New issue

Jump to bottom

[Vulnerability Report Disclosure: ReDoS] Catastrophic Backtracking in User-supplied Regular Expression #10045

Closed **Rongronggg9** opened this issue on Jun 26 · 2 comments

Labels announcement core bug security

Rongronggg9 commented on Jun 26 • edited Contributor

The vulnerability has been fixed (refer to the security advisory). According to the consultation with **@DIYgod**, this is a complete disclosure of the vulnerability report, disclosed 120 hours after the fix commit.

For a better reading experience, download the PDF here.

简要的简体中文翻译附在下一条回复。

Vulnerability Report (RSSHub)

Regular expression Denial of Service (ReDoS)
Catastrophic Backtracking in User-supplied Regular Expression
Author: Rongrong (**@Rongronggg9**)

POC

After reporting to the repository owner, the vulnerability has been fixed (refer to the security advisory). To reproduce the vulnerability, a version before `041cfc3` (inclusive) is needed. `diygod/rsshub:2022-06-20` is the last vulnerable Docker image.

1. Follow the official guide to deploy an RSSHub instance in any method (except Vercel or GAE).
2. `curl 'http://example.com/test/complicated?filter=%28%5B%5E%3C%3E%27%22%5D%2B%29%2A%22%3C&filterout=%28%5B%5E%3C%3E%27%22%5D%2B%29%2A%22%3C'`
3. The instance is now unresponsive to any request (Denial of Service).
4. Monitor the CPU usage of the process `node index.js`. It is now always occupying a whole CPU core. It can last for at least several hours.

EXP

`example.com` is an RSSHub instance.
`/test/complicated` can be nearly any "route" of RSSHub, here is a randomly chosen "test route".
`filter`, `filter_title`, `filter_description`, `filter_author`, `filterout`, `filterout_title`, `filterout_description`, and `filterout_author` are all so-called parameters of RSSHub, which can be supplied in the URI query. These parameters accept user-supplied regular expressions with unconditional trust, then call `String.match()` to perform regular expression matches. All of these parameters are vulnerable.
`%28%5B%5E%3C%3E%27%22%5D%2B%29%2A%22%3C` is the URL encoded form of the regular expression / `([^<>'"]+)*"<` /. The POC shows a catastrophic-backtracking-vulnerable regular expression specially designed for any HTML. But most catastrophic-backtracking-vulnerable regular expression may be able to construct an effective DoS attack (refer to the "Look back" chapter for more details).

Once catastrophic backtracking occurs, `String.match()` can take more than several hours to finish. There is no timeout enforced, causing the ReDoS attack to become "zero-cost": attack once, down for hours. On multi-core servers, the ReDoS attack can probably only affect the RSSHub instance itself; while on single-core servers, especially VPS[1], it can be a disaster.

The component cannot be disabled. Every RSSHub instance is vulnerable "out-of-box" unless additional access control is applied. If an external watchdog or timeout is enforced (e.g. Vercel, GAE), the downtime of each effective attack can be limited. However, the instance must be terminated first in order to resume it, so the attack cost is still too low.

Vulnerable version

Since Git hash `4671720`, Pull Request #3910 on GitHub, which was merge on Feb 9, 2020 (28.5 months ago).
Fixed in `5c41774`.

Condition to be vulnerable

Unless additional access control is applied.
If an external watchdog or timeout is enforced, the downtime can be limited but the instance is still vulnerable.

Vulnerability grade

High.

Possible fix

Either:

1. Drop the regular expression match feature completely and roll back to string match.

2. Enforce additional permission check when accepting user-supplied regular expressions.
3. Migrate to a backtracking-free regular expression engine.
4. (not really[2] a "fix") Enforce a timeout when performing a regular expression match.
5. (not really[3] a "fix") Disable the feature by default and require each instance maintainer to manually enable it.

Timeline

Considering the fix of the vulnerability is simple, I prefer a 28+7 days timeline (UTC).

Jun 21, 2022 (Day 0)

- Vulnerability found.
- Vulnerability reported.

Jul 5, 2022 (Day 14)

- If the email to i@diygod.me gets no response, I will immediately open an issue in the GitHub repository, only informing maintainers there is a DoS vulnerability without disclosing any vulnerability detail.

Jun 22, 2022 - Jul 19, 2022 (Day 1 - Day 28)

- If a fix is committed and pushed to the GitHub repository during this period, I will wait for 120 hours in order that maintainers can disclose a security advisory on GitHub. No matter whether the security advisory has been disclosed or not, once the 120-hour deadline is exceeded, I will immediately disclose the full report[4] by opening an issue in the GitHub repository.

Jul 20, 2022 - Jul 26, 2022 (Day 29 - Day 35)

- Once a fix is committed and pushed to the GitHub repository during this period, I will immediately disclose the full report by opening an issue in the GitHub repository.

Jul 27, 2022 (Day 36)

- If no fix was committed and pushed to the GitHub repository during the previous two periods, I will immediately disclose the full report by opening an issue in the GitHub repository.

Additional conditions

- Maintainers may ask me to disclose the full report before exceeding the 120-hour deadline.
- Maintainers may consult with me to determine a new timeline, but that consultation must be disclosed along with the full report.
- If being replied "won't fix" or "not a vulnerability", I will immediately disclose the full report along with the reply by opening an issue in the GitHub repository.
- I reserve the right to apply for a CVE-ID.

Timeline (in reality)

Jun 21, 2022 15:31 UTC (0h)

- Vulnerability reported.

Jun 21, 2022 15:56 UTC (0.4h)

- Vulnerability fixed.
  - Replace the regular expression engine with a backtracking-free one (`RE2`).
  - Git hash: `5c41774`

Jun 21, 2022 17:52 UTC (2.4h)

- The fix was confirmed to be effective.

Jun 22, 2022 10:43 UTC (19.2h)

- The security advisory was published.

Jun 22, 2022 19:25 UTC (27.9h)

- Two commits "for backward compatibility" opened up the attack surface again by allowing instance maintainers to switch back to the vulnerable regular expression engine.
  - Git hash: `e4478a3`
    - Pull request: **feat(core): regex engine config** #10013
  - Git hash: `dc64deb`
    - Pull request: **feat(core): regex engine naming and test suit** #10014
  - Since they were created by a maintainer (**@NeverBehave**) and approved by the repository owner (**@DIYgod**), I choose not to consider it as a vulnerability but document it here for warning (refer to the next chapter).

Jun 26, 2022, 15:56 UTC (120.4h)

- The full report was able to be disclosed.

Look back

A ReDoS attack is usually severe but cost-less, "defeating the strong with little effort". Carelessness and the over-trust in the regular expression engine requiring backtracking are the root causes of a ReDoS vulnerability, while the common immediate causes are:

1. A regular expression engine requiring backtracking, and
2. A catastrophic-backtracking-vulnerable regular expression, and
3. A user-supplied malicious strings.

ReDoS vulnerabilities are not very common since there must be a catastrophic-backtracking-vulnerable regular expression, which is usually written by programmers and out of the control of end-users. But in this vulnerability of RSSHub, things are different: both the regular expression and the string can be user-supplied. In the POC, I merely showed how to supply a vulnerable regular expression. So how to understand why the string can also be user-supplied? A fact is that some "routes" (e.g. `/twitter/user/:username`) grab posts from social media, resulting in anyone being able to post malicious strings and request RSSHub to grab them. An experienced attacker may merely construct a vulnerable regular expression specially designed for some strings just like the one I have demonstrated. While an inexperienced attacker can easily use known combinations of vulnerable regular expressions and malicious strings. As a result, everyone who knows what catastrophic backtracking is can probably construct an effective attack.

The fix adopts an alternative regular expression engine (`RE2`) which is backtracking-free. It is quite simple, but effective. It shows a nice example with functionality and security balanced.

However, as mentioned, two commits (refer to the previous chapter) have opened up the attack surface again in certain conditions. Those commits re-empower instance maintainers to switch back to the vulnerable regular expression engine, the vanilla built-in-JavaScript one (`RegExp`). To be responsible, I warn instance maintainers again not to switch back to `RegExp` unless:

- Additional access control is applied, or
- Your instance does not expose to the Internet, or
- You understand what are you doing exactly and do not care about ReDoS attacks.

Revisions

Rev. 1

Initial report.

Rev. 2

Document the chosen fix in the chapter "Possible fix".
Correct some wording and statements.
Add two new chapters: "Timeline (in reality)" and "Look back".

Rev. 3

Minor revision.
Document two commits opening up the attack surface again in certain conditions.

Rev. 4

Fix the statement about `/test/complicated`.

Rev. 5

Correct some mistakes in the chapter "Timeline (in reality)".

Rev. 6

Document the last vulnerable Docker image.

---

1. Most VPS providers take the advantage of a technology called "CPU credits". If a VPS has a high CPU load continuously, it can possibly consume all remaining CPU credits, causing the VPS provider to limit the performance to a fairly low level. Even if the high load ends, it can take at least several hours before there are enough CPU credits to make the performance resume. ↩
2. The attack cost is still too low. ↩
3. Still vulnerable if manually enabled and matching the "condition to be vulnerable". ↩
4. What you are reading. ↩

👍 1 💔 1

Rongronggg9 added the RSS bug label on Jun 26

Rongronggg9 commented on Jun 26 • edited ▾ Contributor Author

简体中文翻译由机器翻译协助完成。

该漏洞已得到修复 (请参阅安全公告)。根据与 **@DIYgod** 的协商，这是漏洞报告的完整披露，在修复提交的 120 小时后披露。

漏洞报告 (RSSHub)

正则表达式拒绝服务攻击 (ReDoS)
用户提供的正则表达式中的灾难性回溯
作者：Rongrong (**@Rongronggg9**)

POC

向仓库所有者报告后，漏洞已修复 (参阅安全公告)。复现漏洞需要 `041cfc3` (含) 之前的版本。`diygod/rsshub:2022-06-20` 是最后一个易受攻击的 Docker 镜像。

1. 按照官方指南以任何方法 (Vercel 或 GAE 除外) 部署 RSSHub 实例。
2. `curl 'http://example.com/test/complicated?filter=%28%5B%5E%3C%3E%27%22%5D%2B%29%2A%22%3C&filterout=%28%5B%5E%3C%3E%27%22%5D%2B%29%2A%22%3C'`

3. 该实例现在对任何请求都没有响应 (拒绝服务)。
4. 监控进程 `node index.js` 的 CPU 使用率。它现在总是占据整个 CPU 核心。这种情况可以持续至少几个小时。

EXP

`example.com` 是一个 RSSHub 实例。
`/test/complicated` 可以是 RSSHub 的几乎任何"路由"，这里是一个随机选择的"测试路由"。
`filter`，`filter_title`，`filter_description`，`filter_author`，`filterout`，`filterout_title`，`filterout_description`，和 `filterout_author` 都是 RSSHub 的所谓参数，可以在 URI 查询字符串中提供。这些参数接受用户提供的正则表达式，并给予无条件信任，然后调用 `String.match()` 以执行正则表达式匹配。所有这些参数都是易受攻击的。
`%28%5B%5E%3C%3E%27%22%5D%2B%29%2A%22%3C` 是正则表达式 `/([^<>'"]+)*"</` 的 URL 编码形式。POC 显示了专为任何 HTML 设计的易受灾难性回溯攻击的正则表达式。但是大多数易受灾难性回溯攻击的正则表达式都有可能能够构建有效的 DoS 攻击 (有关详细信息，请参阅"回顾"一章)。

一旦发生灾难性的回溯，`String.match()` 可能需要数个小时才能完成。由于没有强制的执行超时，ReDoS 攻击可谓"零成本"：攻击一次，宕机数小时。在多核服务器上，ReDoS 攻击可能只会影响 RSSHub 实例本身；然而在单核服务器，尤其是 VPS[1] 上，这可以是一场灾难。

该组件无法禁用。除非应用了额外的访问控制，否则每个 RSSHub 实例"一开箱"就易受攻击。如果强制执行外部看门狗或超时 (例如 Vercel、GAE)，则可以限制每次有效攻击导致的宕机时间。但是，实例必须先终止才能恢复，因此攻击成本仍然太低。

有漏洞的版本

自 Git hash `4671720` 以来的所有版本，这个 commit 是 GitHub 上的 Pull Request #3910，于 2020 年 2 月 9 日 (28.5 个月前) 被合并。
已在 `5c41774` 中修复。

易受攻击的条件

除非应用了额外的访问控制。
如果强制执行外部看门狗或超时，则可以限制宕机时间，但实例仍然容易受到攻击。

漏洞等级

高。

可能的修复

<略>

时间轴

<略>

时间轴 (现实中)

<略>

回顾

ReDoS 攻击通常可以造成严重后果，但成本却很低，“四两拨千斤”。粗心和对需要回溯的正则表达式引擎的过度信任是写出 ReDoS 漏洞的根本原因，而造成攻击的常见的直接原因是：

1. 需要回溯的正则表达式引擎，以及
2. 一个易受灾难性回溯攻击的正则表达式，以及
3. 用户提供的恶意字符串。

ReDoS 漏洞并不常见，因为必须有一个易受灾难性回溯攻击的正则表达式，它通常由程序员编写并且不受最终用户的控制。但在 RSSHub 的这个漏洞中，情况有所不同：正则表达式和字符串都可以由用户提供。在 POC 中，我只是展示了如何提供易受攻击的正则表达式。那么如何理解为什么字符串也可以由用户提供呢？事实上，一些“路由” (例如 `/twitter/user/:username`) 从社交媒体抓取帖子，导致任何人都可以发布恶意字符串并请求 RSSHub 抓取它们。一个有经验的攻击者可能只需要构造一个专门为某些字符串设计的易受攻击的正则表达式，就像我演示的那样。然而没有经验的攻击者可以轻松使用易受攻击的正则表达式和恶意字符串的已知组合。因此，每个知道什么是灾难性回溯的人都可以构造有效的攻击。

该修复采用了一种替代正则表达式引擎 (`RE2`)，它无需使用回溯。这很简单，但也很有效。这展示了一个很好的例子，平衡了功能性和安全性。

然而，两个提交 (#10013, #10014) 在某些情况下再次打开了攻击面。这些提交重新授权实例维护者切换回易受攻击的正则表达式引擎，即 JavaScript 中原生的引擎 (`RegExp`)。为了负责任，我在此警示实例维护者不要切换回 `RegExp`，除非：

- 应用了额外的访问控制，或者
- 您的实例未暴露于互联网，或者
- 您完全了解自己在做什么，并且不关心 ReDoS 攻击。

修订记录

<略>

1. 大多数 VPS 提供商利用一种称为“CPU 积分”的技术。如果 VPS 持续有高 CPU 负载，则可能会消耗所有剩余的 CPU 积分，导致 VPS 提供商将性能限制在相当低的水平。即使高负载结束，也可能需要至少几个小时才能有足够的 CPU 积分来恢复性能。 ↩

TonyRL added security core bug announcement and removed RSS bug labels on Jun 26

TonyRL pinned this issue on Jun 26

DIYgod closed this as completed on Jun 30

---

jahid01711 commented 2 days ago

is there any gap to work as team ?
i found there are a lot of repo in github contaning viruses /

Assignees

No one assigned

Labels

announcement core bug security

Projects

None yet

Milestone

No milestone

Development

No branches or pull requests

4 participants