Bloodhound walkthrough. A Tool for Many Tradecrafts | Pen Test Partners

By PTP Admin

Published: 2019-06-07 · Archived: 2026-04-05 15:08:43 UTC

- Related services
- Related blogs

A walkthrough on how to set up and use BloodHound

BloodHound (https://github.com/BloodHoundAD/BloodHound) is an application used to visualize active directory environments. The front-end is built on electron and the back-end is a Neo4j database, the data leveraged is pulled from a series of data collectors also referred to as ingestors which come in PowerShell and C# flavours.

It can be used on engagements to identify different attack paths in Active Directory (AD), this encompasses access control lists (ACLs), users, groups, trust relationships and unique AD objects. The tool can be leveraged by both blue and red teams to find different paths to targets. The subsections below explain the different and how to properly utilize the different ingestors.

Specifically, it is a tool I've found myself using more and more recently on internal engagements and when compromising a domain as it is a quick way to visualise attack paths and understand users' active directory properties.

For the purposes of this blog post we'll be using BloodHound 2.1.0 which was the latest version at the time of writing.

Setup

Initial setup of BloodHound on your host system is fairly simple and only requires a few components, we'll start with setup on Kali Linux, I'm using version 2019.1 which can be acquired from Kali's site here.

It can be installed by either building from source or downloading the pre-compiled binaries OR via a package manager if using Kali or other Debian based OS.

BloodHound is supported by Linux, Windows, and MacOS. BloodHound is built on neo4j and depends on it. Neo4j is a graph database management system, which uses NoSQL as a graph database.

Linux

To install on kali/debian/ubuntu the simplest thing to do is sudo apt install BloodHound, this will pull down all the required dependencies.

However if you want to build from source you need to install NodeJS and pull the git repository which can be found here: https://github.com/BloodHoundAD/BloodHound

Installing NodeJS

npm and nodejs are available from most package managers, however in in this instance we'll use Debian/Ubuntu as an example;

Using Ubuntu

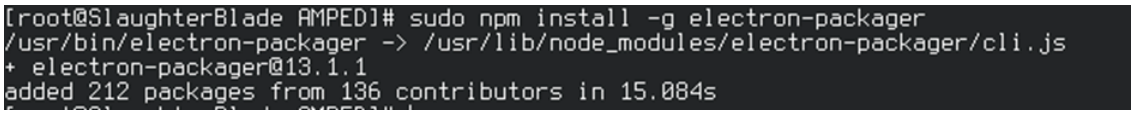
```
curl -sL https://deb.nodesource.com/setup_12.x | sudo -E bash -
sudo apt-get install -y nodejs
```

Using Debian, as root

```
curl -sL https://deb.nodesource.com/setup_12.x | bash -
apt-get install -y nodejs
```

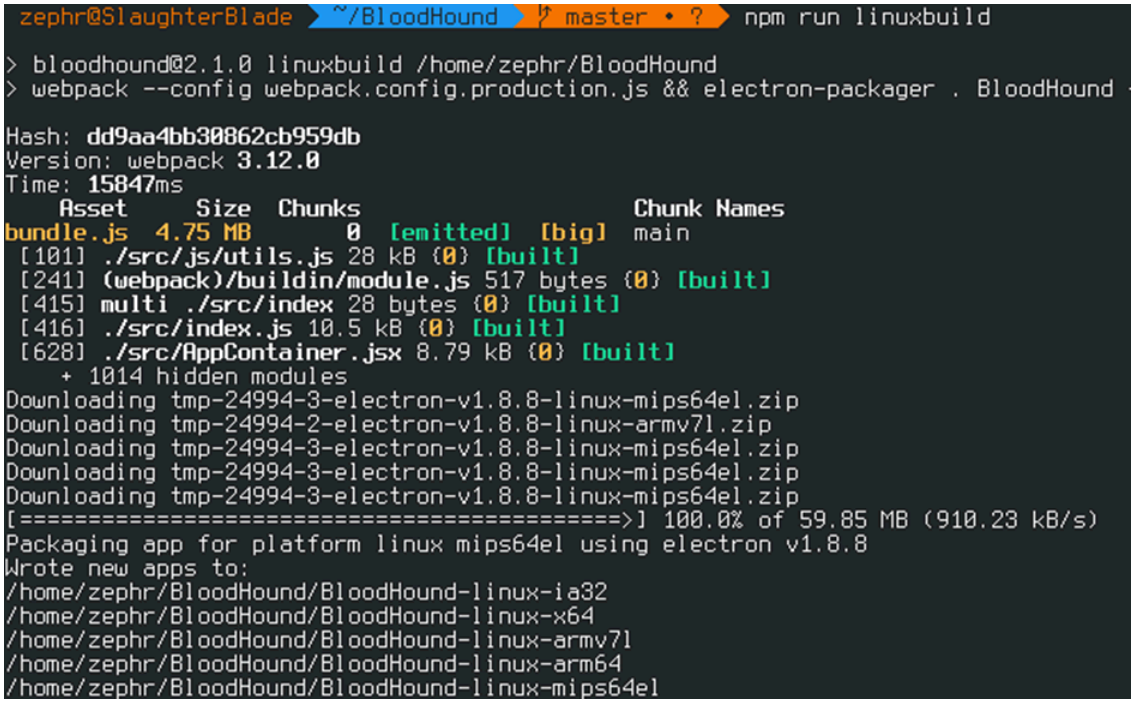
Once node has been installed, you should be able to run npm to install other packages, BloodHound requires electron-packager as a pre-requisite, this can be acquired using the following command:

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

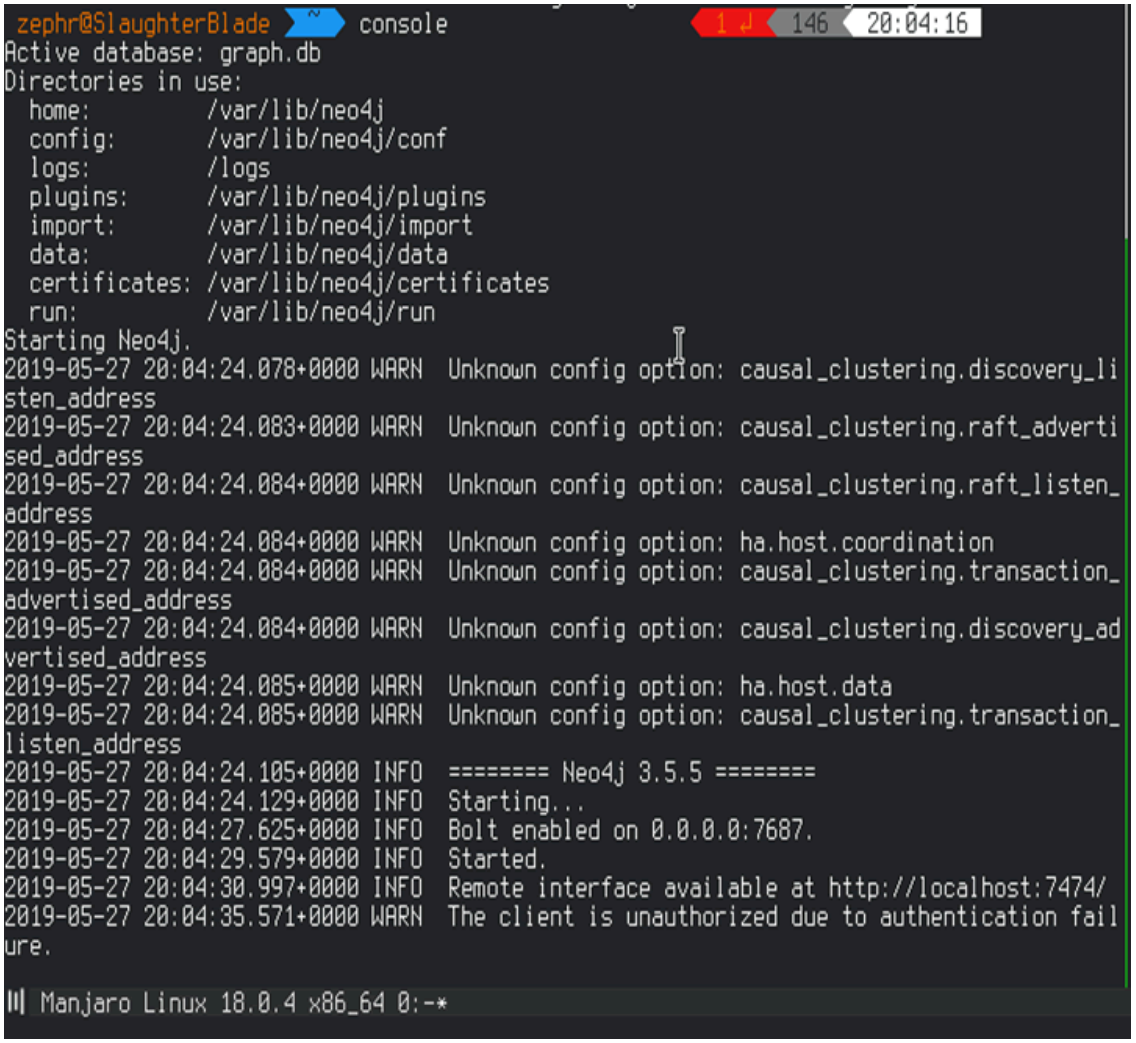


Then clone down the BloodHound from the GitHub link above then run npm install

When this has completed you can build BloodHound with npm run linuxbuild



All going well you should be able to run neo4j console and BloodHound:



MacOS

The setup for MacOS is exactly the same to Linux, except for the last command where you should run npm run macbuild instead of linuxbuilt. Then, again running neo4j console & BloodHound to launch will work. Likewise, the DBCreator tool will work on MacOS too as it is a unix base.

Windows

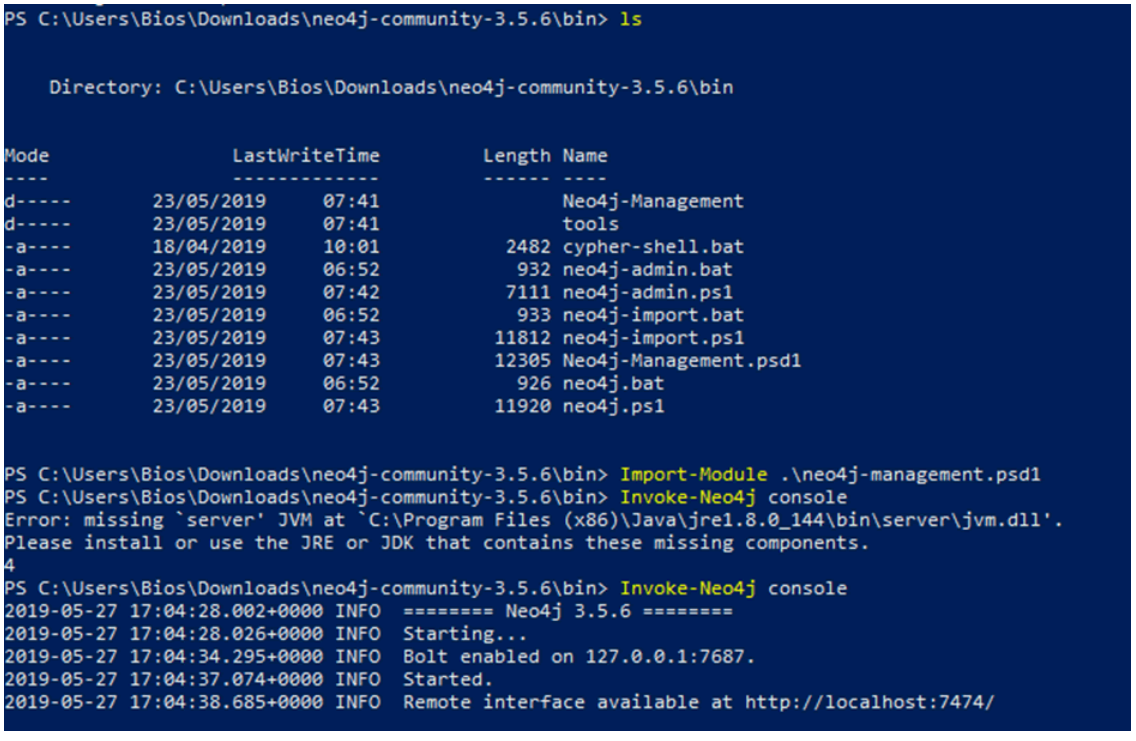
As with the Linux setup, download the repository from GitHub for BloodHound and take note of the example database file as this will be required later. Setting up on windows is similar to Linux however there are extra steps required, we'll start by installing neo4j on windows, this can be acquired from here (https://neo4j.com/download-center/#releases). Ensure you select 'Neo4J Community Server'.

By default, the download brings down a few batch files and PowerShell scripts, in order to run neo4j and BloodHound we want the management one which can be run by importing the module then running neo4j.

First open an elevated PowerShell prompt and set the execution policy:

Then navigate to the bin directory of the downloaded neo4j server and import the module then run it:

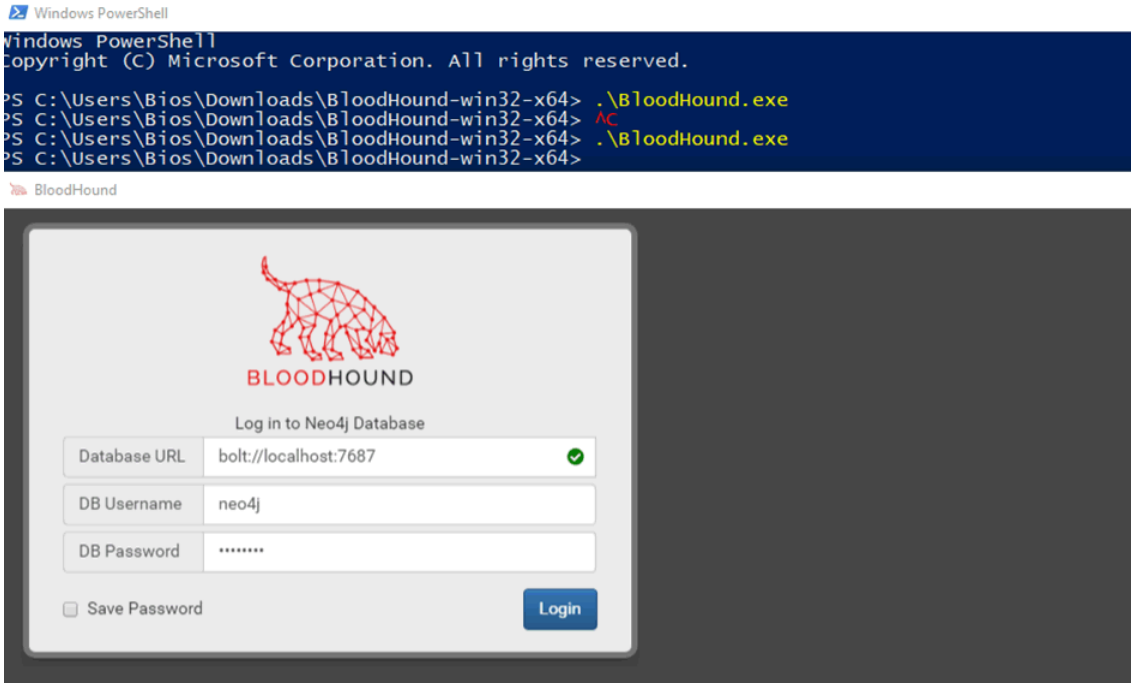
- Import-Module .\neo4j-management.psd1
- Invoke-Neo4j console



Running those commands should start the console interface and allow you to change the default password similar to the Linux stage above.

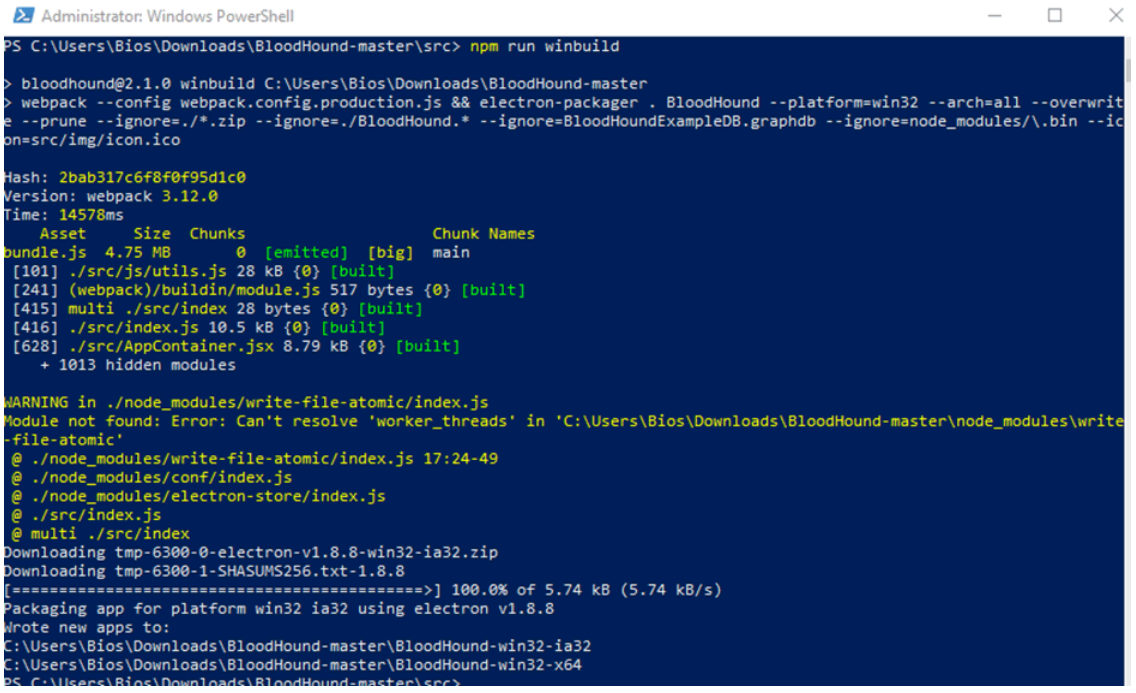
In conjunction with neo4j, the BloodHound client can also be either run from a pre-compiled binary or compiled on your host machine. If you don't want to run nodejs on your host, the binary can be downloaded from GitHub

releases (https://github.com/BloodHoundAD/BloodHound/releases) and run from PowerShell:



To compile on your host machine, follow the steps below:

1. Install NodeJS.
2. Install electron-packager npm install -g electron-packager
3. Clone the BloodHound GitHub repo git clone https://github.com/adaptivethreat/BloodHound
4. From the root BloodHound directory, run npm install
5. Build BloodHound with npm run winbuild



Then simply running BloodHound will launch the client.

Docker and BloodHound

As you've seen above it can be a bit of a pain setting everything up on your host, if you're anything like me you might prefer to automate this some more, enter the wonderful world of docker.

If you've not got docker installed on your system, you can install it by following the documentation on docker's site:

- Windows Install: https://docs.docker.com/docker-for-windows/install/
- Linux Install: https://docs.docker.com/install/linux/
- Mac Install: https://docs.docker.com/docker-for-mac/install/

Once docker is installed, there are a few options for running BloodHound on docker, unfortunately there isn't an official docker image from BloodHound's Github however there are a few available from the community, I've found belane's to be the best so far. To run this simply start docker and run:

```
docker run -it \
-p 7474:7474 \
-e DISPLAY=unix$DISPLAY \
-v /tmp/.X11-unix:/tmp/.X11-unix \
--device=/dev/dri:/dev/dri \
-v $(pwd)/data:/data \
--name bloodhound belane/bloodhound
```

This will pull down the latest version from Docker Hub and run it on your system.

Alternatively you can clone it down from GitHub: https://github.com/belane/docker-BloodHound and run yourself (instructions taken from belane's GitHub readme):

Build

```
docker build . -t Bloodhound
```

Build with example data

```
docker build . -t bloodexample --build-arg data=example
```

Optional Arguments

- **neo4j** version
- **Bloodhound** version

```
docker build . -t Bloodhound --build-arg neo4j=3.4.8 --build-arg Bloodhound=2.1.0
```

Run

```
docker run -it \
-p 7474:7474 \
-e DISPLAY=unix$DISPLAY \
-v /tmp/.X11-unix:/tmp/.X11-unix \
--device=/dev/dri:/dev/dri \
-v ~/Desktop/bloodhound/data:/data \
--name bloodhound bloodhound
```

Run with example data

```
docker run -it \
-e DISPLAY=unix$DISPLAY \
-v /tmp/.X11-unix:/tmp/.X11-unix \
--device=/dev/dri:/dev/dri \
-v ~/Desktop/bloodhound/data:/data \
--name bloodexample bloodexample
```

Start container

```
docker start Bloodhound
```

Use Login:

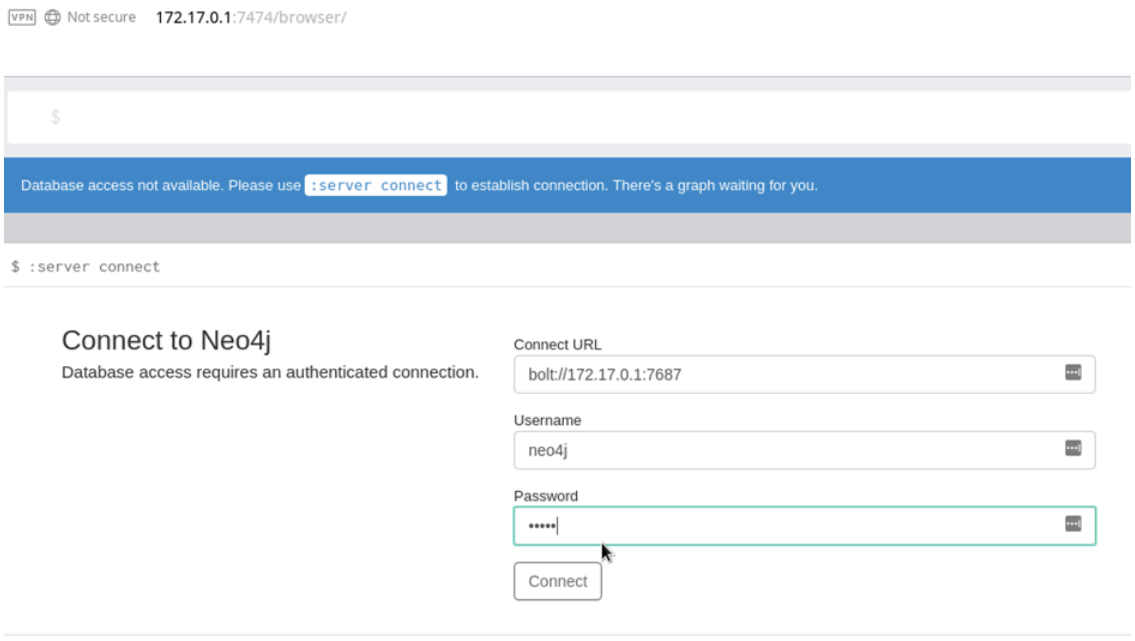
- **Database URL**: bolt://localhost:7687
- **DB Username**: neo4j
- **DB Password**: blood

In addition to BloodHound neo4j also has a docker image if you choose to build hBloodHound from source and want a quick implementation of neo4j, this can be pulled with the following command: docker pull neo4j .

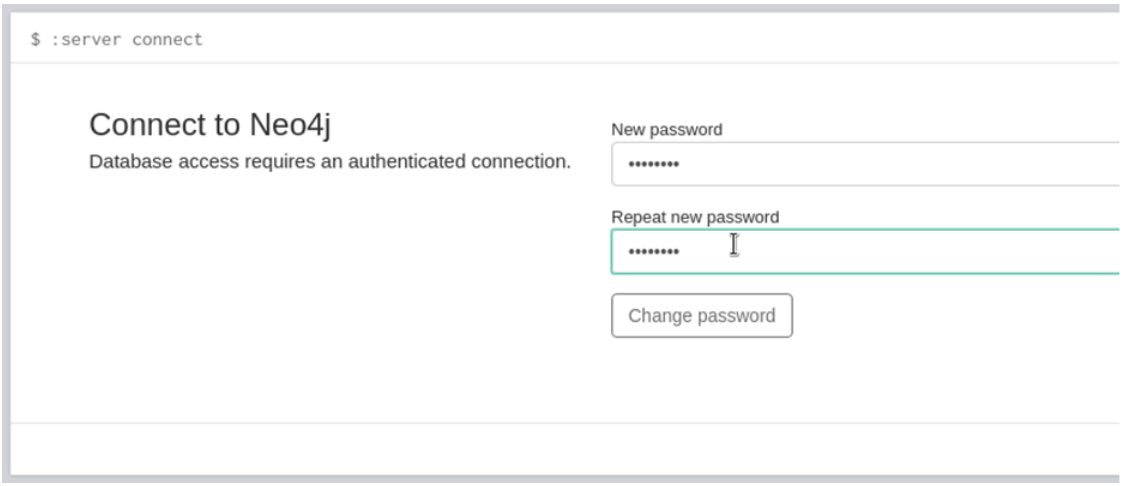
Then simply run **sudo docker run -p 7687:7687 -p 7474:7474 neo4j** to start neo4j for BloodHound as shown below:

```
zephr@SlaughterBlade ❯ /bin ❯ sudo docker run -p 7474:7474 neo4j
Alias tip: _ docker run -p 7474:7474 neo4j
Active database: graph.db
Directories in use:
  home:         /var/lib/neo4j
  config:       /var/lib/neo4j/conf
  logs:         /logs
  plugins:      /var/lib/neo4j/plugins
  import:       /var/lib/neo4j/import
  data:         /var/lib/neo4j/data
  certificates: /var/lib/neo4j/certificates
  run:          /var/lib/neo4j/run
Starting Neo4j.
2019-05-19 18:41:20.193+0000 WARN  Unknown config option: causal_clustering.discovery_listen_address
2019-05-19 18:41:20.200+0000 WARN  Unknown config option: causal_clustering.raft_advertised_address
2019-05-19 18:41:20.201+0000 WARN  Unknown config option: causal_clustering.raft_listen_address
2019-05-19 18:41:20.201+0000 WARN  Unknown config option: ha.host.coordination
2019-05-19 18:41:20.201+0000 WARN  Unknown config option: causal_clustering.transaction_advertised_address
2019-05-19 18:41:20.201+0000 WARN  Unknown config option: causal_clustering.discovery_advertised_address
2019-05-19 18:41:20.202+0000 WARN  Unknown config option: ha.host.data
2019-05-19 18:41:20.202+0000 WARN  Unknown config option: causal_clustering.transaction_listen_address
2019-05-19 18:41:20.226+0000 INFO  ======== Neo4j 3.5.5 ========
2019-05-19 18:41:20.247+0000 INFO  Starting...
2019-05-19 18:41:26.389+0000 INFO  Bolt enabled on 0.0.0.0:7687.
2019-05-19 18:41:30.466+0000 INFO  Started.
2019-05-19 18:41:33.210+0000 INFO  Remote interface available at http://localhost:7474/
```

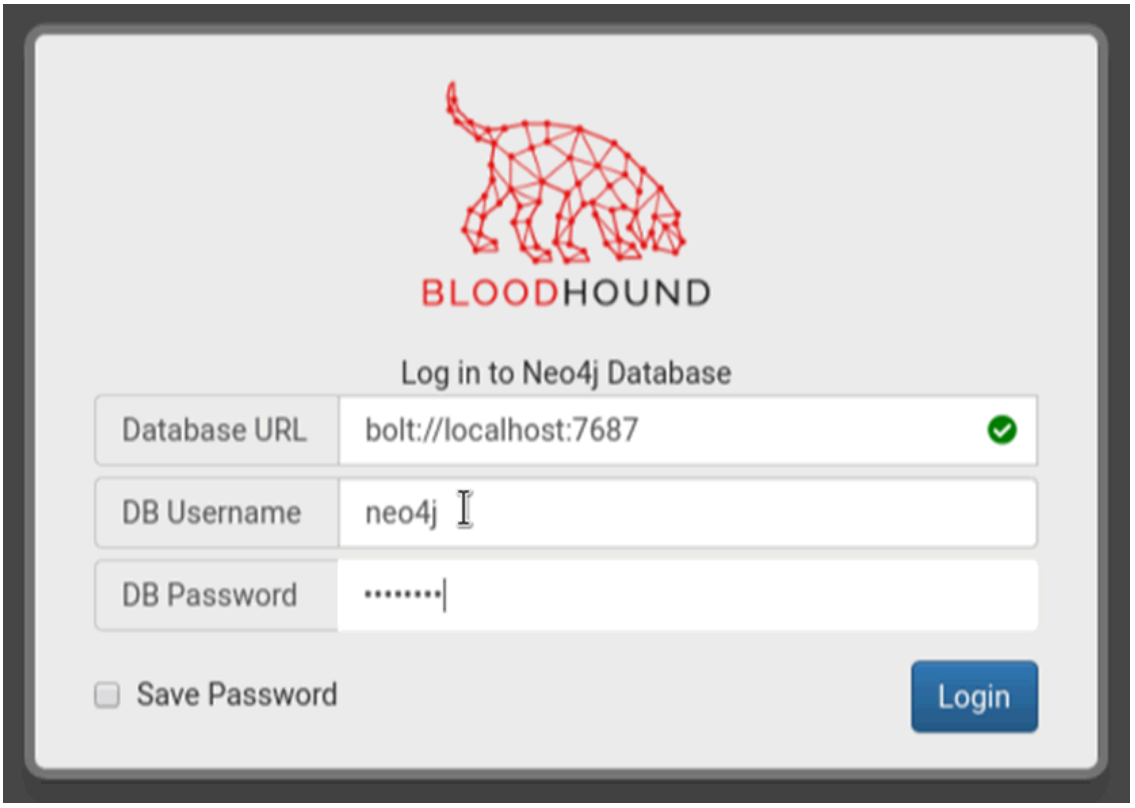
This will start neo4j which is accessible in a browser with the default setup username and password of neo4j, as you're running in docker the easiest way to access is to open a web browser and navigate to http://DOCKERIP:7474:



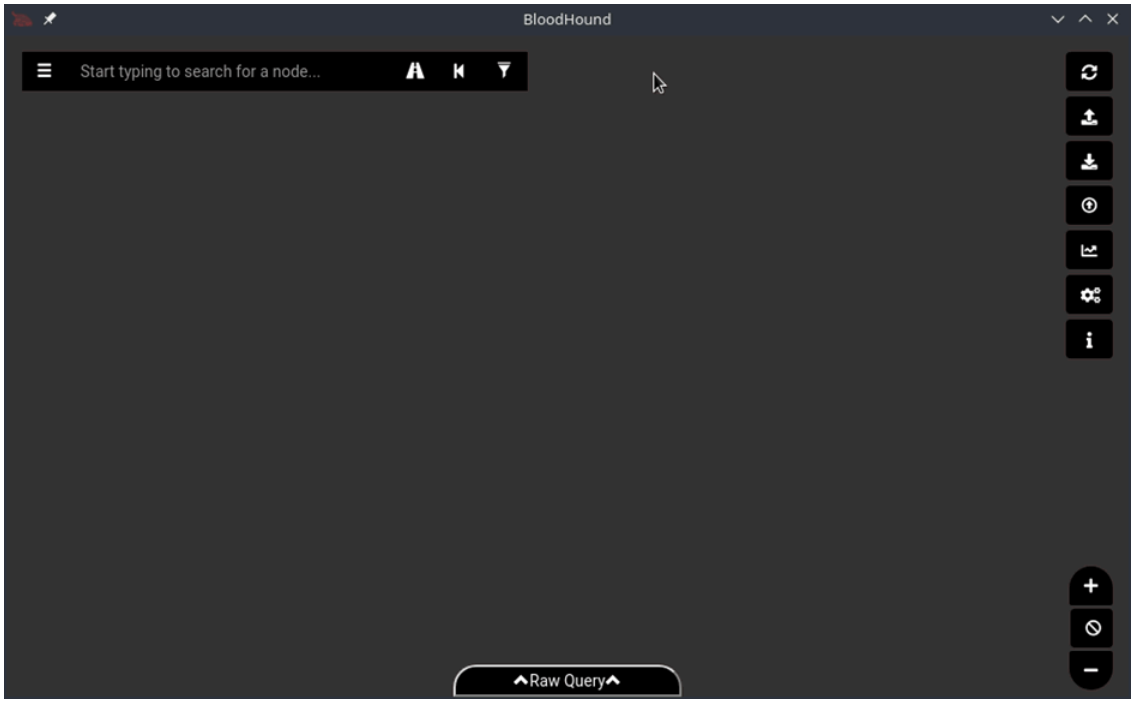
Once entering the default password, a change password prompt will prompt for a new password, make sure it's something easy to remember as we'll be using this to log into BloodHound.



Note down the password and launch BloodHound from your docker container earlier(it should still be open in the background), login with your newly created password:



The default interface will look similar to the image below, I have enabled dark mode (dark mode all the things!), by clicking on the gear icon in middle right menu bar.



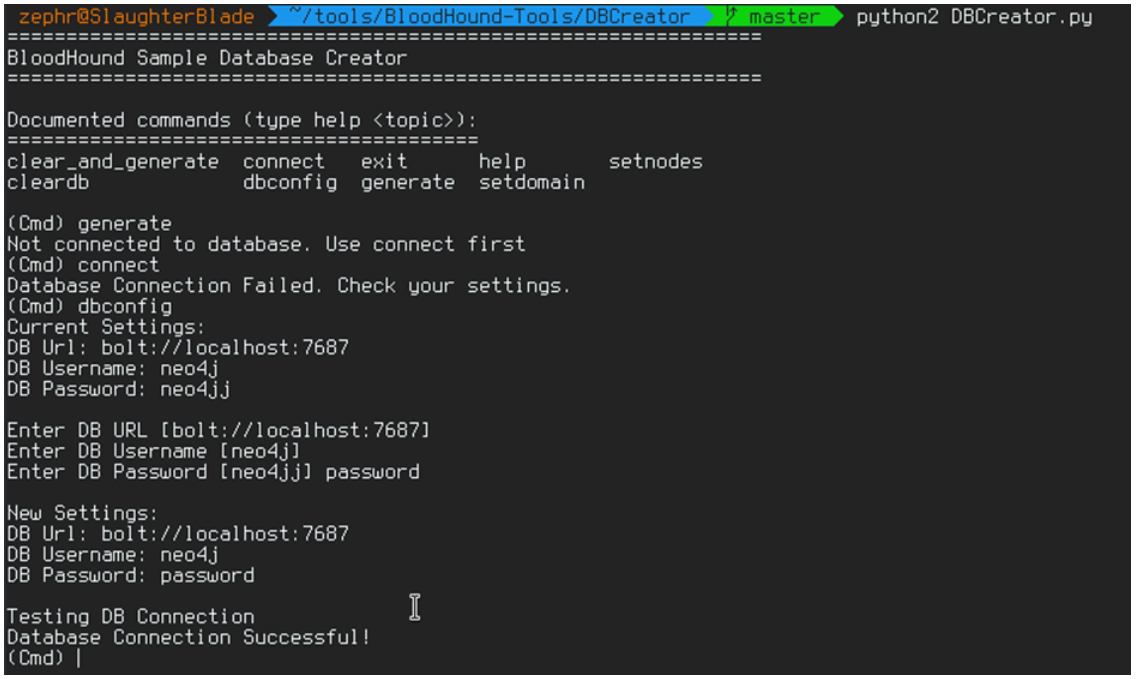
Example Data to Play With

If you want to play about with BloodHound the team have also released an example database generator to help you see what the interface looks like and to play around with different properties, this can be pulled from GitHub here(https://github.com/BloodHoundAD/BloodHound-Tools/tree/master/DBCreator)

To set this up simply clone the repository and follow the steps in the readme, make sure that all files in the repo are in the same directory.

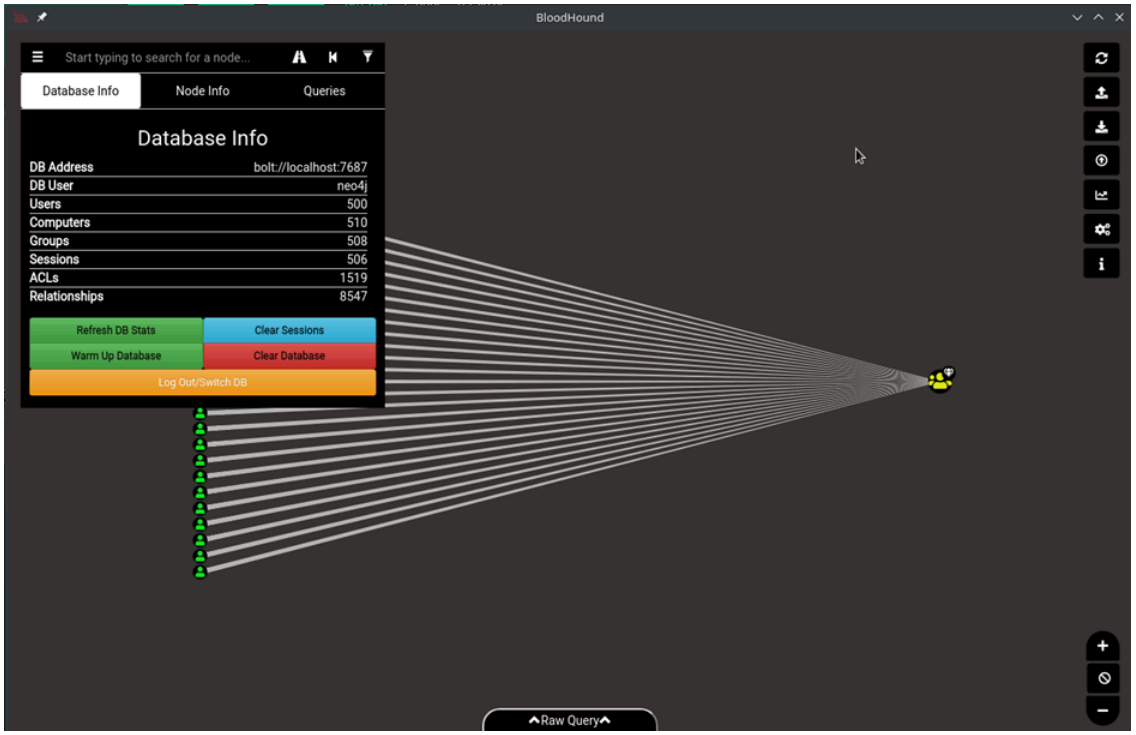
- git clone https://github.com/BloodHoundAD/BloodHound-Tools
- cd DBCreator
- pip install neo4j-driver
- sudo pip2 install neo4j
- python DBCreator.py

The tool is written in python2 so may require to be run as python2 DBCreator.py, the setup for this tooling requires your neo4j credentials as it connects directly to neo4j and adds an example database to play with.

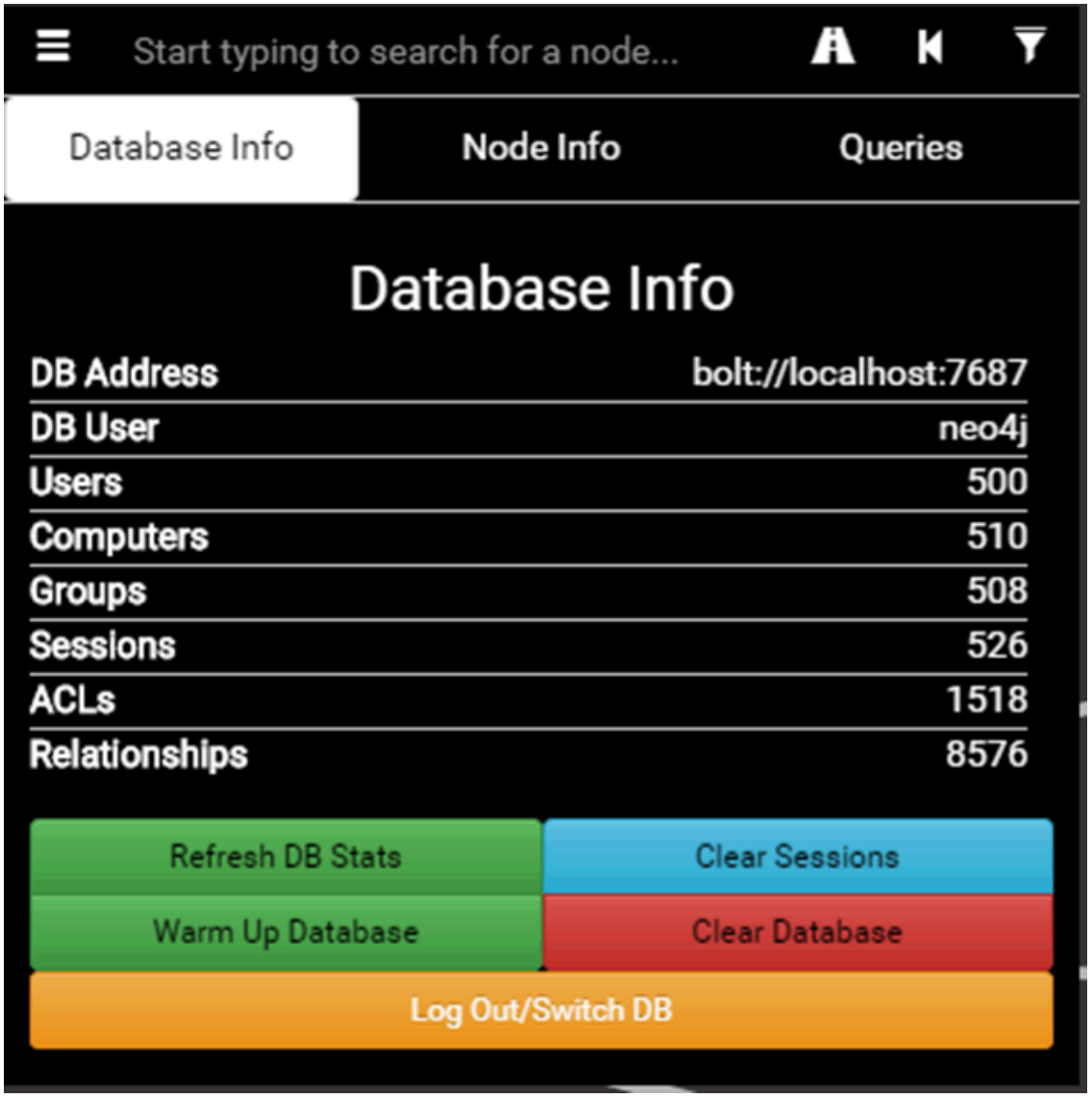


You should be prompted with a ‘Database Connection Successful’ message which assures that the tool is ready to generate and load some example data, simply use the command generate:

The generated data will be automatically loaded into the BloodHound database and can be played with using BloodHound's interface:



The view above shows all the members of the domain admins group in a simple path, in addition to the main graph the Database Info tab in the left-hand corner shows all of the stats in the database.



Explaining the different aspects of this tab are as follows:

- **Users** – The users on the network extracted from active directory.
- **Computers** – The different endpoints on the network, servers, workstations and other devices.
- **Groups** – The different AD groups extracted from AD.
- **Sessions** – The amount of user sessions on computers on the network that the ingestor has extracted (more on this later).
- **ACLs** – Access control lists, the different permissions and access that users and groups have against each other.
- **Relationships** – The different relations that all of the other aspects have to each other such as group memberships, users, user sessions and other related information.

Ingestors & Data Collection

Once you've got BloodHound and neo4j installed, had a play around with generating test data. The next stage is actually using BloodHound with real data from a target or lab network. Essentially it comes in two parts, the interface and the ingestors.

To actually use BloodHound other than the example graph you will likely want to use an ingestor on the target system or domain. Essentially these are used to query the domain controllers and active directory to retrieve all of the trust relationships, group policy settings and active directory objects.

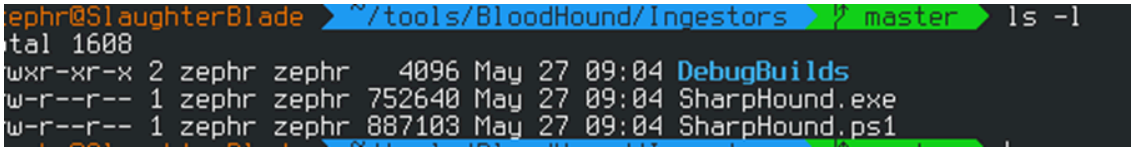
Ingestors are the main data collectors for BloodHound, to function properly BloodHound requires three key pieces of information from an Active Directory environment, these are

- What user is logged on and where?
- Which users have admin rights and what do they have access to?
- What groups do users and groups belong to?

Additionally, BloodHound can also be fed information about what AD principles have control over other users and group objects to determine additional relationships. In the majority of implementations, BloodHound does not require administrative privileges to run and therefore can act as a useful tool to identify paths to privilege escalate.

As of BloodHound 2.1 (which is the version that has been setup in the previous setup steps), data collection is housed in the form of JSON files, typically a few different files will be created depending on the options selected for data collection. Each of which contains information about AD relationships and different users and groups' permissions.

Within the BloodHound git repository (https://github.com/BloodHoundAD/BloodHound/tree/master/Ingestors) there are two different ingestors, one written in C# and a second in PowerShell which loads the C# binary via reflection. Previous versions of BloodHound had other types of ingestor however as the landscape is moving away from PowerShell based attacks and onto C#, BloodHound is following this trend.



The ingestors can be compiled using visual studio on windows or a precompiled binary is supplied in the repo, it is highly recommended that you compile your own ingestor to ensure you understand what you're running on a network. Never run an untrusted binary on a test if you do not know what it is doing.

As well as the C# and PowerShell ingestors there is also a Python based one named BloodHound.Py (https://github.com/fox-it/BloodHound.py) which needs to be manually installed through pip to function. It does not currently support Kerberos unlike the other ingestors. However, it can still perform the default data collection tasks, such as group membership collection, local admin collection, session collection, and tasks like performing domain trust enumeration.

BloodHound python can be installed via pip using the command: pip install BloodHound, or by cloning this repository and running python setup.py install. BloodHound.py requires impacket, ldap3 and dnspython to function. To use it with python 3.x, use the latest impacket from GitHub.

How to Use Sharphound

Typically when you've compromised an endpoint on a domain as a user you'll want to start to map out the trust relationships, enter Sharphound for this task. It needs to be run on an endpoint to do this, as there are two flavours (technically three if we include the python ingestor) we'll want to drop either the PowerShell version or the C# binary onto the machine to enumerate the domain.

It isn't advised that you drop a binary on the box if you can help it as this is poor operational security, you can however load the binary into memory using reflection techniques.

The syntax for running a full collection on the network is as follows, this will use all of the collection method techniques in an attempt to enumerate as much of the network as possible:

```
invoke-Bloodhound -CollectionMethod All -Domain TESTLAB.local -ZipFileName PATHTOZIP\file.zip -JsonF
```

The above command will run Sharphound to collect all information then export it to JSON format in a supplied path then compress this information for ease of import to BloodHound's client.

An overview of all of the collection methods are explained; the CollectionMethod parameter will accept a comma separated list of values. The default if this parameter is not supplied is Default:

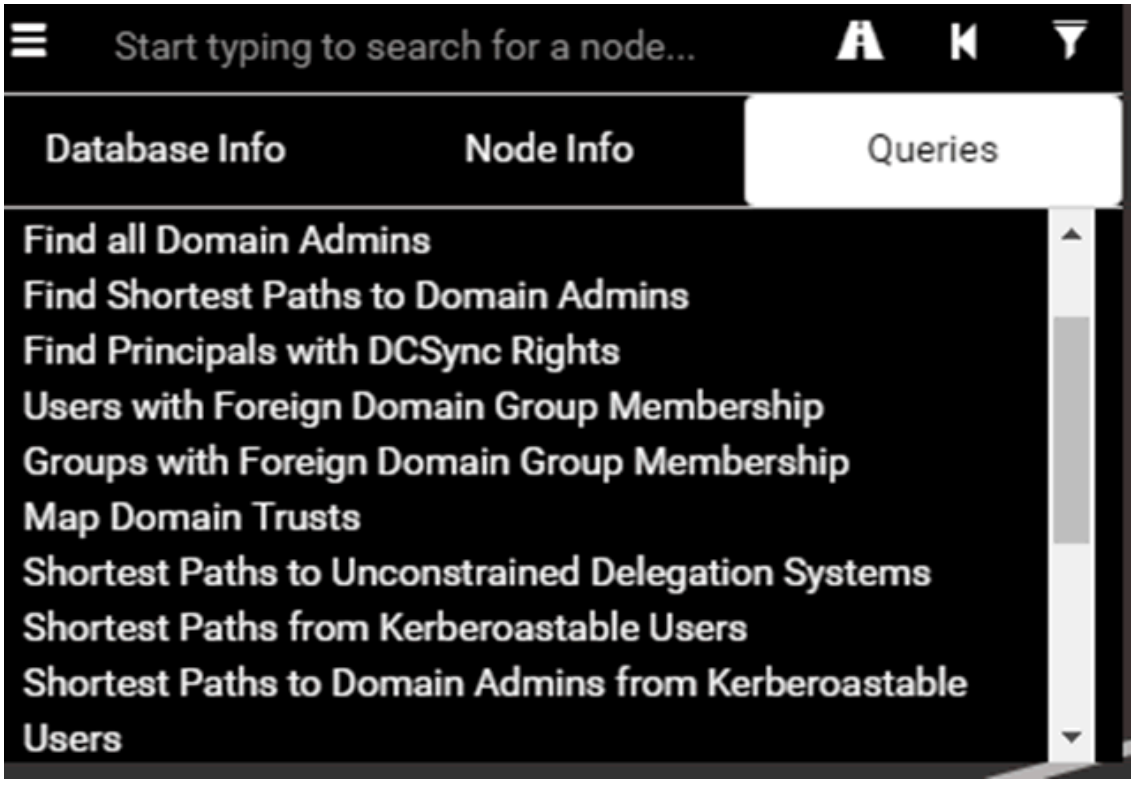
- **Default** – This performs a collection of the local admins on machines, group memberships, domain trusts, and sessions.
- **Group** – Collects the group memberships only
- **LocalGroup** – Collects just the local admins
- **GPOLocalGroup** – Performs local admin collection using Group Policy Objects
- **ComputerOnly** – Performs local admin collection and session collection
- **Session** – Collects the user sessions on machines on the domain
- **LoggedOn** – Performs privileged session collection (this requires local admin rights on target systems)
- **Trusts** – Enumerates the domain trusts for the specified target domain
- **ACL** – Collects the access control lists from the domain
- **Container** – Performs collection of Containers
- **All** – Performs all Collection Methods listed above.

For a full breakdown of the different parameters that BloodHound accepts, refer to the Sharphound repository on GitHub (https://github.com/BloodHoundAD/SharpHound). Alternatively if you want to drop a compiled binary the same flags can be used but instead of a single – a double dash is used:

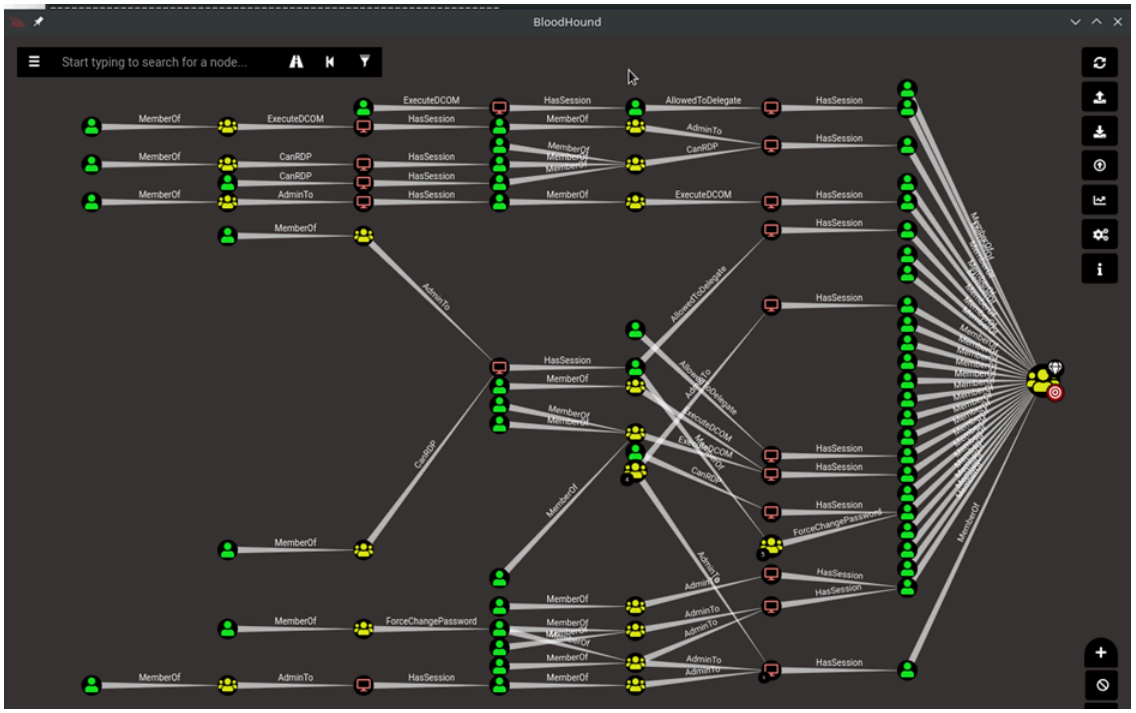
```
Sharphound.exe --ZipFileName PATHTOZIP\file.zip --JsonFolder PATHTOZIP\ --CollectionMethod All -Doma
```

Understanding What You're Looking At

When a graph is generated from the ingestors or an example dataset, BloodHound visualizes all of the relationships in the form of nodes, each node has several properties including the different ties to other nodes. Navigating the interface to the queries tab will show a list of pre-compiled built-in queries that BloodHound provides:



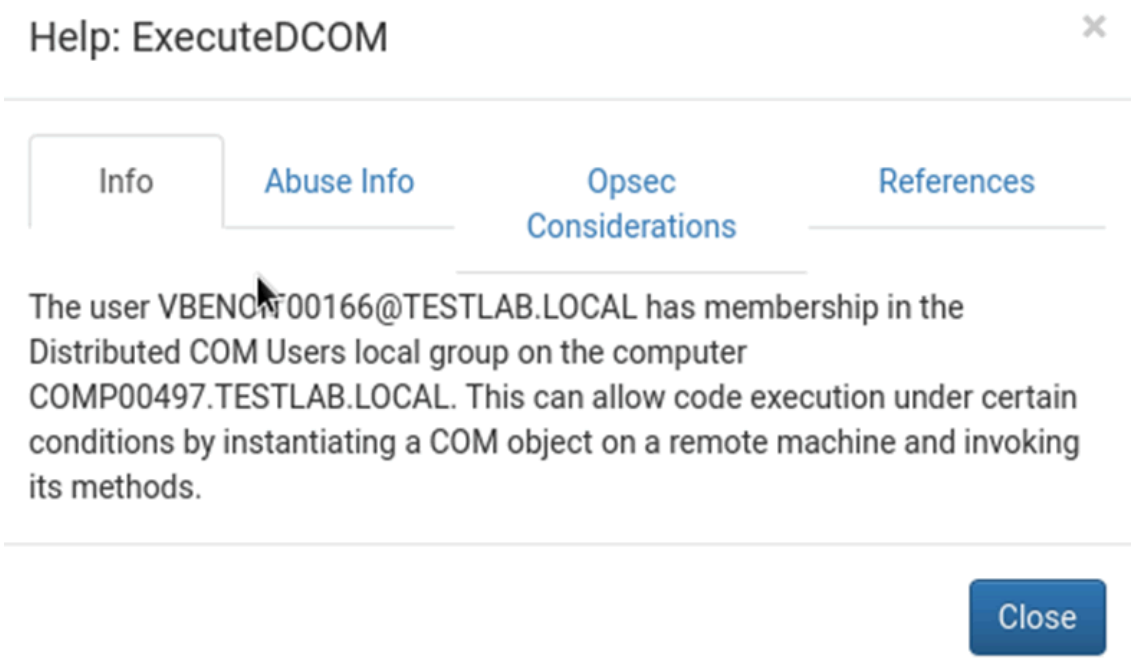
An example query of the shortest path to domain administrator is shown below:



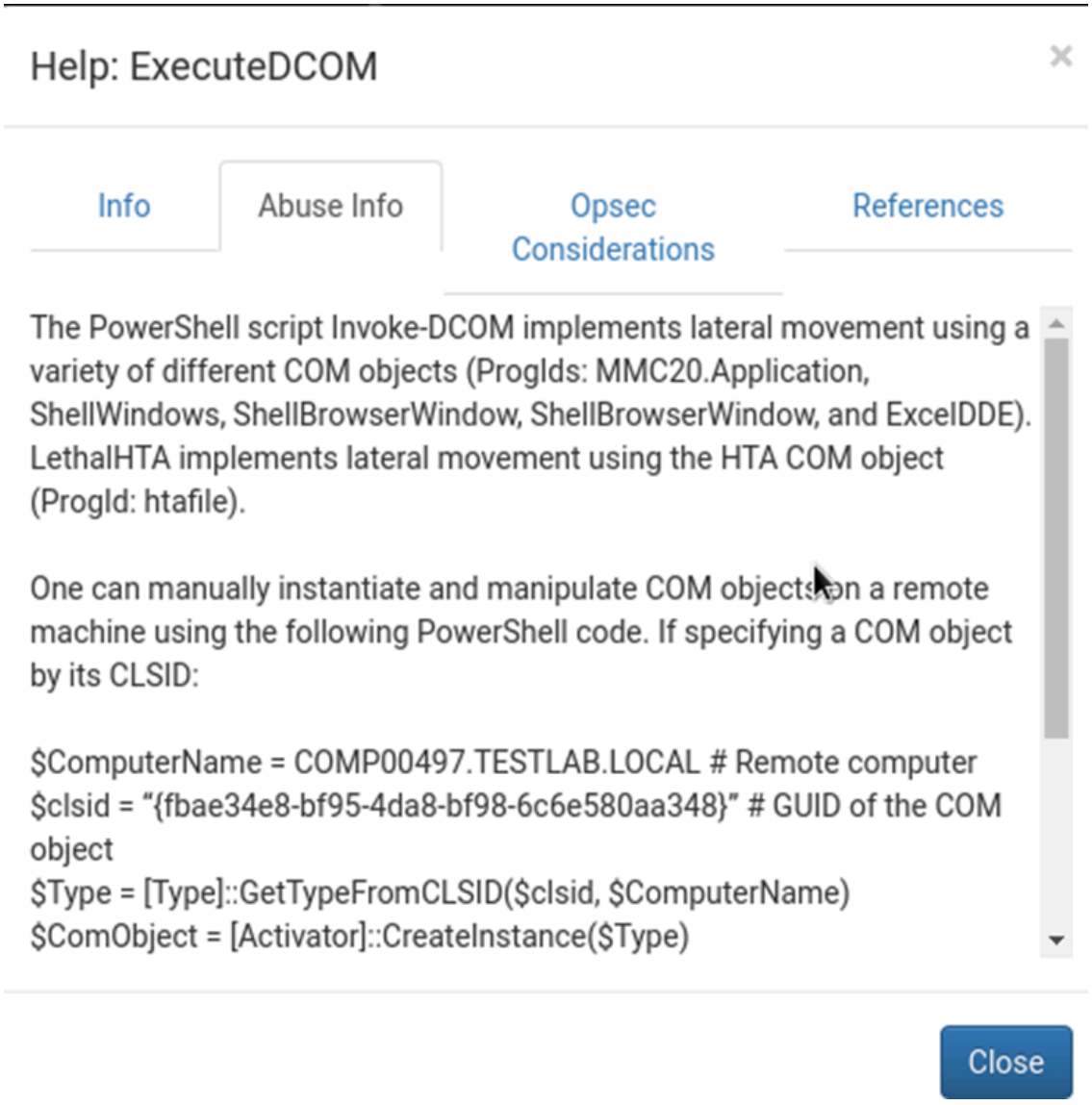
If you have never used BloodHound this will look like a lot going on and it is, but lets break this down. In the graph world where BloodHound operates, a Node is an active directory (AD) object. The different notes in BloodHound are represented using different icons and colours; Users (typically green with a person), Computers (red with a screen), Groups (yellow with a few people) and Domains (green-blue with a globe like icon). There are also others such as organizational units (OUs) and Group Policy Objects (GPOs) which extend the tool's capabilities and help outline different attack paths on a domain.

Essentially from left to right the graph is visualizing the shortest path on the domain to the domain admins group, this is demonstrated via multiple groups, machines and users which have separate permissions to do different things.

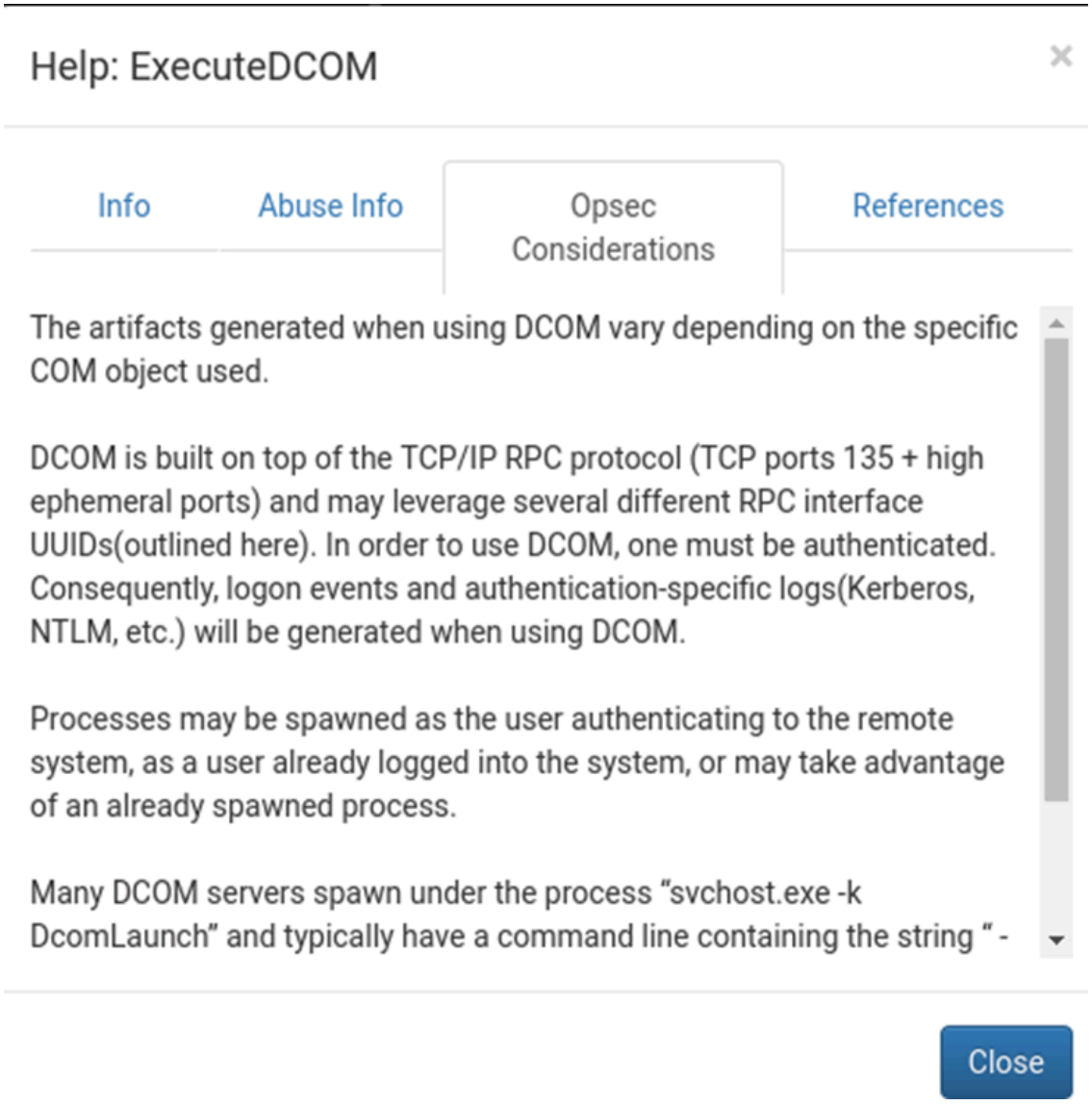
This feature set is where visualization and the power of BloodHound come into their own, from any given relationship (the lines between nodes), you can right click and view help about any given path:



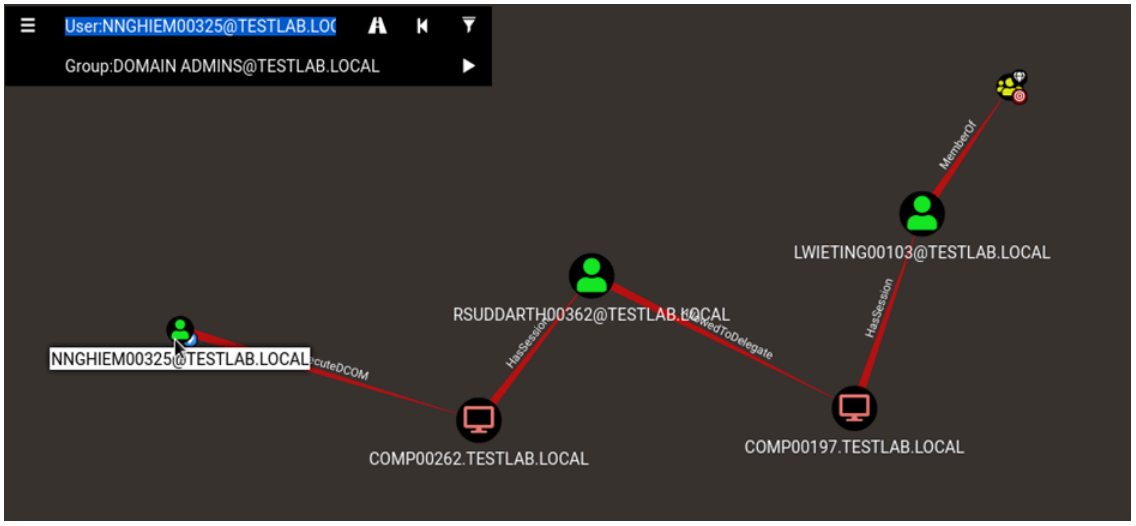
Within the help options of the attack path there is info about what the relationship is, how it can be abused and what operational security (opsec) considerations need to be taken into account:



In the abuse info, BloodHound will give the user the exact commands to drop into PowerShell in order to pivot through a node or exploit a relationship which is incredibly useful in such a complicated path. Additionally, the opsec considerations give more info surrounding what the abuse info does and how it might impact the artefacts dropped onto a machine.



By leveraging this information BloodHound can help red teams identify valid attack paths and blue teams identify indicators and paths of compromise. Back to the attack path, we can set the user as the start point by right clicking and setting as start point, then set domain admins as endpoint, this will make the graph smaller and easier to digest:



The user NNGHIEM00325@TESTLAB.LOCAL is going to be our path to domain administrator, by executing DCOM on COMP00262.TESTLAB.LOCAL, from the information; The user NNGHIEM00325@TESTLAB.LOCAL has membership in the Distributed COM Users local group on the computer COMP00262.TESTLAB.LOCAL. This can allow code execution under certain conditions by instantiating a COM object on a remote machine and invoking its methods. This gains us access to the machine where we can run various tools to hijack RSUDDARTH00362@TESTLAB.LOCAL's session and steal their hash, then leverage Rubeus:

```
Rubeus.exe s4u /user:victim /rc4:2b576acbe6bcfda7294d6bd18041b8fe /impersonateuser:admin /msdsspn:"H
```

Using the above command to impersonate the user and pivot through to COMP00197 where LWIETING00103 has a session who is a domain administrator.

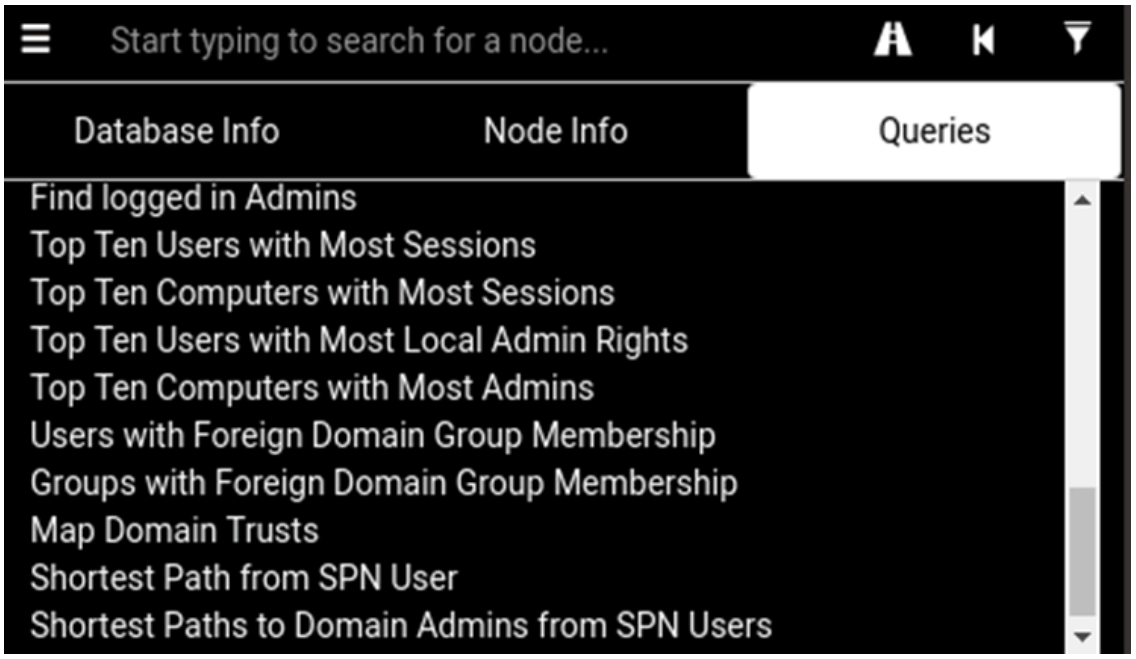
As simple as a small path, and an easy route to domain admin from a complex graph by leveraging the abuse info contained inside BloodHound.

Explaining Queries, How to Input Custom Ones

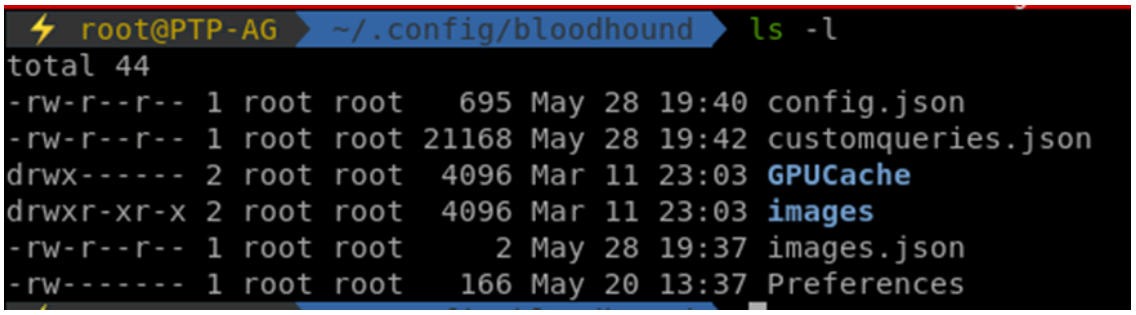
In addition to the default interface and queries there is also the option to add in custom queries which will help visualize more interesting paths and useful information. As of BloodHound 2.0 a few custom queries were removed however to add them back in, this code can be inputted to the interface via the queries tab:

Simply navigate to the queries tab and click on the pencil on the right, this will open customqueries,json where all of your custom queries live:

I have inputted the original BloodHound queries that show top tens and some other useful ones:



If you'd like to add more the custom queries usually lives in ~/.config/bloodhound/customqueries.json



There are endless projects and custom queries available, BloodHound-owned(https://github.com/porterhau5/BloodHound-Owned) can be used to identify waves and paths to domain admin effectively, it does this by connecting to the neo4j database locally and hooking up potential paths of attack. It also features custom queries that you can manually add into your BloodHound instance.

Extra Tips

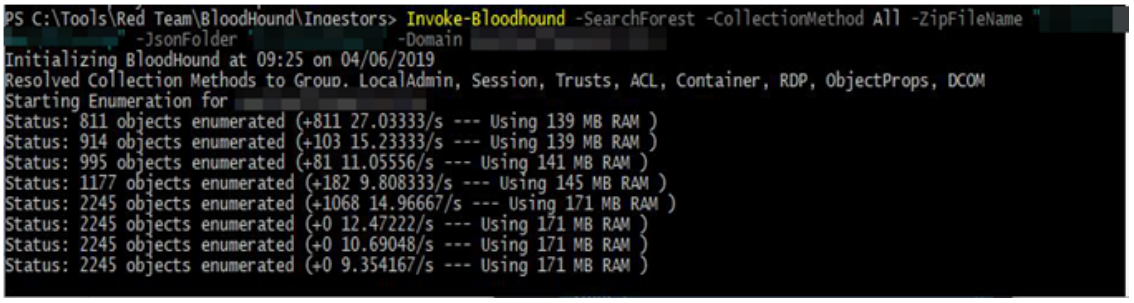
If you don't have access to a domain connected machine but you have creds, BloodHound can be run from your host system using runas. The following lines will enable you to query the Domain from outside the domain:

```
runas /netonly /user:FQDN.local\USER powershell
```

This will prompt for the user's password then should launch a new powershell window, from here you can import sharphound as you would normally:

```
Import-Module Sharphound.ps1

Invoke-BloodHound -ZipFileName 'PATH/TO/ZIP.zip' -JsonFolder 'PATH/TO/folderas above' -CollectionMet
```



This window will use the local DNS settings to find the nearest domain controller and perform the various LDAP lookups that BloodHound normally performs. By leveraging this you are not only less likely to trigger antivirus, you don't have to exfiltrate the results either which reduces the noise level on the network.

Using BloodHound Defensively

Hopefully the above has been a handy guide for those who are on the offensive security side of things however BloodHound can also be leveraged by blue teams to track paths of compromise, identify rogue administrator users and unknown privilege escalation bugs.

Just as visualising attack paths is incredibly useful for a red team to work out paths to high value targets, however it is just as useful for blue teams to visualise their active directory environment and view the same paths and how to prevent such attacks.

BloodHound can do this by showing previously unknown or hidden admin users who have access to sensitive assets such as domain controllers, mail servers or databases. These accounts may not belong to typical privileged Active Directory (AD) groups (i.e. Domain Admins/Enterprise Admins), but they still have access to the same systems. The permissions for these accounts are directly assigned using access control lists (ACL) on AD objects.

These accounts are often service, deployment or maintenance accounts that perform automated tasks in an environment or network. Which naturally presents an attractive target for attackers, who can leverage these service accounts for both lateral movement and gaining access to multiple systems. Exploitation of these privileges allows malware to easily spread throughout an organization. For this reason, it is essential for the blue team to identify them on routine analysis of the environment and thus why BloodHound is useful to fulfil this task.

Detecting BloodHound Usage

In addition to leveraging the same tooling as attackers, it is important for the blue team to be able to employ techniques to detect usage of such tooling for better time to detection and reaction for incident response. To identify usage of BloodHound in your environment it is recommended that endpoints be monitored for access and requests to TCP port 389(LDAP) and TCP port 636(LDAPS) and similar traffic between your endpoints and your domain controllers. A large set of queries to active directory would be very suspicious too and point to usage of BloodHound or similar on your domain.

References

- Bloodhound was created and is developed by @_wald0, @CptJesus, and @harmj0y.
- GitHub page: https://github.com/BloodhoundAD/Bloodhound
- Additional Tooling: https://github.com/BloodhoundAD/Bloodhound-Tools
- BloodHoundOwned Project: https://github.com/porterhau5/BloodHound-Owned
- Sharphound Ingestor: https://github.com/BloodhoundAD/SharpHound

---