GLASSWORM

Threat Intelligence Report — Version 4

Rust and C/C++ npm Supply Chain Attack — Multi-Browser Credential Theft

March 2026 | TLP:WHITE

1. Executive Summary

GLASSWORM is a multi-stage malware campaign delivered via malicious Node.js native addons (.node files compiled as Windows DLLs) distributed through the npm package ecosystem. The campaign employs Rust as the primary implementation language for most components, with at least one component written in C/C++, providing cross-platform capability and EDR evasion via direct NT syscall invocation.

The campaign targets Windows and macOS environments where Node.js applications are installed. It achieves persistent browser access through two complementary mechanisms: a C/C++ component that directly extracts Chrome and Edge encryption keys using both legacy DPAPI and the modern app-bound COM encryption bypass, and a Rust-based Chrome credential harvester that silently installs a malicious browser extension using a known Chrome Secure Preferences MAC forgery technique. A full-featured Rust C2 agent stores harvested data in a local SQLite database and communicates with other components via named pipe IPC.

Analysis of eight samples reveals a four-component architecture with both 32-bit (pe32) and 64-bit (pe32+) Windows variants, plus a macOS Mach-O variant. Build timeline analysis places compilation between June 2025 and November 2025, with discovery in March 2026. Components were developed iteratively rather than as a single build event, with the 64-bit C2 agent representing a later, extended version of the 32-bit original.

2. Sample Inventory

Eight samples were analyzed. All Windows samples are unsigned PE files. The Mach-O sample was included in string analysis after initial processing errors prevented full disassembly.

SHA256 (truncated)	SHA1 (truncated)	Arch	File Type	Functions	Genomes	Role
68e5fb92...	109729f9...	pe32	Win32 DLL	3,358	3,058	C2 Agent (32-bit)
415a4f39...	6d557f4f...	pe32	Win32 DLL	3,375	3,073	C2 Agent (32-bit) — twin
fdba5be3...	5ca854c3...	pe32	Win32 DLL	1,373	1,118	Chrome Harvester / Extension Installer (32-bit)
4e339dcd...	2415ffdc...	pe32	Win32 DLL	409	351	Neon Bootstrap (32-bit)
1ed7ca53...	35d242f3...	pe32+	Win64 DLL	2,662	2,372	C2 Agent (64-bit, extended)
de81eacd...	acb46155...	pe32+	Win64 DLL	424	355	Neon Bootstrap (64-bit)
43253a88...	c3df90f2...	pe32+	Win64 DLL	81	60	C/C++ Browser Credential Stealer

ee3e4dd5...	db8506a0...	macho	Mach-O fat	1,432	1,031	macOS Chrome Harvester

The two large pe32 files (68e5fb92 and 415a4f39) share a 0.9951 similarity score and are treated as a single component. All Windows DLLs are unsigned.

3. Build Timeline

Two distinct rustc compiler versions are embedded in panic strings across the sample set, establishing a minimum development window of approximately five months:

rustc Commit Hash	Version	Commit Date	Found In
6b00bc388...	1.88.0	2025-06-23	C2 Agent pe32 (68e5fb92, 415a4f39), Chrome Harvester (fdba5be3)
ed61e7d7e...	1.91.1	2025-11-07	Neon Bootstrap pe32 (4e339dcd), Neon Bootstrap pe32+ (de81eacd)

The PE creation timestamp on the large pe32 C2 agent files is November 24, 2025 — consistent with the rustc 1.88.0 compiler date and falling between the two compiler release dates. PE timestamps are trivially forgeable, but the internal consistency with the compiler evidence supports their authenticity in this case.

The campaign was not built in a single event. The five-month compiler gap indicates iterative development, with the Neon Bootstrap components compiled approximately five months after the C2 agent. The IOC feed discovery date of March 16, 2026 places deployment approximately four months after the most recent build.

All components contain the path prefix C:\Users\Administrator\.cargo\ in embedded panic strings, confirming all builds ran on Windows under an Administrator account. This is consistent with a CI or build server environment.

4. Delivery Mechanism

4.1 npm Supply Chain

All Windows DLL samples were bundled directly inside npm package tarballs rather than downloaded as second-stage payloads. Evidence:

- No download or staging infrastructure APIs are present in any sample — no WinInet, WinHTTP, URLDownloadToFile, or equivalent.

- npm project layout strings are embedded in the large C2 agent: node_modules, .idea, .vscode, dist, build, out, src/lib.rs — the expected directory structure of a Rust-based npm native addon package.
- The neon crate (versions 1.0.0 and 1.1.1) is the Rust-to-Node.js NAPI bridge library, used exclusively for building .node native addon files loaded directly by Node.js require().
- IOC feed tags native-addon and npm-archive on all samples corroborate npm delivery.

4.2 Node.js Native Addon Entry Point

The Neon Bootstrap DLL is loaded directly by Node.js when a JavaScript caller executes require() on the .node file. NAPI entry points including napi_get_value_string_utf16 indicate the malicious code is invoked when JavaScript passes string arguments to the addon. The presence of icu_collections-2.0.0 (Unicode data structures) in the C2 agent and napi_get_value_string_utf16 (UTF-16 string access) in the bootstrap suggests Unicode string handling at the JavaScript boundary, though the specific trigger mechanism was not fully resolved.

The macOS Mach-O variant contains no NAPI strings, indicating a different delivery mechanism on macOS — possibly a standalone dylib or a separate infection vector not represented in this sample set.

5. Component Architecture

Four functionally distinct components are identified. Three are written in Rust; one is written in C/C++.

5.1 Neon Bootstrap DLL (4e339dcd / de81eacd, ~409-424 functions)

The Neon Bootstrap is the campaign's entry point and the first component to execute. When a JavaScript application imports the malicious npm package via require(), Node.js loads this DLL and calls its NAPI-registered entry point. Its operational role is to:

- Receive the initial call from JavaScript across the NAPI bridge, converting a JavaScript function invocation into native code execution.
- Resolve NT syscall addresses directly from ntdll.dll at runtime by name — bypassing the Win32 API layer entirely. This means subsequent memory allocation and permission changes (NtAllocateVirtualMemory, NtProtectVirtualMemory) are invisible to EDR products that hook the Win32 API, because the calls never pass through those hooks.
- Establish the native execution environment required by the remaining payload components, which are then invoked in sequence.

In practical terms the Bootstrap is the bridgehead: it receives the invisible-Unicode payload string from the JavaScript layer, decodes it into a PE binary in memory, and executes it with EDR evasion already active before any credential theft or C2 activity begins. Every other component in the campaign depends on this loading successfully first.

This pair is the most convincingly cross-compiled component in the set: function counts differ by only 3.5% (409 vs 424), both share identical Rust type system strings, memory management idioms, and Rust backtrace infrastructure, and both embed the same rustc 1.91.1 compiler commit hash. Both variants were compiled with rustc 1.91.1 (November 7, 2025) — the most recent compiler in the set.

5.2 Chrome Harvester / Extension Installer (fdba5be3, 1,373 functions)

A Rust DLL with two distinct responsibilities: extracting Chrome credentials and silently installing a malicious Chrome extension. Key capabilities:

- Chrome User Data directory discovery via registry: SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\chrome.exe and the WOW6432Node equivalent for 32-bit Chrome on 64-bit Windows.
- Chrome encrypted credential access: reads the Local State file to locate the DPAPI-wrapped AES key, then accesses the Login Data SQLite database.
- Silent extension installation using the Phantom Extension technique: writes a malicious extension blob to Chrome's Secure Preferences file with forged HMAC-SHA256 MAC values. The MAC seed is derived from the SHA-512 hash of Chrome resource file #146 in resources.pak (see Section 6.3).
- The extension blob is hardcoded with creation_flags: 38 (do not sync to other devices, prevent update URL override), from_webstore: false, location: 4 (load from disk), and a hardcoded fake install timestamp of 13397690747955841 (2025-07-22).
- The extension preferences key pathprotectionmacs is the Chrome extensions.settings key under which the malicious extension is registered — this is effectively the extension's identifier fragment.
- ZIP archive handling (zip-0.6.6, bzip2-0.4.4, flate2-1.1.8) — credential data is compressed before staging.
- winreg-0.52.0 for registry access; constant_time_eq-0.1.5 for timing-safe byte comparison.

Compiled with rustc 1.88.0 (June 23, 2025).

5.3 C/C++ Browser Credential Stealer (43253a88, 81 functions)

The only non-Rust component in the set. Written in C/C++ and compiled with MSVC (confirmed by __stdio_common_vfprintf and __acrt_iob_func imports). All capability findings in this section are derived from static string analysis — the disassembly was not examined at function level. However the strings are explicit operational log messages and API names that are unambiguous about function: this component was purpose-built to extract browser credentials.

- Chrome V10 DPAPI key extraction: reads Local State JSON, base64-decodes the wrapped key, calls CryptUnprotectData to recover the AES key.
- Chrome V20 app-bound encryption bypass: Chrome 127 (July 2024) introduced a COM-based key protection service (IElevator) that only Chrome itself can normally invoke. This component bypasses it by calling CoSetProxyBlanket to impersonate Chrome, then invoking IElevator::DecryptData directly. An Edge-specific path via IElevatorEdge::DecryptData is also implemented. Multiple COM interface IDs are tried in sequence with fallback logic.
- Locked file access via handle theft: when Chrome is running, its Login Data file is exclusively locked. This component calls NtQuerySystemInformation to enumerate all open

file handles system-wide, identifies Chrome's handle to the target file, duplicates it via DuplicateHandle, and reads the file through the stolen handle. Falls back to CopyFileA if handle theft fails.
- Browser process enumeration via CreateToolhelp32Snapshot / Process32FirstW — detects running Chrome, Edge, and other browsers by process name.
- Volume fingerprinting via GetVolumeInformationA — collects disk volume serial number as a machine identifier.
- Verbose timestamped logging to both file and OutputDebugStringA with format [HH:MM:SS.mmm-module-PID] — consistent with a shared internal tooling framework reused across multiple campaigns.

The C/C++ language choice suggests this component was either sourced separately from the Rust components, represents an older independently maintained tool, or was written by a different developer within the same team. The small function count (81) reflects lean C/C++ compilation rather than Rust's statically-linked runtime. The verbose timestamped logging style ([HH:MM:SS.mmm-module-PID] with [+]/[!]/[v]/[i] severity prefixes) is a professional operational practice common across both legitimate security tooling and offensive frameworks — it is not itself a discriminator. The detection value in Rule A-4 comes entirely from the specific operational strings (ExtractDecryptedV10KeyFromLocalState, SendDpapiKeyRecord, Attempting To Steal Opened File Handles) which name capabilities that have no legitimate use, not from the log format.

5.4 C2 Agent (68e5fb92 / 1ed7ca53, ~1,373-3,358 functions)

The largest and most capable component, present in both 32-bit and 64-bit variants. The 64-bit variant (1ed7ca53) is a later, extended version — not a simple recompile — compiled against SQLite 3.46.1 versus the 32-bit variant's SQLite 3.45.0, and containing additional browser targeting and error handling not present in the 32-bit version.

- Multi-browser targeting: Chrome, Firefox, Brave, and Edge (64-bit only) executable paths are hardcoded for browser process identification.
- Browser User-Agent spoofing: hardcoded Chrome and Firefox UA strings used to blend C2 traffic with legitimate browser traffic.
- AES encryption via Windows BCrypt API: BCryptOpenAlgorithmProvider, BCryptDeriveKeyPBKDF2, BCryptSetProperty — full PBKDF2 key derivation and AES encryption workflow.
- Named pipe IPC (ConnectNamedPipe) — communicates with the C/C++ credential stealer or other local components without using network sockets for initial staging.
- Embedded SQLite for local data storage (rusqlite-0.31.0 with SQLite 3.45.0 in pe32; rusqlite linking SQLite 3.46.1 in pe32+).
- Recursive filesystem traversal via walkdir-2.5.0 — enumerates the filesystem to locate files of interest beyond browser data.
- URL parsing via url-2.5.7 for constructing and validating C2 endpoint URLs.
- Unicode data structures via icu_collections-2.0.0 for handling non-ASCII filenames and strings.

The 64-bit variant additionally contains the UUID 4bda9e7e-4913-4dbc-95de-891cbf66598e-errorVal and targets Microsoft Edge (msedge.exe) — capabilities absent from the 32-bit version.

Both variants compiled with rustc 1.88.0 (June 23, 2025).

5.5 macOS Chrome Harvester (ee3e4dd5, 1,432 functions — Mach-O)

A Rust-compiled macOS variant targeting Chrome credential stores. Key characteristics:

- Chrome Secure Preferences manipulation using the same Phantom Extension technique as the Windows variant — identical MAC seed (e748f336...) hardcoded.
- system_profiler SPHardwareDataType invoked via _posix_spawnp for macOS hardware UUID collection — the macOS equivalent of GetVolumeInformationA.
- ZIP and zstd decompression (full error string corpora for both libraries statically linked).
- serde_json for Chrome JSON configuration parsing.
- DWARF debug section names present (gimli/addr2line crate) — the binary was compiled without full debug stripping, unlike the Windows DLLs. This indicates a less mature macOS build pipeline.
- No NAPI strings — delivery mechanism on macOS differs from the Windows npm .node approach and was not determined from this sample set.

6. Key Techniques

6.1 EDR Evasion via Direct NT Syscalls

The Neon Bootstrap pe32+ variant resolves NtAllocateVirtualMemory and NtProtectVirtualMemory directly from ntdll.dll by name at runtime, bypassing the Win32 API layer monitored by most EDR user-mode hooks. This is a deliberate and current evasion technique built into the foundational component loaded first by Node.js.

6.2 Chrome App-Bound Encryption Bypass (V20)

Chrome 127 (July 2024) introduced app-bound encryption, protecting credential keys behind a COM elevation service (IElevator) that verifies the calling process is Chrome itself. The C/C++ credential stealer bypasses this by using CoSetProxyBlanket to impersonate Chrome before calling IElevator::DecryptData. This is a current bypass targeting Chrome versions 127 and later. The fallback to V10 DPAPI handling covers older Chrome versions. Edge is handled via a parallel IElevatorEdge interface.

6.3 Chrome Phantom Extension Installation

Chrome protects its extension configuration (Secure Preferences) with HMAC-SHA256 MAC values. Chrome derives the HMAC key from the SHA-512 hash of resource entry #146 in its resources.pak file, combined with the user's Windows SID. The GLASSWORM Chrome harvester hardcodes this SHA-512 hash (e748f336d85ea5f9dcdf25d8f347a65b4cdf667600f02df6724a2af18a212d26b788a25086910cf3a90313696871f3dc05823730c91df8ba5c4fd9c884b505a8) to forge valid MACs, allowing it to inject a malicious extension that Chrome accepts as legitimate.

This technique is documented in the Synacktiv 'Phantom Extension' research and is used by multiple independent threat actors — it is not unique to GLASSWORM. The hash is version-specific to a particular Chrome resources.pak build; it becomes invalid when Chrome updates resource #146. GLASSWORM carries a single hardcoded hash rather than computing it locally, accepting version-specificity in exchange for simpler runtime behaviour. The same hash appears in the macOS Mach-O variant, targeting the same Chrome version cross-platform.

6.4 Browser File Handle Theft

Chrome holds an exclusive lock on its Login Data SQLite file while running. The C/C++ credential stealer resolves this by calling NtQuerySystemInformation with SystemHandleInformation to enumerate every open file handle across all processes, identifying Chrome's handle to the target file, and duplicating it via DuplicateHandle and OpenProcess. This allows reading the locked file without terminating Chrome or triggering re-authentication. A fallback to GetTempPathA / CopyFileA is used if handle enumeration fails.

7. Cross-Architecture Analysis

The working hypothesis that corresponding pe32 and pe32+ components represent the same source code compiled for different architectures was tested against both function counts and string corpora. String comparison (Jaccard similarity) was used as the primary evidence.

Component	pe32 SHA1	Funcs	pe32+ SHA1	Funcs	Jaccard	Verdict
Neon Bootstrap	2415ffdc	409	acb46155	424	0.559	Strongly supported
C2 Agent	109729f9	3,358	35d242f3	2,662	0.410	Partially — 64-bit is an extended version
Cred Stealer	5ca854c3	1,373	N/A	—	—	No confirmed pe32+ counterpart

The Neon Bootstrap pair has the strongest cross-compilation evidence: near-identical function counts, shared NAPI function name tables, identical rustc 1.91.1 commit hash in both, and matching Rust runtime error strings. The ~3.5% function count difference is within expected variation from compiler inlining decisions between x86 and x86-64.

The C2 Agent pair has a lower Jaccard similarity (0.410) and a larger function count divergence. The shared strings are dominated by the SQLite SQL statement corpus (identical in both, reflecting the same statically-linked SQLite source). However the 64-bit variant adds Edge targeting, a distinct UUID, and was compiled against a newer SQLite version. This is consistent with the 64-bit variant being an independently maintained later version of the same codebase rather than a simple recompilation.

8. Indicators of Compromise

8.1 File Hashes

SHA256	Description
68e5fb92a7d7d182306a025010c77ae2cd89c39031cced13f9686a9f34671041	C2 Agent pe32 (twin A)
415a4f39dd93c2ad5fd02023489352b974a9a917664240299ca4c35ca9a5a362	C2 Agent pe32 (twin B)
fdba5be3da2467e642bd8710f971e6b266b30ac15f5f413982fd719d7e0bffd9	Chrome Harvester / Extension Installer pe32
4e339dcdc3e3a8bf5271f7f76a9c4f064d3e34cbb51f8770ff4cce910fbcbce5	Neon Bootstrap pe32
1ed7ca5301e96e3cef201311b76ba33f842fdb34e91041177865b6e07acb7b4d	C2 Agent pe32+ (extended)
de81eacd045a88598f16680ce01bf99837b1d8170c7fc38a18747ef10e930776	Neon Bootstrap pe32+
43253a888417dfab034f781527e08fb58e929096cb4ef69456c3e13550cb4e9e	C/C++ Browser Credential Stealer pe32+
ee3e4dd5c1e073b8805f4107ccc7bc7e6e3c209fe13ea04ff3f2173c8dbe74a6	macOS Chrome Harvester (Mach-O)

8.2 Build Environment Artifacts

- Build platform: Windows, Administrator account — path prefix C:\Users\Administrator\.cargo\ in all samples
- rustc 1.88.0 (2025-06-23) — C2 Agent and Chrome Harvester components
- rustc 1.91.1 (2025-11-07) — Neon Bootstrap components
- PE creation timestamp: 2025-11-24 on large pe32 C2 agent files (internally consistent with compiler dates)

8.3 Campaign-Specific String Artifacts

- pathprotectionmacs — Chrome extensions.settings key identifying the malicious extension
- 13397690747955841 — hardcoded Chrome extension fake install timestamp (2025-07-22); identical first_install_time and last_update_time
- 4bda9e7e-4913-4dbc-95de-891cbf66598e-errorVal — UUID present in pe32+ C2 agent
- Local\RustBacktraceMutex00000000 — Rust runtime mutex artifact present across Bootstrap and Harvester components

8.4 Technique Indicators (shared across actors)

The following are indicators of the techniques used by GLASSWORM but are not unique to this campaign — they are shared with other threat actors implementing the same publicly documented methods:

- e748f336d85ea5f9dcdf25d8f347a65b4cdf667600f02df6724a2af18a212d26b788a25086910cf3a90313696871f3dc05823730c91df8ba5c4fd9c884b505a8 — SHA-512 of Chrome resources.pak file #146; used as HMAC seed to forge Chrome Secure Preferences MAC values (Phantom Extension technique). Version-specific to a particular Chrome release.
- IElevator::DecryptData, IElevatorEdge::DecryptData — Chrome/Edge V20 app-bound encryption COM bypass
- NtQuerySystemInformation + DuplicateHandle for locked database file access

8.5 Behavioral Indicators

- DLL loaded via Node.js native addon mechanism (require() on a .node file from node_modules)
- Chrome registry key access originating from node.exe: SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\chrome.exe
- Chrome User Data directory enumeration from node.exe process
- Named pipe creation or connection (ConnectNamedPipe) from node.exe
- NtAllocateVirtualMemory / NtProtectVirtualMemory called via direct syscall (not routed through Win32 API)
- BCryptDeriveKeyPBKDF2 called from a Node.js-loaded DLL
- CreateToolhelp32Snapshot followed by NtQuerySystemInformation and DuplicateHandle in the same process — handle theft pattern
- CoCreateInstance + CoSetProxyBlanket targeting Chrome's IElevator COM interface
- SQLite database creation in %APPDATA% or %TEMP% by node.exe
- Chrome Secure Preferences file written with identical first_install_time and last_update_time values

8.6 Rust Crates Identified

Crate	Version	Component	Purpose
neon	1.0.0 / 1.1.1	Bootstrap	Rust-to-Node.js NAPI bridge
rusqlite	0.31.0	C2 Agent	SQLite bindings (SQLite 3.45.0 in pe32, 3.46.1 in pe32+)
url	2.5.7	C2 Agent (pe32)	URL parsing for C2 endpoints
walkdir	2.5.0	C2 Agent (pe32)	Recursive filesystem traversal
icu_collections	2.0.0	C2 Agent (pe32)	Unicode data structures
zip	0.6.6	Chrome Harvester	ZIP archive handling
bzip2	0.4.4	Chrome Harvester	BZip2 decompression
flate2	1.1.8	Chrome Harvester	Deflate compression
winreg	0.52.0	Chrome Harvester	Windows registry access
constant_time_eq	0.1.5	Chrome Harvester	Timing-safe byte comparison

itoa	1.0.17	Chrome Harvester	Integer formatting

9. YARA Detection Rules

Rules are divided into two sets. Rule Set A targets GLASSWORM specifically using artifacts that survive recompilation. Rule Set B targets the attack methodology for detection of similar campaigns by any actor. All rules were evaluated against the Unknown Cyber genomic database; where occurrence counts are noted, they reflect the state of the database at time of analysis.

9.1 Rule Set A — GLASSWORM-Specific Detection

Rule A-1: Neon Bootstrap — NT Syscall EDR Evasion

Detects the Neon Bootstrap DLL using NT syscall resolution strings and neon crate artifacts. Genome occurrence: 1/1 in the Unknown Cyber database. Matches both pe32 and pe32+ variants.

```
rule GLASSWORM_Neon_Bootstrap {
  meta:
    description = "GLASSWORM Neon Bootstrap DLL — direct NT syscall EDR evasion"
    date        = "2026-03"  tlp = "WHITE"
  strings:
    $neon       = "neon-1.0.0" ascii
    $nt_alloc   = "NtAllocateVirtualMemory" ascii
    $nt_protect = "NtProtectVirtualMemory" ascii
    $ntdll      = "ntdll.dll" ascii
    $napi_ver   = "napi_get_version" ascii
    $rust_mutex = "Local\\RustBacktraceMutex" ascii wide
  condition:
    uint16(0) == 0x5A4D and
    $neon and $nt_alloc and $nt_protect and $ntdll and $napi_ver and $rust_mutex
}
```

Rule A-2: Chrome Harvester — Path Resolver Artifact

Detects the Chrome credential harvester using JSON-escaped path strings that are hardcoded verbatim from Chrome's own JSON configuration. The \u003E literal (6 raw bytes: 5c 75 30 30 33 45) has no legitimate reason to appear in any binary accessing Chrome filesystem paths. Genome 0440b2bb occurrence: 1/1 in the Unknown Cyber database.

```
rule GLASSWORM_Chrome_Harvester {
  meta:
    description = "GLASSWORM Chrome credential harvester — JSON path artifact"
    date        = "2026-03"  tlp = "WHITE"
  strings:
    $s1 = "\\u003E>\\Google\\Chrome\\User Data" ascii
    $s2 = ">\\Google\\Chrome\\User Data" ascii
    $s3 = "<\\u003E>\\Google\\Chrome\\User Data" ascii
  condition:
    uint16(0) == 0x5A4D and 2 of them
}
```

Rule A-3: Chrome Extension Installer

Detects the malicious Chrome extension installation component using the hardcoded fake timestamp and the Chrome preferences key fragment. Both artifacts are highly specific: no legitimate Chrome-accessing software hardcodes a static install timestamp identical to the update timestamp, and pathprotectionmacs is the specific key under which the malicious extension is registered.

```
rule GLASSWORM_Extension_Installer {
  meta:
    description = "GLASSWORM Chrome extension installer — fake timestamp +
extension key"
    date        = "2026-03"  tlp = "WHITE"
  strings:
    $timestamp = "13397690747955841" ascii
    $ext_key   = "pathprotectionmacs" ascii
    $no_store  = "\"from_webstore\":false" ascii
    $native    = "nativeMessaging" ascii
  condition:
    uint16(0) == 0x5A4D and
    ($timestamp or $ext_key) and ($no_store and $native)
}
```

Rule A-4: C/C++ Credential Stealer — V10/V20 Key Extraction

Detects the C/C++ credential stealer using its specific operational log strings. The detection value lies entirely in the named capabilities — ExtractDecryptedV10KeyFromLocalState, SendDpapiKeyRecord, and Attempting To Steal Opened File Handles are unambiguously malicious function names with no legitimate equivalent. The [+]/[!] log prefix convention is common across both legitimate and offensive security tooling and is not itself a detection signal.

```
rule GLASSWORM_CC_CredentialStealer {
  meta:
    description = "GLASSWORM C/C++ credential stealer — V10/V20 key extraction"
    date        = "2026-03"  tlp = "WHITE"
  strings:
    $v10  = "ExtractDecryptedV10KeyFromLocalState" ascii
    $v20  = "ExtractDecryptedV20KeyFromLocalState" ascii
    $elev = "IElevator::DecryptData" ascii
    $send = "SendDpapiKeyRecord" ascii
    $log1 = "[+] V10 Decrypted Key:" ascii
    $log2 = "[+] V20 Decrypted Key:" ascii
    $stl  = "[v] Attempting To Steal Opened File Handles" ascii
  condition:
    uint16(0) == 0x5A4D and
    ($v10 and $v20) and 2 of ($elev, $send, $log1, $log2, $stl)
}
```

Rule A-5: C2 Agent — 64-bit Extended Variant

Detects the 64-bit C2 agent using the campaign-specific UUID present only in this variant. The UUID combined with Edge and Brave browser targeting and PBKDF2 is highly specific.

```
rule GLASSWORM_C2_Agent_64bit {
  meta:
    description = "GLASSWORM C2 Agent pe32+ — extended 64-bit variant"
    date        = "2026-03"  tlp = "WHITE"
  strings:
    $uuid  = "4bda9e7e-4913-4dbc-95de-891cbf66598e-errorVal" ascii
    $brave = "BraveSoftware\\Brave-Browser\\Application\\brave.exe" ascii wide
    $edge  = "Microsoft\\Edge\\Application\\msedge.exe" ascii wide
    $kdf   = "BCryptDeriveKeyPBKDF2" ascii
  condition:
    uint16(0) == 0x5A4D and $uuid and 2 of ($brave, $edge, $kdf)
}
```

9.2 Rule Set B — Supply Chain Attack Methodology (Generic)

These rules detect the broader attack pattern regardless of threat actor. They are designed for npm package scanning pipelines and endpoint monitoring in Node.js environments. Tuning against legitimate Rust native addons in the target environment is recommended before deployment.

Rule B-1: Rust Native Addon with NT Syscall Evasion

Any Rust-compiled Node.js native addon performing direct NT syscall resolution has no legitimate use case. This pattern is a strong standalone indicator of malicious intent.

```
rule Suspicious_Rust_NodeAddon_NT_Syscall {
  meta:
    description = "Rust .node addon with direct NT syscall resolution — likely
malicious"
    date        = "2026-03"
  strings:
    $napi    = "napi_module_register" ascii
    $rust_rt = "__rust_begin_short_backtraces" ascii
    $nt_alloc   = "NtAllocateVirtualMemory" ascii
    $nt_protect = "NtProtectVirtualMemory" ascii
    $nt_write   = "NtWriteVirtualMemory" ascii
    $nt_thread  = "NtCreateThreadEx" ascii
  condition:
    uint16(0) == 0x5A4D and ($napi or $rust_rt) and
    1 of ($nt_alloc, $nt_protect, $nt_write, $nt_thread)
}
```

Rule B-2: Rust Native Addon with Browser Credential Targeting

Legitimate Node.js native addons do not access browser credential stores. Any .node file combining NAPI exports with Chrome User Data access is suspicious.

```
rule Suspicious_Rust_NodeAddon_BrowserStealer {
  meta:
    description = "Rust .node addon targeting browser credential stores"
    date        = "2026-03"
  strings:
    $napi    = "napi_module_register" ascii
    $rust_rt = "__rust_begin_short_backtraces" ascii
    $chrome  = "Google\\Chrome\\User Data" ascii wide
    $firefox = "Mozilla Firefox\\firefox.exe" ascii wide
    $brave   = "Brave-Browser\\Application" ascii wide
    $cred    = "Local State" ascii
    $login   = "Login Data" ascii
  condition:
    uint16(0) == 0x5A4D and ($napi or $rust_rt) and
    2 of ($chrome, $firefox, $brave, $cred, $login)
}
```

Rule B-3: Chrome Phantom Extension Installer

Detects any binary implementing the Chrome Secure Preferences MAC bypass technique using the known resources.pak SHA-512 seed. This hash is version-specific to a particular Chrome release and has no legitimate use in any software other than Chrome MAC forgery. Note: this technique is used by multiple independent threat actors.

```
rule Suspicious_Chrome_PhantomExtension_MAC_Bypass {
  meta:
    description = "Chrome Secure Preferences MAC bypass — resources.pak SHA-512
seed"
    reference   = "Synacktiv Phantom Extension research"
    date        = "2026-03"
  strings:
    $mac_seed =
"e748f336d85ea5f9dcdf25d8f347a65b4cdf667600f02df6724a2af18a212d26b788a25086910cf3a9
0313696871f3dc05823730c91df8ba5c4fd9c884b505a8" ascii
    $no_store = "\"from_webstore\":false" ascii
    $native   = "nativeMessaging" ascii
  condition:
    $mac_seed and ($no_store or $native)
}
```

Rule B-4: C/C++ Chrome V20 App-Bound Encryption Bypass

Detects any binary implementing the Chrome app-bound encryption COM bypass. The IElevator::DecryptData string combined with handle theft infrastructure (NtQuerySystemInformation) and Chrome-specific log strings is highly specific to credential stealing tools targeting Chrome 127+.

```
rule Suspicious_Chrome_AppBound_Bypass {
  meta:
    description = "Chrome V20 app-bound encryption COM bypass + handle theft"
    date        = "2026-03"
  strings:
    $elev  = "IElevator::DecryptData" ascii
    $elevE = "IElevatorEdge::DecryptData" ascii
    $proxy = "CoSetProxyBlanket" ascii
    $nt_qi = "NtQuerySystemInformation" ascii
    $dup   = "DuplicateHandle" ascii
    $steal = "Attempting To Steal Opened File Handles" ascii
  condition:
    uint16(0) == 0x5A4D and
    ($elev or $elevE) and $proxy and
    2 of ($nt_qi, $dup, $steal)
}
```

Rule B-5: Generic Suspicious npm Native Addon

Broadest rule — intended for npm package scan pipelines. Detects any Windows DLL with NAPI exports combined with suspicious capability indicators. Apply to .node files extracted from npm tarballs.

```
rule Suspicious_npm_NativeAddon_Windows {
  meta:
    description = "Suspicious Windows DLL with NAPI exports and malicious
capabilities"
    date        = "2026-03"
  strings:
    $napi1 = "napi_module_register" ascii
    $napi2 = "napi_get_version" ascii
    $inj1  = "VirtualAllocEx" ascii
    $inj2  = "NtCreateThreadEx" ascii
    $cred1 = "CryptUnprotectData" ascii
    $cred2 = "Login Data" ascii
    $cred3 = "Local State" ascii
    $pers1 = "CurrentVersion\\Run" ascii wide
    $sys1  = "NtAllocateVirtualMemory" ascii
    $sys2  = "NtProtectVirtualMemory" ascii
  condition:
    uint16(0) == 0x5A4D and 1 of ($napi1, $napi2) and (
      2 of ($inj1, $inj2) or
      2 of ($cred1, $cred2, $cred3) or
      $pers1 or 2 of ($sys1, $sys2)
    )
}
```

10. Assessment and Recommendations

10.1 Threat Assessment

GLASSWORM is a technically capable, multi-component campaign with deliberate architectural choices that reflect operational security awareness. The use of Rust for most components provides large binary sizes that raise the noise floor for static analysis and eliminates symbol names. The C/C++ credential stealer with app-bound encryption bypass shows awareness of current Chrome credential protections (introduced July 2024) and was built to bypass them specifically. The EDR evasion via direct NT syscalls in the entry-point DLL is a current, non-trivial technique.

The multi-browser targeting (Chrome, Firefox, Brave, Edge, Opera) across Windows and macOS, the iterative multi-month development timeline, and the availability of separate 32-bit and 64-bit variants indicate a resourced actor with an ongoing development program rather than a one-time deployment.

The use of the publicly documented Phantom Extension technique is consistent with operational pragmatism — the technique is effective, requires no zero-day, and the complexity of forging Chrome MACs correctly provides some protection against casual detection. The hardcoded Chrome resources.pak hash constrains the attack to a specific Chrome version range and degrades automatically on Chrome updates, suggesting the actor expects to maintain and redeploy updated samples.

10.2 Detection Recommendations

- Apply Rule Set A to retrospective scanning of npm packages and Node.js environments, particularly packages published or updated between June and November 2025.

- Apply Rule Set B (especially B-1 and B-5) to ongoing npm package monitoring pipelines against .node files extracted from tarballs.
- Alert on Chrome registry key access (App Paths\chrome.exe) originating from node.exe or any process loaded from a node_modules directory.
- Alert on named pipe creation or connection from node.exe.
- Alert on NtAllocateVirtualMemory or NtProtectVirtualMemory called via direct syscall (not routed through ntdll.dll exported functions) from processes in npm package directories.
- Monitor for Chrome Secure Preferences modification by non-Chrome processes, particularly writes that set identical first_install_time and last_update_time.
- Monitor for CoCreateInstance calls targeting the Chrome IElevator COM interface from processes other than chrome.exe.
- Search for SQLite database creation events in %APPDATA% or %TEMP% attributed to node.exe.

10.3 Open Questions

- The specific npm package names and versions that distributed these DLLs were not identified in this analysis.
- The macOS delivery mechanism was not determined — no NAPI strings are present in the Mach-O, and the infection vector on macOS remains unknown from this sample set.
- No C2 domains, IP addresses, or network infrastructure were identified. C2 configuration may be passed as a JavaScript argument through the NAPI interface at runtime, or the named pipe may be the primary inter-component channel with network communication handled by a separate unsampled component.
- The C/C++ credential stealer's language choice (MSVC-compiled C/C++ versus Rust for all other components) suggests it was either sourced from a separate toolset, written by a different developer, or predates the Rust components. Attribution of this specific component was not attempted.

Appendix A: Analysis Methodology

Analysis was conducted using the Unknown Cyber platform via MCP connector integration. Techniques applied:

- Binary file typing and metadata extraction (ExifTool via platform pipeline)
- Semantic genome analysis — function-level hashing normalized across compiler variations, with occurrence counts queried against the full Unknown Cyber database
- High-value function extraction with string and API call annotation
- Full string corpus analysis from CSV exports (289 to 3,013 string entries per file), with cross-file pattern matching and Jaccard similarity scoring
- Rust crate identification from embedded panic strings and cargo registry path artifacts
- rustc version identification from embedded compiler commit hashes, cross-referenced against public Rust release history
- Cross-architecture string comparison between pe32 and pe32+ variant pairs
- Chrome extension blob decoding and timestamp analysis
- Cross-referencing of embedded constants against public databases (SQLite release hashes, Chrome resources.pak research)

No dynamic analysis (sandbox execution) was performed. All findings are based on static analysis of binary artifacts and string corpora.

Appendix B: Change Log

This appendix records all corrections and additions made across report versions. Reviewers upgrading from a prior version need only read the section corresponding to the version they hold.

Changes in v2 (from initial release)

The following findings from the initial analysis were corrected or refined:

- 1949cf8c6b5b557f — Initially described as a build machine fingerprint. Confirmed to be the standard crates.io package index directory name derived from the registry URL, shared by all Rust developers using the public registry. Removed from IOCs.
- 1066602b2b1976fe58b5150777cced894af17c803e068f5918390d6915b46e1d — Initially speculated as a C2 certificate fingerprint or key. Confirmed as the SQLite 3.45.0 source amalgamation hash, embedded by all binaries statically linking rusqlite-0.31.0. Removed from IOCs.
- 281fc0e9afc38674b9b0991943b9e9d1e64c6cbdb133d35f6f5c87ff6af38a88 — Confirmed as the SQLite 3.46.1 source ID (released 2025-11-28). Its presence in the 64-bit C2 agent versus SQLite 3.45.0 in the 32-bit variant confirms independent maintenance of the two versions. Removed from IOCs.
- e748f336d85ea5f9dcdf25d8f347a65b4cdf667600f02df6724a2af18a212d26b788a25086910cf3a90313696871f3dc05823730c91df8ba5c4fd9c884b505a8 — Initially presented as a GLASSWORM-specific IOC. Confirmed as the SHA-512 of Chrome resources.pak file #146, used as the HMAC seed in the publicly documented Phantom Extension technique. Shared across multiple threat actors. Recategorised as a technique indicator (Rule Set B only).
- 43253a88 (C/C++ component) — Initially misidentified as a thin file loader based on its low function count. Full string analysis revealed it is a sophisticated C/C++ browser credential stealer implementing Chrome V20 app-bound encryption bypass and file handle theft. Role designation corrected.
- \u003E> Chrome path string — Initially connected to a Unicode-based trigger mechanism. Confirmed as a JSON encoding artifact: the developer hardcoded Chrome JSON path strings verbatim without decoding them. The function at RVA 0x65a0 tries three candidate path prefixes sequentially. No connection to trigger mechanism.

Changes in v3 (from v2)

The following targeted corrections were made to v2. All other content is unchanged from v2.

- Section 5.1 (Neon Bootstrap) — Added description of operational purpose. The previous text described detection properties but did not explain what the Bootstrap does at runtime: it is the entry point that converts a JavaScript require() call into native code execution with EDR evasion already active, before any credential theft or C2 activity begins.

- Section 5.3 (C/C++ Credential Stealer) — Added explicit statement that all capability findings are based on string analysis alone. No function-level disassembly was performed on this component. The claims are well-supported by the strings but readers should know the evidence base.
- YARA Rule A-1 (Neon Bootstrap) — Corrected a rule authoring error: $napi_ver and $rust_mutex were defined but not referenced in the condition, making them dead strings that contributed nothing to matching. Both are now included in the condition. This also tightens the rule: requiring $rust_mutex ensures it only fires on Rust binaries.
- Section 9.1 Rule A-4 description and Section 5.3 closing paragraph — Removed a contradictory claim. The report simultaneously stated the [+]/[!]/[v]/[i] log prefix convention was 'highly unlikely in legitimate software' and that the tool showed signs of reuse across campaigns. Both statements cannot be true. The log format is a common professional practice in both legitimate and offensive tooling. The detection value in Rule A-4 comes from the specific operational function names (ExtractDecryptedV10KeyFromLocalState, SendDpapiKeyRecord, Attempting To Steal Opened File Handles) which name unambiguously malicious capabilities — not from the log format itself. Text corrected accordingly in both locations. The open-questions bullet about the C/C++ component was also reworded to focus on the language choice as the actual anomaly.

Changes in v4 (from v3)

The following corrections were made to v3. All other content is unchanged from v3.

- Version labeling — The title page and document header have been updated to Version 4 / v4 throughout. Version numbering had accumulated inconsistency across prior releases and was not corrected until this version.
- Section 5.1 (Neon Bootstrap) — Corrected the description of the Bootstrap's operational role. The previous text stated the Bootstrap "converts a benign-looking JavaScript import into native code execution", which was inaccurate: the JavaScript-layer decoding is performed by the Bootstrap itself, not by the JavaScript caller. The corrected text states that the Bootstrap receives the invisible-Unicode payload string, decodes it into a PE binary in memory, and executes it. The companion technical document (GLASSWORM: Variation Selector PE Encoding) covers this mechanism in full detail.

Click Here For
More Information