snyk Vulnerability DB

Regular Expression Denial of Service (ReDoS)

Affecting org.webjars.npm:angular package,
versions (1.7.0,]

5.3
MEDIUM

INTRODUCED: 21 APR 2022 CVE-2022-25844 ❓
CWE-1333 ❓

Share ⌄

Snyk CVSS

Exploit Maturity **Mature** ❓

Attack Complexity **Low** ❓

See more

How to fix?

There is no fixed version for `org.webjars.npm:angular`.

> **NVD**   7.5 HIGH

> **Red Hat**   7.5 HIGH

Overview

org.webjars.npm:angular is a WebJar for angular.

Affected versions of this package are vulnerable to Regular Expression Denial of Service (ReDoS) by providing a custom locale rule that makes it possible to assign the parameter in `posPre: ' '.repeat()` of `NUMBER_FORMATS.PATTERNS[1].posPre` with a very high value.

Note:

1. This package has been deprecated and is no longer maintained.

2. The vulnerable versions are 1.7.0 and higher.

PoC:

```
class AppCtrl { constructor($locale, $timeout) {
  'ngInject'; const ctrl = this;
```

Do your applications use this vulnerable package?

In a few clicks we can analyze your entire

application and see what components are vulnerable in your application, and

```
ctrl.currencySymbol = '$'; ctrl.amount = 100;
ctrl.posPre =
$locale.NUMBER_FORMATS.PATTERNS[1].posPre;
ctrl.onPosPreChange = () => {
$locale.NUMBER_FORMATS.PATTERNS[1].posPre =
ctrl.posPre; const amount = ctrl.amount;
ctrl.amount = 0; $timeout(() => (ctrl.amount =
amount)); }; ctrl.onReDos = () => {
ctrl.currencySymbol = ''; ctrl.posPre = '
'.repeat(1000000);
$locale.NUMBER_FORMATS.PATTERNS[1].posPre =
ctrl.posPre; }; } } export default AppCtrl;
```

Report a new vulnerability

Found a mistake?

Details

Denial of Service (DoS) describes a family of attacks, all aimed at making a system inaccessible to its original and legitimate users. There are many types of DoS attacks, ranging from trying to clog the network pipes to the system by generating a large volume of traffic from many machines (a Distributed Denial of Service - DDoS - attack) to sending crafted requests that cause a system to crash or take a disproportional amount of time to process.

The Regular expression Denial of Service (ReDoS) is a type of Denial of Service attack. Regular expressions are incredibly powerful, but they aren't very intuitive and can ultimately end up making it easy for attackers to take your site down.

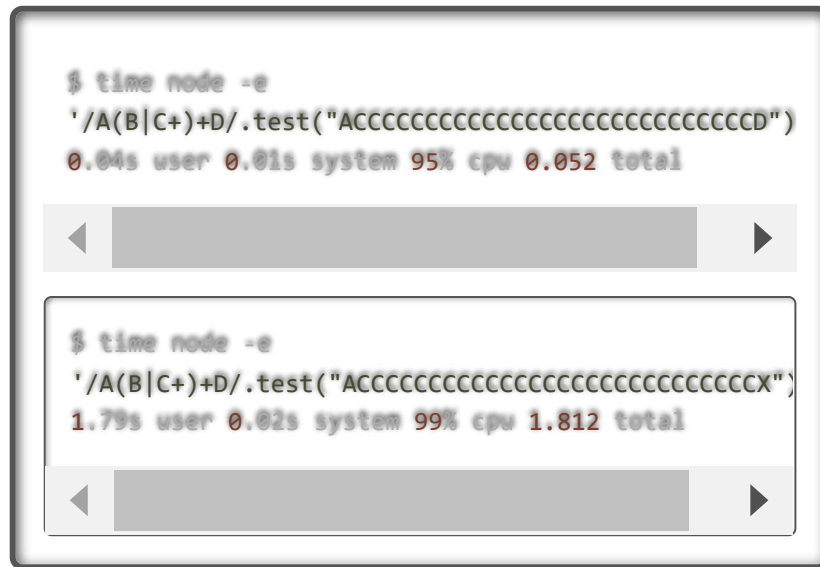Let's take the following regular expression as an example:

```
regex = /A(B|C+)+D/
```

This regular expression accomplishes the following:

- A The string must start with the letter 'A'

- (B|C+)+ The string must then follow the letter A with either the letter 'B' or some number of occurrences of the letter 'C' (the + matches one or more times). The + at the end of this section states that we can look for one or more matches of this section.

- D Finally, we ensure this section of the string ends with a 'D'

The expression would match inputs such as ABBD , ABCCCCD , ABCBCCCD and ACCCCCD

It most cases, it doesn't take very long for a regex engine to find a match:

```
$ time node -e
'/A(B|C+)+D/.test("ACCCCCCCCCCCCCCCCCCCCCCCCCCCCCD")
0.04s user 0.01s system 95% cpu 0.052 total
```

```
$ time node -e
'/A(B|C+)+D/.test("ACCCCCCCCCCCCCCCCCCCCCCCCCCCCCX")
1.79s user 0.02s system 99% cpu 1.812 total
```

The entire process of testing it against a 30 characters long string takes around ~52ms. But when given an invalid string, it takes nearly two seconds to complete the test, over ten times as long as it took to test a valid string. The dramatic difference is due to the way regular expressions get evaluated.

Most Regex engines will work very similarly (with minor differences). The engine will match the first possible way to accept the current character and proceed to the next one. If it then fails to match the next one, it will backtrack and see if there was another way to digest the previous character. If it goes too far down the rabbit hole only to find out the string doesn't match in the end, and if many characters have multiple valid regex paths, the number of backtracking steps can become very large, resulting in what is known as *catastrophic backtracking*.

Let's look at how our expression runs into this problem, using a shorter string: "ACCCX". While it seems fairly straightforward, there are still four different ways that the engine could match those three C's:

1. CCC

2. CC+C

3. C+CC

4. C+C+C.

The engine has to try each of those combinations to see if any of them potentially match against the expression. When you combine that with the other steps the engine must take, we can use RegEx 101 debugger to see the engine has to take a total of 38 steps before it can determine the string doesn't match.

From there, the number of steps the engine must use to validate a
string just continues to grow.

String	Number of C's	Number of steps
ACCCX	3	38
ACCCCX	4	71
ACCCCCX	5	136
ACCCCCCCCCCCCCX	14	65,553

By the time the string includes 14 C's, the engine has to take over
65,000 steps just to see if the string is valid. These extreme situations
can cause them to work very slowly (exponentially related to input size,
as shown above), allowing an attacker to exploit this and can cause the
service to excessively consume CPU, resulting in a Denial of Service.

References

- PoC

PRODUCT

Snyk Open Source

Snyk Code

Snyk Container

Snyk Infrastructure as Code

Test with Github

Test with CLI

RESOURCES

Vulnerability DB

Documentation

Disclosed Vulnerabilities

Blog

FAQs

COMPANY

CONTACT US

Support

Report a new vuln

Press Kit

Events

FIND US ONLINE

TRACK OUR DEVELOPMENT

DevSecCon   Join the >>
                  community