The Ultimate Guide for BloodHound Community Edition (BHCE)

m4lwhere.medium.com/the-ultimate-guide-for-bloodhound-community-edition-bhce-80b574595acf

Chris Haller February 9, 2025

Chris Haller

I've run into many interested hackers who want to learn how to use BloodHound, but struggle to get started. This results in many getting lost and getting frustrated. Beyond this, if we're in a pentest engagement, we do a complete disservice to the customer by not enumerating as many risks as possible.

A happy hacking dog that's proud he's figured out how to do things effectively :)

This article will cover setting up, collecting data, analyzing the data, and providing value with that data. This will use the GOADv2 forest as sample data for collection and analysis. I've already collected some sample data and placed it on a repository to ingest for analysis and practice.

Huge thanks to the team for creating BloodHound and giving it to the community!

The Why

Before we start, we need to understand the . Active Directory is notoriously complex, and with anything that involves complexity, there is plenty of room for errors. AD is interesting in the fact that most environments don't have traditional CVE-based exploits available — instead most escalation attacks rely on misconfigurations. Our job is to enumerate as many of these escalation scenarios as possible, define their risk, and present recommendations to reduce risk to lowest possible levels.

If you want to get **paid** to do this work, you must be able to effectively present these risks in writing!

The Background

BloodHound has been developed by SpecterOps as a way to visualize relationships between objects in AD. Because of the scale and complexity of most AD networks, manually auditing these relationships is a nightmare. Instead, the original BloodHound relied on Neo4j's Graph Theory to visualize this information to escalate between objects.

There are currently three versions of BloodHound you need to know about:

: The original BloodHound, no longer supported. Built on an Electron-based application, somewhat complicated to set up.

: Released in August 2023, actively supported. Leverages docker compose to manage a set of containers, exceptionally easy to deploy. Smooth web application interface.

: Paid version of BloodHound for attack path management. The major difference is that this version is used for risk management and validation.

The Basics

There are a few different parts we need to be aware about. First, the BloodHound application itself is nothing more than a front-end to help visualize, present, and analyze data. We need to gather data about the environment with a collector to have it ingested into the application for analysis.

The Collector

We need to gather the data from the AD environment in order to feed it into BloodHound for analysis. There are two major collectors you need to know about:

: This is the officially supported collector tool for BloodHound, written in C#. Must be run from a Windows-based PC connected to the domain in order to collect the information.

: A community driven python script used to collect AD data. Can be run from a Linux-based machine, such as a Raspberry Pi. Excellent use-case for a pentesting dropbox.

It's important to realize that at the time of this writing, `bloodhound.py` does **not natively support BloodHound-CE**. You must use the `bloodhound-ce` branch of the `bloodhound.py`python collector if you choose to use this. We cannot mix legacy collectors with Community Edition collectors — this will cause the ingest to fail (and it's frustrating!).

The Frontend

BloodHound itself is the web application used to interpret the data from the collector. This is the GUI-based application we interact with to interpret the data for risks and escalation paths. The frontend is only as good as the data it's ingested from the collector.

The Ingest

Within the GUI frontend is the File Ingest. This is what is used to place the data gathered from the collector to be interpreted into the Neo4j database. Once parsed, this data will be accessible to the GUI application for analysis.

The API

One of the most exciting parts about BloodHound-CE is the HTTP API available to query data. This can help us automate and extract data quickly to prove value in a pentesting engagement.

Legacy BloodHound

We've touched on the original version of BloodHound legacy and why it's important earlier, but this is critical to understand. The older collectors DO NOT WORK with the Community Edition of BloodHound. Anyone can still use the legacy version and it still works great — however, it will not be up to date on the latest threats. Since we're proving value to customers, we need to use items which can evaluate the most applicable risk to them.

Getting Started

Great, now that we know all about each part of the puzzle, we can get started by installing BloodHound-CE and collecting data for our analysis.

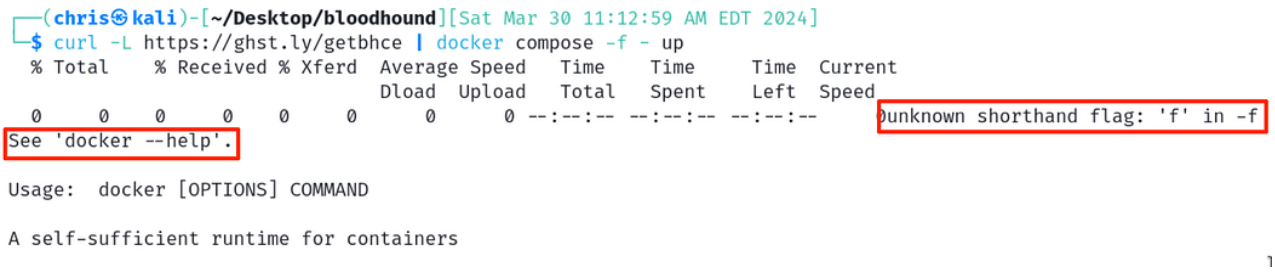
Start the Containers

Community Edition leverages docker compose through a set of containers. This significantly enhances the simplicity of starting up and managing infrastructure for BloodHound, as everything is containerized within a docker network.

To start, all we need to do is download the `docker-compose.yml` file and instruct docker to build the containers. There's a simple one-liner provided by SpecterOps to begin:

```
curl -L https://ghst.ly/getbhce | docker compose -f - up
```

Note: If you're using Kali, there's a chance the installed `docker` version does not have the `compose` command. If this is the case, you will receive an error similar to below:



Error with docker compose command, don't worry — we can fix it!

NOTE: As of early December 2024, Debian has removed `docker-compose` from packages. What we need to do now is to install from `pip` (which I personally have not had luck) or download the precompiled binary from GitHub.

These binaries are available: https://github.com/docker/compose/releases. Download the applicable one for your architecture and then place it into your `$PATH`.

```
# Download ARM based docker-compose from releases
wget https://github.com/docker/compose/releases/download/v2.32.1/docker-compose-linux-aarch64

 +x ./docker-compose-linux-aarch64 ./docker-compose-linux-aarch64 /usr/bin/
```

Now you should be able to properly access the `docker-compose` command from your terminal :)

OUTDATED INFO: This is left in for posterity. To fix this, we can install the `docker-compose` package within Kali's repositories. Then, we can modify the command to run the containers.

```
sudo apt install docker-compose -ydocker-compose -vcurl -L //ghst.ly/getbhce
docker-compose -f - up
```

Back to valid and usable info!

```
┌──(chris㉿kali)-[~/Desktop/bloodhound][Sat Mar 30 11:19:27 AM EDT 2024]
└─$ curl -L https://ghst.ly/getbhce | docker-compose -f - up
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   190  100   190    0     0    506      0 --:--:-- --:--:-- --:--:--   506
100  3618  100  3618    0     0   5345      0 --:--:-- --:--:-- --:--:--  5345
Creating network "bloodhound_default" with the default driver
Creating volume "bloodhound_neo4j-data" with default driver
Creating volume "bloodhound_postgres-data" with default driver
Creating bloodhound_app-db_1    ... done
Creating bloodhound_graph-db_1  ... done
Creating bloodhound_bloodhound_1 ... done
Attaching to bloodhound_app-db_1, bloodhound_graph-db_1, bloodhound_bloodhound_1
```

Much happier!

To hold onto the `docker-compose.yml` file to use again later, we can use this command to download it to our current directory. When the file is saved with this name, we don't need to specify it when starting `docker-compose`.

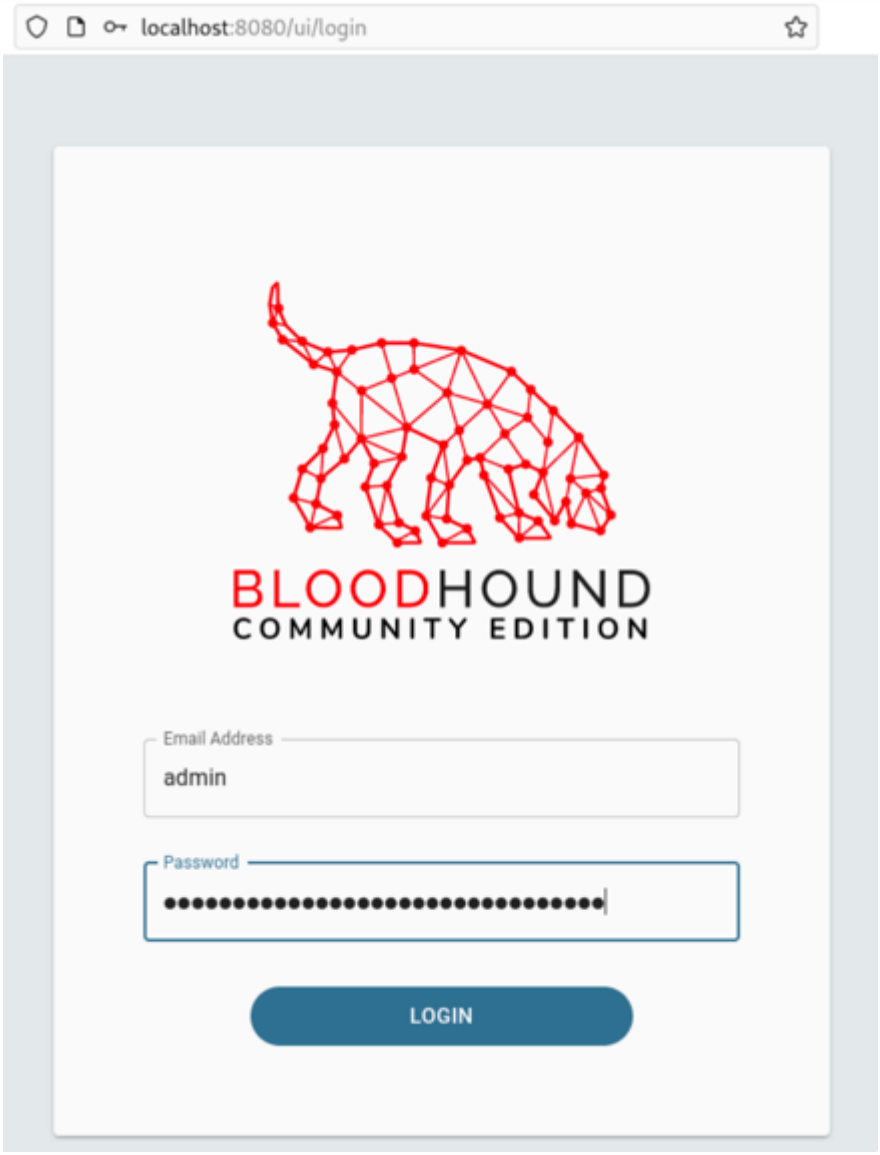
```
wget https://ghst.ly/getbhce -O docker-compose.ymldocker-compose up
```

Great! Looks like the containers have been started with the `docker-compose` command, green is always a good color. Pay attention though, in the logs there will be an initial password we need to use to log into the system.

```
bloodhound_1  | {"level":"info","time":"2024-03-30T15:51:06.334488602Z","message":"#############################################################################"}
bloodhound_1  | {"level":"info","time":"2024-03-30T15:51:06.334497061Z","message":"#                                                                           #"}
bloodhound_1  | {"level":"info","time":"2024-03-30T15:51:06.334498019Z","message":"# Initial Password Set To:    8bPjH9jhch67EhUVrMcIMS3dJZqe0BLL    #"}
bloodhound_1  | {"level":"info","time":"2024-03-30T15:51:06.334498686Z","message":"#                                                                           #"}
bloodhound_1  | {"level":"info","time":"2024-03-30T15:51:06.334499269Z","message":"#############################################################################"}
```

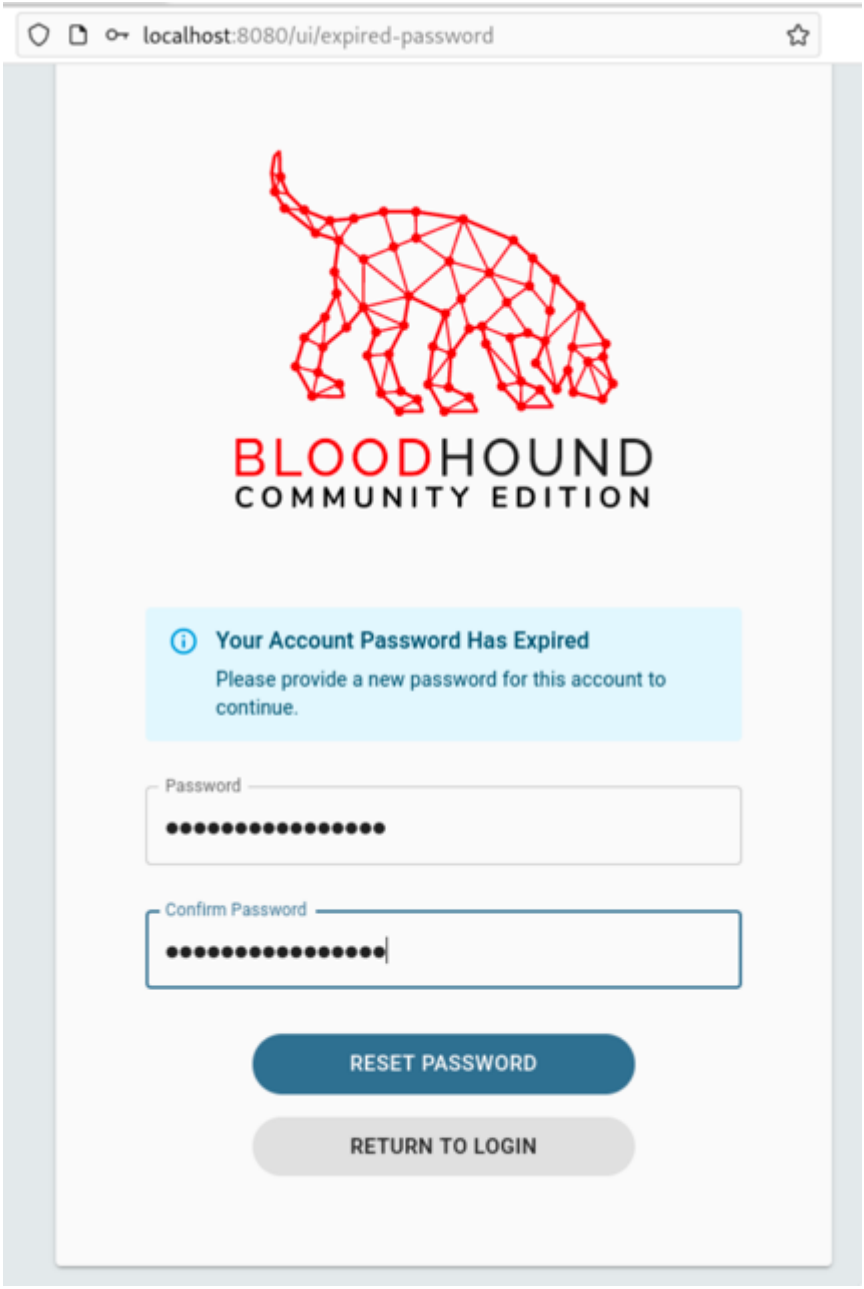
Identifying the initial password to log into the GUI

Copy this password and browse to http://localhost:8080 to log into the GUI. The user will be `admin` and the initial password from the logs.



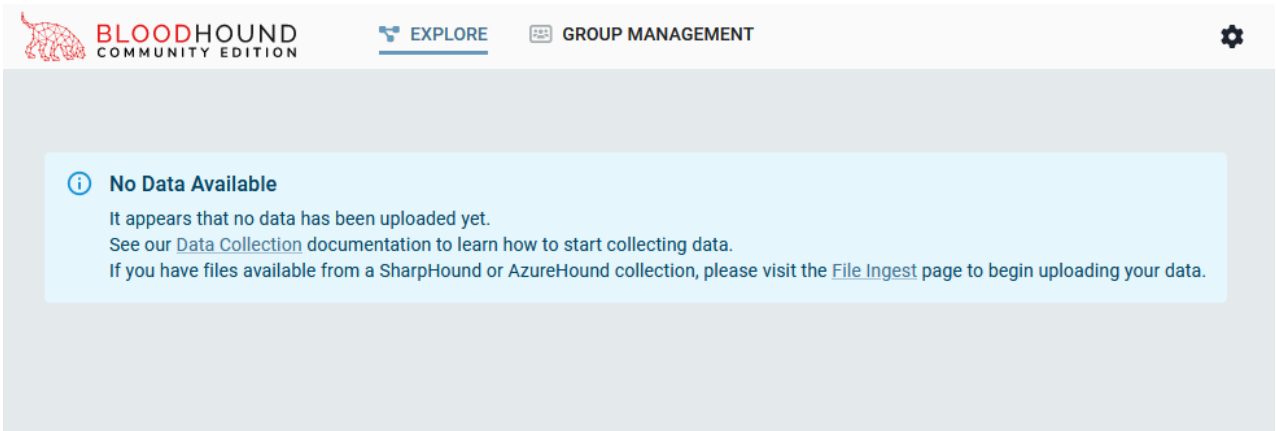
Initial login with the generated password.

When you log in, you will be directed to update the password. Choose one you will remember.



Updating the password for the admin account

Now we're in! But wait, there's no data! So how do we actually get started with the analysis? Well, we need to download the collectors first!

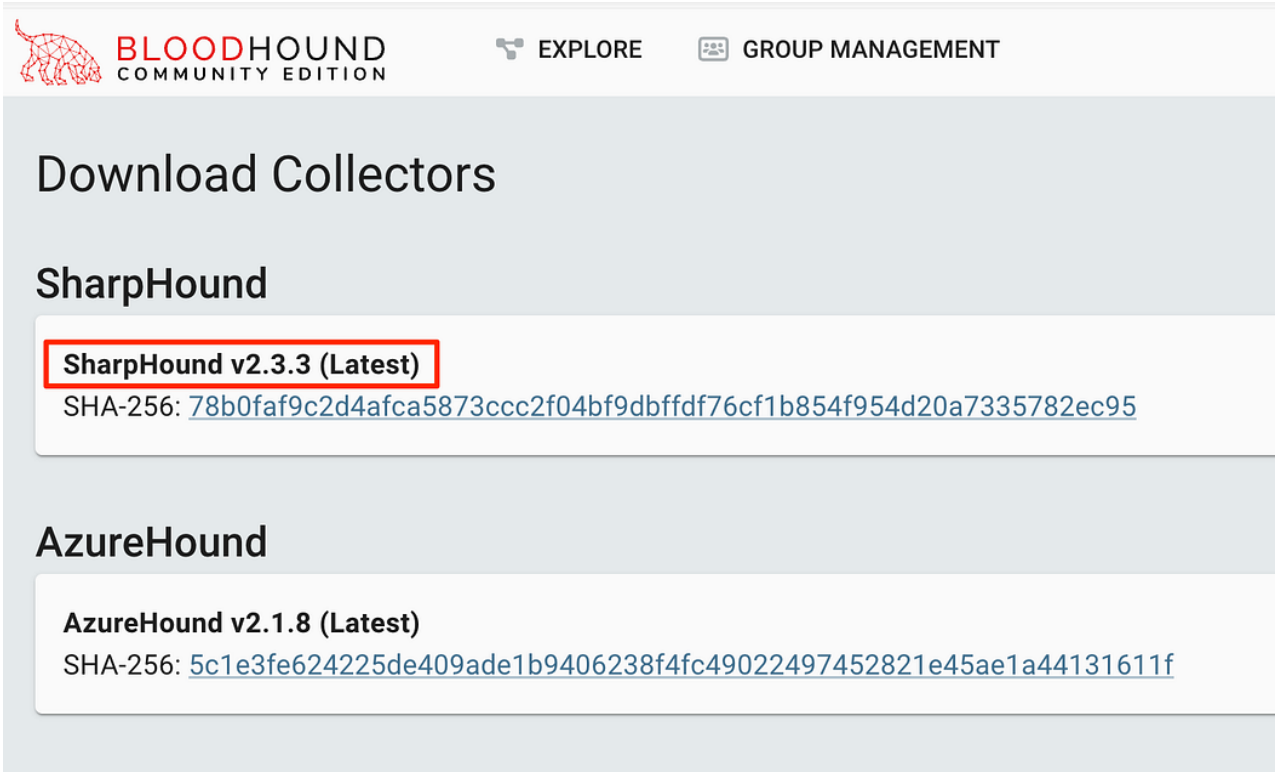


We're in! But there's nothing to analyze :(

Data Collection

We have a few ways for us to do this. We can use the SharpHound.exe C#, PowerShell, or Python collectors to gather this information.

To get a copy of the supported collectors, we can download them straight from the BHCE GUI. Click on the cog and then “Download Collectors”. This will open a page where we can download the collector.



The collector available within the GUI is easily available.

We can always gather the latest versions of the collector which are available on the SharpHound Releases page:

Once unzipped, we can launch this collection tool on the remote host. Choose your favorite way to complete this, whether it’s within a beacon for inline-execute or an interactive RDP session.

C# Collector

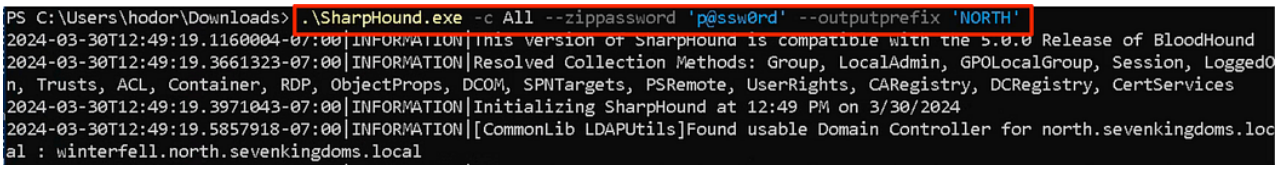
Using SharpHound.exe is straightforward — we can simply execute it without any additional flags and it will happily gather the default information about the current domain with the current user.

```
.\SharpHound.exe
```

To collect all available information, we can specify the flag `-c All`. This will provide information such as ADCS information, RDP, and DCOM. However, in large environments collecting everything could easily overwhelm our machine — keep this in mind!

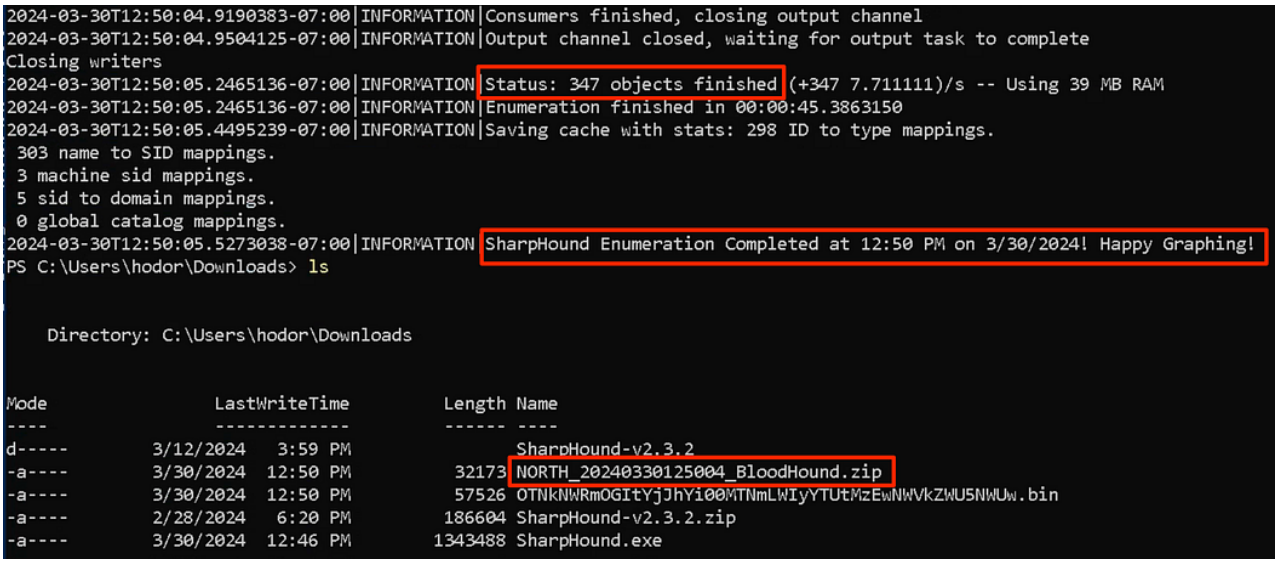
Personally, I like to collect everything and then encrypt the ZIP with a password, while also giving the files a prefix. While this data is available to all users in the domain, in the wrong hands it could prove to be sensitive information.

```
.\SharpHound.exe -c All --zippassword  --outputprefix
```



Running the collector to gather everything, encrypt the zip, and add a prefix.

Once completed, we'll see that the zip file was created with our intended prefix as well. Note the number of objects, this can be a useful number within a report to a customer.



Collection completed for the north.sevenkingdoms.local domain.

Cross-Domain Collection

To collect data from other domains in the same forest, we'll need to add a few additional flags. For instance, we'll need to direct to the correct domain with the `--domain` flag. We'll pivot to the `sevenkingdoms.local` domain next, while running as the `hodor@north.sevenkingdoms.local` user. This machine must be able to resolve the domain within DNS to work.

```
.\SharpHound.exe -c All --domain sevenkingdoms.local --zippassword  --outputprefix
```

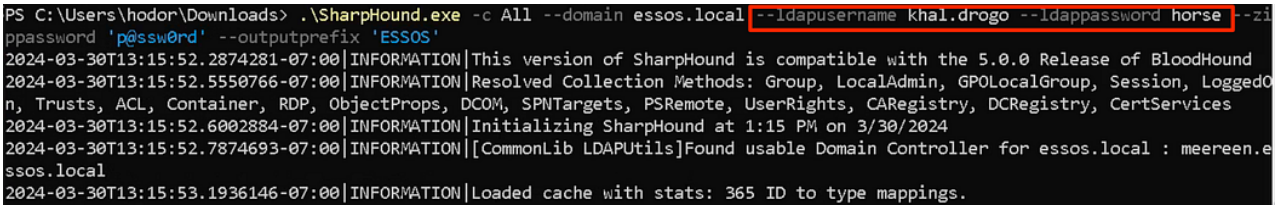
Collection as a Different User

In the event we're in a forest but don't have access to an account trusted on a separate domain, we can always launch `SharpHound.exe` with the `runas.exe` command.

```
runas /netonly /user:khal.drogo@essos.local cmd.\SharpHound.exe -c All --domain
essos.local --zippassword  --outputprefix
```

Alternatively, we can also specify the `--ldapusername` and `--ldappassword` flags to connect to the other domain. This does not require the `runas.exe` command to function.

```
.\SharpHound.exe -c All --domain essos.local --ldapusername khal.drogo --
ldappassword horse --zippassword  --outputprefix
```

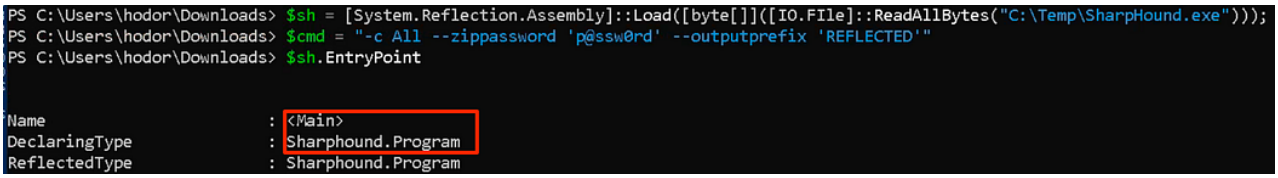


Gathering cross-domain information via LDAP authentication. This gathered from a session as hodor@north.sevenkingdoms.local to use khal.drogo@essos.local for the essos.local domain.

Reflectively Loading SharpHound

Have you run into a scenario where the SharpHound.exe is flagged by detections as malicious? In some scenarios, we can bypass these controls by reflectively loading the C# executable into memory, then executing the entrypoint (ATT&CK ID T1620).

```
 = [System.Reflection.Assembly]::Load([byte[]]([IO.FIle]::ReadAllBytes())); =
[Sharphound.Program]::Main(.Split())
```



Identifying the type and entry to execute the collector within memory.

Note that you may need to add the `--outputdirectory` switch to ensure it is saved in your desired location.

Python Collector

Next we can use the `bloodhound.py` tool to gather this information as well. As noted earlier, the current `bloodhound.py` package in Kali repositories is for Legacy BloodHound only. You will need to download the `bloodhound-ce` branch from their GitHub.

```
sudo apt install bloodhound.pybloodhound-python -d north.sevenkingdoms.local -u
hodor -p hodor -c All -op default_kali_bloodhoundpy --zip -ns 192.168.56.10
```

Since we're interested in data to support BHCE, let's focus on installing that branch and using that one instead. We can clone that branch in particular directly from GitHub.

```
git  -b bloodhound-ce https://github.com/dirkjanm/BloodHound.py.git
```

If we don't want to clobber any dependencies, we can always build a container to run this within. A Dockerfile is already present within the repo which we can use to build this.

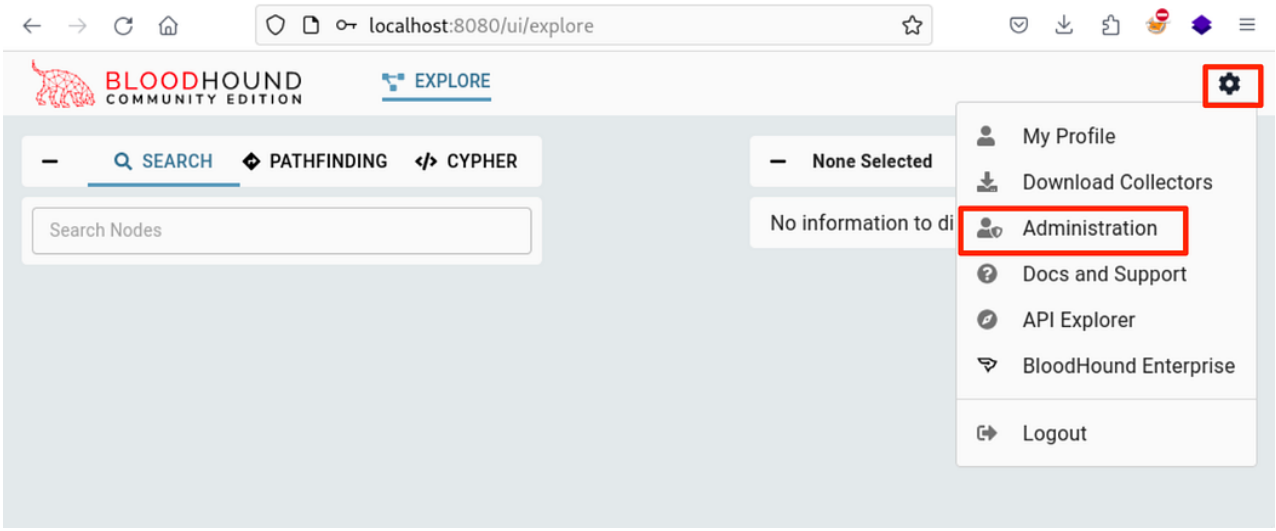
```
# Build the docker container for bloodhound.py
cd BloodHound.py
docker build -t bloodhound .

docker run -v :/bloodhound-data -it bloodhounda0140a0d356a:/bloodhound-data
```

```
┌──(kali㉿GOAD-Kali)-[~]-[SCI-GOAD-RANGE]
└─$ bloodhound-python -d north.sevenkingdoms.local -u hodor -p hodor -c All -op default_kali_bloodhoundpy --zip -ns 192.168.56.10
INFO: Found AD domain: north.sevenkingdoms.local
WARNING: Could not find a global catalog server, assuming the primary DC has this role
If this gives errors, either specify a hostname with -gc or disable gc resolution with --disable-autogc
INFO: Getting TGT for user
INFO: Connecting to LDAP server: winterfell.north.sevenkingdoms.local
INFO: Found 1 domains
INFO: Found 2 domains in the forest
INFO: Found 2 computers
INFO: Connecting to GC LDAP server: winterfell.north.sevenkingdoms.local
INFO: Connecting to LDAP server: winterfell.north.sevenkingdoms.local
INFO: Found 17 users
INFO: Found 51 groups
INFO: Found 3 gpos
INFO: Found 1 ous
INFO: Found 19 containers
INFO: Found 1 trusts
INFO: Starting computer enumeration with 10 workers
INFO: Querying computer: castelblack.north.sevenkingdoms.local
INFO: Querying computer: winterfell.north.sevenkingdoms.local
INFO: Done in 00M 02S
INFO: Compressing output into 20240331205852_bloodhound.zip
```

Default bloodhound-python collection from Kali repositories

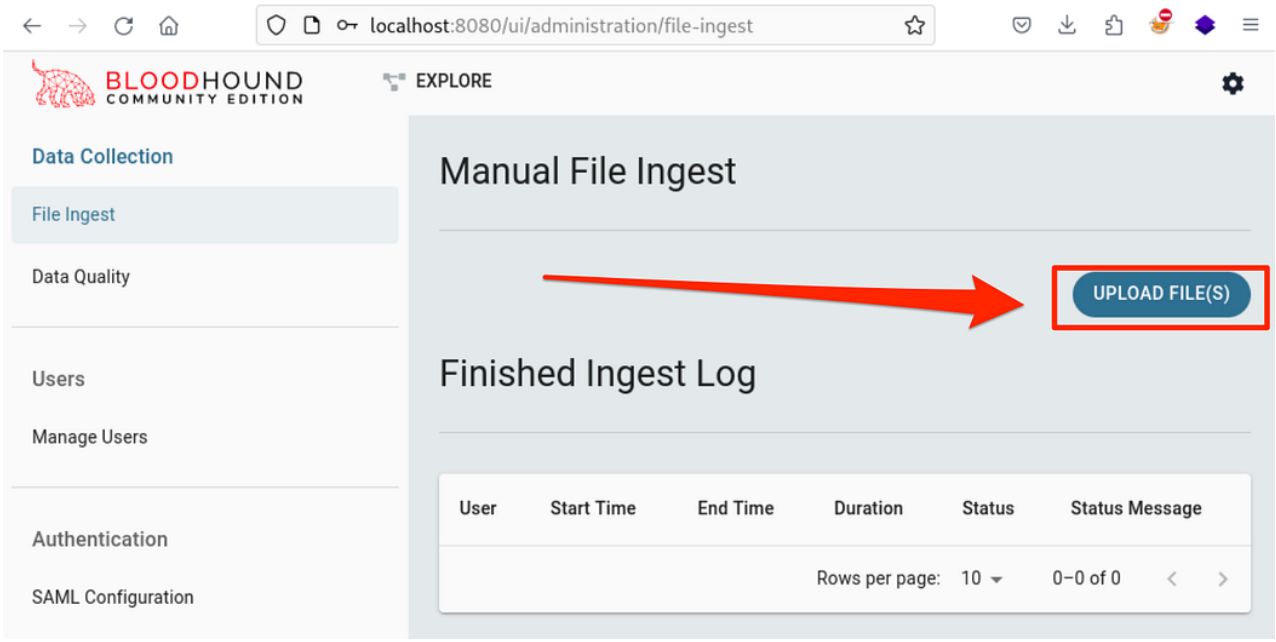
Data Ingest

Now that we've been able to collect data, we need to be able to use it. We need to upload it into the GUI, where it will be ingested into the Neo4j database. This is done by clicking on the cog button and then the Administration button.



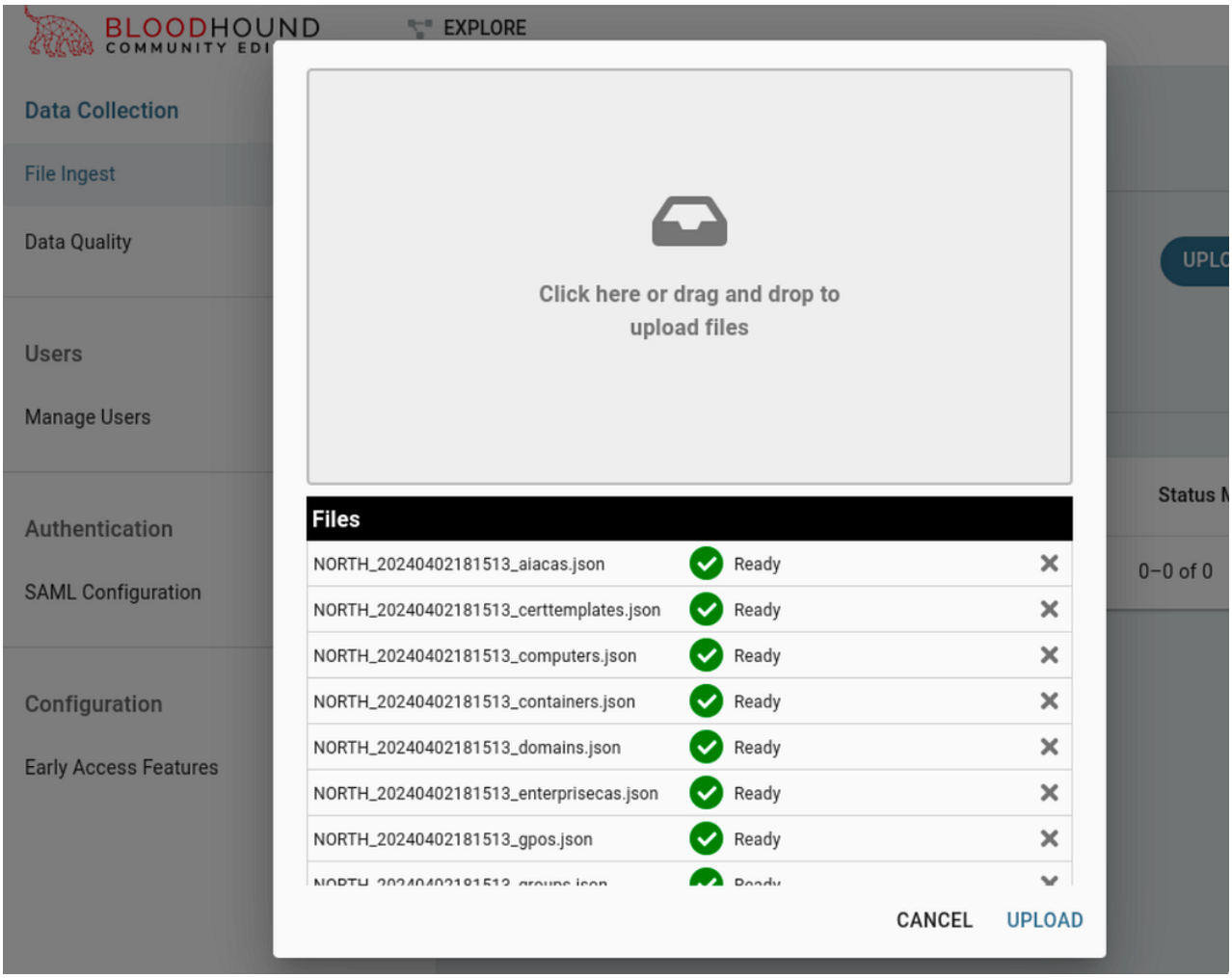
Navigating to ingest data within the GUI.

At this point, we can click the *UPLOAD FILE(S)* button to upload our data. We can drag and drop the ZIP file itself into the interface, no need to unzip for individual files anymore!



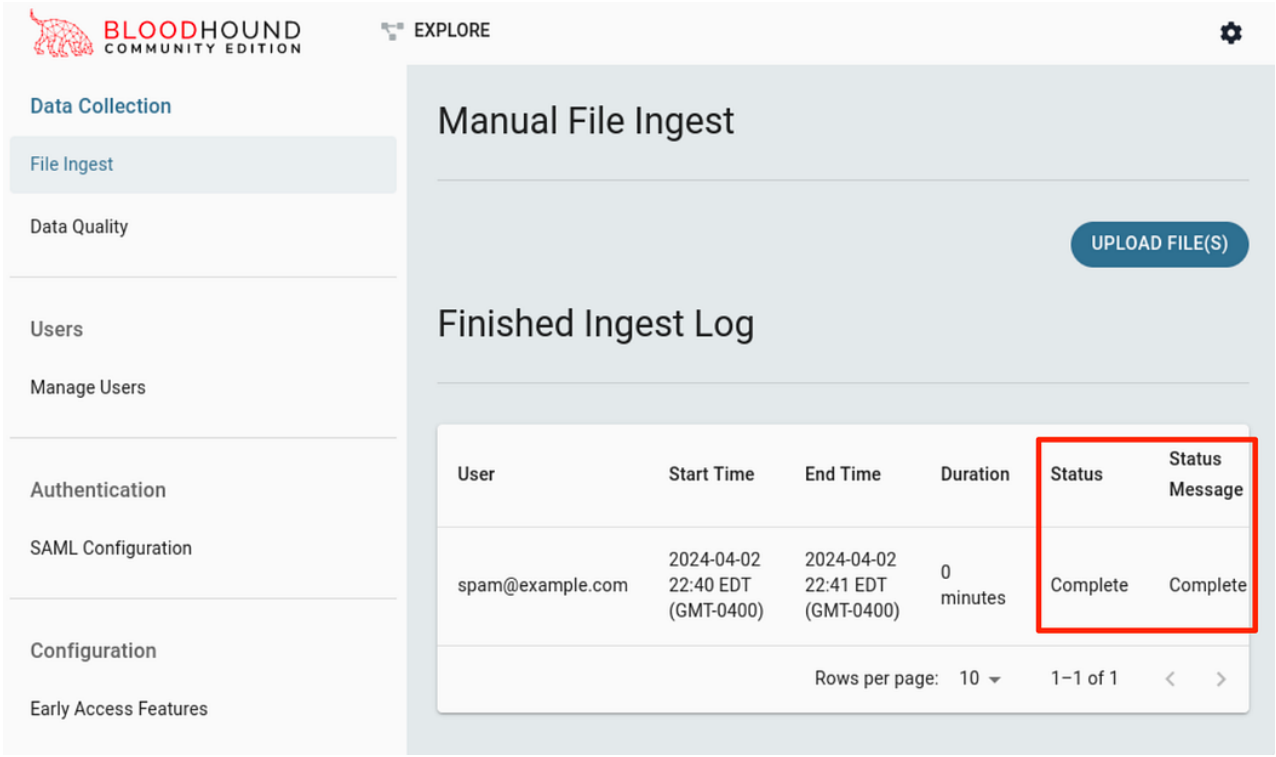
Clicking the UPLOAD FILE(S) button to add our data.

The popup window that appears will allow us to drag and drop files to be ingested.



Uploading the JSON files extracted from the zip created by the collector

After clicking the upload button twice, we'll be brought back to the ingest page. We can see that the status states it's completed ingesting! We can continue to upload additional data for other domains in the forest as well.

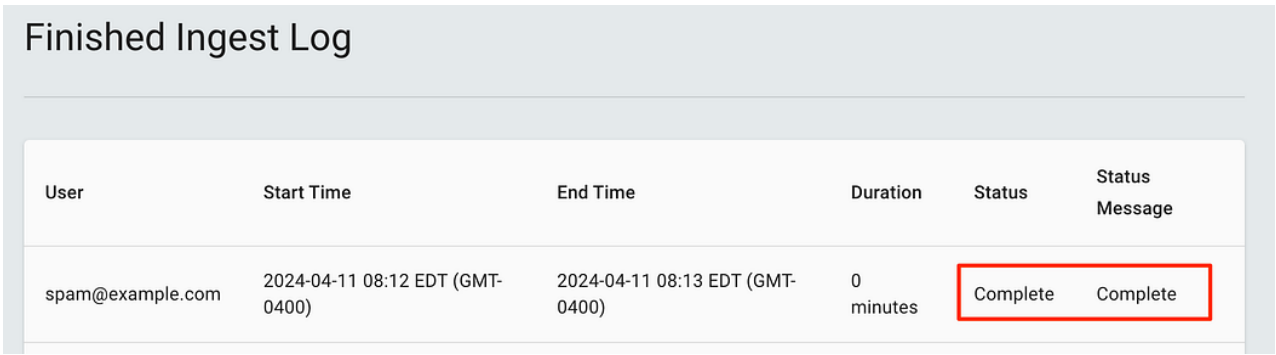


After uploading the files and they have completed ingest.

Ingest Failures

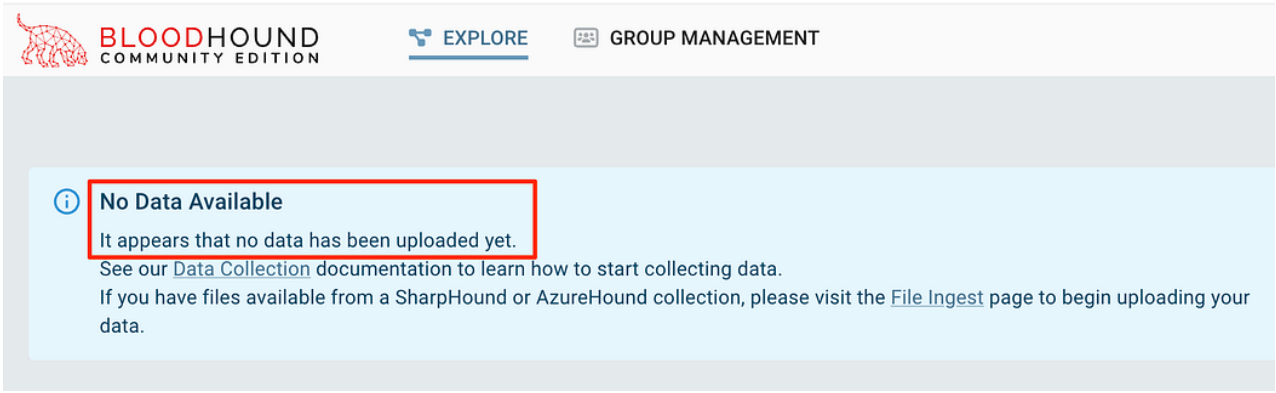
One of the more frustrating pieces about BHCE is that there doesn't appear to be any feedback when outdated information is loaded in for ingest. This scenario in particular is why I believe so many hackers learning how to use this tool get frustrated and quit.

For this scenario, if we gather information from a collector for Legacy Bloodhound and import it into BHCE, there's a chance that the files will pass the initial upload check, be marked as complete, but won't actually be ingested.



Ingested data marked as Complete — but wait where is it?

Browsing back to the Explore page we see that there's still no data ingested. This is where it's easy to get frustrated — after all we uploaded the data right?



Data was not ingested properly, even though it was marked as Complete.

To find this we needed to jump into the logs with `docker compose logs` to see the error. This shows that there was an unmarshalling error from the data uploaded. This was with data collected via the `bloodhound-python` from the Kali repos.

```
bloodhound-1  | {"level":"info","user_id":"752989a6-846d-444f-9075-bbb620351824","remote_addr":"192.168.65.1:4371
3","proto":"HTTP/1.1","referer":"http://192.168.40.180:8080/ui/administration/file-ingest","user_agent":"Mozilla/
5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36 Edg
/123.0.0.0","request_id":"51731b8c-d40f-4172-a47b-ad1d662bbe4c","request_bytes":0,"response_bytes":330,"status":2
00,"elapsed":4.768083,"time":"2024-04-11T12:13:04.805180636Z","message":"GET /api/v2/file-upload?skip=0&limit=10&
sort_by=-id"}
bloodhound-1  | {"level":"error","time":"2024-04-11T12:13:07.84171197Z","message":"Error decoding ein.Domain obje
ct: json: cannot unmarshal number into Go struct field Trust.Trusts.TrustDirection of type string"}
bloodhound-1  | {"level":"info","time":"2024-04-11T12:13:08.438585012Z","message":"Expanding all AD group and loc
al group memberships"}
bloodhound-1  | {"level":"info","time":"2024-04-11T12:13:08.439762804Z","message":"Collected 113 groups to resolv
e"}
bloodhound-1  | {"level":"info","time":"2024-04-11T12:13:08.508270137Z","message":"Finished post-processing 3 act
ive directory computers"}
bloodhound-1  | {"level":"info","time":"2024-04-11T12:13:08.513807346Z","message":"Finished building adcs cache"}
bloodhound-1  | {"level":"info","time":"2024-04-11T12:13:08.530453012Z","message":"Started Data Quality Stats Col
lection"}
bloodhound-1  | {"level":"info","time":"2024-04-11T12:13:08.533645554Z","message":"Cache successfully reset by da
tapipe daemon"}
```

Unmarshalling errors from uploading data using the bloodhound-python from Kali repos.

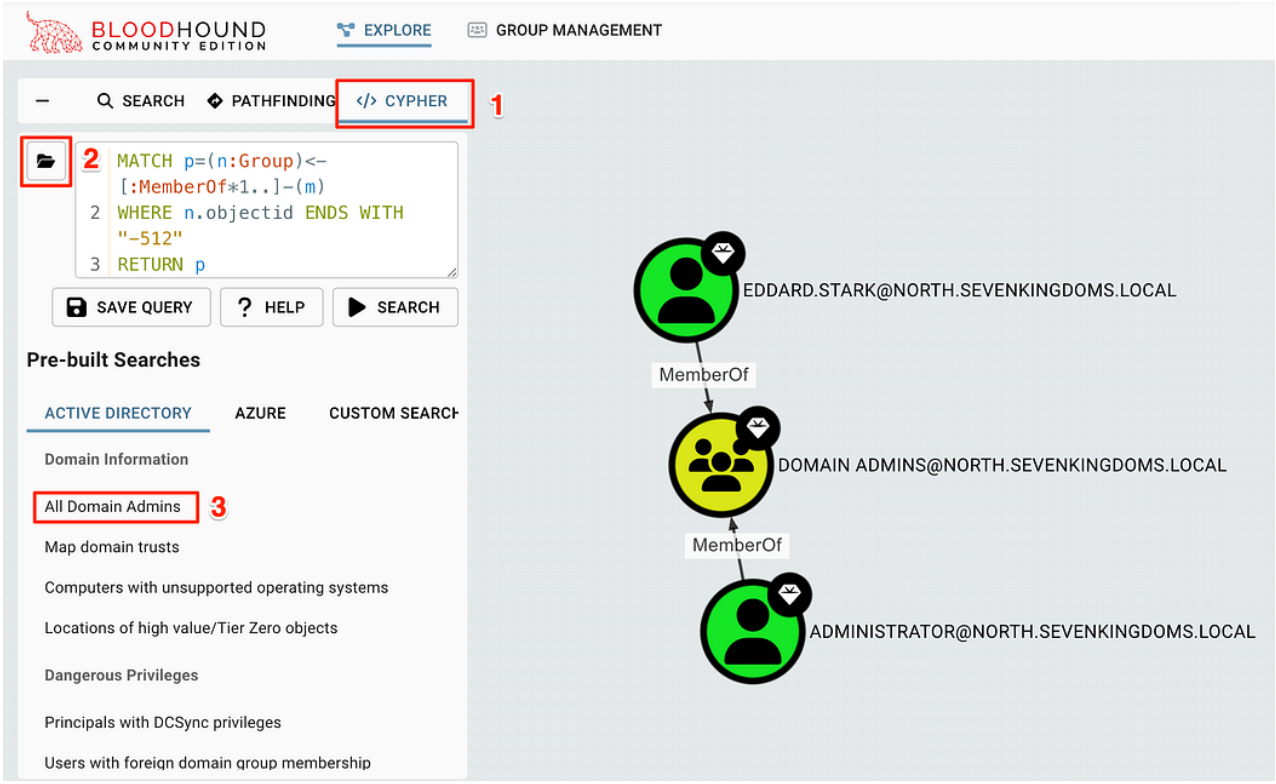
To help fix this, I would recommend using the latest collectors for the BHCE. I would also certainly wish that the BHCE GUI is updated to reflect these unmarshalling errors or at least provide an indication that the ingest did not work successfully.

Ingest via API

To ingest data through the API, we can read the docs. Personally, I do not do this since I upload data through the GUI, however, it is supported.

Browsing the Data

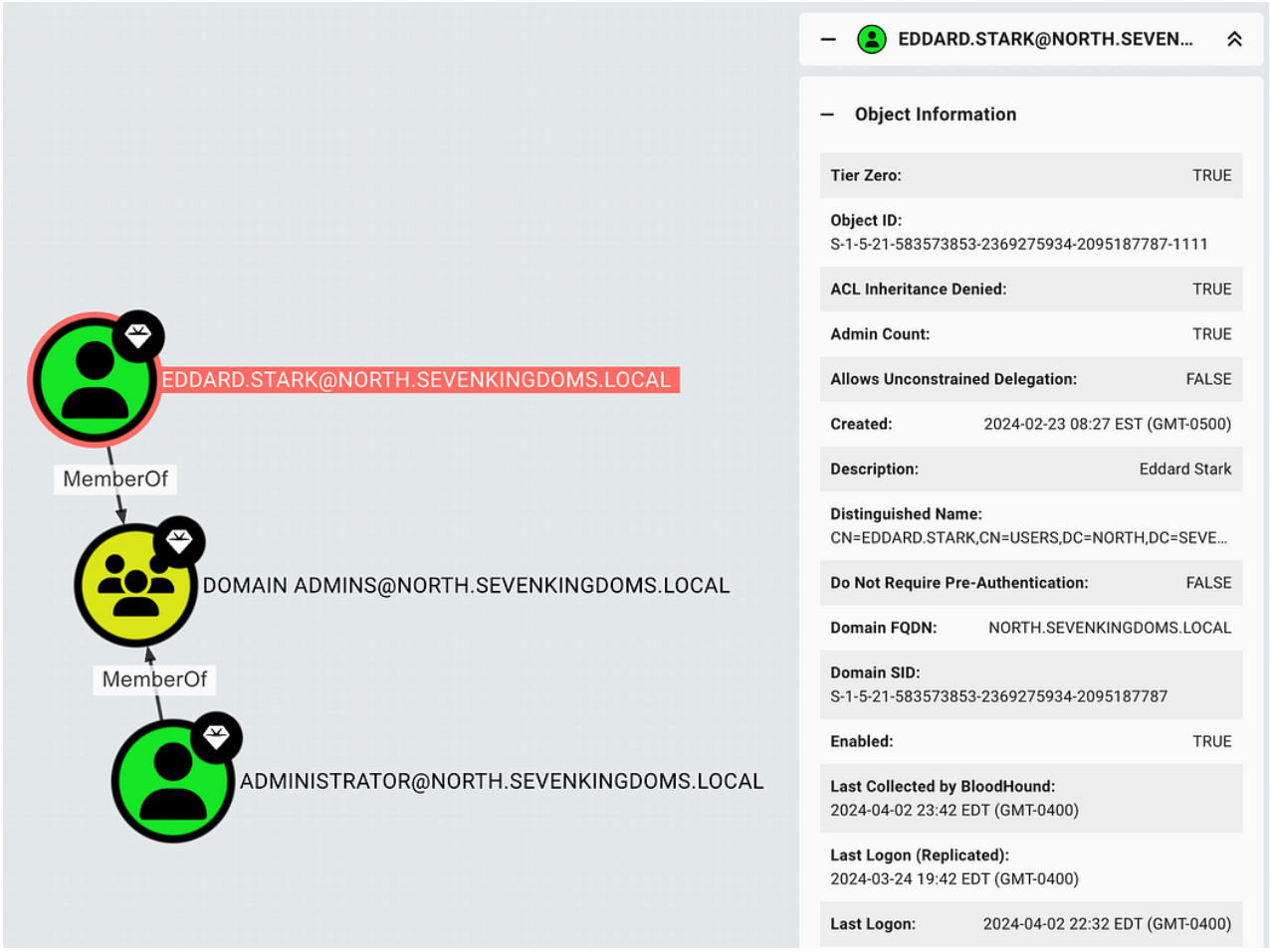
Since you've made it this far, we can actually start to explore the data that was gathered! This allows us to find and understand the relationships between the objects in the forest and how they could be exploited. To get started, we can use the built-in queries to explore the data. This is done by clicking the *CYPHER* button, then the folder icon to open the searches.



Navigating to the Pre-built searches to start exploring data

Clicking on one of the searches, such as “All Domain Admins” will add a Cypher query to the search bar and look for matches in the database. The results will now be on the screen!

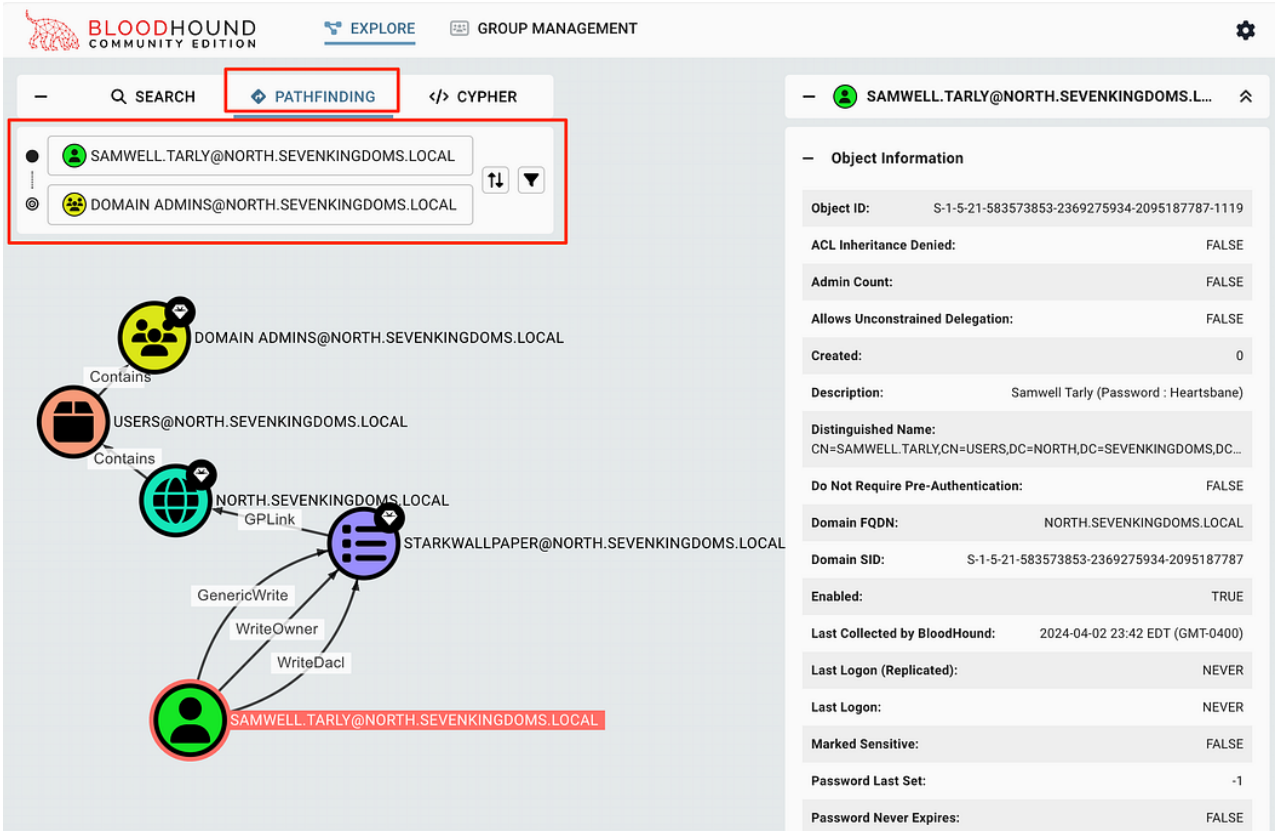
To get detailed information about an object, we can click on it to see its properties. This can help reveal additional info we could use about the account or within the domain.



Reviewing the properties for the EDDARD.STARK account by clicking on it.

Pathfinding

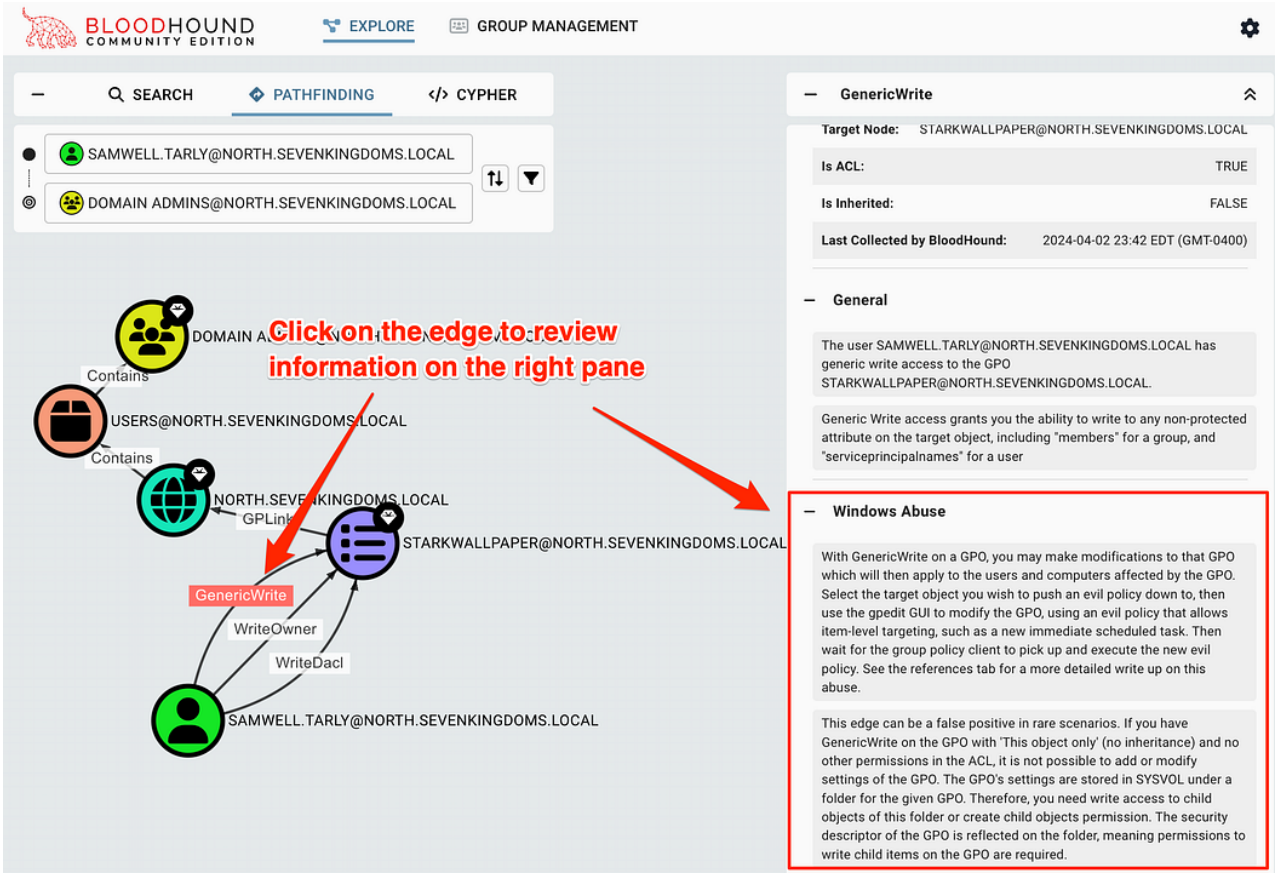
To find specific paths from one object to another, we can use the PATHFINDING button. In this case, we can request how the user `samwell.tarly` can gain access to `Domain Admins` to identify how this user could exploit this path.



Finding paths between specific objects in AD. This scenario shows the samwell.tarly account's ability to escalate to the Domain Admins group.

Edges

In this scenario, we can see that `samwell.tarly` has the `GenericWrite`, `WriteOwner`, and `WriteDacl` permissions over the `STARKWALLPAPER` GPO. If we don't know how this could be exploited, we can click on the edge itself to have the properites open in the right pane. This has information about the edge, to include how to abuse this permission on Windows or Linux based machines.

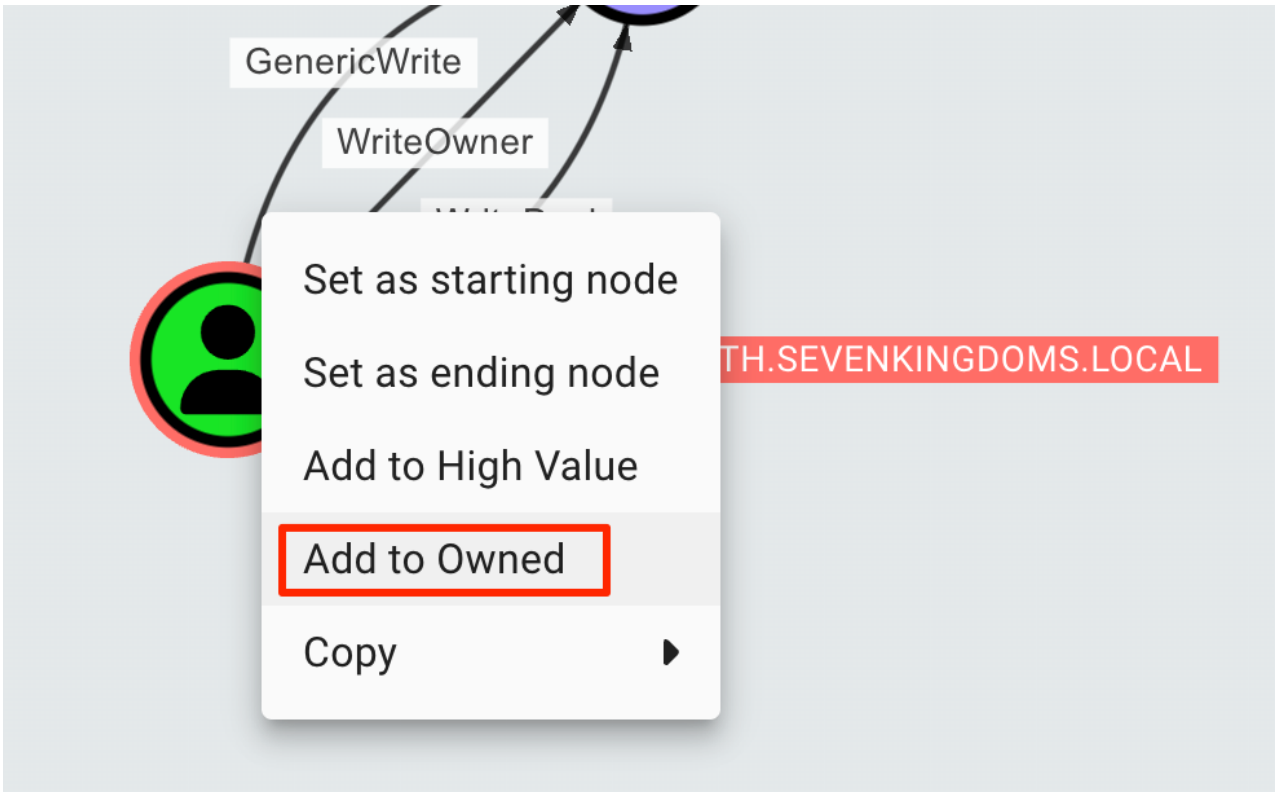


Identifying more information about the edge by clicking on it. This includes how to abuse this case on Windows or Linux based machines.

How cool is that?? It tells us what and how we can use these edges to prove impact. These properties also include several excellent references, which are always worth reading to learn more about a specific abuse scenario.

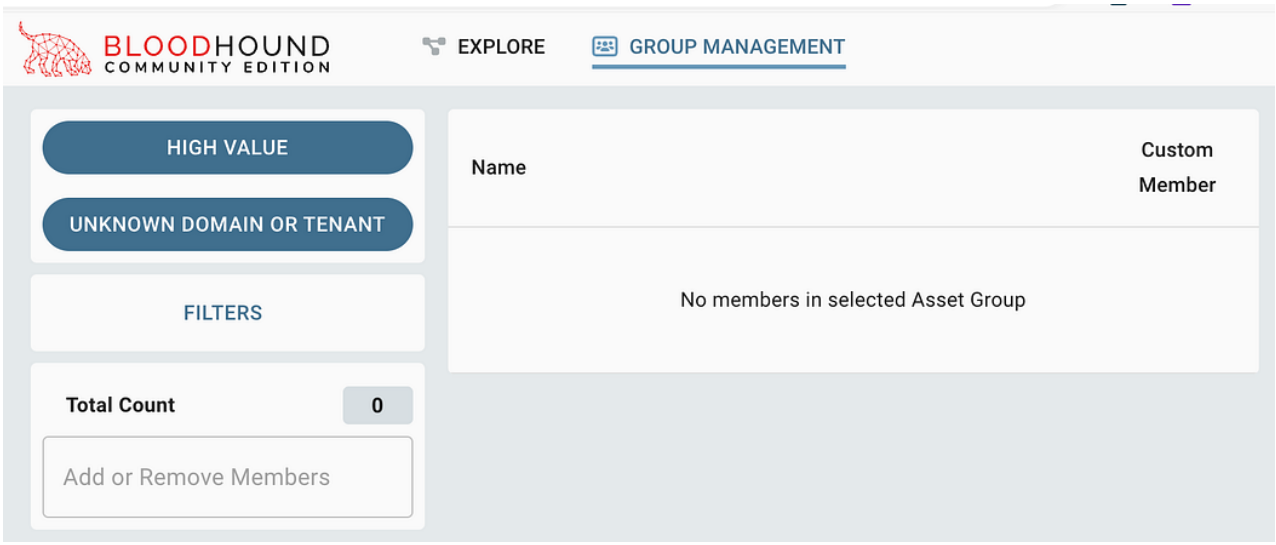
Marking Objects as Owned

To mark an object as owned, we can right click on them in the GUI. This will place a skull icon on their object, allowing us to perform additional queries based on owned objects. This will help us keep track of how we can maneuver within the environment as we continue to gain access.



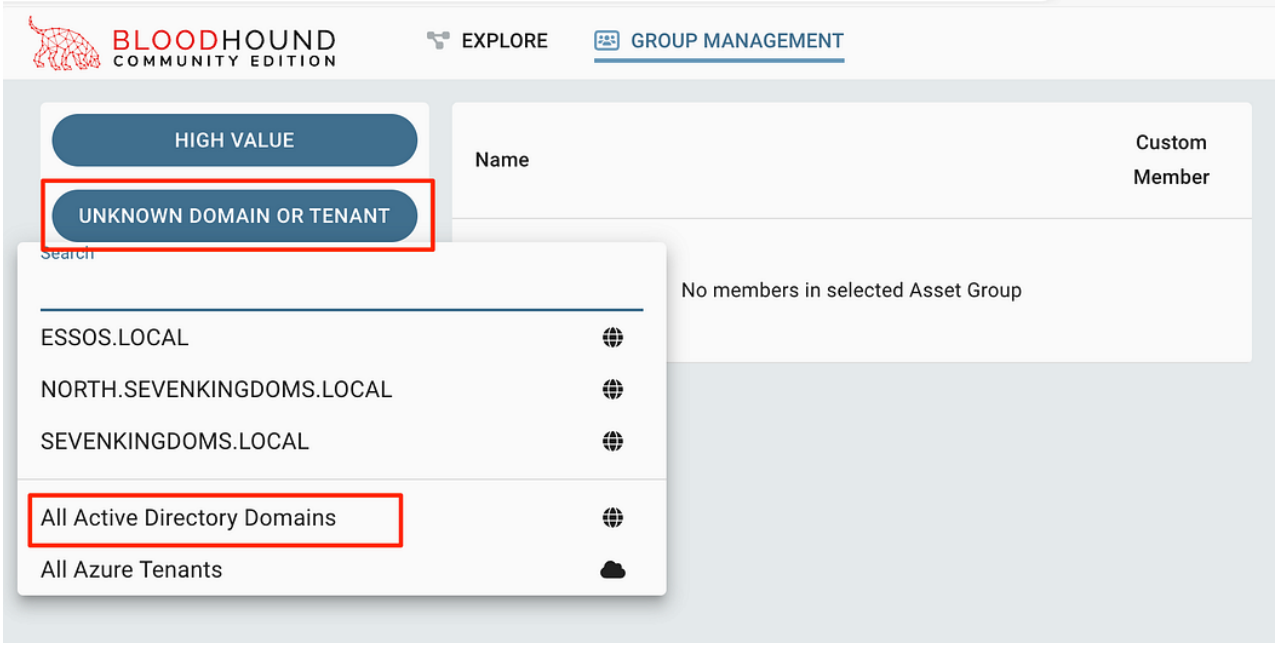
Marking an object as owned. This indicates that we know we can use that object at a later time.

To review objects marked as owned, we can click on the “GROUP MANAGEMENT” button at the top of the page. You’ll notice the page is blank at first, we’ll need to click a few buttons to get the correct information.



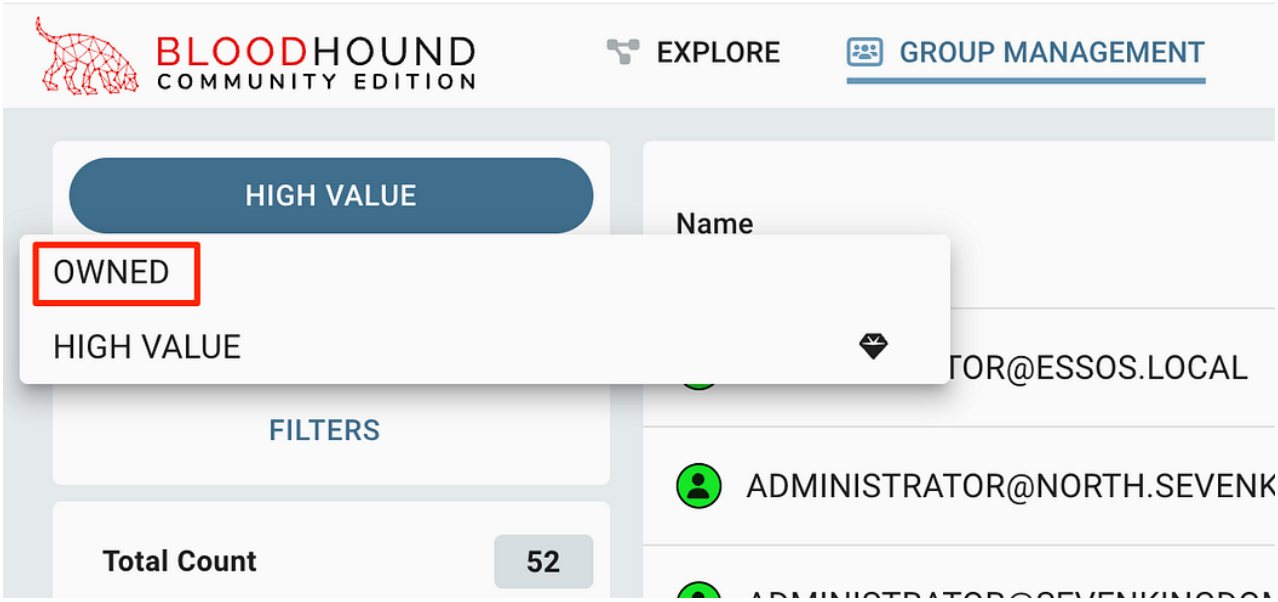
After clicking on GROUP MANAGEMENT, the page appears empty.

Start by clicking the second button to select an entity to search through. In most cases, selecting “All Active Directory Domains” will be enough.



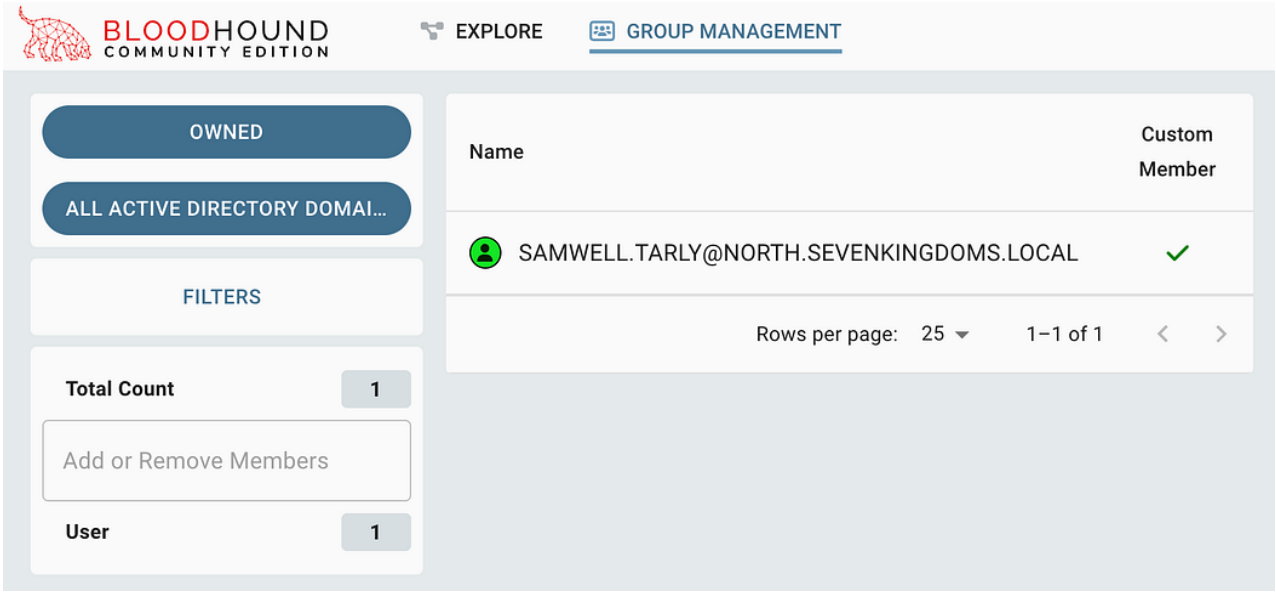
Selecting a domain to view within the Group Management

After, we can click on the top drop-down menu to select “OWNED”.



Selecting OWNED from the drop down to see all objects marked as owned.

At this point, we’ll be able to see the account we marked as owned. This can be useful to help us keep track of objects as we gain access, as well as use in reporting to a customer.



Reviewing all objects marked as owned within Group Management.

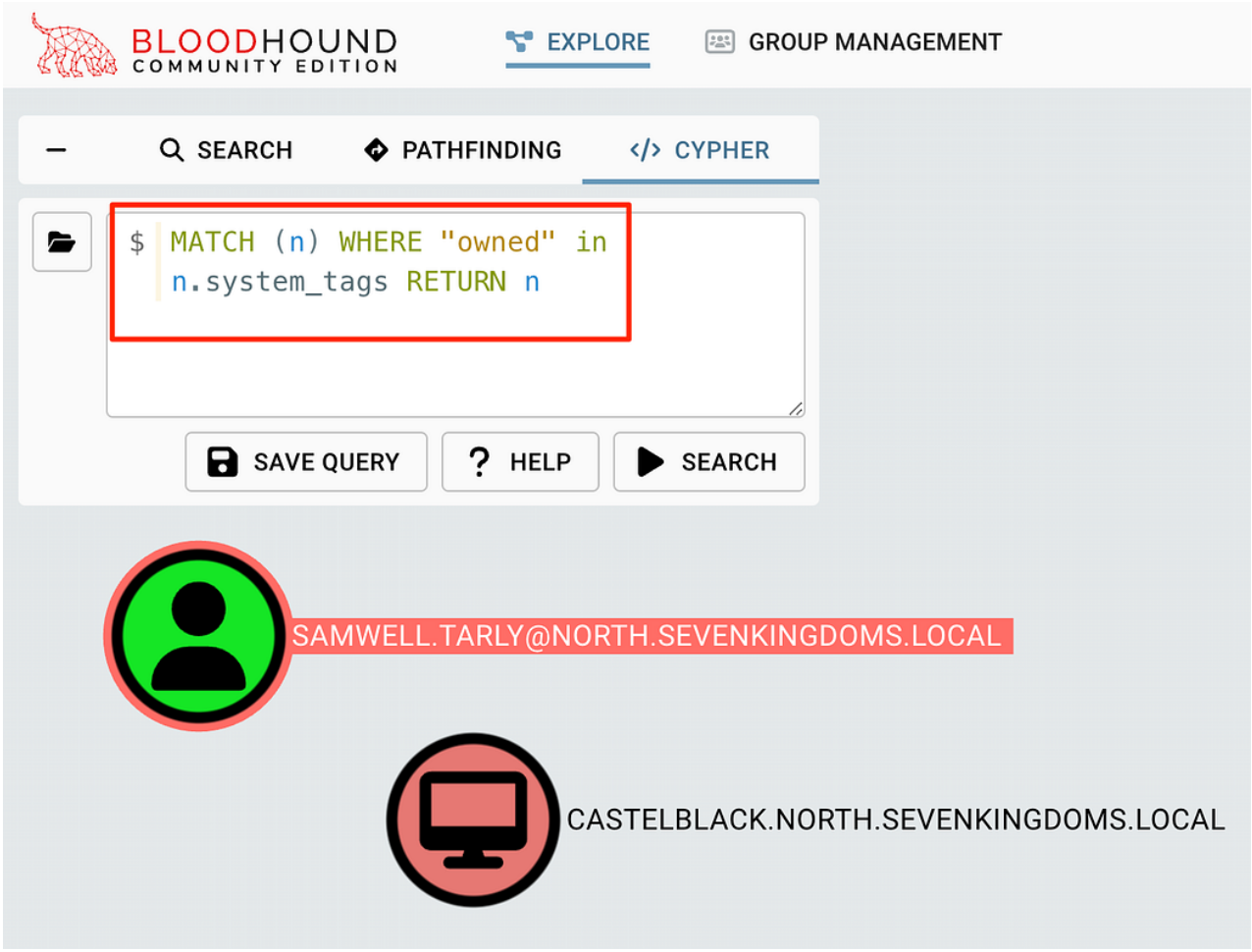
Cypher Queries

While the pre-built searches will be able to help use see interesting items quickly, there are many times where we will need to find something which is not covered by those searches. To solve this, we can create our own queries sent to the Neo4j database. This is how we can validate a search operates in the way we intend it to, where we can save it to a custom search afterwards.

Personally, I feel that the BHCE pre-built queries are missing some critical searches to help match between owned objects and high-value targets. This is handled differently in BHCE vs Legacy and require specific queries in Cypher to achieve it. The way to list all owned Objects in BHCE is below:

```
# List  owned objects (n)  "owned"  n.system_tags  n
```

This can be placed into the Cypher query panel, click search, and see all owned objects. Note that in the image below, the User and Computer objects are both owned, but do not have a path to each other from this search.



Listing all owned objects with a custom cypher query

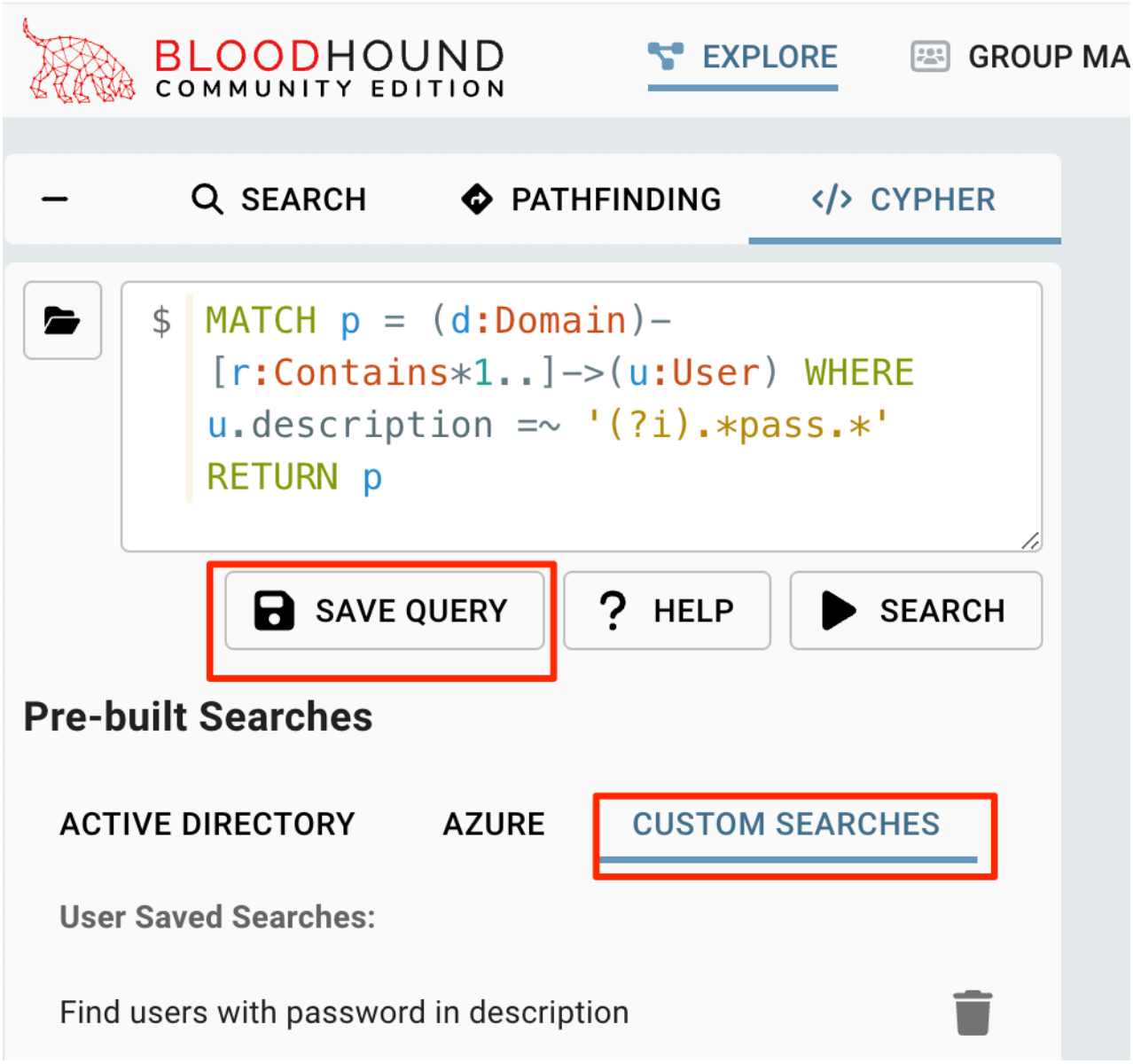
Here's a few useful queries I've been using in the past which have helped me find misconfigurations:

```
# Find all Unconstrained Delegation from non-DCs
MATCH (c1:Computer)-[:MemberOf*1..]->(g:Group) WHERE g.objectid ENDS WITH '-516'
WITH COLLECT(c1.name) AS domainControllers MATCH (c2
{unconstraineddelegation:true}) WHERE NOT c2.name IN domainControllers RETURN c2


# Find users with "pass" in their description
MATCH p = (d:Domain)-[r:Contains*1..]->(u:User) WHERE u.description =~ '(?
i).*pass.*' RETURN p


# List all owned objects
MATCH (n) WHERE "owned" in n.system_tags RETURN n

MATCH p = ((o)-[*..]->(h)) WHERE  in o.system_tags AND  in h.system_tags RETURN p
```

There's some fantastic resources on how to create and understand in-depth Cypher queries on the BloodHound docs if you're interested in learning more:

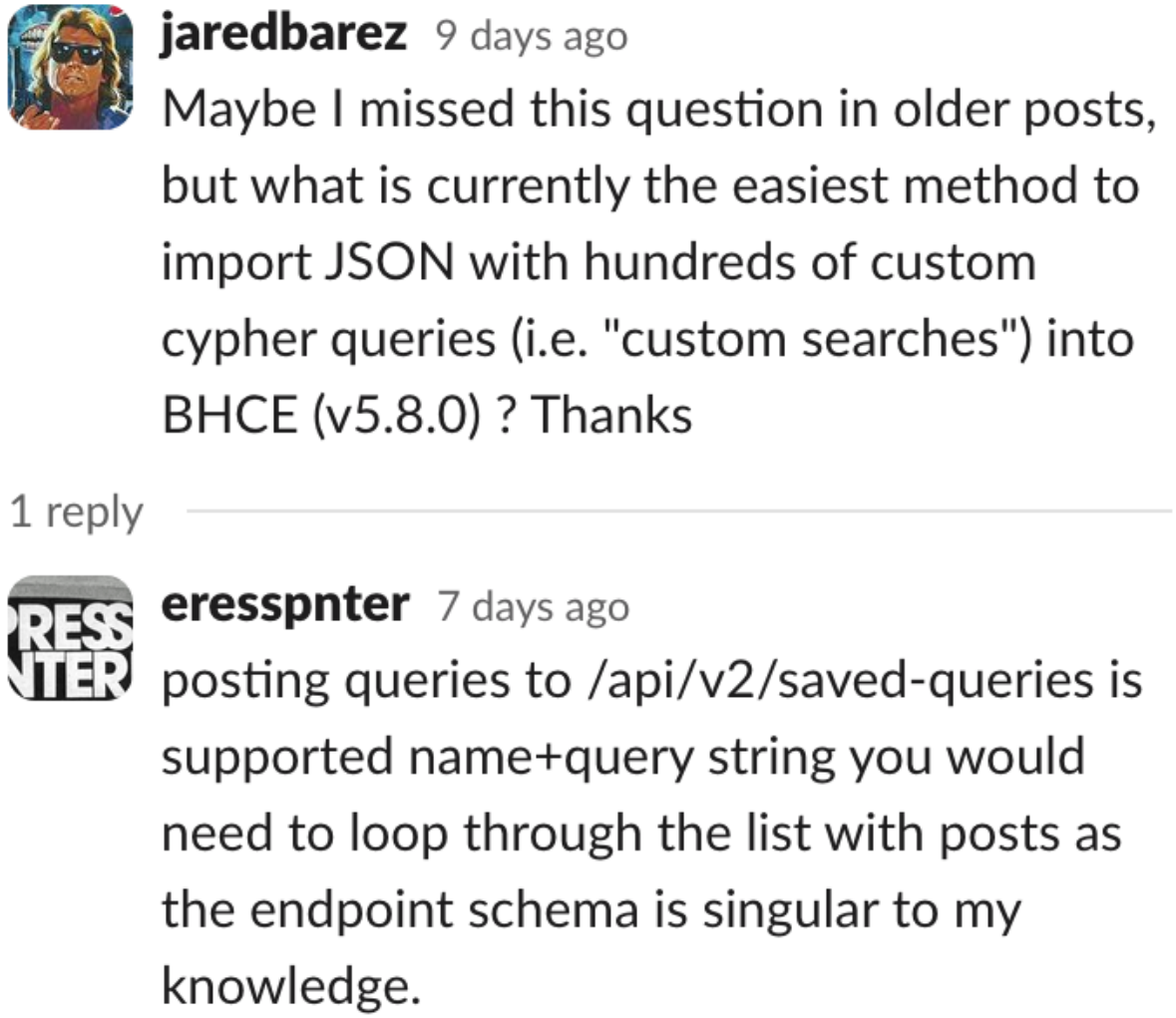
Custom Searches

When we're happy with how a search operates and want to save it for later use, we can save it with the "Save Query" button. This will populate the "Custom Searches" category within the same folder icon for later use.



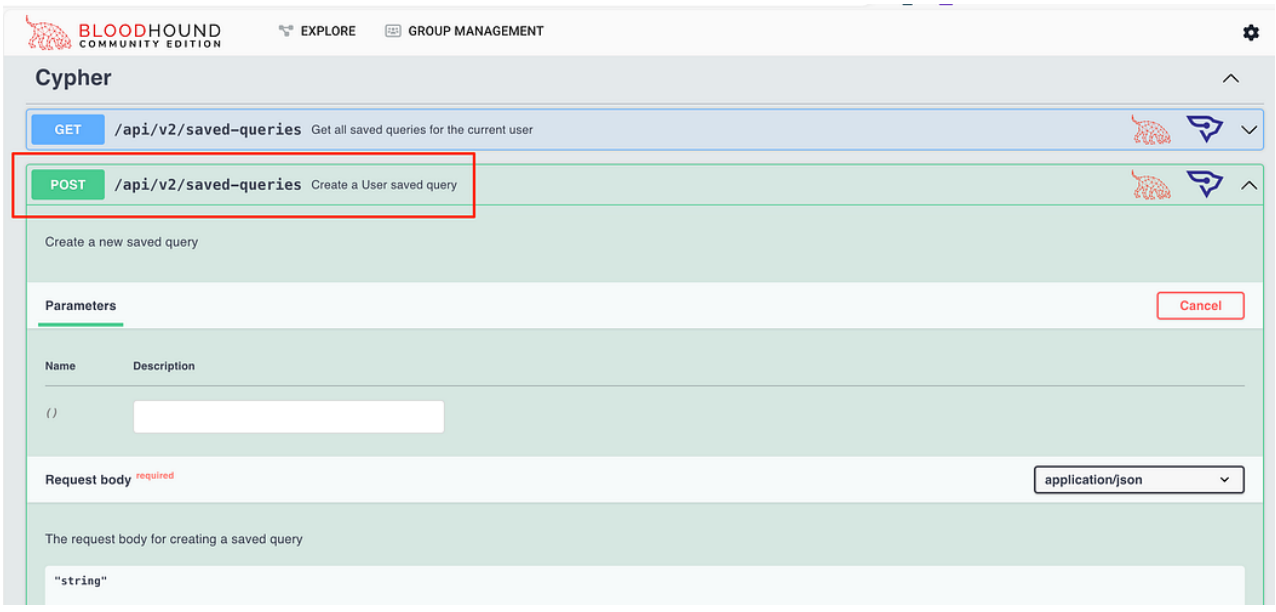
Saving a custom search to continue to use again later.

However, at the moment there doesn't appear to be a way to load the custom queries from disk similar to BloodHound Legacy. This was addressed in the BloodHound Slack a few times, suggesting to use the API instead.



Glad to know I'm not the only one with this problem

Nevertheless, we can use the API docs to try and figure out what we need to do. It's good to note that there are a few different things we can use to read and place new custom-saved queries into the system for us.



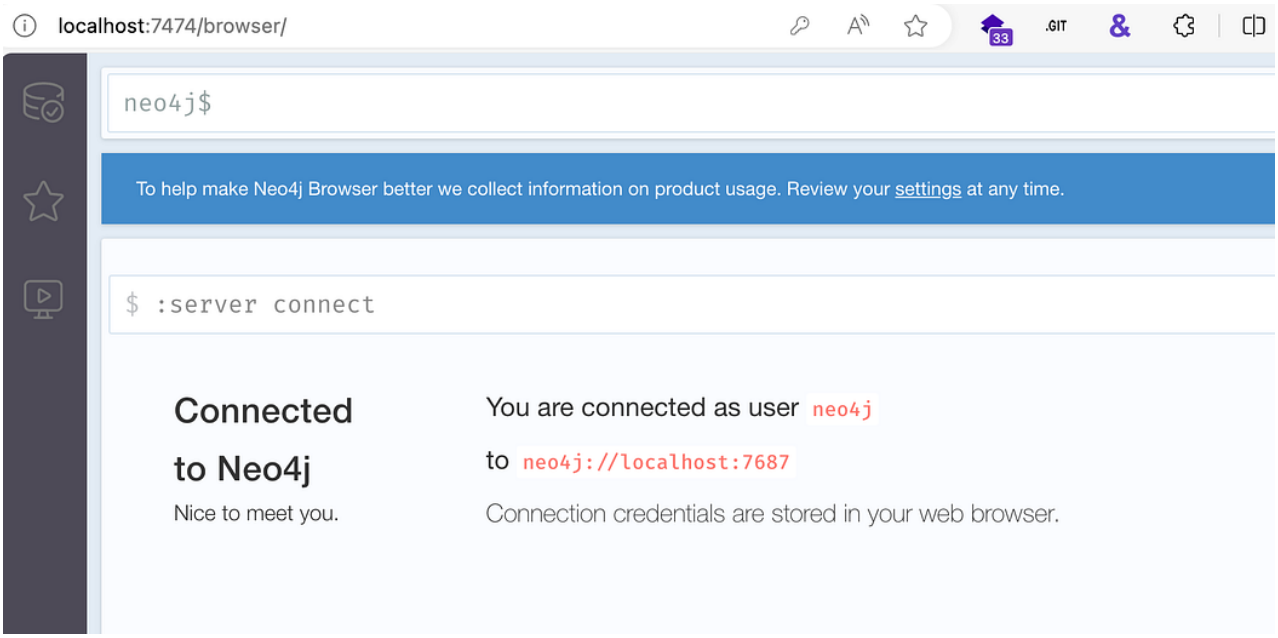
As far as actually uploading these, we can use the `curl` command below to create one uploaded to the GUI. Note that you will need your JWT.

```
curl -X ' \  :  -H : application/json' \  -H -Type: application/json' \  -H :
Bearer {{token}}' \  -d '{:, :}'
```

This is great to upload one or two queries, but would be terrible to do for hundreds of them. So instead, let's use our python scripting skills. I'll probably put this in a follow-up blog post 😁

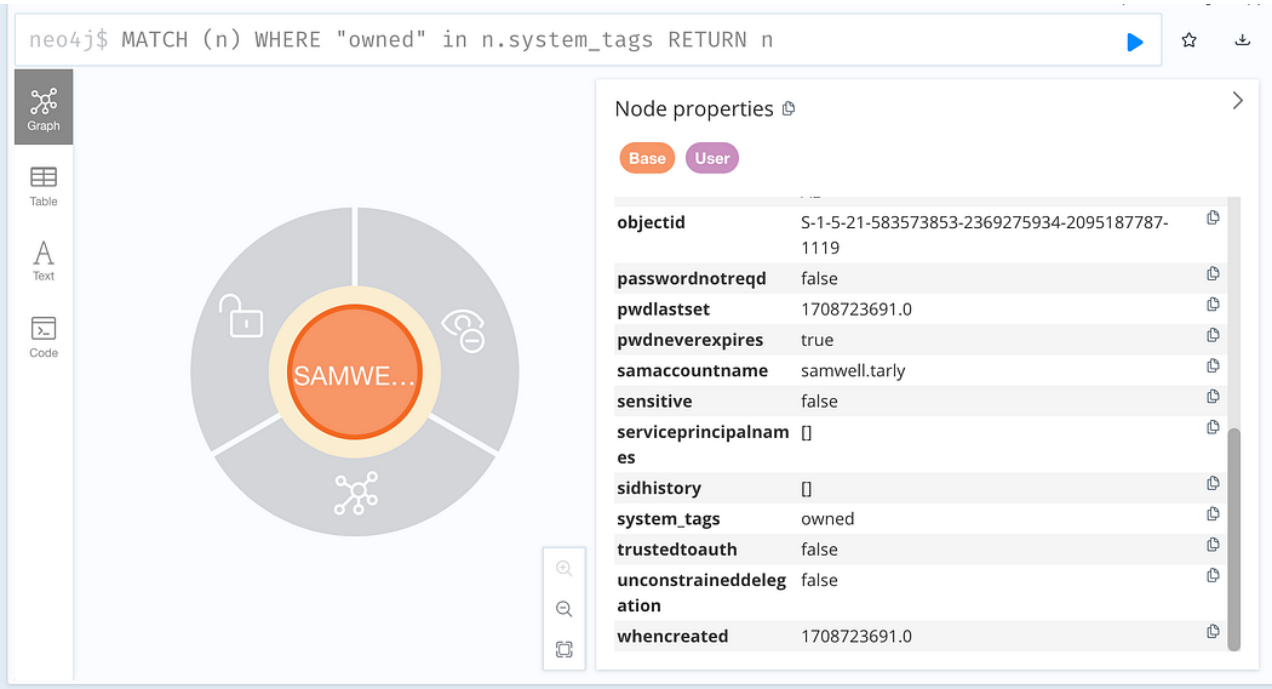
Neo4j Web Console

In the event that we need to directly access the web console for Neo4j, we can do this by browsing to http://localhost:7474. For most scenarios, this is not necessary unless we need direct access to the database. This is the web interface for the Neo4j database and will allow us to run raw cypher queries and review the data. Log into this application with the default `neo4j:bloodhoundcommunityedition` creds in the `docker-compose.yml` file.



Connecting to the Neo4j web browser.

We can place the raw queries into the prompt and see the results, as well as all of the properties for each object returned.



Reviewing the properties for returned objects within the Neo4j web console.

This can help us create and debug custom queries for BHCE. Again, access to the Neo4j web console is generally unnecessary in most scenarios, but it good to have as a backup if needed!

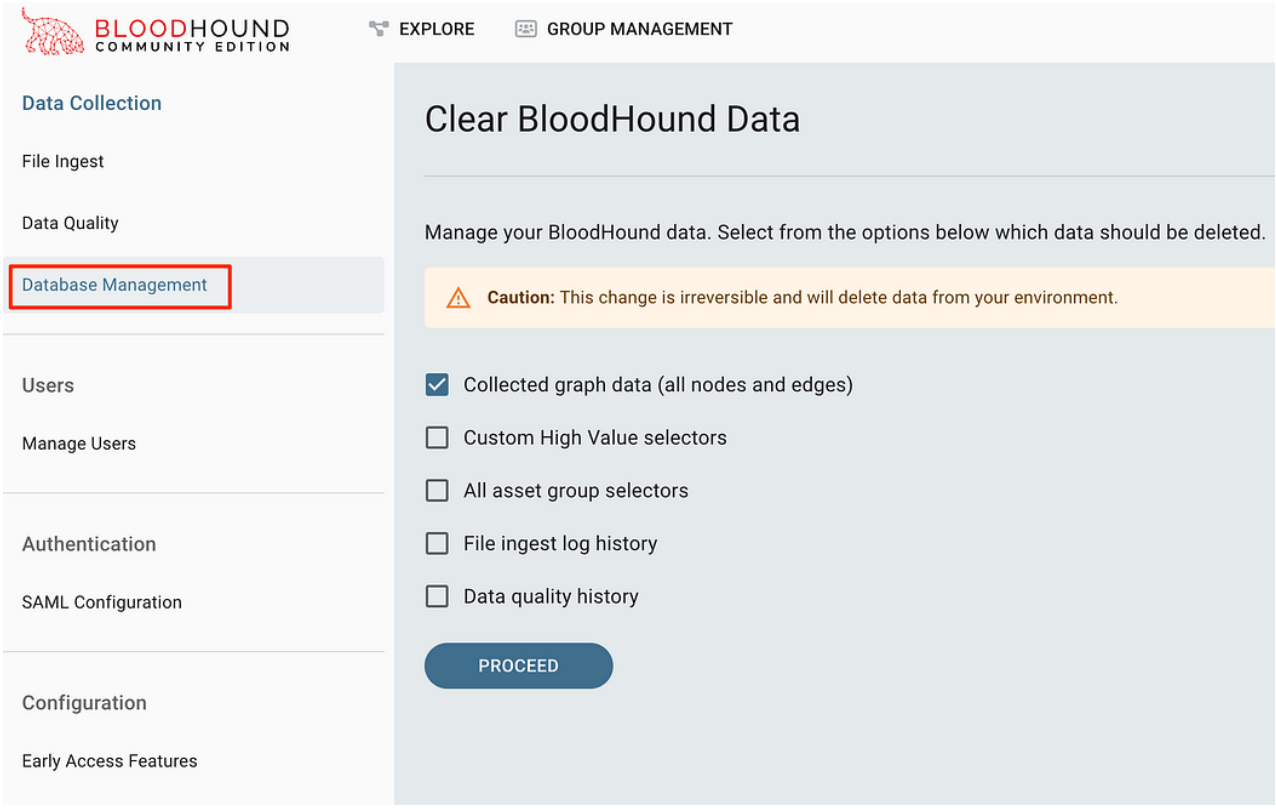
Sample BHCE Data

I've had a hard time locating sample data for BHCE online. There seems to be a significant amount of sample data for legacy BloodHound, but it's harder to find for BHCE. To solve this, I've created a repo of sample data from the GOADv2 range.

This data can be downloaded from the repository below:

Clearing Data

In most consulting environments, we'll need to make sure we clear out the BloodHound data to keep data separated between customers. In the most recent release of BloodHound 5.8.0, this can now be accomplished entirely in the GUI.



Ability to clear data directly from the GUI now exists as of 5.8.0

Prior to version 5.8.0, we needed to remove the volume used by BloodHound in order to clear this out. I've left these instructions in here in case someone else needs to know how to accomplish this, this is documented in issue #107 for BHCE as well.

Let's start by listing the volumes in use by docker first:

```
docker volume
```

Within this data, we can see there are two volumes related to BloodHound.

```
┌──(chris㉿kali)-[~/Desktop/bloodhound][Sat Mar 30 11:32:58 AM EDT 2024]
└─$ docker volume ls
DRIVER    VOLUME NAME
local     578c198ddb80777536632995c6982cdc363088a5fd5e505d4fff9dfeedb7ad40
local     7399ceabb46fa78200eb26c1367a54e0222f074d84ccfbfab6e2055ff8c9cb69
local     a6ca4d6102cc94f55fb7e98cf552747dec2d9d858510a108f31b8e62e1d6a484
local     ae4ae7b57372f36415be697641aa8c069e22753d9712366735ac7f8f3a82fef8
local     bloodhound_neo4j-data
local     bloodhound_postgres-data
```

Identifying the Neo4j volume used by BloodHound

To remove this data and get a fresh instance of BloodHound data, we need to remove this volume. Note that the `bloodhound_postgres-data` volume is used for logging into the GUI and web application. We generally don't want to remove this, unless you want everything reset.

Let's remove this volume to reset the data. We'll use the command below to ask docker to remove this volume.

To simplify things, we can accomplish this all in a single one-liner.

```
docker volume  $(docker volume  -q | grep neo4j-data)
```

Nuke Everything

If you want to nuke everything and pull a fresh copy of all containers, volumes, and configurations, follow these instructions. These assume that you have the `docker-compose.yml` file in the working directory.

We need to first take the containers down and remove the volumes with the command below:

```
docker-compose down -v
```

After, we can pull a fresh copy of the containers with the pull command.

```
docker-compose pull
```

Then we can start the containers as usual, and we should have the latest version of the containers! Note that you will need to set the initial password again, since we removed those previous volumes.

```
docker-compose up
```

Stopping Containers

Stopping the containers is straight-forward, we can just ask docker to stop them. This will stop the containers from serving the application.

```
docker compose stop
```

```
~/dev/bloodhound 1m 22s
❯ docker compose stop
[+] Stopping 3/3
 ✔ Container bloodhound-bloodhound-1  Stopped
 ✔ Container bloodhound-graph-db-1    Stopped
 ✔ Container bloodhound-app-db-1      Stopped
```

Stopping containers, nothing too fancy.

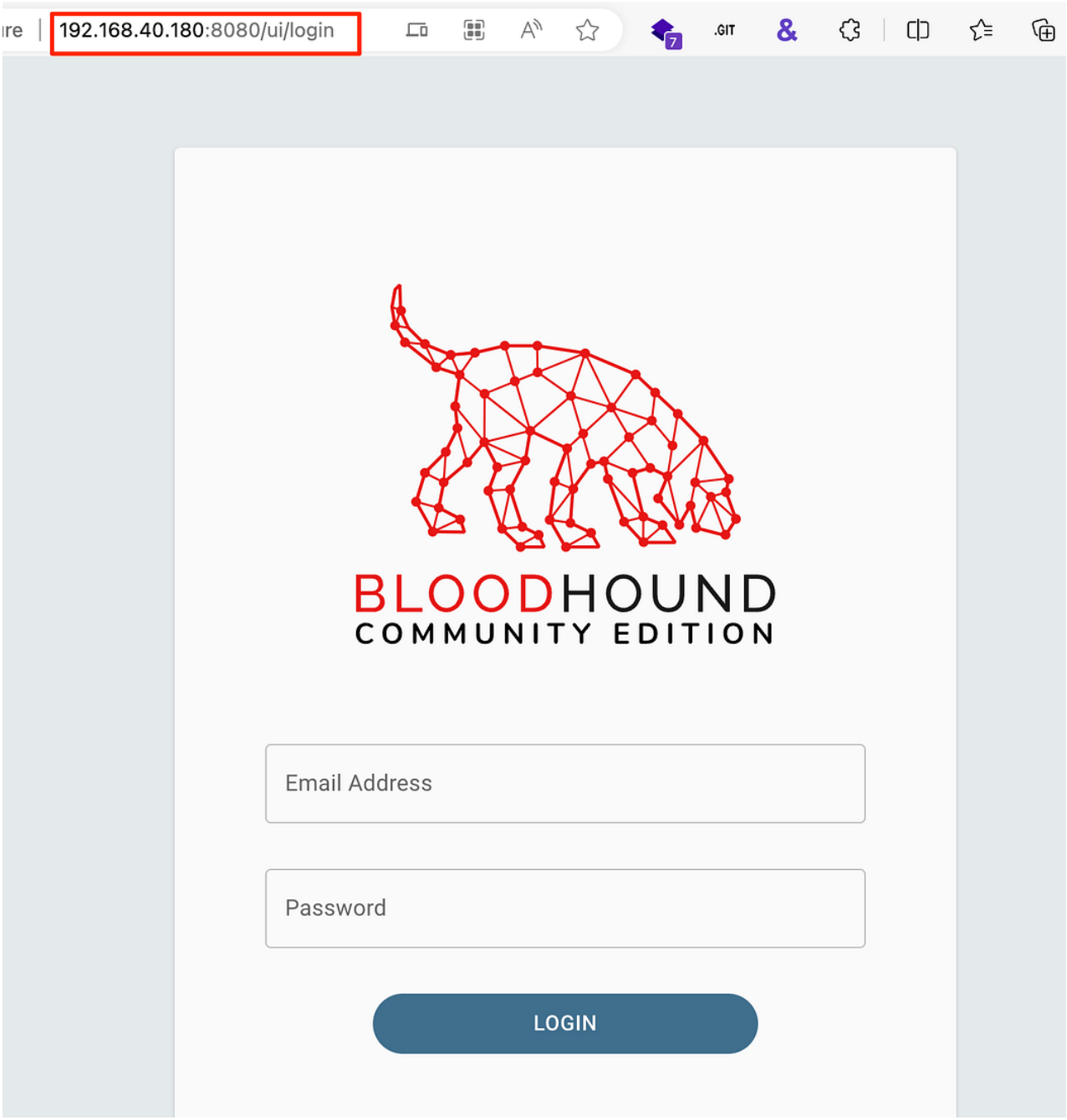
Exposing to Others

In many consulting environments, we'll want to share our instance of BHCE to others so they can all connect to one instance. By default, BHCE is only exposed to the localhost. To achieve this, we'll need to modify the `docker-compose.yml` file to expose it to others.

By changing the line below within the `bloodhound` service, we can tell Docker Compose to expose the GUI to interfaces other than localhost. This is great if we're planning on using a single server for many pentesters to use concurrently.

```
bloodhound:
  image: docker.io/specterops/bloodhound:${BLOODHOUND_TAG:-latest}
  environment:
    - bhe_disable_cypher_qc=${bhe_disable_cypher_qc:-false}
    - bhe_database_connection=user=${POSTGRES_USER:-bloodhound} password=${POSTGRES_PASSWORD:-bloodhoundcommunityedit
    - bhe_neo4j_connection=neo4j://${NEO4J_USER:-neo4j}:${NEO4J_SECRET:-bloodhoundcommunityedition}@graph-db:7687/
    ### Add additional environment variables you wish to use here.
    ### For common configuration options that you might want to use environment variables for, see `.env.example`
    ### example: bhe_database_connection=${bhe_database_connection}
    ### The left side is the environment variable you're setting for bloodhound, the variable on the right in `${}`
    ### is the variable available outside of Docker
  ports:
    ### Default to localhost to prevent accidental publishing of the service to your outer networks
    ### These can be modified by your .env file or by setting the environment variables in your Docker host OS
    - ${BLOODHOUND_HOST:-127.0.0.1}:${BLOODHOUND_PORT:-8080}:8080
  ### Uncomment to use your own bloodhound.config.json to configure the application
  # volumes:
  #   - ./bloodhound.config.json:/bloodhound.config.json:ro
  depends_on:
    app-db:
      condition: service_healthy
    graph-db:
      condition: service_healthy
```

If we want to expose this to all interfaces, change the `BLOODHOUND_HOST` setting to `0.0.0.0`. Note that this will expose it to the Internet if you have that server Internet accessible! Instead, it's usually a better idea to bind it only to a VPN interface such as a WireGuard interface to limit exposure.



Accessing the BHCE GUI across the network instead of only localhost

Accelerating

If you know me, you know that I live to accelerate processes and make things more gooder. So how do we leverage this to get things done faster?

AD-Miner

The AD-Miner toolset leverages the data within Neo4j to search for many known risks and escalation paths, then presents these findings within an HTML file with an overall score.

We can run this by first installing the `AD-Miner` tool via `pipx` then providing the information to connect to the Neo4j database — these credentials are stored in the `docker-compose.yml` file. By default, they are `neo4j:bloodhoundcommunityedition`.

Note that this is the password to the Neo4j database, no to the BHCE GUI!

```
# Install via pipx
pipx install 'git+https://github.com/Mazars-Tech/AD_Miner.git'

AD-miner -u neo4j -p bloodhoundcommunityedition -cf GOAD
```

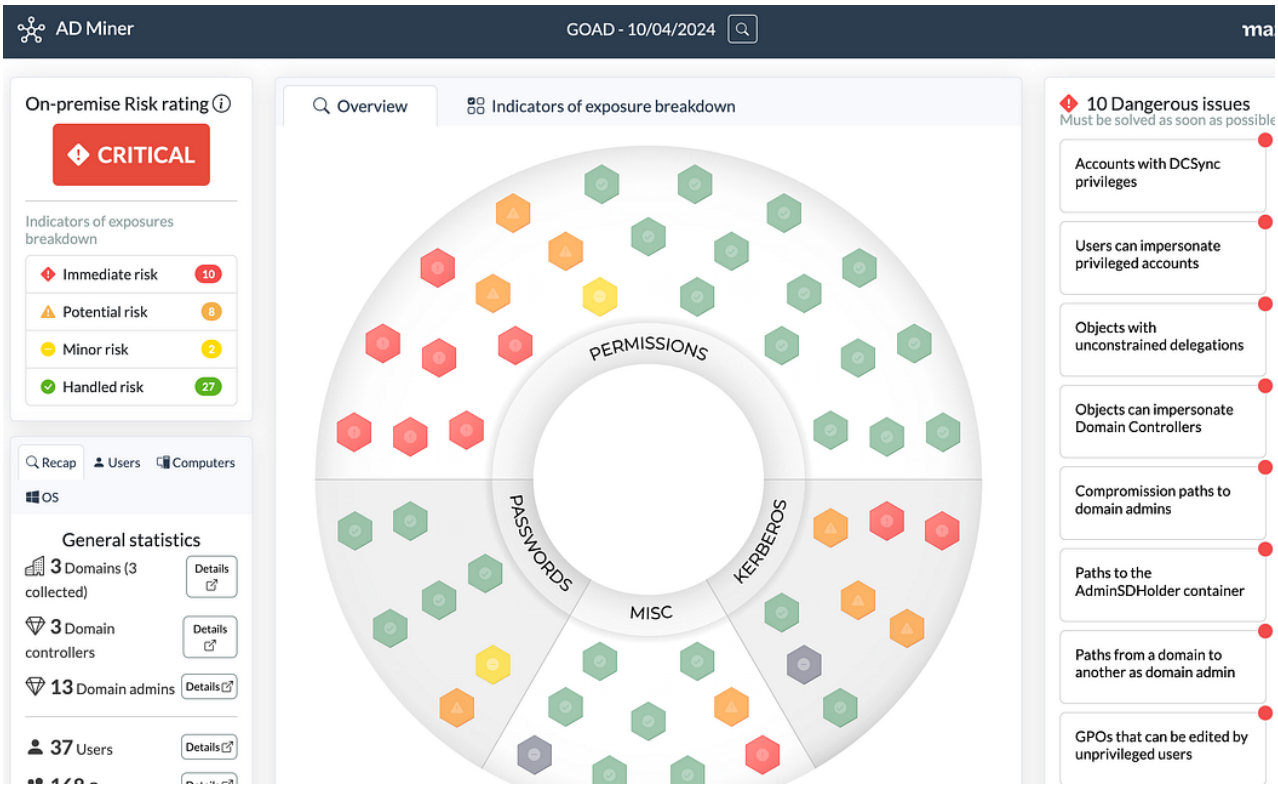
```
~/dev/bloodhound
❯ AD-miner -u neo4j -p bloodhoundcommunityedition -cf GOAD
[+]MigrationData : 1 | Computer : 5 | ADLocalGroup : 1 | Group : 169 | User : 49
[1/145] [+]Requesting : Checking if Graph Data Science neo4j plugin is installed
[+]GDS plugin not installed.
[+]Not using exploitability for paths computation.
[-]Done in 0.04 s - 1 objects
[2/145] [+]Requesting : Delete orphan objects that have no labels
[-]Done in 0.03 s - 0 objects
[3/145] [+]Requesting : Clean AD Miner custom attributes
[-]Done in 0.04 s - 0 objects
[4/145] [+]Requesting : Delete objects for which SID could not resolved
[-]Done in 0.03 s - 0 objects
[5/145] [+]Requesting : Set domain names to upper case when not the case
```

Running the AD-miner collection and analysis tool to generate the HTML file.

This can take a long time depending on the size of the environment — in some domains I've had to wait over 2 hours! Once this has completed, the files will be in a folder named `render_GOAD` using the label provided within the `-cf` switch. We can find this HTML file in the folder created.

```
~/dev/bloodhound/render_GOAD
❯ ll
total 24
drwxr-xr-x   10 chris  staff   320B Apr 11 06:42 .
drwxr-xr-x   19 chris  staff   608B Apr 11 06:41 ..
drwxr-xr-x   13 chris  staff   416B Apr 11 06:15 assets
drwxr-xr-x   14 chris  staff   448B Apr 11 06:15 css
drwxr-xr-x    2 chris  staff    64B Apr 11 06:41 csv
-rw-r--r--    1 chris  staff   5.2K Apr 11 06:42 data_GOAD_20240410.json
drwxr-xr-x  173 chris  staff   5.4K Apr 11 06:42 html
drwxr-xr-x   55 chris  staff   1.7K Apr 11 06:15 icons
-rw-r--r--    1 chris  staff    59B Apr 11 06:41 index.html
drwxr-xr-x   18 chris  staff   576B Apr 11 06:41 js
```

Finding the HTML file for the AD-Miner analysis



Analysis of GOAD based on Neo4j data

How cool is that? We now have an interactive dashboard which brings the most critical misconfigurations to our attention. We can use all of this information to help customers and environments reduce risk and exposure.

AD-Miner is a fantastic tool to help provided additional analysis and enrichment from the BloodHound data — definitely check it out!

Uploading Saved-Queries

Since there's plenty of queries we know we want to explore that are not in the pre-built ones, we can upload them via the API. Again, this is probably something that I'll write at a later time as a follow-up post 😁

Conclusions

While this was pretty comprehensive, hopefully it was helpful! I've seen far too many analysts either become frustrated by not knowing how to use this amazing tool or not use it to its potential. Use it, learn it, and love it.

This certainly wouldn't be possible without the development effort and technical excellence from the team! Huge thanks to the AD-Miner team from Mazars for their tool as well. This would be exceptionally more difficult without the GOAD set to analyze, thanks to Mayfly for that work!

If you're not already in the BloodHound Slack, you should join! There's always tons of great discussions and analysis.

Finally, if you've enjoyed this post, please consider pre-ordering my book I've co-authored, The Hack is Back: Techniques to Beat Hackers at Their Own Games. It's set to release in August of 2024 😁

Updates

17DEC2024: This has been updated to reflect that the `docker-compose` package is no longer available via Debian repos. This binary can be downloaded directly from GitHub instead.

References