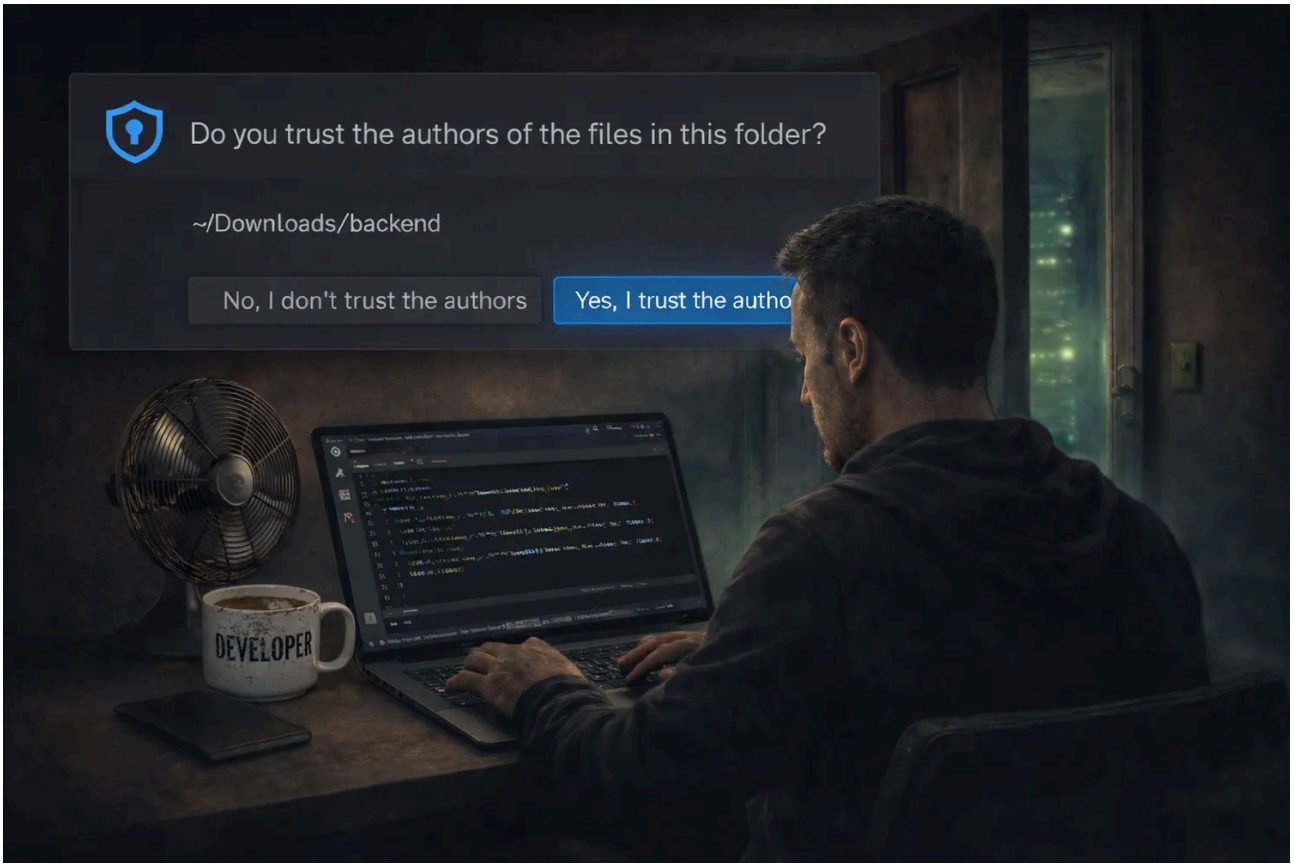
Threat Actors Expand Abuse of Microsoft Visual Studio Code

By Jamf Threat Labs

Archived: 2026-04-05 18:21:19 UTC

Jamf Threat Labs identifies additional abuse of Visual Studio Code. See the latest evolution in the Contagious Interview campaign.



By Thijs Xhaflaire

Introduction

At the end of last year, Jamf Threat Labs published research related to the Contagious Interview campaign, which has been attributed to a threat actor operating on behalf of North Korea (DPRK). Around the same time, researchers from OpenSourceMalware (OSM) released additional findings that highlighted an evolution in the techniques used during earlier stages of the campaign.

Specifically, these newer observations highlight an additional delivery technique alongside the previously documented ClickFix-based techniques. In these cases, the infection chain abuses Microsoft Visual Studio Code task configuration files, allowing malicious payloads to be executed on the victim system.

Following the discovery of this technique, both Jamf Threat Labs and OSM continued to closely monitor activity associated with the campaign. In December, Jamf Threat Labs identified additional abuse of Visual Studio Code `tasks.json` configuration files. This included the introduction of dictionary files containing heavily obfuscated JavaScript, which is executed when a victim opens a malicious repository in Visual Studio Code.

Jamf Threat Labs shared these findings with OSM, who subsequently published a more in-depth technical analysis of the obfuscated JavaScript and its execution flow.

Earlier this week, Jamf Threat Labs identified another evolution in the campaign, uncovering a previously undocumented infection method. This activity involved the deployment of a backdoor implant that provides remote code execution capabilities on the victim system.

At a high level, the chain of events for the malware look like so:

Throughout this blog post we will shed light on each of these steps.

Initial Infection

In this campaign, infection begins when a victim clones and opens a malicious Git repository, often under the pretext of a recruitment process or technical assignment. The repositories identified in this activity are hosted on either GitHub or GitLab and are opened using Visual Studio Code.

When the project is opened, Visual Studio Code prompts the user to trust the repository author. If that trust is granted, the application automatically processes the repository's `tasks.json` configuration file, which can result in embedded arbitrary commands being executed on the system.

On macOS systems, this results in the execution of a background shell command that uses `nohup bash -c` in combination with `curl -s` to retrieve a JavaScript payload remotely and pipe it directly into the Node.js runtime. This allows execution to continue independently if the Visual Studio Code process is terminated, while suppressing all command output.

In observed cases, the JavaScript payload is hosted on `vercel.app`, a platform that has been increasingly used in recent DPRK-related activity following a move away from other hosting services, as previously documented by OpenSourceMalware.

Jamf Threat Labs reported the identified malicious repository to GitHub, after which the repository was removed. While monitoring the activity prior to takedown, we observed the URL referenced within the repository change on multiple occasions. Notably, one of these changes occurred after the previously referenced payload hosting infrastructure was taken down by Vercel.

The JavaScript Payload

Once execution begins, the JavaScript payload implements the core backdoor logic observed in this activity. While the payload appears lengthy, a significant portion of the code consists of unused functions, redundant logic, and extraneous text that is never invoked during execution `(SHA256: 932a67816b10a34d05a2621836cdf7fbf0628bbfdf66ae605c5f23455de1e0bc)`. This additional code increases the size

and complexity of the script without impacting its observed behavior. It is passed to the node executable as one large argument.

Focusing on the functional components, the payload establishes a persistent execution loop that collects basic host information and communicates with a remote command-and-control (C2) server. Hard-coded identifiers are used to track individual infections and manage tasks from the server.

Core backdoor functionality

While the JavaScript payload contains a significant amount of unused code, the backdoor's core functionality is implemented through a small number of routines. These routines provide remote code execution, system fingerprinting, and persistent C2 communication.

Remote code execution capability

The payload includes a function that enables the execution of arbitrary JavaScript while the backdoor is active. At its core, this is the main functionality of this backdoor.

This function allows JavaScript code supplied as a string to be dynamically executed over the course of the backdoor lifecycle. By passing the `require` function into the execution context, attacker-supplied code can import additional Node.js modules allowing additional arbitrary node functions to be executed.

System fingerprinting and reconnaissance

To profile the infected system, the backdoor collects a small set of host-level identifiers:

This routine gathers the system hostname, MAC addresses from available network interfaces, and basic operating system details. These values provide a stable fingerprint that can be used to uniquely identify infected hosts and associate them with a specific campaign or operator session.

In addition to local host identifiers, the backdoor attempts to determine the victim’s public-facing IP address by querying the external service ipify.org, a technique that has also been observed in prior DPRK-linked campaigns.

Command-and-control beaconing and task execution

Persistent communication with the C2 server is implemented through a polling routine that periodically sends host information and processes server responses. The beaconing logic is handled by the following function:

This function periodically sends system fingerprinting data to a remote server and waits for a response. The beacon executes every five seconds, providing frequent interaction opportunities.

The server response indicates successful connectivity and allows the backdoor to maintain an active session while awaiting tasking.

If the server response contains a specific status value, the contents of the response message are passed directly to the remote code execution routine, mentioned prior.

Further Execution and Instructions

While monitoring a compromised system, Jamf Threat Labs observed further JavaScript instructions being executed roughly eight minutes after the initial infection. The retrieved JavaScript went on to set up a very similar payload to the same C2 infrustructure.

Review of this retrieved payload yields a few interesting details...

1. It beacons to the C2 server every 5 seconds, providing its system details and asks for further JavaScript instructions.
2. It executes that additional JavaScript within a child process.
3. It's capable of shutting itself and child processes down and cleaning up if asked to do so by the attacker.
4. It has inline comments and phrasing that appear to be consistent with AI-assisted code generation.

Conclusion

This activity highlights the continued evolution of DPRK-linked threat actors, who consistently adapt their tooling and delivery mechanisms to integrate with legitimate developer workflows. The abuse of Visual Studio Code task configuration files and Node.js execution demonstrates how these techniques continue to evolve alongside commonly used development tools.

Jamf Threat Labs will continue to track these developments as threat actors refine their tactics and explore new ways to deliver macOS malware. We strongly recommend that customers ensure Threat Prevention and Advanced Threat Controls are enabled and set to block mode in Jamf for Mac to remain protected against the techniques described in this research.

Developers should remain cautious when interacting with third-party repositories, especially those shared directly or originating from unfamiliar sources. Before marking a repository as trusted in Visual Studio Code, it’s important to review its contents. Similarly, "npm install" should only be run on projects that have been vetted, with particular attention paid to package.json files, install scripts, and task configuration files to help avoid unintentionally executing malicious code.

Indicators or Compromise

Dive into more Jamf Threat Labs research on our blog.
