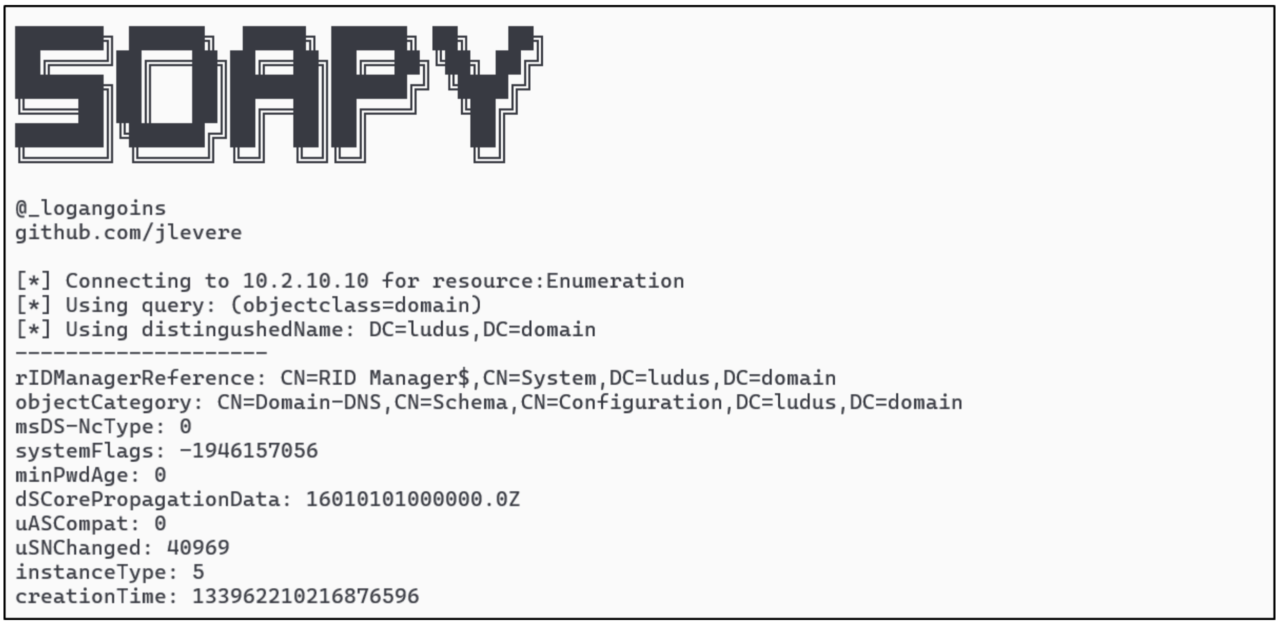
Make Sure to Use SOAP(y) – An Operators Guide to Stealthy AD Collection Using ADWS

specterops.io/blog/2025/07/25/make-sure-to-use-soapy-an-operators-guide-to-stealthy-ad-collection-using-adws

Logan Goins July 25, 2025



```
@_logangoins
github.com/jlevere

[*] Connecting to 10.2.10.10 for resource:Enumeration
[*] Using query: (objectclass=domain)
[*] Using distingushedName: DC=ludus,DC=domain
--------------------
rIDManagerReference: CN=RID Manager$,CN=System,DC=ludus,DC=domain
objectCategory: CN=Domain-DNS,CN=Schema,CN=Configuration,DC=ludus,DC=domain
msDS-NcType: 0
systemFlags: -1946157056
minPwdAge: 0
dSCorePropagationData: 16010101000000.0Z
uASCompat: 0
uSNChanged: 40969
instanceType: 5
creationTime: 133962210216876596
```

Gather domain object using SoaPy

__TL;DR__ Active Directory Web Services (ADWS) is a default service on domain controllers (DCs) which allows clients an alternative method to collect information stored in LDAP using the SOAP protocol for subsequent attacks. This article will look into maximizing this protocol's OPSEC-considerate collection workflow to its fullest, while exploring detection considerations along the way. This will mostly be done by utilizing SoaPy's newest features, in combination with Matt Creel's BOFHound.

If you would like to skip the background for understanding the context of SoaPy's newest operational workflow, please skip to the section "SoaPy in Action – Operationalizing ADWS Recon from Linux".

Background – What is ADWS and How Does it Work?

Microsoft introduced Active Directory Web Services (ADWS) in the Windows Server 2008 R2 release, providing a pseudo web interface to access Active Directory Domain Services (ADDS) and Active Directory Lightweight Directory Services (ADLDS) from the alternative port of *9389/TCP* using the SOAP protocol. Because of this, ADWS can be used to pull AD information such as users, groups, ACLs, etc. which can be paramount for attacker reconnaissance.

The ADWS service has been bundled in every Windows Server installation since 2008 R2, and is automatically exposed when promoting a server to a DC. Additionally, a large amount of Microsoft utilities used to manage ADDS over the years since ADWS's introduction in 2008 R2 use this service. This includes most of the Remote Server Administration Tools (RSAT) used on Windows server, including both PowerShell cmdlets and graphical tools.

While the service is named Active Directory **Web** Services, there's ironically near zero web technology included in the communication with this service. Instead of HTTP or HTTPS, the overarching ADWS communication uses a few layered proprietary Microsoft .NET focused technologies, including .NET Message Framing Protocol (MC-NMF), .NET Negotiate Stream Protocol (MS-NNS), and .NET Binary Format: SOAP Extension (MC-NBFSE); which in turn is an extension of .NET Binary Format: SOAP Data Structure (MC-NBFS) which itself is also an extension of .NET Binary Format: XML Data Structure (MC-NBFX). The IBM X-Force article I wrote last year on the topic, which can be found here, contains more information and an in-depth detailing about the layered protocols and how they function.

Past Tools and Previous Research

While this topic has been covered semi-in depth, being a relatively niche topic in Active Directory focused offensive tradecraft, only a few big sources of information online cover this service, with even fewer sources showcasing utilization in an operational context.

This idea was summarized flawlessly in a conversation with Garrett Foster here at SpecterOps. As I attempted to find documentation sources to extend my ADWS tooling SoaPy, he mentioned, "You chose the COM of AD enumeration", which is unfortunately and entirely true.

Early last year, when ADWS-focused collection tradecraft gained traction in the industry for the first time, two critical blog posts were released that showcased the service and offensive related use cases. This would be FalconForce's SOAPHound blog, which documented FalconForce's research and was used as a medium to release their SOAPHound tool. This being a .NET utility to perform BloodHound ingestion over ADWS. FalconForce's release of their research led to MDSec releasing their Active Directory Enumeration for Red Teams article, which stated that they held tradecraft related to this service since October 2021 but only decided to publicly release after noticing FalconForce's detailing of the technique. MDSec never released their .NET tooling associated with their ADWS collection research.

The community excitement generated from these two articles inspired Jackson Leverett and I to perform our own research and development on this technology during our summer 2024 internship at IBM X-Force Red. We recognized the limitations of SOAPHound from an OPSEC context, which is the requirement of both of a .NET assembly being executed on the host and the large-scale unconstrained queries that SOAPHound makes by default for BloodHound ingestion.

We asked the question, "What if we could interact with ADWS on the Linux platform, allowing the operator to proxy their traffic through a SOCKS proxy hosted on the C2 post-ex utility?" While this would be significantly more OPSEC safe, we realized that a large portion of the engineering required for SOAPHound leveraged .NET abstraction to interact with ADWS; which is something we don't have access to while using Python and Linux as our platform. After a significant struggle, a few months later we produced SoaPy: a utility written in native python for the Linux platform.

SoaPy is mostly a recreation of the various .NET only layered protocols previously mentioned completely from scratch, mostly following the byte-by-byte examples documented in various technical specifications from Microsoft. It allows an operator to proxy through their post-ex command and control (C2) utility and perform basic recon of users, computers, groups, etc. While additionally allowing an operator to write critical attributes leading to other attacks, such as writing `msDs-AllowedToActOnBehalfOfOtherIdentity` for Resource-Based Constrained Delegation (RBCD) Attacks.

SoaPy – What Now? Strengths and Weaknesses

After nearly a year of SoaPy being engineered, and a number of months after the public research and tool release, I wanted to reflect on SoaPy's after-release place in the industry, while highlighting the SoaPy's strengths and weaknesses.

While SoaPy was a big step in the community and industry understanding of ADWS protocol specifics, it was a much smaller step in improving stealth in the AD collection tradecraft space. It mainly provided a stepping stone to assist in expanding on our work using ADWS from Linux, with Jack and I hoping other people would develop additional tools that capitalized on the protocol-specific work we had done.

SoaPy was simple and less operationally viable than we hoped for. It was mainly good at small scale reconnaissance of objects, while the unintended write capability ended up being more useful than previously imagined. From a collection perspective, while SoaPy could be used to identify basic users, groups, computers, and output then to the terminal, it didn't have the operational viability of ingesting directly into BloodHound that SOAPHound had. While SoaPy's usefulness for writing objects as a part of post-ex is a welcome strength, collection was always what we were aiming for. Initially, we just couldn't find a way to make our tool as operationally useful as SOAPHound. Months after release, I even attempted to pull apart Dirk-Jan's BloodHound.py to see if I could integrate our ADWS collection library instead of the ldap3 library. It proved to be incredibly difficult and complex.

SoaPy – The Next Evolution

After talking to Matt Creel, he recommended I make SoaPy compatible with BOFHound. I was familiar with using the ldapsearch BOF from TrustedSec's Situational Awareness repo in combination with Matt's BOFHound to collect Active Directory information in an incremental or constrained manner, allowing an operator to slowly build their collection of the environment. This technique is preferable on red team ops due to the fine-grained control over queries, making reconnaissance less likely to trip potential canary objects or have defensive solutions such as Microsoft Defender for Identity (MDI) detect signatured queries.

Before having this conversation with Matt, I overestimated the constraints of BOFHound. My previous understanding was that BOFHound was required to ingest data from Cobalt Strike logs; however, after looking into it further, BOFHound just ingests specially formatted text representing LDAP objects regardless of the C2 framework or execution tool. This was a pretty easy solution to implement into the already present attribute parsing functionality, with the only major change that required implementation was finding out how to configure LDAP control attributes to show the nTSecurityDescriptor attribute through ADWS.

Now with the additional compatibility layer between SoaPy and BloodHound attack graphing capabilities, OPSEC safe and operationally efficient collection can take place from ADWS from Linux. Now in addition to ingesting BloodHound data from SOAPHound, BloodHound data can be ingested from SoaPy. On top of the other benefits that come from collection over ADWS, this technique still holds the operational benefits that come with using BOFHound with ldapsearch.

A few other changes were introduced into SoaPy as part of the attribute output workflow, such as limiting the parsing of the attributes on SoaPy's side. Because BOFHound takes a certain format of LDAP attributes for its own parsing, I removed some of the additional parsing SoaPy does by default. This includes displaying additional information as part of the attribute output, such as formatted timestamps, userAccountControl values, etc. In an attempt to keep the small-scale use case functionality of SoaPy, I've included a "`--parse`" flag, which allows you to parse the unparsed by default attributes in case you wish to view complex attributes directly in the terminal for easy and quick reference.

SoaPy in Action – Operationalizing ADWS Recon From Linux

After establishing some form of ceded access into an internal Active Directory environment, it's now time to perform some recon/collection of the internal Active Directory data. Assuming our agent is operating in a low-privilege user context, we want to find potential paths to Tier Zero assets which might allow us to accomplish our objectives. We want to remain undetected, as to not trigger any alerts or incident response (IR) procedures.

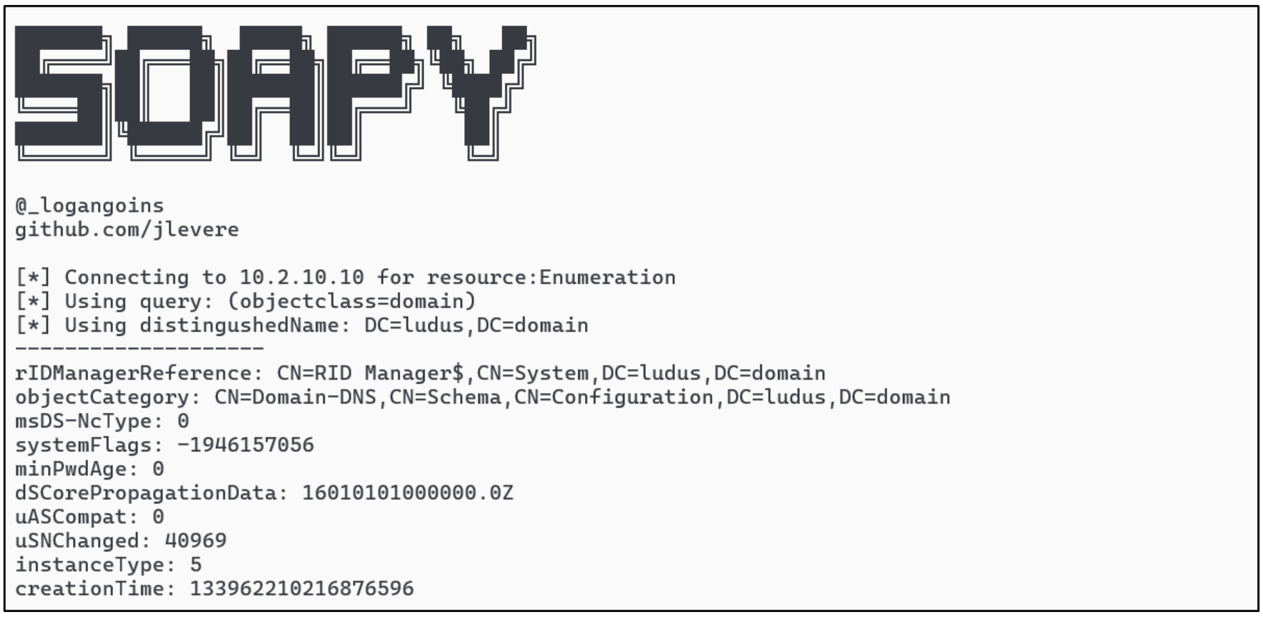
Our best options for collection include methods which use constrained or incremental queries, which either allow us to collect groups of LDAP objects slowly or allow us to gather information by some sort of incremental identifier, such as objectSid. We want to collect over ADWS to provide a more OPSEC considerate approach than using LDAP, so we'll have to use SOAPHound or SoaPy. SOAPHound would be particularly noisy, since we would need to execute a .NET assembly on a host in memory to perform collection and write the resulting JSON collection to disk. Additionally, the larger scale queries might trip a SACL canary object (which, yes, still works as the best detection method for ADWS).

Meanwhile, a less noisy solution for this problem would be proxying SoaPy through our compromised host. Due to the new BOFHound integration and fine grained control over queries with no reliance on host-based code execution, we can perform collection slowly or group objects with incremental recon while still holding the stealthy considerations of performing object interaction over ADWS.

For example, after acquiring credentials in the form of a password or NT hash we can target high impact objects directly using constrained queries, such as the Certificate Authority (CA) and template objects defined in the internal environments Active Directory Certificate Services (ADCS). Although, before directly targeting the ADCS objects, it’s important to ingest the root domain object.

This can be done with the following SoaPy ingestion command as an example:

```
soapy ludus.domain/jdoe:'P@ssw0rd'@10.2.10.10 -q '(objectclass=domain)' | tee
data/domain.log
```



```
@_logangoins
github.com/jlevere

[*] Connecting to 10.2.10.10 for resource:Enumeration
[*] Using query: (objectclass=domain)
[*] Using distingushedName: DC=ludus,DC=domain
--------------------
rIDManagerReference: CN=RID Manager$,CN=System,DC=ludus,DC=domain
objectCategory: CN=Domain-DNS,CN=Schema,CN=Configuration,DC=ludus,DC=domain
msDS-NcType: 0
systemFlags: -1946157056
minPwdAge: 0
dSCorePropagationData: 16010101000000.0Z
uASCompat: 0
uSNChanged: 40969
instanceType: 5
creationTime: 133962210216876596
```

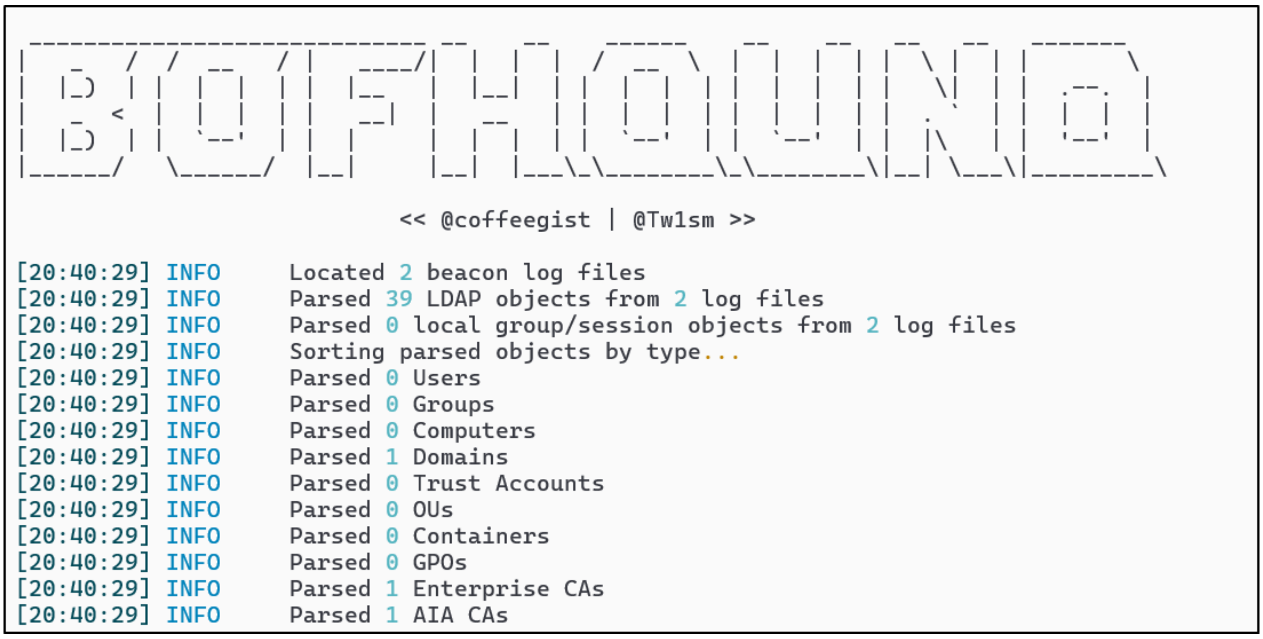
Gather domain object using SoaPy

Next, ingest all the ADCS objects necessary from the Configuration Naming Context. This includes objects of class `pkiCertificateTemplate`, `certificationAuthority`, `mspki-Enterprise-Oid`, and `pkiEnrollmentService`, using the command:

```
soapy ludus.domain/jdoe:'P@ssw0rd'@10.2.10.10 -q '(|
(objectclass=pkiCertificateTemplate)(objectclass=CertificationAuthority)
(objectClass=pkiEnrollmentService)(objectclass=msPKI-Enterprise-Oid))' -dn
'CN=Configuration,DC=ludus,DC=domain' | tee data/adcs.log
```

After all data required for display in BloodHound is accounted for, we can begin converting it to BloodHound compatible JSON data with BOFHound using the command:

```
bofhound -i data --zip
```



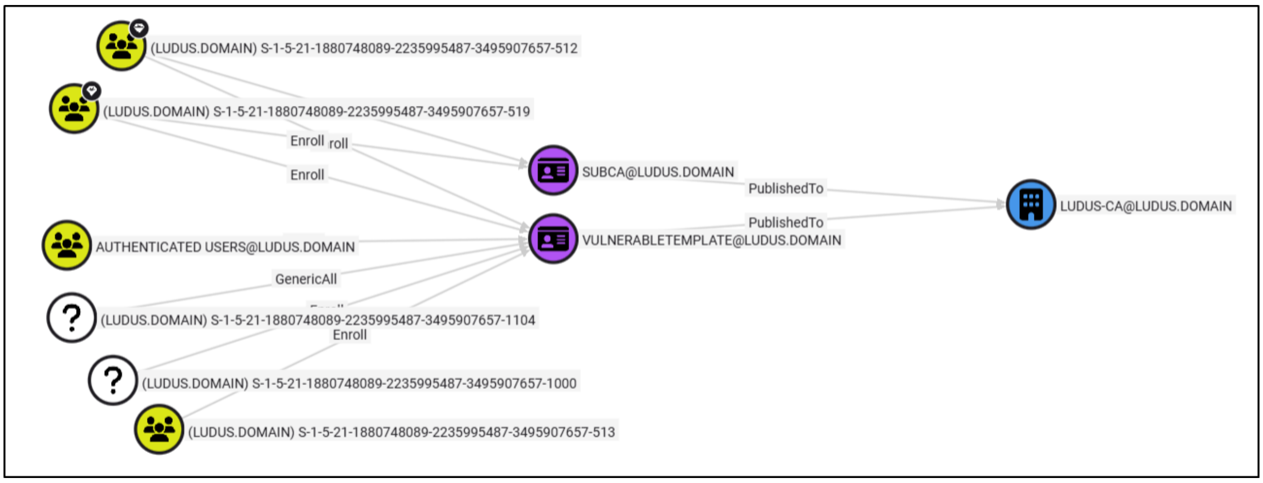
```
 _____________________________ __    __    ______    __    __   __   __   _______
|   _   /  /  __   / |    ____/|  |  |  |  /  __  \  |  |  |  | |  \ |  | |       \
|  |_)  | |  |  |  | |   |__   |  |__|  | |  |  |  | |  |  |  | |   \|  | |  .--.  |
|   _  <  |  |  |  | |    __|  |   __   | |  |  |  | |  |  |  | |  . `  | |  |  |  |
|  |_)  | |  `--'  | |   |     |  |  |  | |  `--'  | |  `--'  | |  |\   | |  '--'  |
|______/   \______/  |__|      |__|  |___\_\________\_\________\|__| \___\|_________\

                            << @coffeegist | @Tw1sm >>

[20:40:29] INFO     Located 2 beacon log files
[20:40:29] INFO     Parsed 39 LDAP objects from 2 log files
[20:40:29] INFO     Parsed 0 local group/session objects from 2 log files
[20:40:29] INFO     Sorting parsed objects by type...
[20:40:29] INFO     Parsed 0 Users
[20:40:29] INFO     Parsed 0 Groups
[20:40:29] INFO     Parsed 0 Computers
[20:40:29] INFO     Parsed 1 Domains
[20:40:29] INFO     Parsed 0 Trust Accounts
[20:40:29] INFO     Parsed 0 OUs
[20:40:29] INFO     Parsed 0 Containers
[20:40:29] INFO     Parsed 0 GPOs
[20:40:29] INFO     Parsed 1 Enterprise CAs
[20:40:29] INFO     Parsed 1 AIA CAs
```

Ingest Objects into BOFHound

After converting the SoaPy data to BloodHound formatted JSON, we can now upload it into the BloodHound graphical interface. Use the built-in cypher queries to identify potential ADCS attack vectors, such as the Enrollment rights on published ESC1 certificate templates as an example. After using this query, we can see some enrollment permissions on a few ESC1 vulnerable templates.



(LUDUS.DOMAIN) S-1-5-21-1880748089-2235995487-3495907657-512
(LUDUS.DOMAIN) S-1-5-21-1880748089-2235995487-3495907657-519
Enroll
roll
Enroll
SUBCA@LUDUS.DOMAIN
PublishedTo
LUDUS-CA@LUDUS.DOMAIN
PublishedTo
VULNERABLETEMPLATE@LUDUS.DOMAIN
AUTHENTICATED USERS@LUDUS.DOMAIN
GenericAll
(LUDUS.DOMAIN) S-1-5-21-1880748089-2235995487-3495907657-1104
Enroll
(LUDUS.DOMAIN) S-1-5-21-1880748089-2235995487-3495907657-1000
(LUDUS.DOMAIN) S-1-5-21-1880748089-2235995487-3495907657-513

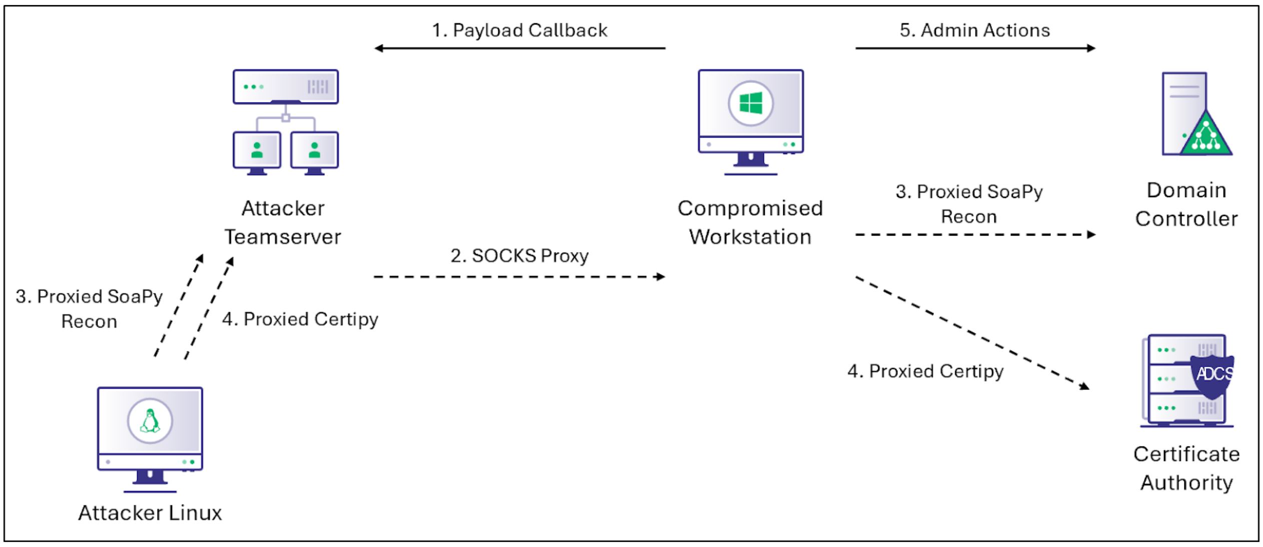
Potential ADCS Attack Path

Note some principals which have enrollment permissions over the “Vulnerable Template” certificate template are unknown, this is because they have not been ingested yet due to our queries being constrained to ADCS related objects. If information on these unknown principals is required, perform another query and filter by `(objectSid=<sid>)`, then convert with BOFHound and upload into BloodHound.

Taking a look at our potential paths of exploitation, we can see that the built-in group “Authenticated Users” has enrollment permissions. The reason this group is propagated within BloodHound without being ingested is because the “Authenticated Users” `objectSid` is static in all domains as `S-1-5-11`. With this information, we can request the "Vulnerable Template" certificate template and specify an arbitrary `SubjectAltName` (SAN), leading to full domain compromise.

The most OPSEC safe example I can think of to carry out domain privilege escalation using this acquired knowledge would be to first proxy certipy through your post-exploitation utility to request the certificate as a Domain Admin so you don't have to execute Certify as a .NET assembly. Then use `certipy auth -pfx <cert>` while using the -no-hash option to not perform UnPAC-the-Hash which is commonly detected, additionally use the `-kirbi` option so we can utilize the requested ticket on Windows. Finally, using Kerbeus or some other Kerberos BoF, enter the acquired ticket in base64 format into the current session and perform actions.

A diagram showcasing this path is displayed below:



Attack Path Diagram

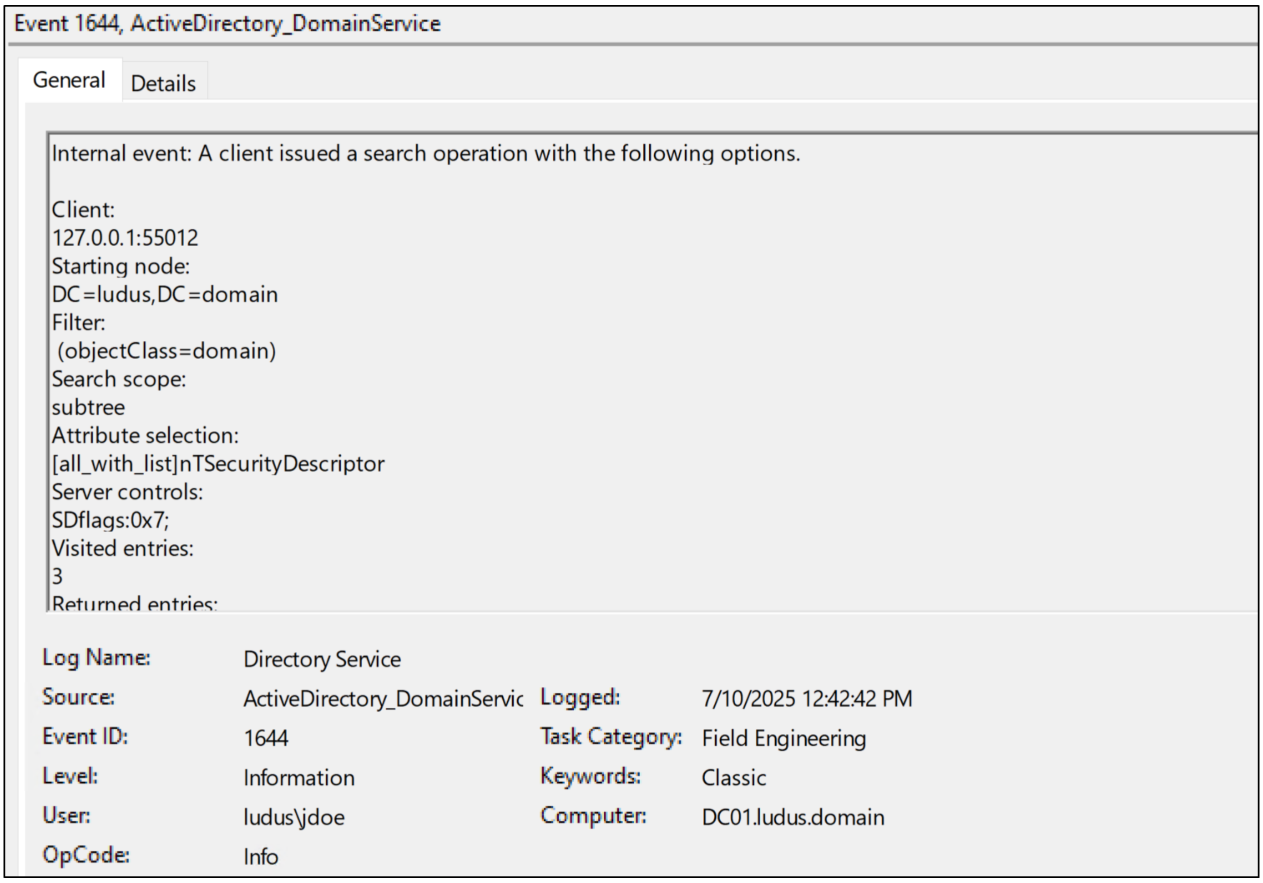
Detecting ADWS Collection – What Can Defenders Do?

How can we detect an attack similar to this? LDAP collection is at the start of every Active Directory-focused attack chain. Attackers use it to understand the environment's weaknesses and what actions they need to take to exploit them. An adversary's ability to access LDAP data without being detected is nearly always critical to their operation, so detecting them when they're just starting to understand your environment could be a big win before the impact typically associated with a breach. While applying preventative measures or creating custom detections for Active Directory collection isn't the only defensive mechanism you should be using to secure your internal environment; it is one layer as a part of a defense-in-depth approach.

There is a major single point of detection that can be applied to both LDAP collection and ADWS collection which can optimize the chances of detecting adversarial reconnaissance: Active Directory Domain Services (ADDS). Logging for ADDS occurs for both LDAP and ADWS collection, allowing security teams and system administrators additional insight into the "Directory Service" object related actions taken. ADDS logging is also usually used to detect LDAP recon, making it the perfect source to also detect ADWS because it's likely already configured in your environment if you're ingesting LDAP related logging. Microsoft's recommended AD and LDS logging instructions can be found here, but if you want a quick way to test these detections in a lab environment, you can set or create these registry keys on your DC:

- `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\NTDS\Diagnostics\15 Field Engineering`
- `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\NTDS\Parameters\Expensive Search Results Threshold`
- `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\NTDS\Parameters\Inefficient Search Results Threshold`
- `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\NTDS\Parameters\Search Time Threshold (msecs)`

After configuring the "Field Engineering" Task Category to maximum verbosity, we'll have additional visibility. As an example, here's an event collected when executing SoaPy against the DC with the query `(objectClass=domain)`:



Event 1644, ActiveDirectory_DomainService

General | Details

```
Internal event: A client issued a search operation with the following options.

Client:
127.0.0.1:55012
Starting node:
DC=ludus,DC=domain
Filter:
 (objectClass=domain)
Search scope:
subtree
Attribute selection:
[all_with_list]nTSecurityDescriptor
Server controls:
SDflags:0x7;
Visited entries:
3
Returned entries:
```

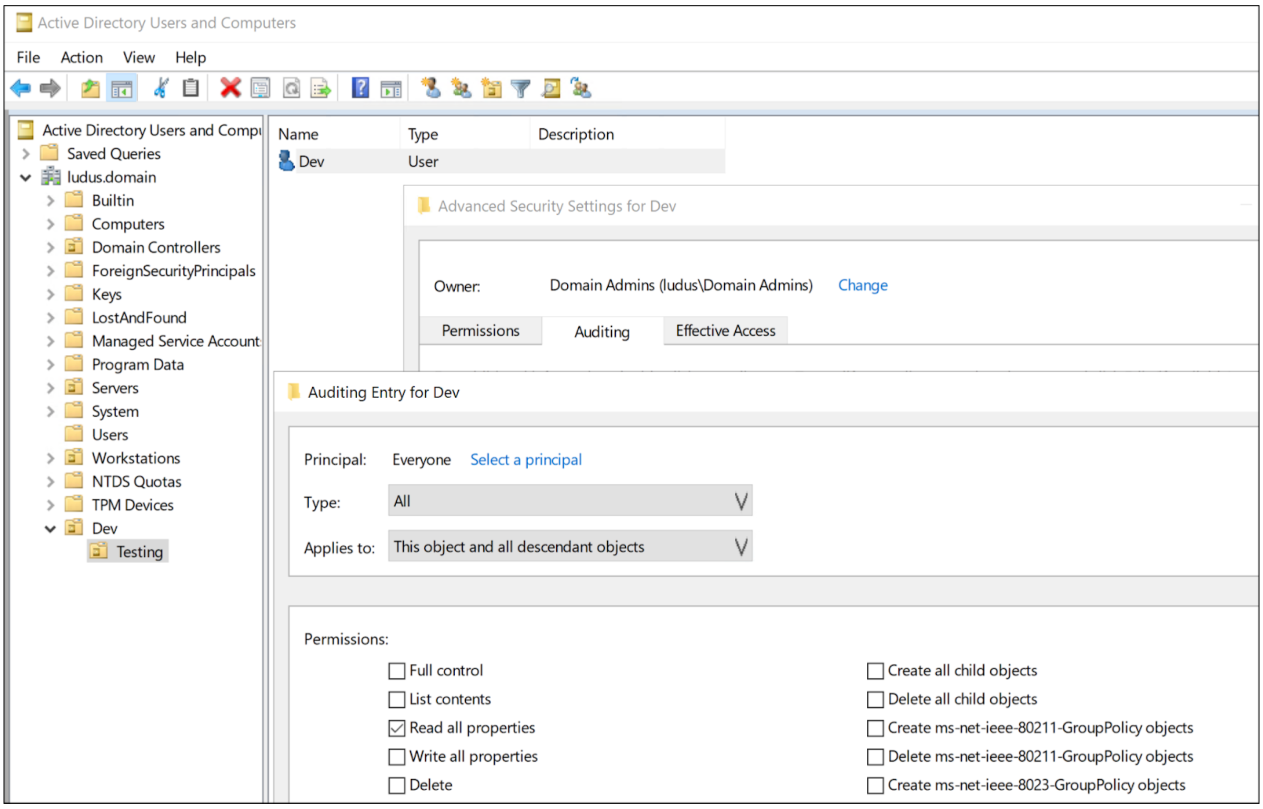
Log Name:	Directory Service		
Source:	ActiveDirectory_DomainServic	Logged:	7/10/2025 12:42:42 PM
Event ID:	1644	Task Category:	Field Engineering
Level:	Information	Keywords:	Classic
User:	ludus\jdoe	Computer:	DC01.ludus.domain
OpCode:	Info		

Sample LDAP Logging Event

Note a few details listed in this event. First off, the query field is completely visible, the computer field contains the DC where the query executed instead of the real host the request originated from, the client address is listed as loopback, and the user is listed as a low-privilege domain user instead of the DC. This isn't a whole lot of information you can use for triage or detection, especially since the host the query executed on is abstracted by how ADWS works on the backend and low-privilege interaction with ADWS is conducted commonly by system administrators using RSAT. From a triage perspective, the best way you might be able to figure out where the query originates from is by using netsession enumeration to find sessions opened with the originating user in the log; however, there is no guarantee the adversary isn't using a different set of stolen credentials when executing SoaPy on their Linux host.

While it's difficult to detect ADWS recon based on any of the fields in the ADDS interaction, there is still an extremely underrated and underutilized technique of ADDS recon detection that will work for both LDAP and ADWS: System Access Control List (SACL) canaries. Specifically referring to SACL canary objects which send audit logs on ReadProperty interaction, because as seen from Alex Demine's Don't Touch That Object! Finding SACL Tripwires During Red Team Ops blog, attackers easily detect and avoid SACLs which only alert on object modification. But, as additionally discussed in Alex's blog, SACL canary objects which log on simply accessing or reading the "tripwire" is the optimal way to force the adversary to trigger it during malicious recon.

SACLs can be configured in the Advanced Security Settings of an object on the Auditing tab in Active Directory Users and Computers (ADUC). Here, I've configured a SACL in my custom `Dev` organizational unit (OU) which alerts on `ReadProperty` from the `Everyone` group:



Owner: Domain Admins (ludus\Domain Admins)

Principal: Everyone

Type: All

Applies to: This object and all descendant objects

Permissions:

☐ Full control	☐ Create all child objects
☐ List contents	☐ Delete all child objects
☑ Read all properties	☐ Create ms-net-ieee-80211-GroupPolicy objects
☐ Write all properties	☐ Delete ms-net-ieee-80211-GroupPolicy objects
☐ Delete	☐ Create ms-net-ieee-8023-GroupPolicy objects

Example SACL Canary Object

As a simple example, this user canary object has been configured with a dummy `servicePrincipalName` attribute. This means that in addition to triggering when listing objects with `objectclass=user`/`objectclass=person`, adversaries attempting to locate potentially Kerberoastable objects will trip this SACL canary.

Events triggering these canaries will be logged as event code 4662, which defensive products already commonly use. For example, this Elastic pre-built detection rule with the query:

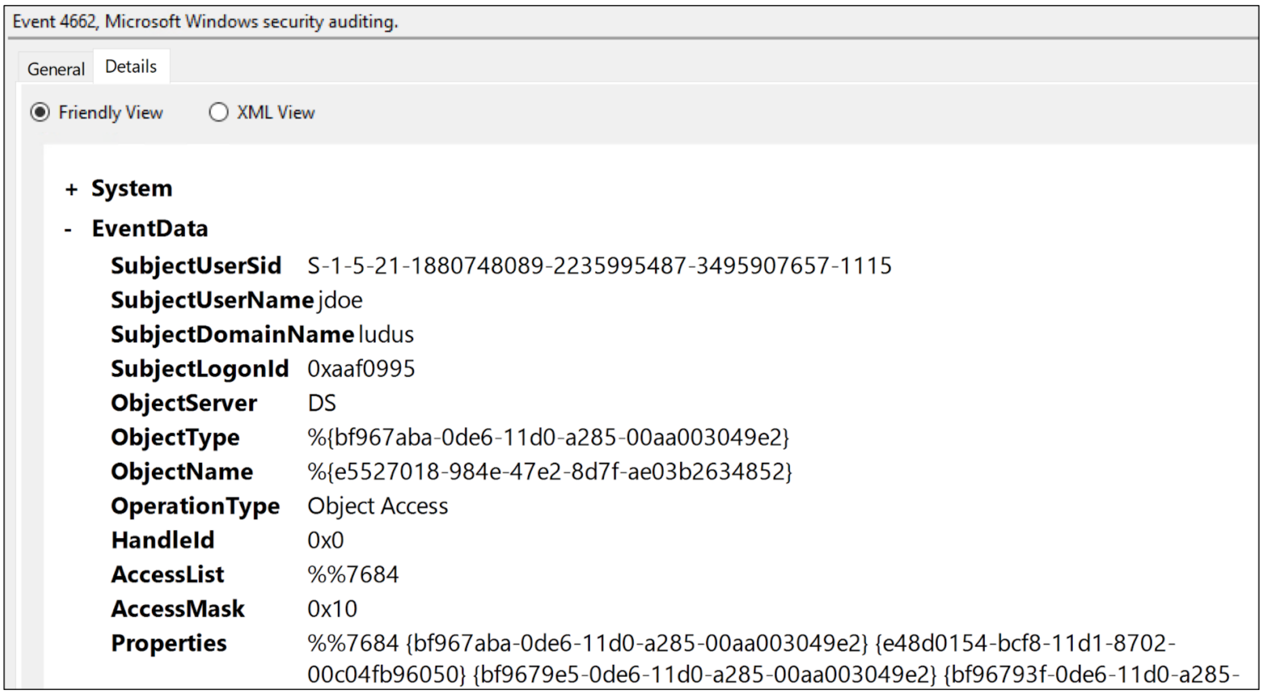
```
any where event.code == "4662" and not winlog.event_data.SubjectUserSid : "S-1-5-18" and
winlog.event_data.AccessMaskDescription == "Read Property" and
length(winlog.event_data.Properties) >= 2000
```

SACL Alert Detection Query

This detection is triggered for events with event code `4662` which do not include the `SubjectUserSid` of `LOCAL SYSTEM` using `ReadProperty`. For ADWS collection, thankfully when the SACL is triggered the `SubjectUserSid` is logged as the user context conducting the query, so this alert fires correctly. For example, when I execute the command:

```
soapy ludus.domain/jdoe:'P@ssw0rd'@10.2.10.10 -q '(serviceprincipalname=*)'
```

The event details logged are:



```
+ System
- EventData
    SubjectUserSid     S-1-5-21-1880748089-2235995487-3495907657-1115
    SubjectUserName    jdoe
    SubjectDomainName  ludus
    SubjectLogonId     0xaaf0995
    ObjectServer       DS
    ObjectType         %{bf967aba-0de6-11d0-a285-00aa003049e2}
    ObjectName         %{e5527018-984e-47e2-8d7f-ae03b2634852}
    OperationType      Object Access
    HandleId           0x0
    AccessList         %%7684
    AccessMask         0x10
    Properties         %%7684 {bf967aba-0de6-11d0-a285-00aa003049e2} {e48d0154-bcf8-11d1-8702-
                       00c04fb96050} {bf9679e5-0de6-11d0-a285-00aa003049e2} {bf96793f-0de6-11d0-a285-
```

SACL Trigger Event

Conclusion

Fine-grained stealthy ADWS collection of AD data got a nice small level up with SoaPy's newest additions, including the ability to list DACLs with ADWS integrated LDAP controls combined with a formatting overhaul for integration into Matt Creel's BOFHound. While attackers can now perform AD recon over ADWS from Linux far more efficiently while evading common detections, that doesn't mean there's nothing that can be done defensively to detect adversarial collection.

Configuring SACL canaries is the best way to detect both LDAP and ADWS collection, plus it’s an extremely underrated and underutilized method of detection with currently no way around it. Creating canary objects could allow your organization to detect and stop adversaries before they perform execution of dangerous Active Directory privilege escalation attacks, forcing them to trigger detections directly in their recon phase when they’re just starting to understand your environment. Apply defense-in-depth, and ensure your internal environment is just as locked down as your external network perimeter from both a preventative and detection standpoint.

Post Views: 2,827