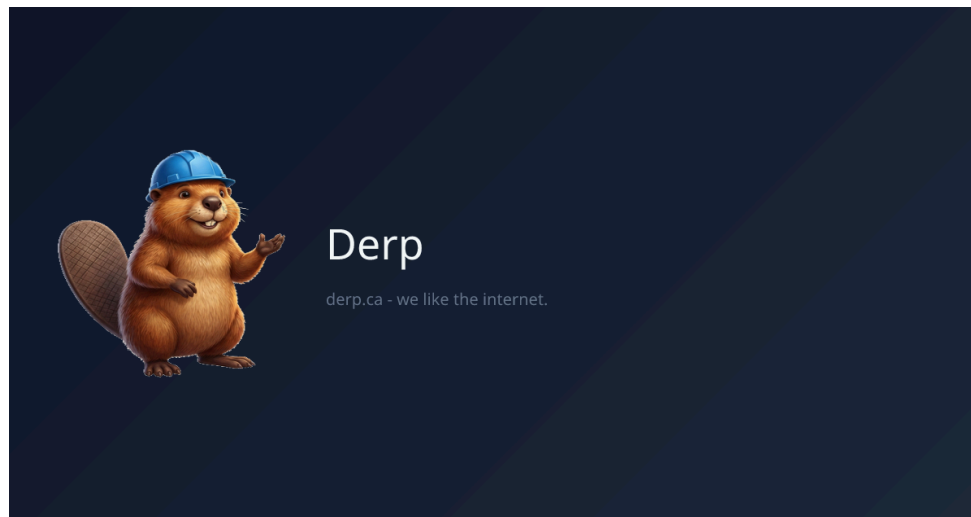


InterLock: full tooling teardown of a ransomware operation

 derp.ca/research/interlock-tooling-teardown

Kirk

March 25, 2026



[malware](#) [ransomware](#) [interlock](#) [reverse-engineering](#) [encryption](#)

InterLock is a double-extortion ransomware operation active since at least October 2024. The group exfiltrates data before encrypting, runs a Tor-based leak site, and deploys encryptors across both FreeBSD/ESXi and Windows. CISA issued advisory AA25-203A in July 2025. Victims span healthcare, education, government, and enterprise environments across multiple countries.

The 15 samples analysed here are not a single intrusion chain. They are the operation's toolkit at a point in time -- four ScreenConnect MSI installers for initial access (one code-signed with a certificate issued to a South African entity), a WebSocket backdoor family called NodeSnake in three language implementations, an NTLM credential harvester, and four ransomware binaries. The samples range from October 2024 to February 2026.

The attribution across sample families rests on overlapping infrastructure and shared code. The JavaScript, Java, and native PE implants all hardcode the same three C2 IP addresses. One JAR sample reuses the JavaScript variant's Cloudflare Tunnel host set exactly. The two NodeSnake PE crypter shells share a byte-identical XOR decryption loop with the Windows ransomware shell -- same instruction sequence, different keys. The ELF and Windows ransomware use the same encryption scheme, the same `.interlock` extension, and the same onion negotiation URL. The four ScreenConnect MSIs share a common relay key, and their two domains obtained Let's Encrypt certificates nine days apart.

NodeSnake is the persistent access layer. It exists as a 74 KB JavaScript variant for Node.js, two Java JAR variants bundling Tyrus and Grizzly, and two native C++ binaries wrapped in crypter shells. All three implementations share the same RC4-encrypted WebSocket framing, the same 4-byte initialize prefix, the same 8-field host profiling format, and the same three C2 IP addresses. The language switches; the protocol does not. The C2 infrastructure runs through free Cloudflare Tunnel endpoints as disposable WebSocket relays, falling back to hardcoded IPs on hosting providers. All three IPs are clean across every threat intel source we checked.

The same crypter framework that wraps the NodeSnake PE backdoors also wraps the Windows ransomware binary. The XOR decryption loop compiles to an identical byte sequence in all three shells. Different keys, same architecture. This links the backdoor and the ransomware to the same build pipeline.

The ransomware encrypts with AES-256-CBC and wraps per-file keys with RSA-4096 OAEP. The PRNG seeding those AES keys is `rand()` XOR `clock()` -- weak enough that key recovery may be feasible with approximate execution timing.

If you operate a threat intelligence platform with API access and can provide a researcher account, please reach out to . Additional data sources directly increase the quality and coverage of the threat intel published here.

Sample overview

Delivery

SHA256	Type	Size	Host	Label
0708a518ef644a3911a717220706190fbd5e5246c533845887c5fbd967953799	MSI	13M	flowmiceornfidgring[.]cc	123
1794e4b1d961bc7fde7b60f4d90e54162e641ec2433574194118a84953e3bc6c	MSI	13M	partyglacierhip[.]top	office
0b92b60be728961d4af818290880d0f1178314bee94482facedb89b6168cc783	MSI	13M	partyglacierhip[.]top	superder
5a499038069af6abe8212d2acb71655fe20b08d7a3150c438f00d81e9043c422	MSI	13M	partyglacierhip[.]top	internal

Implants

SHA256	Type	Size	Role
4a738a947f4e4b22f36b532a2b7a3af1ae0fe7481d954d467eae80dd765efab6	JS	74K	NodeSnake JavaScript WebSocket implant
21d9ed48d51a5b5edae7eb7f99d1648a3ce7d419bc46234143c37dec4638c60d	JAR	5.1M	NodeSnake Java WebSocket implant
c9882720ac7095d416d2f80cf997b55217e06fb597e84f30a4a38027be30467e	JAR	5.1M	NodeSnake Java WebSocket implant (rotated host set)
5a0d393de439aebc236a87f3ff18c89c5634f8b7b9331c54986b113b75e8df7f	PE	1.5M	Crypter shell -> NodeSnake C++ backdoor
396eb0e817d90cf366b5648f9a97c51bfb37737af13dc4a9e1a768885a867dc5	PE	1.7M	Crypter shell -> NodeSnake C++ backdoor

Credential theft and helpers

SHA256	Type	Size	Role
fb68797872dedd29a86db18ca41350155249718b3b0372e1985c980d4e09edf9	PE	52K	NtlmThief NTLM credential harvester
6c8efbcef3af80a574cb2aa2224c145bb2e37c2f3d3f091571708288ceb22d5f	PE	25K	Fullscreen shock image display

Ransomware

SHA256	Type	Size	Target	First seen
e86bb8361c436be94b0901e5b39db9b6666134f23cce1e5581421c2981405cb1	ELF	789K	FreeBSD/ESXi	2024-10-09
28c3c50d115d2b8ffc7ba0a8de9572fbe307907aaae3a486aab8c0266e9426f	ELF	789K	FreeBSD/ESXi	2024-10-29
f00a7652ad70ddb6871eeef5ece097e2cf68f3d9a6b7acfbfd33f82558ab50e	ELF	1.2M	FreeBSD/ESXi	2024-11-04

SHA256	Type	Size	Target	First seen
a26f0a2da63a838161a7d335aaa5e4b314a232acc15dcabdb6f6dbec63cda642	PE	1.9M	Windows	2024-10-16

Decrypted payloads (extracted from crypter shells)

SHA256	Size	Compiled	Compiler	Source	Paylc
85030b66cb1558a2c1eb249b0c6b1407d393ba996cb96be499efeaef487c97f55	405K	2026-01-30	LLVM MinGW	396eb0e8	Node backc
2470fd1ea4d64caa4e162f382775ad9d1c9ae7dcc25020497dc8a4b50ac411a8	275K	2026-02-13	GCC MinGW	5a0d393d	Node backc
0870c0136c0b7092dc79f3169f58a6e32011743586a18ed8a300010f34c2d44d	718K	2024-10-11	GCC MinGW	a26f0a2d	InterL ranso

ScreenConnect delivery

Four MSI installers deliver ConnectWise ScreenConnect 25.2.4.9229. All four install an identical 14-file payload to Program Files/ScreenConnect Client (f424f8352c6b46fa)/. The only variation across the four builds is the SERVICE_CLIENT_LAUNCH_PARAMETERS MSI property and the per-sample ProductCode.

The launch parameter URL follows the pattern ?e=Access&y=Guest&h=<host>&p=8041&k=<key>&c=<label>. Two ScreenConnect hosts are configured:

- flowmiceornfidgring[.]cc on port 8041 (sample 0708a518, label 123)
- partyglacierhip[.]top on port 8041 (samples 1794e4b1, 0b92b60b, 5a499038, labels office, superdental, internal)

The k= value is identical across all four MSI samples. SHA256 of the decoded value: 790a4f45a50255c1d9cdfa9398886097d40a508d846d6e7dde0dfec0f09b0e80.

Only 0708a518 carries a valid Authenticode signature:

Field	Value
Subject	UMNOTHO SOFTWARE SA CC
Issuer	SSL.com Code Signing Intermediate CA RSA R1
Serial	4F10DF22BF708616F9437AB06E0FAAB0
Valid	2025-11-19 to 2026-11-18

The other three MSI samples are unsigned. The ServiceInstall.idt table passes SERVICE_CLIENT_LAUNCH_PARAMETERS directly into the service arguments for the installed ScreenConnect client. The URL scheme is sc-f424f8352c6b46fa and the credential provider CLSID is {6FF59A85-BC37-4CD4-26F5-546AAE262457}.

The campaign labels -- office, superdental, internal -- suggest these MSIs target different organisations or departments within the same operation.

NodeSnake implant family

NodeSnake is a WebSocket-based backdoor implemented in three languages: JavaScript (Node.js), Java, and native C++. All three tiers use the same transport protocol, the same RC4-encrypted message framing, the same host profiling format, and the same C2 infrastructure. The difference is capability: the native PE variant adds TCP tunnelling, thread execution hijacking, and anti-debugging checks not present in the scripted versions.

JavaScript WebSocket implant

The JavaScript sample is a 74 KB Node.js script. The source is a single obfuscated line built around a rotated string array with an RC4-style decoder function. After deobfuscation, the implant's structure is readable.

Transport. The implant connects over `ws://` and rotates across nine Cloudflare Tunnel domains plus three fallback IP addresses:

Type	Target
Tunnel	<code>gzip-picked-istanbul-maple.trycloudflare[.]com</code>
Tunnel	<code>logan-practitioners-percent-cartridges.trycloudflare[.]com</code>
Tunnel	<code>silk-lift-porter-correctly.trycloudflare[.]com</code>
Tunnel	<code>offers-listing-screenshot-alpha.trycloudflare[.]com</code>
Tunnel	<code>wives-bufing-humans-prot.trycloudflare[.]com</code>
Tunnel	<code>communist-flying-provision-calendar.trycloudflare[.]com</code>
Tunnel	<code>safe-accepted-salem-early.trycloudflare[.]com</code>
Tunnel	<code>forget-canal-chancellor-mas.trycloudflare[.]com</code>
Tunnel	<code>electrical-protect-molecular-underground.trycloudflare[.]com</code>
IP	<code>77.42.75.119</code>
IP	<code>23.227.203.123</code>
IP	<code>172.86.68.64</code>

The WebSocket path is generated dynamically: `/<random>?v=2;<random>[/<random>...]&<sha256(time_bucket)>[/<random>...]`. The time bucket is SHA-256 over an 8-byte little-endian Unix timestamp with the low 12 bits cleared, creating 4096-second (approximately 68-minute) rotation windows.

Message framing. Every message is wrapped in a `SocksMsg` envelope. Each message gets a fresh 16-byte random RC4 key. The structure is a 25-byte header followed by the payload, both RC4-encrypted with the per-message key. The initialize handshake uses the fixed prefix `92 01 88 fe`.

Host profiling. On connection, the implant collects and sends eight fields:

Field	Collection method
<code>connected_to</code>	Current C2 target
<code>domain</code>	Machine domain

Field	Collection method
hostname	Machine hostname
username	Current user
version_os	OS version via PowerShell <code>systeminfo /FO CSV</code>
version_build	Hardcoded <code>2</code>
runas	ADMIN or USER (checked via <code>net1 session</code>)
type_file	File type identifier

Operator commands. The implant supports 12 message types:

Type	Value	Capability
SOCKS5	0x05	SOCKS5 proxy (IPv4, IPv6, domain)
PING	0xff	Keep-alive
TERMINAL	0xa0	Interactive <code>cmd.exe</code> shell
TERMINAL_COMMAND	0xa1	One-shot <code>cmd.exe /c</code> , output to <code>C:\Users\Public\<random>.txt</code>
SLEEP	0x06	Operator-controlled delay
DISCONNECT	0x07	Close tracked connection
OFF	0x0b	Close transport and exit
DELETE	0x0c	<code>fs.rmSync(__filename)</code>
FILE_GET_SOCKS	0x21	Stream local file to operator
FILE_PUT_SOCKS	0x22	Write operator-supplied file
UPDATE	0xe0	Overwrite self, relaunch via <code>process.execPath</code>
INITIALIZE	0xf0	Handshake (responses: 0x00 continue, 0x01 delete, 0x02 exit)

Retry logic. The connection retry counter starts at `-10` and increments to `40`. Values at or below zero trigger 10-second sleeps. Positive values sleep `300000 * counter` milliseconds (5 minutes per unit). If the counter passes 40, the implant deletes itself.

Java WebSocket implants

Two JAR files implement the same implant in Java. Both are approximately 5.1 MB and bundle Tyrus 1.17 (WebSocket client), Grizzly 2.3.22 (HTTP framework), JNA 5.14.0, and `json-20210307.jar`. The main class is `main.Main`.

The internal class structure mirrors the JavaScript version:

Class	Role
main/WebSocket	Transport handler
main/Socks	SOCKS proxy
main/crypto/Protection	String decoding (stack strings)
main/crypto/RC4	Stream cipher
main/msg/SocksMsg	Message framing
main/threads/Socks5Thread	SOCKS5 handler
main/threads/TerminalThread	Interactive shell
main/threads/TerminalCommandThread	One-shot command
main/threads/ReadFileThread	File download
main/threads/WriteFileThread	File upload
main/threads/UpdateThread	Self-update

The transport protocol is identical to the JavaScript variant: `ws://`, same path shape, same RC4 `SocksMsg` framing, same initialize prefix `92 01 88 fe`, same message type values. The retry counter runs from `-10` to `40` with `counter * 5` minute sleeps for positive values.

Sample `21d9ed48` uses the exact same nine Cloudflare Tunnel domains and three fallback IPs as the JavaScript implant. Sample `c9882720` rotates the first six tunnel domains while keeping the last three domains and all three IPs:

Domain (c9882720 only)
playback-attributes-interviews-processing.trycloudflare[.]com
chronic-dividend-amendments-das.trycloudflare[.]com
yen-hansen-cartoon-aims.trycloudflare[.]com
bridal-custody-private-bodies.trycloudflare[.]com
planners-mixing-edmonton-endless.trycloudflare[.]com
module-source-tree-diverse.trycloudflare[.]com

Both JARs declare an `IP_PORT = 2234` constant that is never referenced in the actual transport code. Both use the same host profiling commands as the JavaScript variant: `systeminfo /FO CSV` via PowerShell and `net1 session` via `cmd.exe`.

The Java variant adds two features not present in the JavaScript version. The `UpdateThread` creates a self-deleting scheduled task:

```
schtasks /create /tn "<name>" /tr "cmd.exe /c del \"<jar>\" & schtasks /delete /tn \"<name>\" /f" /sc once /st <HH:mm> /f
```

The update path uses a ping delay to allow file replacement:

```
ping 127.0.0.1 -n 5 > nul & move /Y "<tmp>" "<jar>" & start "" "<java>" -jar "<jar>"
```

Native PE NodeSnake backdoor

Two PE samples serve as crypter shells protecting native C++ NodeSnake backdoors. The crypter framework is detailed in the below. Here we focus on the decrypted payloads.

Crypter shells:

Property	5a0d393d	396eb0e8
Export DLL	_socks.dll	m5b2G1qAyWB4.dll
Version	aaa v2.1.3	312 v2.0.7
ImageBase	0x30e4d0000	0x180000000
XOR key	46a6e09639e77b6f	00244c6e55becd26

After decryption, both payloads export `socks.dll::start` and import `WS2_32.dll`, `KERNEL32.dll`, `ADVAPI32.dll`, and `msvcrt.dll`.

Decrypted payloads:

Property	396e payload	5a0d payload
SHA256	85030b66cb1558a2c1eb249b0c6b1407d393ba996cb96be499efea487c97f55	2470fd1ea4d64caa4e162f382775ad9d1c9ae7dcc25
Size	405,504 bytes	274,944 bytes
Compiled	2026-01-30 14:53:23 UTC	2026-02-13 20:54:06 UTC
Compiler	LLVM MinGW (libc++)	GCC MinGW-w64 (libstdc++)

Both payloads hardcode all three C2 IPs: `172.86.68.64`, `23.227.203.123`, `77.42.75.119`. Both use the same 8-field host profiling format documented in the JavaScript section above.

Thread architecture. The native implant runs a multi-threaded design:

Thread	Purpose
SocksThread	SOCKS4 proxy handler
Socks5Thread	SOCKS5 proxy handler
TcpTunnel	TCP tunnel relay
ReadFileThread	File download to operator
WriteFileThread	File upload from operator
TerminalThread	Interactive <code>cmd.exe</code> shell

Thread	Purpose
TerminalCommandThread	One-shot <code>cmd.exe /c</code>
UpdateThread	Self-update and replacement

Capabilities beyond JS/JAR. The native variant adds several features not present in the scripted tiers:

- **TCP tunnel relay** (`TcpTunnel1`): forwards arbitrary TCP connections through the implant, allowing the operator to reach internal hosts.
- **Thread execution hijacking**: uses `SetThreadContext/GetThreadContext` to inject into running threads.
- **Anti-debugging**: timing checks via `GetTickCount` and `QueryPerformanceCounter`, plus `IsDebuggerPresent`.
- **DLL execution**: loads payloads via `rundll32.exe`.
- **Privilege-aware behaviour**: distinguishes `ADMIN` from `USER_NO_ADMIN` to adjust execution paths.

Crypto stack: RC4 for transport encryption, SHA256 for hashing, MurmurHash3 for non-cryptographic hashing, XOR encoding, and Mersenne Twister PRNG.

Persistence. Like the Java variant, the PE uses self-deleting scheduled tasks (`schtasks /create /sc once`) and replaces itself via `move /Y` with a temp file at `/tmp/test.update`.

Cross-tier comparison

Feature	JavaScript	Java	Native PE
Language	Node.js	Java (Tyrys/Grizzly)	C++ (MinGW)
Size	74 KB	~5.1 MB	275-405 KB (payload)
Transport	<code>ws://</code>	<code>ws://</code>	Raw TCP
Framing	RC4 <code>SocksMsg</code>	RC4 <code>SocksMsg</code>	RC4 <code>SocksMsg</code>
Initialize prefix	<code>92 01 88 fe</code>	<code>92 01 88 fe</code>	<code>92 01 88 fe</code>
Host profiling	8 fields	8 fields	8 fields
SOCKS proxy	SOCKS5	SOCKS5	SOCKS4 + SOCKS5
TCP tunnel	No	No	Yes
Thread hijacking	No	No	Yes
Anti-debug	No	No	Yes
Self-update	Yes	Yes (schtasks)	Yes (schtasks)
Self-delete	<code>fs.rmSync</code>	schtasks	schtasks
Obfuscation	String array rotation	Stack strings	Crypter shell
C2 IPs	All 3	All 3	All 3

The protocol is the same across all three. The native PE adds operational capabilities -- TCP tunnelling, thread hijacking, anti-debugging -- while the scripted variants offer faster deployment with lighter tooling requirements.

NtlmThief

fb687978 is a 52 KB MSVC DLL exporting NtlmThief.dll::start. It performs NTLM credential theft using the local SSPI interface.

Field	Value
SHA256	fb68797872dedd29a86db18ca41350155249718b3b0372e1985c980d4e09edf9
Size	52,224 bytes
Compiled	2026-02-11
Compiler	MSVC
PDB	C:\\Users\\tututu\\Desktop\\other\\NtlmThief\\x64\\Release\\NtlmThief.pdb
Export	NtlmThief.dll::start
Imphash	60447f89c0eb870e071cb55cd57678cf

The tool uses a fixed NTLM challenge of 1122334455667788. It calls AcquireCredentialsHandlew to obtain credentials, then runs both InitializeSecurityContextw (client) and AcceptSecurityContext (server) locally against the fixed challenge. The output is a crackable hash in the format domain::user:hash:hash:challenge.

Two CLI flags:

- downgrade: modifies three registry values under HKLM\\SYSTEM\\CurrentControlSet\\Control\\Lsa to force NTLMv1 authentication:
 - LMCompatibilityLevel -- downgraded to allow NTLMv1
 - NtlmMinClientSec -- reduced security requirements
 - RestrictSendingNTLMTraffic -- restrictions removed
- pid <PID>: steals tokens from a target process via OpenProcess -> OpenProcessToken -> DuplicateTokenEx -> ImpersonateLoggedOnUser, then runs the NTLM capture under that process's identity.

NtlmThief has no network capability. It is designed to be dropped and executed through one of the NodeSnake implant shells, with output captured via the TERMINAL_COMMAND message type.

The second helper, 6c8efbce (25 KB), is a GDI+ application that displays a fullscreen borderless window containing a 5,806-byte JPEG embedded in the .data section. It exits on ESC. The image is explicit content. The binary was compiled November 2024 with GCC 14.1.0 / MSYS2 MinGW-w64. It has no network capability and no operational relationship to the rest of the toolkit.

Crypter framework

Three PE samples use an identical crypter framework: the two NodeSnake shells (5a0d393d, 396eb0e8) and the Windows ransomware shell (a26f0a2d, matrix.exe).

Identification. The XOR decryption loop compiles to the same byte sequence in all three binaries:

```
48 8b 04 1f    mov rax, [rdi + rbx]      ; read encrypted qword from .rdata
48 33 45 f0    xor rax, [rbp - 0x10]     ; XOR with fixed 8-byte key
48 89 04 1e    mov [rsi + rbx], rax      ; write decrypted qword to buffer
```

Byte pattern: 488b041f483345f04889041e. This sequence appears exactly once in each binary and serves as a reliable detection anchor.

Architecture. The crypter follows a fixed sequence:

1. CRT initialization allocates a ~197 KB output buffer.
2. `time(NULL)` calls seed timing values and introduce anti-analysis jitter.
3. PEB walk (`gs:[0x60]` -> `Ldr` -> `InMemoryOrderModuleList`) resolves `GetProcAddress`.
4. A custom export resolver locates additional API functions.
5. Stack-built "VirtualProtect" string is resolved and called to mark the entire image as `PAGE_EXECUTE_READWRITE`.
6. The XOR loop decrypts the `.rdata` blob 8 bytes at a time using a fixed key.
7. Zlib decompresses the XOR'd output into the allocated buffer, producing a PE payload.
8. Control transfers to the payload.

Between each XOR iteration, approximately 4.5 million junk instructions execute. These are valid x86-64 operations that produce no useful result. They inflate the binary, slow emulation, and make stepping through the loop impractical.

Obfuscation layers:

- Three TLS callbacks per sample, all junk stubs or dead computation.
- A fake import table -- the real API resolution happens through the PEB walk. Neither crypter shell imports `ws2_32.dll` or any networking library.
- The `.data` section (~180 KB) is filled with English dictionary words as decoy strings.
- 21 named exports per shell with randomized names (e.g., `mole_handbill_rallies`).
- Dead code references to `_execve`, `_spawnve`, `SetThreadContext`, and `MoveFileTransactedW` -- none of these are called in the live execution path.
- The `.rdata` encrypted region measures approximately 7.95 bits/byte entropy, consistent with encrypted data.

Per-sample details:

Sample	Stage	XOR key	Source VA	Blob size	Payload size	Payload
396eb0e8	4	00244c6e55becd26	0x180190b5a	166,774	405,504	NodeSnake backdoor
5a0d393d	4	46a6e09639e77b6f	0x30e63bc9b	117,477	274,944	NodeSnake backdoor
a26f0a2d	7	a32f2d26e8b5fe1b	0x14019c6f9	283,479	718,607	InterLock ransomware

The same framework, the same byte-level decryption loop, three different keys, two different payload families. The NodeSnake backdoors and the InterLock ransomware are products of the same build tooling.

InterLock ransomware

ELF variant (FreeBSD/ESXi)

Three ELF binaries target FreeBSD 10.4 on x86-64. All are statically linked and use LibTomCrypt for cryptographic operations with LibTomMath for the underlying big-number arithmetic.

Samples `e86bb836` and `28c3c50d` are nearly identical. They differ in three fields: the BuildID, the embedded RSA-4096 public key, and the Company ID. Two Company IDs means two victims. Sample `f00a7652` is a stripped variant of `28c3c50d` with corrupted ELF section headers and 460 KB of `0xFF` padding appended. It carries the same Company ID as `28c3c50d`.

Encryption scheme:

1. `generateKey()` produces a 48-byte random buffer. The PRNG is `rand()` XOR `clock()`, seeded once at startup. The first 32 bytes become the AES-256 key. Bytes 32-48 become the AES-CBC initialisation vector.
2. Files are encrypted in 1 MB chunks (`0x100000` bytes). Each chunk is AES-256-CBC encrypted via LibTomCrypt's `cbc_encrypt()`, then written back in place using `fseeko` backward seeks.
3. The 48-byte per-file key is wrapped with the embedded RSA-4096 public key using OAEP padding (mode 1). The wrapped blob (up to `0x500` bytes) and a 2-byte length field are appended to the end of the encrypted file.
4. Encrypted files are renamed with the `.interlock` extension.
5. PKCS padding handles files not aligned to the 16-byte AES block size.

PRNG weakness. `clock()` returns process CPU time with microsecond granularity. With a rough estimate of when the ransomware executed, the key space narrows to the range of plausible `clock()` return values XOR'd with the output of the linear congruential generator behind `rand()`. This is not a trivial brute force, but it is orders of magnitude smaller than 256-bit key space.

Threading. The binary reads the CPU count via `sysconf(0x3a)` and spawns that many worker threads. A global queue protected by a pthread mutex feeds file paths to the workers. Each thread polls the queue with a 5 ms sleep (`usleep(0x1388)`) between iterations.

Ransom note. `!__README__.txt` is dropped in every traversed directory:

Field	Value
Header	INTERLOCK / CRITICAL SECURITY ALERT
Onion URL	ebhmkoohccl45qesdbvrjqtyro2hmhkmh6vkyfyjjzfl1m3ix72aqaid[.]onion
Deadline	96 hours
Company ID	60-character alphanumeric string (per-victim)

Exclusions. The ransomware skips 24 Linux FreeBSD system directories (`bin`, `boot`, `dev`, `etc`, `lib`, `proc`, `sys`, `usr`, `var`, and others) and 13 VMware ESXi configuration extensions (`.sf`, `.b00`, `.v00` through `.v07`, `.t00`, `boot.cfg`). The ESXi exclusions preserve hypervisor boot while encrypting the virtual machine disk files.

CLI flags: `--directory`, `--file`, `--delete` (self-delete after encryption), `--system` (no-op).

There is no network C2. No socket or connect calls appear in the live execution path. The ELF variant is a standalone encryption tool.

Windows PE variant

The Windows ransomware is `matrix.exe`, versioned `v2.1.9` with copyright `matrix Team 2024`. The manifest specifies `asInvoker` -- no UAC elevation request. The outer binary is a crypter shell using the same framework documented above, with XOR key `a32f2d26e8b5fe1b`.

The decrypted payload is a 718 KB PE compiled 2024-10-11 with GCC MinGW. It imports `KERNEL32.dll`, `msvcrt.dll`, and `wevtapi.dll`.

Encryption. Identical to the ELF variant: AES-256-CBC per-file keys generated from the same weak `rand()` XOR `clock()` PRNG, wrapped with RSA-4096 OAEP via LibTomCrypt. Same `.interlock` extension. Same ransom note filename and onion URL.

Anti-forensics. The Windows variant imports `wevtapi.dll` and calls `EvtClearLog` to wipe Windows event logs. This is the only variant in the toolkit that clears event logs.

Persistence. A daily scheduled task runs the ransomware at 20:00 as SYSTEM:

```
schtasks /create /sc DAILY /tn "TaskSystem" /tr "cmd /C cd %s && %s" /st 20:00 /ru system > nul
```

The task is cleaned up after execution:

```
schtasks /delete /tn TaskSystem /f > nul
```

DLL payloads are executed via `rundll32.exe %s,run %s`.

CLI flags: `--directory` (target specific path), `--file` (target specific file), `--delete` (self-delete).

Exclusions. 25 extensions are skipped: `.bat`, `.bin`, `.cab`, `.cmd`, `.com`, `.cur`, `.diagcab`, `.diagcfg`, `.diagpkg`, `.dll`, `.drv`, `.exe`, `.hlp`, `.hta`, `.ico`, `.ini`, `.msi`, `.ocx`, `.ps1`, `.psm1`, `.scr`, `.sys`, `.url`, `Thumbs.db`, and the ransom note itself. 14 directories are skipped, including `Windows`, `ProgramData`, `AppData`, `Boot`, `Recovery`, `System Volume Information`, and `Windows Defender` paths.

Company ID: ELCI3EJW80U6DEII01Y1Y32I7IESI3XH693KEQJG5997MPF9SH7HE0E5J702.

ELF vs Windows comparison

Feature	ELF (FreeBSD)	PE (Windows)
Compiled	Oct-Nov 2024	2024-10-11
Obfuscation	None (plaintext strings)	Crypter shell (8-byte XOR + zlib)
Encryption	AES-256-CBC + RSA-4096 OAEP	AES-256-CBC + RSA-4096 OAEP
PRNG	rand() XOR clock() (weak)	rand() XOR clock() (weak)
Extension	.interlock	.interlock
Ransom note	!__README__.txt	!__README__.txt
Onion URL	Same	Same
Persistence	None	Daily schtask at 20:00
Anti-forensics	None	EvtClearLog (event log wipe)
Self-delete	--delete flag	--delete flag
Network C2	None	None
Crypto library	LibTomCrypt	LibTomCrypt
Threading	CPU count (pthread)	CPU count

The encryption is the same. The Windows variant adds operational features -- persistence, event log clearing, and the crypter shell -- that the ELF variant does without.

C2 infrastructure

NodeSnake C2 IPs

Three IP addresses appear in all three NodeSnake tiers (JavaScript, Java, and native PE payloads):

IP	ASN	Provider	Type	Country	rDNS
172.86.68.64	AS14956	RouterHosting LLC	hosting	US	64.68.86.172.static.cloudzy.com
23.227.203.123	AS29802	Hivelocity, Inc.	hosting	US	23-227-203-123.static.hvvc.us
77.42.75.119	AS24940	Hetzner Online GmbH	hosting	DE	static.119.75.42.77.clients.your-server.de

All three are VPS or dedicated hosting providers. 172.86.68.64 resolves to a Cloudzy hostname. Shodan InternetDB shows port 22 (OpenSSH 9.6p1) and port 5000 (Python/Gunicorn) on 172.86.68.64. Port 5000 running a Python WSGI server is consistent with a WebSocket C2 backend. 23.227.203.123 exposes only port 22 (OpenSSH 9.6p1). 77.42.75.119 exposes ports 80 and 443.

ScreenConnect domains

Domain	Port	First cert	Issuer
flowmiceornfidgring[.]cc	8041	2025-12-04	Let's Encrypt R13
partyglacierhip[.]top	8041	2025-12-13	Let's Encrypt R13

Both domains obtained certificates in December 2025. The MSI samples were tagged on 2026-03-24.

Cloudflare Tunnel domains

Fifteen `trycloudflare[.]com` subdomains serve as primary WebSocket C2 targets across the JavaScript and Java implant samples. These are free Cloudflare Tunnel endpoints -- no account required, no persistent infrastructure. The operator generates new tunnel URLs as needed and hardcodes them into each build.

Nine domains appear in the JavaScript sample. Sample `21d9ed48` (JAR) reuses the same nine. Sample `c9882729` (JAR) shares the last three and rotates the first six to a different set. The full domain list is in the IOC summary below.

Threat intel coverage

All three C2 IPs are absent from GreyNoise, IPsum, Feodo Tracker, ThreatFox, URLhaus, SSLBL, Pulsedive, and the derpcron Triage C2 pipeline. None are flagged as scanning or malicious by any source checked on 2026-03-25.

IOC summary

Network

Type	Value	Context
IP	172.86.68.64	NodeSnake C2 (RouterHosting/Cloudzy, US)
IP	23.227.203.123	NodeSnake C2 (Hivelocity, US)
IP	77.42.75.119	NodeSnake C2 (Hetzner, DE)
Domain	flowmiceornfidgring[.]cc	ScreenConnect host (port 8041)
Domain	partyglacierhip[.]top	ScreenConnect host (port 8041)
Tunnel	gzip-picked-istanbul-maple.trycloudflare[.]com	JS + JAR C2
Tunnel	logan-practitioners-percent-cartridges.trycloudflare[.]com	JS + JAR C2
Tunnel	silk-lift-porter-correctly.trycloudflare[.]com	JS + JAR C2
Tunnel	offers-listing-screenshot-alpha.trycloudflare[.]com	JS + JAR C2
Tunnel	wives-bufing-humans-prot.trycloudflare[.]com	JS + JAR C2
Tunnel	communist-flying-provision-calendar.trycloudflare[.]com	JS + JAR C2
Tunnel	safe-accepted-salem-early.trycloudflare[.]com	JS + JAR C2

Type	Value	Context
Tunnel	forget-canal-chancellor-mas.trycloudflare[.]com	JS + JAR C2
Tunnel	electrical-protect-molecular-underground.trycloudflare[.]com	JS + JAR C2
Tunnel	playback-attributes-interviews-processing.trycloudflare[.]com	JAR C2 (c9882720 only)
Tunnel	chronic-dividend-amendments-das.trycloudflare[.]com	JAR C2 (c9882720 only)
Tunnel	yen-hansen-cartoon-aims.trycloudflare[.]com	JAR C2 (c9882720 only)
Tunnel	bridal-custody-private-bodies.trycloudflare[.]com	JAR C2 (c9882720 only)
Tunnel	planners-mixing-edmonton-endless.trycloudflare[.]com	JAR C2 (c9882720 only)
Tunnel	module-source-tree-diverse.trycloudflare[.]com	JAR C2 (c9882720 only)
Onion	ebhmkoohcc145qesdbvrjqtyro2hmhkmh6vkyfyjjzfllm3ix72aqaid[.]onion	Ransomware negotiation portal

Host

Type	Value	Context
Export	socks.dll::start	NodeSnake PE payload entry point
Export	NtlmThief.dll::start	NtlmThief entry point
PDB	C:\\Users\\tututu\\Desktop\\other\\NtlmThief\\x64\\Release\\NtlmThief.pdb	NtlmThief build path
Scheduled task	TaskSystem	InterLock Windows ransomware persistence
Registry	HKLM\\SYSTEM\\CurrentControlSet\\Control\\Lsa\\LMCompatibilityLevel	NtlmThief NTLMv1 downgrade
Registry	HKLM\\SYSTEM\\CurrentControlSet\\Control\\Lsa\\NtlmMinClientSec	NtlmThief NTLMv1 downgrade
Registry	HKLM\\SYSTEM\\CurrentControlSet\\Control\\Lsa\\RestrictSendingNTLMTraffic	NtlmThief NTLMv1 downgrade
Install path	Program Files/ScreenConnect Client (f424f8352c6b46fa)/	ScreenConnect payload
URL scheme	sc-f424f8352c6b46fa	ScreenConnect client identifier
Extension	.interlock	Encrypted file extension
Ransom note	!__README__.txt	InterLock ransom note

Type	Value	Context
Temp file	%s/tmp%d.wasd	InterLock Windows variant temp pattern

Hashes

SHA256	Role
0708a518ef644a3911a717220706190fbd5e5246c533845887c5fbd967953799	MSI (flowmiceornfidgring, signed)
1794e4b1d961bc7fde7b60f4d90e54162e641ec2433574194118a84953e3bc6c	MSI (partyglacierhip, office)
0b92b60be728961d4af818290880d0f1178314bee94482facedb89b6168cc783	MSI (partyglacierhip, superdental)
5a499038069af6abe8212d2acb71655fe20b08d7a3150c438f00d81e9043c422	MSI (partyglacierhip, internal)
4a738a947f4e4b22f36b532a2b7a3af1ae0fe7481d954d467eae80dd765efab6	NodeSnake JS implant
21d9ed48d51a5b5edae7eb7f99d1648a3ce7d419bc46234143c37dec4638c60d	NodeSnake JAR implant
c9882720ac7095d416d2f80cf997b55217e06fb597e84f30a4a38027be30467e	NodeSnake JAR implant (rotated hosts)
5a0d393de439aebc236a87f3ff18c89c5634f8b7b9331c54986b113b75e8df7f	Crypter shell -> NodeSnake PE
396eb0e817d90cf366b5648f9a97c51bfb37737af13dc4a9e1a768885a867dc5	Crypter shell -> NodeSnake PE
85030b66cb1558a2c1eb249b0c6b1407d393ba996cb96be499efeaf487c97f55	NodeSnake PE payload (decrypted)
2470fd1ea4d64caa4e162f382775ad9d1c9ae7dcc25020497dc8a4b50ac411a8	NodeSnake PE payload (decrypted)
fb68797872dedd29a86db18ca41350155249718b3b0372e1985c980d4e09edf9	NtlmThief
6c8efbcef3af80a574cb2aa2224c145bb2e37c2f3d3f091571708288ceb22d5f	Shock image display
e86bb8361c436be94b0901e5b39db9b6666134f23cce1e5581421c2981405cb1	InterLock ELF ransomware
28c3c50d115d2b8ffc7ba0a8de9572fbe307907aaae3a486aabdb8c0266e9426f	InterLock ELF ransomware
f00a7652ad70ddb6871eeef5ece097e2cf68f3d9a6b7acfbfd33f82558ab50e	InterLock ELF ransomware (stripped)
a26f0a2da63a838161a7d335aaa5e4b314a232acc15dcabdb6f6dbec63cda642	Crypter shell -> InterLock PE ransomware
0870c0136c0b7092dc79f3169f58a6e32011743586a18ed8a300010f34c2d44d	InterLock PE ransomware payload (decrypted)

Behavioural

Indicator	Context
RC4 per-message key (16 bytes) + 25-byte header	NodeSnake WebSocket framing

Indicator	Context
Initialize prefix <code>92 01 88 fe</code>	NodeSnake handshake marker
Path pattern <code>/<random>?v=2;...&sha256>...</code>	NodeSnake WebSocket URL
<code>systeminfo /FO CSV ConvertFrom-Csv ConvertTo-Json</code>	NodeSnake host profiling (PowerShell)
<code>cmd.exe /c net1 session</code>	NodeSnake privilege check
<code>schtasks /create /sc once with self-deleting /tr</code>	NodeSnake persistence/cleanup
<code>schtasks /create /sc DAILY /tn "TaskSystem" /st 20:00 /ru system</code>	InterLock Windows persistence
<code>EvtClearLog</code> via <code>wevtapi.dll</code>	InterLock Windows anti-forensics
NTLM challenge <code>1122334455667788</code>	NtlmThief fixed challenge
XOR byte pattern <code>488b041f483345f04889041e</code>	Shared crypter decryption loop
<code>C:\\Users\\Public\\<random>.txt</code>	NodeSnake command output staging
<code>rundll32.exe %s,run %s</code>	NodeSnake PE / InterLock DLL execution

Assessment

The shared crypter framework is the operational link. The same build pipeline that produces NodeSnake PE backdoors also wraps the InterLock Windows ransomware. If a defender recovers a crypter shell with byte pattern `488b041f483345f04889041e` from a network, both backdoor and ransomware deployment should be assumed. The weak PRNG in the ransomware encryption (`rand()` XOR `clock()`) may allow key recovery for incident responders with approximate execution timestamps.

Kirk

I like the internet. Want to get in touch? ·