July 21, 2021

GHSL-2020-310: ReDoS (Regular Expression Denial of Service) in Rocket Chat - CVE-2021-32832

GitHub Security Lab

Coordinated Disclosure Timeline

- 2020-11-29: Report sent to security@rocket.chat
- 2020-11-30: Issue is acknowledged
- 2020-07-01: GHSL requests status update
- 2020-08-01: Maintainers confirm that they have verified the vulnerabilities and are actively working on a fix
- 2021-03-26: Fix is published

Summary

The project contains one or more regular expressions that are vulnerable to ReDoS (Regular Expression Denial of Service)

Product

Rocket.Chat

Tested Version

Latest commit at the time of reporting (November 30, 2020).

Details

ReDoS

ReDoS, or Regular Expression Denial of Service, is a vulnerability affecting poorly constructed and potentially inefficient regular expressions which can make them perform extremely badly given a creatively constructed input string.

For the specific regular expression reported, it is possible to force it to work with an `O(2^n)` runtime performance when there is exponential backtracking.

ReDoS can be caused by ambiguity or overlapping between some regex clauses. These badly performing regular expressions can become a security issue if a user can control the input. For example if the project is an input validation library, then the project could be used by a server to validate untrusted user input. There is no one size fits all when it comes to fixing ReDoS. But in general it is about removing ambiguity/overlap inside the regular expression.

Before showing the vulnerable regex, it may be helpful to show some examples of regular expressions vulnerable to ReDoS and how to fix them. If you are familiar with this vulnerability and how to fix it, please skip this section.

```
var reg = /<!--(.|\s)*?-->/g;
```

The above regular expression matches the start of an HTML comment, followed by any characters, followed by the end of a HTML comment. The dot in the regular expression (`.`) matches any char except newlines, and `\s` matches any whitespace. Both `.` and `\s` matches whitespace such as the space character. There are therefore many possible ways for this regular expression to match a sequence of spaces. This becomes a problem if the input is a string that starts with `<!--` followed by a long sequence of spaces, because the regular expression evaluator will try every possible way of matching the spaces (see this debugging session for an example: https://regex101.com/r/XvYgkN/1/debugger).

The fix is to remove the ambiguity, which can be done by changing the regular expression to the below, where there is no overlap between the different elements of the regular expression.

```
var reg = /<!--(.|\r|\n)*?-->/g;
```

```
var reg = /(\w+_?)+_(\d+)/;
```

The above matches a snake_case identifier that ends with some digits. However, for a string starting with lots of letters, there many ways for the regular expression to match those letters, due to the regex having a repetition matching letters (`w+`) inside another repetition that can match letters (`(\w+_?)+`). The regular expression evaluator will try every possible grouping of letters into smaller groups of letters (see this debugging session for an example: https://regex101.com/r/fmci1j/1/debugger). The fix is again to remove ambiguity, by changing the inner repetition to match groups of letters that must end with an underscore.

```
var reg = /(\w+_)+(\d+)/;
```

Often the regular expressions are not as simple as the examples above. Like the below regular expression that used to be part of VS Code. (the top regular expression is the vulnerable, the bottom is the fixed)

```
var linkPattern = /(\[((!\[[^\]]*?\]\(\s*)([^\s\(\)]+?)\s*\)\]|(?:\\\]|[^\]])*\])\(\s*)(([^\s\(\)]|\(\S*?\))+)\s*(".*?")?\)/g;
var linkPattern = /(\[((!\[[^\]]*?\]\(\s*)([^\s\(\)]+?)\s*\)\]|(?:\\\]|[^\]])*\])\(\s*)(([^\s\(\)]|\([^\s\(\)]*?\))+)\s*(".*?")?\)/g;
```

But this example is actually very similar to the first example. A section of the regular expression that was too general (`\S`) was changed to something a bit more specific (`[^\s\(\)]`) to remove overlap with another part of the regular expression.

Vulnerability

Please note that we haven't set up the application to test the feasibility of exploiting ReDoS on these regular expressions (a quick attempt failed). We could have set up a free cloud trial, but we didn't want to risk overloading your server infrastructure. Therefore, we have only tested these attacks on the isolated regular expressions as shown below.

This vulnerability was found using a CodeQL query which identified several regular expressions as vulnerable. Link to query.

We've found two regular expressions that can be potentially problematic.

The first one is on line 27 in api.js. And the command below showcases why this regular expression can be problematic. My machine uses about 20 seconds to evaluate the below.

```
time node -e "/\{ ?([^\} ]+)(( ([^\}]+))+)? ?\}/.test(\"{ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | \")"
```

The second one is on line 15 in CustomFieldsAdditionalForm.js.
And the command below showcases why this regular expression can be problematic.
My machine uses 15 seconds to evaluate the below.

```
time node -e "(/^([a-zA-Z0-9-_ ]+)(,\s*[a-zA-Z0-9-_ ]+)*$/i).test(\"a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a|\")"
```

Impact

This issue may lead to a denial of service.

CVE

- CVE-2021-32832

Credit

This issue was discovered and reported by GitHub team member @erik-krogh (Erik Krogh Kristensen).

Contact

You can contact the GHSL team at `securitylab@github.com`, please include a reference to `GHSL-2020-310` in any communication regarding this issue.