Enriching BloodHound Data

blog.spookysec.net/Enriching-BloodHound-Data

02 Aug 2023

I’m going to preface this post with BloodHound is an awesome tool. It’s one of the best tools in my mind that’s ever been introduced to Information Security; It empowers Pentesters, Red Teams and Identity Security professionals to identify and remediate attack paths within their own environment. Since their announcement of “BloodHound Community Edition” (BHCE) on 8/1, I’ve always wanted to write a blog post on how you can extend BloodHound’s functionality. I’m still a ways off of that, but I still wanted to throw something out there as a celebration to SpecterOps and how you can enrich your data.

So, what’ll we be discussing today? Excellent question dear writer! We’re going to be talking about enriching information on Login Information and Kerberoastable users. Though, first I want to talk a little bit about BloodHound/SharpHound and it’s current architecture/functionality so we’re all on the same playing field as I’m going to geek out on Neo4j in a little while. So let’s dive into it!

BloodHound/SharpHound/Neo4j Basics

For those of you who aren’t aware, BloodHound is an electron application that interfaces with a a graphing database called Neo4j. Neo4j is fairly simple to digest, you have Nodes and Relationships. Some practical examples of Nodes are:

- Users
- Computers
- Domains
- OUs More information on Nodes specific to BloodHound can be found here on their Nodes page within the docs.

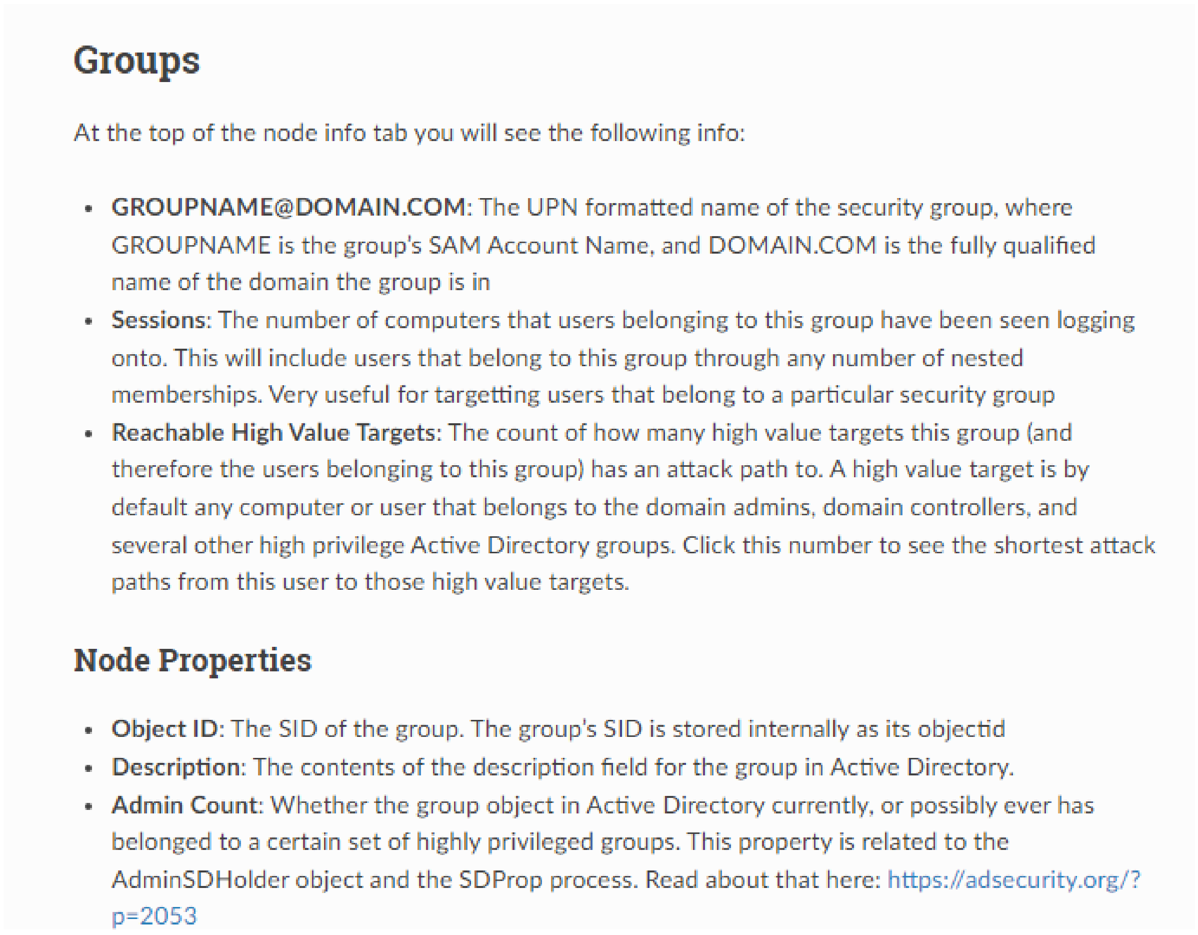
Some practical examples of relationships between nodes are:

- adminTo
- canRDP
- GenericWrite
- memberOf
- Owns More information on Relationships (or Edges) can be found here on their “edge” analysis page..

And properties of nodes could be:

- isOwned
- lastLogonTimestamp
- pwdLastSet

- hasSPN Within the Nodes page above, there is a list of properties. An example can be found below:



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

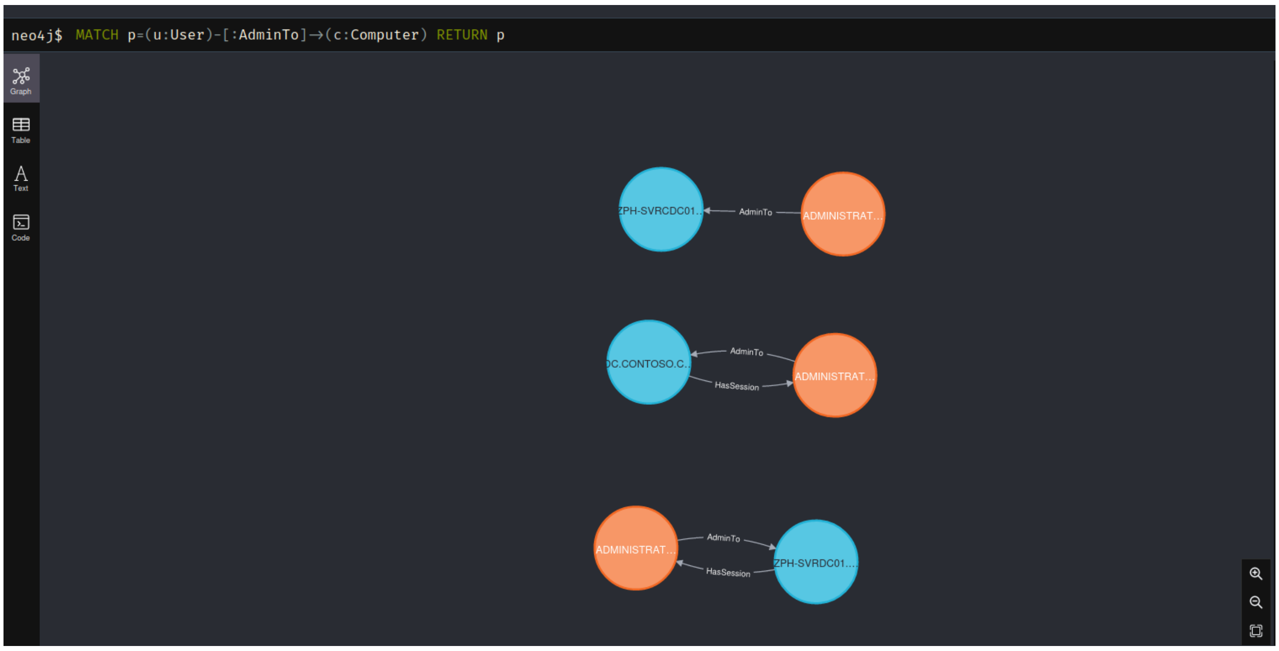
Ultimately, BloodHound tries to find relevant paths between nodes to achieve a specific path to a specific thing based on the current level of access you have. Most often this can be described by a query like “Shortest path to Domain Admin from $X”. $X can be a principal you have owned, or are targeting.

BloodHound has a ton of incredibly useful prebuilt queries to help you find that, so you don’t need to learn the query language for Neo4j. If you do however, you can make some really cool things happen. Cypher is the language Neo4j uses to query their graphing database, it’s kind of like SQL, it looks kind of the same, but is different. It’s pretty easy to pick up.

Cypher 101 - So, as we established before, BloodHound/Neo4j has Nodes, Relationships and Node properties. Let’s say we wanted to find all the computers that a user is Admin to in the environment. We could find this by performing the following query:

```
MATCH p=(u:User)-[:AdminTo]->(c:Computer) WHERE u.name =~ "(?i)dave.+" RETURN p
```

This query creates a variable, P. P contains a search for two nodes. A User and a Computer that is connected via a relationship. The relationship we care about here is AdminTo. Running this query in BloodHound/Neo4j **should** work if the data exists.



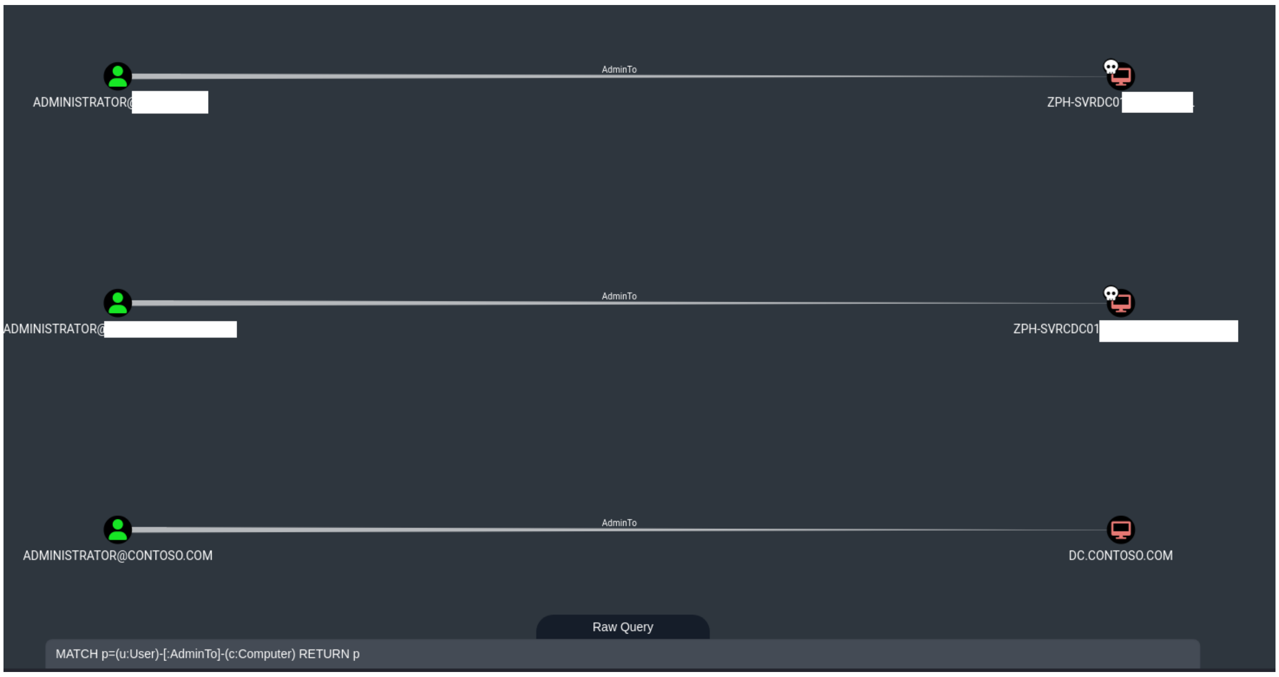
```
neo4j$ MATCH p=(u:User)-[:AdminTo]→(c:Computer) RETURN p
```

We can see that we have a couple of sets of nodes where this is true. You may notice there is an extra set of relationships here that we didn't reference in the query, this being "HasSession". Neo4j likes to be more verbose than is necessary sometimes. Sometimes this is a great thing as you can see multiple interesting relationships when you execute a query like:

```
MATCH (u:User)-[:HasSession]->(c:Computer)
```

You may discover (for example) that a user is an Administrator on that device and also has a session on that device. Important stuff.

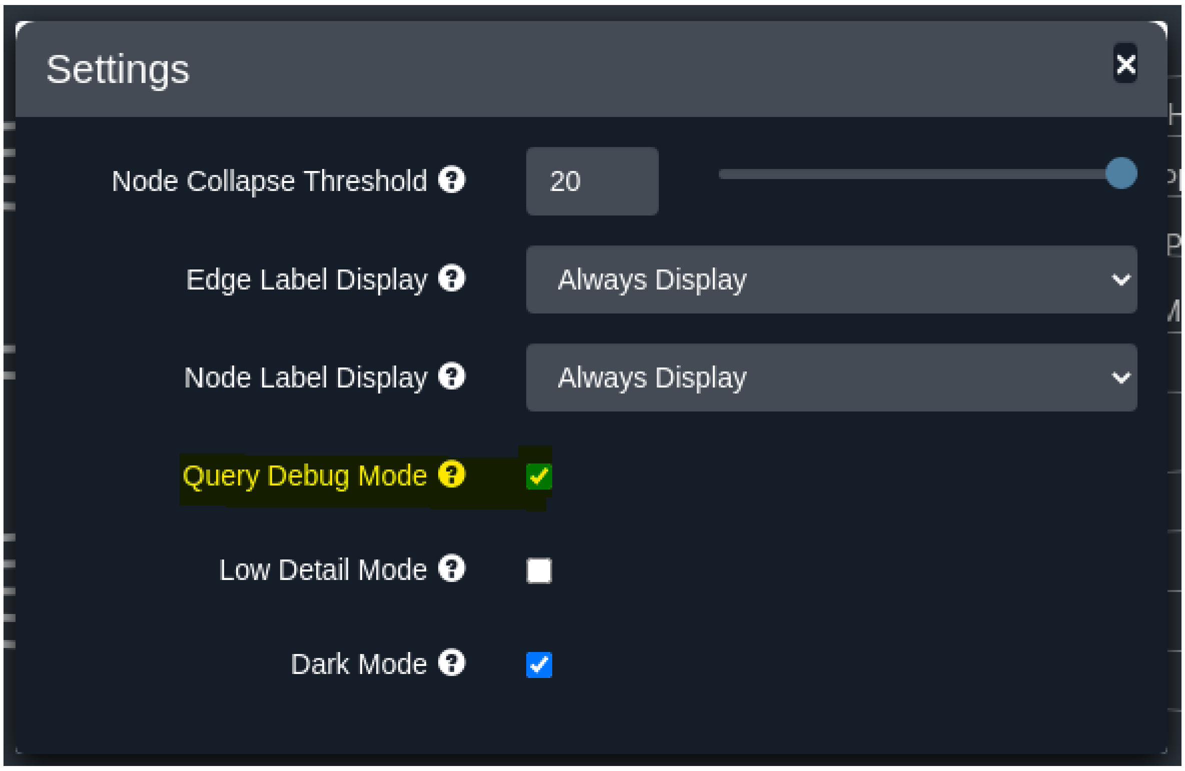
The same query can be executed in BloodHound and we should expect to see the same, or at least similar results.



```
MATCH p=(u:User)-[:AdminTo]-(c:Computer) RETURN p
```

Given this information here, how could we modify this query to search for Groups that are Admins to these computers? Or how could we modify the query to find out who Owns these computers? Remember that a Group or a User could own the computer. Research Cypher and figure out how you could query “Any Node type that has a relationship to this specific node”. It shouldn’t take too much tweaking :D

This wraps up our Neo4j primer. If you find yourself confused, I’d recommend reading the Neo4j Cypher documentation and looking at some BloodHound Query CheatSheets. You can also enable “Query Debug Mode” which prints the queries that are being executed. This can be done by selecting the settings cog on the right and checking the checkbox.



Settings

Node Collapse Threshold: 20

Edge Label Display: Always Display

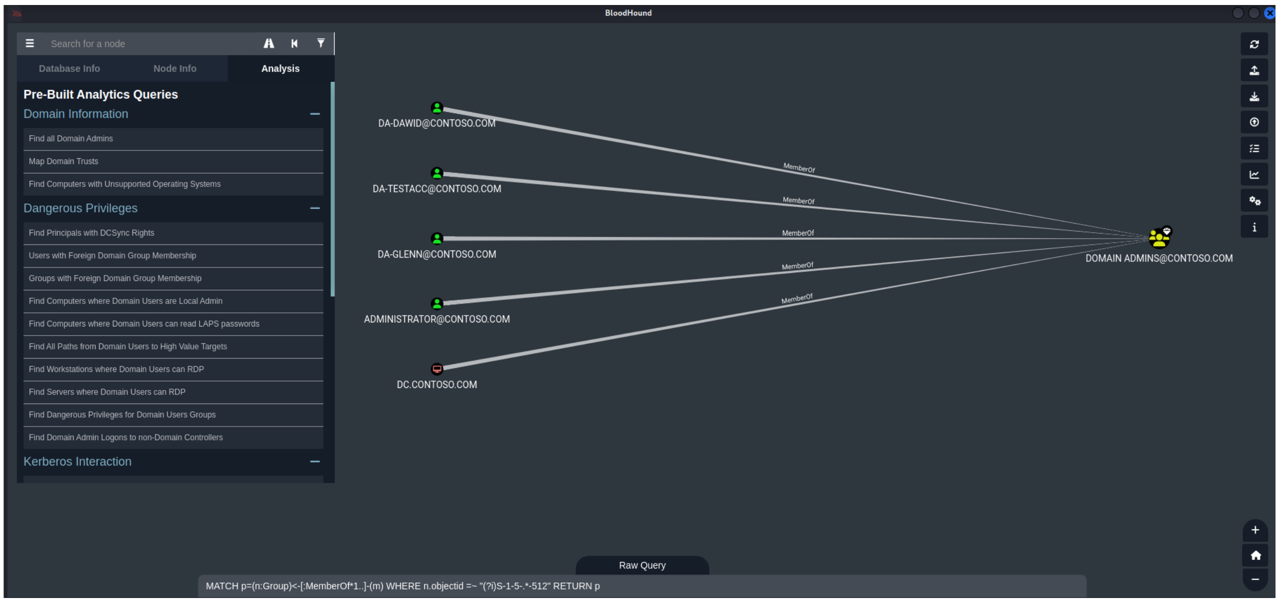
Node Label Display: Always Display

Query Debug Mode: ✓

Low Detail Mode: ☐

Dark Mode: ✓

You may then want to click on the “Raw Query” button. The next time you run a prebuilt query, you’ll see the outputs below. Here is an example of “Find all Domain Admins”:



```
MATCH p=(n:Group)<-[:MemberOf*1..]-(m) WHERE n.objectid =~ "(?i)S-1-5-.*-512" RETURN p
```

BloodHound
Search for a node
Database Info
Node Info
Analysis
Pre-Built Analytics Queries
Domain Information
Find all Domain Admins
Map Domain Trusts
Find Computers with Unsupported Operating Systems
Dangerous Privileges
Find Principals with DCSync Rights
Users with Foreign Domain Group Membership
Groups with Foreign Domain Group Membership
Find Computers where Domain Users are Local Admin
Find Computers where Domain Users can read LAPS passwords
Find All Paths from Domain Users to High Value Targets
Find Workstations where Domain Users can RDP
Find Servers where Domain Users can RDP
Find Dangerous Privileges for Domain Users Groups
Find Domain Admin Logons to non-Domain Controllers
Kerberos Interaction
DA-DAWID@CONTOSO.COM
DA-TESTACC@CONTOSO.COM
DA-GLENN@CONTOSO.COM
ADMINISTRATOR@CONTOSO.COM
DC.CONTOSO.COM
MemberOf
DOMAIN ADMINS@CONTOSO.COM
Raw Query

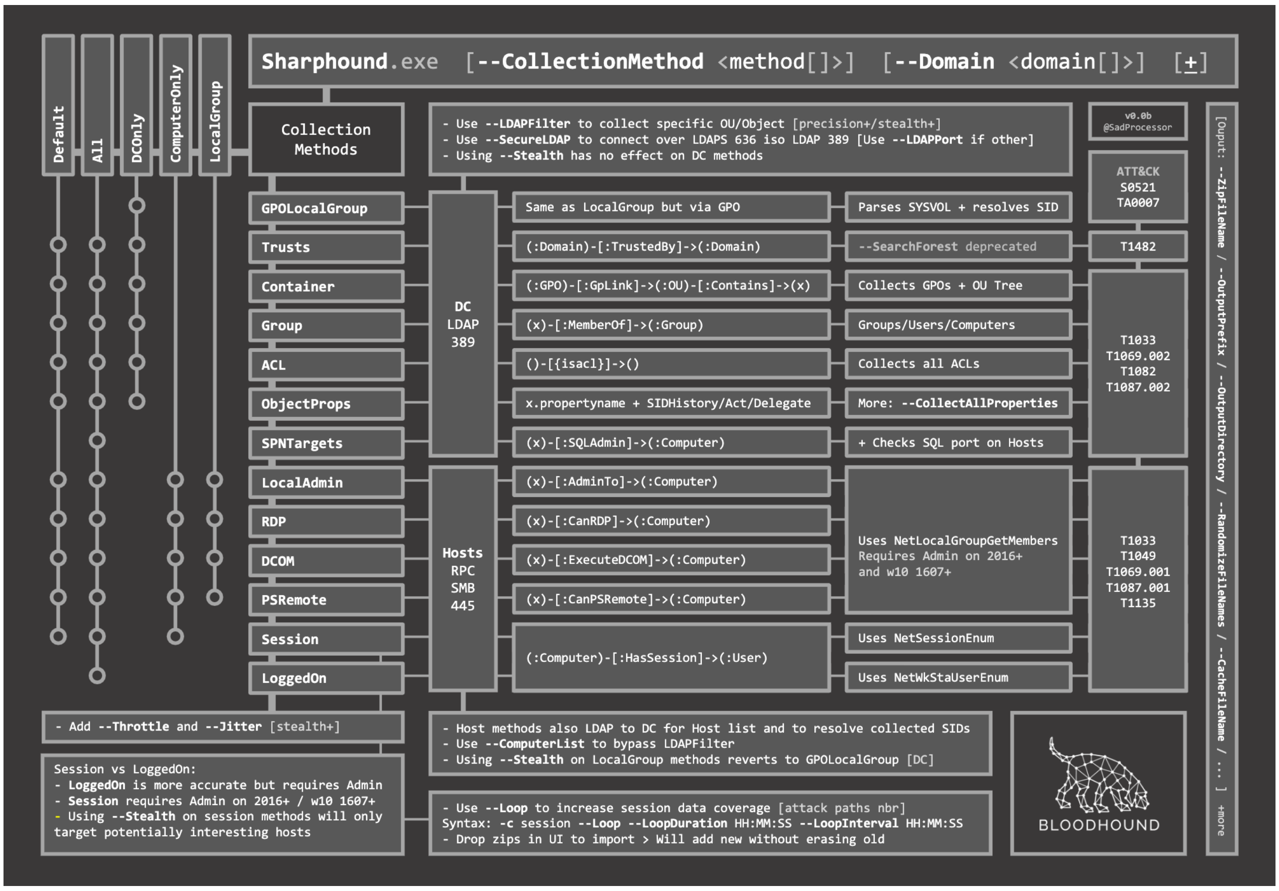
So, we’ve talked about working with this data, but how is it collected? How can you collect it? How can you make your own interesting data?

SharpHound Data Collection

If you’re not aware, SharpHound is the collector for BloodHound - It creates a bunch of LDAP queries that search for user/computer/group data and then writes them out to a JSON file that can be imported through the BloodHound UI. It doesn’t stop there though, based on the flags you supply, it can reach out to the endpoint and query certain data like:

- Administrators to Computers (as we demonstrated above)
- Who can Remotely or Locally Access Devices
- Who is Currently Logged In

and several other things. I recommend you refer to this page in the BloodHound docs that explains all of SharpHounds flags. There is a really awesome Cheat Sheet by @SadProcessor that outlines how each thing is enumerated, what privileges are required and more. It also includes some handy references for Neo4j.


Default
All
DCOnly
ComputerOnly
LocalGroup
Collection Methods
GPOLocalGroup
Trusts
Container
Group
ACL
ObjectProps
SPNTargets
LocalAdmin
RDP
DCOM
PSRemote
Session
LoggedOn
DC LDAP 389
Hosts RPC SMB 445
Same as LocalGroup but via GPO
(:Domain)-[:TrustedBy]->(:Domain)
(:GPO)-[:GpLink]->(:OU)-[:Contains]->(x)
(x)-[:MemberOf]->(:Group)
()-[{isacl}]->()
x.propertyname + SIDHistory/Act/Delegate
(x)-[:SQLAdmin]->(:Computer)
(x)-[:AdminTo]->(:Computer)
(x)-[:CanRDP]->(:Computer)
(x)-[:ExecuteDCOM]->(:Computer)
(x)-[:CanPSRemote]->(:Computer)
(:Computer)-[:HasSession]->(:User)
Parses SYSVOL + resolves SID
--SearchForest deprecated
Collects GPOs + OU Tree
Groups/Users/Computers
Collects all ACLs
More: --CollectAllProperties
+ Checks SQL port on Hosts
Uses NetLocalGroupGetMembers Requires Admin on 2016+ and w10 1607+
Uses NetSessionEnum
Uses NetWkStaUserEnum
v0.0b @SadProcessor
ATT&CK S0521 TA0007
T1482
T1033 T1069.002 T1082 T1087.002
T1033 T1049 T1069.001 T1087.001 T1135
[Ouput: --ZipFileName / --OutputPrefix / --OutputDirectory / --RandomizeFileNames / --CacheFileName / ...] +more
BLOODHOUND

```
Sharphound.exe  [--CollectionMethod <method[]>]  [--Domain <domain[]>]  [±]
```

- Use **--LDAPFilter** to collect specific OU/Object [precision+/stealth+]
- Use **--SecureLDAP** to connect over LDAPS 636 iso LDAP 389 [Use **--LDAPPort** if other]
- Using **--Stealth** has no effect on DC methods

- Add **--Throttle** and **--Jitter** [stealth+]

Session vs LoggedOn:
- **LoggedOn** is more accurate but requires Admin
- **Session** requires Admin on 2016+ / w10 1607+
- Using **--Stealth** on session methods will only target potentially interesting hosts

- Host methods also LDAP to DC for Host list and to resolve collected SIDs
- Use **--ComputerList** to bypass LDAPFilter
- Using **--Stealth** on LocalGroup methods reverts to GPOLocalGroup [DC]

- Use **--Loop** to increase session data coverage [attack paths nbr]
Syntax: **-c** session **--Loop --LoopDuration** HH:MM:SS **--LoopInterval** HH:MM:SS
- Drop zips in UI to import > Will add new without erasing old

After gazing at the wonderful illustration for a minute, there are a couple of things that catches my attention

- LoggedOn/Session enumeration require Admin access on Server 2016+ and Win10 1607+.
- RDP/DCOM/PSRemote and others uses NetLocalGroupGetMembers which require Admin access on Server 2016+ and Win10 1607+

Wow, those are two major hits seeing **a ton** of systems are on or above Win10 1607+ and Server 2016+ now a days. 2016 was a **long** time ago. This effectively means you'll have a massive visibility gap if you're using SharpHound with a low privileged user. Most people do not know this going into BloodHound. If you're not aware of that important caveat, BloodHound's effectiveness can be severely diminished, it can lead to an incomplete dataset that may hide problems. There's a really easy way to solve this, however I don't recommend it. Run SharpHound as a Domain Admin. Again, I strongly **do not** recommend you do this. Especially if SMB Signing is disabled within your environment, or not enforced, as this can lead to the potential for NTLM Relaying.

Skipping back up to the beginning of this section - I mentioned two things that I haven't addressed yet. I briefly mentioned it, but SharpHound is BloodHound's data collector. You can run this as any Domain User using the following command. Please note that any identity security and EDR product will quarantine and flag SharpHound, so please get explicit permission from your SOC/Identity Security team before running it. Communication is key. Once you've got that permission, you can run something as basic as:

```
SharpHound.exe -c all
or
SharpHound.exe -c DCOnly
```

and you'll get your data in a nice .zip file! There's also a Python based collector by Dirkjanm called BloodHound.py. It's almost as fully featured as SharpHound, but there are a few things missing. See PRs and Issues for more info. If you have a **lab environment** aka a Domain Controller and want to generate a bunch of data to play with, you can use a tool like BadBlood or GOAD to automate everything.

So, given that very important disclaimer, how can we modernize Session collection?

Modern User Session Collection using SIEM/EDR

Now it's problem solving time! So, how in the world can we fix an issue like this? Well, there are two really good ways that you could solve it. You might even be thinking of it because it's right above and below you. It being:

- Your EDR
- Windows Event Logs

This is by no means a hard limit. If you have other applications that provides that level of telemetry, by all means feel free to include this in your list. The biggest thing I want to emphasize here is the sky is the limit; If you have the data, understand it, know its integrity, you are in a really good spot and can work on the concept of expanding BloodHound's capabilities. Again, out of scope for this article, we're only focusing on the two above for now.

So, back to Session collection. In modern day Windows, there are two important session types; Network based logons and Interactive logons (or logons that will cache your credentials in memory). This isn't really true, but it categorizes the two important ones. I like to refer to them by their newly given relationships in my Neo4j database. These being:

- HasNetSession
- HasSession

The two have a very important distinction between them, a network session uses NetNTLM to validate their credentials (NetSession) when accessing a network based resource (like an SMB Share or NTLM Web Endpoints), meaning no credentials are stored in memory and the other being credentials are cached in memory (Session). So, how can we get that data from our SIEM(s)?

It's actually pretty simple. I like Splunk, so I'm going to use SPL-style pseudo syntax to provide examples. We can execute a query like so:

```
index=CompanyName sourcetype=WinEvtLogs EventID=4624 AND (LogonType=2 OR
LogonType=4 OR LogonType=5 OR LogonType=7 OR LogonType=9 OR LogonType=10 OR
LogonType=11) | dedup ComputerName,UserName | table ComputerName,UserName
```

This will search for EventID of 4624 (or a successful Windows Logon) for Logon types 2 (Interactive), 4 (Batch), 5 (Service), 7 (unlock), 9 (RunAs), 10 (Remote Interactive Logon), or 11 (Cached Creds). For more information, see here this article from Ultimate Windows Security. This will give us our HasSession oriented relationships as if a user logs by those methods, credentials **should** stored in LSASS, or will be domain cached credentials. The other LogonTypes 3 (Network) and 8 (NetworkCleartext) types may not store it in memory like we hope. So, we can craft a similar but different query to give us that information. It may not be as important to us to collect this type of data, so use your best judgement on if its something you care about!

```
index=CompanyName sourcetype=WinEvtLogs EventID=4624 AND (LogonType=3 OR
LogonType=8) | dedup ComputerName,UserName | table ComputerName,UserName
```

One thing I will note is it may aid in threat hunting or insider threat identification if you can visualize who is connecting to what servers/devices over network based sessions over a short/long period of time. Continuing on!

We can also make some inferences like if a user logs on with the LogonType of 10, they have the ability to RDP to the device. This may be something you want to include and expand upon.

Okay, so that's great and all, but what are my options if my company doesn't use WEF/WEC? Great question, as I mentioned before, it's possible to get this data with your EDR - Note that this may not have 100% visibility, just like your WEF/WEC. I'd recommend using a hybrid of the two to get the best visibility *if possible*.

So - This is a bit more theory driven but has been tested and used. If you have a cloud-based EDR that has a full fledge SIEM (MDE, CrowdStrike, Elastic, etc.) you may be able to leverage that to collect the same data, just in a bit of a round about way.

Process Execution data can help here if your EDR isn't capturing logon events. Sometimes that data is exclusive to their identity product (ex. MDI or Falcon ITP). Because EDR is mostly process and endpoint protection centric. So, keeping this in mind we can craft a query that looks like so:

```
index=Events sourcetype=ProccessCreation peName=explorer.exe | dedup
ComputerName,UserName | table ComputerName,UserName
```

Note that this is completely fictional field names. I don't have any of the products on hand to test with, nor do I know KQL (Kusto or Kibana) well enough to craft an example query to demo. Though the methodology here is if a user creates a process (explorer.exe), this must mean that they have logged into the system and triggered an interactive logon. This isn't perfect though, as the results from RunAs (for example) aren't from a child process of explorer.exe. Though, one can infer that if an application is running on the system as a user that they have signed in, or the credentials are in memory somehow. Like I said, this one requires a bit more theory behind it. You can work it and adjust based on your tolerances. This is just what I've had luck with in the past.

Now you're likely left with a CSV list of data and you're probably wondering "How am I going to get this data into Neo4j/BloodHound"? Great question. Let's talk about it in the next section.

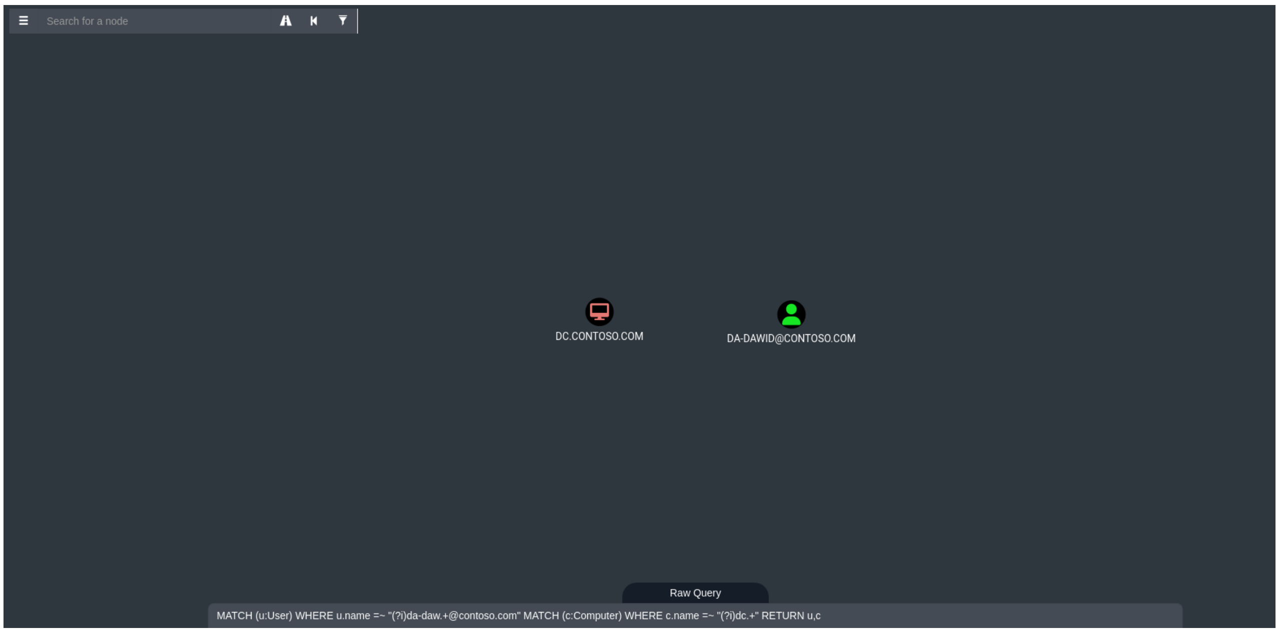
Loading New Data into Neo4j

This section is a bit of a fun one as it involves writing code & Neo4j queries (Yay!!). First, I want to mention that you **do not** have to write code. Neo4j provides a way to insert data via CSVs. You can do this if you like, however, I think working with the Neo4j Python library is fairly straight forward to use. Let's dive into it - as stated before, we're after creating a relationship between two nodes. This is a fairly trivial process in Neo4j, before we get started, let's craft a query that we'll use to model the rest of our queries off of.

According to the Neo4j documentation, we need to match our two nodes, this can be done with the following query:

```
MATCH (u:User) WHERE u.name =~ "(?i)da-daw.+@contoso.com" MATCH (c:Computer) WHERE
c.name =~ "(?i)dc.+" RETURN u,c
```

This should return our two nodes to the screen:



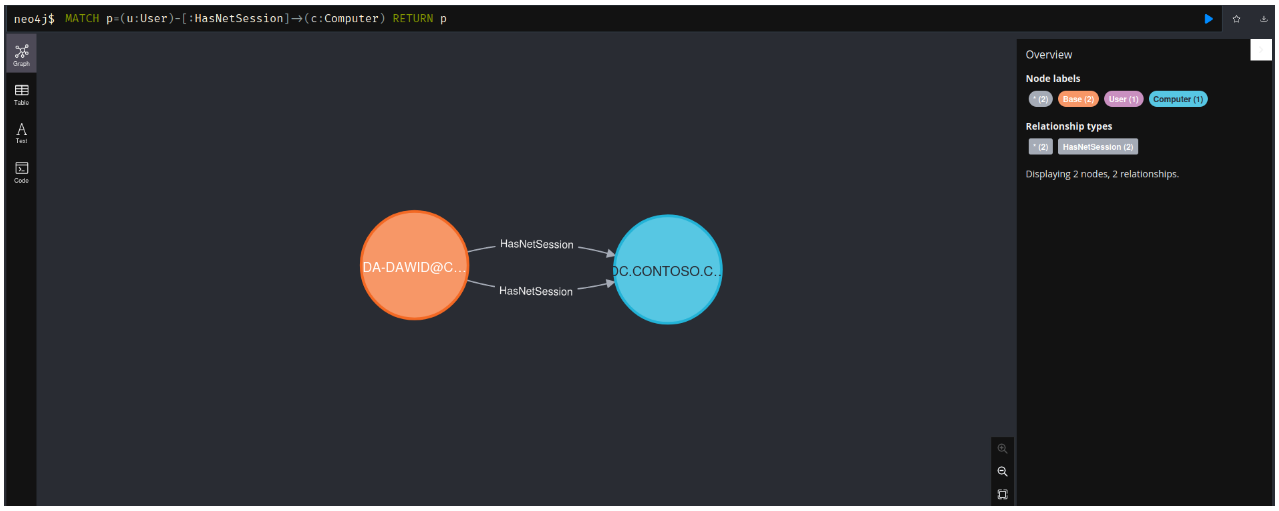
```
MATCH (u:User) WHERE u.name =~ "(?i)da-daw.+@contoso.com" MATCH (c:Computer) WHERE c.name =~ "(?i)dc.+" RETURN u,c
```

Now, we can *create* a relationship between the two nodes by adding this to the query:

```
MATCH (u:User) WHERE u.name =~ "(?i)da-daw.+@contoso.com" MATCH (c:Computer) WHERE
c.name =~ "(?i)dc.+" CREATE (u)-[r:HasNetSession]->(c)
```

Entering this query will return no data. That is to be expected as this reduces the overall processing power required for the workload. Imagine if you had to draw all the nodes to the screen before the query completed. Yikes. Okay! Now, let's verify our query:

```
MATCH p=(u:User)-[:HasNetSession]->(c:Computer) RETURN p
```



```
neo4j$ MATCH p=(u:User)-[:HasNetSession]→(c:Computer) RETURN p
```

Overview

Node labels: * (2), Base (2), User (1), Computer (1)

Relationship types: * (2), HasNetSession (2)

Displaying 2 nodes, 2 relationships.

Success! We have a template query that we can build off of. Alls we need to do is make sure we have some wiggle room with RegEx. This may lead to some false positive results *if* there's two servers with similar names (ex. Exchange and Exchange-DR), though I think we can tolerate that level of risk, as it can manually be validated and cleaned up later.

So, the Neo4j library (depending on the documentation you read) can be pretty difficult to use, so here's roughly what our code is going to look like. It's a bunch of data handling (stripping some characters that may jack things up, connecting to Neo4j, running a query and running our query).

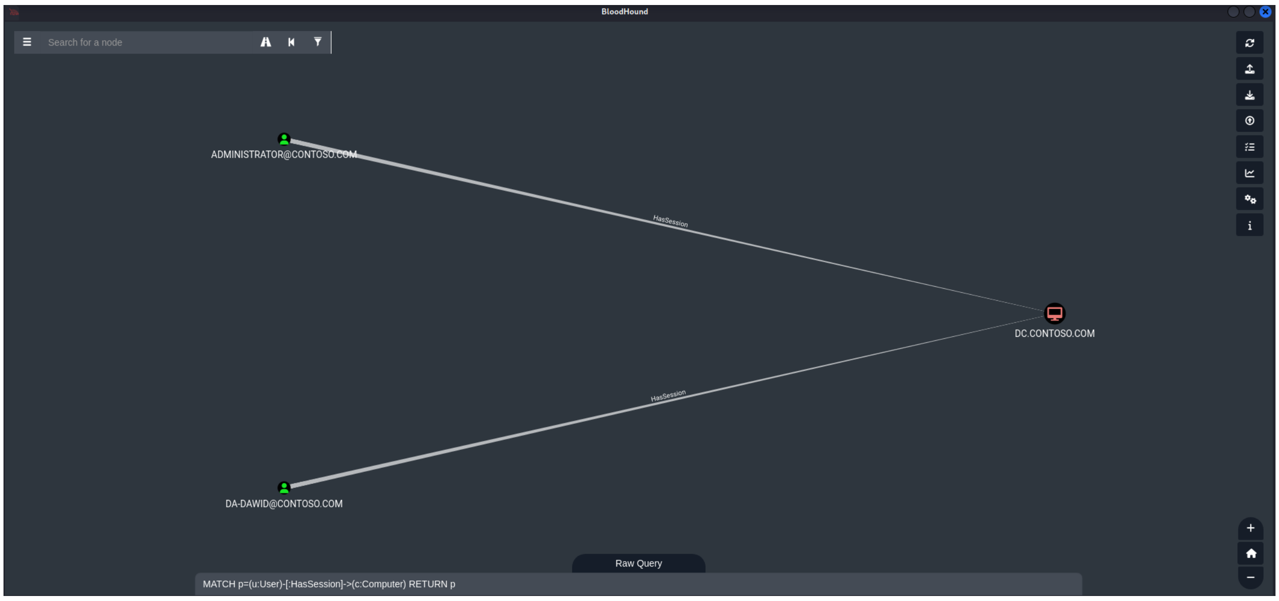
```
#!/usr/bin/python3
from neo4j import GraphDatabase
# Variables, I'm not going to implement arg parsing here because its a bit
overkill
neo4jUsername = "neo4j"
neo4jPassword = "bloodhound"
neo4jIP = "neo4j://localhost:7687"
csvFile = "file.csv"
f = open(csvFile, "r")

# Connecting to the Neo4j Database
try:
        neo4jdbDriver = GraphDatabase.driver(neo4jIP, auth=(neo4jUsername,
neo4jPassword))
except Exception as e:
        print("An error occured when connecting or logging into Neo4j: " + str(e))

# Looping throug the lines in the file
for LINE in f:
        #Stipping unnecessary characters out to ensure the N4j query doesn't fail
        n4jlist = LINE.split(",")
        username = n4jlist[0].strip("\n")
        username = username.strip("\"")
        username = username.strip("'")
        computer = n4jlist[1].strip("\n")
        computer = computer.strip("\"")
        computer = computer.strip("'")
# Debug print statement
        print("Creating Session Relationship for " + username + " on " + computer)
        # Our query template
        queryTemplate = 'MATCH (u:User) WHERE u.name =~ "(?i)' + username + '.+"
MATCH (c:Computer) WHERE c.name =~ "(?i)' + computer + '.+" CREATE (u)-
[r:HasSession]->(c)'
        # Printing our query template
        print(queryTemplate)
        try:
                # Using the Neo4j driver to execute our query against the Neo4j
database.
                neo4jdbDriver.execute_query(queryTemplate, database='neo4j')
        except Exception as e:
                print("An error occured while executing the query: " + str(e))
```

This would take in a CSV named file.csv which contains some data that looks like so:

```
root@pandorasbox~ cat test.csv
da-dawid,dc01
administrator,workstation42
```

We could modify this to take in a third column which could be a relationship value. This may make it a bit more flexible as we could merge our two lists (HasNetSession and HasSession) and differentiate the two by the third column. This is merely a suggestion though. I'd encourage you all to go try to implement that yourselves. It shouldn't be too big of a lift!

After executing the Python Script, we should receive an error or so complaining that execute_query may be removed… Just ignore that. We can now see our results below!



```
MATCH p=(u:User)-[:HasSession]->(c:Computer) RETURN p
```

If you scale it up, I'd recommend functionalizing it and implementing multi-threading. Be warned, Neo4j transactions can time out, so rate limiting is important too. Or figuring out how to execute bulk queries.

So, that's about it. That's how you load data into Neo4j. The introduction of HasNetSession vs HasSession provides some extra truth about how the user is connected to the device - Over a network based session (which may introduce the idea of relaying traffic to that host to a remote server) as a new attack vector that you could try. Asset and User intelligence is the name of the game here. I wish I could show you all live results of a production environment because it is **really** interesting. Especially when you execute a query that's like, "Show me all the users that have a SMB session on 5 or more hosts". It paints a really interesting picture.

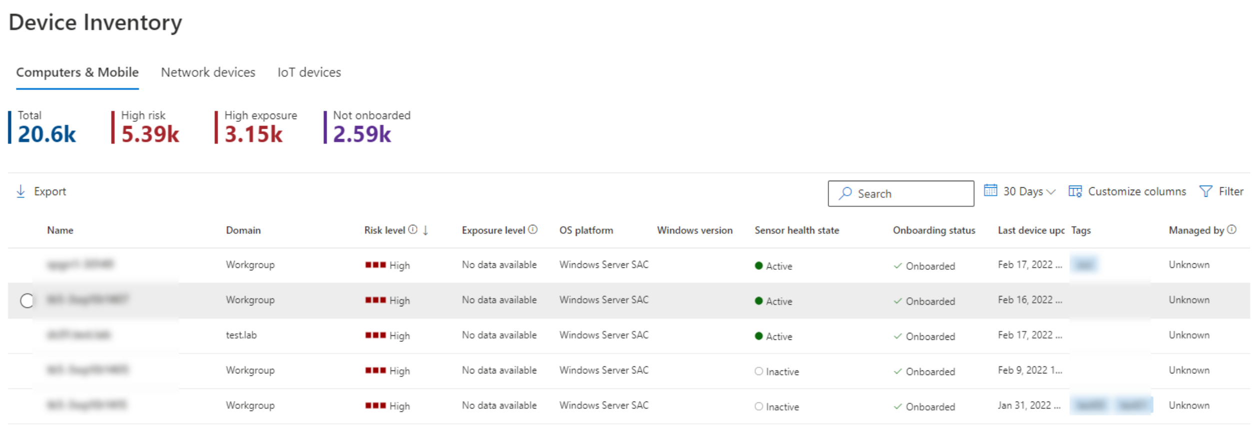
What's Next?

So, I don't really want to leave you all on an empty note here, I really like the idea of asset intelligence. I wanted to make this post about 2-3x the current size, but it's going to be **a lot** as is. There's one other trick that I think is interesting that you can do that'll help Cyber Assurance, Endpoint Security and Red Teams.

Since AD is the target of choice, and all centrally managed assets *should* be joined to the domain, it's a pretty authoritative list. So, SharpHound collects the data, and we're reaching out to another data source to add additional context (This being our EDR). We'll be adding EDR data into Neo4j/Bloodhound. We're going to use this to create a new property on each Computer Node called "hasEDR".

This presents an interesting opportunity to show where there is EDR visibility gaps in the network. This is an awesome place for an attackers to live in, but also an awesome thing that you can use to visualize how big of a problem you may have (i.e. hosts w/o EDR). If you collect IP addressing info and connect IP nodes into the list, you may be able to see if a whole network segment has communication issues, which could help determine root-cause analysis as to why your device hasn't registered or checked into your EDR platform.

Ok, analysis of thingz aside, let's use MDE as an example. This document page from Microsoft shows how you can go to your device inventory and export hosts.


Device Inventory

Computers & Mobile | Network devices | IoT devices

Total 20.6k | High risk 5.39k | High exposure 3.15k | Not onboarded 2.59k

Export | Search | 30 Days | Customize columns | Filter

Name	Domain	Risk level	Exposure level	OS platform	Windows version	Sensor health state	Onboarding status	Last device upc	Tags	Managed by
[redacted]	Workgroup	■■■ High	No data available	Windows Server SAC		● Active	✓ Onboarded	Feb 17, 2022 ...	[redacted]	Unknown
[redacted]	Workgroup	■■■ High	No data available	Windows Server SAC		● Active	✓ Onboarded	Feb 16, 2022 ...		Unknown
[redacted]	test.lab	■■■ High	No data available	Windows Server SAC		● Active	✓ Onboarded	Feb 17, 2022 ...		Unknown
[redacted]	Workgroup	■■■ High	No data available	Windows Server SAC		○ Inactive	✓ Onboarded	Feb 9, 2022 1...		Unknown
[redacted]	Workgroup	■■■ High	No data available	Windows Server SAC		○ Inactive	✓ Onboarded	Jan 31, 2022 ...	[redacted]	Unknown

Notice in the left side below the count, there is an export button. You'll need to determine what CSV field has the name (likely the first?). So, we'll go off of this. We're going to make a slight modification to our first script to create a new property.

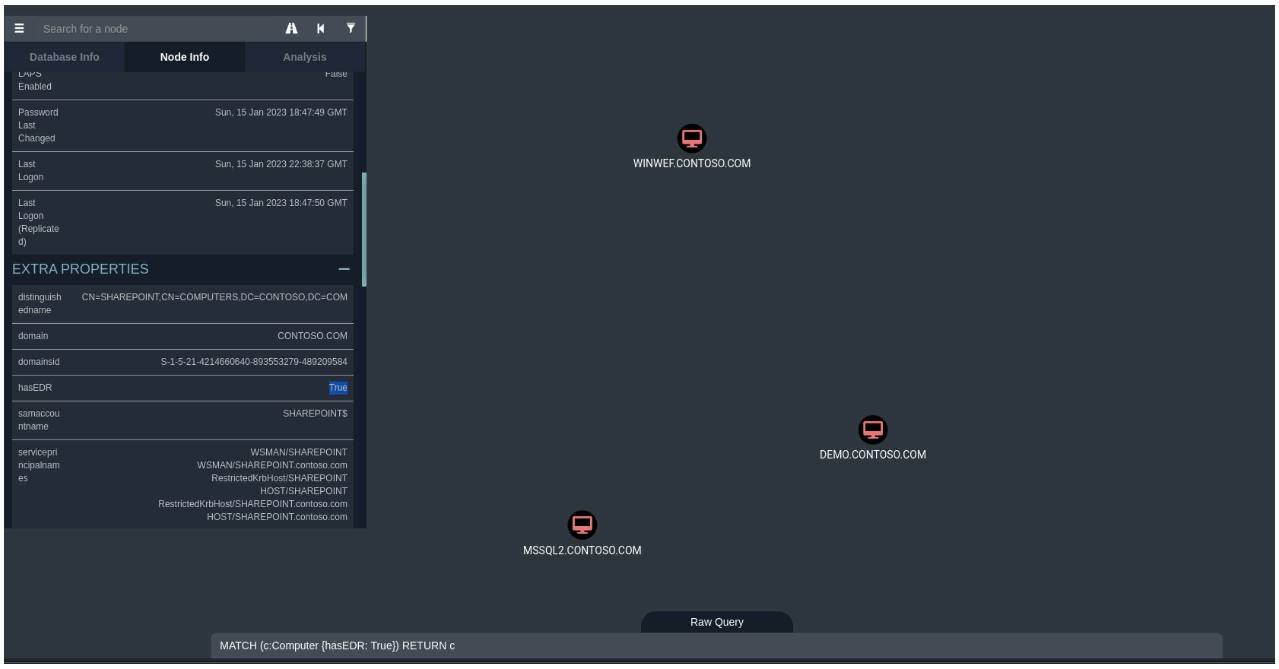
```
#!/usr/bin/python3
from neo4j import GraphDatabase
# Variables, I'm not going to implement arg parsing here because its a bit
overkill
neo4jUsername = "neo4j"
neo4jPassword = "bloodhound"
neo4jIP = "neo4j://localhost:7687"
csvFile = "file.csv"
f = open(csvFile, "r")

# Connecting to the Neo4j Database
try:
        neo4jdbDriver = GraphDatabase.driver(neo4jIP, auth=(neo4jUsername,
neo4jPassword))
except Exception as e:
        print("An error occured when connecting or logging into Neo4j: " + str(e))

# Looping throug the lines in the file
for LINE in f:
        #Stipping unnecessary characters out to ensure the N4j query doesn't fail
        n4jlist = LINE.split(",")
        computer = n4jlist[0].strip("\n")
        computer = computer.strip("\"")
        computer = computer.strip("'")
# Debug print statement
        print("Setting hasEDR for on " + computer)
        # Our query template
        queryTemplate = 'MATCH (c:Computer) SET (CASE WHEN c.name =~ "(?i)' +
computer + '.+" THEN c END).hasEDR = True'
        # Printing our query template
        print(queryTemplate)
        try:
                        # Using the Neo4j driver to execute our query against the
Neo4j database.
                neo4jdbDriver.execute_query(queryTemplate, database='neo4j')
        except Exception as e:
                print("An error occured while executing the query: " + str(e))
```

Note that Username was removed and Computer was adjusted to be the first field in the CSV instead of the second. The query template also changed to:

```
MATCH (c:Computer) SET (CASE WHEN c.name =~ "(?i)DC.+" THEN c END).hasEDR = True
```

You can try this in Neo4j/BloodHound if you'd like to see it in action. Else, continue on. Now, we'll run our script again. We can run the following query to make sure that it worked.

```
MATCH (c:Computer {hasEDR:True}) RETURN c
```



LAPS Enabled	False
Password Last Changed	Sun, 15 Jan 2023 18:47:49 GMT
Last Logon	Sun, 15 Jan 2023 22:38:37 GMT
Last Logon (Replicated)	Sun, 15 Jan 2023 18:47:50 GMT

EXTRA PROPERTIES

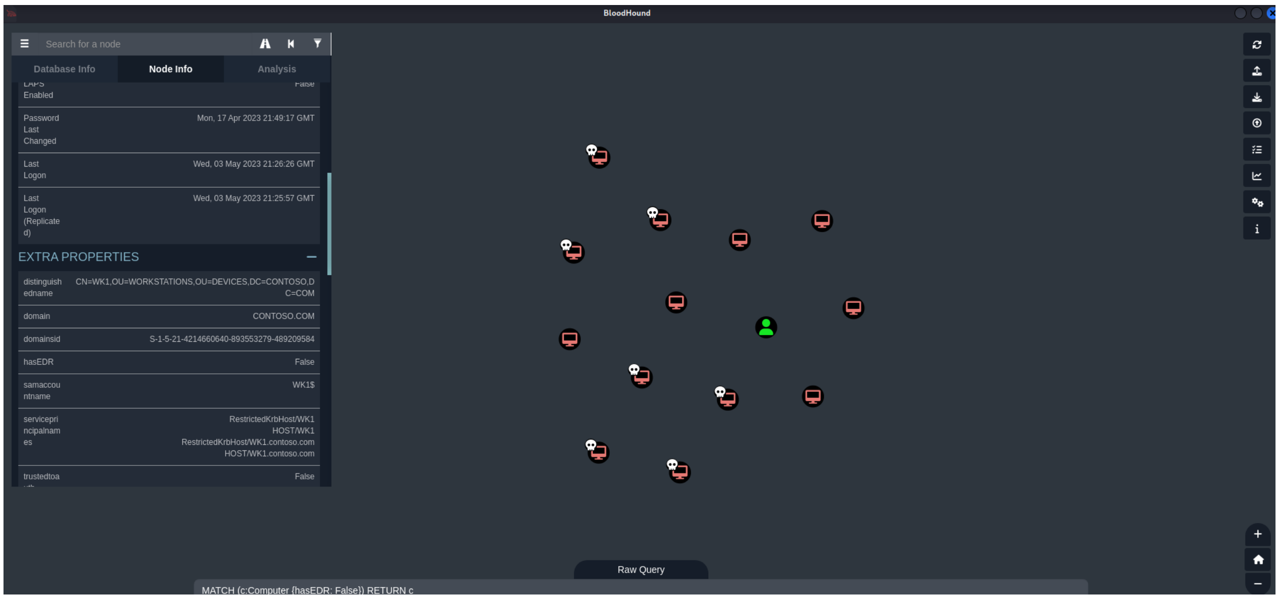
distinguishedname	CN=SHAREPOINT,CN=COMPUTERS,DC=CONTOSO,DC=COM
domain	CONTOSO.COM
domainsid	S-1-5-21-4214660640-893553279-489209584
hasEDR	True
samaccountname	SHAREPOINT$
serviceprincipalnames	WSMAN/SHAREPOINT WSMAN/SHAREPOINT.contoso.com RestrictedKrbHost/SHAREPOINT HOST/SHAREPOINT RestrictedKrbHost/SHAREPOINT.contoso.com HOST/SHAREPOINT.contoso.com

```
MATCH (c:Computer {hasEDR: True}) RETURN c
```

In BloodHound, we can see there is a new property called “hasEDR” and it’s set to True. If you accidentally mess up the query by leaving some extra whitespace at the end of the file, you can run something like:

```
MATCH (c:Computer) SET (CASE WHEN c.name =~ "(?i).+" THEN c END).hasEDR = False
```

to revert the results back to a clean slate. If you want to view the inverse, you can modify the query to `hasEDR: False` instead. Fairly straight forward!



LAPS Enabled	False
Password Last Changed	Mon, 17 Apr 2023 21:49:17 GMT
Last Logon	Wed, 03 May 2023 21:26:26 GMT
Last Logon (Replicated)	Wed, 03 May 2023 21:25:57 GMT

EXTRA PROPERTIES

distinguishedname	CN=WK1,OU=WORKSTATIONS,OU=DEVICES,DC=CONTOSO,DC=COM
domain	CONTOSO.COM
domainsid	S-1-5-21-4214660640-893553279-489209584
hasEDR	False
samaccountname	WK1$
serviceprincipalnames	RestrictedKrbHost/WK1 HOST/WK1 RestrictedKrbHost/WK1.contoso.com HOST/WK1.contoso.com
trustedtoauth	False

```
MATCH (c:Computer {hasEDR: False}) RETURN c
```

It looks like someone might have some cleaning up to do! There are other tools that one could use to query this data without your EDR platform (ex. Using the IPC$ share to see if your EDR process has an open pipe). You could use a project like serviceDetector to build a CSV if you don’t have access to that kind of data.

Anyways - I apologize for the over 4,000 word blog post. I really only meant for it to be about half that. I threw this together mostly last night and after work today. I wanted to show off something a little bit different. I’m hoping to put some more instances together like this soon - I’ve really got to rebuild my homelab so I can get some proper infrastructure setup. I hope this helps and you learned something about enhancing, extending and enriching Neo4j/BloodHound’s capabilities. I had a lot of fun researching this and wanted to throw some ideas out there for others.

See you all at DEFCON <3

Comments