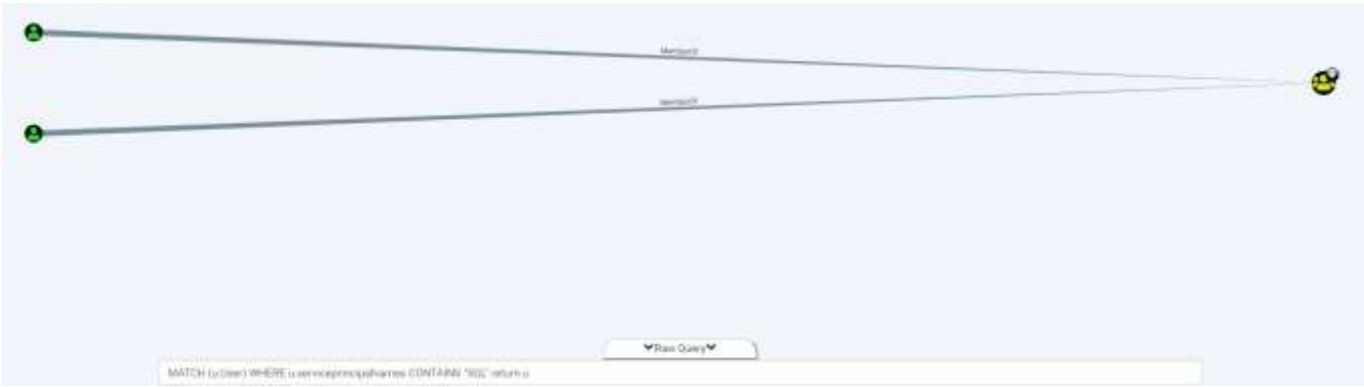
BLOODHOUND CYPHER CHEATSHEET

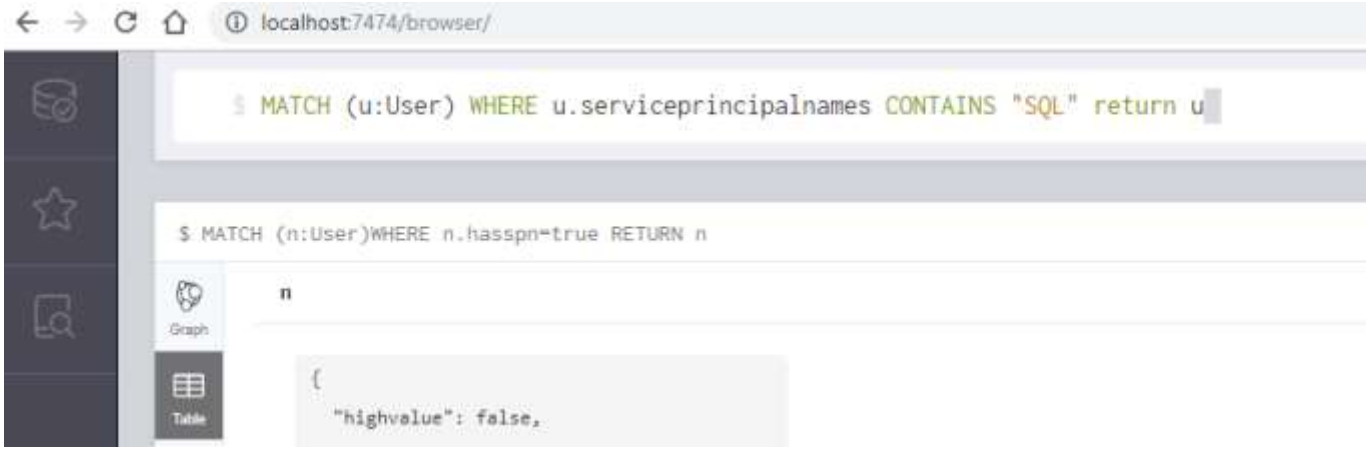
September 9, 2019 Hausec Infosec 6 comments

Bloodhound uses Neo4j, a graphing database, which uses the Cypher language. Cypher is a bit complex since it's almost like programming with ASCII art. This cheatsheet aims to cover some Cypher queries that can easily be pasted into Bloodhound GUI and or Neo4j Console to leverage more than the default queries. This cheatsheet is separated by whether the query is for the GUI

or console. For the console, it means they cannot be executed via Bloodhound GUI and must be done via the neo4j console.



Executing via GUI



Executing via neo4j console

To also make life easier, I've taken the applicable queries here and made them compatible within the "Custom Queries" section in the GUI. You can download that here: https://github.com/hausec/Bloodhound-Custom-Queries

Table of Contents

GUI/Graph Queries

Find All edges any owned user has on a computer

```
MATCH p=shortestPath((m:User)-[r]->(b:Computer)) WHERE m.owned RETURN p
```

Find All Users with an SPN/Find all Kerberoastable Users

```
MATCH (n:User)WHERE n.hasspn=true
RETURN n
```

Find All Users with an SPN/Find all Kerberoastable Users with passwords last set > 5 years ago

```
MATCH (u:User) WHERE u.hasspn=true AND u.pwdlastset < (datetime().epochseconds - (1825 * 86400)) AND
RETURN u.name, u.pwdlastset order by u.pwdlastset
```

Find SPNs with keywords (swap SQL with whatever)

```
MATCH (u:User) WHERE ANY (x IN u.serviceprincipalnames WHERE toUpper(x) CONTAINS 'SQL')RETURN u
```

Kerberoastable Users with a path to DA

```
MATCH (u:User {hasspn:true}) MATCH (g:Group) WHERE g.name CONTAINS 'DOMAIN ADMINS' MATCH p = s
```

Find workstations a user can RDP into.

```
match p=(g:Group)-[:CanRDP]->(c:Computer) where g.objectid ENDS WITH '-513'  AND NOT c.operatingsyste
```

Find servers a user can RDP into.

```
match p=(g:Group)-[:CanRDP]->(c:Computer) where  g.objectid ENDS WITH '-513'  AND c.operatingsystem C
```

DA sessions not on a certain group (e.g. domain controllers)

```
OPTIONAL MATCH (c:Computer)-[:MemberOf]->(t:Group) WHERE NOT t.name = 'DOMAIN CONTROLLERS@T
```

Find all computers with Unconstrained Delegation

```
MATCH (c:Computer {unconstraineddelegation:true}) return c
```

Find unsupported OSs

```
MATCH (H:Computer) WHERE H.operatingsystem =~ '.*(2000|2003|2008|xp|vista|7|me)*.' RETURN H
```

Find users that logged in within the last 90 days. Change 90 to whatever threshold you want.

```
MATCH (u:User) WHERE u.lastlogon < (datetime().epochseconds - (90 * 86400)) and NOT u.lastlogon IN [-1.0
```

Find users with passwords last set thin the last 90 days. Change 90 to whatever threshold you want.

```
MATCH (u:User) WHERE u.pwdlastset < (datetime().epochseconds - (90 * 86400)) and NOT u.pwdlastset IN [-
```

Find all sessions any user in a specific domain has

```
MATCH p=(m:Computer)-[r:HasSession]->(n:User {domain: "TEST.LOCAL"}) RETURN p
```

View all GPOs

```
Match (n:GPO) return n
```

View all GPOs that contain a keyword

```
Match (n:GPO) WHERE n.name CONTAINS "SERVER" return n
```

View all groups that contain the word ‘admin’

```
Match (n:Group) WHERE n.name CONTAINS "ADMIN" return n
```

Find user that doesn’t require kerberos pre-authentication (aka AS-REP Roasting)

```
MATCH (u:User {dontreqpreauth: true}) RETURN u
```

Find a group with keywords. E.g. SQL ADMINS or SQL 2017 ADMINS

```
MATCH (g:Group) WHERE g.name =~ '(?i).SQL.ADMIN.*' RETURN g
```

Show all high value target group

```
MATCH p=(n:User)-[r:MemberOf*1..]->(m:Group {highvalue:true}) RETURN p
```

Shortest paths to Domain Admins group from computers:

```
MATCH (n:Computer),(m:Group {name:'DOMAIN ADMINS@DOMAIN.GR'}),p=shortestPath((n)-[r:MemberOf
```

Shortest paths to Domain Admins group from computers excluding potential DCs (based on ldap/ and GC/ spns):

```
WITH '(?i)ldap/.*' as regex_one WITH '(?i)gc/.*' as regex_two MATCH (n:Computer) WHERE NOT ANY(item IN
```

Shortest paths to Domain Admins group from all domain groups (fix-it):

```
MATCH (n:Group),(m:Group {name:'DOMAIN ADMINS@DOMAIN.GR'}),p=shortestPath((n)-[r:MemberOf|Has
```

Shortest paths to Domain Admins group from non-privileged groups (AdminCount=false)

```
MATCH (n:Group {admincount:false}),(m:Group {name:'DOMAIN ADMINS@DOMAIN.GR'}),p=shortestPath((r
```

Shortest paths to Domain Admins group from the Domain Users group:

```
MATCH (g:Group) WHERE g.name =~ 'DOMAIN USERS@.*' MATCH (g1:Group) WHERE g1.name =~ 'DOMAIN
```

Find interesting privileges/ACEs that have been configured to DOMAIN USERS group:

```
MATCH (m:Group) WHERE m.name =~ 'DOMAIN USERS@.*' MATCH p=(m)-[r:Owns|:WriteDacl|:GenericAll|:
```

Shortest paths to Domain Admins group from non privileged users (AdminCount=false):

```
MATCH (n:User {admincount:false}),(m:Group {name:'DOMAIN ADMINS@DOMAIN.GR'}),p=shortestPath((n)-
```

Find all Edges that a specific user has against all the nodes (HasSession is not calculated, as it is an edge that comes from computer to user, not from user to computer):

```
MATCH (n:User) WHERE n.name =~ 'HELPDESK@DOMAIN.GR'MATCH (m) WHERE NOT m.name = n.name MA
```

Find all the Edges that any UNPRIVILEGED user (based on the admincount:False) has against all the nodes:

```
MATCH (n:User {admincount:False}) MATCH (m) WHERE NOT m.name = n.name MATCH p=allShortestPaths(
```

Find interesting edges related to "ACL Abuse" that uprivileged users have against other users:

```
MATCH (n:User {admincount:False}) MATCH (m:User) WHERE NOT m.name = n.name MATCH p=allShortestP
```

Find interesting edges related to "ACL Abuse" that unprivileged users have against computers:

```
MATCH (n:User {admincount:False}) MATCH p=allShortestPaths((n)-[r:AllExtendedRights|GenericAll|Generi
```

Find if unprivileged users have rights to add members into groups:

```
MATCH (n:User {admincount:False}) MATCH p=allShortestPaths((n)-[r:AddMember*1..]->(m:Group)) RETURN
```

Find the active user sessions on all domain computers:

```
MATCH p1=shortestPath(((u1:User)-[r1:MemberOf*1..]->(g1:Group))) MATCH p2=(c:Computer)-[*1]->(u1) RE
```

Find all the privileges (edges) of the domain users against the domain computers (e.g. CanRDP, AdminTo etc. HasSession edge is not included):

```
MATCH p1=shortestPath(((u1:User)-[r1:MemberOf*1..]->(g1:Group))) MATCH p2=(u1)-[*1]->(c:Computer) RE
```

Find only the AdminTo privileges (edges) of the domain users against the domain computers:

```
MATCH p1=shortestPath(((u1:User)-[r1:MemberOf*1..]->(g1:Group))) MATCH p2=(u1)-[:AdminTo*1..]->(c:Cor
```

Find only the CanRDP privileges (edges) of the domain users against the domain computers:

```
MATCH p1=shortestPath(((u1:User)-[r1:MemberOf*1..]->(g1:Group))) MATCH p2=(u1)-[:CanRDP*1..]->(c:Com
```

Display in BH a specific user with constrained deleg and his targets where he allowed to delegate:

```
MATCH (u:User {name:'USER@DOMAIN.GR'}),(c:Computer),p=((u)-[r:AllowedToDelegate]->(c)) RETURN p
```

Console Queries

Find what groups can RDP

```
MATCH p=(m:Group)-[r:CanRDP]->(n:Computer) RETURN m.name, n.name ORDER BY m.name
```

Find what groups can reset passwords

```
MATCH p=(m:Group)-[r:ForceChangePassword]->(n:User) RETURN m.name, n.name ORDER BY m.name
```

Find what groups have local admin rights

```
MATCH p=(m:Group)-[r:AdminTo]->(n:Computer) RETURN m.name, n.name ORDER BY m.name
```

Find what users have local admin rights

```
MATCH p=(m:User)-[r:AdminTo]->(n:Computer) RETURN m.name, n.name ORDER BY m.name
```

List the groups of all owned users

```
MATCH (m:User) WHERE m.owned=TRUE WITH m MATCH p=(m)-[:MemberOf*1..]->(n:Group) RETURN m.na
```

List the unique groups of all owned users

```
MATCH (m:User) WHERE m.owned=TRUE WITH m MATCH (m)-[r:MemberOf*1..]->(n:Group) RETURN DISTIN
```

All active DA sessions

```
MATCH (n:User)-[:MemberOf*1..]->(g:Group) WHERE g.objectid ENDS WITH '-512' MATCH p = (c:Computer)-[
```

Find all active sessions a member of a group has

```
MATCH (n:User)-[:MemberOf*1..]->(g:Group {name:'DOMAIN ADMINS@TESTLAB.LOCAL'}) MATCH p = (c:Cor
```

Can an object from domain ‘A’ do anything to an object in domain ‘B’

```
MATCH (n {domain:"TEST.LOCAL"})-[r]->(m {domain:"LAB.LOCAL"}) RETURN LABELS(n)[0],n.name,TYPE(r),LA
```

Find all connections to a different domain/forest

```
MATCH (n)-[r]->(m) WHERE NOT n.domain = m.domain RETURN LABELS(n)[0],n.name,TYPE(r),LABELS(m)[0],r
```

Find All Users with an SPN/Find all Kerberoastable Users with passwords last set > 5 years ago (In Console)

```
MATCH (u:User) WHERE n.hasspn=true AND WHERE u.pwdlastset < (datetime().epochseconds - (1825 * 864(
```

Kerberoastable Users with most privileges

```
MATCH (u:User {hasspn:true}) OPTIONAL MATCH (u)-[:AdminTo]->(c1:Computer) OPTIONAL MATCH (u)-[:Me
```

Find users that logged in within the last 90 days. Change 90 to whatever threshold you want. (In Console)

```
MATCH (u:User) WHERE u.lastlogon < (datetime().epochseconds - (90 * 86400)) and NOT u.lastlogon IN [-1.0
```

Find users with passwords last set within the last 90 days. Change 90 to whatever threshold you want. (In Console)

```
MATCH (u:User) WHERE u.pwdlastset < (datetime().epochseconds - (90 * 86400)) and NOT u.pwdlastset IN [-
```

List users and their login times + pwd last set times in human readable format

```
MATCH (n:User) WHERE n.enabled = TRUE RETURN n.name, datetime({epochSeconds: toInteger(n.pwdlasts
```

Find constrained delegation (In Console)

```
MATCH (u:User)-[:AllowedToDelegate]->(c:Computer) RETURN u.name,COUNT(c) ORDER BY COUNT(c) DESC
```

View OUs based on member count. (In Console)

```
MATCH (o:OU)-[:Contains]->(c:Computer) RETURN o.name,o.guid,COUNT(c) ORDER BY COUNT(c) DESC
```

Return each OU that has a Windows Server in it (In Console)

```
MATCH (o:OU)-[:Contains]->(c:Computer) WHERE toUpper(c.operatingsystem) STARTS WITH "WINDOWS SER
```

Find computers that allow unconstrained delegation that AREN'T domain controllers. (In Console)

```
MATCH (c1:Computer)-[:MemberOf*1..]->(g:Group) WHERE g.objectsid ENDS WITH '-516' WITH COLLECT(c1.
```

Find the number of principals with control of a "high value" asset where the principal itself does not belong to a "high value" group

```
MATCH (n {highvalue:true}) OPTIONAL MATCH (m1)-[{isacl:true}]->(n) WHERE NOT (m1)-[:MemberOf*1..]->(:
```

Enumerate all properties (In Console)

```
Match (n:Computer) return properties(n)
```

Match users that are not AdminCount 1, have generic all, and no local admin

```
MATCH (u:User)-[:GenericAll]->(c:Computer) WHERE  NOT u.admincount AND NOT (u)-[:AdminTo]->(c) RETU
```

What permissions does Everyone/Authenticated users/Domain users/Domain computers have

```
MATCH p=(m:Group)-[r:AddMember|AdminTo|AllExtendedRights|AllowedToDelegate|CanRDP|Contains|
```

Find computers with descriptions and display them (along with the description, sometimes admins save sensitive data on domain objects descriptions like passwords):

```
MATCH (c:Computer) WHERE c.description IS NOT NULL RETURN c.name,c.description
```

Return the name of every computer in the database where at least one SPN for the computer contains the string “MSSQL”:

```
MATCH (c:Computer) WHERE ANY (x IN c.serviceprincipalnames WHERE toUpper(x) CONTAINS 'MSSQL') RET
```

Find any computer that is NOT a domain controller and it is trusted to perform unconstrained delegation:

```
MATCH (c1:Computer)-[:MemberOf*1..]->(g:Group) WHERE g.objectid ENDS WITH '-516' WITH COLLECT(c1.n
```

Find every computer account that has local admin rights on other computers. Return in descending order of the number of computers the computer account has local admin rights to:

```
MATCH (c1:Computer) OPTIONAL MATCH (c1)-[:AdminTo]->(c2:Computer) OPTIONAL MATCH (c1)-[:Member
```

Alternatively, find every computer that has local admin rights on other computers and display these computers:

```
MATCH (c1:Computer) OPTIONAL MATCH (c1)-[:AdminTo]->(c2:Computer) OPTIONAL MATCH (c1)-[:Member
```

Get the names of the computers without admins, sorted by alphabetic order:

```
MATCH (n)-[r:AdminTo]->(c:Computer) WITH COLLECT(c.name) as compsWithAdmins MATCH (c2:Computer)
```

Show computers (excluding Domain Controllers) where Domain Admins are logged in:

```
MATCH (n:User)-[:MemberOf*1..]->(g:Group {name:'DOMAIN ADMINS@DOMAIN.GR'}) WITH n as privusers
```

Find the percentage of computers with path to Domain Admins:

```
MATCH (totalComputers:Computer {domain:'DOMAIN.GR'}) MATCH p=shortestPath((ComputersWithPath:C
```

Find on each computer who can RDP (searching only enabled users):

```
MATCH (c:Computer) OPTIONAL MATCH (u:User)-[:CanRDP]->(c) WHERE u.enabled=true OPTIONAL MATCH
```

Find on each computer the number of users with admin rights (local admins) and display the users with admin rights:

```
MATCH (c:Computer) OPTIONAL MATCH (u1:User)-[:AdminTo]->(c) OPTIONAL MATCH (u2:User)-[:MemberOf
```

Active Directory group with default privileged rights on domain users and groups, plus the ability to logon to Domain Controllers

```
MATCH (u:User)-[r1:MemberOf*1..]->(g1:Group {name:'ACCOUNT OPERATORS@DOMAIN.GR'}) RETURN u.n
```

Find which domain Groups are Admins to what computers:

```
MATCH (g:Group) OPTIONAL MATCH (g)-[:AdminTo]->(c1:Computer) OPTIONAL MATCH (g)-[:MemberOf*1..]-
```

Find which domain Groups (excluding the Domain Admins and Enterprise Admins) are Admins to what computers:

```
MATCH (g:Group) WHERE NOT (g.name =~ '(?i)domain admins@.*' OR g.name =~ "(?i)enterprise admins@.*"
```

Find which domain Groups (excluding the high privileged groups marked with AdminCount=true) are Admins to what computers:

```
MATCH (g:Group) WHERE g.admincount=false OPTIONAL MATCH (g)-[:AdminTo]->(c1:Computer) OPTIONAL
```

Find the most privileged groups on the domain (groups that are Admins to Computers. Nested groups will be calculated):

```
MATCH (g:Group) OPTIONAL MATCH (g)-[:AdminTo]->(c1:Computer) OPTIONAL MATCH (g)-[:MemberOf*1..]-
```

Find the number of computers that do not have local Admins:

```
MATCH (n)-[r:AdminTo]->(c:Computer) WITH COLLECT(c.name) as compsWithAdmins MATCH (c2:Computer)
```

Find groups with most local admins (either explicit admins or derivative/unrolled):

```
MATCH (g:Group) WITH g OPTIONAL MATCH (g)-[r:AdminTo]->(c1:Computer) WITH g,COUNT(c1) as explicitA
```

Find percentage of non-privileged groups (based on admincount:false) to Domain Admins group:

```
MATCH (totalGroups:Group {admincount:false}) MATCH p=shortestPath((GroupsWithPath:Group {admincou
```

Find every user object where the “userpassword” attribute is populated (wald0):

```
MATCH (u:User) WHERE NOT u.userpassword IS null RETURN u.name,u.userpassword
```

Find every user that doesn’t require kerberos pre-authentication (wald0):

```
MATCH (u:User {dontreqpreauth: true}) RETURN u.name
```

Find all users trusted to perform constrained delegation. The result is ordered by the amount of computers:

```
MATCH (u:User)-[:AllowedToDelegate]->(c:Computer) RETURN u.name,COUNT(c) ORDER BY COUNT(c) DESC
```

Find the active sessions that a specific domain user has on all domain computers:

```
MATCH p1=shortestPath(((u1:User {name:'USER@DOMAIN.GR'})-[r1:MemberOf*1..]->(g1:Group))) MATCH (c
```

Count the number of the computers where each domain user has direct Admin privileges to:

```
MATCH (u:User)-[:AdminTo]->(c:Computer) RETURN count(DISTINCT(c.name)) AS COMPUTER, u.name AS US
```

Count the number of the computers where each domain user has derivative Admin privileges to:

```
MATCH (u:User)-[:MemberOf*1..]->(:Group)-[:AdminTo]->(c:Computer) RETURN count(DISTINCT(c.name)) AS
```

Display the computer names where each domain user has derivative Admin privileges to:

```
MATCH (u:User)-[:MemberOf*1..]->(:Group)-[:AdminTo]->(c:Computer) RETURN DISTINCT(c.name) AS COMP
```

Find Kerberoastable users who are members of high value groups:

```
MATCH (u:User)-[r:MemberOf*1..]->(g:Group) WHERE g.highvalue=true AND u.hasspn=true RETURN u.name
```

Find Kerberoastable users and where they are AdminTo:

```
OPTIONAL MATCH (u1:User) WHERE u1.hasspn=true OPTIONAL MATCH (u1)-[r:AdminTo]->(c:Computer) RET
```

Find the percentage of users with a path to Domain Admins:

```
MATCH (totalUsers:User {domain:'DOMAIN.GR'}) MATCH p=shortestPath((UsersWithPath:User {domain:'DO
```

Find the percentage of enabled users that have a path to high value groups:

```
MATCH (u:User {domain:'DOMAIN.GR',enabled:True}) MATCH (g:Group {domain:'DOMAIN.GR'}) WHERE g.hi
```

List of unique users with a path to a Group tagged as “highvalue”:

```
MATCH (u:User) MATCH (g:Group {highvalue:true}) MATCH p = shortestPath((u:User)-[r:AddMember|Admin
```

Find users who are NOT marked as “Sensitive and Cannot Be Delegated” and have Administrative access to a computer, and where those users have sessions on servers with Unconstrained Delegation enabled (by NotMedic):

```
MATCH (u:User {sensitive:false})-[:MemberOf*1..]->(:Group)-[:AdminTo]->(c1:Computer) WITH u,c1 MATCH (
```

Find users with constrained delegation permissions and the corresponding targets where they allowed to delegate:

```
MATCH (u:User) WHERE u.allowedtodelegate IS NOT NULL RETURN u.name,u.allowedtodelegate
```

Alternatively, search for users with constrained delegation permissions,the corresponding targets where they are allowed to delegate, the privileged users that can be impersonated (based on sensitive:false and admincount:true) and find where these users (with constrained deleg privs) have active sessions (user hunting) as well as count the shortest paths to them:

```
OPTIONAL MATCH (u:User {sensitive:false, admincount:true}) WITH u.name AS POSSIBLE_TARGETS OPTION
```

Find computers with constrained delegation permissions and the corresponding targets where they allowed to delegate:

```
MATCH (c:Computer) WHERE c.allowedtodelegate IS NOT NULL RETURN c.name,c.allowedtodelegate
```

Alternatively, search for computers with constrained delegation permissions, the corresponding targets where they are allowed to delegate, the privileged users that can be impersonated (based on sensitive:false and admincount:true) and find who is LocalAdmin on these computers as well as count the shortest paths to them:

```
OPTIONAL MATCH (u:User {sensitive:false, admincount:true}) WITH u.name AS POSSIBLE_TARGETS OPTION
```

Find if any domain user has interesting permissions against a GPO:

```
MATCH p=(u:User)-[r:AllExtendedRights|GenericAll|GenericWrite|Owns|WriteDacl|WriteOwner|GpLink*1.
```

Show me all the sessions from the users in the OU with the following GUID

```
MATCH p=(o:OU {guid:'045939B4-3FA8-4735-YU15-7D61CFOU6500'})-[r:Contains*1..]->(u:User) MATCH (c:C
```

Creating a property on the users that have an actual path to anything high_value(heavy query. takes hours on a large dataset)

```
MATCH (u:User) MATCH (g:Group {highvalue: true}) MATCH p = shortestPath((u:User)-[r:AddMember|Admir
```

What “user with a path to any high_value group” has most sessions?

```
MATCH (c:Computer)-[rel:HasSession]->(u:User {has_path_to_da: true}) WITH COLLECT(c) as tempVar,u UNW
```

Creating a property for Tier 1 Users with regex:

```
MATCH (u:User) WHERE u.name =~ 'internal_naming_convention[0-9]{2,5}@EXAMPLE.LOCAL' SET u.tier_1_u
```

Creating a property for Tier 1 Computers via group-membership:

```
MATCH (c:Computer)-[r:MemberOf*1..]-(g:Group {name:'ALL_SERVERS@EXAMPLE.LOCAL'}) SET c.tier_1_com
```

Creating a property for Tier 2 Users via group name and an exclusion:

```
MATCH (u:User)-[r:MemberOf*1..]-(g:Group)WHERE g.name CONTAINS  'ALL EMPLOYEES' AND NOT u.name
```

Creating a property for Tier 2 Computers via nested groups name:

```
MATCH (c:Computer)-[r:MemberOf*1..]-(g:Group) WHERE g.name STARTS WITH 'CLIENT_' SET c.tier_2_comp
```

List Tier2 access to Tier 1 Computers

```
MATCH (u)-[rel:AddMember|AdminTo|AllowedToDelegate|CanRDP|ExecuteDCOM|ForceChangePassword
```

List Tier 1 Sessions on Tier 2 Computers

```
MATCH (c:Computer)-[rel:HasSession]->(u:User) WHERE u.tier_1_user = true AND c.tier_2_computer = true R
```

List all users with local admin and count how many instances

```
OPTIONAL MATCH (c1)-[:AdminTo]->(c2:Computer) OPTIONAL MATCH (c1)-[:MemberOf*1..]->(:Group)-[:Adm
```

Find all users a part of the VPN group

```
Match (u:User)-[:MemberOf]->(g:Group) WHERE g.name CONTAINS "VPN" return u.name,g.name
```

Find users that have never logged on and account is still active

```
MATCH (n:User) WHERE n.lastlogontimestamp=-1.0 AND n.enabled=TRUE RETURN n.name ORDER BY n.nan
```

Adjust Query to Local Timezone (Change timezone parameter)

```
MATCH (u:User) WHERE NOT u.lastlogon IN [-1.0, 0.0] return u.name, datetime({epochSeconds:toInteger(u.l
```

Thank you to the following contributors:

@sbooker_, @_wald0, @cptjesus, @ScoubiMtl, @sysop_host, @haus3c, @bravo2day, @rvrsh3ll, @Krelkci

Share this:

Twitter Facebook

Like

One blogger likes this.

« Offensive Lateral Movement

Attacking Azure, Azure AD, and Introducing PowerZure »

6 COMMENTS

John Smith says:

December 11, 2019 at 5:22 pm

Thank you….that worked

Anonymous says:

December 11, 2019 at 5:21 pm

Thank you…that worked..did have to replace the “-” with mine….thanks again

Hausec says:

December 10, 2019 at 4:37 pm

Oh you’re right, sorry. That query looks like an older one that I had, try this

> MATCH (u:User) WHERE u.hasspn=true AND u.pwdlastset < (datetime().epochseconds – (1825 * 86400)) AND NOT u.pwdlastset IN [-1.0, 0.0] RETURN u.name, u.pwdlastset order by u.pwdlastset

If you still get an error with that, replace the "-" after epochseconds with your own. This query is very fickle.

John Smith says:

December 10, 2019 at 1:31 am

Thanks for the reply...for this query I don't see any quotation marks that could be giving me the error. Here's the query again:

MATCH (u:User) WHERE n.hasspn=true AND WHERE u.pwdlastset < (datetime().epochseconds – (1825 * 86400)) and NOT u.pwdlastset IN [-1.0, 0.0] RETURN u.name, u.pwdlastset order by u.pwdlastset

---

Hausec says:

December 9, 2019 at 2:02 pm

I just ran the query again and it worked for me, but I think it's because you have a different keyboard and using a different type of ASCII quote than what I do. Try replacing all the quotation

marks with yours (double/single doesn't matter) and it should work. There's a difference in UK and US keyboard that gives that error sometimes.

John Smith says:

December 9, 2019 at 12:07 am

Hello…first of all thank you for this great blog…learning a lot from your posts. I tried one of the cypher query above in the console for "Find All Users with an SPN/Find all Kerberoastable Users with passwords last set > 5 years ago (In Console)". I get the below error after I cut and paste the exact query as stated above:

Neo.ClientError.Statement.SyntaxError
Invalid input '.': expected 'n/N' (line 1, column 47 (offset: 46))
"MATCH (u:User) WHERE n.hasspn=true AND WHERE u.pwdlastset < (datetime().epochseconds – (1825 * 86400)) and NOT u.pwdlastset IN [-1.0, 0.0] RETURN u.name, u.pwdlastset order by u.pwdlastset"

Curious if it is happening for just me or something wrong with the query. Thank you

LEAVE A REPLY

Enter your comment here...