ShadowHound: A SharpHound Alternative Using Native PowerShell

blog.fndsec.net/2024/11/25/shadowhound

November 25, 2024

*Written by **Yehuda Smirnov***

TL;DR

- ShadowHound is a **PowerShell tool** designed to map Active Directory environments for BloodHound without introducing known-malicious / foreign binaries like SharpHound or SoapHound.
- Two methods (protocols) can be used for data collection:
  - Using the **Active Directory module (ADWS)**
  - Using the native **DirectorySearcher class** (**LDAP**).
- Bonus – Avoiding Defender for Identity detections (and how to still detect such enumerations)

Find the ShadowHound GitHub repo here.

Introduction

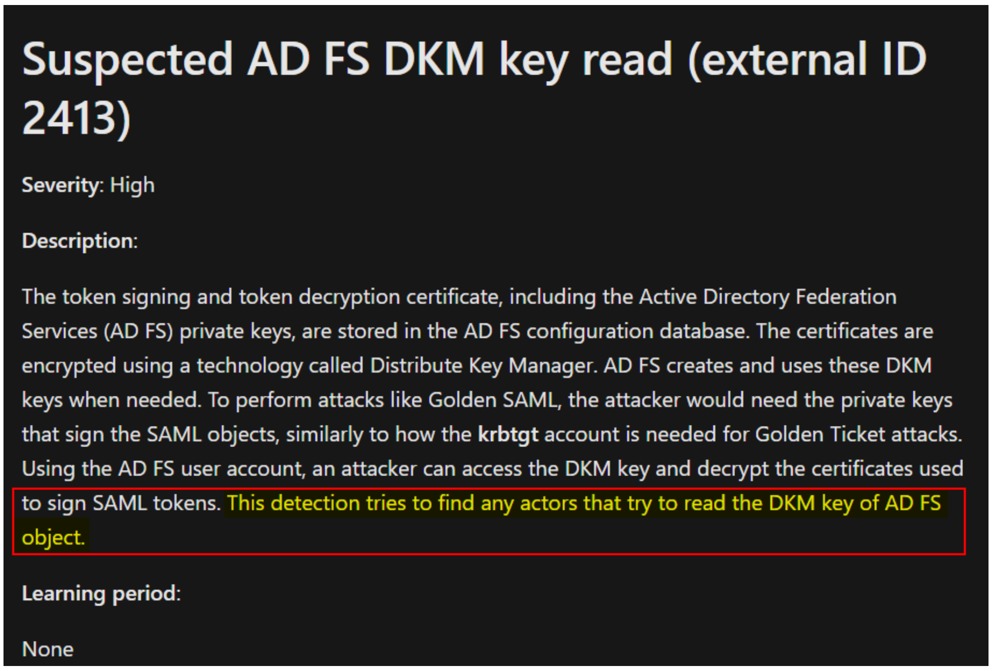
While SharpHound has been the cornerstone for collecting data for BloodHound, deploying it in a target environment poses detection risks (even after some obfuscation and customization). EDRs are getting increasingly better at detecting and flagging such binaries, whether they are reflectively loaded, etc. **ShadowHound aims to avoid this issue by utilizing native PowerShell or legitimate tools (AD Module) to obtain the BloodHound Data.**

It should be noted that **Domain Controllers can still flag unusual LDAP queries** (with the help of Defender for Identity for example). However, by avoiding the introduction of known-malicious binaries, we can minimize our footprint on the target machine and avoid some of the common LDAP filters which trigger alerts (more on this later).

ADExplorer Snapshot Detections (sort of)

One alternative to SharpHound is using ADExplorer to create snapshots of the Active Directory and then convert those snapshots into BloodHound JSON files using tools like ADExplorerSnapshot.py. While this method can be effective, it might fail in large domains when dealing with poor connectivity (or the VPN auto disconnects after a period of time). **In such cases, ADExplorer may error out due to connectivity issues**, and you will be left without any snapshot data.

During a recent engagement, we also encountered detections when using ADExplorer. Specifically, **when Active Directory Federation Services (ADFS) is deployed**, creating a snapshot with **ADExplorer can trigger an alert** because it reads the ADFS LDAP container, as noted in the detections section for Microsoft Defender for Identity:


Suspected AD FS DKM key read (external ID 2413)

Severity: High

Description:

The token signing and token decryption certificate, including the Active Directory Federation Services (AD FS) private keys, are stored in the AD FS configuration database. The certificates are encrypted using a technology called Distribute Key Manager. AD FS creates and uses these DKM keys when needed. To perform attacks like Golden SAML, the attacker would need the private keys that sign the SAML objects, similarly to how the **krbtgt** account is needed for Golden Ticket attacks. Using the AD FS user account, an attacker can access the DKM key and decrypt the certificates used to sign SAML tokens. This detection tries to find any actors that try to read the DKM key of AD FS object.

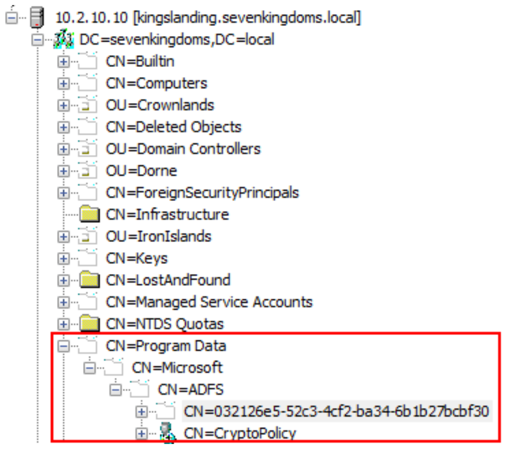
Learning period:

None

Defender for Identity's detection for reading ADFS containers

During a default install, the container exists under the `Program Data` container:

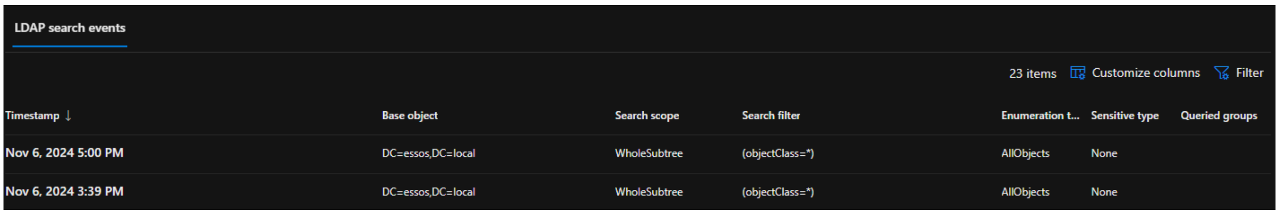
Faced with these challenges, we realized we needed a solution that could avoid both the detection issues and effectively handle large domains. This led us (Itay Yashar, my teammate, and I) to develop ShadowHound, turning an idea from our R&D shelf into a practical tool.


```
10.2.10.10 [kingslanding.sevenkingdoms.local]
  DC=sevenkingdoms,DC=local
    CN=Builtin
    CN=Computers
    OU=Crownlands
    CN=Deleted Objects
    OU=Domain Controllers
    OU=Dorne
    CN=ForeignSecurityPrincipals
    CN=Infrastructure
    OU=IronIslands
    CN=Keys
    CN=LostAndFound
    CN=Managed Service Accounts
    CN=NTDS Quotas
    CN=Program Data
      CN=Microsoft
        CN=ADFS
          CN=032126e5-52c3-4cf2-ba34-6b1b27bcbf30
          CN=CryptoPolicy
```

Picture of the ADFS container in a default installation

Avoiding Defender for Identity Detections

During our recent engagement and also during testing, we've noticed Defender for Identity **detections** occur for queries such as **`Get-ADUser -Filter *`** or when using `pyLdapSearch` with **`(objectClass=*)`** LDAP filter, which is widely known as the query performed by ADExplorer (and hence it is considered more 'opsec'):


LDAP search events

23 items | Customize columns | Filter

Timestamp ↓	Base object	Search scope	Search filter	Enumeration t...	Sensitive type	Queried groups
Nov 6, 2024 5:00 PM	DC=essos,DC=local	WholeSubtree	(objectClass=*)	AllObjects	None	
Nov 6, 2024 3:39 PM	DC=essos,DC=local	WholeSubtree	(objectClass=*)	AllObjects	None	

Defender for Identity detection for (objectClass=*)

Upon researching we've discovered that the query `(objectGuid=*)` **seemed to not raise any Defender for Identity detections** (for now), and hence why we've opted to use this LDAP filter for the tool.

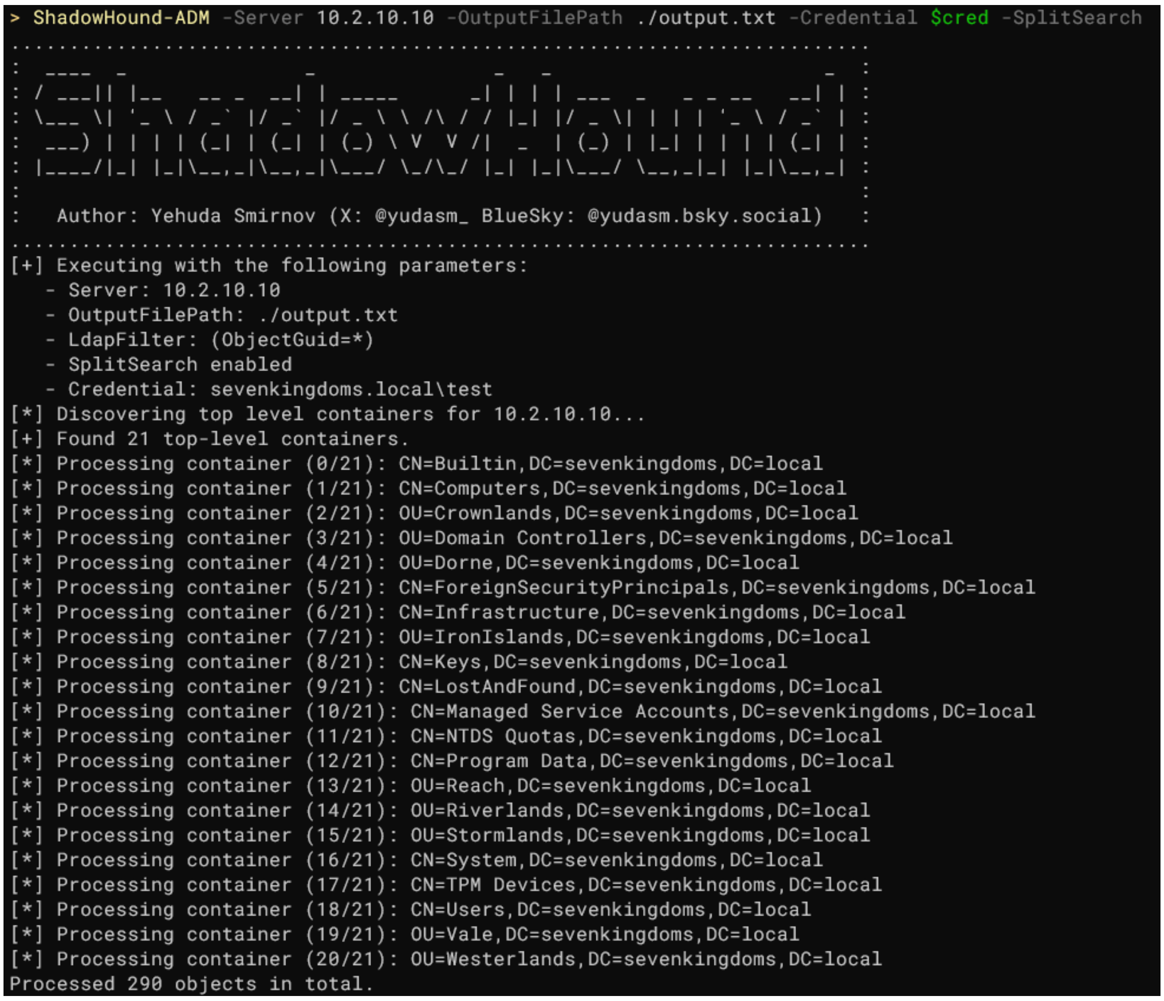
We should also note that ShadowHound is essentially just a post-processing script to convert DirectorySearcher or ADModule output to pyLdapSearch output. However, pyLdapSearch does not support ADWS (at least as of this moment), and requires Python to be installed, etc.

Shoutout to SoapHound

In the context of ADWS, it's worth mentioning **SoapHound** by **FalconForce, they made the initial public discovery about using ADWS** for Active Directory enumeration and created a tool for the task.

While SoapHound provides valuable functionality and incredible research, it is yet another binary that we need to introduce to monitored endpoints, and unfortunately we had a bit of trouble with using it against a very large domain (hence why we created the `-SplitSearch` flag to **split the enumeration to containers** and borrowed their split search idea with the `-LetterSplitSearch` flag to further split the search to objects beginning with a, b, c…).

How to Use ShadowHound



```
> ShadowHound-ADM -Server 10.2.10.10 -OutputFilePath ./output.txt -Credential $cred -SplitSearch
..........................................................................
:  ____  _               _               _   _                       _  :
: / ___|| |__   __ _  __| | _____      _| | | | ___  _   _ _ __   __| | :
: \___ \| '_ \ / _` |/ _` |/ _ \ \ /\ / / |_| |/ _ \| | | | '_ \ / _` | :
:  ___) | | | | (_| | (_| | (_) \ V  V /|  _  | (_) | |_| | | | | (_| | :
: |____/|_| |_|\__,_|\__,_|\___/ \_/\_/ |_| |_|\___/ \__,_|_| |_|\__,_| :
:                                                                        :
:   Author: Yehuda Smirnov (X: @yudasm_ BlueSky: @yudasm.bsky.social)   :
..........................................................................
[+] Executing with the following parameters:
   - Server: 10.2.10.10
   - OutputFilePath: ./output.txt
   - LdapFilter: (ObjectGuid=*)
   - SplitSearch enabled
   - Credential: sevenkingdoms.local\test
[*] Discovering top level containers for 10.2.10.10...
[+] Found 21 top-level containers.
[*] Processing container (0/21): CN=Builtin,DC=sevenkingdoms,DC=local
[*] Processing container (1/21): CN=Computers,DC=sevenkingdoms,DC=local
[*] Processing container (2/21): OU=Crownlands,DC=sevenkingdoms,DC=local
[*] Processing container (3/21): OU=Domain Controllers,DC=sevenkingdoms,DC=local
[*] Processing container (4/21): OU=Dorne,DC=sevenkingdoms,DC=local
[*] Processing container (5/21): CN=ForeignSecurityPrincipals,DC=sevenkingdoms,DC=local
[*] Processing container (6/21): CN=Infrastructure,DC=sevenkingdoms,DC=local
[*] Processing container (7/21): OU=IronIslands,DC=sevenkingdoms,DC=local
[*] Processing container (8/21): CN=Keys,DC=sevenkingdoms,DC=local
[*] Processing container (9/21): CN=LostAndFound,DC=sevenkingdoms,DC=local
[*] Processing container (10/21): CN=Managed Service Accounts,DC=sevenkingdoms,DC=local
[*] Processing container (11/21): CN=NTDS Quotas,DC=sevenkingdoms,DC=local
[*] Processing container (12/21): CN=Program Data,DC=sevenkingdoms,DC=local
[*] Processing container (13/21): OU=Reach,DC=sevenkingdoms,DC=local
[*] Processing container (14/21): OU=Riverlands,DC=sevenkingdoms,DC=local
[*] Processing container (15/21): OU=Stormlands,DC=sevenkingdoms,DC=local
[*] Processing container (16/21): CN=System,DC=sevenkingdoms,DC=local
[*] Processing container (17/21): CN=TPM Devices,DC=sevenkingdoms,DC=local
[*] Processing container (18/21): CN=Users,DC=sevenkingdoms,DC=local
[*] Processing container (19/21): OU=Vale,DC=sevenkingdoms,DC=local
[*] Processing container (20/21): OU=Westerlands,DC=sevenkingdoms,DC=local
Processed 290 objects in total.
```

Using AD Module based ShadowHound to enumerate all containers

ShadowHound comes in two flavors:

1. **ADModule Based**: This script leverages the Active Directory module (`Get-ADObject` cmdlet) in PowerShell, which communicates via the Active Directory Web Services (ADWS) protocol over port 9389.
2. **DirectorySearcher Based**: This script uses the `DirectorySearcher` class, performing LDAP queries directly.

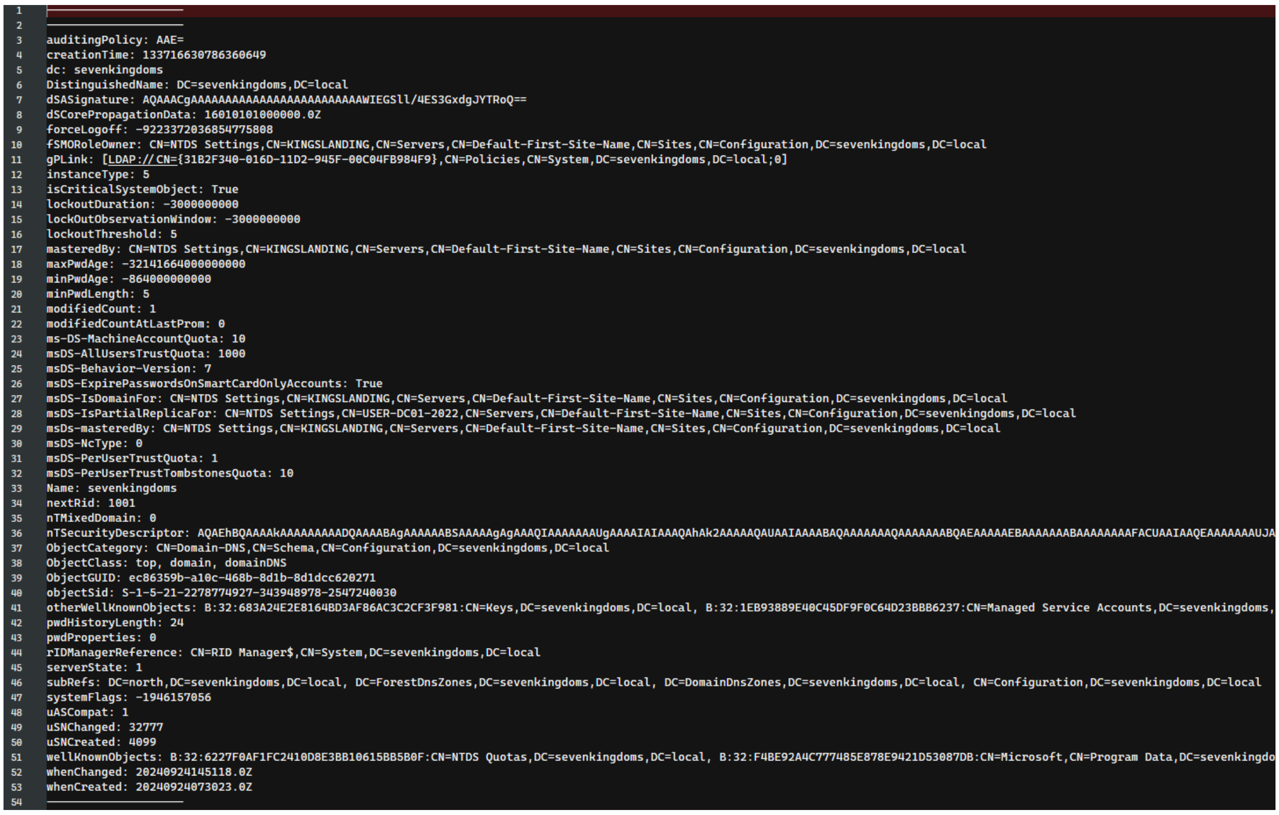
Getting started with ShadowHound is straightforward. Here's a basic example using the ADModule-based version:

```
Import-Module .\ShadowHound-ADM.ps1
ShadowHound-ADM -OutputFilePath "C:\Results\ldap_output.txt"
ShadowHound-ADM -Server "dc.domain.local" -OutputFilePath
"C:\Results\ldap_output.txt" -LdapFilter "(objectClass=user)"
$cred = Get-Credential
ShadowHound-ADM -OutputFilePath "C:\Results\ldap_output.txt" -
Credential $cred -SearchBase
"CN=Infrastructure,DC=sevenkingdoms,DC=local"
ShadowHound-ADM -OutputFilePath "C:\Results\ldap_output.txt" -
SplitSearch -LetterSplitSearch
ShadowHound-ADM -OutputFilePath "C:\Results\output.txt" -Certificates
ShadowHound-ADM -OutputFilePath "C:\Results\output.txt" -SplitSearch -
ParsedContainers "./parsed.txt"
```

And using the DirectorySearcher-based version:

```
Import-Module .\ShadowHound-DS.ps1
ShadowHound-DS -OutputFile "C:\Results\ldap_output.txt"
ShadowHound-DS -Server "dc.domain.local" -OutputFile
"C:\Results\ldap_output.txt"
ShadowHound-DS -OutputFile "C:\Results\ldap_output.txt" -LdapFilter "
(objectClass=computer)"
ShadowHound-DS -OutputFile "C:\Results\ldap_output.txt" -SearchBase
"CN=Infrastructure,DC=sevenkingdoms,DC=local"
ShadowHound-DS -OutputFile "C:\Results\cert_output.txt" -Certificates
$cred = Get-Credential
ShadowHound-DS -OutputFile "C:\Results\ldap_output.txt" -Credential
$cred
```

After running the script, you'll have an output file containing LDAP search results formatted for processing:



```
────────────────────
────────────────────
auditingPolicy: AAE=
creationTime: 133716630786360649
dc: sevenkingdoms
DistinguishedName: DC=sevenkingdoms,DC=local
dSASignature: AQAAACgAAAAAAAAAAAAAAAAAAAAAAAWIEGSll/4ES3GxdgJYTRoQ==
dSCorePropagationData: 16010101000000.0Z
forceLogoff: -9223372036854775808
fSMORoleOwner: CN=NTDS Settings,CN=KINGSLANDING,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=sevenkingdoms,DC=local
gPLink: [LDAP://CN={31B2F340-016D-11D2-945F-00C04FB984F9},CN=Policies,CN=System,DC=sevenkingdoms,DC=local;0]
instanceType: 5
isCriticalSystemObject: True
lockoutDuration: -3000000000
lockOutObservationWindow: -3000000000
lockoutThreshold: 5
masteredBy: CN=NTDS Settings,CN=KINGSLANDING,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=sevenkingdoms,DC=local
maxPwdAge: -32141664000000000
minPwdAge: -864000000000
minPwdLength: 5
modifiedCount: 1
modifiedCountAtLastProm: 0
ms-DS-MachineAccountQuota: 10
msDS-AllUsersTrustQuota: 1000
msDS-Behavior-Version: 7
msDS-ExpirePasswordsOnSmartCardOnlyAccounts: True
msDS-IsDomainFor: CN=NTDS Settings,CN=KINGSLANDING,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=sevenkingdoms,DC=local
msDS-IsPartialReplicaFor: CN=NTDS Settings,CN=USER-DC01-2022,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=sevenkingdoms,DC=local
msDs-masteredBy: CN=NTDS Settings,CN=KINGSLANDING,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=sevenkingdoms,DC=local
msDS-NcType: 0
msDS-PerUserTrustQuota: 1
msDS-PerUserTrustTombstonesQuota: 10
Name: sevenkingdoms
nextRid: 1001
nTMixedDomain: 0
nTSecurityDescriptor: AQAEhBQAAAAkAAAAAAAAAADQAAAABAgAAAAAABSAAAAAgAgAAAQIAAAAAAAUgAAAAIAIAAAQAhAk2AAAAAQAUAAIAAAABAQAAAAAAAQAAAAAAABQAEAAAAAEBAAAAAAABAAAAAAAAFACUAAIAAQEAAAAAAAUJA
ObjectCategory: CN=Domain-DNS,CN=Schema,CN=Configuration,DC=sevenkingdoms,DC=local
ObjectClass: top, domain, domainDNS
ObjectGUID: ec86359b-a10c-468b-8d1b-8d1dcc620271
objectSid: S-1-5-21-2278774927-343948978-2547240030
otherWellKnownObjects: B:32:683A24E2E8164BD3AF86AC3C2CF3F981:CN=Keys,DC=sevenkingdoms,DC=local, B:32:1EB93889E40C45DF9F0C64D23BBB6237:CN=Managed Service Accounts,DC=sevenkingdoms,
pwdHistoryLength: 24
pwdProperties: 0
rIDManagerReference: CN=RID Manager$,CN=System,DC=sevenkingdoms,DC=local
serverState: 1
subRefs: DC=north,DC=sevenkingdoms,DC=local, DC=ForestDnsZones,DC=sevenkingdoms,DC=local, DC=DomainDnsZones,DC=sevenkingdoms,DC=local, CN=Configuration,DC=sevenkingdoms,DC=local
systemFlags: -1946157056
uASCompat: 1
uSNChanged: 32777
uSNCreated: 4099
wellKnownObjects: B:32:6227F0AF1FC2410D8E3BB10615BB5B0F:CN=NTDS Quotas,DC=sevenkingdoms,DC=local, B:32:F4BE92A4C777485E878E9421D53087DB:CN=Microsoft,CN=Program Data,DC=sevenkingdo
whenChanged: 20240924145118.0Z
whenCreated: 20240924073023.0Z
────────────────────
```

Example output from ShadowHound

Handling Large Domains with ADWS

One of the challenges when dealing with extensive Active Directory environments is the potential for queries to time out using ADWS (generate the invalid enumeration context error). The ADModule version of ShadowHound addresses this with the `SplitSearch`, `Recurse` and `LetterSplitSearch` functionalities:

**** Once again thanks to FalconForce for the research and for coming up with a solution!**

- **SplitSearch**: When enabled, ShadowHound will split the search across top-level containers within the domain root or the within the provided container distinguished name (if you have those pesky ginormous containers), querying each one individually. This approach helps manage large datasets by breaking them into more manageable chunks.
- **Recurse**: If a container fails to return results within a 30-minute window (the point at which the enumeration context expires due to ADWS limitations), ShadowHound will automatically split that container into its sub-containers and attempt to retrieve the data again. This recursive approach ensures we (hopefully) do not miss data.
- **LetterSplitSearch**: Like SoapHound, if this flag is provided, the queries will be split into `(cn=a**)`, `(cn=b**)`, etc. If that query fails, the query will split yet again – `(cn=aa**)`, `(cn=ab**)`, `(cn==ac**)`.
  This flag can be combined with the SplitSearch flag to split each container into smaller queries.

Converting Data with BofHound

Once you've collected the data (which should look like ldapsearch output), the next step is to convert it into BloodHound-compatible JSON files using **BofHound**, which was recently updated:

```
python3 bofhound.py -i ldap_output.txt -p All --parser ldapsearch
```

BofHound will process the LDAP output and generate the JSON files you can import into BloodHound CE for analysis.

Splitting the JSON files

Depending on the file size (>100MB), you may want to split the output JSON file using tools like ShredHound or mgeeky's snippet.
From our experience, for large domains, even when BloodHound CE says the data was ingested, you should double check and try to query some data from the JSONs to confirm they were ingested properly.
If the data was not digested properly, you should use one the tools above to split it, even if it seems like a small amount of computers.

Detection Strategies

The effective method to detect ShadowHound (and the likes which perform AD enumeration) is to track identities that read an exceptionally high number of LDAP records within a specific time frame, as this behavior can indicate malicious enumeration similar to SharpHound's activities. Elastic's detection rule, for instance, flag such anomalies by identifying excessive directory object attribute requests. This was also noted by FalconForce's blog under the "Detecting SOAPHound" section.

Another interesting idea might be to setup honeypots in LDAP containers which trigger upon read access. This could be an indicator of attack.

Final Thoughts

ShadowHound offers an alternative approach to gather Bloodhound data by eliminating the need to deploy SharpHound or other binaries on the target endpoint. By leveraging the AD module or the DirectorySearcher class, we can reduce our the chance of detection on the endpoint itself.

While ShadowHound might minimize endpoint level detection, network activity should always remain a consideration. Be mindful of the queries you're performing and be aware that DCs (with the help of MDI and the likes) can flag unusual or suspicious LDAP activity (and with Machine Learning becoming more of a thing, we expect this to happen sooner than later).

Give ShadowHound a try and see how it works for your engagements. If you have any questions, run into issues, or have feedback—whether it’s about something that might be off or ideas for improvement—don’t hesitate to reach out. You can find me on X – @yudasm_ or send an email to info@fndsec.net.