OtterCookie: Analysis of Lazarus Group Malware Targeting Finance and Tech Professionals

any.run/cybersecurity-blog/ottercookie-malware-analysis/

June 3, 2025

OtterCookie: Analysis of Lazarus Group Malware Targeting Finance and Tech Professionals

Editor's note: ***The current article is authored by Mauro Eldritch, offensive security expert and threat intelligence analyst. You can find Mauro on X.***

What looks like a simple freelance bug fix turns out to be a full-blown malware infection. OtterCookie, a new tool from the Lazarus Group APT, hides behind clean code and fake job offers, then silently steals credentials, crypto wallets, and more.

In this step-by-step technical analysis, Mauro Eldritch breaks down the full attack chain, supported by live insights from ANY.RUN's Interactive Sandbox.

Overview of OtterCookie Malware

North Korean state-sponsored groups, most notably Lazarus, continue to target the financial and cryptocurrency sectors using a range of custom malware families. Previously observed campaigns included threats like InvisibleFerret and Beavertail, which were distributed through elaborate social engineering tactics such as fake developer interviews and staged business calls with executives.

A new addition to this toolkit is **OtterCookie**, a stealer malware that, much like its predecessors, isn't spread through random means like pirated software or infected USB drives. Instead, it is part of a broader, coordinated campaign targeting professionals in the tech, financial, and crypto industries. By staging fake interviews, threat actors deliver malware disguised either as coding challenges (or their dependencies) or video call software, in a campaign now known as Contagious Interview or DevPopper.

OtterCookie, written in heavily obfuscated JavaScript, was uncovered during a recent investigation conducted with the Bitso Quetzal Team. Notably, the delivery method used in this case stands out for its creativity and level of deception.

Picture 1: Obfuscated code. Lazarus loves Deobfuscator.io

Key Takeaways

OtterCookie is a new stealer malware linked to North Korean APT Lazarus, delivered through fake job offers.

Payload is fetched from an external API and executed using a require() call—no local implant needed.

Targets include browser credentials, macOS keychains, and crypto wallets like Solana and Exodus.

Data is exfiltrated via port 1224 to a U.S.-based C2 server, following patterns seen in Beavertail and InvisibleFerret.

ANY.RUN detects OtterCookie early, before deobfuscation, and maps its behavior in the ATT&CK Matrix.

OtterCookie eventually deploys InvisibleFerret, continuing Lazarus's modular, multi-stage approach.

Social Engineering Delivery: The "Job Offer" Trap

As part of the *Contagious Interview* campaign, one observed variation involved a new form of social engineering distributed through LinkedIn. Instead of requesting participation in a coding challenge or scheduling a business call, as seen in previous campaigns, the attacker proposed freelance contract work. The task was simple: resolve a minor visual bug in the frontend of a decentralized application (DApp).

The sender claimed their development team was unavailable due to vacation and shared access to a Bitbucket repository containing Node.js code.

Picture 2: Bitbucket repo

Surprisingly, the repository appeared entirely clean. No implants, no hidden payloads, and none of the suspicious NPM dependencies commonly associated with earlier malware like Beavertail. This wasn't an example of FUD (Fully Undetectable) malware bypassing antivirus detection, it was genuinely clean. The kind of clean that instills confidence and lowers suspicion.

Picture 3: VirusTotal

A Closer Look at OtterCookie Malware

The code simulates a NodeJS web service and frontend based on Express, with two interesting functions. First, there's an error section that looks hastily written, with a particularly odd error message.

Picture 4: Badly written Error

Next, there's a notable try/catch block in the code. For context, a try/catch block is a common programming construct that allows an application to attempt an operation. If the operation fails, due to either a specific error or a general exception, the catch block executes to handle the failure without crashing the application.

Picture 5: Try/Catch block

Execution Through Controlled Failure

This particular implementation is one of the most creative ways of deploying malware seen recently. The app's initialization sequence is wrapped in a try/catch block. When an error is triggered, it fetches a response from an external API that appears to provide contextual error information, and then… executes it.

You read it right – it uses a require() statement to execute whatever response comes back from the external API.

The first thought that comes to mind: *"Does that mean the system gets infected if the app fails?"*

And yes, that's exactly the point! The failure is intentional and triggered during the app's bootstrap phase. It kicks in, catches the error, prints it to the console, and pretends it just handled the issue gracefully—like everything's fine now and ready to go. In the background, it already fetched "the error" and is executing it.

Interactive Sandbox Analysis with ANY.RUN

Let's take a closer look at how this plays out in ANY.RUN's interactive sandbox

View analysis session

Picture 6: A forced failure

After launching an Ubuntu instance and installing Node.js, the next step involves adding the legacy peer dependencies from NPM—around 1,540 packages in total. Running the web server then triggers the expected error routine: *“Unexpected reserved word.”* Despite the wording, this error is anything but unexpected.

Originally, the task was to fix a simple visual bug. But that raises the question—how did a blatant, critical error like using a reserved word make it into the code? The answer becomes obvious a bit too late: while the app was running, it quietly queried a remote API in Finland—chainlink-api-v3[.]cloud—and received what appeared to be an error response.

Or at least something that looked like one. And it got executed.

Picture 7: The response, obfuscated in JavaScript

Deobfuscation and Payload Behavior

Let’s try to deobfuscate that response.

Lazarus is known for its frequent use of a legitimate online tool: **deobfuscate[.]io**. This platform has been used to obfuscate JavaScript payloads in fake NPM packages, and even entire malware families like Beavertail.

Picture 8: Decoded malware

When the obfuscated code is pasted, the webapp recognizes which version was used to scramble it and offers to redirect you straight to the right decoder. One click later, you get the original code, which is nice and readable. Let me introduce you to OtterCookie. Let’s analyze it.

Inside OtterCookie: What It Targets

OtterCookie begins by requesting libraries that allow interaction with the operating system, such as fs, os, path, request, and child_process. It also includes modules specifically designed to target major browsers like Brave, Google Chrome, Opera, and Mozilla Firefox, along with numerous browser extensions, primarily those related to cryptocurrency wallets and password managers.

Picture 9: Imported libraries and dedicated malicious modules

This behavior may sound familiar to those who've followed earlier DPRK-linked malware campaigns, such as Beavertail and InvisibleFerret.

Credential and Wallet Theft

In this case, OtterCookie specifically targets Firefox profile directories, copying the user's Solana-related profile data for exfiltration.

Picture 10: Firefox and Solana profiles are stolen

In addition to Solana, other wallets, such as Exodus, are also targeted, with sensitive files being copied for exfiltration. This aligns with the broader pattern observed in DPRK campaigns, where cryptocurrency assets are a primary focus due to their relative ease of laundering and anonymization.

And it’s not just about cryptocurrency. Some NFTs, despite having little market value, are used as authentication mechanisms in certain Web3 environments, which are increasingly widespread. These, too, can be valuable to threat actors.

Picture 11: Exodus Wallet is actively targeted

Next, OtterCookie attempts to access the macOS login keychain, along with credential databases from various browsers, extracting saved passwords, session tokens, and other sensitive authentication data.

Exfiltration Tactics and Infrastructure

Once everything is staged, the malware sends the loot to a webserver in the US (144.172.101.45), using port 1224 and the /uploads path.

We've seen this exact pattern before… in InvisibleFerret.

It's safe to assume that some practices—and even bits of code—are being recycled across these malware strains.

Picture 12: Remembrances of InvisibleFerret and BeaverTail

Before exfiltration, OtterCookie attempts to compress the collected data using tar. At this stage, some familiar filenames appear, **p.zi** and **p2.zip,** previously seen in related campaigns.

That definitely rings a bell. Similar filenames were seen in the Beavertail campaign, used to download and install its partner-in-crime and next stage: InvisibleFerret, pulled from an endpoint called /pdown. Just like in the snippet at the end of this script.

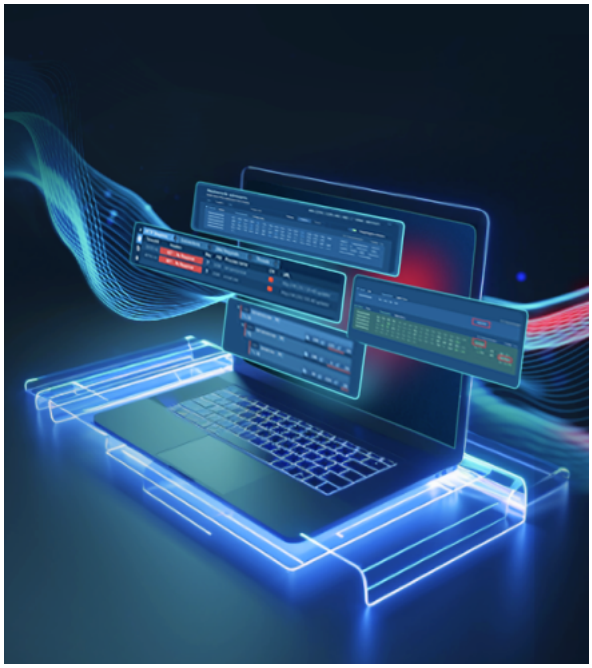
Picture 13: Downloading the next stage: InvisibleFerret

Next Stage: Delivering InvisibleFerret

At this stage, the malware attempts to download a portable Python distribution, compatible with either Windows or Unix, from its command-and-control (C2) server. Once installed, it proceeds to execute InvisibleFerret as the next stage of the attack. For context, InvisibleFerret is a cross-platform remote access trojan (RAT) written in Python, known for leveraging legitimate tools such as AnyDesk to maintain persistent access to the victim's system.

Picture 14: Preparing the next stage by setting up Python

The good news is that ANY.RUN successfully detects all three malware strains—OtterCookie, InvisibleFerret, and Beavertail.



Learn to analyze cyber threats

Follow along a detailed guide to using ANY.RUN's Interactive Sandbox for malware and phishing analysis

Read full guide

In this case, the obfuscated payload was flagged even before manual deobfuscation could begin.

With that covered, it's time to move on to the MITRE ATT&CK Matrix, which ANY.RUN conveniently generates as part of the analysis.

Picture 15: Detected as OTTERCOOKIE

The OtterCookie Matrix

OtterCookie shares several Tactics, Techniques, and Procedures (TTPs) with its counterparts, InvisibleFerret and Beavertail. Some of the most notable include:

T1082 – System Information Discovery
OtterCookie collects detailed information from the victim's system to build a comprehensive host profile.

T1003 – OS Credential Dumping
The malware accesses sensitive local files such as /etc/passwd and /etc/shadow, along with browser credential stores and OS keychains. The harvested data is then compressed and prepared for exfiltration.

T1071 – Application Layer Protocol
This technique is used to communicate with the command-and-control server (144.172.101.45) for data exfiltration.

T1571 – Non-Standard Port
Supporting T1071, this technique involves the use of an uncommon port—1224—to evade standard detection mechanisms.

Picture 16: MITRE ATT&CK Matrix

Conclusion

OtterCookie is yet another reminder of how advanced and deceptive modern malware has become. Hidden behind a routine bug fix task, it exfiltrates credentials, crypto wallet data, and system information, while quietly setting up a second-stage payload like InvisibleFerret.

Attacks like this demand more than traditional detection. They require a dynamic, transparent environment to truly understand what's happening.

With **ANY.RUN's interactive sandbox**, security teams can:

Cut investigation time from hours to seconds by getting clear verdicts in under 40 seconds even for obfuscated, evasive malware.

Understand threats in real time, helping analysts take action before damage is done.

Train junior analysts faster by giving them a safe, hands-on environment to explore real malware behavior without risking the network.

Improve response quality and speed, thanks to visualized tactics, techniques, and clear IOCs that can be used immediately in detection rules.

Boost team efficiency with easy-to-share sessions and collaborative analysis tools, reducing back-and-forth and enabling faster decision-making.

Whether you're investigating OtterCookie or preparing for what's next, **ANY.RUN helps** you detect, understand, and respond faster with clarity and control.

Register now with a business email to try ANY.RUN →

Gathered IOCs

IPv4: 135.181.123.177

IPv4: 144.172.101.45

Domain: chainlink-api-v3.cloud

URL: http://144.172.101.45:1224/

URL: http[:]//chainlink-api-v3[.]cloud/api/service/token/56e15ef3b5e5f169fc063f8d3e88288e

URL:http[:]//chainlink-api-v3[.]cloud/api/

URL: https[:]//bitbucket.org/0xhpenvynb/mvp_gamba/downloads/

SHA256: aa0d64c39680027d56a32ffd4ceb7870b05bdd497a3a7c902f23639cb3b43ba1

SHA256: 071aff6941dc388516d8ca0215b757f9bee7584dea6c27c4c6993da192df1ab9

SHA256: 486f305bdd09a3ef6636e92c6a9e01689b8fa977ed7ffb898453c43d47b5386d

SHA256: ec234419fc512baded05f7b29fefbf12f898a505f62c43d3481aed90fef33687

FileName: 0xhpenvynb-mvp_gamba-6b10f2e9dd85.zip

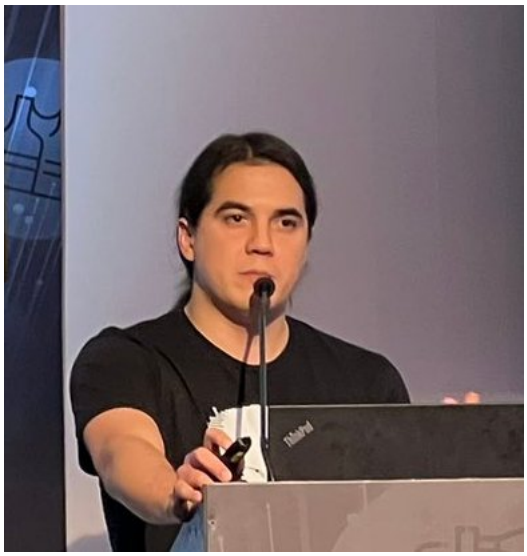
SOLWallet: V2grJiwjs25iJYqumbHyKo5MTK7SFqZSdmoRaj8QWb9

Mauro Eldritch

+ posts

Mauro Eldritch is an Argentinian-Uruguayan hacker, founder of BCA LTD and DC5411 (Argentina / Uruguay). He has spoken at various events, including DEF CON (12 times). He is passionate about Threat Intelligence and Biohacking. He currently leads Bitso's Quetzal Team, the first in Latin America dedicated to Web3 Threat Research.

Follow Mauro on:
X
LinkedIn
GitHub


Mauro Eldritch

Mauro Eldritch is an Argentinian-Uruguayan hacker, founder of BCA LTD and DC5411 (Argentina / Uruguay). He has spoken at various events, including DEF CON (12 times). He is passionate about Threat Intelligence and Biohacking. He currently leads Bitso's Quetzal Team, the first in Latin America dedicated to Web3 Threat Research.

Follow Mauro on:
X
LinkedIn
GitHub
View all posts Twitter
What do you think about this post?

7 answers

- Awful
- Average
- Great

No votes so far! Be the first to rate this post.

0 comments