**ReDoSHunter: A Combined Static and Dynamic Approach for Regular Expression DoS Detection**

**引文信息**

[1] LI Y, CHEN Z, CAO J, 等. ReDoSHunter: A Combined Static and Dynamic Approach for Regular Expression DoS Detection[J]. 19.  
<https://www.usenix.org/conference/usenixsecurity21/presentation/li-yeting>

**作者**

* Yeting Li ——SKLCS, ISCAS UCAS
* Zixuan Chen——SKLCS, ISCAS UCAS
* Jialun Cao——HKUST
* Zhiwu Xu——Shenzhen University
* Qiancheng Peng——SKLCS, ISCAS UCAS
* Haiming Chen ——SKLCS, ISCAS
* Liyuan Chen——Tencent
* Shing-Chi Cheung——HKUST

**Summary**

1. 在本文提出了ReDoSHunter ——一个ReDoS-vulnerable正则表达式检测框架：可以确定多个漏洞的根本原因，并生成attack-triggering字符串。

静态检测中提出了ReDoS漏洞五种模式：NQ、EOD、EOA 、POA 、SLQ

1. 与其他漏洞检测方法比较，ReDoSHunter平衡了静态和动态方法的优势，实现了精确率100%和[召回率](https://so.csdn.net/so/search?q=%E5%8F%AC%E5%9B%9E%E7%8E%87&spm=1001.2101.3001.7020" \t "_blank)100%效果。
2. ReDoSHunter成功识别了所有由ReDoS引起的35个CVEs，其他方法识别率远不如该方法，其中几个CVEs只有ReDoSHunter可以识别。
3. 在流行的26个[开源项目](https://so.csdn.net/so/search?q=%E5%BC%80%E6%BA%90%E9%A1%B9%E7%9B%AE&spm=1001.2101.3001.7020" \t "_blank)中发现了28个新ReDoS漏洞，其他方法效果比较差。

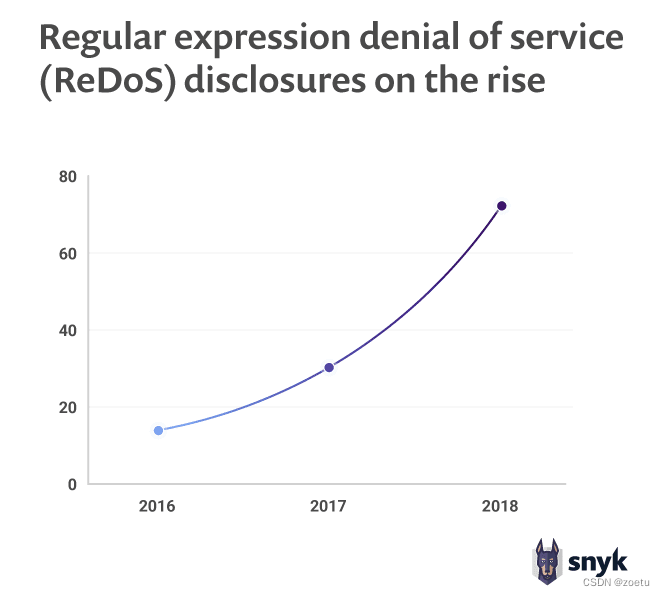
其中26个分配了CVE id，2个由维护者修复

**Part1. 研究背景**

**ReDoS漏洞定义**

如果正则的匹配时间复杂度是超线性的，容易被攻击者利用恶意输入（即攻击串）耗尽服务器的计算资源，造成系统拒绝服务。

**ReDoS漏洞普遍且严重**

* Snyk安全研究团队发现：被披露的ReDoS漏洞越来越多，仅2018年就激增143%；  
  
  + <https://snyk.io/blog/redos-vulnerabilities-in-npm-spikes-by-143-and-xss-continues-to-grow/>
  + <https://snyk.io/blog/a-snyk-peek-into-node-js-and-npms-state-of-open-source-security-report-2019/>

1. 2016年，ReDoS漏洞导致了Stack OverFlow的34分钟全球故障；

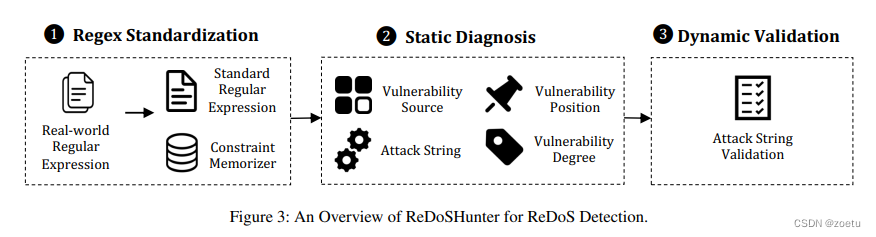
<https://stackstatus.net/post/147710624694/outage-postmortem-july-20-2016>

1. 2019年，因为ReDoS漏洞Cloudfare服务中断了27分钟。

<https://blog.cloudflare.com/details-of-the-cloudflare-outage-on-july-2-2019/>

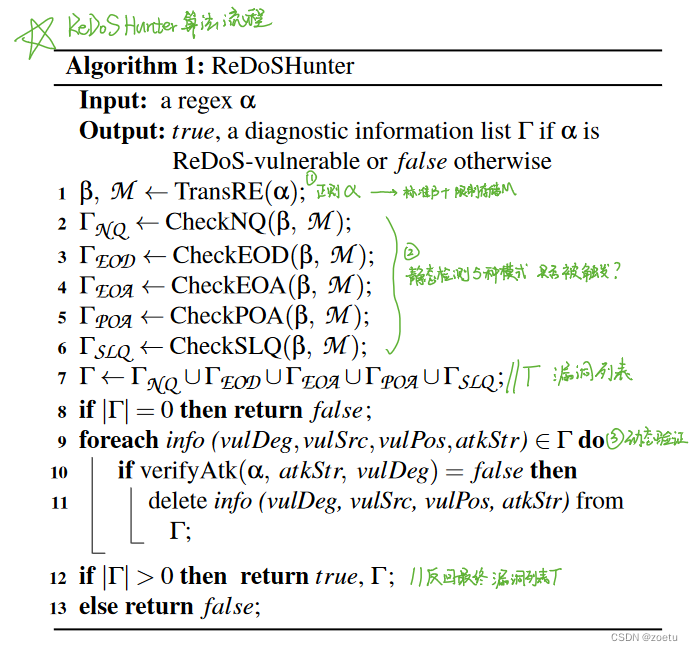
**Part2. 方法**

**ReDoSHunter算法概述**

设计了动静态结合的ReDoS检测算法，不仅可以有效地检测ReDoS漏洞，还可以诊断漏洞原因，分析漏洞严重程序，追踪漏洞位置并生成出发攻击的字符串。  
  
**关键思想：**

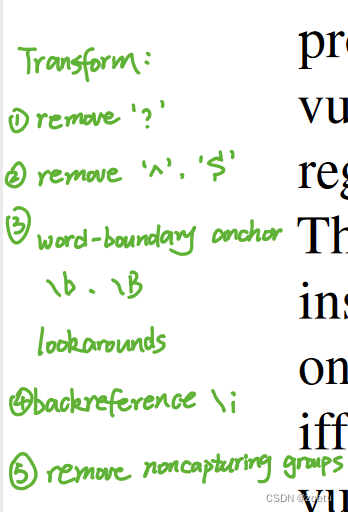
1. 正则标准化：支持扩展特性
2. 静态诊断：检测更多漏洞
3. 动态验证：检测无误报

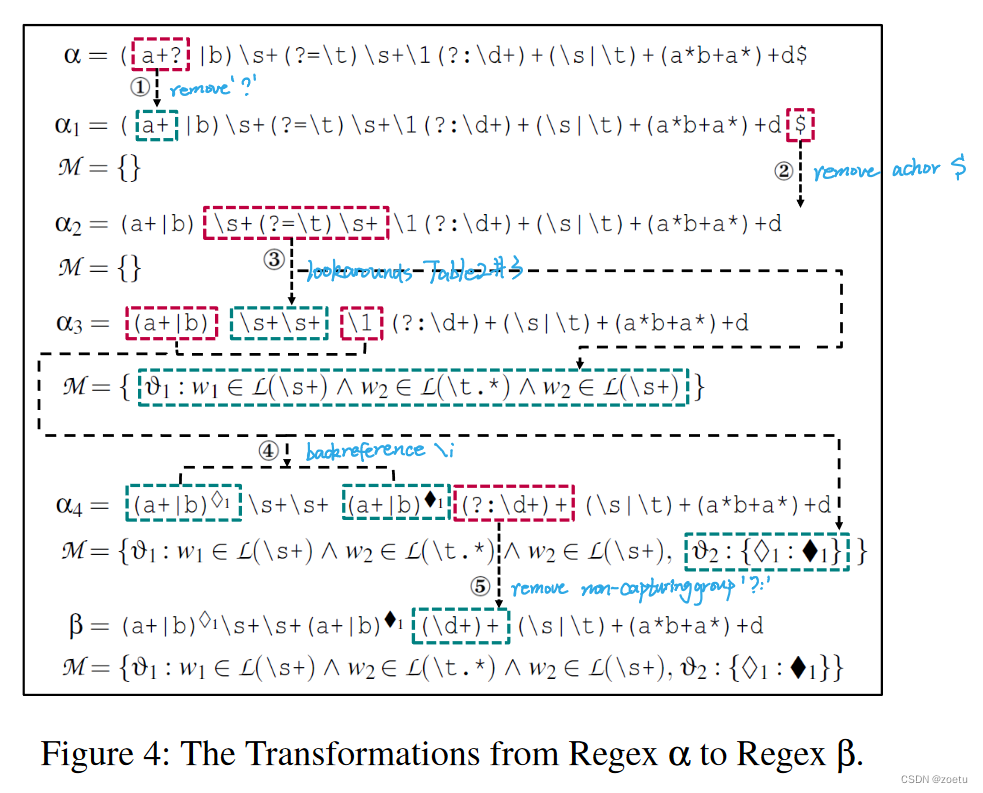
**算法伪代码**



**1. 正则标准化**

* 删除或者转变扩展
* 某些扩展增加额外限制

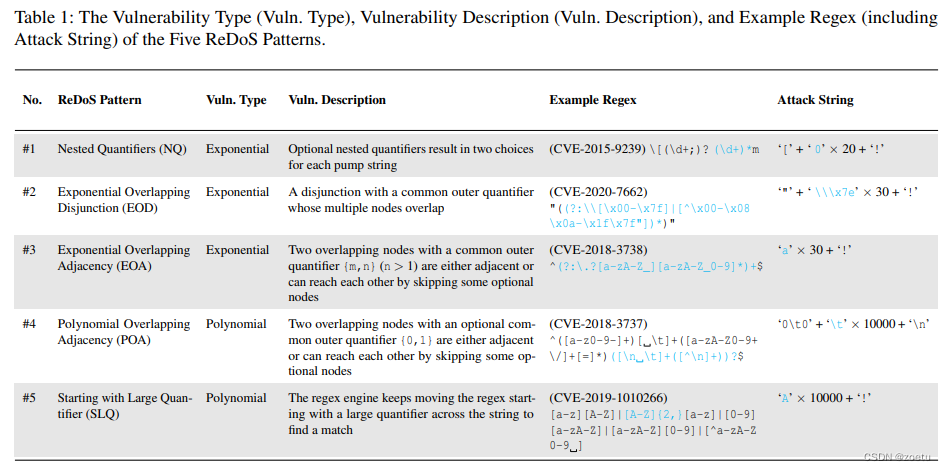
扩展转变顺序：（顺序很重要）  
  
举个例子：

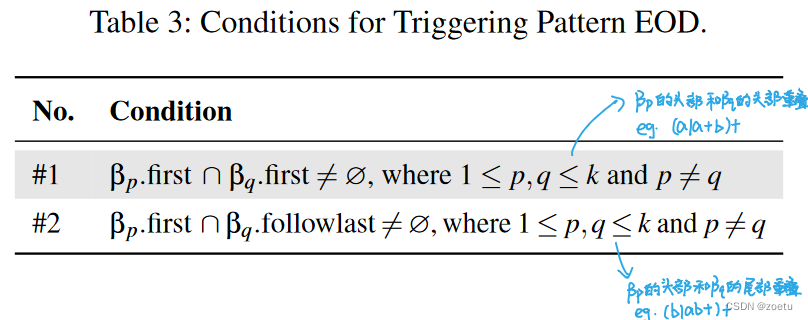


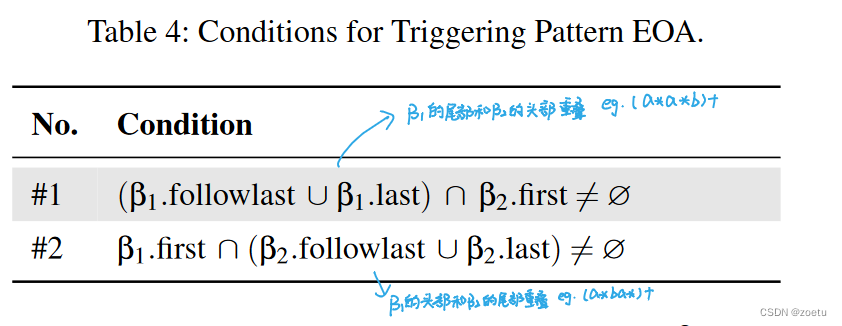
**2. 静态诊断**

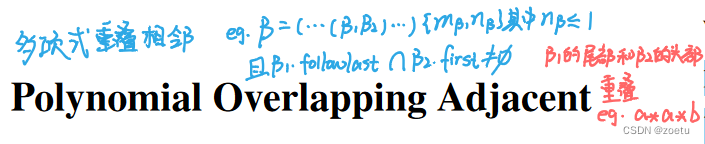
本文从大量调查和分析总结出了5种正则表达式漏洞模式：

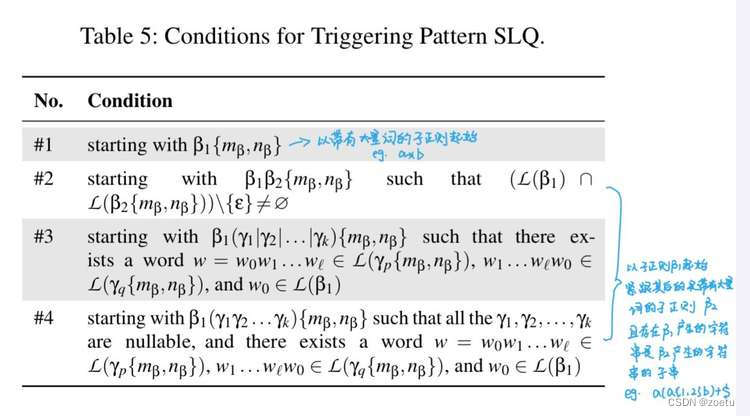
**5种ReDoS类型**

  
a. 嵌套量词（NQ）模式：具有嵌套量词的正则表达式。例如(\d+)+。

b. 指数重叠析取（EOD）模式：β = (…(β1|β2|…|βk)…){mβ,nβ} 其中nβ>1，且满足以下两个条件之一。  


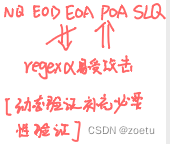
c. 指数重叠相邻（EOA）模式：β=(…(β1β2)…){mβ,nβ}，其中nβ> 1,满足以下两个条件之一。  


d. 多项式重叠相邻（POA）模式：：β=(…(β1β2)…){mβ,nβ}，其中nβ <= 1，且满足条件β1.followlast ∩ β2.first ≠ \not= ​= ∅.。  


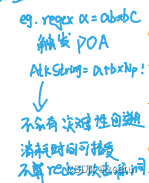
e. 从大量词开始（SLQ）模式：有四种可能的触发条件，其中 n β > n m i n n\_β>n\_{min} nβ​>nmin​。  


**3. 动态验证**

**动态验证解决两个问题：**  
a. 静态诊断可能产生误报

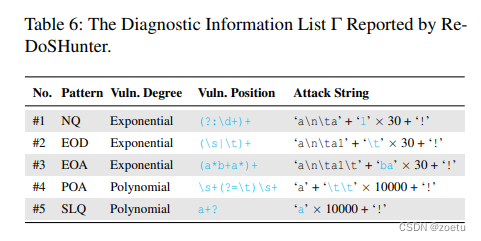
静态诊断中的五种ReDoS模式是判断regex易受攻击的必要不充分条件。  


b. 经过第一步转换的正则表达式更容易产生漏洞，动态验证保证不发生误报。

如果没有动态验证，误报就会发生。  
比如：  


**怎么动态验证？**

验证原正则表达式匹配攻击串的时间是否超过阈值。

如果时间超过阈值，则通过动态验证即漏洞属实，返回诊断信息。各类漏洞信息如下所示：  


**Part3. 实验评估**

通过三个研究问题（Research Question）评估本文算法：

**实验环境**

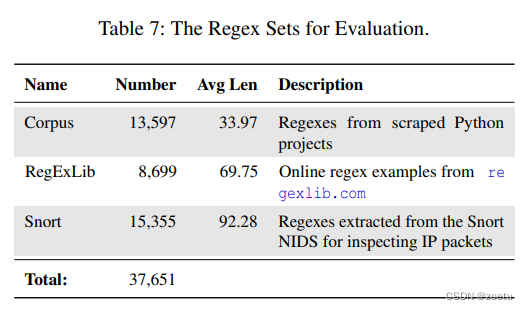
* 操作系统：Windows 10
* 硬件配置：2.20 GHz Intel Xeon(R) Silver processor and 128G RAM
* 参数设置： N E = 100 ， N P = 30000 , T E = 0.1 s ， T P = 1 s , n m i n = 100 N\_E=100，N\_P=30000,T\_E=0.1s，T\_P=1s,n\_{min}=100 NE​=100，NP​=30000,TE​=0.1s，TP​=1s,nmin​=100

N E 代 表 指 数 模 式 串 重 复 次 数 ， 如 N Q 、 E O D 、 E O A ； N\_E代表指数模式串重复次数，如NQ、EOD、EOA； NE​代表指数模式串重复次数，如NQ、EOD、EOA；  
N P 代 表 多 项 式 模 式 重 复 次 数 N\_P代表多项式模式重复次数 NP​代表多项式模式重复次数  
T E 代 表 匹 配 指 数 攻 击 串 匹 配 时 间 阈 值 T\_E代表匹配指数攻击串匹配时间阈值 TE​代表匹配指数攻击串匹配时间阈值  
T P 代 表 多 项 式 攻 击 串 匹 配 时 间 阈 值 T\_P代表多项式攻击串匹配时间阈值 TP​代表多项式攻击串匹配时间阈值  
n m i n 代 表 最 少 重 复 次 数 n\_{min}代表最少重复次数 nmin​代表最少重复次数

**实验数据**

可以去github库中获取相关内容

**大规模正则集：**

共37651条正则表达式  


* Corpus：发表在ISSTA 2016的论文Exploring Regular Expression Usage and Context in Python数据集
* RegExLib：https://regexlib.com/
* Snort：https://www.snort.org/

**已知ReDoS漏洞集：**  
CVEs中与ReDoS相关的35个漏洞

**检测未知漏洞项目：**  
26个GitHub/npm/PyPI中的高下载量项目

**BaseLines**

*最近用来做ReDoS检测的SOTA：*

1. RXXR2
2. Rexploiter
3. NFAA
4. safe-regex
5. Regexploit
6. SDL
7. ReScue

其中：

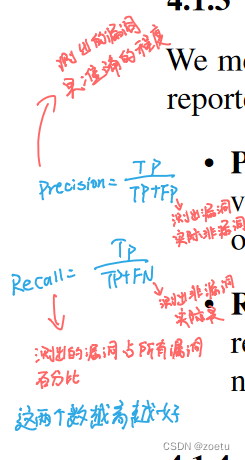
方法1-5是静态分析方法  
方法6-7是动态分析方法

**RQ1: ReDoSHunter在大规模正则表达式集上的有效性和效率**

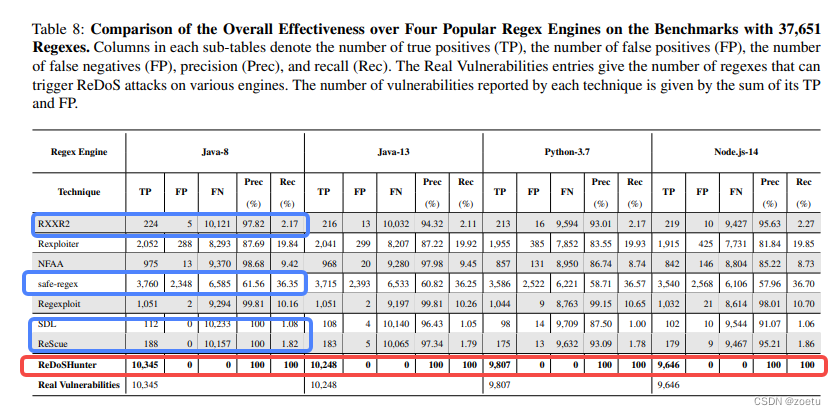
**评估指标**

**有效性：**

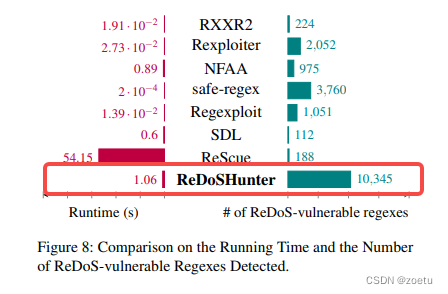
* Precision //正确率
* Recall //召回率  
  **效率：**
* 检测正则表达式速度
* 检测结果覆盖率

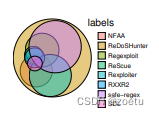


**有效性结果**

  
在Java-8引擎下，静态分析方法正确率最低的方法safe-regex只有61.56%，正确率最高97.82%的方法RXXR2的召回率2.17%；  
动态分析方法SDL和ReScue牺牲了召回率（仅~1%）得到100%正确率。  
**ReDoSHunter在所有引擎上的precision和recall都达到100%** ，解决了其他方法没办法保证召回率和精确率都高的问题。

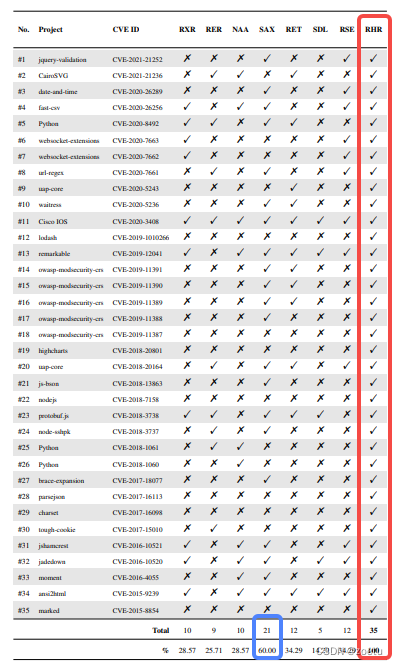
**效率结果**

  
ReDoSHunter检测每条正则表达式的速度为1s左右，其检测的ReDoS漏洞比其他方法多。

  
ReDoSHunter检测结果可以把其方法检测到的结果覆盖。

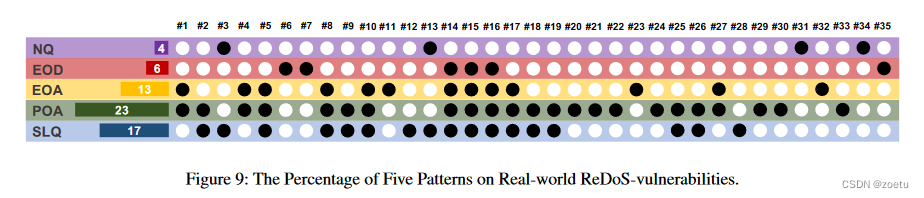
**RQ2: ReDoSHunter识别已知漏洞有效性**

**评估指标：已有35个CVEs识别率**

**结果：**  


ReDoSHunter可以检测所有的已知的漏洞，其他方法中最好的是safe-regex可以识别60%的CVEs。

从上面的实验结果看出，CVEs #12,#18, #19, #22, #28, #29, #35只有ReDoSHunter可以识别出来。

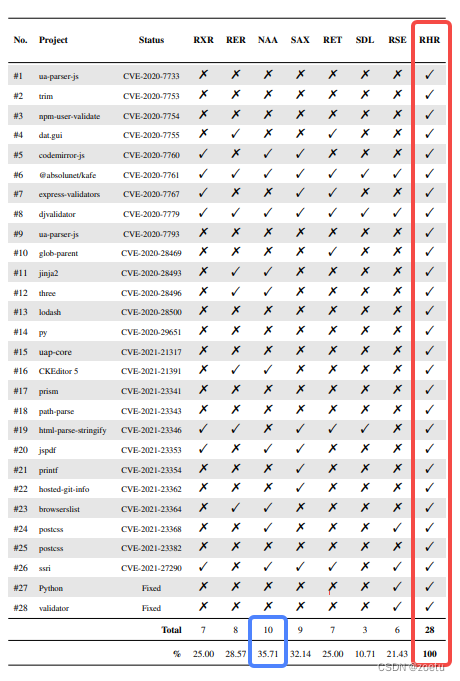
**因此，ReDoSHunter在真实场景中检测漏洞是一种先进的方法。**  


本文所提出的五种静态正则漏洞模式在CVEs中是有效的，可以看到能够成功识别出来各漏洞类型。

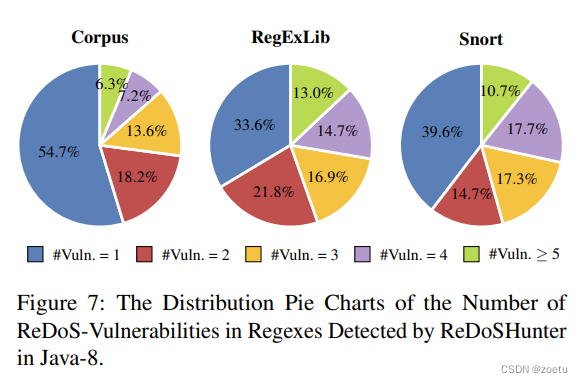
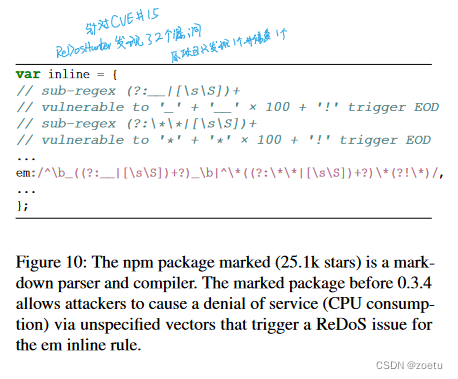
**RQ3: ReDoSHunter检测未知漏洞有效性**

**评估指标：在26个项目中发现的未知漏洞数**

**结果：**

  
ReDoSHunter发现了28个未知漏洞，其他方法最多只检测出10个。

**检测一条正则表达式的多个漏洞**

**真实场景：**  
对所使用的三个正则表达式数据集的漏洞分析，可以看出来一条正则表达式中有多个漏洞是比较常见的。  
  
**ReDoSHunter实验**  


项目地址：

* <https://github.com/markedjs/marked/issues/497>
* <https://github.com/markedjs/marked/commit/a37bd643f05bf95ff18cafa2b06e7d741d2e2157>

对于CVEs#35，其漏洞发现者只发现了一个漏洞，但是ReDoSHunter检测出来两个EOD漏洞。

**ReDoSHunter是现有方法中唯一可以做到检测一条正则中多个漏洞的方法。**

**Part4. 讨论**

1. 支持更多扩展

在本文中可以支持大多数常用扩展，但是一些不常用的比如条件语句ReDoSHunter还不支持。  
*[？ 为什么不支持，难度在哪呢（待考虑）]*

例如，正则表达式的条件语句 ( r 1 ) ? ( ? ( 1 ) r 2 ∣ r 3 ) (r1) ? (? (1) r2 | r3) (r1)?(?(1)r2∣r3)，它可以转化为ReDoSHunter可以处理的一个over-approximate条件的statement-free正则表达式和一些外部约束。

1. 支持更多的字符

目前,ReDoSHunter支持常见的字符包括从unicode字符U+0000 -U+FFFF,可以覆盖用于实践的大部分字符。  
对于字符超出这个范围的，ReDoSHunter可能发现不了。这种限制也可以通过一个适当的预处理处理。

**Part5. 相关工作**

便于跟进工作

**1. BaseLines**

| **-** | **分类** | **简述原理** | **优缺点** | **出处** | **时间** |
| --- | --- | --- | --- | --- | --- |
| RXXR2 | 静态分析方法 | RXXR2将给定的正则表达式转换成power DFA（论文提出），并在power DFA的基础上搜索攻击字符串。 | RXXR2不支持大多数正则表达式扩展（例如，预查和反向引用）；无法检测带有多项式的ReDoS漏洞。 | [RXXR](https://link.springer.com/chapter/10.1007/978-3-642-38631-2_11) ，[RXXR2](https://arxiv.org/abs/1405.7058) | 2013、2014 |
| Rexploiter | 静态分析方法 | Rexploiter结合了NFA的复杂性分析和可清除的漏洞分析来检测易受ReDoS攻击的正则表达式。 | 虽然该方法提供了一个额外的功能(例如，排除用户输入不受控制的正则表达式)，但它不支持大多数扩展(例如，预查、反向引用和非捕获组)。 | [Rexploiter](https://link.springer.com/chapter/10.1007/978-3-662-54580-5_1) | 2017 |
| NFAA | 静态分析方法 | – | NFAA可以支持查找和非捕获组等扩展，但它不支持反向引用。 | [NFAA](https://link.springer.com/chapter/10.1007/978-3-319-40946-7_27) | 2016 |
| safe-regex | 静态分析方法 | saferegex工具通过检测NQ模式是否被触发，或者Kleene-Star数量是否大于预设阈值来进行检测。 | 虽然这种模式串匹配方法运行效率很高，但是很多ReDoS模式串超出了它的能力范围。 | [safe-regex](https://dl.acm.org/doi/10.1145/3236024.3236027) | 2018 |
| Regexploit | 静态分析方法 | - | - | <https://github.com/doyensec/regexploit> | 2021 |
| SDL | 动态分析方法 | SDL通过测试正则表达式与一系列随机生成的字符串的匹配时间来检测ReDoS。 | SDL不支持大多数扩展(例如\b和\B、贪婪量词、预查、反向引用等)。 | [https://doces.microsoft.com/en-us/archive/msdn-magazine/2010/may/security-briefs-regular-expression-denial-of-service-attacks-and-defenses](https://docs.microsoft.com/en-us/archive/msdn-magazine/2010/may/security-briefs-regular-expression-denial-of-service-attacks-and-defenses) | 2010 |
| ReScue | 动态分析方法 | Rescue生成用于搜索耗时的字符串。 | 由于字符串搜索空间巨大，只能识别指数级或更高多项式级的redos漏洞，无法检测更低多项式级的redos漏洞或深藏较深的redos漏洞； 遗传搜索的有效性也受到初始化的影响，使得每次搜索的结果都不稳定。 | <https://dl.acm.org/doi/10.1145/3238147.3238159> | 2018 |

**2. ReDoS检测**

**静态分析方法**

属于静态分析的方法主要是通过将正则表达式转换为其自定义的模型，并从模型中静态地识别易受redos攻击的结构来检测redos漏洞。

* RXXR2
* Rexploiter
* NFAA
* safe-regex
* Regexploit

**动态分析方法**

基于动态的方法在运行时检测redos漏洞，与静态分析相比，通常具有较高的精度。大多数动态分析工具使用动态模糊，它不断地用实际的正则表达式引擎搜索耗时的字符串，并从中推断正则表达式的最坏情况下的时间复杂度。

* SDL
* ReScue

这些基于动态的方法输出一个随机的攻击字符串，不能提供任何对redos漏洞根源的洞察。

**3. ReDoS预防与缓解**

各种技术已经被提出通过等效/近似正则表达式变换或正则表达式匹配加速来防止或减轻ReDoS攻击。

**等效/近似正则表达式变换**

这一系列研究试图找到等效/近似的不受redos攻击的正则表达式来取代易受redos攻击的正则表达式。

* V an der Merwe et al.和Cody-Kenny et al.[45]致力于寻找等价的不受redos攻击的正则表达式来替代原来的正则表达式。然而，它们在实践中使用的完全等价性太强了，这限制了它们在实际应用中的部署。

[45] Brink van der Merwe, Nicolaas Weideman, and Martin Berglund. Turning Evil Regexes Harmless. In Proceedings of the South African Institute of Computer Scientists and Information Technologists, SAICSIT 2017, Thaba Nchu, South Africa, September 26-28, 2017, pages 38:1– 38:10, 2017.

* Chida和Terauchi[10]和Li等人[24]通过采用实例编程算法推导反redos正则表达式来解决这个问题。然而，由他们推导出的antiReDoS正则表达式的质量在很大程度上取决于用户提供的示例的质量。

[10] Nariyoshi Chida and Tachio Terauchi. Automatic Repair of Vulnerable Regular Expressions. CoRR, abs/2010.12450, 2020.

**正则表达式匹配加速**

* ReDoS攻击也可以通过regex匹配加速来缓解，regex匹配加速是一些特殊情况下的替代解决方案，如并行算法[25]，基于gpu的算法[50]，状态合并算法[2]，解析表达式语法(PEGs)[17,19,29]，计数自变量匹配算法[44]，基于记忆的优化[16]和递归限制/回溯限制/时间限制[30,32,34]。

[25] Cheng-Hung Lin, Chen-Hsiung Liu, and Shih-Chieh Chang. Accelerating Regular Expression Matching Using Hierarchical Parallel Machines On GPU. In Proceedings of the Global Communications Conference, GLOBECOM 2011, 5-9 December 2011, Houston, Texas, USA, pages 1–5. IEEE, 2011.  
[50] Xiaodong Yu and Michela Becchi. GPU Acceleration Of Regular Expression Matching For Large Datasets: Exploring The Implementation Space. In Hubertus Franke, Alexander Heinecke, Krishna V. Palem, and Eli Upfal, editors, Computing Frontiers Conference, CF’13, Ischia, Italy, May 14 - 16, 2013, pages 18:1–18:10. ACM, 2013.  
[2] Michela Becchi and Srihari Cadambi. Memory-Efficient Regular Expression Search Using State Merging. In INFOCOM 2007. 26th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 6-12 May 2007, Anchorage, Alaska, USA, pages 1064–1072. IEEE, 2007.  
[17] Bryan Ford. Parsing Expression Grammars: A Recognition-Based Syntactic Foundation. In Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, Venice, Italy, January 14-16, 2004, pages 111–122, 2004.  
[19] IBM. Rosie Pattern Language (RPL), 2020. https: //rosie-lang.org/.  
[29] Sérgio Medeiros, Fabio Mascarenhas, and Roberto Ierusalimschy. From Regexes to Parsing Expression Grammars. Sci. Comput. Program., 93:3–18, 2014.  
[44] Lenka Turonová, Lukás Holík, Ondrej Lengál, Olli Saarikivi, Margus Veanes, and Tomás Vojnar. Regex Matching with Counting-Set Automata. Proc. ACM Program. Lang., 4(OOPSLA):218:1–218:30, 2020.  
[16] James C. Davis, Francisco Servant, and Dongyoon Lee. Using Selective Memoization to Defeat Regular Expression Denial of Service (ReDoS). In 2021 IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, May 23-27, 2021, page To appear, 2021.  
[30] Microsoft. Regex class - C#, 2020. https://docs.m icrosoft.com/en-us/dotnet/api/system.text .regularexpressions.regex?view=net-5.0.  
[32] PCRE. PCRE - Perl Compatible Regular Expressions, 2020. https://pcre.org/.  
[34] PHP. PHP: preg\_match - Manual, 2020. https://www. php.net/manual/en/function.preg-match.php.

* 这些工作可以缓解ReDoS漏洞问题，但它们不能解决ReDoS易受攻击的正则表达式本身，它们仍然受到ReDoS攻击。