JavaScript Web服务器ReDoS漏洞分析

angel010 / 2018-09-10 15:57:10 / 浏览数 3523 安全技术 漏洞分析 顶(1) 踩(0)

本文摘自论文《Freezing the Web: A Study of ReDoS Vulnerabilities in JavaScript-based Web Servers》，发表于27届USENIX Security Symposium（2018）。

正则表达式(regular expression)就是用一个"字符串"来描述一个特征,然后去验证另一个"字符串"是否符合这个特征，是各类软件中广泛应用。但正则表达式有一缺点，就是很容易出错，而这可

正则表达式的一个安全方面的考虑却常常被忽略——■■，即字符串匹配正则表达式的时间。一般来说，匹配正则表达式可能需要几分钟到几个小时。比如，30个a匹配正则JavaScript平台上需要15秒钟时间，而匹配35个a需要8分钟时间。如果服务器应用时存在性能问题，那么攻击者就可以利用该漏洞构造一起很难匹配的输入，发起ReDoS（expression denial of service）攻击。

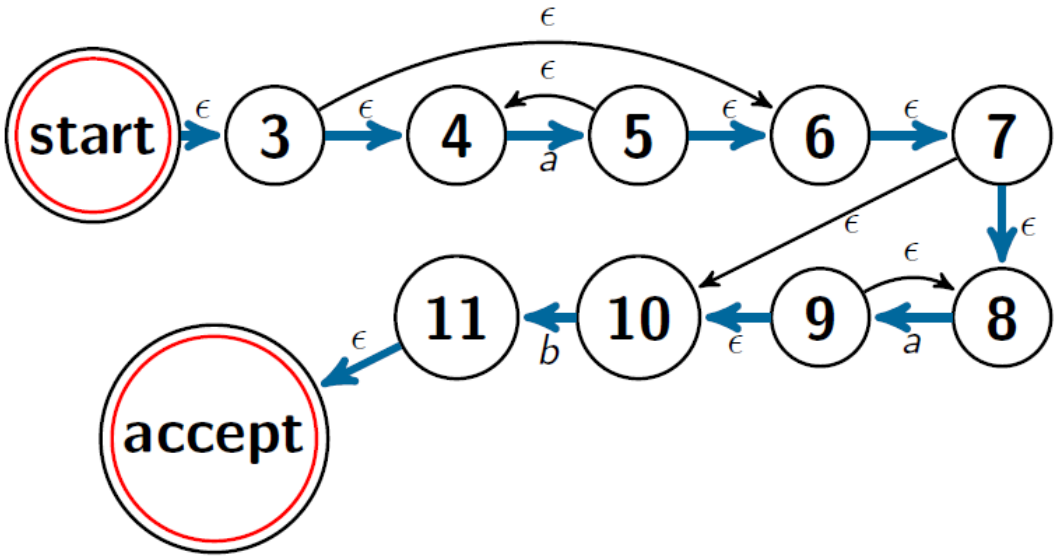
ReDoS攻击

ReDoS攻击是一种算法复杂性攻击，利用了字符串匹配正则表达式的最坏情况。

回溯算法并不是正则匹配中用的最多的，但因其容易实现是使用最广泛的一种匹配算法。下面是一些正则表达式和输入，算法需要进行多次回溯操作。

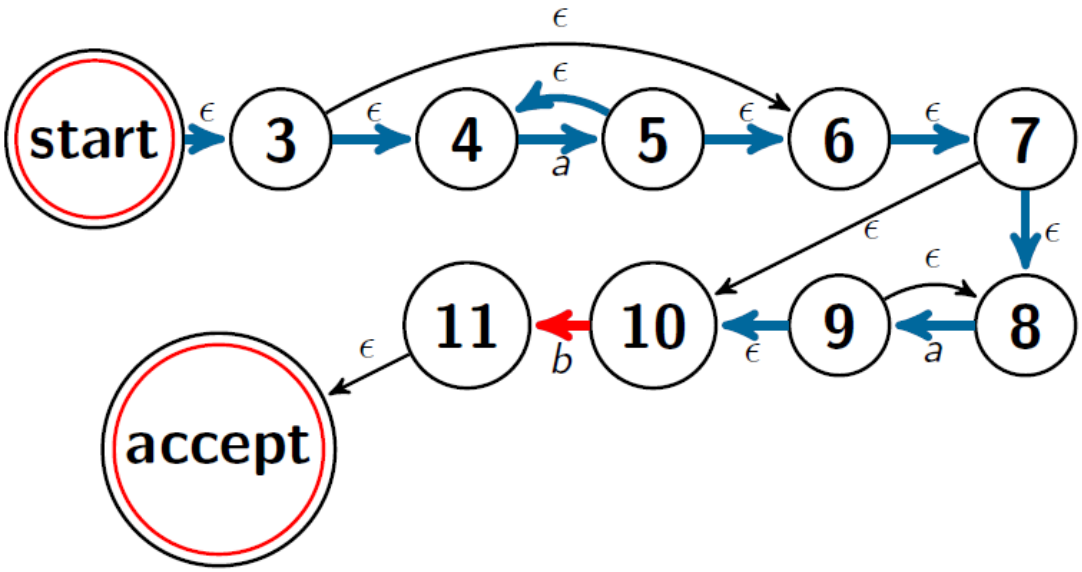
回溯算法匹配

```
var regEx = /^a*a*b$/;
```



input: "aab"

```
var regEx = /^a*a*b$/;
```



input: "aaaaaaaaaaaaaaaaaaa"

ReDoS攻击

ReDOS攻击就利用了这些pathological cases。以正则表达式/^a*a*b$/为例，如图2所示，输入字符串为aaa。每个字符a都可以用两次传递来匹配，4 ！5 and 8 ！9。在每一步，算法需要决定采取了哪些过渡。最后，因为输入字符串中没有字母b，所以算法在状态11会失败。但在做出输入与模式不匹配的结论前，算法会尝试所有可能

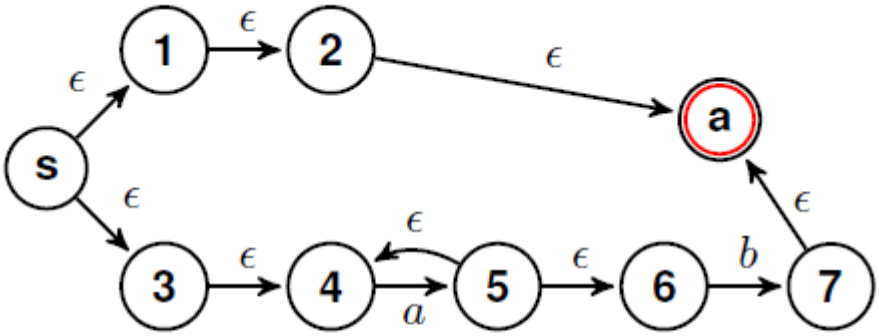


Fig. 1: Automaton for the regular expression /^(a+b)?$/. s is the starting state and a is the accepting state.

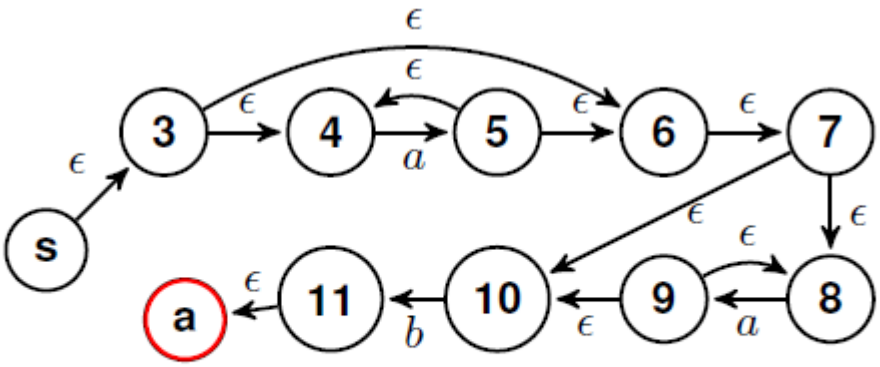
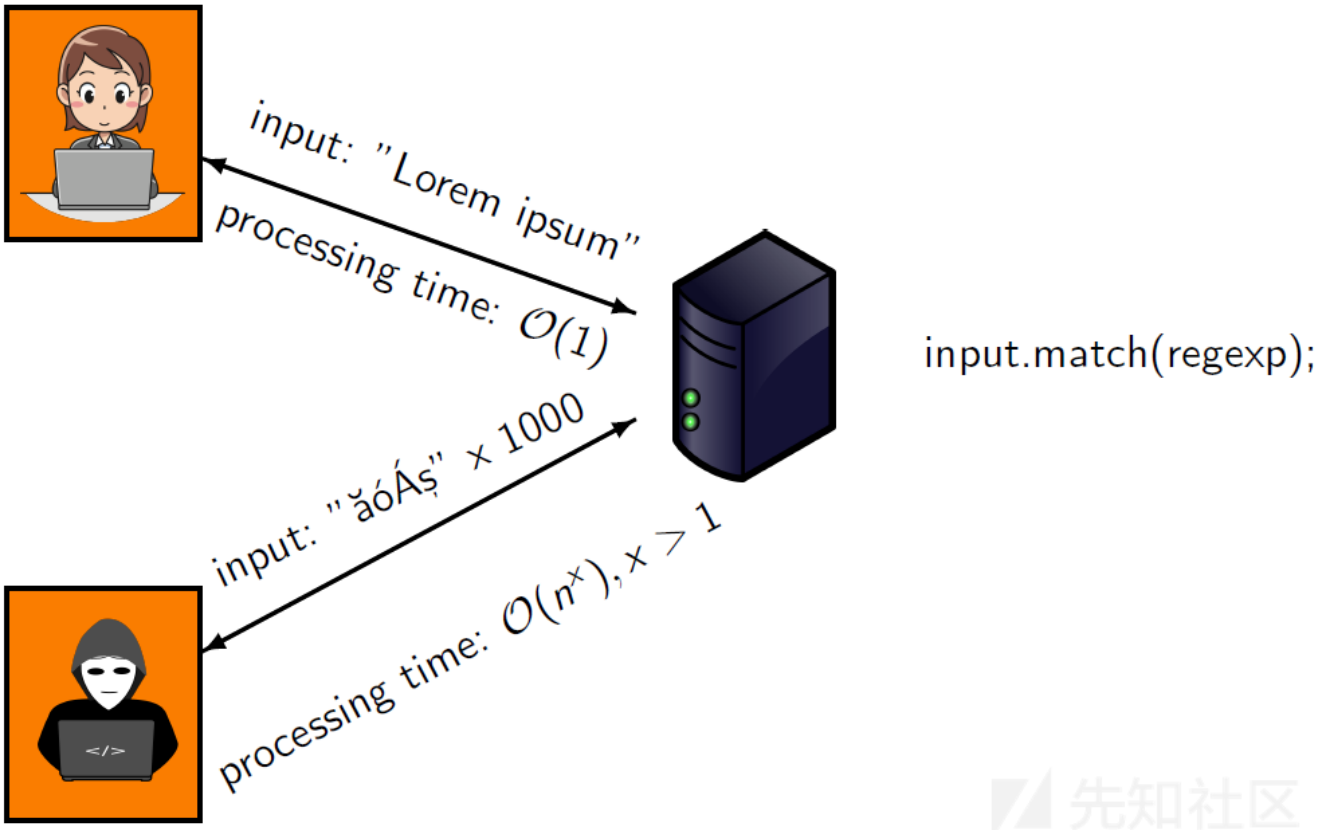


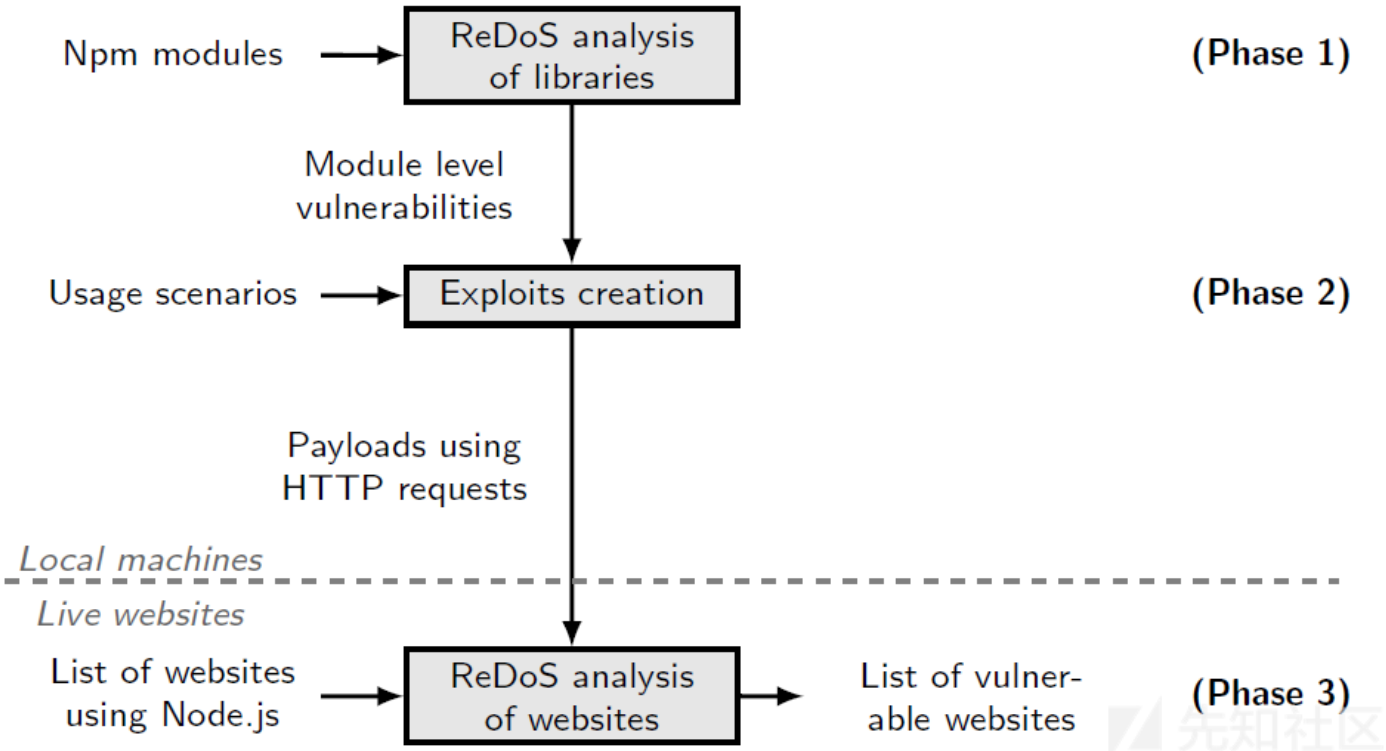
Fig. 2: Automaton for the regular expression /^a⋆a⋆b$/. s is the starting state and a is the accepting state.



Server-side JavaScript

JavaScript是最流行的编程语言之一，但主要用于客户端任务。而Node.js的广泛应用使JavaScript开始应用到服务端。但一个主循环的计算请求会减慢所有的入请求，比如

使用的方法



npm分析

研究人员认为如果正则表达式的输入线性增长，匹配时间超线性增长，那么这个正则表达式就是有漏洞的。研究人员发现主流的npm模块中存在ReDoS漏洞。

- 首先，下载主流的模块并用JS代码的AST来提取正则表达式；
- 然后，通过查询数据库来找出有漏洞的特定模式；

创建漏洞利用

基于npm模块的ReDoS
漏洞，研究人员创建了利用这些模块的攻击web服务器的漏洞利用。主要思想是假设服务端web应用可能使用该模块。最后，建立一个快速安装并实现使用该模块的web应用

有漏洞的正则表达式：
例1: content

```
/^([^\/]+\/[^\ s ;]+) (?:(?:\ s *;\s* boundary =(?: " ([^"
]+)" |([^; "]+) )) |(?:\ s*;\ s *[^=]+=(?:(?: " (?:[^ "
]+)") |(?:[^; "]+) ))*$/i
```

例2: ua-parser-js

```
/ip[ honead ]+(.* os\s([\w]+) *\ slike \ smac |;\ sopera )/
```

ID	Vulnerable regular expression
1	`/(?:charset|encoding)\s*=\s*['"]? *([\w\-]+)/i`
2	`/^([^\/]+\/[^\s;]+)(?:(?:\s*;\s*boundary=(?:"([^"]+)"|([^;"]+)))|(?:\s*;\s*[^=]+=(?:(?:"(?:[^"]+)")|(?:[^;"]+))))*$/i`
3	`/ *, */`
4	`/ *, */`
5	`new RegExp("Dell.*Streak|Dell.*Aero|Dell.*Venue|DELL.*Venue Pro|Dell Flash|Dell Smoke|Dell Mini 3iX|XCD28|XCD35|\\b001DL\\b|\\b101DL\\b|\\bGS01\\b")`
6	`/^ +| +$/g`
7	`/ip[honead]+(?:.*os\s([\w]+)*\slike\smac|;\sopera)/`
8	`/((?:[A-z0-9]+|[A-z\-]+ ?)?(?: the)?(?:[Ss][Pp][Ii][Dd][Ee][Rr]|[Ss]crape|[A-Za-z0-9-]*(?:[^C][^Uu])[Bb]ot|[Cc][Rr][Aa][Ww][Ll])[A-z0-9]*)(?:(?:[ \/]| v)(\d+)(?:\.(\d+)(?:\.(\d+))?)?)?/`

Fig. 8: Vulnerable regular expressions.

HTTP级的Payload创建

对每个payload，创建应用场景：

```
var MobileDetect = require ("mobile - detect ");
var headers = req . headers ["user - agent "];
var md = new MobileDetect ( headers );
md. phone ();
```

分析网站ReDoS漏洞影响

研究人员发现通过模块接口，有漏洞的正则表达式可以利用这些模块发现ReDoS攻击。每个漏洞都至少出现在一个包中，而不同包的依赖和下载数是不同的。

Module	Version	Number of dependencies	Downloads in July 2017
debug	2.6.8	16,055	54,885,335
lodash	4.17.4	49,305	44,147,504
mime	1.3.6	2,798	22,314,018
ajv	5.2.2	758	17,542,357
tough-cookie	2.3.2	302	15,981,922
fresh	0.5.0	197	14,151,270
moment	2.18.1	14,421	10,102,601
forwarded	0.1.0	31	9,883,630
underscore.string	3.3.4	2,486	7,277,966
ua-parser-js	0.7.14	225	5,332,979
parsejson	0.0.3	19	4,897,928
useragent	2.2.1	191	3,515,292
no-case	2.3.1	18	3,321,043
marked	0.3.6	2,624	3,012,792
content-type-parser	1.0.1	8	2,337,147
platform	1.3.4	128	757,174
timespan	2.3.0	34	523,290
string	3.3.3	911	421,700
content	3.0.5	9	316,083
slug	0.9.1	499	151,004
htmlparser	1.7.7	178	138,563
charset	1.0.0	36	112,001
mobile-detect	1.3.6	101	107,672
ismobilejs	0.4.1	50	44,246
dns-sync	0.1.3	7	10,599

Fig. 6: Modules with at least one previously unknown vulnerability. The emphasized modules are used to analyze real websites because we found an exploit

针对不同的漏洞模块和header的使用场景，研究人员给出了漏洞利用。

ID	Module	Header	Usage scenario	JavaScript example
1	charset	Content-Type	The website uses this package to parse the content type of every request.	`require("charset")(req.headers);`
2	content	Content-Type	The website uses this package to parse the content type of every request.	`var content = require("content");` `content.type(req.headers["content-type"]);`
3	fresh	If-None-Match	The website uses `express`, which by default uses this package to check the freshness of every request.	`var fresh = require("fresh");` `fresh(req.headers);`
4	forwarded	X-Forwarded-For	The website uses `express` and the "trust proxy" option is set. This package is then used to check which proxies a request came through.	`var forwarded = require("forwarded");` `var addrs = forwarded(req);`

5	mobile-detect	User-Agent	The website uses this package to get information about the requester.	`var MobileDetect = require("mobile-detect");`
				`var md = new MobileDetect(req.headers["user-agent"]);`
				`md.phone();`

6	platform	User-Agent	The website uses this package to get information about the requester.	`var platform = require("platform");` `var agent = platform.parse(req.headers["user-agent"]);`
7	ua-parser-js	User-Agent	The website uses this package to get information about the requester.	`var useragent = require("ua-parser-js");` `var agent = useragent.parse(req.headers["user-agent"]);`
8	useragent	User-Agent	The website uses this package to get information about the requester.	`var useragent = require("useragent");` `var agent = useragent.parse(req.headers["user-agent"]);`

Fig. 7: Usage scenarios for vulnerable modules and the headers we hypothesize the modules to process.

预防措施

首先，为了限制通过HTTP header传播的payload的效果，header的大小应该做出限制。这种方法可以缓解一些潜在攻击的效果，但对于与HTTP header相关的漏洞的效果是有限的。因为从网络接收的输入也可以被利用来进行攻击。

第二种预防机制是使用更复杂的正则表达式引擎，这些引擎应该确保线性匹配的时间。问题是这些引擎并不支持高级正则表达式特征，比如先行断言(lookahead)和后行断言

结论

本文分析了基于JavaScript的web服务器的ReDoS漏洞，并说明该漏洞是影响主流网站的重要问题。研究人员在实验中共发现8个漏洞，影响超过339个主流网站。攻击者可

论文下载地址：https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-staicu.pdf
研究PPT下载地址：https://www.usenix.org/conference/usenixsecurity18/presentation/staicu

目录