A9-Using Components with Known Vulnerabilities

Exploitability: AVERAGE | Prevalence: WIDESPREAD | Detectability: DIFFICULT | Technical Impact: MODERATE

Description

Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts.

Using insecure npm packages can lead to this vulnerability. Some projects today help test and alert on insecure dependencies:

1. The Node Security project (https://nodesecurity.io/advisories) is a great initiative and resource to know about related vulnerabilities.
2. Snyk.io (https://snyk.io/) is another Node.js CLI tool and Platform to scan and detect vulnerable packages
3. npm-check (https://www.npmjs.com/package/npm-check) Check for outdated, incorrect, and unused dependencies.
4. David DM (https://david-dm.org/) gets you an overview of your project dependencies, the version you use and the latest available, so you can quickly see what's drifting
5. requireSafe (https://www.npmjs.com/package/requiresafe) helps you keep your node applications secure
6. bithound.io (http://www.bithound.io) is a Node.js code analysis service.
7. npm outdated as well as 'yarn outdated' are both command line ways to show possibly out of date dependencies

Attack Mechanics

The npm packages are essential part of our node application. These packages could either accidentally or maliciously contain insecure code. Through insecure packages an attacker can:

- Create and run scripts at different stages during installation or usage of the package.
- Read, write, update, delete files on system
- Write and execute binary files
- Collect sensitive data send it remotely

How Do I Prevent It?

These are few measures we can take to protect against malicious npm packages

- Do not run application with root privileges
- Prefer packages that include static code analysis. Check JSHint/JSLint the configuration to know what rules code abide by
- Prefer packages that contain comprehensive unit tests and review tests for the functions our application uses
- Review code for any unexpected file or database access
- Research about how popular the package is, what other packages use it, if any other packages are written by the author, etc
- Lock version of packages used
- Watch Github repositories for notifications. This will inform us if any vulnerabilities are discovered in the package in future

Insecure Dependencies Example

Description

The demo web application is using a popular library called Marked (https://github.com/chjj/marked) which is a Markdown parser in JavaScript and provides an easy way to integrate markdown syntax for rich text to a website, replacing the need to build WYSIWYG editors.

This library has reached almost millions of downloads a month, making it quite popular with also **11,000** stars on GitHub at one point.

Attack Mechanics

In this demo project we are using an insecure version of the Marked library that is vulnerable to XSS exploits.

Scenario: A form on a page allows free text user input which is later parsed using the Marked library to markdown format and compiled in a dedicated view to show the rich text version. An attacker can exploit this form to insert malicious XSS strings which the Markdown library isn't filtering very well, resulting in an XSS attack.

Try sending one of the following markdown syntax strings in the Memos section to exploit it and see which one succeeds:

1.

```
[Nice try](javascript:alert(1))
```

2.

```
[Hi there](javascript&#58;alert(1&#41;)
```

3.

```
[I'm here!](javascript&#58this;alert(1&#41;)
```