

取证

介绍

何为取证？

电子取证是指利用计算机软硬件技术，以符合法律规范的方式对计算机入侵、破坏、欺诈、攻击等犯罪行为进行证据获取、保存、分析和出示的过程。从技术方面看，计算机犯罪取证是一个对受侵计算机系统进行扫描和破解，对入侵事件进行重建的过程。具体而言，是指把计算机看作犯罪现场，运用先进的辨析技术，对计算机犯罪行为进行解剖，搜寻罪犯及其犯罪证据。

接下来我们从常用的取证工具入手，来讲解取证的常见内容。

之所以从工具入手，是因为取证过程中如果不依靠现有的强大工具，就需要取证的人自己对相关数据文件的数据存储格式有详细的了解乃至是掌握。取证工具的本质其实就是对已知存储格式的数据从格式上进行自动化地解析使得使用者可以轻松提取相应的数据资料。如果以手工的方式的话则需要大篇幅的内容来讲解各种诸如硬盘系统数据文件、内存镜像数据文件一类的相关数据存储的格式。

内存取证——Volatility

Volatility是开源的Windows, Linux, MaC, Android的内存取证分析工具，由python编写成，命令行操作，支持各种操作系统。

并且该工具属于框架类工具。即其本身除却官方自己实现的功能插件外，用户可以根据自己需要来制作自定义插件。

通过-h参数可以列举出本地工具已经集成了的功能插件以及相关描述。

```
root@xibai-kali:~/桌面/study/2020/hxb/misc# volatility -f 1.raw -h
Volatility Foundation Volatility Framework 2.6
Usage: Volatility - A memory forensics analysis platform.

Options: 收藏
-h, --help      list all available options and their default values.
                Default values may be set in the configuration file
                (/etc/volatilityrc)
--conf-file=/root/.volatilityrc
                User based configuration file
-d, --debug     Debug volatility
--plugins=PLUGINS Additional plugin directories to use (colon separated)
--info          Print information about all registered objects
--cache-directory=/root/.cache/volatility
                Directory where cache files are stored
--cache         Use caching
--tz=TZ         Sets the (Olson) timezone for displaying timestamps
                using pytz (if installed) or tzset
-f FILENAME, --filename=FILENAME
                Filename to use when opening an image
--profile=WinXPSP2x86
                Name of the profile to load (use --info to see a list
                of supported profiles)
+ 其他位置      session_0.           session_0.
                Default.png
                session_0.
                Service-0
                Default
                Default

mem.raw
ML

```

这里介绍一个叫dumpIt的工具，它可以把当前运行的系统的内存数据导出为静态镜像文件。

imageinfo

对于内存取证，不同版本的系统运行时的内存数据格式是不一样的，利用这一点，可以先行分析出目标内存镜像对应的系统版本。然后再根据系统版本进行下一步的分析。

功能插件为imageinfo，

```
root@xibai-kali:~/桌面/study/2020/hxb/misc# volatility -f 1.raw imageinfo
Volatility Foundation Volatility Framework 2.6
INFO    : volatility.debug    : Determining profile based on KDBG search...
ML-PL  Suggested Profile(s) : Win7SP1x86_23418, Win7SP0x86, Win7SP1x86_24000,
Win7SP1x86                         msswindowstation.

AS Layer1 : IA32PagedMemoryPae (Kernel AS)
AS Layer2 : FileAddressSpace (/root/study/2020/hxb/misc/1.r
aw)

PAE type : PAE
DTB : 0x185000L
KDBG : 0x83f61c28L

Number of Processors : 2
Image Type (Service Pack) : 1
KPCR for CPU 0 : 0x83f62c00L
session_0.           KPCR for CPU 1 : 0x807ca000L
Service-0x0-3e          KUSER_SHARED_DATA : 0xfffff0000L
Default             Image date and time : 2019-09-27 15:20:52 UTC+0000
Image local date and time : 2019-09-27 23:20:52 +0800
```

psTree|psList|psscan|psxview

ps插件全家桶。它们的功能如图：对内存数据进行分析显然不能错过系统运行时的进程信息分析。而这四个命令则类似Linux系统中的ps命令，可以列举系统运行中的进程。根据列举出的进程可以初步猜测是否受到了进程注入类的攻击。

pslist	Print all running processes by following the EPROCESS lists
psscan	Pool scanner for process objects
pstree	Print process list as a tree
psxview	Find hidden processes with various process listings

三者的详细区别：

- pslist。不仅显示了所有正在运行的进程，而且给出了有价值的信息，比如PID、PPID、启动的时间。
- pstree。pslist的改进版，可以识别子进程以及父进程
- psscan。可以显示出被恶意软件比如rootkit为了躲避用户或杀毒软件而隐藏的进程
- psxview。psscan的改进版。

Volatility Foundation Volatility Framework 2.6							
Offset(V)	Name	PID	PPID	Thds	Hnds	Sess	Wow64 Start
0x8634b7e0	System	4648	0	91	472	-----	0 2019-09-27 15:19:00 UTC+0000
0x86f4b280	smss.exe	252	4	2	30	-----	0 2019-09-27 15:19:00 UTC+0000
0x875d4030	csrss.exe	336	320	10	486	0	0 2019-09-27 15:19:03 UTC+0000
0x87786390	csrss.exe	388	380	10	218	1	0 2019-09-27 15:19:04 UTC+0000
0x8778e030	wininit.exe	396	320	4	82	0	0 2019-09-27 15:19:04 UTC+0000
0x877ae3e0	winlogon.exe	448	380	6	120	1	0 2019-09-27 15:19:04 UTC+0000
0x877ec300	services.exe	496	396	15	224	0	0 2019-09-27 15:19:05 UTC+0000
0x877f77a8	lsass.exe	504	396	10	579	0	0 2019-09-27 15:19:05 UTC+0000
0x877f8b70	lsm.exe	512	396	11	151	0	0 2019-09-27 15:19:05 UTC+0000
0x87896530	svchost.exe	608	496	13	372	0	0 2019-09-27 15:19:07 UTC+0000
0x8659a030	vmaclhlp.exe	676	496	3	55	0	0 2019-09-27 15:19:07 UTC+0000
0x878bc030	svchost.exe	720	496	8	280	session_0.	0 2019-09-27 15:19:08 UTC+0000
0x878e94b8	svchost.exe	Se800ce	496	3e420	401	vice-0x0-3e40	20 2019-09-27 15:19:08 UTC+0000
0x878f2a00	svchost.exe	852	fau	496	21	395Defau0t.png	0 2019-09-27 15:19:08 UTC+0000
0x878f7b08	svchost.exe	884	496	44	877	0	0 2019-09-27 15:19:08 UTC+0000
0x879110b0	audiogd.exe	964	820	7	131	0	0 2019-09-27 15:19:09 UTC+0000
0x87927d40	svchost.exe	1036	496	14	556	0	0 2019-09-27 15:19:09 UTC+0000
0x87946030	svchost.exe	1124	496	17	365	0	0 2019-09-27 15:19:09 UTC+0000
0x8783b030	spoolsv.exe	1284	496	16	316	0	0 2019-09-27 15:19:11 UTC+0000
0x87853030	svchost.exe	1312	496	22	319	0	0 2019-09-27 15:19:11 UTC+0000
0x879db820	VGAuthService.	1480	496	3	84	0	0 2019-09-27 15:19:12 UTC+0000
0x879c4aa0	vmtoolsd.exe	1520	496	10	271	0	0 2019-09-27 15:19:13 UTC+0000
0x870e6838	svchost.exe	1740	re	496	7	94	0 2019-09-27 15:19:15 UTC+0000
0x87156398	dllhost.exe	1984	496	22	200	0	0 2019-09-27 15:19:17 UTC+0000

memdump

memdump可以提取出内存中的进程数据。许多进程在运行时，原始数据都是在进程中存储的，比较经典的例子就是 提取lsass进程的数据，然后利用mimikatz工具抓取用户的明文密码或者说还原用户认证信息。

Volatility Foundation Volatility Framework 2.6							

Writing notepad.exe [2440] to 2440.dmp							
root@xibai-kali:	~/桌面/study/2020/hxb/misc#	1.raw	1.txt	1.vmem			
桌面							
视频							

procdump

提取进程的可执行文件。通过导出可疑进程的可执行文件来对其进行逆向分析，挖掘可能存在的后门病毒木马等。

```
root@xibai-kali:~/桌面/study/2020/hxb/misc# volatility -f 1.vmem --profile=Win2003SP1x86 procDump -p 2440 -D ./notepad.exe
Volatility Foundation Volatility Framework 2.6 444.exe 4.1 kB 2020 年 11 月 1 日 ★
Process(V) ImageBase Name Result
-----
0x87e37958 0x01000000 notepad.exe 444.dmp OK: executable.2440.exe 146.1 MB 2020 年 11 月 1 日 ★
```

timeliner

根据时间线列举系统行为。通过时间线的排布来对可疑程序的可疑行为进行顺藤摸瓜式的排查。

```
root@xibai-kali:~/桌面/study/2020/hxb/misc# volatility -f 1.vmem --profile=Win2003SP1x86 timeliner
Volatility Foundation Volatility Framework 2.6
2019-12-20 15:42:24 UTC+0000|[LIVE RESPONSE]| (System time)
2019-09-20 13:40:35 UTC+0000|[IEHISTORY]| explorer.exe->Visited: Administrator@about:Home| PID: 2028/Cache type "URL" at 0xb25100 End: 2019-09-20 13:40:35 UTC+0000
2019-12-20 15:41:43 UTC+0000|[IEHISTORY]| explorer.exe->Visited: Administrator@file:///C:/Documents%20and%20Settings/Administrator/????/file.txt| PID: 2028/Cache type "URL" at 0xb25200 End: 2019-12-20 15:41:43 UTC+0000
2019-12-20 22:12:07 UTC+0000|[IEHISTORY]| explorer.exe->:2019122020191221: Administrator@Host: ????????| PID: 2028/Cache type "URL" at 0xbd5100 End: 2019-12-20 14:12:07 UTC+0000
2019-12-20 23:41:43 UTC+0000|[IEHISTORY]| explorer.exe->:2019122020191221: Administrator@file:///C:/Documents%20and%20Settings/Administrator/????/file.txt| PID: 2028/Cache type "URL" at 0xbd5200 End: 2019-12-20 15:41:43 UTC+0000
```

cmdline|cmdscan|consoles

这三个功能插件可以列举系统运行时由cmd执行过的命令。对于一些后台调用了cmd的程序可以看到它们的调用历史以及传入参数，遇见不常见的后台调用或者说可疑传参的时候可以结合其它功能对其进行深入分析。

cmdline	Display process command-line arguments
cmdscan	Extract command history by scanning for _COMMAND_HISTORY_ML

```
root@xibai-kali:~/桌面/study/2020/hxb/misc# volatility -f 1.raw --profile=Win7SP0x86 cmdline
Volatility Foundation Volatility Framework 2.6
*****
System pid: 4
*****
smss.exe pid: 252
Command line : \SystemRoot\System32\smss.exe
*****
csrss.exe pid: 336 2648.data mem.raw ML.py session_0.mssrestri
Command line : %SystemRoot%\system32\csrss.exe ObjectDirectory=\Windows SharedSection=1024,12288,512 Windows=On SubSys
verDll=winsrv:ConServerDllInitialization,2 ServerDll=sxssrv,4 ProfileControl=Off MaxRequestThreads=16
*****
csrss.exe pid: 388
Command line : %SystemRoot%\system32\csrss.exe ObjectDirectory=\Windows SharedSection=1024,12288,512 Windows=On SubSys
verDll=winsrv:ConServerDllInitialization,2 ServerDll=sxssrv,4 ProfileControl=Off MaxRequestThreads=16
*****
wininit.exe pid: 396
Command line : wininit.exe
*****
winlogon.exe pid: 448 session_0. session_0. session_0.
Command line : winlogon.exe Service-0x0-3e4$. Service-0x0-3e5$. Service-0x0-3e7$.
*****
services.exe pid: 496 Default.png Default.png Default.png
Command line : C:\Windows\system32\services.exe
*****
lsass.exe pid: 504
Command line : C:\Windows\system32\lsass.exe
*****
```

iehistory

此插件可以查看系统运行时的浏览记录。这个浏览记录包括本地文件记录和浏览器网络链接记录，借此可以分析攻击者进入系统后的行为。

```
root@xibai-kali:~/桌面/study/2020/hxb/misc# volatility -f 1.vmem --profile=Win2003SP1x86 iehistory
Volatility Foundation Volatility Framework 2.6
*****
Process: 2028 explorer.exe
Cache type "DEST" at 0x14389d
Last modified: 2019-12-20 23:41:43 UTC+0000
Last accessed: 2019-12-20 15:41:44 UTC+0000
URL: Administrator@file:///C:/Documents%20and%20Settings/Administrator/Lhb/file.txt
*****
Process: 2028 explorer.exe
Cache type "DEST" at 0x143b15
Last modified: 2019-12-20 23:41:43 UTC+0000
Last accessed: 2019-12-20 15:41:44 UTC+0000
URL: Administrator@file:///C:/Documents%20and%20Settings/Administrator/Lhb/file.txt
*****
Process: 2028 explorer.exe
Cache type "URL " at 0xb25100
Record length: 0x100
Location: Visited: Administrator@about:Home.n_0.
Last modified: 2019-09-20 13:40:35 UTC+0000 0x-3e4$.Service-0x0-3e5$.Default.png
Last accessed: 2019-09-20 13:40:35 UTC+0000 t.png
File Offset: 0x100, Data Offset: 0x0, Data Length: 0x8c
*****
Process: 2028 explorer.exe
Cache type "URL " at 0xb25200
Record length: 0x100
Location: Visited: Administrator@file:///C:/Documents%20and%20Settings/Administrator/????/file.txt
Last modified: 2019-12-20 15:41:43 UTC+0000
Last accessed: 2019-12-20 15:41:43 UTC+0000
File Offset: 0x100, Data Offset: 0x0, Data Length: 0xc4
*****
Process: 2028 explorer.exe
Cache type "URL " at 0xbd5100
Record length: 0x100
Location: :2019122020191221: Administrator@Host: ????????
```

connections|connscan

这两个插件则可以列举系统当时的网络连接情况。根据网络连接的IP和端口可以初步分析是否收到了常见的漏洞攻击。同时也可以在掌握了攻击者的攻击痕迹之后模拟现场。

```
root@xibai-kali:~/桌面/study/2020/hxb/misc# volatility -f 1.vmem --profile=Win2003SP1x86 connscan
Volatility Foundation Volatility Framework 2.6
Offset(P) Local Address          Remote Address        Pid
-----  -----
0x04d2d820 192.168.202.136:135  192.168.202.136:1027  828
0x04d2da08 192.168.202.136:1027  192.168.202.136:135  1240
```

notepad|editbox

这两个插件可以找出正在编辑中的文本数据。`editbox`比`notepad`适用性广一点。

```
root@xibai-kali:~/桌面/study/2020/hxb/misc# volatility -f 1.vmem --profile=Win2003SP1x86 notepad
Volatility Foundation Volatility Framework 2.6
Process: 2440
Text: 文档
Text: 下载
Text: 音乐
Text: 回收站
Text: GParted-live
Text: 其他位置
Text: flag?????????md5??
??md5(???)
root@xibai-kali:~/桌面/study/2020/hxb/misc# volatility -f 1.vmem --profile=Win2003SP1x86 editbox
Volatility Foundation Volatility Framework 2.6
*****
Wnd Context : 0\WinSta0\Default
Process ID : 2440
ImageFileName : notepad.exe
IsWow64 : No
atom_class : 6.0.3790.1830!Edit
value-of WndExtra : -
```

filescan|dumpfiles

filescan可以输出系统文件列表。 **dumpfiles**可以提取被加载进内存的文件数据。 比如在查看cmd命令时发现执行了可疑的可执行程序或者说脚本文件时可以直接提取出来分析其内容。

```
root@xibai-kali:~/桌面/study/2020/hxb/misc# volatility -f 1.vmem --profile=Win2003SP1x86 filescan | grep txt
Volatility Foundation Volatility Framework 2.6
0x0000000000412cde0 1 0 RW-r-- \Device\HarddiskVolume1\Documents and Settings\Administrator\桌面\file.txt
0x0000000000426b890 1 0 R-rw- \Device\HarddiskVolume1\Documents and Settings\All Users\Application Data\VMware\VMware Tools\Unity Filters\win7gadgets.txt
0x0000000000426ba90 1 0 R-rw- \Device\HarddiskVolume1\Documents and Settings\All Users\Application Data\VMware\VMware Tools\Unity Filters\vmwarefilters.txt
0x0000000000426bc90 1 0 R-rw- \Device\HarddiskVolume1\Documents and Settings\All Users\Application Data\VMware\VMware Tools\Unity Filters\visualstudio2005.txt
0x0000000000426be90 1 0 R-rw- \Device\HarddiskVolume1\Documents and Settings\All Users\Application Data\VMware\VMware Tools\Unity Filters\vistasidebar.txt
0x0000000000479d4a8 4 2 -W-rw- \Device\HarddiskVolume1\Documents and Settings\All Users\Application Data\VMware\VGAuth\logfile.txt.0
0x000000000049e1cf0 1 0 R-rw- \Device\HarddiskVolume1\Program Files\VMware\VMware Tools\vmacthlp.txt
0x000000000049e6228 1 0 RW-rw- \Device\HarddiskVolume1\Documents and Settings\Administrator\Recent\file.txt.lnk
0x00000000004a511a0 1 0 R-rw- \Device\HarddiskVolume1\Documents and Settings\All Users\Application Data\VMware\VMware Tools\Unity Filters\microsoftoffice.txt
0x00000000004a513a0 1 0 R-rw- \Device\HarddiskVolume1\Documents and Settings\All Users\Application Data\VMware\VMware Tools\Unity Filters\googledesktop.txt
0x00000000004a51770 1 0 R-rw- \Device\HarddiskVolume1\Documents and Settings\All Users\Application Data\VMware\VMware Tools\Unity Filters\adobephotoshopcs3.txt
0x00000000004c05370 1 0 R-rwd \Device\HarddiskVolume1\Documents and Settings\All Users\Application Data\VMware\VMware Tools\Unity Filters\adobephotoshopcs3.manifest.txt
0x00000000004c70ae8 1 0 RW---- \Device\HarddiskVolume1\WINDOWS\system32\CatRoot2\dberr.txt
0x00000000004d44028 1 0 R-rw- \Device\HarddiskVolume1\Documents and Settings\All Users\Application Data\VMware\VMware Tools\Unity Filters\adobeflashcs3.txt
root@xibai-kali:~/桌面/study/2020/hxb/misc# volatility -f 1.vmem --profile=Win2003SP1x86 dumpfiles -Q 0x000000000412cde0 -D ../
Volatility Foundation Volatility Framework 2.6
DataSectionObject 0x0412cde0 None \Device\HarddiskVolume1\Documents and Settings\Administrator\桌面\file.txt session_0.
```

hashdump

该工具可以抓取当前系统中的用户名及其密码对应的**NTML**哈希值。如果攻击者创建了影子账户，利用该命令可直接发现。

```
[root@xibai-kali:~/桌面/study/2020/hxb/misc# volatility -f 1.vmem --profile=Win2003SP1x  
+86 hashdump  
Volatility Foundation Volatility Framework 2.6  
Administrator:500:f0d412bd764ffe81aad3b435b51404ee:209c6174da490caeb422f3fa5a7ae634:::  
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::  
SUPPORT_388945a0:1001:aad3b435b51404eeaad3b435b51404ee:8d9221b8e70124641a83291d3d21f7e  
0:::  
9w3a6J0$:1003:e761601f5cf981c136077a718ccdf409:ec9dc7d0895ad3dae1feba8ffdeacffd:::  
4hiU9ZK$:1004:de5eea9d3fd12c34aad3b435b51404ee:2f2d544c53b3031f24d63402ea7fb4f9:::  
A4W7iKb$:1005:61339c1be342167eaad3b435b51404ee:b6e6f6a85f90219d619aca4706f354fc:::  
oeTQczq$:1006:b4d2cf4a862f6fc当地3b435b51404ee:3fbc1f9dc4416f6fb3666de834185cb4:::  
CALrXyU$:1007:8ea6fb8594a1b952aad3b435b51404ee:51d603c77a884df049f7ed4dabed4fd4:::  
AVqKsvQ$:1008:939e0f8990e68047aad3b435b51404ee:1796c2db94ce6276744f88b740152154:::  
scdTbYy$:1009:792677ee54a26732891c5133c13673e8:138393419f9b418eb735d36e1da50a5e:::
```

hivelist|hivescan|hivedump

hivescan插件显示了可用的注册表配置单元的物理地址

```
[root@xibai-kali:~/桌面/study/2020/hxb/misc# volatility -f 1.vmem --profile=Win2003SP1x  
+86 hivescan  
Volatility Foundation Volatility Framework 2.6  
Offset(P)  
-----  
0x0d7eb008  
0x0d95b008  
0x0da280b0  
0x0da35a80  
0x0e0dda80  
0x12d41008  
0x12d8e860  
0x12df8200  
0x12eab008  
0x1776ea80  
0x17851260  
0x17b28008  
0x17b31008
```

更加详细的信息可以通过hivelist命令查看，这条命令会显示虚拟地址、物理地址的细节以及更容易识别的路径等

```
root@xibai-kali:~/桌面/study/2020/hxb/misc# volatility -f 1.vmem --profile=Win2003SP1x86 hivelist
Volatility Foundation Volatility Framework 2.6
Virtual Physical Name
-----
0xe174a008 0x12eab008 \Device\HarddiskVolume1\Documents and Settings\Administrator\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
0xe1013008 0x17b28008 [no name]
0xe101d008 0x17b31008 \Device\HarddiskVolume1\WINDOWS\system32\config\system
0xe12fb260 0x17851260 [no name]
0xe1756200 0x12df8200 \Device\HarddiskVolume1\WINDOWS\system32\config\SECURITY
0xe1763008 0x12d41008 \Device\HarddiskVolume1\WINDOWS\system32\config\default
0xe13c9a80 0x1776ea80 \Device\HarddiskVolume1\WINDOWS\system32\config\software
0xe1757860 0x12d8e860 \Device\HarddiskVolume1\WINDOWS\system32\config\SAM
0xe24560b0 0x0da280b0 \Device\HarddiskVolume1\Documents and Settings\NetworkService\NTUSER.DAT
0xe2460a80 0x0da35a80 \Device\HarddiskVolume1\Documents and Settings\NetworkService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
0xe247c008 0x0d95b008 \Device\HarddiskVolume1\Documents and Settings\LocalService\NTUSER.DAT
0xe2484008 0xd7eb008 \Device\HarddiskVolume1\Documents and Settings\LocalService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
```

hivedump则可以导出注册表信息

```
root@xibai-kali:~/桌面/study/2020/hxb/misc# volatility -f 1.vmem --profile=Win2003SP1x86 dumpregistry -o 0xe175860 -D ./位置 executable.444.exe
Volatility Foundation Volatility Framework 2.6
*****
Writing out registry: registry.0xe1757860.SAM.reg
*****
```

printkey

查看内存加载的注册表中的键值。比如在进程分析时发现了对注册表的修改痕迹，可以直接查询对应注册表的键值判断是否是攻击行为的修改。

```
root@xibai-kali:~/桌面/study/2020/hxb/misc# volatility -f 1.vmem --profile=Win2003SP1x86 printkey -K "SAM"
Volatility Foundation Volatility Framework 2.6
Legend: (S) = Stable (V) = Volatile
[图片] wtnmd.txt 7.1 MB 2020 年 11 月 1 日
Registry: \Device\HarddiskVolume1\WINDOWS\system32\config\SECURITY
Key name: SAM (V)
Last updated: 2019-12-20 14:18:07 UTC+0000
[音乐] Subkeys: 804.dmp 140.2 MB 2020 年 11 月 1 日
Values:
REG_LINK executable.804.exe 133.1 kB 2020 年 11 月 1 日
SymbolicLinkValue : (V) \Registry\Machine\SAM\SAM
[恢复] Registry: \Device\HarddiskVolume1\WINDOWS\system32\config\SAM
Key name: SAM (S)
Last updated: 2019-12-20 14:02:09 UTC+0000
Subkeys: 1.txt 29.2 kB 2020 年 11 月 1 日
(S) Domains
(S) RXACT
已选中 "registry.0xe1757860.SAM.reg" (471.0 kB)
```

dlllist|dlldump

dlllist可以看到每个进程运行需要的dll, dlldump可以导出进程运行中加载的dll。这一条针对windows系统中的dll注入。

```
root@xibai-kali:~/桌面/study/2020/hxb/misc# volatility -f 1.vmem --profile=Win2003SP1x86 dlllist
Volatility Foundation Volatility Framework 2.6 2440.exe 66.0 kB 17:24
*****
System pid: 4 registry.0xe1757860.SAM.reg 471.0 kB 17:14
Unable to read PEB for task.
*****
smss.exe pid: 296 2440.dmp 139.8 MB 15:49
Command line : \SystemRoot\System32\smss.exe
file.None.0x87955ea0.dat 4.1 kB 15:36
[文档]
Base 下载 Size LoadCount LoadTime Path
----- -----
0x48580000 0x10000 0xffff \SystemRoot\System32\smss.exe
0x7c930000 0xd0000 0xffff file.None.0x878eac60.dat C:\WINDOWS\system32\ntdll.dll 11月 1日
*****
csrss.exe pid: 444 804.dmp 140.2 MB 2020 年 11 月 1 日
Command line : C:\WINDOWS\system32\csrss.exe ObjectDirectory=\Windows SharedSection=1024,3072,512 Windows=On Sub
SystemType=Windows ServerDll=basesrv,1 ServerDll=winsrv:UserServerDllInitialization,3 ServerDll=winsrv:ConServer
DllInitialization,2 ProfileControl=Off MaxRequestThreads=16 133.1 kB 2020 年 11 月 1 日
Service Pack 1
Base 下载 Size LoadCount LoadTime Path
----- -----
0x4a680000 0x4000 0xffff executable.444.exe 4.1 kB 2020 年 11 月 1 日
0x7c930000 0xd0000 0xffff 444.dmp \??\C:\WINDOWS\system32\csrss.exe 10月 1日
0x7c930000 0xd0000 0xffff C:\WINDOWS\system32\ntdll.dll 66.0 kB 10月 1日
```

Process(V) Name	Module Base	Module Name	Result	Size	Time	Star
0x87e37958 notepad.exe	0x00100000	NOTEPAD.EXE	OK: module.2440.4d36958.100000.dll	14		
0x87e37958 notepad.exe	0x07c93000	ntdll.dll	OK: module.2440.4d36958.7c930000.dll			
0x87e37958 notepad.exe	0x07ca1000	SHELL32.dll	OK: module.2440.4d36958.7ca10000.dll			
0x87e37958 notepad.exe	0x077f3000	ADVAPI32.dll	OK: module.2440.4d36958.77f30000.dll	49		
0x87e37958 notepad.exe	0x072f4000	WINSPOOL.DRV	OK: module.2440.4d36958.72f40000.dll			
0x87e37958 notepad.exe	0x04b21000	MSCTF.dll	OK: module.2440.4d36958.4b210000.dll	36		
0x87e37958 notepad.exe	0x077b7000	msvcrt.dll	OK: module.2440.4d36958.77b70000.dll			
0x87e37958 notepad.exe	0x07618000	IMM32.DLL	OK: module.2440.4d36958.76180000.dll			
0x87e37958 notepad.exe	0x071ad0000	UxTheme.dll	OK: module.2440.4d36958.71ad0000.dll			
0x87e37958 notepad.exe	0x077bd0000	GDI32.dll	OK: module.2440.4d36958.77bd0000.dll			
0x87e37958 notepad.exe	0x0774b0000	ole32.dll	OK: module.2440.4d36958.774b0000.dll			
0x87e37958 notepad.exe	0x0761a0000	comdlg32.dll	OK: module.2440.4d36958.761a0000.dll			
0x87e37958 notepad.exe	0x077eb0000	SHLWAPI.dll	OK: module.2440.4d36958.77eb0000.dll			
0x87e37958 notepad.exe	0x04c510000	msctftime.ime	OK: module.2440.4d36958.4c510000.dll			
0x87e37958 notepad.exe	0x077cd0000	COMCTL32.dll	OK: module.2440.4d36958.77cd0000.dll			
0x87e37958 notepad.exe	0x063090000	LPK.DLL	OK: module.2440.4d36958.63090000.dll			
0x87e37958 notepad.exe	0x074ae0000	USP10.dll	OK: module.2440.4d36958.74ae0000.dll			
0x87e37958 notepad.exe	0x075d60000	apphelp.dll	OK: module.2440.4d36958.75d60000.dll			
0x87e37958 notepad.exe	0x07c800000	kernel32.dll	OK: module.2440.4d36958.7c800000.dll			
0x87e37958 notepad.exe	0x077e10000	USER32.dll	OK: module.2440.4d36958.77e10000.dll			
0x87e37958 notepad.exe	0x077c20000	RPCRT4.dll	OK: module.2440.4d36958.77c20000.dll			

svscan (限windows)

查看开启的windows服务。通过开启的服务可以对常见的出名的漏洞攻击做一个快速过滤。

Offset:	Start:	Service Name:	Display Name:	Service Type:	Service State:	Binary Path:	File Size	Time	Star
0x621e90	SERVICE_DISABLED	Abiosdsk	Abiosdsk	SERVICE_KERNEL_DRIVER	SERVICE_STOPPED	file.None.0x87955ea0.dat	4.1 kB	15 : 36	
Order: 1	Start: SERVICE_BOOT_START	wtnmd.txt					7.1 MB	2020 年 11 月 1 日	
Process ID: -	Process ID: -	Service Name: ACPI	Microsoft ACPI Driver	Service Type: SERVICE_KERNEL_DRIVER	Service State: SERVICE_RUNNING	file.None.0x878eac60.dat	8.2 kB	2020 年 11 月 1 日	
Binary Path: -	Binary Path: -	Display Name: Microsoft ACPI Driver		Binary Path: -	Binary Path: -	804.dmp	140.2 MB	2020 年 11 月 1 日	
Offset: 0x621f28	Start: SERVICE_BOOT_START	Service Name: ACPI	Microsoft ACPI Driver	Service Type: SERVICE_KERNEL_DRIVER	Service State: SERVICE_RUNNING	executable.804.exe	133.1 kB	2020 年 11 月 1 日	
Order: 2	Process ID: -	Display Name: Microsoft ACPI Driver		Order: 2	Process ID: -	executable.444.exe	4.1 kB	2020 年 11 月 1 日	
Start: SERVICE_BOOT_START	Service Name: ACPI	Service Type: SERVICE_KERNEL_DRIVER	Service State: SERVICE_RUNNING	Start: SERVICE_BOOT_START	Service Name: ACPI	44.dmp	146.1 MB	2020 年 11 月 1 日	

modules|modscan|driverscan

查看系统内核驱动。隐藏的用modscan或者driverscan

Volatility Foundation Volatility Framework 2.6				
Offset(V)	Name	Base	Size	File
0x8823a308	ntoskrnl.exe	0x80800000	0x247000	\WINDOWS\system32\ntkrnlpa.exe
0x8823a2a0	hal.dll	0x80a47000	0x2c000	\WINDOWS\system32\hal.dll
0x8823a238	kdcom.dll	0xf7707000	0x8000	\WINDOWS\system32\KDCOM.DLL
0x8823a1c8	BOOTVID.dll	0xf770f000	0x8000	\WINDOWS\system32\BOOTVID.dll
0x8823a160	ACPI.sys	0xf7352000	0x34000	ACPI.sys
0x8823a0f0	WMILIB.SYS	0xf7487000	0x9000	\WINDOWS\system32\DRIVERS\WMILIB.SYS
0x8823a088	pci.sys	0xf733d000	0x15000	pci.sys
0x881f7008	isapnp.sys	0xf7497000	0xf000	isapnp.sys
0x881f7f98	compbatt.sys	0xf7897000	0x3000	compbatt.sys
0x881f7f30	BATTC.SYS	0xf7717000	0x5000	\WINDOWS\system32\DRIVERS\BATTC.SYS
0x881f7ec0	intelide.sys	0xf771f000	0x7000	intelide.sys
0x881f7e50	PCIIDEX.SYS	0xf74a7000	0xd000	\WINDOWS\system32\DRIVERS\PCIIDEX.SYS
0x881f7de0	MountMgr.sys	0xf74b7000	0x10000	MountMgr.sys
0x881f7d70	ftdisk.sys	0xf7317000	0x26000	ftdisk.sys
0x881f7d00	dmload.sys	0xf7727000	0x7000	dmload.sys
0x881f7c98	dmio.sys	0xf72ec000	0x2b000	dmio.sys
0x881f7c28	volsnap.sys	0xf72c3000	0x29000	volsnap.sys

screenshot

查看当前屏幕每个窗口中内容的轮廓线。属于侧信道信息分析范畴，一般用不到。

Volatility Foundation Volatility Framework 2.6				
Wrote ./session_0.SAWinSta.SADesktop.png	session_0.	SAWinSta.	session_0.	session_0.
Wrote ./session_0.Service-0x0-3e7\$.Default.png	session_0.	Service-0x0-3e7\$.	Default.	WinSta0.Default.
Wrote ./session_0.Service-0x0-3e4\$.Default.png	session_0.	Service-0x0-3e4\$.	Default.	WinSta0.
Wrote ./session_0.Service-0x0-3e5\$.Default.png	session_0.	Service-0x0-3e5\$.	Default.	Disconnect.png
Wrote ./session_0.WinSta0.Default.png	session_0.	WinSta0.	Default.	
Wrote ./session_0.WinSta0.Disconnect.png	session_0.	WinSta0.	Disconnect.	
Wrote ./session_0.Winlogon.png	session_0.	Winlogon.	png	



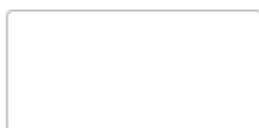
session_0.
SAWinSta.
SADesktop.png



session_0.
Service-0x0-3e4\$.
Default.png



session_0.
Service-0x0-3e5\$.
Default.png



session_0.
Service-0x0-3e7\$.
Default.png



session_0.
WinSta0.Default.
png



session_0.
WinSta0.
Disconnect.png

硬盘取证

硬盘取证的常见场景一般多为可疑文件数据恢复与查找

一般来说，恢复被恶意删除的文件会遇到如下四种情况。

1. 目录项未覆盖，文件数据未覆盖
2. 目录项已覆盖，文件数据未覆盖
3. 目录项未覆盖，文件数据已覆盖
4. 目录项已覆盖，文件数据已覆盖

什么是目录项呢？

关于目录项，它是文件系统在存储数据时规定的一个固定数据格式的数据，不同的文件系统会有不同的规定内容，但必然会存在这个结构，不然文件系统自己也无法判断在一个地方放的是什么东西有多长叫什么名字。每个目录下的所有文件和文件夹的目录项都会按顺序放在同一个数据区存储。

这个数据区中的每一条数据便对应了硬盘中存储的一个文件或者说一个文件夹，其中有着这个文件的名字大小类型位置等基础信息。

也就是说，**文件系统在寻找存储在硬盘中的文件时，是先去目录区查找对应文件的目录项，从中得知目标文件的各种信息，然后直接定位文件。而文件系统删除文件时之所以如此之快也正是因为它并非真正意义上的删除了文件，而是修改了文件目录项的状态值。**

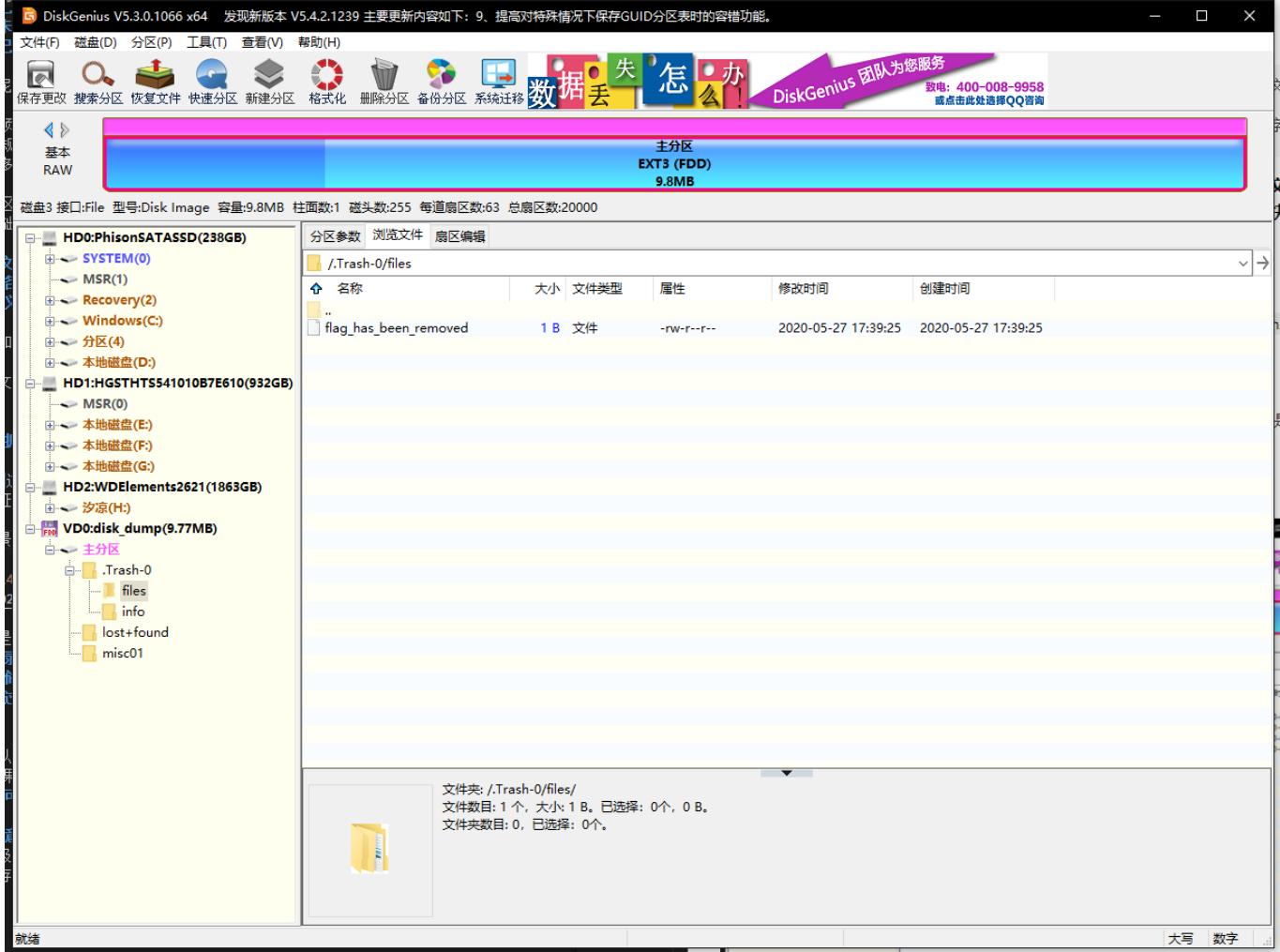
现在也就可以知道以上四种情况意味着什么了。

前两种意味着文件可以被恢复，但是第一种可以简单的恢复，即利用DiskGenius这种工具直接提取恢复：

文件的数据恢复与查询——DiskGenius

DiskGenius这个工具其实正常多用在重装系统或者说修电脑的辅助工具。但是因为功能强大，于是在进行硬盘取证的时候也常常可以用到。

最为常见的场景便是**对于删除文件的恢复**。



第二种情况只是理论上可被恢复，实际情况根据文件本身的大小以及文件类型格式，恢复难度也不一样。因为目录项已被覆盖，以及无法简单的通过工具的方式提取。而文件系统在存储大文件时，往往会将其分块存储，所以对于被分块的大文件已经可以说是不能被恢复了。至于体积较小未能分块存储的文件则可以通过文件特征值扫描的方式尝试提取恢复。这里可以考虑使用binwalk工具自动扫描。

第三种则是可以根据工具快速查看到删除文件“生前”的基础信息，如名字大小格式等，但已无法直接恢复。
(如上图) 如果文件数据仅仅是某些固定的格式信息区被覆盖尚有恢复可能，但如若记载的数据都已经覆盖，则可以当作无法恢复处理。

最后一种则意味着彻底移除，已不存在恢复可能。低级格式化之所以安全就是因为达到了这种效果，相反，高级格式化之所以仍有数据泄露的可能，正是因为高级格式化主要是清空了文件目录项，但对于硬盘本身存储的数据区内容为了效率原因并未做过多处理。

文件系统格式损坏

除了恢复被删文件，还有另外一些常见场景，那便是文件系统格式损坏，正在运行的机械硬盘摔一下就坏了主要就是这种原因。除了目录项以外，文件系统还有其它关键性的自身基础信息，当它们被破坏后，便无法被正常识别是何系统，有哪些基础设置，从而导致操作系统无法解析该硬盘。

但是根据前文我们可以知道，文件系统的其他信息损坏并没有从根本上损坏存储介质中的数据。这个时候只要能知道原存储介质中使用的是哪种文件系统，即可采取手工恢复的方式抢救重要数据或者说取证。

这里我们用一道CTF中的例题来讲解一下硬盘取证的姿势。

判断文件系统信息

这里题目文件故意抹去了文件开头0x200字节的代表着文件系统信息的数据。并在文件末尾丢了一个烟雾弹出来。

The screenshot shows the WinHex application window. The title bar reads "WinHex - [[raw.img]]". The menu bar includes "文件(F)", "编辑(E)", "搜索(S)", "位置(P)", "查看(V)", "工具(T)", "专业工具(I)", "选项(O)", "窗口(W)", and "帮助(H)". The toolbar contains icons for opening files, saving, zooming, and navigating. The main area displays a hex dump of the file [raw.img]. The left pane shows the file structure with sections like "Offset", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "A", "B", "C", "D", "E", "F". The right pane provides file details: "文件系统: 未知", "状态: 原始的", "撤消级数: 0", "反向撤消: n/a", "总容量: 500 MB", "524,288,000 字节", "每扇区字节数: 512", "扇区统计: 1,024,000", "模式: 文本", "字符集: ANSI ASCII", "偏移地址: 十六进制", "每页字节数: 25x16=400", "当前窗口: 1", and "窗口总数: 1". The status bar at the bottom shows "扇区 1023999 / 1024000", "偏移地址: 1F3FFF35", "= 104", "选块: n/a", "大小: n/a".

抹去了原本的信息，在结尾放上了一个NTFS文件系统的格式数据。

这里我们先不去在意这个烟雾弹，去看看其他数据有无类似特征值一般的存在。比如说第一张图中那看起来容易引人注意的RRaA，有经验的人就知道，这四个字节代表着原文件系统很可能是FAT系列的文件系统，这里我们往下看一下，发现距离RRaA不到0x200字节就有另外一个大小写相反的rrAa字符。

000003B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000003C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000003D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000003E0	00 00 00 00 72 72 41 61 E9 51 00 00 1B A1 00 00rrAaéQ...i...
000003F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 AAUa
00000400	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000410	nn

简单用搜索引擎查找一下资料就可以知道，这确实是FAT系列文件系统的特征值之一。通过查询FAT相关的文件系统的资料。我们可以得知，FAT后跟的数字其实是类似数据总线宽度的存在。这也是为什么FAT32文件系统最大仅支持4GB大小的文件存储的原因，32bit的数据宽度，意味着在描述文件大小时，最大值即为0xffffffff byte，也即是4GB。

对于FAT文件系统，它一般有四个组成部分。

DBR及其保留扇区 FAT1 FAT2 DATA

DBR及其保留扇区：DBR的含义是DOS引导记录，也称为操作系统引导记录，在DBR之后往往有一些保留扇区。这一块即为文件系统的基础信息数据。

FAT1：FAT的含义是文件分配表，FAT32一般有两份FAT，FAT1是第一份，也是主FAT。文件分配表，和前文提到的文件目录项存在对应关系。文件分配表的开头一般固定为F8FFFF0FFFFFF.

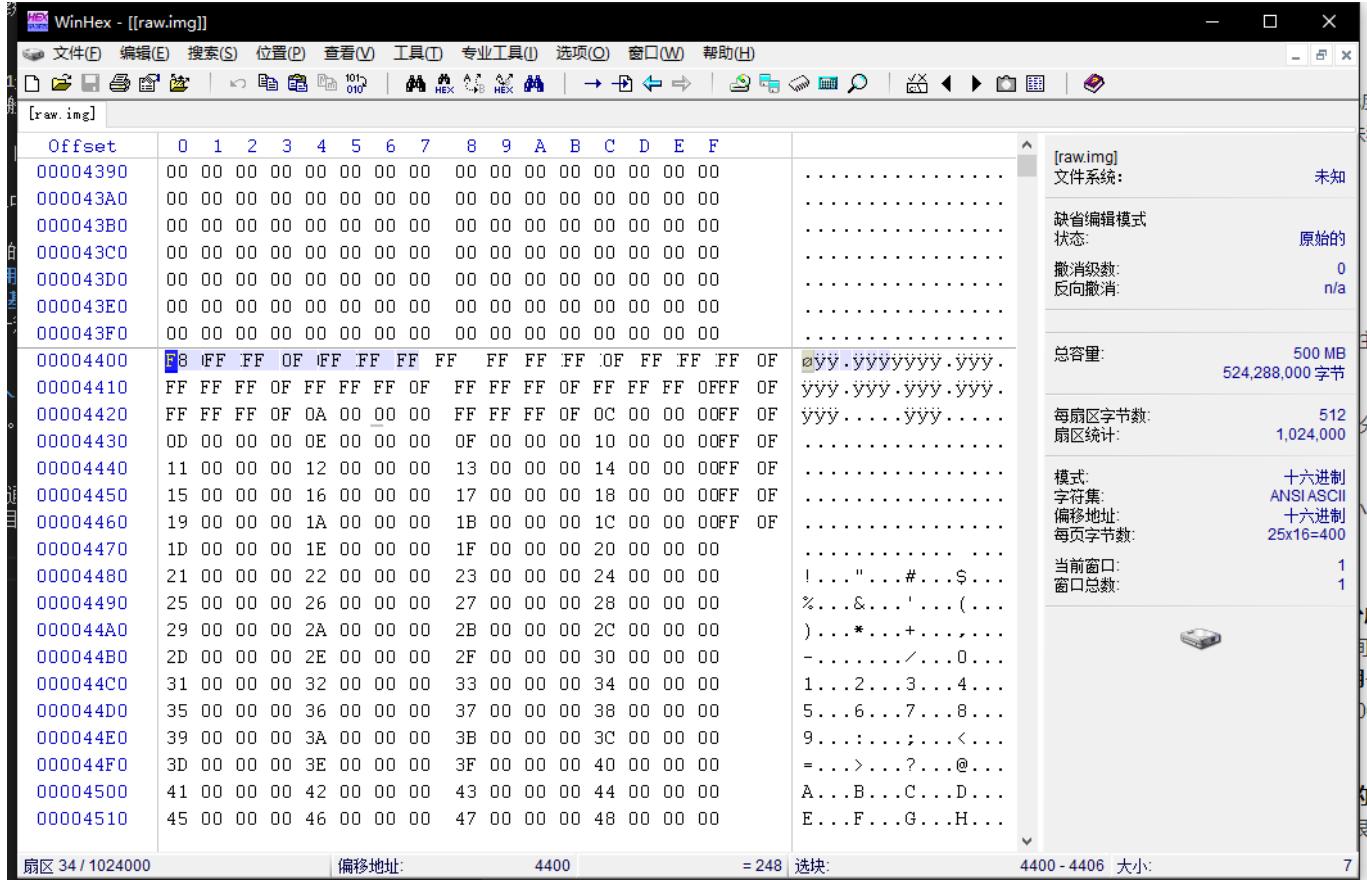
FAT2：FAT2是FAT32的第二份文件分配表，也是FAT1的备份。既然是备份，自然与FAT1大小相同。

DATA：DATA也就是数据区，是FAT32文件系统的主要区域，其中包含存储目录项的区域。

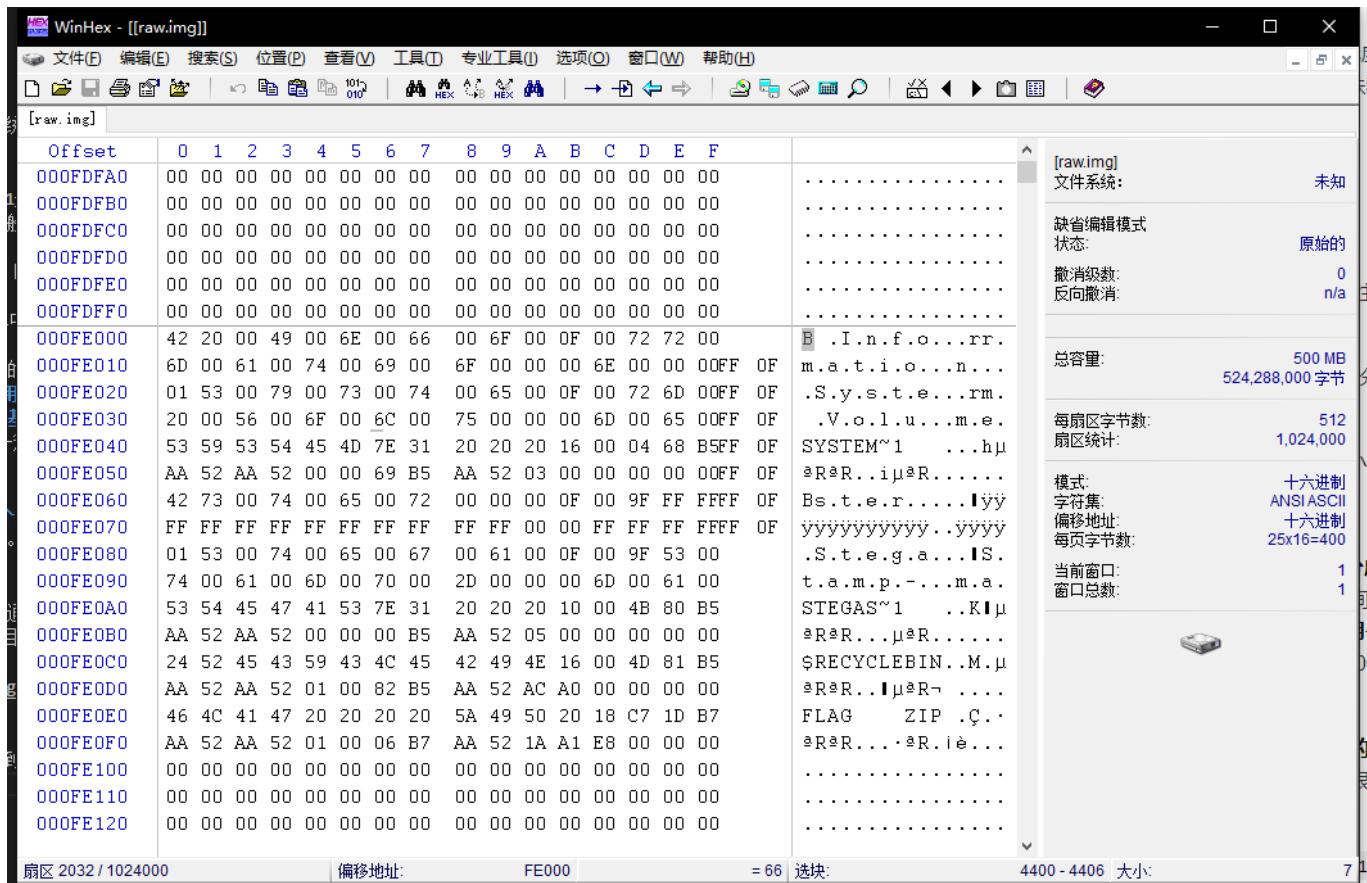
另外这里简单说一下，磁盘的存储空间为方便定位，都有扇区的概念。即把磁盘总空间平均分成固定份，一份就是一个扇区。常用的扇区大小一般默认使用一扇区200bytes。然后因为现在的存储空间相对与200bytes来说太大了，所以文件系统在扇区的基础上又引入了一个单位叫做簇，一般默认使用一簇对应八个扇区。一扇区实际对应多少字节，一簇实际对应多少扇区，都直接记录在开头那被抹掉的200字节基础信息中。

对于FAT系统而言有两个特殊簇，data区域前面的三个区域固定占据两个簇，无论实际规定的一簇为多大。簇号从0开始，也就是data区从第2簇开始。而第2簇在FAT文件系统中默认分配给根目录的目录项表使用。

这里我们并不确定该硬盘镜像原本的数据分配，但我们可以先通过搜索特征值的方式找到FAT1和FAT2，如此便可以直接确认DATA区的起始位置，也即根目录目录项的所在。



这里我们定位到了FAT1和2的位置，直接二者起始地址相减得到FAT1和2的大小，然后即可找到DATA的起始地址为FE000。



目录项结构

这里我们先简单介绍一下FAT32目录项的数据结构。一个目录项固定为32字节。其中分为长目录项和短目录项。区分长短的原因就是因为，文件名是人为自定义的，不可能用短长度定死，所以对于名字较长的文件或者文件夹就用长目录项描述。

短目录项

字节偏移(16进制)	字节数	定义
0x0~0x7	8	文件名
0x8~0xA	3	扩展名
0xB*	1	属性字节 00000000 (读写) 00000001 (只读) 00000010 (隐藏) 00000100 (系统) 00001000 (卷标) 00010000 (子目录) 00100000 (归档)
0xC	1	系统保留
0xD	1	创建时间的10毫秒位
0xE~0xF	2	文件创建时间
0x10~0x11	2	文件创建日期
0x12~0x13	2	文件最后访问日期
0x14~0x15	2	文件起始簇号的高16位
0x16~0x17	2	文件的最近修改时间
0x18~0x19	2	文件的最近修改日期
0x1A~0x1B	2	文件起始簇号的低16位
0x1C~0x1F	4	表示文件的长度

* 此字段在短文件目录项中不可取值0FH, 如果设值为0FH, 目录段为长文件名目录段

时间的解析规则：

Bits	Description
15-11	Hours (0-23)
10-5	Minutes (0-59)
4-0	Seconds/2 (0-29)

日期的解析规则：

Bits	Description
15-9	Year (0 = 1980, 119 = 2099 supported under DOS/Windows, theoretically up to 127 = 2107)
8-5	Month (1-12)
4-0	Day (1-31)

长目录项

表15 FAT32长文件目录项32个字节的表示定义

字节偏移 (16进制)	字节数	定义																
0x0	1	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>7</td><td>保留未用</td></tr> <tr><td>6</td><td>1表示长文件最后一个目录项</td></tr> <tr><td>5</td><td>保留未用</td></tr> <tr><td>4</td><td></td></tr> <tr><td>3</td><td></td></tr> <tr><td>2</td><td>顺序号数值</td></tr> <tr><td>1</td><td></td></tr> <tr><td>0</td><td></td></tr> </table>	7	保留未用	6	1表示长文件最后一个目录项	5	保留未用	4		3		2	顺序号数值	1		0	
7	保留未用																	
6	1表示长文件最后一个目录项																	
5	保留未用																	
4																		
3																		
2	顺序号数值																	
1																		
0																		
0x1~0xA	10	长文件名unicode码																
0xB	1	长文件名目录项标志, 取值0FH																
0xC	1	系统保留																
0xD	1	校验值(根据短文件名计算得出)																
0xE~0x19	12	长文件名unicode码																
0x1A~0x1B	2	文件起始簇号(目前常置0)																
0x1C~0x1F	4	长文件名unicode码																

这里需要注意一点。对于FAT32中的长目录项而言，因为主要内容都用来存放文件名了，所以本身并不能存储文件或者说文件夹的其他信息，仅有名字显然不能用了当作合格的格式规定。

另外凡是涉及整形数据存储皆为小端序。

所以fat中的长目录项结束后必然跟着一个短目录项，长目录项主要用来存放文件名，短目录项用来存放文件的各种基础信息，这里举例说明：

42 20 00 49 00 6E 00 66	00 6F 00 OF 00 72 72 00	B .I.n.f.o....rr.
6D 00 61 00 74 00 69 00	6F 00 00 00 6E 00 00 00	m.a.t.i.o....n...
01 53 00 79 00 73 00 74	00 65 00 OF 00 72 6D 00	.S.y.s.t.e....rm.
20 00 56 00 6F 00 6C 00	75 00 00 00 6D 00 65 00].V.o.l.u....m.e.
53 59 53 54 45 4D 7E 31	20 20 20 16 00 04 68 B5	SYSTEM^1 ...hp
AA 52 AA 52 00 00 69 B5	AA 52 03 00 00 00 00 00	¤R¤R..ip¤R.....
42 73 00 74 00 65 00 72	00 00 00 OF 00 9F FF FF	Bs.t.e.r.....Iyy
FF FF FF FF FF FF FF	FF FF 00 00 FF FF FF FF	yyyyyyyyyyyy...yyyy
01 53 00 74 00 65 00 67	00 61 00 OF 00 9F 53 00	.S.t.e.g.a....IS.
74 00 61 00 6D 00 70 00	2D 00 00 00 6D 00 61 00	t.a.m.p.-....m.a.
53 54 45 47 41 53 7E 31	20 20 20 10 00 4B 80 B5	STEGAS^1 ..K p
AA 52 AA 52 00 00 00 B5	AA 52 05 00 00 00 00 00	¤R¤R...¤R.....
24 52 45 43 59 43 4C 45	42 49 4E 16 00 4D 81 B5	\$RECYCLEBIN..M. p
AA 52 AA 52 01 00 82 B5	AA 52 AC A0 00 00 00 00	¤R¤R.. p¤R-
46 4C 41 47 20 20 20 20	5A 49 50 20 18 C7 1D B7	FLAG ZIP .ç..
AA 52 AA 52 01 00 06 B7	AA 52 1A A1 E8 00 00 00	¤R¤R...¤R.iè...
00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00

我们用图中第一个长目录项来解释。

0x00 : 42代表这是长目录项的结尾，且他是长目录项组合中的第二个。 0x01-0x0a: 用utf-16格式存储的部分文件名。这里为(空格)Info。 0x0b : 0x0f 代表这是长目录项。 0x0c : 系统保留，默认为0。 (扩展，可以利用系统保留位实现简单的消息隐藏，是为基于结构的隐写) 0x0d : 通过文件名计算出来的校验值 0x0e-0x19: 用utf-16格式存储的部分文件名。这里为rmatio 0x1a-0x1b: 说是存放文件起始地址的簇号，其实完全没有用到，固定置0。这里可以结合上面那个系统保留位扩大隐写数据的大小。 0x1c-0x1f: 用utf-16格式存储的部分文件名。这里为n。(另外，若长目录项的文件名没有使用到的数据区默认用0xff填充。这里最后两字节为0000的原因是文件名作为字符串，需要用00截断，所以最后的0000其实是文件名的一部分，这里刚好用完，所以没有用到0xff填充多余空间)

往下看，第二条长目录项：

0x00 : 01代表这是长目录项的起始也即文件名开头。 0x01-0x0a: 用utf-16格式存储的部分文件名。这里为Syste 0x0b : 0x0f 代表这是长目录项。 0x0c : 系统保留, 默认为0。(扩展, 可以利用系统保留位实现简单的消息隐藏, 是为基于结构的隐写) 0x0d : 通过文件名计算出来的校验值 0x0e-0x19: 用utf-16格式存储的部分文件名。这里为m Volu 0x1a-0x1b: 说是存放文件起始地址的簇号, 其实完全没有用到, 固定置0。这里可以结合上面那个系统保留位扩大隐写数据的大小。 0x1c-0x1f: 用utf-16格式存储的部分文件名。这里为me.(另外, 若长目录项的文件名没有使用到的数据区默认用0xff填充。这里最后两字节为0000的原因是文件名作为字符串, 需要用00截断, 所以最后的0000其实是文件名的一部分, 这里刚好用完, 所以没有用到0xff填充多余空间)

再往下看，长目录项已经结束，这里是一个短目录项，用来记录上面那组长目录项记录的名字所对应的文件（夹）的基础信息。

0x00-0x07: 长文件名对应的短名。这里为SYSTEM~1. 0x08-0x0a: 文件扩展名, 文件夹没有扩展名, 故为三个空格. 0x0b: 0x16, 代表是隐藏的系统级目录 0x0c : 保留位。 0x0d : 创建时间的10毫秒位 0x0e-0x0f: 文件创建时

间 68b5，考虑小端序，实际为：b568，可解析出时间为：22:43:16 0x10-0x11: 文件创建日期 aa52，考虑小端序，实际为：52aa，可解析出日期为：2021-05-10 0x12-0x13: 文件最后访问日期。0x14-0x15: 起始簇号高16位，依旧小端序。0x16-0x17: 文件最近修改时间。0x18-0x19: 文件最近修改日期。0x1a-0x1b: 起始簇号高16位，依旧小端序。这里相当于 3 0x1c-0x1f: 文件长度，小端序。目录没有长度属性，故置零。

关于文件簇号和存储地址的转换，满足如下关系：

$$address_{file} = (cluster_{num} - 2) * size_{cluster} + address_{DATA}$$

$address_{file}$ 代表 文件的实际存储地址区。

$cluster_{num}$ 代表 文件目录项中的起始簇号。

$size_{cluster}$ 代表 该文件系统中，一簇占用的空间大小

$address_{DATA}$ 代表数据的起始地址

这里可以以用类对象的方式实现自动解析目录项数据

```
class Directory_entry():
    def __init__(self,data):
        self.data = data
        if data[0xb] == 0xf:
            self.long_filename()
            self.type = 1
        else:
            self.short_filename()
            self.type = 0

    def long_filename(self):
        self.final = (self.data[0] >> 6) & 1
        self.num = self.data[0] & 0x1f
        self.name = self.data[1:0xb].decode("utf-16")+self.data[0xe:0x1a].decode("utf-16")+self.data[-4:].decode("utf-16")

    def short_filename(self):
        self.name = self.data[:8].decode()
        self.typename = self.data[8:11].decode()
        self.cluster_num =
        struct.unpack('<I',self.data[0x1a:0x1c]+self.data[0x14:0x16])[0]
        self.length = struct.unpack('<I',self.data[-4:])[0]
```

然后便要想办法确认两个关键点，扇区大小，簇大小。

扇区大小的通过被抹去的DBR扇区数据大小以及RRaA对应的一个信息扇区可以确定就是200字节。（FAT32文件系统在DBR的保留扇区中安排了一个文件系统信息扇区，用以记录数据区中空闲簇的数量及下一个空闲簇的簇号，该扇区一般在分区的1号扇区，也就是紧跟着DBR后的一个扇区，其内容如下：）

00004129280	52	52	61	41	00	00	00	00	00	00	00	00	00	00	00	00	00	RRaA
00004129296	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004129312	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004129328	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004129344	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004129360	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004129376	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004129392	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004129408	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004129424	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004129440	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004129456	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004129472	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004129488	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004129504	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004129520	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004129536	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004129552	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004129568	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004129584	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004129600	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004129616	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004129632	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004129648	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004129664	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004129680	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004129696	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004129712	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004129728	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004129744	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004129760	00	00	00	00	72	72	41	61	9E	07	02	00	8F	01	00	00	rrAaI	
00004129776	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	55	AA	U8

现在只需要知道一个簇有几个扇区，重要信息就全部掌握了。一般通过逆推的方式求簇中有几个扇区。

这里因为根目录有 `flag.zip`, 所以我们通过特征值搜索的方式找到它的地址。

644D169	flag	?	?
1A21601E	flag	?	?
1A2160AA	flag	?	?

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
1A216000	50	4B	03	04	15	00	01	00	09	00	33	B4	AA	52	B0	93	PK.....3`@R°
1A216010	EE	A0	5A	00	00	00	6D	01	00	00	04	00	00	00	66	6C	i Z...m.....fl
1A216020	61	67	D0	02	06	1C	4D	E2	73	57	23	58	39	36	F7	6F	agD...MåsW#X96÷o
1A216030	19	8B	16	7B	C8	F3	2E	9A	EC	CA	A1	90	F3	04	37	8C	.I.{Éó.IiÉi.6.7
1A216040	13	E6	3C	CD	03	F7	90	BA	CE	72	4E	78	2E	E7	4C	59	.æ<í.-.ºÍrNx.çLY
1A216050	8A	1C	74	66	F4	FC	DB	79	07	3F	21	CE	51	05	E1	E7	I.tfóüÜy.?ÍQ.áç
1A216060	1D	4E	BF	16	44	FB	78	CE	FC	5B	3C	7F	16	F5	DA	36	.Nç.DúxÍü[<.8Ú6
1A216070	67	12	3A	C2	01	50	A6	2E	FB	1A	2E	DD	50	4B	01	02	g.:Å.P .ù..ÝPK..
1A216080	3F	00	15	00	01	00	09	00	33	B4	AA	52	B0	93	EE	A0	?.....3`@R° i
1A216090	5A	00	00	00	6D	01	00	00	04	00	24	00	00	00	00	00	Z...m.....\$.....
1A2160A0	00	00	20	00	00	00	00	00	00	00	66	6C	61	67	0A	00flag..

根据上述关系式代入计算。可得：

$$(0x1a216000 - 0xfe000) / (0x1a11a - 2) = 4096 = 0x1000$$

可知，一簇为八个扇区，刚好是默认值。至此，即可通过脚本自动提取文件：

```

import struct
import os

FAT1_addr = 0x4400 # FAT1 起始地址
FAT1_size = 999 # FAT1 占用的扇区数量
cluster = 8 # 一簇为8扇区
chunk_size = 0x200 # 一扇区为 200 bytes

class Directory_entry():
    def __init__(self,data):
        self.data = data
        if data[0xb] == 0xf:
            self.long_filename()
            self.type = 1
        else:
            self.short_filename()
            self.type = 0

    def long_filename(self):
        self.final = (self.data[0] >> 6) & 1
        self.num = self.data[0] & 0x1f
        self.name = self.data[1:0xb].decode("utf-16")+self.data[0xe:0x1a].decode("utf-16")+self.data[-4:].decode("utf-16")

    def short_filename(self):
        self.name = self.data[:8].decode()
        self.typename = self.data[8:11].decode()
        self.cluster_num =
    struct.unpack('<I',self.data[0x1a:0x1c]+self.data[0x14:0x16])[0]
        self.length = struct.unpack('<I',self.data[-4:])[0]

    # 根据簇号计算地址
    def get_cluster_addr(cluster_num):
        return (cluster_num - 2) * 0x1000 + 0xfe000

    # 采用递归方式实现自动遍历目录提取文件
    def files_entry(f,files_path,cluster_num):
        f.seek(get_cluster_addr(cluster_num),0)
        addr = 0
        tmp = f.read(0x20) ; addr += 0x20
        while tmp != b'\x00'*0x20 and tmp != b'\x00':
            directory_entry = Directory_entry(tmp)
            if directory_entry.type == 1:
                tmp_num = directory_entry.num
                filename = directory_entry.name
                while tmp_num != 1:
                    tmp_directory_entry = Directory_entry(f.read(0x20))

```

```

        addr += 0x20
        tmp_num = tmp_directory_entry.num
        filename = tmp_directory_entry.name + filename
    else:
        directory_entry = Directory_entry(f.read(0x20))
        addr += 0x20
    else:
        filename = directory_entry.name

    file_type = directory_entry.typename
    # 移除文件名中无效部分
    filename = filename.strip('\x20')
    filename = filename.strip('\x00')
    filename = filename.strip('\uffff')
    filename = filename.strip('\x20')
    filename = filename.strip('\x00')
    filename = filename.strip('\uffff')
    filename = filename.strip('\x20')
    filename = filename.strip('\x00')
    filename = filename.strip('\uffff')
    if file_type != '\x20\x20\x20' and file_type != None:
        filename += '.' + file_type

    if filename[0] == '\x2e':
        tmp = f.read(0x20) ; addr += 0x20
        continue
    elif filename[0] == '\xe5':
        filename = filename[1:] + '__deleted'

    if (directory_entry.data[11] >> 4) & 1 == 1:
        os.mkdir(files_path+'\\"'+filename)

files_entry(f,files_path+'\\"'+filename,directory_entry.cluster_num)
    f.seek(get_cluster_addr(cluster_num)+addr,0)
else:
    f.seek(get_cluster_addr(directory_entry.cluster_num),0)
    with open(files_path+'\\"'+filename,'wb') as o:
        o.write(f.read(directory_entry.length))
    f.seek(get_cluster_addr(cluster_num)+addr,0)
tmp = f.read(0x20) ; addr += 0x20

# 提取到 extra 目录下
files_path = 'extra'
os.mkdir(files_path)

with open('./raw.img','rb') as f:
    addr = 0
    tmp = f.read(0x4400) ; addr += 0x4400
    FAT1 = f.read(FAT1_size*chunk_size) ; addr += FAT1_size*chunk_size
    f.read(FAT1_size*chunk_size) ; addr += FAT1_size*chunk_size

files_entry(f,files_path,2)

```

日志分析

计算机、网络和其他IT系统生成审计跟踪记录或记录系统活动的日志。通过对这些记录的分析评估，帮助公司缓解各种风险或者发现受到的攻击行为。

日志分析主要分成两种：

- Web日志分析
- 系统日志分析

Web日志分析

日志格式类型

分析日志之前我们先了解一下日志的格式有哪些。

目前比较常见的WEB日志格式主要有两类：

- Apache的NCSA日志格式， NCSA格式分为：
 - NCSA普通日志格式 (CLF)
 - NCSA扩展日志格式 (ECLF)
- IIS的W3C日志格式

除了格式不同之外，一般的分析方法基本相似，下面用NCSA普通日志格式进行讲解。

首先，它的格式如下：

远程主机 IP	(E-mail)	(登录名)	请求时间	方法+资源+协议	状态代码	发送给客户端的字节数
192.168.153.1			[21/Apr/2019:18:48:29 +0800]	POST /admin/login.action.php HTTP/1.1	200	78

常用日志分析方法

常见的日志分析方法有两种：

1. 特征字符分析
2. 访问频率分析

特征字符分析

特征字符分析法：顾名思义，就是根据攻击者利用的漏洞特征，进行判断攻击者使用的是哪一种攻击。

常见的类型有SQL注入、XSS跨站脚本攻击、恶意文件上传、一句话木马连接等。

SQL注入

漏洞特征：存在SQL注入语句

常见的SQL注入语句有：

- 通过报错注入、布尔盲注、时间盲注判断是否存在注入：
 - 字符型
 - 参数后加单引号，报错：sql1.php?name=admin'
 - 参数后加' and '1'='2和' and '1'='2，访问正常：sql1.php?name=admin' and '1'='1 /sql1.php?name=admin' and '1'='2
 - 参数后加' and sleep(3) --，是否延迟3秒打开：sql1.php?name=admin' and/or sleep(3)--
 - 数字型
 - 参数后加单引号，报错：sql2.php?id=1'
 - 参数后加and 1=1和and 1=2，访问正常：sql2.php?id=1 and 1=1/sql2.php?id=1 and 1=2
 - 参数后加and sleep(5)，是否延迟3秒打开：sql2.php?id=1 and sleep(5)
- 通过各种注入语句进行SQL注入攻击：
 - 联合查询注入
 - union select
 - order by
 - 报错注入(常见报错注入函数)
 - floor()
 - extractvalue()
 - updatexml()
 - geometrycollection()
 - multipoint()
 - polygon()
 - multipolygon()
 - linestring()
 - multilinestring()
 - exp()
 - 常见数据库类型判断
 - ACCESS and (select count () from sysobjects)>0返回异常 and (select count () from msysobjects)>0返回异常
 - SQLSERVER and (select count () from sysobjects)>0返回正常 and (select count () from msysobjects)>0返回异常 and left(version(),1)=5%23参数5也可能是4
 - MYSQL id=2 and version()>0返回正常 id=2 and length(user())>0返回正常 id=2 CHAR(97, 110, 100, 32, 49, 61, 49)返回正常
 - Oracle and length (select user from dual)>0返回正常

上述内容并非全部，只是举出来的部分常见例子。

XSS跨站脚本攻击

漏洞特征：明显的js恶意执行代码

常见的XSS跨站脚本攻击中存在的一些代码：

- 标签
 - <script>

- <body>
- <input>
-
- <a>
- <svg>
- <BG SOUND>
- <LINK>
- <META>
- <TABLE>
- <DIV>
- <IFRAME>
- <FRAMESET>
- <STYLE>
- <OBJECT>
-

- 常用触发事件

- oninput
- onload
- oncut
- onclick
- onerror
- onmouseover
- onfocus
- onblur
- poster
- onscroll
-

- 常用恶意代码

- prompt
- confirm
- alert
- javascript
- eval
- expression
- window.location
-

这里要注意一点：由于Apache日志的特性，如果是通过Post请求，则无法准确判断出是否存在XSS跨站脚本攻击

恶意文件上传

通常存在于upload、file等出现类似字样的文件，均可能存在恶意文件上传，具体还需结合日志进行判断，一般是判断后续是否有出现Webshell等一些可以的web操作，可通过查看下图，发现在file.php页面的前后日志中，有存在一个带着日期的php页面，很可能就是利用file.php上传的文件，服务器自动生成名字，因此判断此处可能存在恶意文件上传。

```
192.168.153.1 -- [21/Apr/2019:19:10:55 +0800] "GET / HTTP/1.1" 200 7523
192.168.153.1 -- [21/Apr/2019:19:11:10 +0800] "GET /admin/ HTTP/1.1" 302 3
192.168.153.1 -- [21/Apr/2019:19:11:10 +0800] "GET /admin/login.php HTTP/1.1" 200 2060
192.168.153.1 -- [21/Apr/2019:19:11:15 +0800] "POST /admin/login.action.php HTTP/1.1" 302 -
192.168.153.1 -- [21/Apr/2019:19:11:15 +0800] "GET /admin/index.php HTTP/1.1" 200 783
192.168.153.1 -- [21/Apr/2019:19:11:15 +0800] "GET /admin/header.php HTTP/1.1" 200 3020
192.168.153.1 -- [21/Apr/2019:19:11:15 +0800] "GET /admin/menu.php HTTP/1.1" 200 3541
192.168.153.1 -- [21/Apr/2019:19:11:15 +0800] "GET /admin/category.php HTTP/1.1" 200 2911
192.168.153.1 -- [21/Apr/2019:19:11:15 +0800] "GET /admin/images/logo.gif HTTP/1.1" 200 926
192.168.153.1 -- [21/Apr/2019:19:11:15 +0800] "GET /admin/images/ico_03.gif HTTP/1.1" 200 169
192.168.153.1 -- [21/Apr/2019:19:11:15 +0800] "GET /admin/images/img_03.gif HTTP/1.1" 200 74
192.168.153.1 -- [21/Apr/2019:19:11:15 +0800] "GET /admin/images/img_04.gif HTTP/1.1" 200 143
192.168.153.1 -- [21/Apr/2019:19:11:15 +0800] "GET /admin/images/img_07.gif HTTP/1.1" 200 202
192.168.153.1 -- [21/Apr/2019:19:11:15 +0800] "GET /include/js/jquery.js HTTP/1.1" 200 55805
192.168.153.1 -- [21/Apr/2019:19:11:15 +0800] "GET /admin/images/img_09.gif HTTP/1.1" 200 169
192.168.153.1 -- [21/Apr/2019:19:11:22 +0800] "GET /admin/file.php HTTP/1.1" 200 4821
192.168.153.1 -- [21/Apr/2019:19:11:22 +0800] "GET /attachment/201807/20180718100338_42.php HTTP/1.1" 200 -
192.168.153.1 -- [21/Apr/2019:19:11:22 +0800] "GET /admin/images/del.gif HTTP/1.1" 200 203
192.168.153.1 -- [21/Apr/2019:19:11:22 +0800] "GET /admin/images/img_10.gif HTTP/1.1" 200 162
192.168.153.1 -- [21/Apr/2019:19:11:22 +0800] "GET /admin/admin/images/20070907_03.gif HTTP/1.1" 404 232
192.168.153.1 -- [21/Apr/2019:19:21:49 +0800] "POST /admin/file.action.php HTTP/1.1" 302 3
192.168.153.1 -- [21/Apr/2019:19:21:52 +0800] "GET /admin/file.php HTTP/1.1" 200 5858
192.168.153.1 -- [21/Apr/2019:19:22:07 +0800] "GET /admin/file.php HTTP/1.1" 200 5858
```

一般地，如果Post请求的数据未被显示出来，则需要我们通过访问的链接以及上下文的访问详情确认此处是否存在恶意文件上传

一句话木马（Webshell）

一般名字可疑的文件，如带日期字样的页面(.php、.asp、.aspx、.ash、.jsp等)、一串随机值的页面等，并且是通过Post请求，同时会返回一定的数据，此时可判断可能存在一句话木马、webshell等恶意文件，有些日志可能还有post请求参数，可结合参数，更准确地判断出是否存在一句话木马、webshell等恶意文件。

访问频率分析

访问频率分析：不难理解，就是通过查看攻击者访问的频率来判断攻击者使用的是哪一种攻击。 **这一特点分析法也常常用在流量分析中。**

常见的类型有以下：SQL盲注、敏感目录爆破、账号爆破、Web扫描。

- SQL盲注 一般访问比较有规律，基本都包含SQL语句，并且大体都相似，有个别字符不同
- 敏感目录爆破 一般会有大量的探测目录，一般以Head方法为主进行探测
- 账号爆破 通常都是对一个登录页面进行大量post请求，并且返回值基本相似
- Web扫描 一般来说，访问的目标比较离散，并且来源地址相对固定，同时访问的结果大多数也都是失败的，并且在参数中有比较明显的扫描器特征字段 常见扫描器在url上的特征：
 - AWVS 10.5或11 acunetix-wvs-test-for-some-inexistent-file by_wvs acunetix_wvs_security_test acunetix acunetix_wvs acunetix_test wvs_test
 - Netsparker netsparker Netsparker ns: netsparker
 - Appscan Appscan
 - Webinspect HP404
 - Rsas nsfocus
 - Nessus nessus Nessus
 - Sqlmap sqlmap

系统日志分析

Linux操作系统

Linux的系统日志一般存放在/var/log目录下，常见的日志（列举部分）有以下：

日志文件	基本详情
/var/log/messages	关于Linux操作系统信息，还包括了系统启动情况等
/var/log/boot.log	系统启动日志
/var/log/lastlog	记录所有用户的近期信息，也可用lastlog命令查看具体内容
/var/log/maillog	邮件日志信息
/var/log/cron	Cron计划任务相关信息的日志
/var/log/secure	系统安全、验证以及授权信息的日志
/var/log/faillog	用户登陆失败信息，包括失败次数、错误登陆命令等
/var/log/btmp	所有登陆失败信息，包括（远程服务、IP地址等）

- /var/log/messages

用于记录系统相关信息，如执行程序、系统错误、启动信息等，一般我们会使用message进行查看可疑程序执行的可疑操作，系统在执行程序时出现错误等，

```

Sep 2 18:33:15 xibai-kali gdm3: Gdm: Child process -729 was already dead.
Sep 2 18:33:18 xibai-kali gnome-shell[1375]: Ignoring length property that isn't a number at line 1348, col 24
Sep 2 18:33:19 xibai-kali gnome-shell[1375]: Ignoring excess values in shadow definition
Sep 2 18:33:19 xibai-kali gnome-shell[1375]: Ignoring excess values in shadow definition
Sep 2 18:33:19 xibai-kali gnome-shell[1375]: Ignoring excess values in shadow definition
Sep 2 18:33:19 xibai-kali gnome-shell[1375]: Ignoring excess values in shadow definition
Sep 2 18:33:24 xibai-kali gnome-shell[1375]: Ignoring excess values in shadow definition
Sep 2 18:33:24 xibai-kali gnome-shell[1375]: Ignoring excess values in shadow definition
Sep 2 18:33:24 xibai-kali gnome-shell[1375]: Ignoring excess values in shadow definition
Sep 2 18:33:24 xibai-kali gnome-shell[1375]: Ignoring excess values in shadow definition
Sep 2 18:33:25 xibai-kali pipewire-media-session[741]: error id:0 seq:158 res:-32 (Broken pipe): connection error
Sep 2 18:33:25 xibai-kali tracker-miner-f[728]: Error while sending AddMatch() message: 连接已关闭
Sep 2 18:33:25 xibai-kali tracker-miner-f[728]: Error while sending AddMatch() message: 连接已关闭
Sep 2 18:33:25 xibai-kali tracker-miner-f[728]: Error while sending AddMatch() message: 连接已关闭
Sep 2 18:33:26 xibai-kali gnome-shell[1375]: Ignoring excess values in shadow definition
Sep 2 18:33:26 xibai-kali gnome-shell[1375]: Ignoring excess values in shadow definition
Sep 2 18:33:33 xibai-kali gsd-media-keys[1569]: Failed to create proxy for screencast: 为 org.gnome.Shell .Screencast 调用 StartServiceByName 时出错：已到超时限制
Sep 2 18:33:45 xibai-kali gnome-shell[1375]: Ignoring excess values in shadow definition
Sep 2 18:33:45 xibai-kali gnome-shell[1375]: Ignoring excess values in shadow definition
Sep 2 18:33:58 xibai-kali geoclue[1440]: Service not used for 60 seconds. Shutting down..
root@xibai-kali:~/桌面#

```

对应的格式：

日期 时间 主机 执行的程序[进程ID]: 具体信息

- /var/log/boot.log

用于记录系统启动信息的日志，一般用于查看在系统启动时所有相关信息，具体如下：

```

[ OK ] Started Load/Save RF Kill Switch Status.
[ OK ] Started System Logging Service.
[ OK ] Finished Raise network interfaces.
[ OK ] Started User Login Management.
[ OK ] Started Authorization Manager.
      Starting Modem Manager...
[ OK ] Started Network Manager.
[ OK ] Reached target Network.
      Starting OpenBSD Secure Shell server...
      Starting Permit User Sessions...
[ OK ] Started Accounts Service.
[ OK ] Finished Permit User Sessions.
      Starting GNOME Display Manager...
      Starting Hold until boot process finishes up...
[ OK ] Starting Hostname Service...
[ OK ] Started Hostname Service.
      Starting Network Manager Script Dispatcher Service...
[ OK ] Started Network Manager Script Dispatcher Service.
[ OK ] Started OpenBSD Secure Shell server.
[ OK ] Started GNOME Display Manager.
root@xibai-kali:~/桌面#

```

不难发现，该日志记录的是系统启动时的启动信息，比如开启了哪些服务、做了什么操作都能一目了然。

- /var/log/lastlog

用于记录了用户近期登陆情况，直接查看lastlog，可能信息不太明显，但是也可以使用lastlog命令进行查看，会比较详细：

```
dnsmasq      **从未登录过**
sshd        **从未登录过**
postgres    **从未登录过**
usbmux      **从未登录过**
rtkit       **从未登录过**
_rpc        **从未登录过**
Debian-snmp  **从未登录过**
statd       **从未登录过**
inetsim     **从未登录过**
sshd        **从未登录过**
pulse       **从未登录过**
speech-dispatcher  **从未登录过**
avahi       **从未登录过**
saned       **从未登录过**
colord      **从未登录过**
geoclue     **从未登录过**
King-phisher  **从未登录过**
Debian-gdm   **从未登录过**
dradis      **从未登录过**
beef-xss    **从未登录过**
systemd-coredump  **从未登录过**
tcpdump     **从未登录过**
xibai      **从未登录过**
root@xibai-kali:~/桌面#
```

- /var/log/cron

Linux的计划任务相关信息的日志，我们也会使用它来找寻攻击者可能会写入的一些恶意计划任务，其中可能会带有一些恶意软件等相关信息。

- /var/log/secure

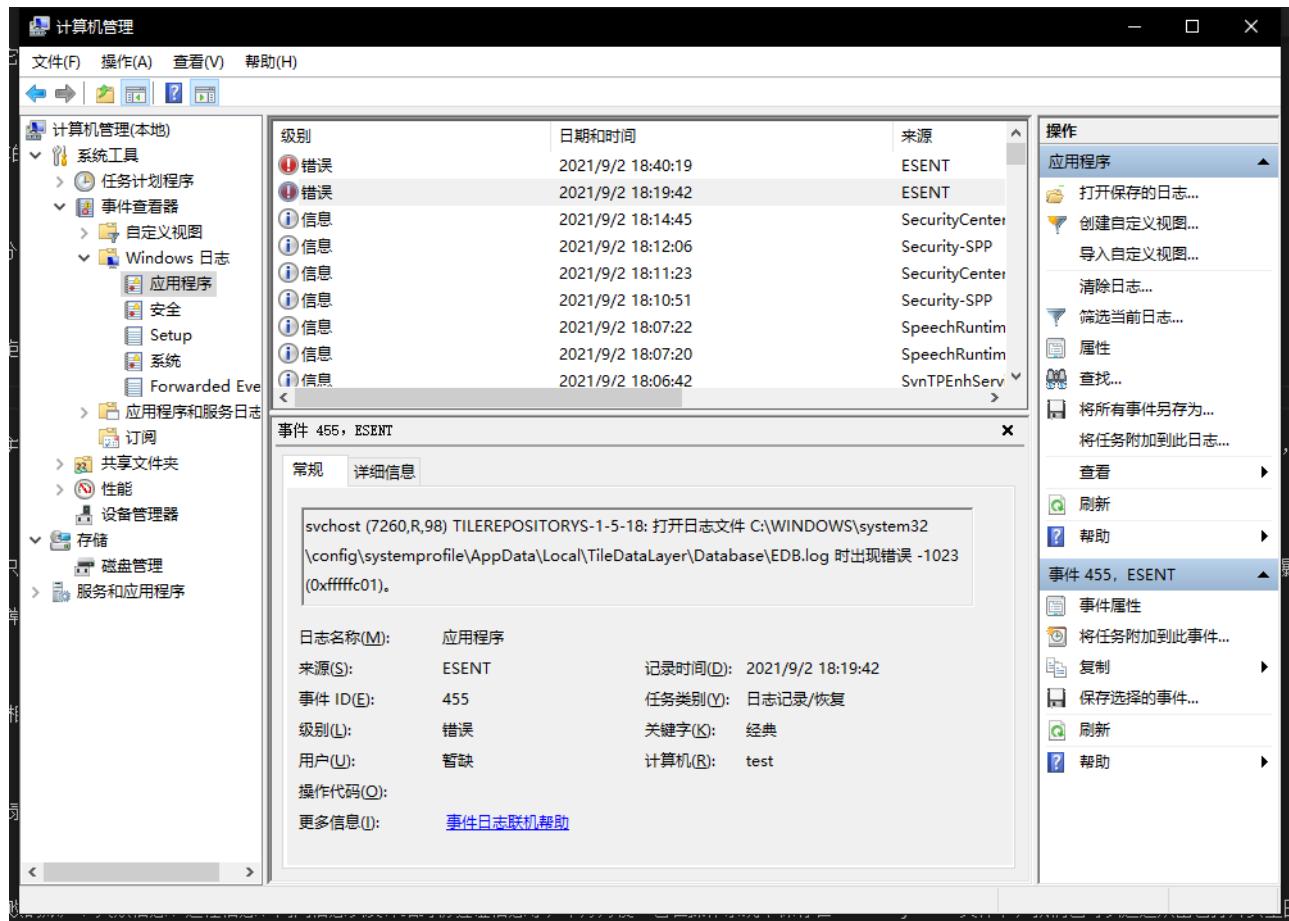
此日志是linux 的安全日志，被用于记录用户工作的安全相关问题以及登陆认证情况，

Windows操作系统

Windows日志一般在事件查看器中可以进行查看，通常分为五个：应用程序、安全、Setup、系统、转发事件。并且这五个中又以应用程序、安全以及系统日志较为常见。

- 应用程序日志

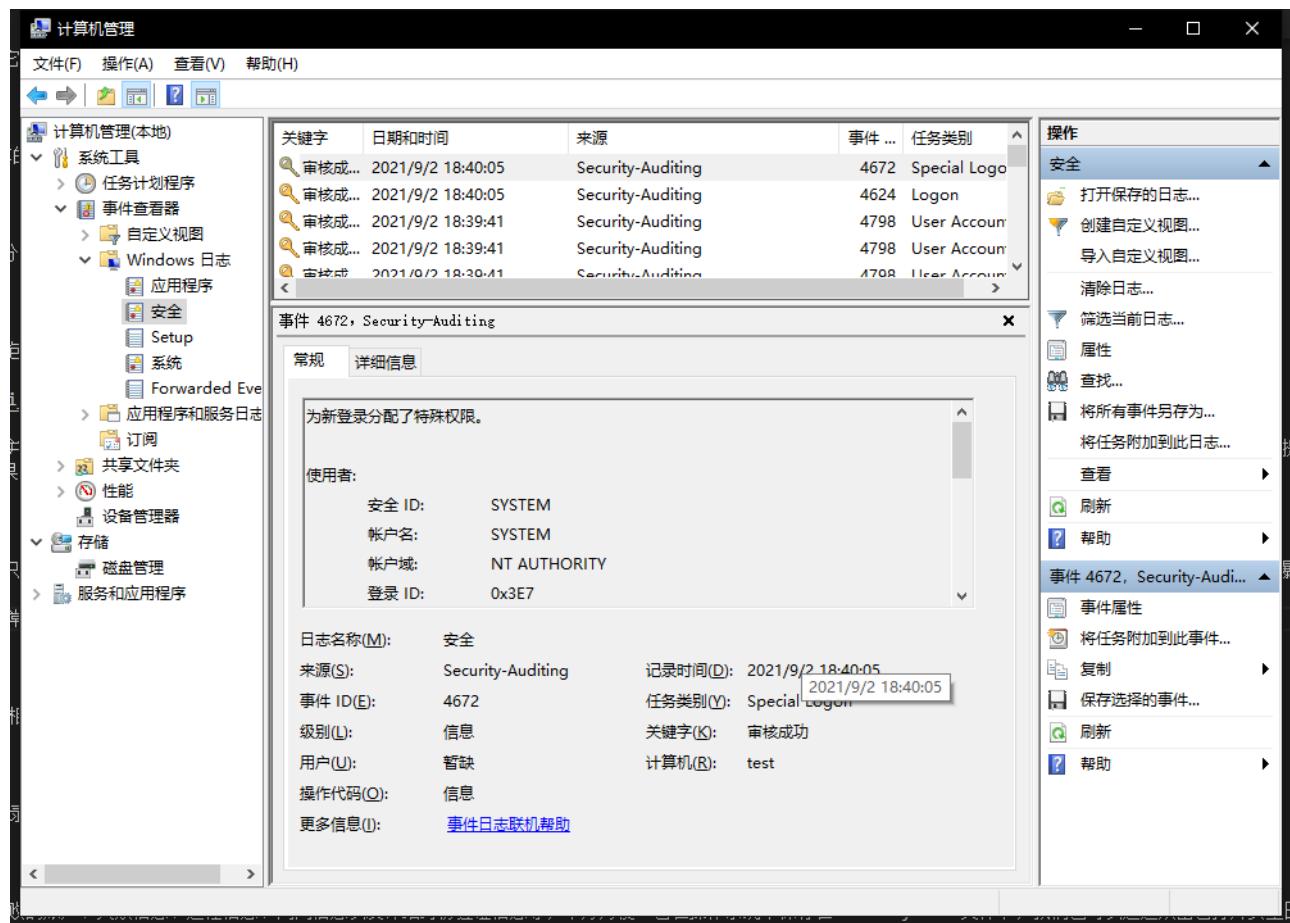
此日志顾名思义便是记录了应用程序的运行情况，包括运行出错、甚至于出错的原因，如：



它指出了错误应用程序名称、版本、具体时间错，并且还指出了错误的模块以及异常代码，故而，我们可以通过这些信息，进行对应的故障排查，具体如何排查可通过适当的资料等进行，这里不做过多说明，需要提的是它在Windows中保存在Application.evtx文件中，如果在CTF比赛中看到这个文件，那么可能就是让你进行应用程序日志分析了。

- 安全日志

此处的安全日志和Linux的安全日志相似，但是它只记录用户登陆情况、用户访问时间以及访问是否授权等，通过它我们可以轻松的发现是否存在爆破风险（一般在短时间内发现大量登陆失败，即可认为该账号被爆破了）。



- 系统日志

系统日志则是记录了操作系统安装的应用程序软件相关的事件。它包括了错误、警告及任何应用程序需要报告的信息等。

相比于Linux 的日志，Windows对于系统日志的记录，也是挺详细的，我们可以通过它来进行一些分析判断，它存在于System.evtx文件中。

它详细到可以发现使用者信息、登陆类型、登陆失败的账户、失败信息、进程信息、内网信息以及详细身份验证信息等，十分方便。它在操作系统中保存在Security.evtx文件下，我们也可以通过双击它打开安全日志。