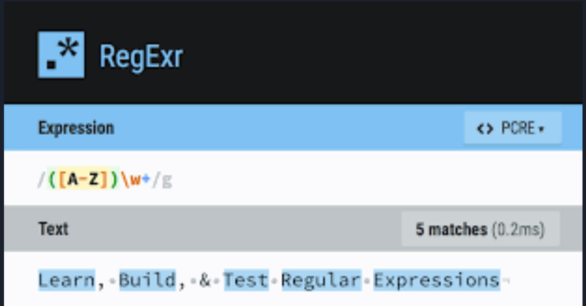
/(**Re**+|**DoS**)*/

For fun and stuck!

O que é uma expressão regular (regex)?

- Uma expressão regular (regex) é uma sequência de caracteres que especificam um padrão de busca.
- Surgiu a partir dos estudos do matemático Stephen Cole Kleene no ano de 1950.
- Definiu as bases para os primeiros algoritmos de busca.



```
/([A-Z])\w+/g
```

5 matches (0.2ms)

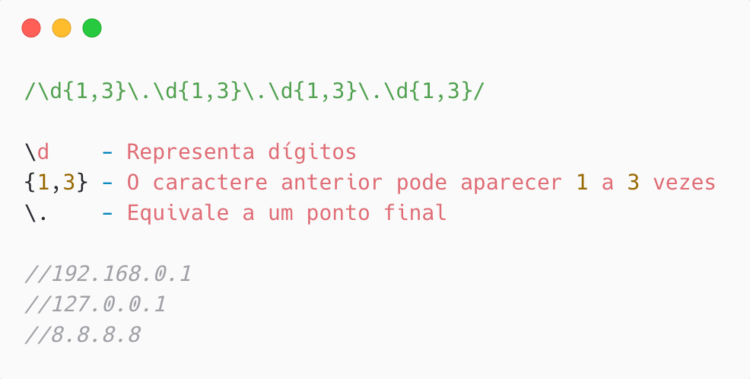
```
Learn, Build, & Test Regular Expressions
```

Pra que serve uma (expressão regular) regex?

- São utilizadas para busca, extração, substituição e validação de padrões de caracteres.
- Utilizada por softwares que realizam busca, SGDBs, IDEs, processadores de texto, WAFs, IDSs, motores de busca, parsers, utilitários de linha de comando e etc.
- MySQL, MongoDB, VS Code, grep, findstr e etc.

Pra que serve uma (expressão regular) regex?

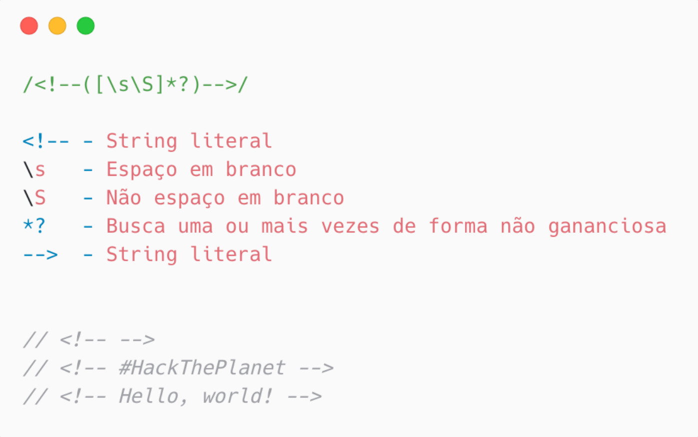
- Validação de IPv4



```
/\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}/

\d    - Representa dígitos
{1,3} - O caractere anterior pode aparecer 1 a 3 vezes
\.    - Equivale a um ponto final

//192.168.0.1
//127.0.0.1
//8.8.8.8
```

Pra que serve uma (expressão regular) regex?

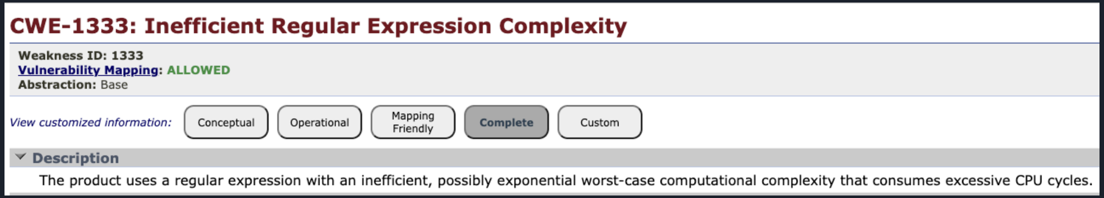
- Comentários HTML / XML



```
/<!--([\s\S]*?)-->/

<!-- - String literal
\s   - Espaço em branco
\S   - Não espaço em branco
*?   - Busca uma ou mais vezes de forma não gananciosa
-->  - String literal


// <!-- -->
// <!-- #HackThePlanet -->
// <!-- Hello, world! -->
```

O que é um ataque ReDoS?

- É uma abreviação de Regular Expression Denial of Service.
- Pode causar indisponibilidade de um sistema por lentidão ou por queda.
- É um ataque que ocorre quando um engine de expressão regular tentar validar uma entrada (texto) e consome muito recurso computacional para realizar a análise.



CWE-1333: Inefficient Regular Expression Complexity

Weakness ID: 1333
Vulnerability Mapping: ALLOWED
Abstraction: Base

Description

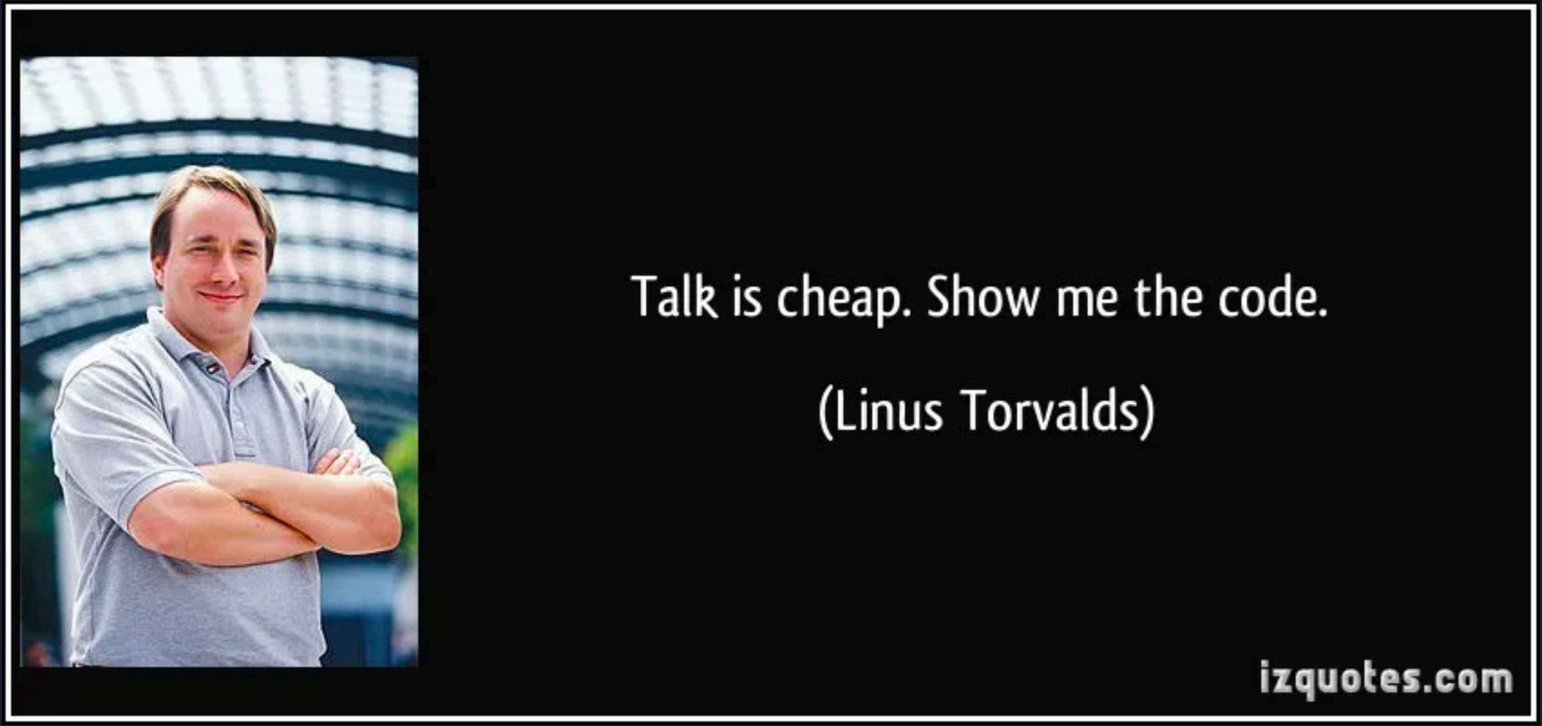
The product uses a regular expression with an inefficient, possibly exponential worst-case computational complexity that consumes excessive CPU cycles.

Porque um ReDoS ocorre?

- Alguns algoritmos de análise de expressão regular implementam backtracking*.
- Dependendo do tamanho do input, esse algoritmo com backtracking é sobrecarregado e consome muito processamento para analisar um regex.
- Expressões regulares ambíguas "triggam" esse cenário.
- Linguagens que possuem engine que implementam backtracking são vulneráveis. (C#, JavaScript, Java, Python, PHP, Ruby, etc.)

*Algoritmo de busca profunda (Depth-First Search)

Show me the code



Talk is cheap. Show me the code.

(Linus Torvalds)

izquotes.com

Incidentes e vulnerabilidades de ReDoS

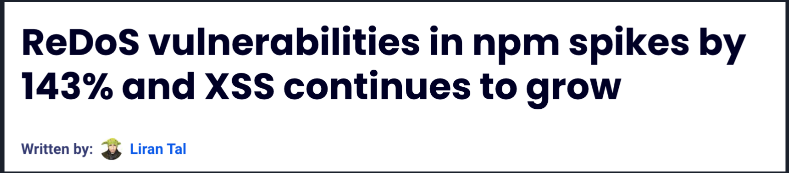
CloudFlare (02/07/19**)** - DoS de domínios devido a uma regex em uma regra de WAF.

StackOverflow (20/07/2016**)** - DoS devido a uma regex que removia espaços no início e ao final do texto.

validator.js (CVE-2021-3765**)** - Já teve várias vulnerabilidades de ReDoS e é utilizada por mais de 5600 bibliotecas.

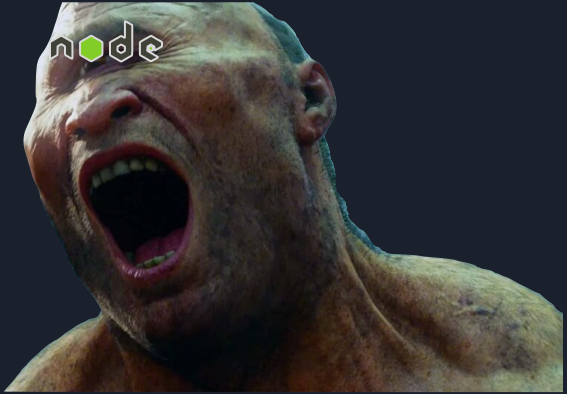
jsPDF (CVE-2021-23353) - Vulnerabilidade de ReDoS ao validar uma imagem.

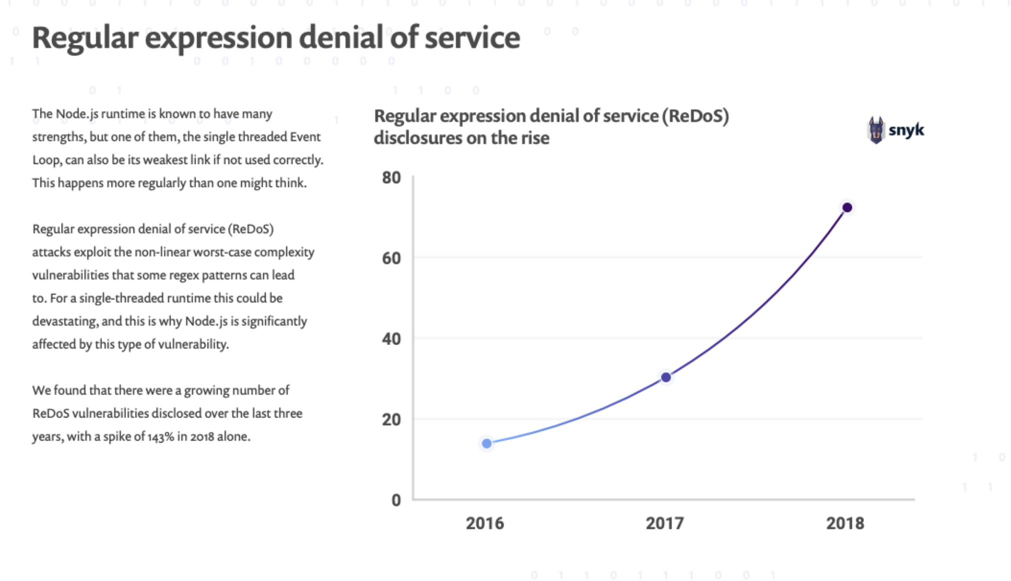
Falhas ReDoS não param de crescer



ReDoS vulnerabilities in npm spikes by 143% and XSS continues to grow

Written by: Liran Tal





Regular expression denial of service

The Node.js runtime is known to have many strengths, but one of them, the single threaded Event Loop, can also be its weakest link if not used correctly. This happens more regularly than one might think.

Regular expression denial of service (ReDoS) attacks exploit the non-linear worst-case complexity vulnerabilities that some regex patterns can lead to. For a single-threaded runtime this could be devastating, and this is why Node.js is significantly affected by this type of vulnerability.

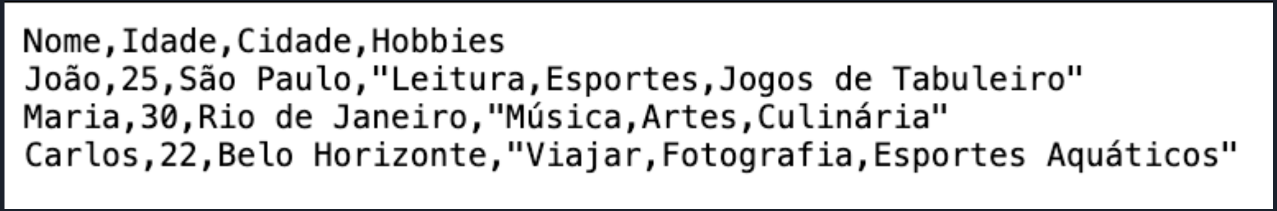
We found that there were a growing number of ReDoS vulnerabilities disclosed over the last three years, with a spike of 143% in 2018 alone.

Regular expression denial of service (ReDoS) disclosures on the rise
snyk
80
60
40
20
0
2016
2017
2018

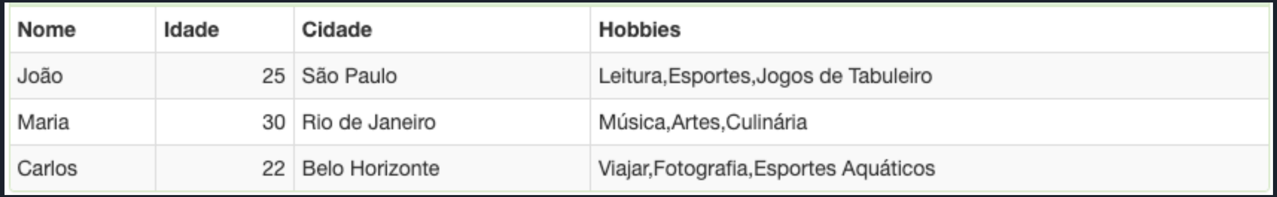
O formato CSV

- É um formato de texto que utiliza vírgulas para separar valores
- É um formato de arquivo level e simples para representar dados em forma de linhas e colunas
- Muito usado em softwares como Microsoft Excel e Libre Office

O formato CSV

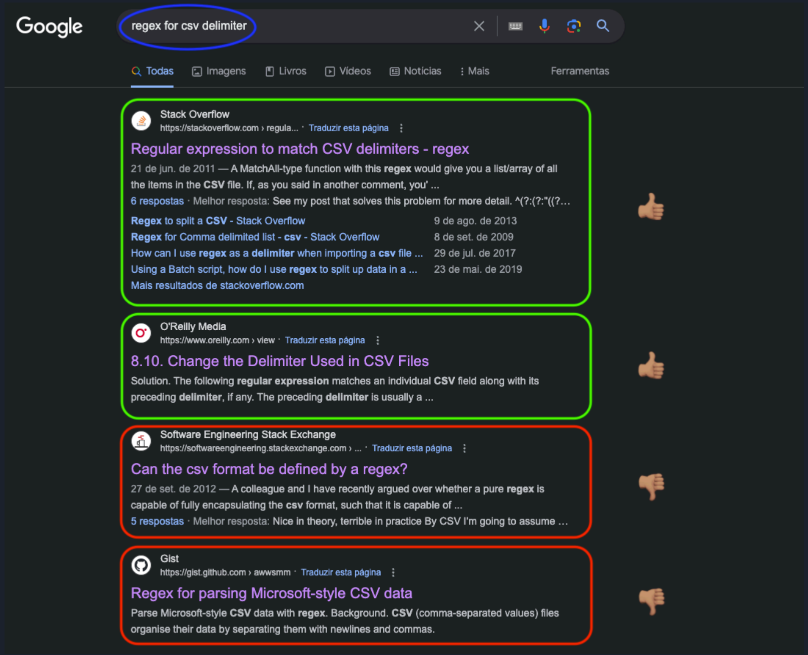


```
Nome,Idade,Cidade,Hobbies
João,25,São Paulo,"Leitura,Esportes,Jogos de Tabuleiro"
Maria,30,Rio de Janeiro,"Música,Artes,Culinária"
Carlos,22,Belo Horizonte,"Viajar,Fotografia,Esportes Aquáticos"
```



Nome	Idade	Cidade	Hobbies
João	25	São Paulo	Leitura,Esportes,Jogos de Tabuleiro
Maria	30	Rio de Janeiro	Música,Artes,Culinária
Carlos	22	Belo Horizonte	Viajar,Fotografia,Esportes Aquáticos

Google e sites estilo Stack Overflow



regex for csv delimiter

Stack Overflow
https://stackoverflow.com › regula... · Traduzir esta página

Regular expression to match CSV delimiters - regex

21 de jun. de 2011 — A MatchAll-type function with this **regex** would give you a list/array of all the items in the **CSV** file. If, as you said in another comment, you' ...

6 respostas · Melhor resposta: See my post that solves this problem for more detail. ^(?:(?:"((?...

Regex to split a **CSV** - Stack Overflow 9 de ago. de 2013
Regex for Comma delimited list - **csv** - Stack Overflow 8 de set. de 2009
How can I use **regex** as a **delimiter** when importing a **csv** file ... 29 de jul. de 2017
Using a Batch script, how do I use **regex** to split up data in a ... 23 de mai. de 2019
Mais resultados de stackoverflow.com

O'Reilly Media
https://www.oreilly.com › view · Traduzir esta página

8.10. Change the Delimiter Used in CSV Files

Solution. The following **regular expression** matches an individual **CSV** field along with its preceding **delimiter**, if any. The preceding **delimiter** is usually a ...

Software Engineering Stack Exchange
https://softwareengineering.stackexchange.com › ... · Traduzir esta página

Can the csv format be defined by a regex?

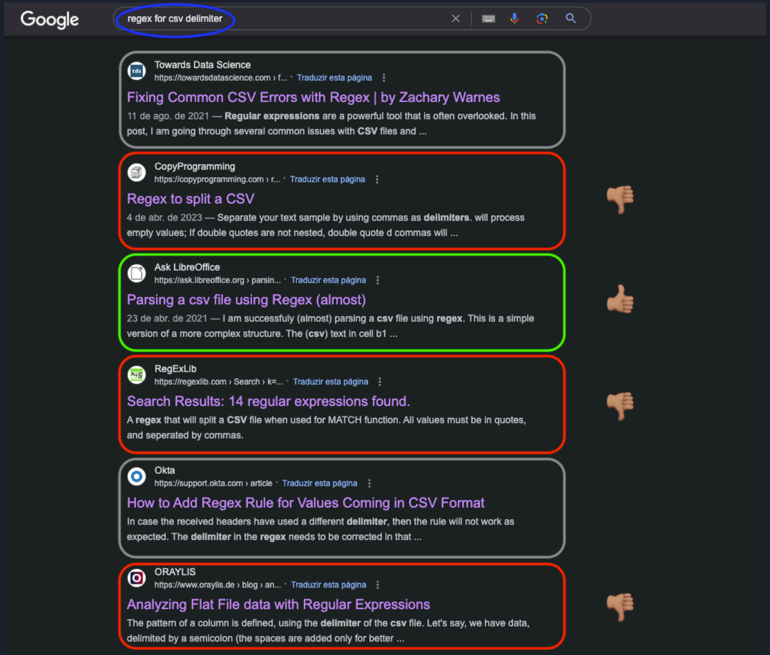
27 de set. de 2012 — A colleague and I have recently argued over whether a pure **regex** is capable of fully encapsulating the **csv** format, such that it is capable of ...

5 respostas · Melhor resposta: Nice in theory, terrible in practice By CSV I'm going to assume ...

Gist
https://gist.github.com › awwsmm · Traduzir esta página

Regex for parsing Microsoft-style CSV data

Parse Microsoft-style **CSV** data with **regex**. Background. **CSV** (comma-separated values) files organise their data by separating them with newlines and commas.



regex for csv delimiter

Towards Data Science
https://towardsdatascience.com › f... · Traduzir esta página

Fixing Common CSV Errors with Regex | by Zachary Warnes

11 de ago. de 2021 — **Regular expressions** are a powerful tool that is often overlooked. In this post, I am going through several common issues with **CSV** files and ...

CopyProgramming
https://copyprogramming.com › r... · Traduzir esta página

Regex to split a CSV

4 de abr. de 2023 — Separate your text sample by using commas as **delimiters**. will process empty values; If double quotes are not nested, double quote d commas will ...

Ask LibreOffice
https://ask.libreoffice.org › parsin... · Traduzir esta página

Parsing a csv file using Regex (almost)

23 de abr. de 2021 — I am successfuly (almost) parsing a **csv** file using **regex**. This is a simple version of a more complex structure. The (**csv**) text in cell b1 ...

RegExLib
https://regexlib.com › Search › k=... · Traduzir esta página

Search Results: 14 regular expressions found.

A **regex** that will split a **CSV** file when used for MATCH function. All values must be in quotes, and seperated by commas.

Okta
https://support.okta.com › article · Traduzir esta página

How to Add Regex Rule for Values Coming in CSV Format

In case the received headers have used a different **delimiter**, then the rule will not work as expected. The **delimiter** in the **regex** needs to be corrected in that ...

ORAYLIS
https://www.oraylis.de › blog › an... · Traduzir esta página

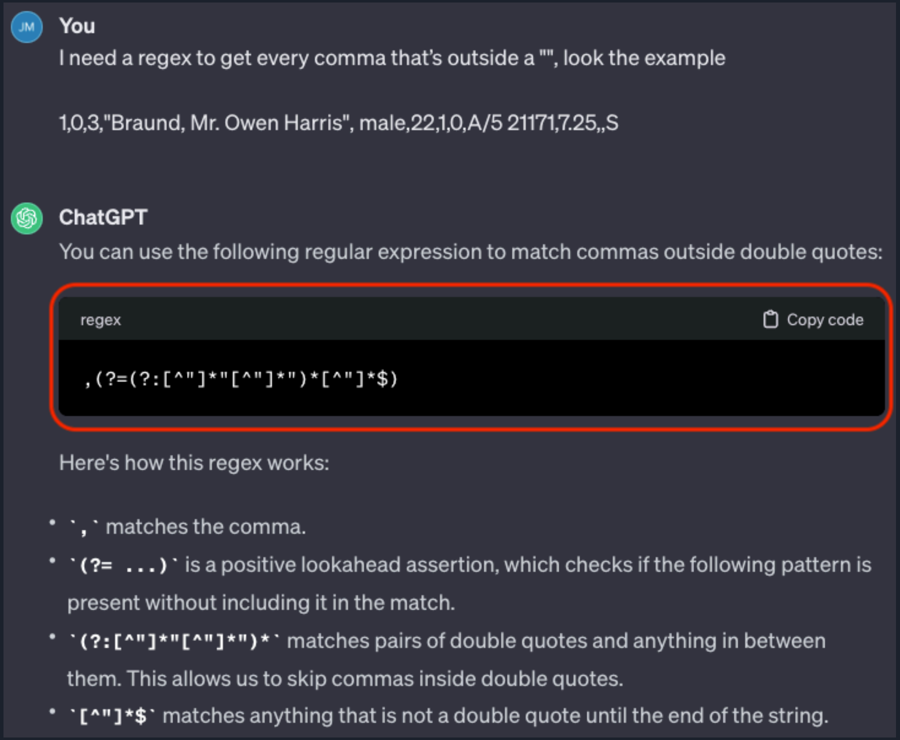
Analyzing Flat File data with Regular Expressions

The pattern of a column is defined, using the **delimiter** of the **csv** file. Let's say, we have data, delimited by a semicolon (the spaces are added only for better ...

Google e sites estilo Stack Overflow

Regex vulneráveis	Regex não vulneráveis
5	3

Chat GPT



You

I need a regex to get every comma that's outside a "", look the example

1,0,3,"Braund, Mr. Owen Harris", male,22,1,0,A/5 21171,7.25,,S

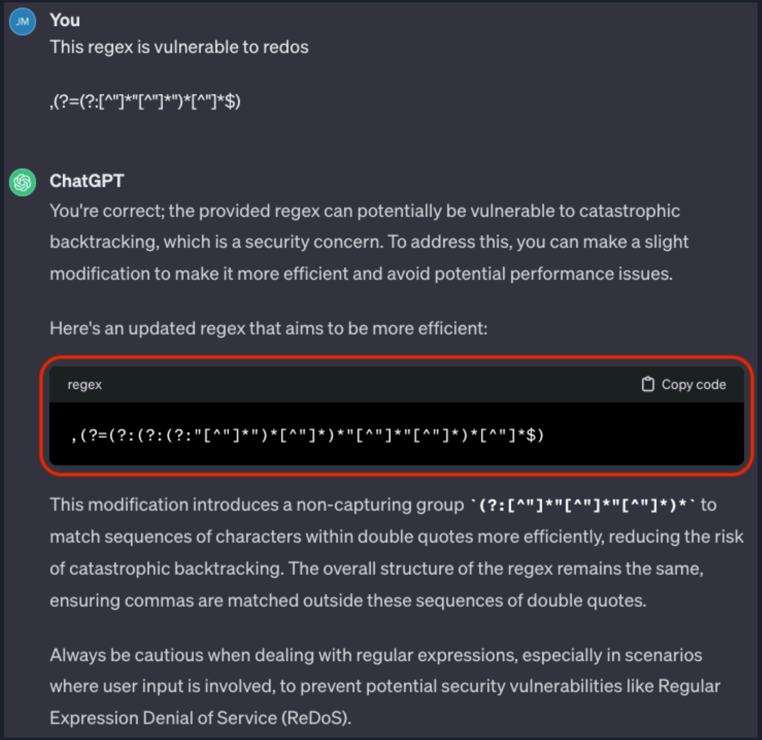
ChatGPT

You can use the following regular expression to match commas outside double quotes:

```regex
,(?=(?:[^"]*"[^"]*")*[^"]*$)
```

Here's how this regex works:

- `` `,` `` matches the comma.
- `` `(?= ...)` `` is a positive lookahead assertion, which checks if the following pattern is present without including it in the match.
- `` `(?:[^"]*"[^"]*")*` `` matches pairs of double quotes and anything in between them. This allows us to skip commas inside double quotes.
- `` `[^"]*$` `` matches anything that is not a double quote until the end of the string.



You

This regex is vulnerable to redos

,(?=(?:[^"]*"[^"]*")*[^"]*$)

ChatGPT

You're correct; the provided regex can potentially be vulnerable to catastrophic backtracking, which is a security concern. To address this, you can make a slight modification to make it more efficient and avoid potential performance issues.

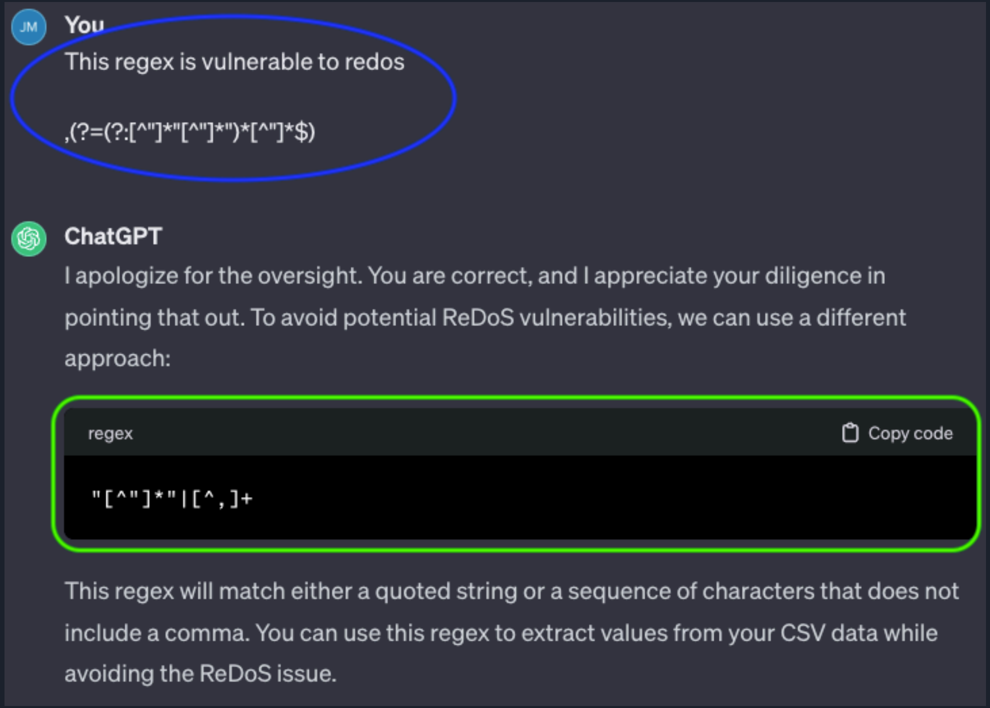
Here's an updated regex that aims to be more efficient:

```regex
,(?=(?:(?:(?:"[^"]*")*[^"]*)*"[^"]*"[^"]*)*[^"]*$)
```

This modification introduces a non-capturing group `` `(?:[^"]*"[^"]*"[^"]*)*` `` to match sequences of characters within double quotes more efficiently, reducing the risk of catastrophic backtracking. The overall structure of the regex remains the same, ensuring commas are matched outside these sequences of double quotes.

Always be cautious when dealing with regular expressions, especially in scenarios where user input is involved, to prevent potential security vulnerabilities like Regular Expression Denial of Service (ReDoS).

Chat GPT



You

This regex is vulnerable to redos

,(?=(?:[^"]*"[^"]*")*[^"]*$)

ChatGPT

I apologize for the oversight. You are correct, and I appreciate your diligence in pointing that out. To avoid potential ReDoS vulnerabilities, we can use a different approach:

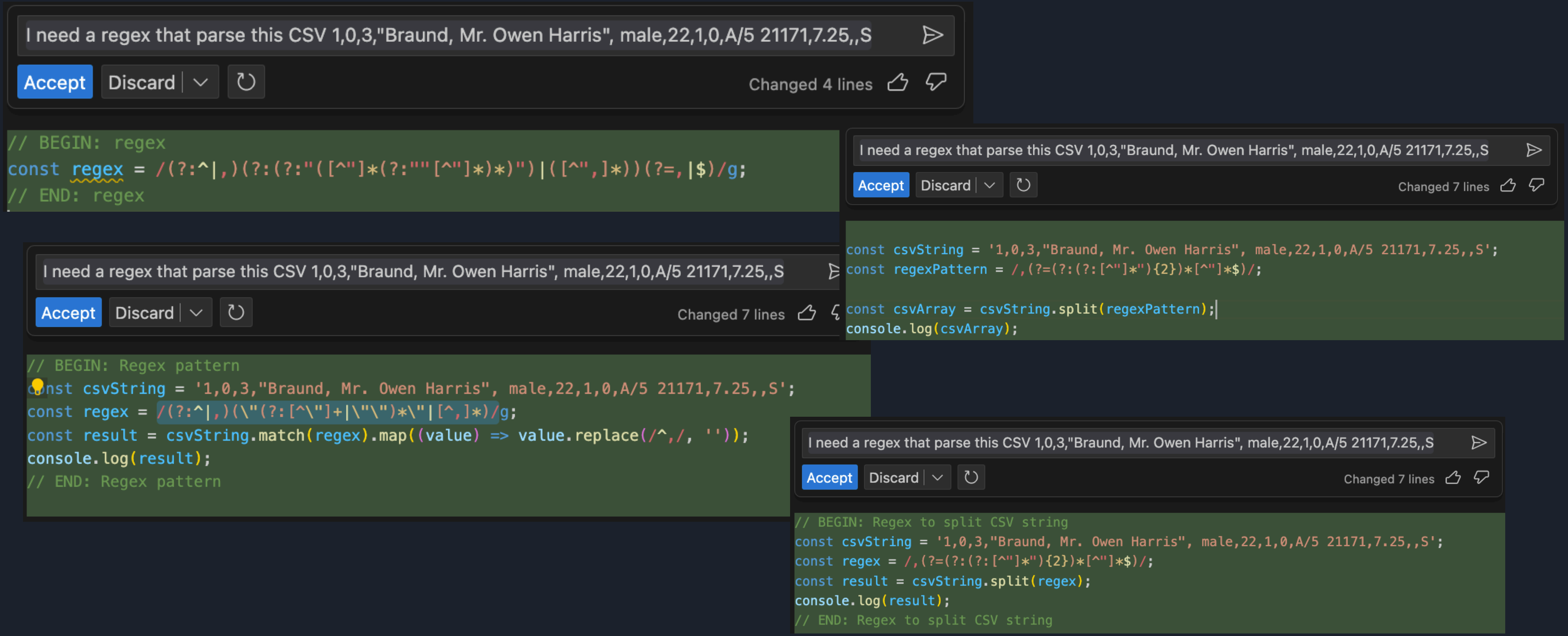
```regex
"[^"]*"|[^,]+
```

This regex will match either a quoted string or a sequence of characters that does not include a comma. You can use this regex to extract values from your CSV data while avoiding the ReDoS issue.

Chat GPT

Regex vulneráveis	Regex não vulneráveis
2	1

Github Copilot



I need a regex that parse this CSV 1,0,3,"Braund, Mr. Owen Harris", male,22,1,0,A/5 21171,7.25,,S

```
// BEGIN: regex
const regex = /(?:^|,)(?:(?:"([^"]*(?:""[^"]*)*)")|([^",]*))(?=,|$)/g;
// END: regex
```

I need a regex that parse this CSV 1,0,3,"Braund, Mr. Owen Harris", male,22,1,0,A/5 21171,7.25,,S

```
const csvString = '1,0,3,"Braund, Mr. Owen Harris", male,22,1,0,A/5 21171,7.25,,S';
const regexPattern = /,(?=(?:(?:[^"]*"){2})*[^"]*$)/;

const csvArray = csvString.split(regexPattern);
console.log(csvArray);
```

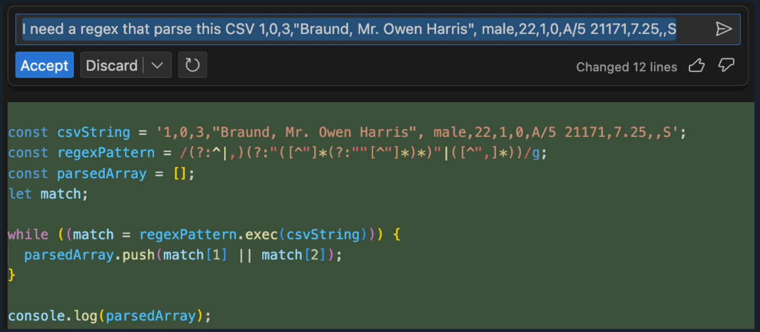
I need a regex that parse this CSV 1,0,3,"Braund, Mr. Owen Harris", male,22,1,0,A/5 21171,7.25,,S

```
// BEGIN: Regex pattern
const csvString = '1,0,3,"Braund, Mr. Owen Harris", male,22,1,0,A/5 21171,7.25,,S';
const regex = /(?:^|,)(\"(?:[^\"]+|\"\")*\"|[^,]*)/g;
const result = csvString.match(regex).map((value) => value.replace(/^,/, ''));
console.log(result);
// END: Regex pattern
```

I need a regex that parse this CSV 1,0,3,"Braund, Mr. Owen Harris", male,22,1,0,A/5 21171,7.25,,S

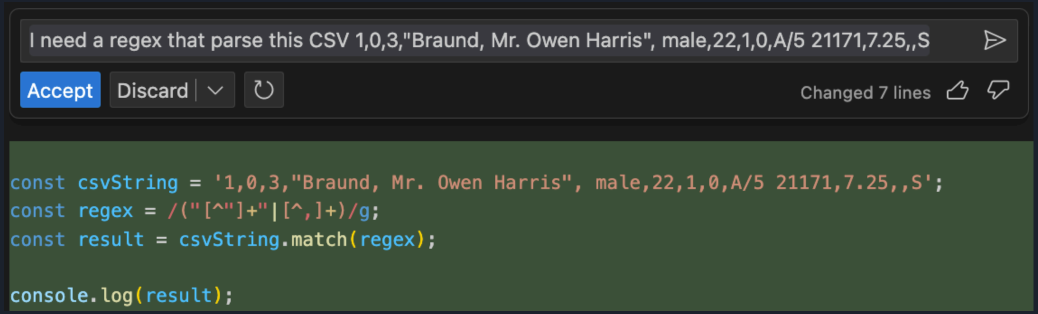
```
// BEGIN: Regex to split CSV string
const csvString = '1,0,3,"Braund, Mr. Owen Harris", male,22,1,0,A/5 21171,7.25,,S';
const regex = /,(?=(?:(?:[^"]*"){2})*[^"]*$)/;
const result = csvString.split(regex);
console.log(result);
// END: Regex to split CSV string
```

Github Copilot



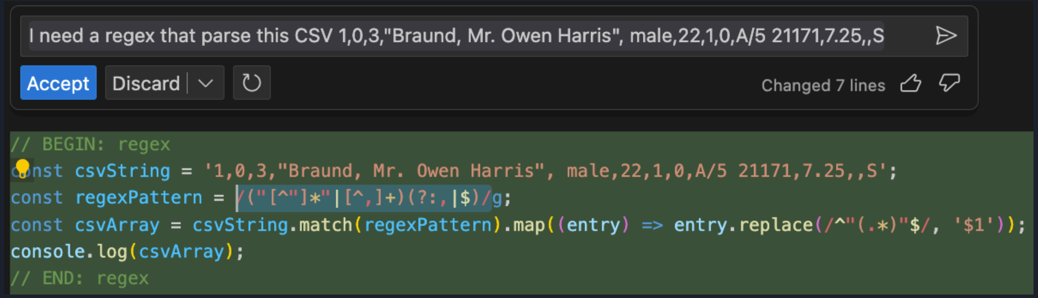
I need a regex that parse this CSV 1,0,3,"Braund, Mr. Owen Harris", male,22,1,0,A/5 21171,7.25,,S

```
const csvString = '1,0,3,"Braund, Mr. Owen Harris", male,22,1,0,A/5 21171,7.25,,S';
const regexPattern = /(?:^|,)(?:"([^"]*(?:""[^"]*)*)"|([^",]*))/g;
const parsedArray = [];
let match;

while ((match = regexPattern.exec(csvString))) {
  parsedArray.push(match[1] || match[2]);
}

console.log(parsedArray);
```



I need a regex that parse this CSV 1,0,3,"Braund, Mr. Owen Harris", male,22,1,0,A/5 21171,7.25,,S

```
const csvString = '1,0,3,"Braund, Mr. Owen Harris", male,22,1,0,A/5 21171,7.25,,S';
const regex = /("[^"]+"|[^,]+)/g;
const result = csvString.match(regex);

console.log(result);
```



I need a regex that parse this CSV 1,0,3,"Braund, Mr. Owen Harris", male,22,1,0,A/5 21171,7.25,,S

```
// BEGIN: regex
const csvString = '1,0,3,"Braund, Mr. Owen Harris", male,22,1,0,A/5 21171,7.25,,S';
const regexPattern = /("[^"]*"|[^,]+)(?:,|$)/g;
const csvArray = csvString.match(regexPattern).map((entry) => entry.replace(/^"(.*)"$/, '$1'));
console.log(csvArray);
// END: regex
```

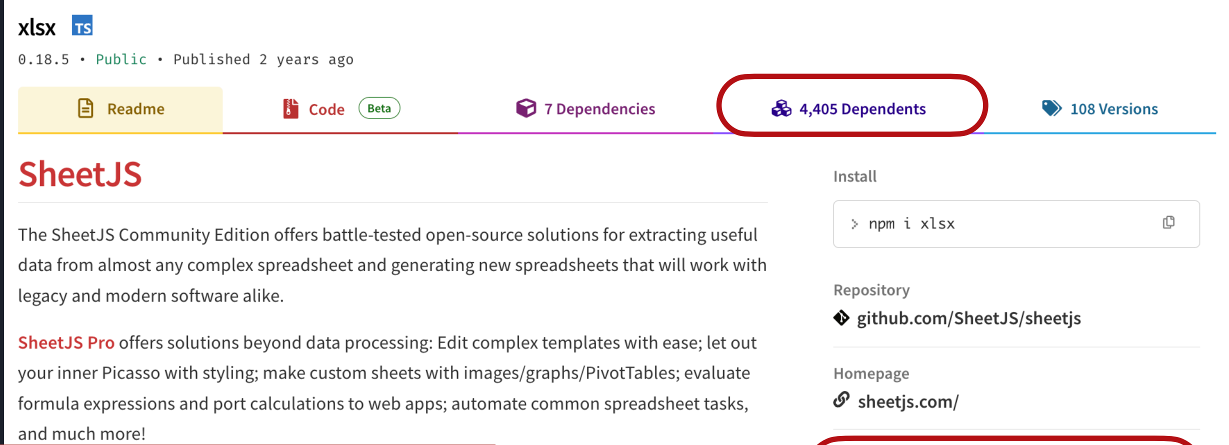
Github Copilot

Regex vulneráveis	Regex não vulneráveis
4	3

Google, Chat GPT e GitHub Copilot

Regex vulneráveis	Regex não vulneráveis
11	7

Minha primeira CVE 🎉



xlsx TS

0.18.5 • Public • Published 2 years ago

SheetJS

The SheetJS Community Edition offers battle-tested open-source solutions for extracting useful data from almost any complex spreadsheet and generating new spreadsheets that will work with legacy and modern software alike.

SheetJS Pro offers solutions beyond data processing: Edit complex templates with ease; let out your inner Picasso with styling; make custom sheets with images/graphs/PivotTables; evaluate formula expressions and port calculations to web apps; automate common spreadsheet tasks, and much more!

erabilities no longer available

Install

```
> npm i xlsx
```

Repository

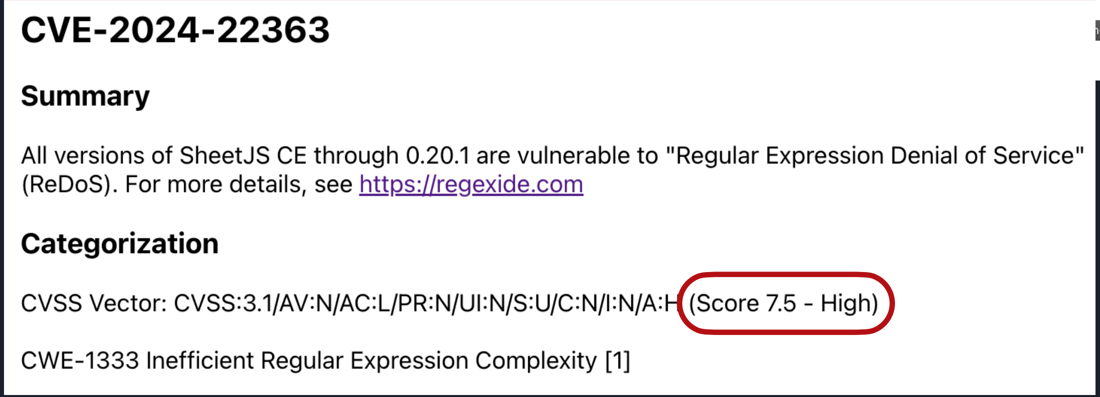
github.com/SheetJS/sheetjs

Homepage

sheetjs.com/

Weekly Downloads

1,906,276



CVE-2024-22363

Summary

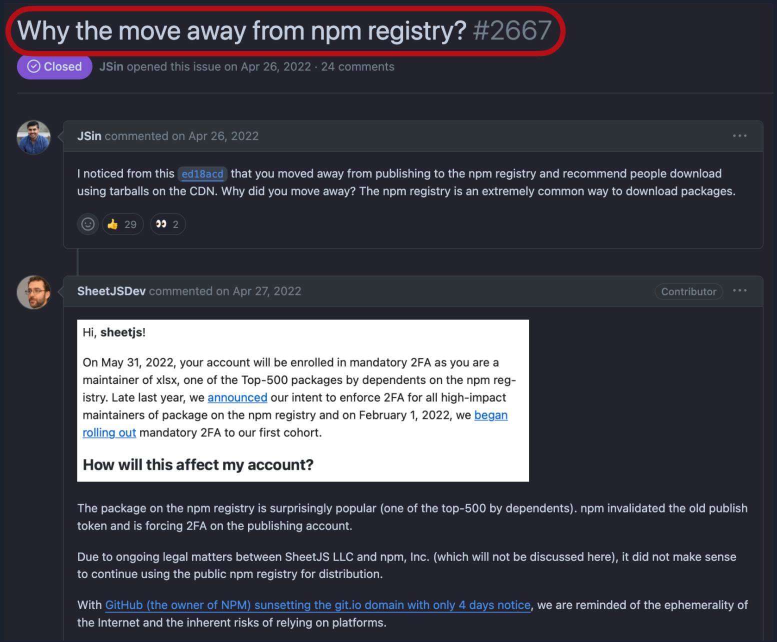
All versions of SheetJS CE through 0.20.1 are vulnerable to "Regular Expression Denial of Service" (ReDoS). For more details, see https://regexide.com

Categorization

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H (Score 7.5 - High)

CWE-1333 Inefficient Regular Expression Complexity [1]

Minha primeira CVE 🎉



Why the move away from npm registry? #2667

JSin opened this issue on Apr 26, 2022 · 24 comments

JSin commented on Apr 26, 2022

I noticed from this `ed18acd` that you moved away from publishing to the npm registry and recommend people download using tarballs on the CDN. Why did you move away? The npm registry is an extremely common way to download packages.

SheetJSDev commented on Apr 27, 2022

Hi, **sheetjs**!

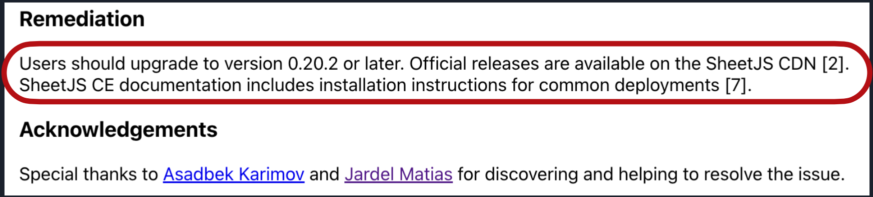
On May 31, 2022, your account will be enrolled in mandatory 2FA as you are a maintainer of xlsx, one of the Top-500 packages by dependents on the npm registry. Late last year, we announced our intent to enforce 2FA for all high-impact maintainers of package on the npm registry and on February 1, 2022, we began rolling out mandatory 2FA to our first cohort.

How will this affect my account?

The package on the npm registry is surprisingly popular (one of the top-500 by dependents). npm invalidated the old publish token and is forcing 2FA on the publishing account.

Due to ongoing legal matters between SheetJS LLC and npm, Inc. (which will not be discussed here), it did not make sense to continue using the public npm registry for distribution.

With GitHub (the owner of NPM) sunsetting the git.io domain with only 4 days notice, we are reminded of the ephemerality of the Internet and the inherent risks of relying on platforms.



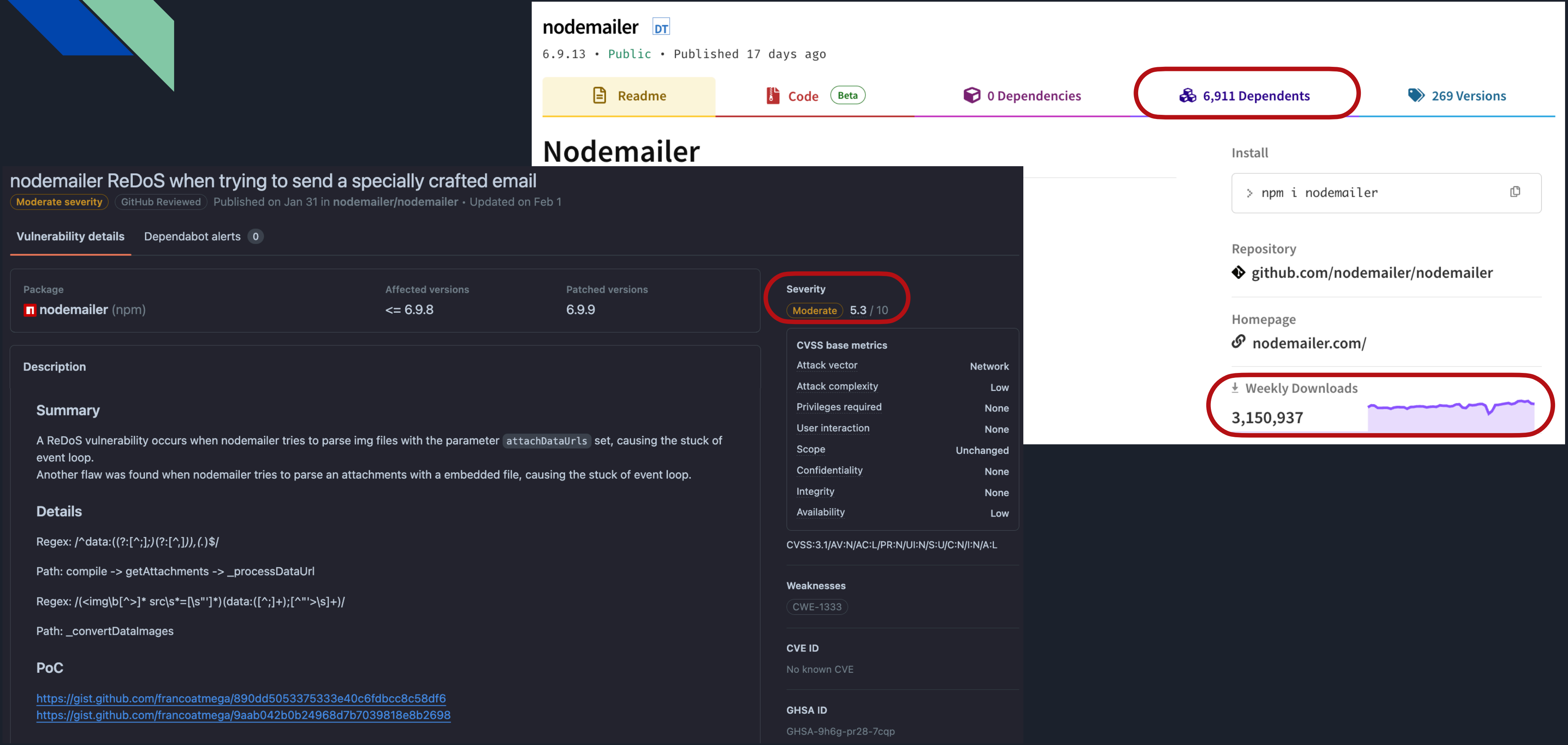
Remediation

Users should upgrade to version 0.20.2 or later. Official releases are available on the SheetJS CDN [2]. SheetJS CE documentation includes installation instructions for common deployments [7].

Acknowledgements

Special thanks to Asadbek Karimov and Jardel Matias for discovering and helping to resolve the issue.

Minha quase segunda CVE



nodemailer DT

6.9.13 • Public • Published 17 days ago

Nodemailer

Install

```
npm i nodemailer
```

Repository

github.com/nodemailer/nodemailer

Homepage

nodemailer.com/

Weekly Downloads

3,150,937

nodemailer ReDoS when trying to send a specially crafted email

Moderate severity · GitHub Reviewed · Published on Jan 31 in nodemailer/nodemailer · Updated on Feb 1

Package	Affected versions	Patched versions
nodemailer (npm)	<= 6.9.8	6.9.9

Severity

Moderate 5.3 / 10

CVSS base metrics

Attack vector	Network
Attack complexity	Low
Privileges required	None
User interaction	None
Scope	Unchanged
Confidentiality	None
Integrity	None
Availability	Low

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:L

Weaknesses

CWE-1333

CVE ID

No known CVE

GHSA ID

GHSA-9h6g-pr28-7cqp

Description

Summary

A ReDoS vulnerability occurs when nodemailer tries to parse img files with the parameter `attachDataUrls` set, causing the stuck of event loop.
Another flaw was found when nodemailer tries to parse an attachments with a embedded file, causing the stuck of event loop.

Details

Regex: /^data:((?:[^;];)(?:[^,])),(.)$/

Path: compile -> getAttachments -> _processDataUrl

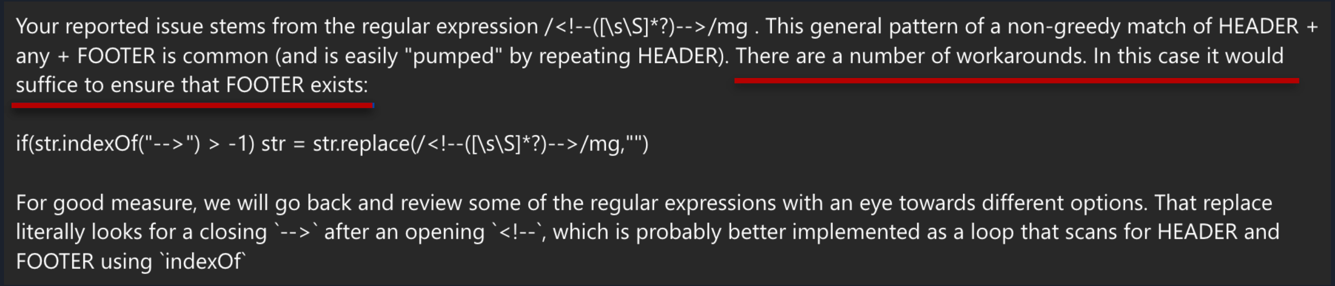
Regex: /(<img\b[^>]* src\s*=[\s"']*)(data:([^;]+);[^"'>\s]+)/

Path: _convertDataImages

PoC

https://gist.github.com/francoatmega/890dd5053375333e40c6fdbcc8c58df6
https://gist.github.com/francoatmega/9aab042b0b24968d7b7039818e8b2698

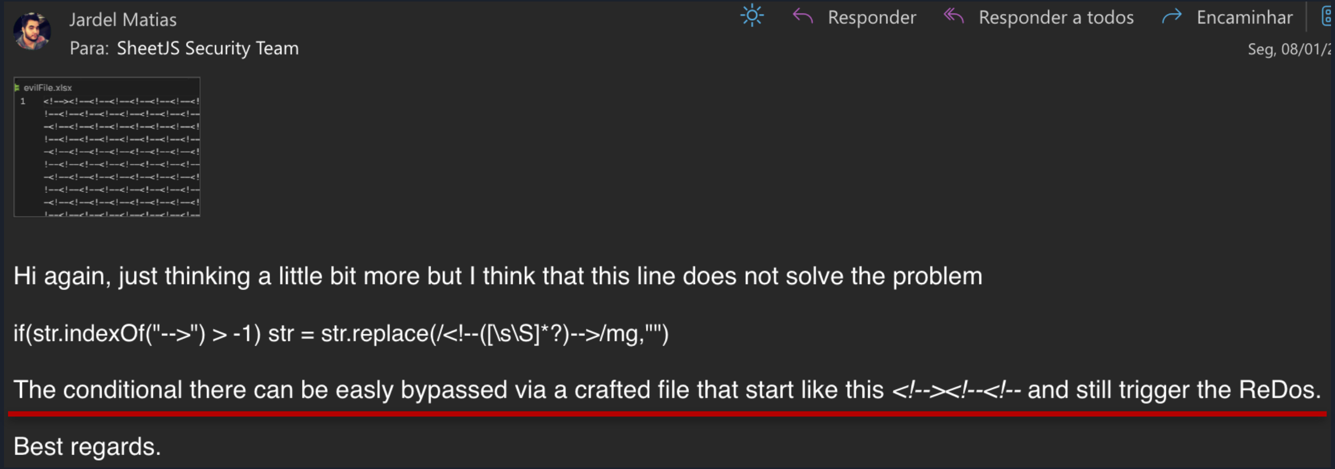
Fun facts



Your reported issue stems from the regular expression /<!--([\s\S]*?)-->/mg . This general pattern of a non-greedy match of HEADER + any + FOOTER is common (and is easily "pumped" by repeating HEADER). There are a number of workarounds. In this case it would suffice to ensure that FOOTER exists:

```
if(str.indexOf("-->") > -1) str = str.replace(/<!--([\s\S]*?)-->/mg,"")
```

For good measure, we will go back and review some of the regular expressions with an eye towards different options. That replace literally looks for a closing `-->` after an opening `<!--`, which is probably better implemented as a loop that scans for HEADER and FOOTER using `indexOf`



Jardel Matias
Para: SheetJS Security Team

Seg, 08/01/

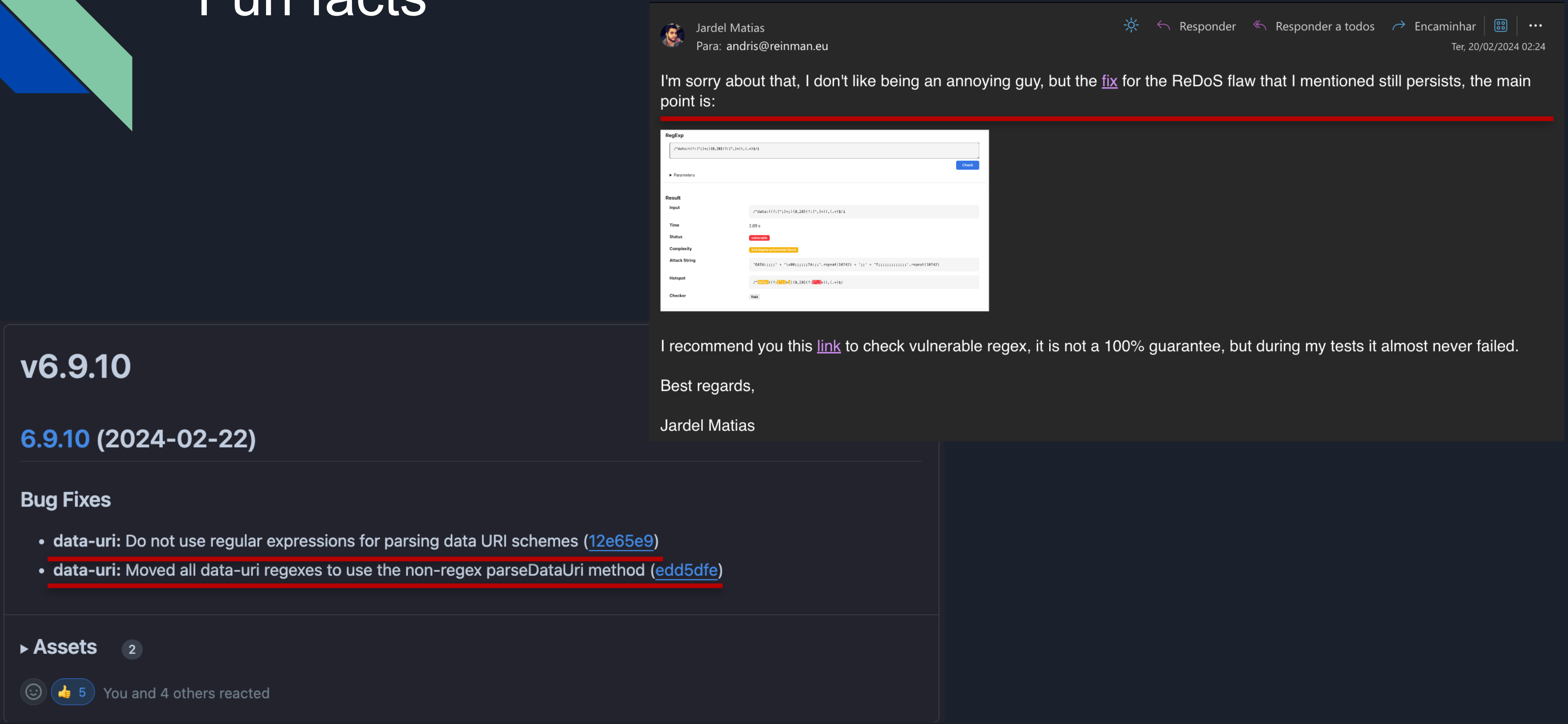
Hi again, just thinking a little bit more but I think that this line does not solve the problem

```
if(str.indexOf("-->") > -1) str = str.replace(/<!--([\s\S]*?)-->/mg,"")
```

The conditional there can be easly bypassed via a crafted file that start like this *<!--><!--<!--* and still trigger the ReDos.

Best regards.

Fun facts



Jardel Matias
Para: andris@reinman.eu
Ter, 20/02/2024 02:24

I'm sorry about that, I don't like being an annoying guy, but the fix for the ReDoS flaw that I mentioned still persists, the main point is:

RegExp

```
/^data:((?:[^;]*;){0,20}(?:[^,]*)),(.*)$/i
```

Result

Input	`/^data:((?:[^;]*;){0,20}(?:[^,]*)),(.*)$/i`
Time	2.89 s
Status	vulnerable
Complexity	2nd degree polynomial (fuzz)
Attack String	`'DATA:;;;;' + '\x00;;;;;;TA:;;'.repeat(10742) + ';;' + 'T;;;;;;;;;;;;;'.repeat(10742)`
Hotspot	`/^data:((?:[^;]*;){0,20}(?:[^,]*)),(.*)$/`
Checker	fuzz

I recommend you this link to check vulnerable regex, it is not a 100% guarantee, but during my tests it almost never failed.

Best regards,

Jardel Matias

---

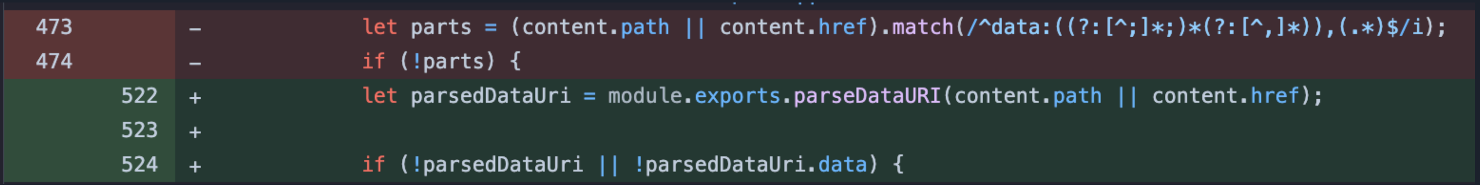
v6.9.10

6.9.10 (2024-02-22)

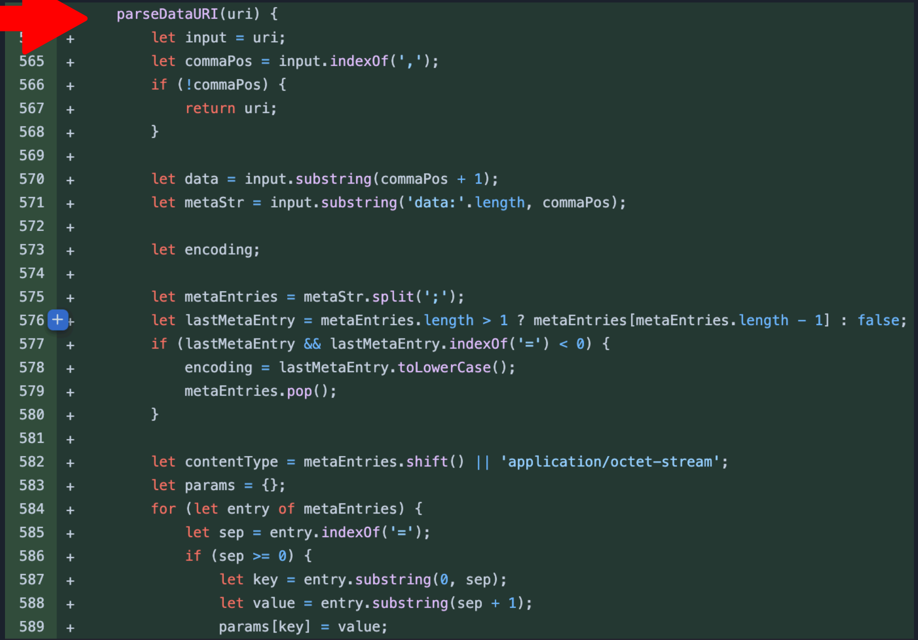
Bug Fixes

- **data-uri:** Do not use regular expressions for parsing data URI schemes (12e65e9)
- **data-uri:** Moved all data-uri regexes to use the non-regex parseDataUri method (edd5dfe)

Fun facts

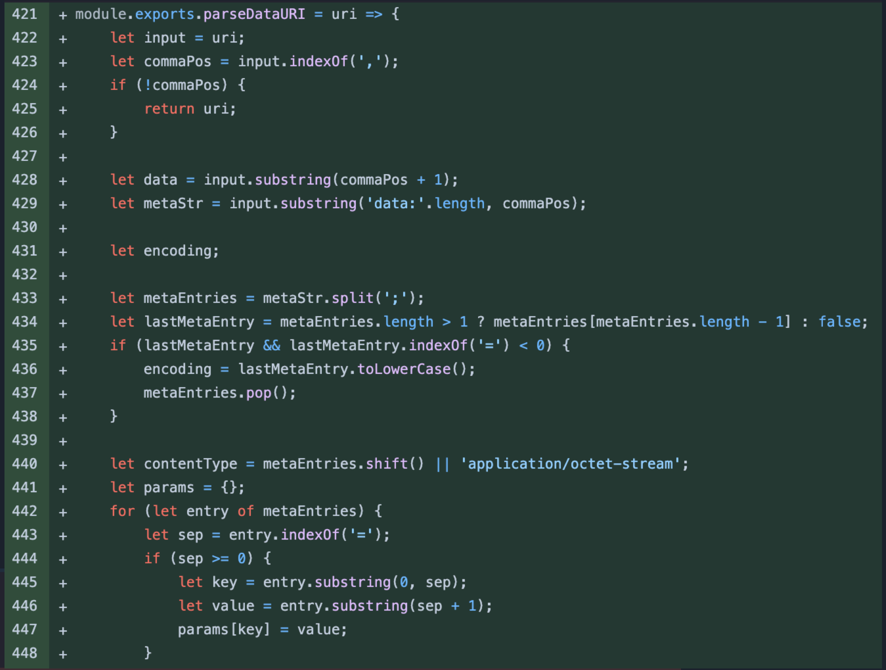


```
-         let parts = (content.path || content.href).match(/^data:((?:[^;]*;)*(?:[^,]*)),(.*)$/i);
-         if (!parts) {
+         let parsedDataUri = module.exports.parseDataURI(content.path || content.href);
+
+         if (!parsedDataUri || !parsedDataUri.data) {
```

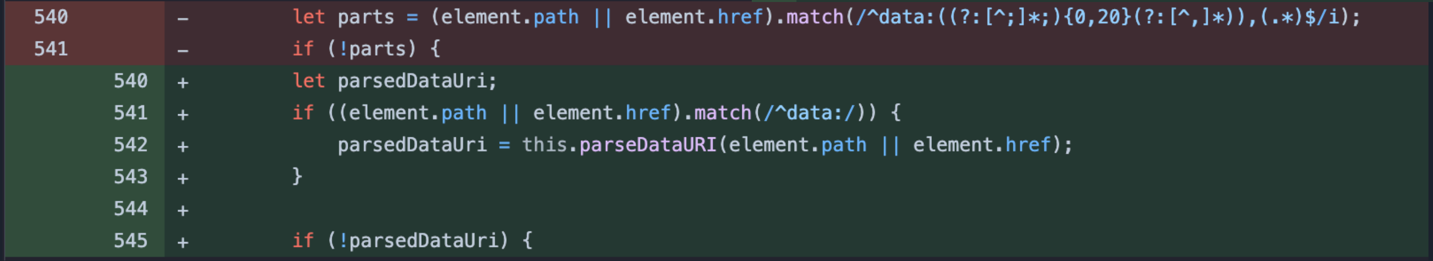


```
      parseDataURI(uri) {
+         let input = uri;
+         let commaPos = input.indexOf(',');
+         if (!commaPos) {
+             return uri;
+         }
+
+         let data = input.substring(commaPos + 1);
+         let metaStr = input.substring('data:'.length, commaPos);
+
+         let encoding;
+
+         let metaEntries = metaStr.split(';');
+         let lastMetaEntry = metaEntries.length > 1 ? metaEntries[metaEntries.length - 1] : false;
+         if (lastMetaEntry && lastMetaEntry.indexOf('=') < 0) {
+             encoding = lastMetaEntry.toLowerCase();
+             metaEntries.pop();
+         }
+
+         let contentType = metaEntries.shift() || 'application/octet-stream';
+         let params = {};
+         for (let entry of metaEntries) {
+             let sep = entry.indexOf('=');
+             if (sep >= 0) {
+                 let key = entry.substring(0, sep);
+                 let value = entry.substring(sep + 1);
+                 params[key] = value;
```

Fun facts



```
+ module.exports.parseDataURI = uri => {
+     let input = uri;
+     let commaPos = input.indexOf(',');
+     if (!commaPos) {
+         return uri;
+     }
+
+     let data = input.substring(commaPos + 1);
+     let metaStr = input.substring('data:'.length, commaPos);
+
+     let encoding;
+
+     let metaEntries = metaStr.split(';');
+     let lastMetaEntry = metaEntries.length > 1 ? metaEntries[metaEntries.length - 1] : false;
+     if (lastMetaEntry && lastMetaEntry.indexOf('=') < 0) {
+         encoding = lastMetaEntry.toLowerCase();
+         metaEntries.pop();
+     }
+
+     let contentType = metaEntries.shift() || 'application/octet-stream';
+     let params = {};
+     for (let entry of metaEntries) {
+         let sep = entry.indexOf('=');
+         if (sep >= 0) {
+             let key = entry.substring(0, sep);
+             let value = entry.substring(sep + 1);
+             params[key] = value;
+         }
```



```
       * @return {Object} Parsed element
       */
      _processDataUrl(element) {
-         let parts = (element.path || element.href).match(/^data:((?:[^;]*;){0,20}(?:[^,]*)),(.*)$/i);
-         if (!parts) {
+         let parsedDataUri;
+         if ((element.path || element.href).match(/^data:/)) {
+             parsedDataUri = this.parseDataURI(element.path || element.href);
+         }
+
+         if (!parsedDataUri) {
```

Como prevenir ataques ReDoS?

- Utilize bibliotecas de terceiros.

- Evite criar regex que possam levar a resultados ambíguos, evite agrupamentos com repetições ou alternância. Como por exemplo:

```
^data:(\w*\/\w*);*(charset=[\w=-]*)*;*$

^([a-zA-Z0-9])(([\-.]|[_]+)?([a-zA-Z0-9]+))*(@){1}[a-z0-9]+[.]{1}(([a-z]{2,3})|([a-z]{2,3}[.]{1}[a-z]{2,3}))$

^-?\D*[\d.]+,\d{2}$

^(a|a)*$
```

- Utilize engines de regex que não sejam vulneráveis a backtracing, como por exemplo a re2 da Google.*

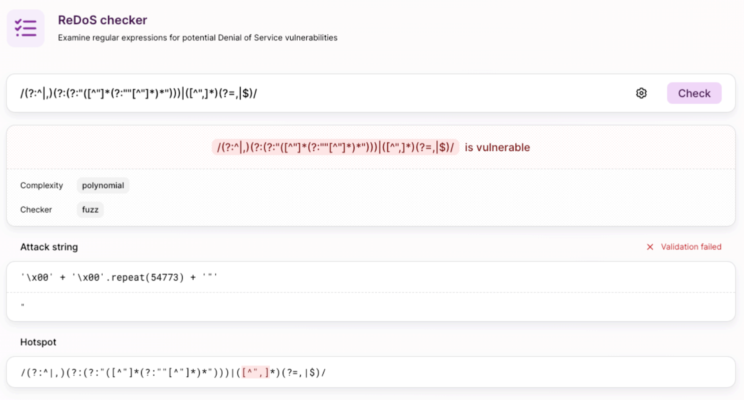
- Mantenham suas dependências atualizadas.

---

*Linguagens como Go e Rust não são vulneráveis.

Como prevenir ataques ReDoS?

- Valide as suas regex
- https://devina.io/redos-checker
- Projeto **recheck**
- Biblioteca JS, biblioteca Scala e Plugin ESLint
- https://makenowjust-labs.github.io/recheck/



ReDoS checker

Examine regular expressions for potential Denial of Service vulnerabilities

```
/(?:^|,)(?:(?:"([^"]*(?:""[^"]*)*")))|([^",]*)(?=,|$)/
```

`/(?:^|,)(?:(?:"([^"]*(?:""[^"]*)*")))|([^",]*)(?=,|$)/` is vulnerable

Complexity: polynomial

Checker: fuzz

Attack string — ✕ Validation failed

```
'\x00' + '\x00'.repeat(54773) + '"'
"
```

Hotspot

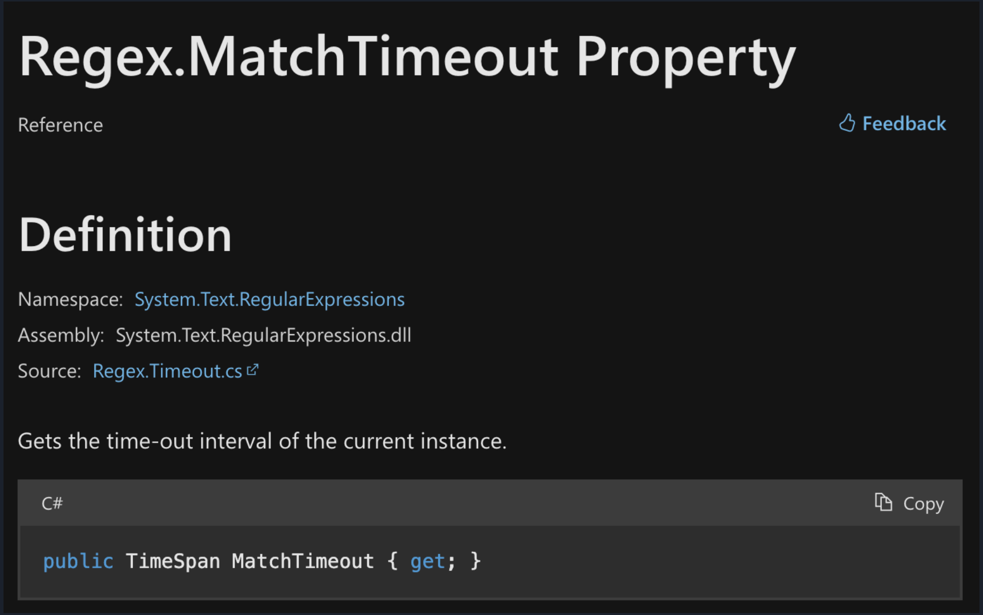
```
/(?:^|,)(?:(?:"([^"]*(?:""[^"]*)*")))|([^",]*)(?=,|$)/
```

recheck

The trustworthy ReDoS checker

Como prevenir ataques ReDoS?

- Utilize as funcionalidades da linguagem que você utiliza
- C# (Backtrack abortion)



Regex.MatchTimeout Property

Reference

Definition

Namespace: System.Text.RegularExpressions

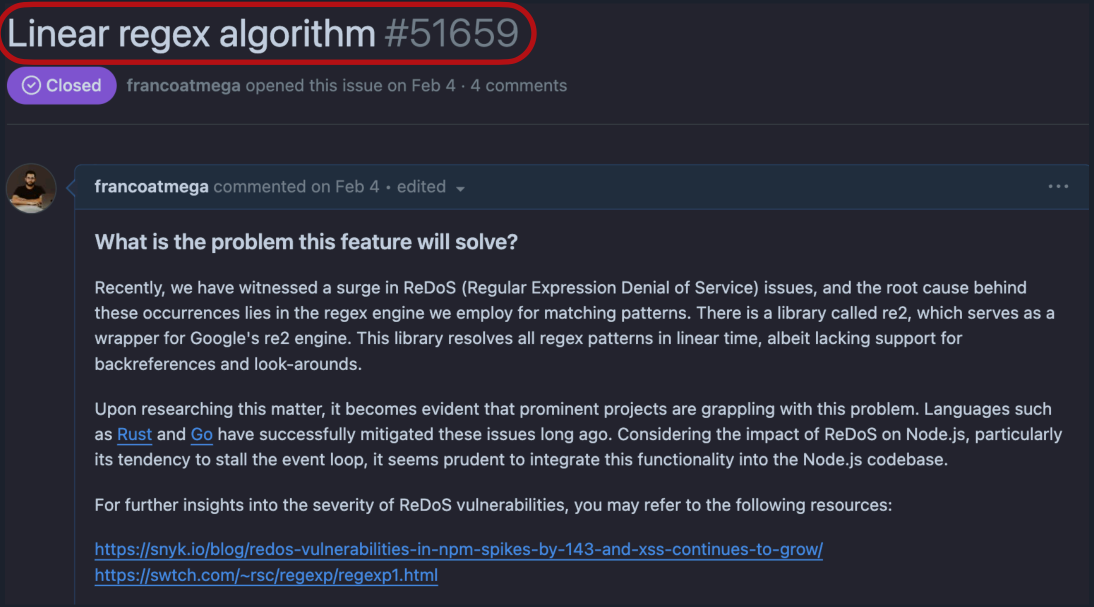
Assembly: System.Text.RegularExpressions.dll

Source: Regex.Timeout.cs

Gets the time-out interval of the current instance.

```csharp
public TimeSpan MatchTimeout { get; }
```

Como prevenir ataques ReDoS?



Linear regex algorithm #51659

francoatmega opened this issue on Feb 4 · 4 comments

francoatmega commented on Feb 4 • edited

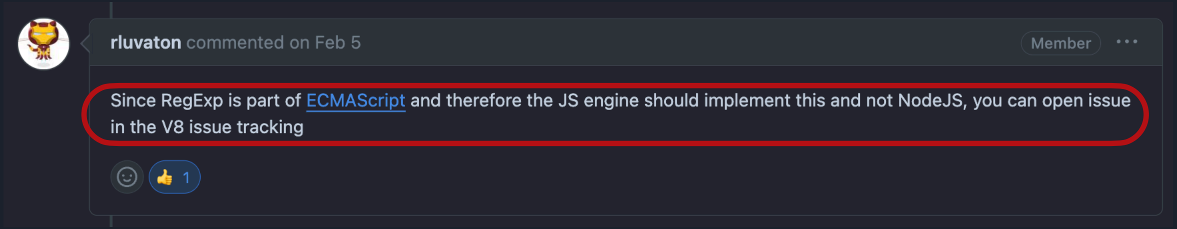
What is the problem this feature will solve?

Recently, we have witnessed a surge in ReDoS (Regular Expression Denial of Service) issues, and the root cause behind these occurrences lies in the regex engine we employ for matching patterns. There is a library called re2, which serves as a wrapper for Google's re2 engine. This library resolves all regex patterns in linear time, albeit lacking support for backreferences and look-arounds.

Upon researching this matter, it becomes evident that prominent projects are grappling with this problem. Languages such as Rust and Go have successfully mitigated these issues long ago. Considering the impact of ReDoS on Node.js, particularly its tendency to stall the event loop, it seems prudent to integrate this functionality into the Node.js codebase.

For further insights into the severity of ReDoS vulnerabilities, you may refer to the following resources:

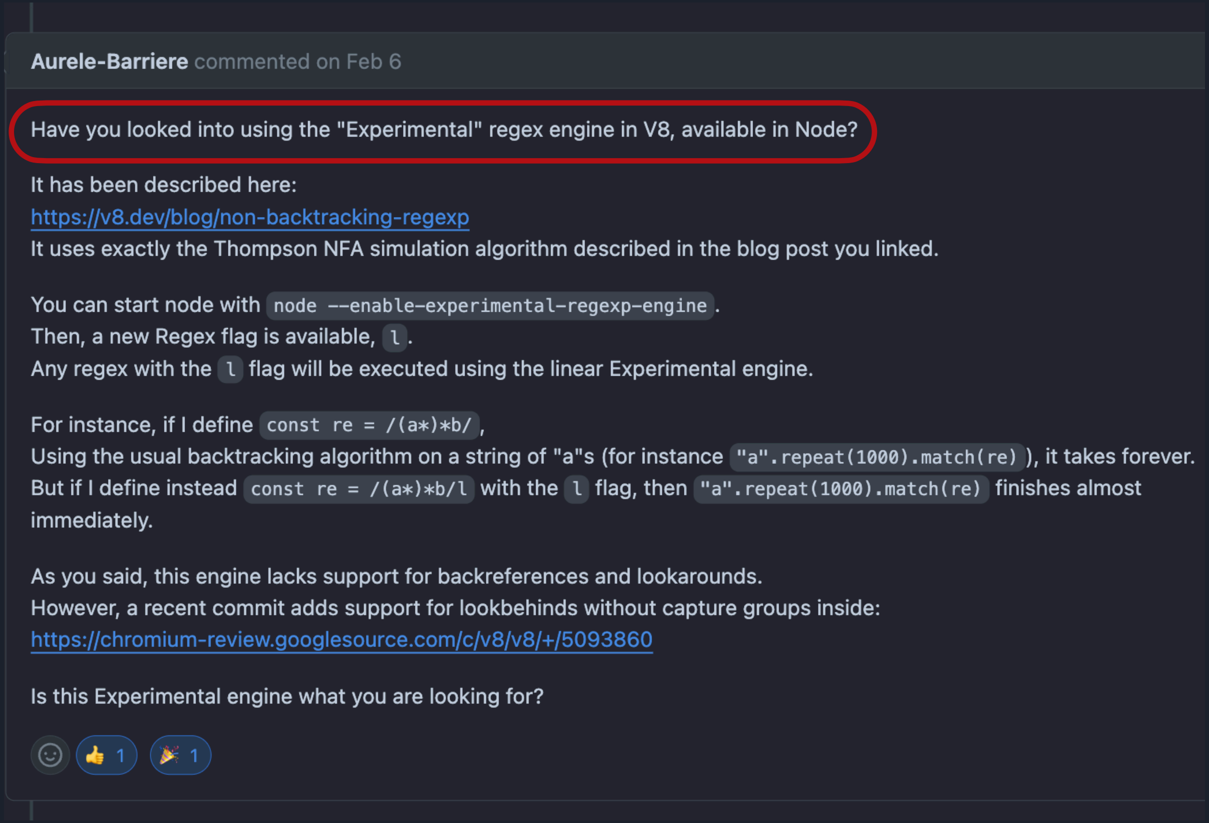
https://snyk.io/blog/redos-vulnerabilities-in-npm-spikes-by-143-and-xss-continues-to-grow/
https://swtch.com/~rsc/regexp/regexp1.html



rluvaton commented on Feb 5

Since RegExp is part of ECMAScript and therefore the JS engine should implement this and not NodeJS, you can open issue in the V8 issue tracking

Como prevenir ataques ReDoS?



Aurele-Barriere commented on Feb 6

Have you looked into using the "Experimental" regex engine in V8, available in Node?

It has been described here:
https://v8.dev/blog/non-backtracking-regexp
It uses exactly the Thompson NFA simulation algorithm described in the blog post you linked.

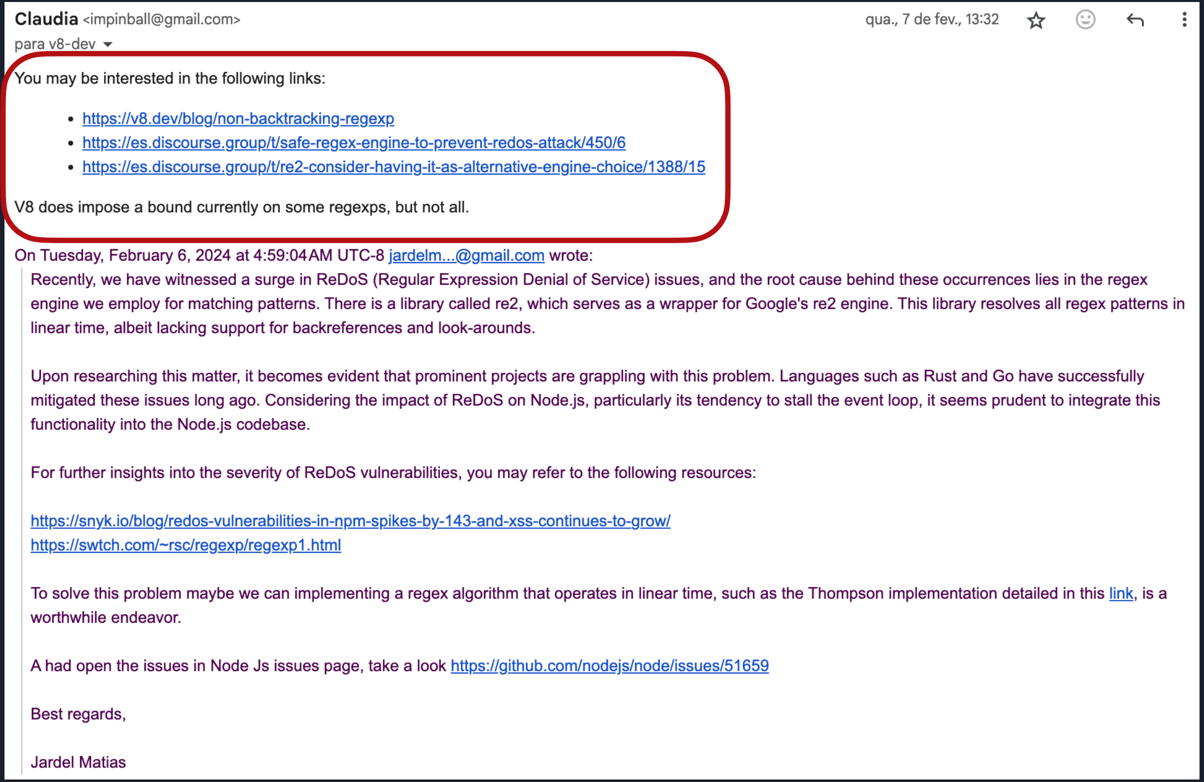
You can start node with `node --enable-experimental-regexp-engine`.
Then, a new Regex flag is available, `l`.
Any regex with the `l` flag will be executed using the linear Experimental engine.

For instance, if I define `const re = /(a*)*b/`,
Using the usual backtracking algorithm on a string of "a"s (for instance `"a".repeat(1000).match(re)`), it takes forever.
But if I define instead `const re = /(a*)*b/l` with the `l` flag, then `"a".repeat(1000).match(re)` finishes almost immediately.

As you said, this engine lacks support for backreferences and lookarounds.
However, a recent commit adds support for lookbehinds without capture groups inside:
https://chromium-review.googlesource.com/c/v8/v8/+/5093860

Is this Experimental engine what you are looking for?

Como prevenir ataques ReDoS?



Claudia <impinball@gmail.com>
para v8-dev

You may be interested in the following links:

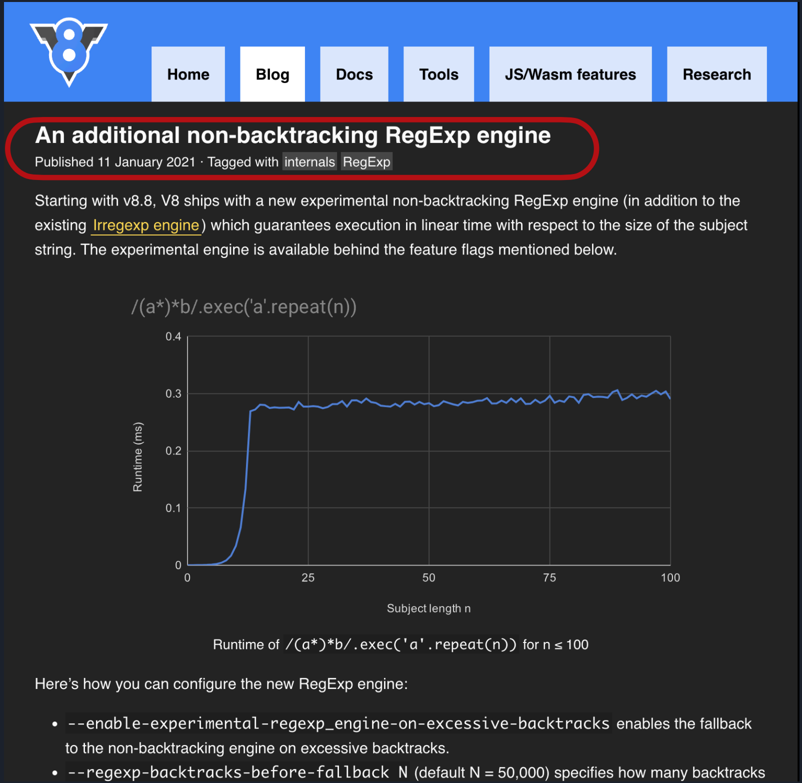
- https://v8.dev/blog/non-backtracking-regexp
- https://es.discourse.group/t/safe-regex-engine-to-prevent-redos-attack/450/6
- https://es.discourse.group/t/re2-consider-having-it-as-alternative-engine-choice/1388/15

V8 does impose a bound currently on some regexps, but not all.

On Tuesday, February 6, 2024 at 4:59:04 AM UTC-8 jardelm...@gmail.com wrote:

> Recently, we have witnessed a surge in ReDoS (Regular Expression Denial of Service) issues, and the root cause behind these occurrences lies in the regex engine we employ for matching patterns. There is a library called re2, which serves as a wrapper for Google's re2 engine. This library resolves all regex patterns in linear time, albeit lacking support for backreferences and look-arounds.
>
> Upon researching this matter, it becomes evident that prominent projects are grappling with this problem. Languages such as Rust and Go have successfully mitigated these issues long ago. Considering the impact of ReDoS on Node.js, particularly its tendency to stall the event loop, it seems prudent to integrate this functionality into the Node.js codebase.
>
> For further insights into the severity of ReDoS vulnerabilities, you may refer to the following resources:
>
> https://snyk.io/blog/redos-vulnerabilities-in-npm-spikes-by-143-and-xss-continues-to-grow/
> https://swtch.com/~rsc/regexp/regexp1.html
>
> To solve this problem maybe we can implementing a regex algorithm that operates in linear time, such as the Thompson implementation detailed in this link, is a worthwhile endeavor.
>
> A had open the issues in Node Js issues page, take a look https://github.com/nodejs/node/issues/51659
>
> Best regards,
>
> Jardel Matias

Como prevenir ataques ReDoS?



An additional non-backtracking RegExp engine

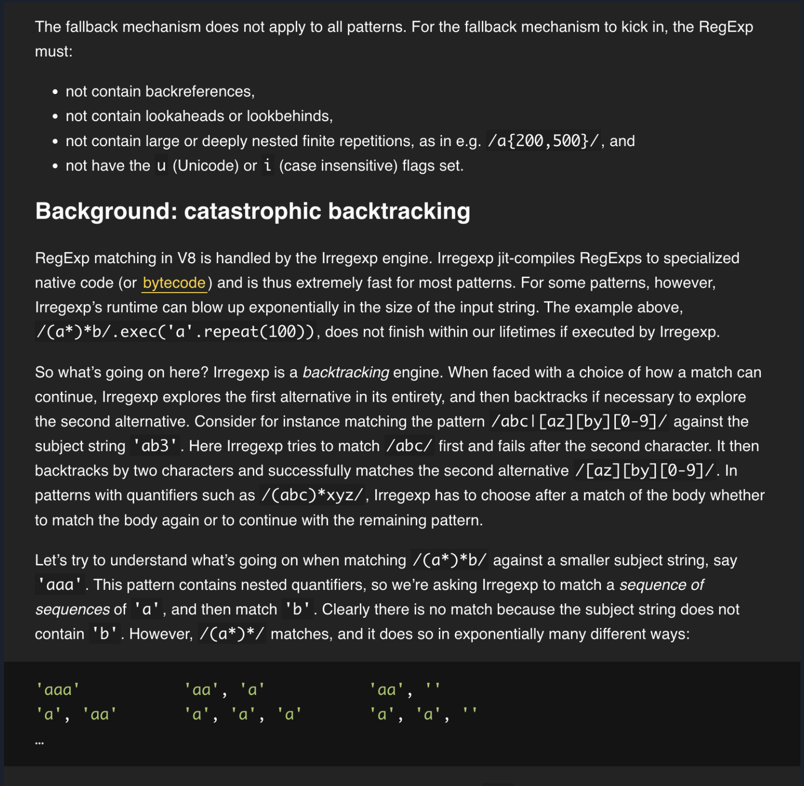
Published 11 January 2021 · Tagged with internals RegExp

Starting with v8.8, V8 ships with a new experimental non-backtracking RegExp engine (in addition to the existing Irregexp engine) which guarantees execution in linear time with respect to the size of the subject string. The experimental engine is available behind the feature flags mentioned below.

Runtime of `/(a*)*b/.exec('a'.repeat(n))` for n ≤ 100

Here's how you can configure the new RegExp engine:

- `--enable-experimental-regexp_engine-on-excessive-backtracks` enables the fallback to the non-backtracking engine on excessive backtracks.
- `--regexp-backtracks-before-fallback N` (default N = 50,000) specifies how many backtracks



The fallback mechanism does not apply to all patterns. For the fallback mechanism to kick in, the RegExp must:

- not contain backreferences,
- not contain lookaheads or lookbehinds,
- not contain large or deeply nested finite repetitions, as in e.g. `/a{200,500}/`, and
- not have the `u` (Unicode) or `i` (case insensitive) flags set.

Background: catastrophic backtracking

RegExp matching in V8 is handled by the Irregexp engine. Irregexp jit-compiles RegExps to specialized native code (or bytecode) and is thus extremely fast for most patterns. For some patterns, however, Irregexp's runtime can blow up exponentially in the size of the input string. The example above, `/(a*)*b/.exec('a'.repeat(100))`, does not finish within our lifetimes if executed by Irregexp.

So what's going on here? Irregexp is a *backtracking* engine. When faced with a choice of how a match can continue, Irregexp explores the first alternative in its entirety, and then backtracks if necessary to explore the second alternative. Consider for instance matching the pattern `/abc|[az][by][0-9]/` against the subject string `'ab3'`. Here Irregexp tries to match `/abc/` first and fails after the second character. It then backtracks by two characters and successfully matches the second alternative `/[az][by][0-9]/`. In patterns with quantifiers such as `/(abc)*xyz/`, Irregexp has to choose after a match of the body whether to match the body again or to continue with the remaining pattern.

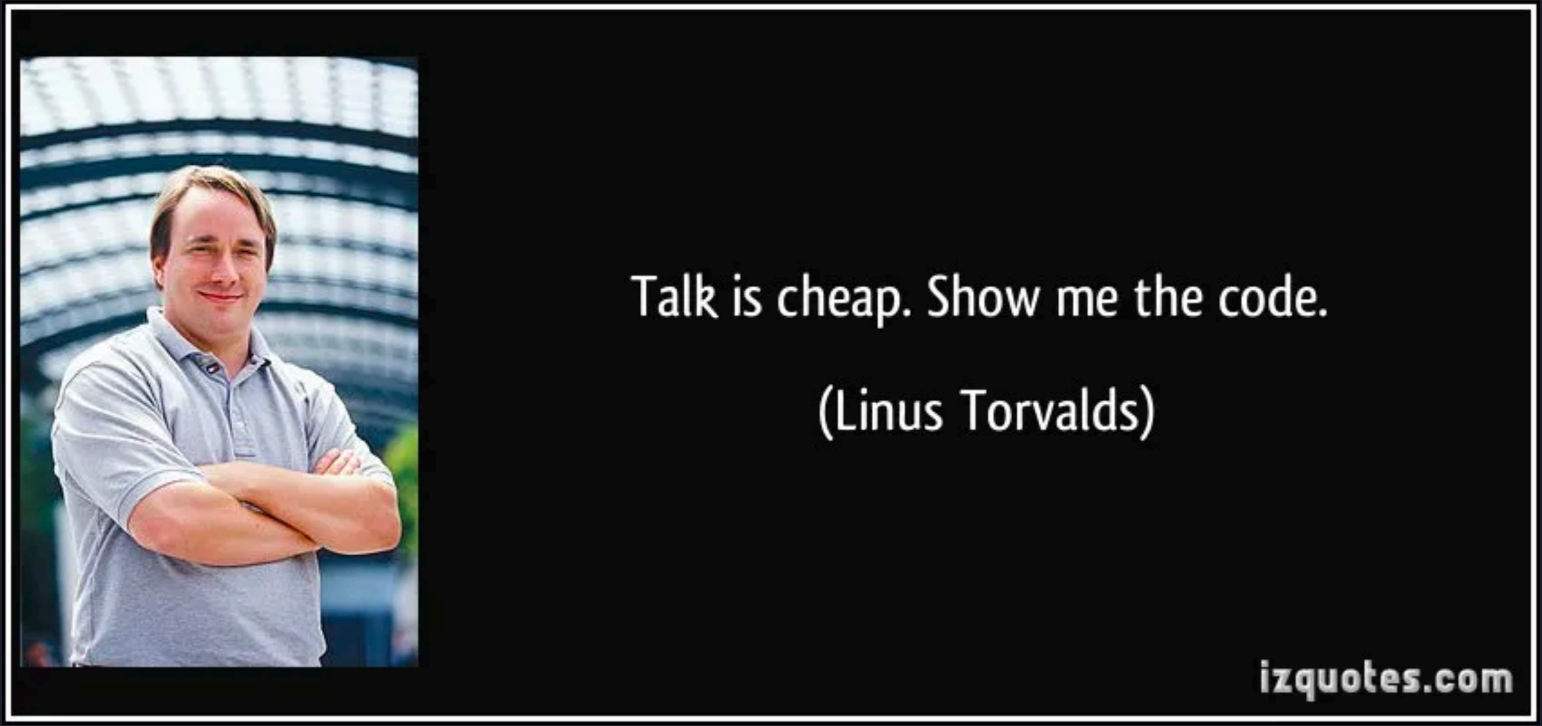
Let's try to understand what's going on when matching `/(a*)*b/` against a smaller subject string, say `'aaa'`. This pattern contains nested quantifiers, so we're asking Irregexp to match a *sequence of sequences* of `'a'`, and then match `'b'`. Clearly there is no match because the subject string does not contain `'b'`. However, `/(a*)*/` matches, and it does so in exponentially many different ways:

```
'aaa'          'aa', 'a'          'aa', ''
'a', 'aa'      'a', 'a', 'a'      'a', 'a', ''
…
```

Como prevenir ataques ReDoS?

- Habilitar a feature flag **--enable-experimental-regexp-engine**
- Adicionar a flag ***l*** nas regex
- Be happy ;)

Como prevenir ataques ReDoS?



Talk is cheap. Show me the code.

(Linus Torvalds)

izquotes.com

Referências

https://owasp.org/www-community/attacks/Regular_expression_Denial_of_Service_-_ReDoS

https://en.wikipedia.org/wiki/Regular_expression

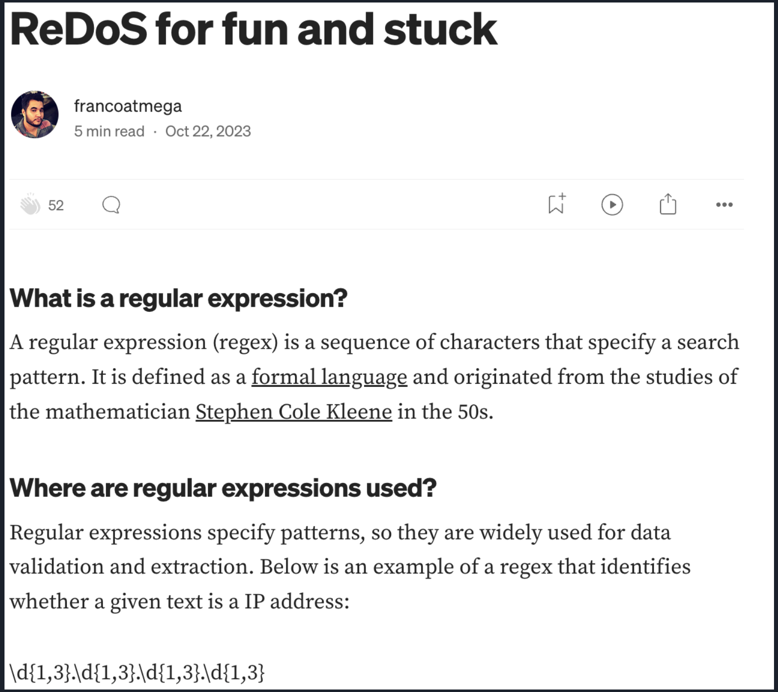
https://www.regular-expressions.info/catastrophic.html

https://lirantal.medium.com/node-js-pitfalls-how-a-regex-can-bring-your-system-down-cbf1dc6c4e02

https://levelup.gitconnected.com/the-regular-expression-denial-of-service-redos-cheat-sheet-a78d0ed7d865

Dúvidas

Post sobre ReDos



ReDoS for fun and stuck

francoatmega
5 min read · Oct 22, 2023

What is a regular expression?

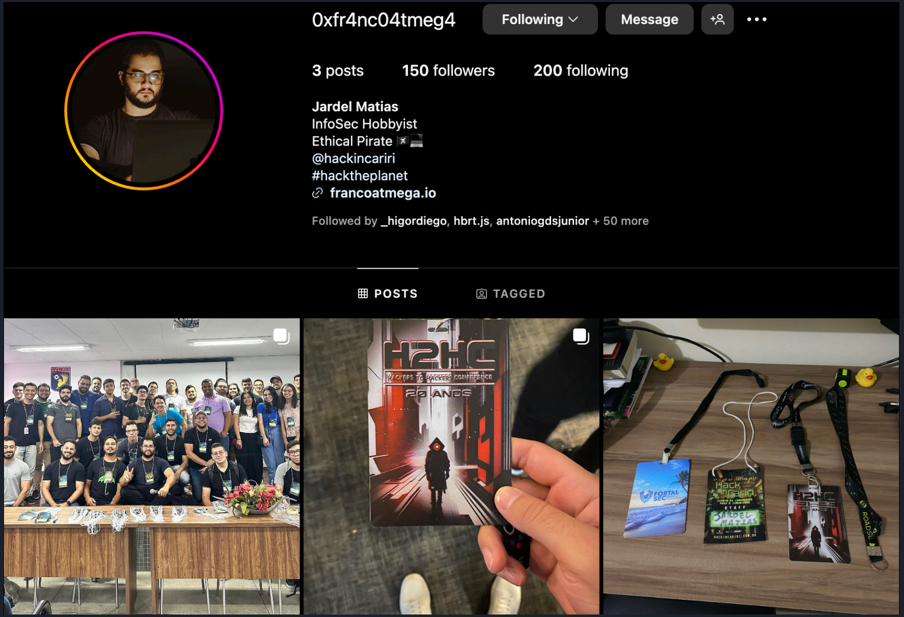
A regular expression (regex) is a sequence of characters that specify a search pattern. It is defined as a formal language and originated from the studies of the mathematician Stephen Cole Kleene in the 50s.

Where are regular expressions used?

Regular expressions specify patterns, so they are widely used for data validation and extraction. Below is an example of a regex that identifies whether a given text is a IP address:

```
\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3}
```

Contato



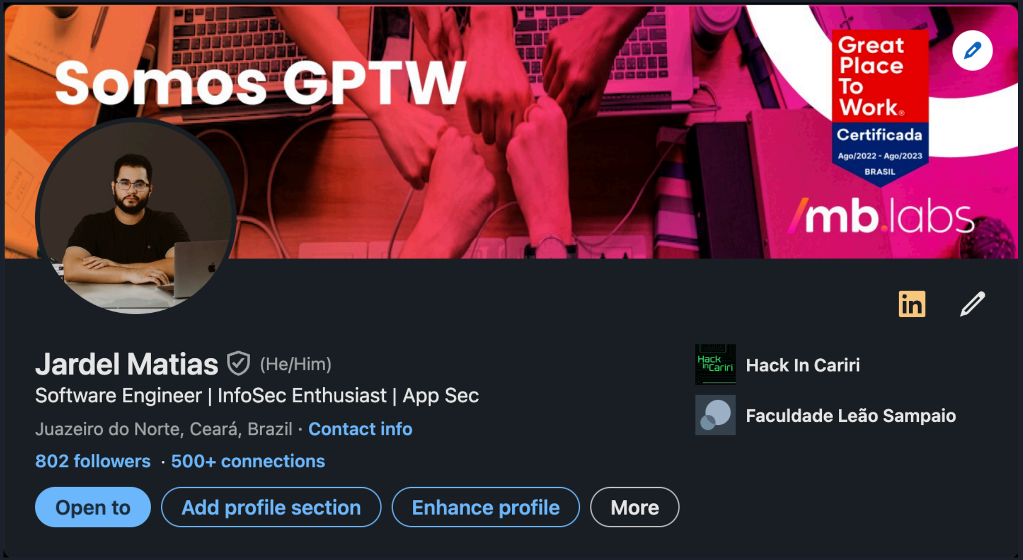
0xfr4nc04tmeg4

3 posts 150 followers 200 following

Jardel Matias
InfoSec Hobbyist
Ethical Pirate ☠💻
@hackincariri
#hacktheplanet
francoatmega.io

Followed by _higordiego, hbrt.js, antoniogdsjunior + 50 more

Contato



Jardel Matias (He/Him)

Software Engineer | InfoSec Enthusiast | App Sec

Juazeiro do Norte, Ceará, Brazil · Contact info

802 followers · 500+ connections

Hack In Cariri

Faculdade Leão Sampaio