How Scanning Your Projects for Security Issues Can Lead to Remote Code Execution

Application Security

Ron Masas
Sep 29, 2022 3 min read

The Imperva Red Team recently discovered and disclosed CVE-2022-40764, a command injection vulnerability affecting Snyk CLI. Snyk is a security company best known for its dependency vulnerability management software.

The disclosed command injection vulnerability affects the Snyk command-line interface tool that helps developers find and fix vulnerabilities. Exploitation of CVE-2022-40764 allows attackers to execute arbitrary commands on the host system where the Snyk CLI is installed. This can be exploited to gain access to sensitive data, modify files, or install malicious software.

We'd like to note Snyk's responsible response and cooperation on this matter. Snyk released a patch for the vulnerability and recommends that users update to the latest version of Snyk CLI.

In this post, we explain the method we used to discover the vulnerability and different exploit scenarios.

Monkey Patching

Traditionally, monkey patching is used to alter the behavior of a program at runtime. This technique can be used to add or modify a program functionality without changing the program code.

We used monkey patching to log information about function calls, parameters, and return values in order to locate potential vulnerabilities.

You can view the monkey patching script we used in our Github repository: monkeyp.

Finding the Vulnerability

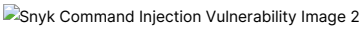
When using dynamic analysis techniques to hunt for vulnerabilities, supplying the right input is crucial. We started by broadly defining the input range of our target. In Snyk's case, it was a variety of programming languages and package managers. These projects were easily gathered using the Github API.

We monkey patched the Snyk CLI so it logged sensitive NodeJS operations such as file system access and shell/program execution. We then wrote a script that used our patched Snyk CLI to scan each of the projects we collected.

After a few minutes, we had all the execution logs. We started looking into them when we noticed that the "child_process.spawn" method was called with what seems to be a "user-supplied" input and the shell option set to true. The "shell" option is a boolean option that, when set to true, runs commands inside of a shell (i.e. uses '/bin/sh' on Unix, and process.env.ComSpec on Windows). Thus, if we could control any part of the input, we would be able to execute arbitrary shell commands.

After further investigation, we landed on the "vendor.json" file, which is used to specify the packages used by go projects. The "ignore" property value seemed to be the one used in the spawn

However, it did not work. The Snyk CLI output the following error:



Judging by the first line, we saw that some of our input was interpreted as a shell command. Looking at monkeyp logs, we could clearly see the issue.

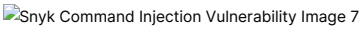
The Snyk CLI modifies our “ignore” value. Below, you can see the relevant code from the “snyk-go-parser” package which is responsible for these modifications.

Putting this all together, it means we either execute a file or we need to find a way to execute shell commands without using spaces.

When our command was interpreted by bash or zsh, we could use IFS (Input Field Separator), which is a common WAF bypass technique. IFS is a special shell variable, where each character is treated like a delimiter. If IFS is unset, the default value will be , which was ideal for our objective.

Finally, we had to include at least one forward slash to ensure our input was added to the “ignorePkgs“ array.

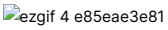
You can see the final payload below; it simply opens the calculator app on macOS.



Exploitation

To successfully exploit this vulnerability, a target would have to execute the “snyk test” command on untrusted files. While this is bad, practical exploitation is hard. We decided to go back and look for libraries that use Snyk CLI as a dependency to see if there would be a way to make exploitation more practical.

After going over a few libraries, we found that Snyk built a VSCode extension that uses the CLI to automatically scan every project opened with VSCode. This is already an improvement as the user would only need to view untrusted files in the VSCode editor to trigger our exploit, which is more likely in our view.



Closing Thoughts

The Imperva Red Team mission is to make attackers’ jobs harder by finding and responsibly disclosing vulnerabilities in popular services. It was a privilege to contribute to protecting the privacy of Snyk’s user community, as we continuously do for our own Imperva customers.

We hope you found this post informative and we look forward to sharing more technical walkthroughs in the future.

Trending

Application Security

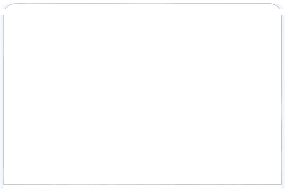
The Worrying Rise of Cybercrime as a Service (CaaS)

Nik Hewitt
Nov 7, 2022 5 min read

Application Security

From Online Fraud to DDoS and API Abuse: The State of Security Within eCommerce...

Erez Hasson
Nov 3, 2022 3 min read



Industry Perspective

13 Cybersecurity Horror Stories to Give you Sleepless Nights

Nik Hewitt
Oct 27, 2022 8 min read

Application Security

Imperva Stops Hordes of Bots from Hijacking Financial Accounts in Largest...

Erez Hasson
Oct 27, 2022 4 min read

Application Security

Out with the WAF, in wi the WAAP

Kelsey Winiarski
Oct 18, 2022 7 min

+1 866 926 4678

Partners

- Imperva Partner Ecosystem
- Channel Partners
- Technology Alliances
- Find a Partner
- Partner Portal Login

Resources

- Imperva Blog
- Resource Library
- Case Studies
- Learning Center

About Us

- Who We Are
- Events
- Careers
- Press & Awards
- Contact Information

Network

- Network Map
- System Status

Support

- Emergency DDoS Protection
- Support Portal
- Imperva Community
- Documentation Portal
- API Integration
- Trust Center

English

Cookies Settings

Trust Center