Table of Contents

Windows Event Logs Table

Log Name	Provider Name	Event IDs	Description
System		7045	A service was installed in the system
System		7030	...service is marked as an interactive service. However, the system is configured to not allow interactive services. This service may not function properly.
System		1056	Create RDP certificate
Security		7045, 10000, 10001, 10100, 20001, 20002, 20003, 24576, 24577, 24579	Insert USB
Security		4720	A user account was created
Security		4722	A user account was enabled
Security		4724, 4738	Additional user creation events
Security		4728	A member was added to a security-enabled global group
Security		4732	A member was added to a security-enabled local group
Security		4740, 4767	User account locked out, unlocked
Security		4624, 4647	Successful logon, user initiated logoff
Security		4625	Logon failure
Security		4778, 4779	RDP reconnected, disconnected

Security		4800, 4801	Workstation locked, workstation unlocked
Security	Logon types	2	Interactive
Security	Logon types	3	Network (i.e. mapped drive)
Security	Logon types	4	Batch (i.e. schedule task)
Security	Logon types	5	Service (i.e. service startup)
Security	Logon types	7	Unlock (i.e. unnattended workstation with password protected screen saver)
Security	Logon types	8	Network Cleartext (Most often indicates a logon to IIS with “basic authentication”)
Security	Logon types	10	Remote Desktop
Security	Logon types	11	Logon with cached credentials
Security		1102	Clear Event log
Application	EMET	2	EMET detected ... mitigation and will close the application: ...exe
Firewall		2003	Disable firewall
Microsoft-Windows-AppLocker/EXE and DLL		8003	(EXE/MSI) was allowed to run but would have been prevented from running if the AppLocker policy were enforced
Microsoft-Windows-AppLocker/EXE and DLL		8004	(EXE/MSI) was prevented from running.
Microsoft-Windows-WindowsDefender/Operational		1116	Windows Defender has detected malware or other potentially unwanted software

Microsoft-Windows-WindowsDefender/Operational		1117	Windows Defender has taken action to protect this machine from malware or other potentially unwanted software

Get-WinEvent

View all events in the live system Event Log:

```
Get-WinEvent -LogName system
```

View all events in the live security Event Log (requires administrator PowerShell):

```
Get-WinEvent -LogName security
```

View all events in the file example.evtx, format list (fl) output:

```
Get-WinEvent -Path example.evtx | fl
```

View all events in example.evtx, format GridView output:

```
Get-WinEvent -Path example.evtx | Out-GridView
```

Perform long tail analysis of example.evtx:

```
Get-WinEvent -Path example.evtx | Group-Object id -NoElement | sort count
```

Pull events 7030 and 7045 from system.evtx:

```
Get-WinEvent -FilterHashtable @{Path="system.evtx"; ID=7030,7045}
```

Same as above, but use the live system event log:

```
Get-WinEvent -FilterHashtable @{logname="system"; id=7030,7045}
```

Search for events containing the string "USB" in the file system.evtx:

```
Get-WinEvent -FilterHashtable @{Path="system.evtx"} | Where {$_.Message -like
"*USB*"}
```

'grep'-style search for lines of events containing the case insensitive string "USB" in the file system.evtx:

```
Get-WinEvent -FilterHashtable @{Path="system.evtx"} | fl | findstr /i USB
```

Pull all errors (level=2) from application.evtx:

```
Get-WinEvent -FilterHashtable @{Path="application.evtx"; level=2}
```

Pull all errors (level=2) from application.evtx and count the number of lines ('wc'-style):

```
Get-WinEvent -FilterHashtable @{Path="application.evtx"; level=2} | Measure-Object -Line
```

AppLocker

Pull all AppLocker logs from the live AppLocker event log (requires Applocker):

```
Get-WinEvent -logname "Microsoft-Windows-AppLocker/EXE and DLL"
```

Search for live AppLocker EXE/MSI block events: "(EXE) was prevented from running":

```
Get-WinEvent -FilterHashtable @{logname="Microsoft-Windows-Applocker/EXE and DLL"; id=8004}
```

Search for live AppLocker EXE/MSI audit events: "(EXE) was allowed to run but would have been prevented from running if the AppLocker policy were enforced":

```
Get-WinEvent -FilterHashtable @{logname="Microsoft-Windows-Applocker/EXE and DLL"; id=8003}
```

EMET

Pull all EMET logs from the live Application Event log (requires EMET):

```
Get-WinEvent -FilterHashtable @{logname="application"; providername="EMET"}
```

Pull all EMET logs from a saved Application Event log (requires EMET):

```
Get-WinEvent -FilterHashtable @{path="application.evtx"; providername="EMET"}
```

Sysmon

Pull all Sysmon logs from the live Sysmon Event log (requires Sysmon and an admin PowerShell):

```
Get-WinEvent -LogName "Microsoft-Windows-Sysmon/Operational"
```

Pull Sysmon event ID 1 from the live Sysmon Event log

```
Get-WinEvent -FilterHashtable @{logname="Microsoft-Windows-Sysmon/Operational"; id=1}
```

Windows Defender

Pull all live Windows Defender event logs

```
Get-WinEvent -FilterHashtable @{logname="Microsoft-Windows-Windows
Defender/Operational"}
```

Pull Windows Defender event logs 1116 and 1117 from the live event log

```
Get-WinEvent -FilterHashtable @{logname="Microsoft-Windows-Windows
Defender/Operational";id=1116,1117}
```

Pull Windows Defender event logs 1116 (malware detected) and 1117 (malware blocked) from a saved evtx file

```
Get-WinEvent -FilterHashtable @{path="WindowsDefender.evtx";id=1116,1117}
```

Bro

Bro is an open-source network security platform that illuminates your network's activity in detail, with the stability and flexibility for production deployment at scale.

Bro reduces incoming packet streams into higher-level events and applies customizable scripts to determine the necessary course of action. This simple design allows you to configure an array of real-time alerts, execute arbitrary programs on demand, and log data for later use.

Run bro against a pcap, create bro log files in the current directory. Some of following logs files may be created, dpending on the pcap content:
conn.log dns.log
files.log http.log
irc.log packet_filter.log
ssl.log weird.log

```
$ bro -r /pcaps/virut-worm.pcap
```

Carve executables from a file:

```
$ sudo bro -r /pcaps/virut-worm.pcap /opt/bro/share/bro/file-extraction/extract.bro
$ ls -la /nsm/bro/extracted
```

Carve multiple file types: exe, txt, jpg, png, html and "other" (uses the extension .xxx):

```
$ sudo bro -r /pcaps/virut-worm.pcap /opt/bro/share/bro/file-extraction/extract-all.bro
$ ls -la /nsm/bro/extracted
```

Display x.509 issuer subjects:

```
$ bro -C -r /pcaps/normal/https/alexa-top-500.pcap
$ cat ssl.log | bro-cut issuer_subject
```

Tshark

TShark is a network protocol analyzer. It lets you capture packet data from a live network, or read packets from a previously saved capture file, either printing a decoded form of those packets to the standard output or writing the packets to a file. TShark's native capture file format is pcap format, which is also the format used by tcpdump and various other tools.

Without any options set, TShark will work much like tcpdump. It will use the pcap library to capture traffic from the first available network interface and displays a summary line on stdout for each received packet.

Read a pcap file:

```
$ tshark -r /pcaps/zeus-gameover-loader.pcap
```

Read a pcap, don't resolve names (layers 3 or 4):

```
$ tshark -nr /pcaps/zeus-gameover-loader.pcap
```

Read a pcap, use the display filter "http.request.method==GET":

```
$ tshark -r /pcaps/zeus-gameover-loader.pcap -R "http.request.method==GET"
```

Read a pcap, show TCP SYN packets not sent to port 80, don't resolve names:

```
$ tshark -r /pcaps/zeus-gameover-loader.pcap -n -R "not tcp.port==80 and tcp.flags ==
0x0002"
```

Print TCP conversations in a pcap:

```
$ tshark -n -r /pcaps/virut-worm.pcap -q -z conv,tcp
```

Print HTTP User-Agents in a pcap:

```
$ tshark -nr /pcaps/normal/http/normal-user-agent.pcap -R "http.user_agent" -Tfields -e
http.user_agent
```

Print X.509 certificates in a pcap:

```
$ tshark -r /pcaps/normal/https/alexa-top-500.pcap -T fields -R "ssl.handshake.certificate"
-e x509sat.printableString
```

Linux

awk

Print the length of each line of a file (/etc/passwd in this case), followed by the line itself:

```
$ cat /etc/passwd | awk '{print length, $0;}'
```

Print the 2nd field from a file using the string 'Mozilla/' as a delimiter:

```
$ cat /var/log/apache2/access.log | awk -F "Mozilla/" '{print $2}'
```

Print the last period delimited field

```
$ cat domains.txt | awk -F "." '{print $(NF)}'
```

checksum tools

Generate the MD5 checksum of a file:

```
$ md5sum /etc/passwd
```

Generate the SHA1 checksum of a file. The three following commands are equivalent:

```
$ sha1sum /etc/passwd
$ shasum /etc/passwd
$ shasum -a1 /etc/passwd
```

Generate the SHA-256 checksum of a file:

```
$ shasum -a256 /etc/passwd
```

Generate the SHA-512 checksum of a file:

```
$ shasum -a512 /etc/passwd
```

cut

Cut the 2nd field from a file, using the space as a delimiter:

```
$ cat /var/log/dpkg.log | cut -d' ' -f2
```

Cut the 6th field from a file, using the colon as a delimiter:

```
$ cat /etc/passwd | cut -d: -f6
```

Cut the 2nd and 3rd field from a file, use the comma as a delimiter:

```
$ cat /labs/honeytokens/pilots.csv | cut -d, -f2-3
```

Cut beginning at the 7th field, to end of line, using the space as a delimiter:

```
$ cat /var/log/dpkg.log | cut -d' ' -f7-
```

Cut the 6th field, using the double-quote (") as a delimiter, and escaping it to treat it as a literal character:

```
$  cat  /var/log/apache2/access.log  | cut -d\" -f6
```

Cut the beginning at the 11th character, to end of line:

```
$ ifconfig | cut -c11-
```

file

Determine the file type, using the file's magic bytes:

```
$ file /usr/local/bin/*
```

grep

Search for lines containing the string "bash", case sensitive:

```
$ grep bash /etc/passwd
```

Search for lines containing the string "bash", case insensitive:

```
$ grep -i bash /etc/passwd
```

Search for lines that do not contain the string "bash", case insensitive:

```
$ grep -vi bash /etc/passwd
```

Search for lines containing the string "root", case sensitive, plus print the next 5 lines:

```
$ grep -A5 root /etc/passwd
```

head

Print the first 10 lines of a file:

```
$ head -n 10 /etc/passwd
```

sed

grep for lines containing "Mozilla", then change "Mozilla" to "MosaicKilla":

```
$ grep Mozilla /var/log/apache2/access.log | sed "s/Mozilla/MosaicKilla/g"
```

grep for lines containing "Mozilla", then delete all characters up to and including "Mozilla":

```
$ grep Mozilla /var/log/apache2/access.log | sed "s/^.*Mozilla//g"
```

grep for lines containing "Mozilla", then delete all characters that precede "Mozilla":

```
$ grep Mozilla /var/log/apache2/access.log | sed "s/^.*Mozilla/Mozilla/g"
```

sort

The following examples will run strings on a file, search for user-agent (ignore case), and use various sort options

Simple alphabetic sort (may include duplicates)

```
$ strings /pcaps/fraudpack.pcap | grep -i user-agent | sort
```

Sort and unique lines. The two following sets of commands are equivalent:

```
$ strings /pcaps/fraudpack.pcap | grep -i user-agent | sort -u
$ strings /pcaps/fraudpack.pcap | grep -i user-agent | sort | uniq
```

Get a numeric count of each unique entry:

```
$ strings /pcaps/fraudpack.pcap | grep -i user-agent | sort | uniq -c
```

Get a numeric count of each unique entry, perform a numeric sort of that count:

```
$ strings /pcaps/fraudpack.pcap | grep -i user-agent | sort | uniq -c | sort -n
```

Sort and unique lines, print the length of each unique line followed by the line itself, perform a reverse numeric sort of that count:

```
$ strings /pcaps/fraudpack.pcap | grep -i user-agent | sort -u | awk '{print length, $0}'|
sort -rn
```

Sort on the the second comma separated field

```
$ cat /bonus/alexa/top-1m.csv sort -t, -k2
```

wc

Determine number of lines in a file (the flag is the letter "ell", not the number one):

```
$ wc -l /etc/passwd
```

xxd

xxd creates a hexdump, or converts a hexdump into binary. A lot of malware hex-encodes web traffic or malicious payloads (such as DOS executables) in order to avoid signature matching. Useful hex patterns to look for are 4d5a90 (the magic bytes for a DOS executable: "MZ<90>"), and "DOS mode" (444f53206d6f6465, see commands below).

xxd cannot natively handle percent-encoded hex, such as "%63%67%69%2D%62%69%6E", but can if the percent signs are removed (see below).

Convert the string "DOS mode" to hex, grouped in sets of 4 hex characters (default):

```
$ echo -n "DOS mode" | xxd
0000000: 444f 5320 6d6f 6465                  DOS mode
```

Convert the string "DOS mode" to hex, ungrouped:

```
$ echo -n "DOS mode" | xxd -g0
0000000: 444f53206d6f6465                  DOS mode
```

Convert the hex string "444f53206d6f6465" to binary:

```
$ echo 444f53206d6f6465 | xxd -r -p
DOS mode
```

Use sed to remove the percent signs from the percent-encoded hex string "%63%67%69%2D%62%69%6E", then translate to binary:

```
echo "%63%67%69%2D%62%69%6E" | sed "s/\%//g" | xxd -r -p
cgi-bin
```

Index

A

B

C

D

E

I

J

K

T

Z