XOR-decrypts the COM Elevation Moniker at runtime, 66 bytes from .rdata xored against seed 0xE95CA237, which decodes to Elevation:Administrator!new:{A6BFEA43-501F-456F-A845-983D3AD7B8F0}.
4. Calls CoGetObject(moniker, &BIND_OPTS3{dwClassContext=CLSCTX_LOCAL_SERVER}, IID_IElevatedFactoryServer, &factory) to get an elevated IElevatedFactoryServer*, then factory->ServerCreateElevatedObject(CLSID_TaskScheduler, IID_ITaskService, &elevatedTaskService) to get an elevated ITaskService* that inherits the elevation.
5. Registers a transient task DotNetSvcElevateTask at HighestAvailable RunLevel with the rundll32 action above.
6. Deletes any pre-existing non-elevated persistent tasks so the old low-IL persistence cannot race the elevated relaunch.
7. Calls ITaskService::Run on the transient task and deletes it immediately afterward. Releases the single-instance mutex and exits the medium-IL implant.

Fallback retry via proxy rundll32

If the CoGetObject / RegisterTask chain fails, a fallback path takes over. A separate startup path spawns a fresh rundll32.exe " <deployed_dll>\",DllRegisterServer directly via CreateProcessW, with retries (">>> .elevate redirect attempt %d"). Inside that rundll32 the implant detects isProxy=1, elevated=0 and **retries the schuac sequence** with three registration variants (ELEVATED+INTERACTIVE+user, ELEVATED+INTERACTIVE, INTERACTIVE). On success, the elevated task fires and the elevated relaunch takes over. On exhaustion, the proxy logs ">>> Phase 1: all registration methods failed, cleaning marker" and exits.

Elevated rundll32 relaunch

When the transient task fires, `svchost.exe` (Schedule) launches `rundll32.exe "<deployed_dll>",DllRegisterServer` under a fresh high-IL token. The implant's `DllRegisterServer` export runs as the entry; at startup it sees the `.elevate` marker and a high-IL token and routes to the elevated path:

- Reads and **deletes** the `.elevate` marker.
- **Reinstalls persistence under elevated context**, including the boot task `DotNetSvcCoreTask` under `\Microsoft\Windows\NetFramework\`, registered with `INTERACTIVE_TOKEN + BootTrigger`, which requires admin to register.
- Continues normal implant operation as a high-IL service.

Marker-state routing

At every startup, `MainEntryLogic` calls `GetFileAttributesW` on the `.elevate` path. If it returns `INVALID_FILE_ATTRIBUTES`, the implant skips all elevation logic and starts normally. If the marker exists, the implant gathers two more facts: whether the current process is a `rundll32.exe/regsrvr32.exe` proxy, and whether the token is elevated, then routes on the combination:

Marker	Proxy	Elevated	What happens
absent	n/a	n/a	normal startup, no special routing
present	no	no	spawn proxy <code>rundll32</code> to retry the schuac chain, then exit
present	yes	no	retry schuac inside the <code>rundll32</code> ; on exhaustion delete marker and exit
present	*	yes	delete marker, reinstall persistence with the boot task, continue as high-IL

The marker file's *contents* are never read, its presence alone drives the routing.

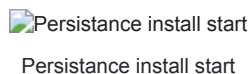
Elastic Security Labs' [Exploring Windows UAC Bypasses](#) covers detection patterns for the `IElevatedFactoryServer` class of bypass directly.

Persistence

Three scheduled tasks

PHANTOMPULSE installs three scheduled tasks via the COM `ITaskService` interface, each executing `rundll32.exe "<stub_dll>",DllRegisterServer:`

Task	Trigger	Interval	RunLevel
<code>DotNetSvcUpdateTask</code>	User Logon + Time	3 min	Standard
<code>DotNetSvcCoreTask</code>	Boot + Time	15 min	HighestAvailable + Hidden
<code>DotNetSvcUserTask</code>	User Logon	On logon	Standard



The boot task lives under `\Microsoft\Windows\NetFramework\` to blend with legitimate .NET maintenance tasks. Its full XML body is embedded in `.rdata`, not constructed at runtime. Signatures on this literal blob are durable across builds.

Installation tries multiple `RegisterTask` fallback variants per task, combining `INTERACTIVE_TOKEN` and `S4U` logon types with different user-identifier formats (`COMPUTER\User`, `SID`, short username). The exact order of attempts depends on the implant's privilege context (`SYSTEM`, `admin`, or standard user); failures fall through to the next variant until one succeeds.

Stub DLL drop

The deployed agent lives **encrypted and compressed inside the EXE's own .rdata**. FilelessPersist decrypts it on demand and writes the result to disk via (NtCreateFile + NtWriteFile).

Decryption is two stages:

1. **XOR-decode** a blob against a rotating 16-byte key at $(\text{decoded}[i] = \text{blob}[i] \wedge \text{key}[i \ \& \ 0xF])$, with $\text{decoded}[0] = 0x1E$ as a hard-coded first byte that primes the LZNT1 chunk header).
2. **RtlDecompressBuffer(COMPRESSSION_FORMAT_LZNT1, ...)** on the decoded buffer, which is a DLL (svcagent.dll).

Drop paths in priority order, falling through on access errors:

1. %ProgramData%\AssetMon\svcagent.dll (primary)
2. %APPDATA%\AssetMon\svcagent.dll
3. %TEMP%\svcagent.dll
4. A redundant %ProgramData% "sleeper" copy at a separate path

Analysts can reproduce the deployed DLL offline by reading the two regions above out of the EXE, applying the XOR loop, and feeding the result to RtlDecompressBuffer (or any LZNT1 implementation) as seen in the CyberChef screenshot below.



Cyberchef decrypting the DLL

DLL sideload migration

A block inside SetupRegistryPE (logged with the MigrateSideload / MigrateLegacySideloads debug-string prefixes) enumerates running processes and their **executable directories**, hunting for diagcore.dll. When found, it overwrites the file with the current stub via CopyFileW.

Self-healing

Self-healing runs on iteration 2 of the C2 loop and every 10th iteration thereafter, gated on a deferred-persist flag being clear. The check order:



Self healing start

1. **Registry-persistence check first.** CheckRegistryPersistence runs at the top of the block. If it reports unhealthy, the implant immediately re-runs FilelessPersist (re-decrypts and re-drops the stub DLL) and InstallPersistence (re-registers the task triggers).
2. **Task verification.** SelfHealCheckTasks then verifies the three persistence tasks (DotNetSvcUpdateTask, DotNetSvcCoreTask, DotNetSvcUserTask) and reinstalls any that are missing. The boot task check is gated on SYSTEM-or-admin context; non-privileged callers skip it.
3. **AV inventory refresh.** DetectInstalledAV runs at the end to refresh the operator-visible AV product list.

The privilege gating matters for eviction. From a non-elevated context, the boot-task check is skipped, so the boot task is not inspected during cleanup. Full eviction requires removing all three tasks plus the registry artifact in one window from an elevated context.

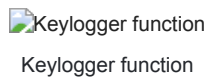
Beyond the iteration-based self-heal, the implant carries a **deferred-persist mechanism** keyed off a single flag. On heartbeat-success paths in C2Loop_Main, once the heartbeat-success counter exceeds one with the flag set, the implant re-runs FilelessPersist + InstallPersistence and clears the flag. This gives PHANTOMPULSE a second persistence-repair path that fires on a different trigger than the iteration-based self-heal.

Collection

Inline keylogger with clipboard monitoring

The keylogger runs inline in the C2 loop with no dedicated thread. It resolves APIs from `user32.dll` at runtime:

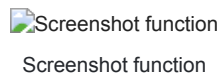
API	Purpose
<code>GetAsyncKeyState</code>	Polling key state
<code>GetForegroundWindow</code>	Active window detection
<code>GetWindowTextA</code>	Window title capture
<code>MapVirtualKeyA / ToUnicode</code>	Key translation
<code>GetClipboardSequenceNumber</code>	Clipboard change detection
<code>OpenClipboard / GetClipboardData</code>	Clipboard reading (CF_UNICODETEXT)



The log file is XOR-encrypted with the `0xE95CA237` seed. Uploads send only the delta to avoid retransmission.

Screenshot

Screenshots use GDI APIs resolved by hash. If desktop width exceeds 960 px, the image is downscaled before upload. The raw BMP is built in memory and uploaded with Content-Type: `image/bmp`. Triggered on-demand by the screenshot C2 command (hash `0x9A37F083`).



System reconnaissance

Recon data the implant gathers:

Data	Source
CPU	Registry: ProcessorNameString
GPU	Registry display adapter DriverDesc (filters "Microsoft Basic")
RAM	GlobalMemoryStatusEx
OS	RtlGetVersion with build-to-version mapping (Win7 through Win11, Server 2008 through Server 2025)
Username	GetUserNameW with fallback to LookupAccountSidW from explorer.exe token
Privilege	Token elevation type: user, admin, admin_nouac, system
AV	DetectInstalledAV matches running processes against a hardcoded list of ~25–30 AV vendor process names
Apps	DetectInstalledApps checks a curated 19-name targeted-app list
Firewall state	Reads SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\{Domain,Standard,Public}Profile to record per-profile enabled state

Data	Source
Services	Running-service count via service enumeration
Machine ID	DJB2(module name) ^ volume serial
Public IP	Multi-API HTTPS chain



System reconnaissance

The AV-detection list is unusually broad, covering standard Western consumer AV products such as Defender, Norton, McAfee, Avast, AVG, Avira, Bitdefender, ESET, F-Secure, G Data, Kaspersky, Panda, Sophos, Trend Micro, VIPRE, Webroot, ZoneAlarm, Comodo along with EDR vendors (CrowdStrike, SentinelOne, Cylance, Malwarebytes, HitmanPro) are all covered. The implant also probes for **AhnLab V3** (South Korean), **Qihoo 360 / 360 Total Security** and **Tencent QQPC** (Chinese), and **K7 Computing** (Indian). The Asian-AV inclusion is uncommon for Western-targeted commodity stealers and consistent with an implant designed for victims across multiple regional markets.

The implant also checks for a curated list of 19 high-value applications by name and flags matches in the heartbeat (App detection: found %d apps):

Category	Targets
Cryptocurrency wallets	ledger, trezor, bitcoin-core, electrum, exodus, atomic, guarda
Messengers	telegram, discord, signal, viber, slack, whatsapp
Mail clients	thunderbird, outlook
2FA app	authy
File transfer / SSH	filezilla, winscp
Gaming	steam

The detection function (DetectInstalledApps) does **not** scan the registry or enumerate processes. It expands three environment-variable roots (%LOCALAPPDATA%, %APPDATA%, %ProgramFiles(x86)%), concatenates a hardcoded UTF-16 relative-path suffix per app (e.g. \Telegram Desktop\, \Authy Desktop\, \Ledger Live\, \@trezor\trezor-suite\, \Steam\steam.exe), and calls GetFileAttributesW on each path. A non-error return means the app is installed, and the name is recorded in the heartbeat results buffer.



List of Targeted apps

PHANTOMPULSE itself does not extract data from any of these. The list is target reconnaissance for follow-on tasking. The operator sees in the heartbeat which high-value applications a given victim has and decides what specialized payload to push next via inject or drop.

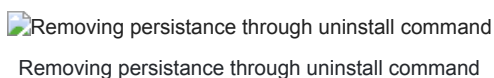
No wallet, browser, messenger, or credential stealer functionality was identified in the analyzed sample; the targeting list is purely a presence-check feeding the operator's decision tree.

Uninstall

A 6-step cleanup, triggered by the uninstall command, by "status": "deleted" in a heartbeat response, or by a kill flag in the registry:

Step	Action
1/6	Write kill flag to HKCU + HKLM, kill host process
2/6	Remove all 3 scheduled tasks via COM + CreateProcessW fallback
3/6	Remove legacy registry: NTLoad value, COM hijack keys, print monitor keys
4/6	Delete stub DLLs, sleeper logs, registry PE blob, ProgramData directories
5/6	Delete install path and self path from disk
6/6	Terminate residual healthmon.exe and any rundll32.exe instances hosting svcagent.dll

Step 3 reveals the legacy persistence techniques: cleanup logic for COM hijack and print monitor keys that this build never installs.



Attribution

PHANTOMPULSE's tradecraft, targeting, and infrastructure choices align with the **DPRK-aligned crypto-targeting intrusion clusters** that include Lazarus, BlueNoroff, UNC5342 (Contagious Interview), and APT38. Multiple independent dimensions match recent public reporting on those clusters.

Signals aligning with DPRK reporting:

- **Blockchain-resolved C2 via transaction input fields** matches the dead-drop-resolver pattern Mandiant attributes to UNC5342 (Contagious Interview) in [DPRK Adopts EtherHiding](#). PHANTOMPULSE's specifics (wallet-byte XOR, multi-chain Blockscout) are not a 1:1 fingerprint, but the technique class is now DPRK-tagged.
- **Desktop crypto-wallet enumeration set** (ledger, trezor, bitcoin-core, electrum, exodus, atomic, guarda) closely matches Unit 42's [RustDoor / Koi Stealer for macOS](#) targeting list, which is DPRK-attributed.
- **Cross-platform Windows + macOS implants** for the same victim profile (the prior REF6598 post documented a macOS sibling with C2 at 0x666[.]info and a Telegram fallback at t[.]me/ax03bot) is a BlueNoroff signature.
- **Telegram and messenger targeting** is specifically a BlueNoroff specialty per [Arctic Wolf BlueNoroff coverage](#).

Hunting for new C2 domains via the resolver wallet's known-plaintext signature

The XOR scheme used by the blockchain resolver leaks a stable 2-byte signature defenders can hunt against the entire chain, not just one wallet.

Two facts combine: every C2 URL begins with ht (from http:// or https://), and the XOR key is the wallet's ASCII address verbatim, so its first two key bytes are always the literal characters 0 and x. XOR-ing ht against 0x yields \x58 \x0c. **Every encrypted input field produced by a PHANTOMPULSE-style resolver, on any chain, signed by any related wallet, begins with the four hex characters 580c.**

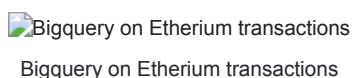
This converts the hunt from monitoring one wallet into sweeping the chain for the signature. Public Ethereum, Base, and Optimism transaction data is queryable via BigQuery, Dune, or full archive nodes. A query against the public Ethereum transactions dataset for input values starting with 0x580c, scoped to a recent block-timestamp window, surfaces previously-unknown resolver wallets used by the same codebase. Each match is validated by decoding with the sender wallet's ASCII address as the key: a real C2 URL begins with http after decoding. The following [CyberChef recipe](#) can be used to decrypt the C2 URL.

```
SELECT
    block_timestamp AS block_time,
```

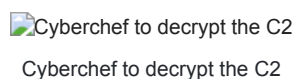
```

from_address AS `from`,
to_address AS `to`,
input AS data
FROM `bigquery-public-data.crypto_ethereum.transactions`
WHERE block_timestamp >= '2026-04-01 00:00:00'
AND input LIKE '0x580c%'
ORDER BY block_timestamp DESC
LIMIT 10000;

```



CyberChef can decrypt the input data to reveal the domain, as shown in the screenshot below.



Conclusion

PHANTOMPULSE is engineered from published components: module stomping, debug-API state machines, manual mapping, hardware-breakpoint AMSI/WLDP/ETW bypass, scheduled-task persistence, and blockchain C2. The combination and the hardening that ties it together point to a mature codebase under active development. The durable signals are behavioral and are covered by Elastic's [behavioral protections for REF6598](#).

PHANTOMPULSE and MITRE ATT&CK

Elastic uses the MITRE ATT&CK framework to document common tactics, techniques, and procedures that advanced persistent threats use against enterprise networks.

Tactics

Tactics represent the why of a technique or sub-technique. It is the adversary's tactical goal: the reason for performing an action.

- [Initial Access](#)
- [Execution](#)
- [Persistence](#)
- [Privilege Escalation](#)
- [Defense Evasion](#)
- [Credential Access](#)
- [Discovery](#)
- [Collection](#)
- [Command and Control](#)
- [Exfiltration](#)

Techniques

Techniques represent how an adversary achieves a tactical goal by performing an action.

- [Phishing: Spearphishing via Service](#)
- [Command and Scripting Interpreter: PowerShell](#)
- [Process Injection](#)
- [Process Injection: DLL Injection](#)
- [System Binary Proxy Execution: Msiexec](#)

- [System Binary Proxy Execution: Rundll32](#)
- [Scheduled Task/Job: Scheduled Task](#)
- [Boot or Logon Autostart Execution](#)
- [Modify Registry](#)
- [Impair Defenses: Disable or Modify Tools](#)
- [Indicator Removal: File Deletion](#)
- [System Information Discovery](#)
- [System Owner/User Discovery](#)
- [Process Discovery](#)
- [Software Discovery: Security Software Discovery](#)
- [Input Capture: Keylogging](#)
- [Clipboard Data](#)
- [Screen Capture](#)
- [Exfiltration Over C2 Channel](#)
- [Application Layer Protocol: Web Protocols](#)
- [Web Service](#)
- [Encrypted Channel](#)
- [Obfuscated Files or Information](#)
- [Deobfuscate/Decode Files or Information](#)
- [Access Token Manipulation](#)
- [Abuse Elevation Control Mechanism: Bypass User Account Control](#)
- [Native API](#)
- [Virtualization/Sandbox Evasion: Time Based Evasion](#)
- [Hijack Execution Flow: DLL Side-Loading](#)
- [Reflective Code Loading](#)

Remediating

YARA

Elastic Security has created YARA rules to identify this activity.

[Windows.Trojan.PhantomPulse](#)

Observations

Observable	Type	Name	Reference
33dacf9f854f636216e5062ca252df8e5bed652efd78b86512f5b868b11ee70f	SHA-256	PHANTOMPULSE RAT	Final payload
70bbb38b70fd836d66e8166ec27be9aa8535b3876596fc80c45e3de4ce327980	SHA-256	syncobs.exe	PHANTOMPULL loader
def66275fa3baffb16e6e4ae0297861d9790ae7161fbc271a2ba05d121f13c70	SHA-256	Go beacon	GTESTIC_WIN check-in
panel.fefea22134[.]net	domain	C2 panel	PHANTOMPULSE hardcoded fallback
fea22134[.]net	domain	C2 domain	Encrypted in binary
195.3.222[.]251	ipv4-addr	Staging server	PowerShell/loader delivery

Observable	Type	Name	Reference
0xc117688c530b660e15085bF3A2B664117d8672aA	crypto-wallet	Blockchain C2 wallet	ETH/Base/Optimism
0x38796B8479fDAE0A72e5E7e326c87a637D0Cbc0E	crypto-wallet	Funding wallet	C2 resolution funding
eth.blockscout[.]com	domain	Blockchain provider	C2 URL resolution
base.blockscout[.]com	domain	Blockchain provider	C2 URL resolution
optimism.blockscout[.]com	domain	Blockchain provider	C2 URL resolution
hVNBu0RXNiFLhYYh	mutex	Single instance	XOR-decrypted
svcagent.dll	file-name	Stub DLL	Persistence payload
AssetMon	directory	Stub DLL directory	%ProgramData% or %APPDATA%
healthmon.exe	file-name	Dropper	Original executable name
diagcore.dll	file-name	Legacy sideload DLL	Migrated by MigrateSideload
.elevate	file-name	Elevation marker	Routes the elevated relaunch
DotNetSvcUpdateTask	scheduled-task	Primary persistence	3-min interval
DotNetSvcCoreTask	scheduled-task	SYSTEM persistence	15-min, hidden
DotNetSvcUserTask	scheduled-task	User persistence	Logon trigger
EdgeWebViewUpdateTask	scheduled-task	Legacy task	Cleaned during uninstall
\Microsoft\Windows\NetFramework\DotNetSvcCoreTask	task-uri	Boot task path	Hidden scheduled task
Elevation:Administrator!new:{A6BFEA43-501F-456F-A845-983D3AD7B8F0}	commoniker	UAC bypass	Elevated ITaskService
0x666[.]info	domain	macOS C2	macOS dropper
t[.]me/ax03bot	url	Telegram fallback	macOS C2 dead-drop
thoroughly-publisher-troy-clara[.]trycloudflare[.]com	domain	Prior C2	Cloudflare Tunnel

References

Prior reporting and toolkits referenced in this analysis:

- [Phantom in the vault: Obsidian abused to deliver PhantomPulse RAT](#)
- [Blockscout Ethereum Explorer](#)
- [DPRK Adopts EtherHiding](#)
- [Contagious Interview / fake-recruiter targeting of crypto-sector developers](#)
- [RustDoor and Koi Stealer for macOS](#)
- [BeaverTail and OtterCookie evolution](#)
- [DbgNexum proof-of-concept](#)
- [UACME issue #129: schuac UAC bypass](#)
- [Exploring Windows UAC Bypasses](#)
- [Memory Patching AMSI Bypass](#)

Jump to section

- [Key takeaways](#)
- [A note on AI-assisted development](#)
- [Execution chain](#)
- [Defense evasion](#)
- [Direct syscalls and API wrappers](#)
- [String and config obfuscation](#)
- [AMSI, WLDP, and ETW bypass via hardware breakpoints](#)
- [Build variant: active and dormant subsystems](#)
- [Command and control](#)
- [Blockchain C2 resolution](#)

Elastic Security Labs Newsletter

[Sign Up](#)