LDAP authentication in Active Directory environments

offsec.almond.consulting/ldap-authentication-in-active-directory-environments.html

Published on Tue 31 October 2023 (2023-10-31T13:37:00+01:00) by @lowercase_drm

Understanding the different types of LDAP authentication methods is fundamental to apprehend subjects such as relay attacks or countermeasures. This post introduces them through the lens of Python libraries.

LDAP authentication methods

According to Microsoft, Active Directory supports 3 authentication methods on LDAP connection:

- **Simple**: Simple username/password as defined in (one of) the LDAP RFC.
- **Sicily**: This legacy protocol is another protocol to negotiate underlying authentication method. Active Directory only supports NTLM as an authentication protocol with Sicily.
- **Simple Authentication and Security Layer (SASL)**: SASL is a framework for authentication, it allows client and server to negotiate an authentication method among those supported. SASL authentication adds 4 authentication subtypes:
  - **GSS-SPNEGO**: Simple and Protected GSSAPI Negotiation Mechanism, yet another protocol to negotiate authentication. Active Directory provides NTLM or Kerberos as underlaying methods.
  - **GSSAPI**: Kerberos (*well, actually, GSSAPI is a lot more than "Kerberos", but in Active Directory environment and to simplify the topic, consider they are the same*)
  - **EXTERNAL**: in the SASL framework, EXTERNAL mode is used by the client to tell the server to use information provided outside of what is strictly considered LDAP communications, for example, the GID/UID of the client process if the communication is done through a Unix socket or the TLS client certificate if the connection is done via TLS. Active Directory seems to exclusively use this mode to support TLS authentication via Schannel.
  - **DIGEST-MD5**: This legacy protocol is based on the HTTP Digest Authentication; it is thus a challenge/response authentication protocol. It is required by the LDAPv3 RFC but is now deprecated.

In total, LDAP on Active Directory supports 6 "kinds" of authentication.

Side note: SASL mechanisms supported by the target domain controller can be anonymously requested:

```
>>> import ldap3
>>> server = ldap3.Server('ldap://kingslanding.sevenkingdoms.local',
get_info=ldap3.ALL)
>>> connection = ldap3.Connection(server)
>>> connection.bind()
True
>>> connection.server.info.supported_sasl_mechanisms
['GSSAPI', 'GSS-SPNEGO', 'EXTERNAL', 'DIGEST-MD5']
```

Protections

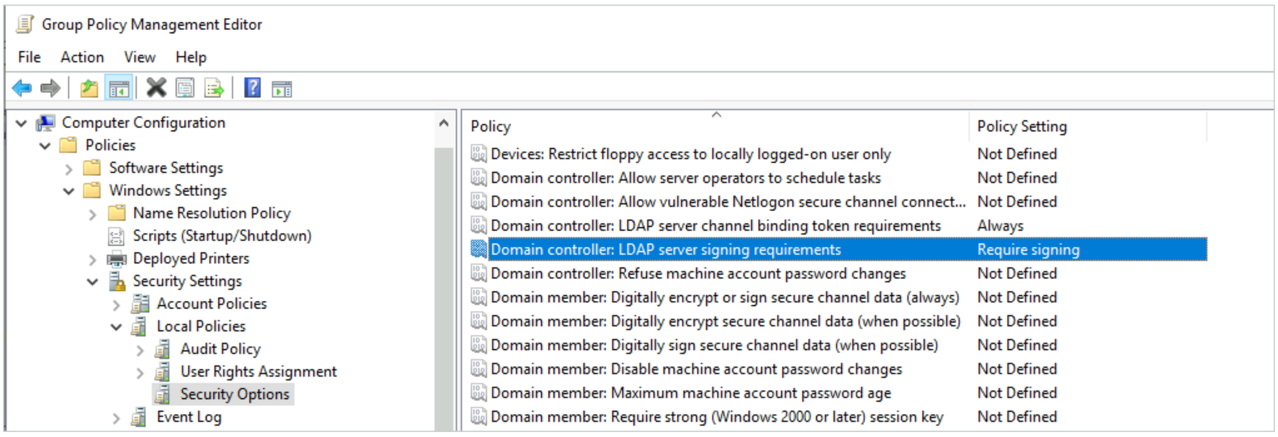
The most (in)famous attacks against LDAP authentication are relay attacks, and more precisely NTLM relay. Keep in mind that relaying Kerberos authentication is theoretically possible but not as easy to do in practice and beyond the scope of this blog post. Moreover, authenticating with certificates through Schannel is not impacted by relay attacks because of how it intrinsically works.

Microsoft provides two countermeasures to protect LDAP authentications against relay attacks, as described in KB4520412. Those two protections are LDAP signing and LDAP channel binding.

LDAP Signing

The idea here is to encrypt or sign the LDAP payload with a shared secret between the client and the server. NTLM, Kerberos and DIGEST-MD5 based authentications implement this protection. Unlike what the name suggests, "LDAP signing" protection entails the signing of the LDAP payload, as well as its (optional) encryption, depending on what is negotiated between the client and the server. When signature or encryption is negotiated, it is no longer possible for an attacker to send arbitrary command after the authentication relay.

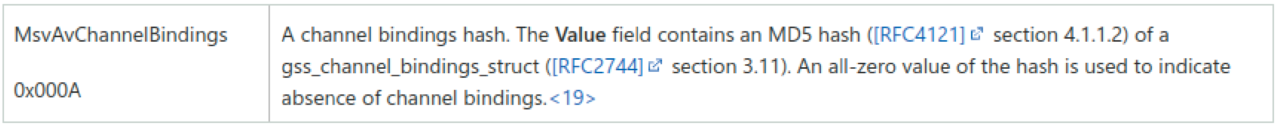
To enable LDAP signing, `Domain controller: LDAP server signing requirements` must be set to `Require signing` within the domain controller policy.



Policy	Policy Setting
Devices: Restrict floppy access to locally logged-on user only	Not Defined
Domain controller: Allow server operators to schedule tasks	Not Defined
Domain controller: Allow vulnerable Netlogon secure channel connect...	Not Defined
Domain controller: LDAP server channel binding token requirements	Always
Domain controller: LDAP server signing requirements	Require signing
Domain controller: Refuse machine account password changes	Not Defined
Domain member: Digitally encrypt or sign secure channel data (always)	Not Defined
Domain member: Digitally encrypt secure channel data (when possible)	Not Defined
Domain member: Digitally sign secure channel data (when possible)	Not Defined
Domain member: Disable machine account password changes	Not Defined
Domain member: Maximum machine account password age	Not Defined
Domain member: Require strong (Windows 2000 or later) session key	Not Defined

Channel Binding

This protection is designed to prevent relaying authentications to LDAPS. The idea is to bind the outer secure connection (TLS in our case) to application data over an inner client-authenticated channel (NTLM here). More precisely, Active Directory implements 'tls-server-end-point' channel binding Type, the general concept is covered here. This kind of channel binding seems to be the only one supported by Active Directory during NTLM and Kerberos authentications with LDAP. Channel binding during NTLM authentication is performed by adding a new `AV_PAIR` (attributes/value pair structure defined in MS-NLMP 2.2.2.1) within the `AUTHENTICATE_MESSAGE` (MS-NLMP 2.2.1.3). This new `AV_PAIR` has `AvId` which is `0x000A` (`MsvAvChannelBindings`).



MsvAvChannelBindings 0x000A	A channel bindings hash. The **Value** field contains an MD5 hash ([RFC4121] section 4.1.1.2) of a gss_channel_bindings_struct ([RFC2744] section 3.11). An all-zero value of the hash is used to indicate absence of channel bindings.<19>

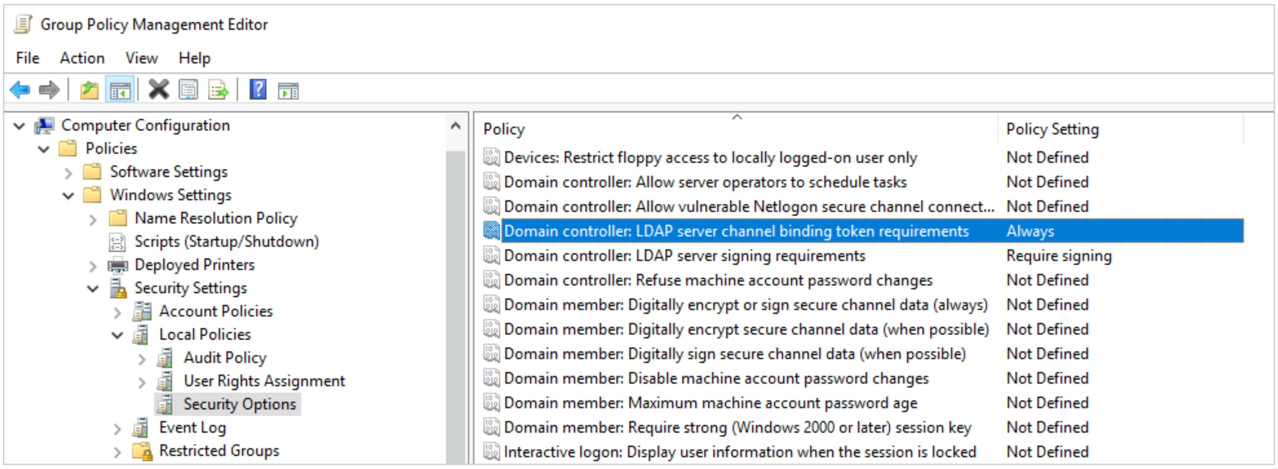
The `Value` field contains a MD5 hash of a `gss_channel_bindings_struct` struct which basically is a magic constant concatenated with the SHA256 hash of the server certificate. Here is an example of the simplified struct represented in pseudo-code:

```
gss_channel_bindings_struct {
    initiator_addrtype: '\x00\x00\x00\x00';
    initiator_address:  '\x00\x00\x00\x00';
    acceptor_addrtype:  '\x00\x00\x00\x00';
    acceptor_address:   '\x00\x00\x00\x00';
    application_data:   '\x55\x00\x00\x00' + 'tls-server-end-point:' +
'ae2670228880c00ad0448c156575212d1bbdaea7dde68f88a41ff5e084940f92';
                        # [len(magic+hash)]        # [magic]
# [hash]
}
```

Active Directory services use Schannel as Security Service Provider (SSP) to handle TLS. According to the documentation, this SSP only uses the `application_data` field to perform channel binding.

Regarding Kerberos, instead of being added within the `AUTHENTICATE_MESSAGE`, the hash is put into the `cksum` field and then the client adds it to the `authenticator`.

To require channel binding on LDAP, `Domain controller: LDAP server channel binding token requirements` must be set to `Always` within the domain controller policy. Keep in mind that 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 (though, if your domain controllers have not been upgraded since March 2020, maybe GPO support for channel binding is not your priority). Microsoft provides a guide to secure LDAP/S services: ADV190023.


Group Policy Management Editor

File Action View Help

- Computer Configuration
  - Policies
    - Software Settings
    - Windows Settings
      - Name Resolution Policy
      - Scripts (Startup/Shutdown)
      - Deployed Printers
      - Security Settings
        - Account Policies
        - Local Policies
          - Audit Policy
          - User Rights Assignment
          - Security Options
        - Event Log
        - Restricted Groups

Policy	Policy Setting
Devices: Restrict floppy access to locally logged-on user only	Not Defined
Domain controller: Allow server operators to schedule tasks	Not Defined
Domain controller: Allow vulnerable Netlogon secure channel connect...	Not Defined
Domain controller: LDAP server channel binding token requirements	Always
Domain controller: LDAP server signing requirements	Require signing
Domain controller: Refuse machine account password changes	Not Defined
Domain member: Digitally encrypt or sign secure channel data (always)	Not Defined
Domain member: Digitally encrypt secure channel data (when possible)	Not Defined
Domain member: Digitally sign secure channel data (when possible)	Not Defined
Domain member: Disable machine account password changes	Not Defined
Domain member: Maximum machine account password age	Not Defined
Domain member: Require strong (Windows 2000 or later) session key	Not Defined
Interactive logon: Display user information when the session is locked	Not Defined

ldap3

The ldap3 library is a pure python implementation of the LDAP 3 RFC and is widely used in offensive tools. It natively supports 5 (sub) authentication methods when used against domain controllers:

- Simple
- Sicily
- SASL (GSSAPI)
- SASL (EXTERNAL)
- SASL (DIGEST-MD5)

But this number decreases if the target domain implements protection.

NTLM authentication

LDAP signing

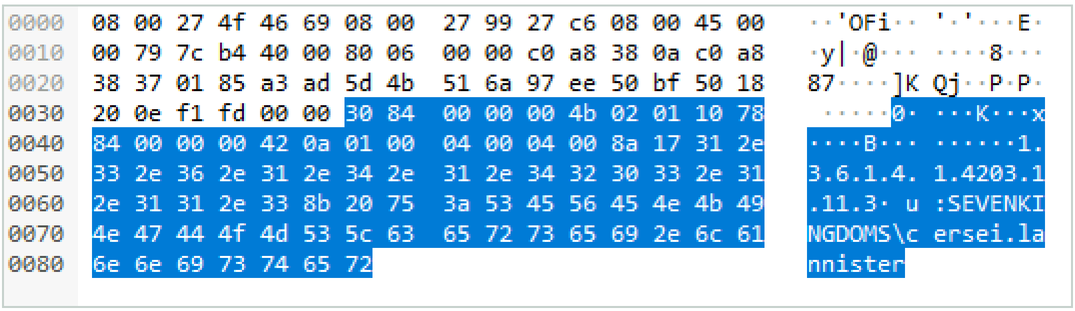
If `Domain controller: LDAP server signing requirements` is set to `Require signing` and a client uses Sicily on LDAP port 389 during the authentication, the following response will be returned by the server:

```
>>> import ldap3
>>> server = ldap3.Server('ldap://192.168.56.10')
>>> connection = ldap3.Connection(server, authentication=ldap3.NTLM,
user='sevenkingdoms.local\\cersei.lannister', password='il0vejaime')
>>> connection.bind()
False
>>> connection.result
{'result': 8, 'description': 'strongerAuthRequired', 'dn': '', 'message':
'00002028: LdapErr: DSID-0C090254, comment: The server requires binds to turn on
integrity checking if SSL\\TLS are not already active on the connection, data 0,
v4f7c\x00', 'referrals': None, 'saslCreds': None, 'type': 'bindResponse'}
```

The error message is straightforward: the authentication scheme does not provide integrity for communication; thus, the authentication is rejected. Fortunately, user CravateRouge submitted a pull request to the ldap3 project that implements message confidentiality as described within MS-NLMP 3.4.3. With this PR, the DC accepts the connection:

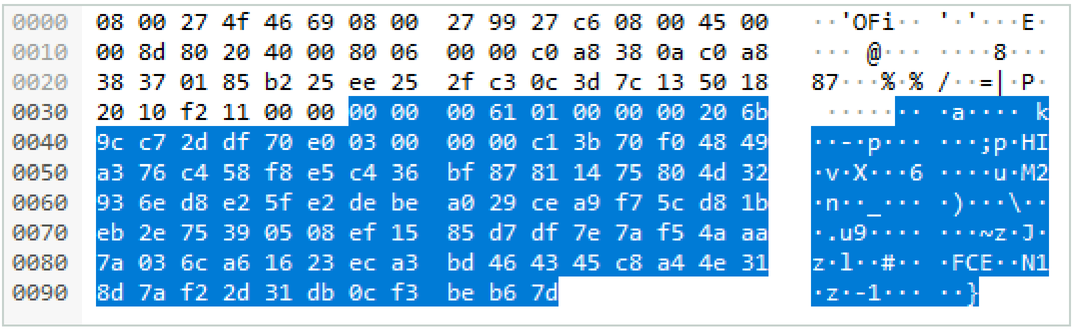
```
>>> import ldap3
>>> server = ldap3.Server('ldap://kingslanding.sevenkingdoms.local')
>>> connection = ldap3.Connection(server, authentication=ldap3.NTLM,
user='sevenkingdoms.local\\cersei.lannister', password='il0vejaime',
session_security=ldap3.ENCRYPT)
>>> connection.bind()
True
>>> connection.result
{'result': 0, 'description': 'success', 'dn': '', 'message': '', 'referrals':
None, 'saslCreds': None, 'type': 'bindResponse'}
```

Here is an example of an LDAP PDU containing a whoami response without confidentiality:



```
0000  08 00 27 4f 46 69 08 00  27 99 27 c6 08 00 45 00   ··'OFi·· '·'···E·
0010  00 79 7c b4 40 00 80 06  00 00 c0 a8 38 0a c0 a8   ·y|·@··· ····8···
0020  38 37 01 85 a3 ad 5d 4b  51 6a 97 ee 50 bf 50 18   87····]K Qj··P·P·
0030  20 0e f1 fd 00 00 30 84  00 00 00 4b 02 01 10 78    ·····0· ···K···x
0040  84 00 00 00 42 0a 01 00  04 00 04 00 8a 17 31 2e   ····B··· ······1.
0050  33 2e 36 2e 31 2e 34 2e  31 2e 34 32 30 33 2e 31   3.6.1.4. 1.4203.1
0060  2e 31 31 2e 33 8b 20 75  3a 53 45 56 45 4e 4b 49   .11.3· u :SEVENKI
0070  4e 47 44 4f 4d 53 5c 63  65 72 73 65 69 2e 6c 61   NGDOMS\c ersei.la
0080  6e 6e 69 73 74 65 72                               nnister
```

The same request with confidentiality enabled:



```
0000  08 00 27 4f 46 69 08 00  27 99 27 c6 08 00 45 00   ··'OFi·· '·'···E·
0010  00 8d 80 20 40 00 80 06  00 00 c0 a8 38 0a c0 a8   ··· @··· ····8···
0020  38 37 01 85 b2 25 ee 25  2f c3 0c 3d 7c 13 50 18   87···%·% /··=|·P·
0030  20 10 f2 11 00 00 00 00  00 61 01 00 00 00 20 6b    ······· ·a···· k
0040  9c c7 2d df 70 e0 03 00  00 00 c1 3b 70 f0 48 49   ··-·p··· ···;p·HI
0050  a3 76 c4 58 f8 e5 c4 36  bf 87 81 14 75 80 4d 32   ·v·X···6 ····u·M2
0060  93 6e d8 e2 5f e2 de be  a0 29 ce a9 f7 5c d8 1b   ·n··_··· ·)···\··
0070  eb 2e 75 39 05 08 ef 15  85 d7 df 7e 7a f5 4a aa   ·.u9···· ···~z·J·
0080  7a 03 6c a6 16 23 ec a3  bd 46 43 45 c8 a4 4e 31   z·l··#·· ·FCE··N1
0090  8d 7a f2 2d 31 db 0c f3  be b6 7d                  ·z·-1··· ··}
```

The pull request implements encryption with Extended Session Security, so the encrypted LDAP PDU is built like this:

```
[size_of_the_PDU][signature][encrypted_payload]
    4 bytes        16 bytes         X bytes
                  |         |
                  |         |
                  |         |
  ________________|         |__________________
 |                                             |
 |                                             |
 [signature_version][checksum][sequence_number]
      4 bytes          8 bytes      4 bytes
```

Where:

- `signature_version` should always be `1`.
- `checksum` is the 8 first bytes of the resulting MD5: `HMAC_MD5(SIGNING_KEY, CONCAT(SEQUENCE_NUMBER,CLEAR_TEXT_MESSAGE))`. `SIGNING_KEY` is derived from the user's password.
- `sequence_number`: number incrementing every time the client sends a message.
- `encrypted_payload` is the cleartext message encrypted with RC4. The key used by this algorithm is also derived from the user's password.

The previous Wireshark screenshot can be interpreted as:

```
[00000061][01000000][206b9cc72ddf70e0][03000000][c13b.........b67d]
[PDU size][version ][    checksum    ][seq num ][encrypted payload]
```

Moreover, the connection is now protected against network sniffing, thus it is possible to create users or computers within the directory as it would be possible with LDAPS (not possible by default on unencrypted connections):

```
>>> import ldap3, string, random
>>> server = ldap3.Server('ldap://kingslanding.sevenkingdoms.local')
>>> connection = ldap3.Connection(server, authentication=ldap3.NTLM,
user='sevenkingdoms.local\\cersei.lannister', password='il0vejaime',
session_security=ldap3.ENCRYPT)
>>> connection.bind()
True
>>> new_user_dn = 'CN=%s,%s' % ('newuser',
'OU=Crownlands,DC=sevenkingdoms,DC=local') # sorry for the GoT lore...
>>> new_password = ''.join(random.choice(string.ascii_letters + string.digits +
string.punctuation) for _ in range(15))
>>> ucd = {
... 'objectCategory':
'CN=Person,CN=Schema,CN=Configuration,DC=sevenkingdoms,DC=local',
... 'distinguishedName': 'CN=%s,%s' % ('newuser',
'OU=Crownlands,DC=sevenkingdoms,DC=local'),
... 'cn': 'newuser',
... 'sn': 'newuser',
... 'givenName': 'newuser',
... 'displayName': 'newuser',
... 'name': 'newuser',
... 'userAccountControl': 512,
... 'accountExpires': '0',
... 'sAMAccountName': 'newuser',
... 'unicodePwd': '"{}"'.format(new_password).encode('utf-16-le')
... }
>>>
>>> connection.add(new_user_dn, ['top', 'person', 'organizationalPerson', 'user'],
ucd)
True
>>> connection.result
{'result': 0, 'description': 'success', 'dn': '', 'message': '', 'referrals':
None, 'type': 'addResponse'}
```

This result is consistent with the Microsoft documentation:

> To modify this attribute [the password], the client must have a 128-bit Transport Layer Security (TLS)/Secure Socket Layer (SSL) connection to the server. An encrypted session using SSP-created session keys using Windows New Technology LAN Manager (NTLM) or Kerberos are also acceptable as long as the minimum key length is met.

As an alternative, it is also possible to use LDAPS instead of LDAP. Since the TLS layer provides integrity, LDAP signing is not enforced on LDAPS connections.

Channel Binding

If the domain controller forces channel binding by setting the option `LDAP server channel binding token req` to `Always`, the vanilla ldap3 library is not able to authenticate:

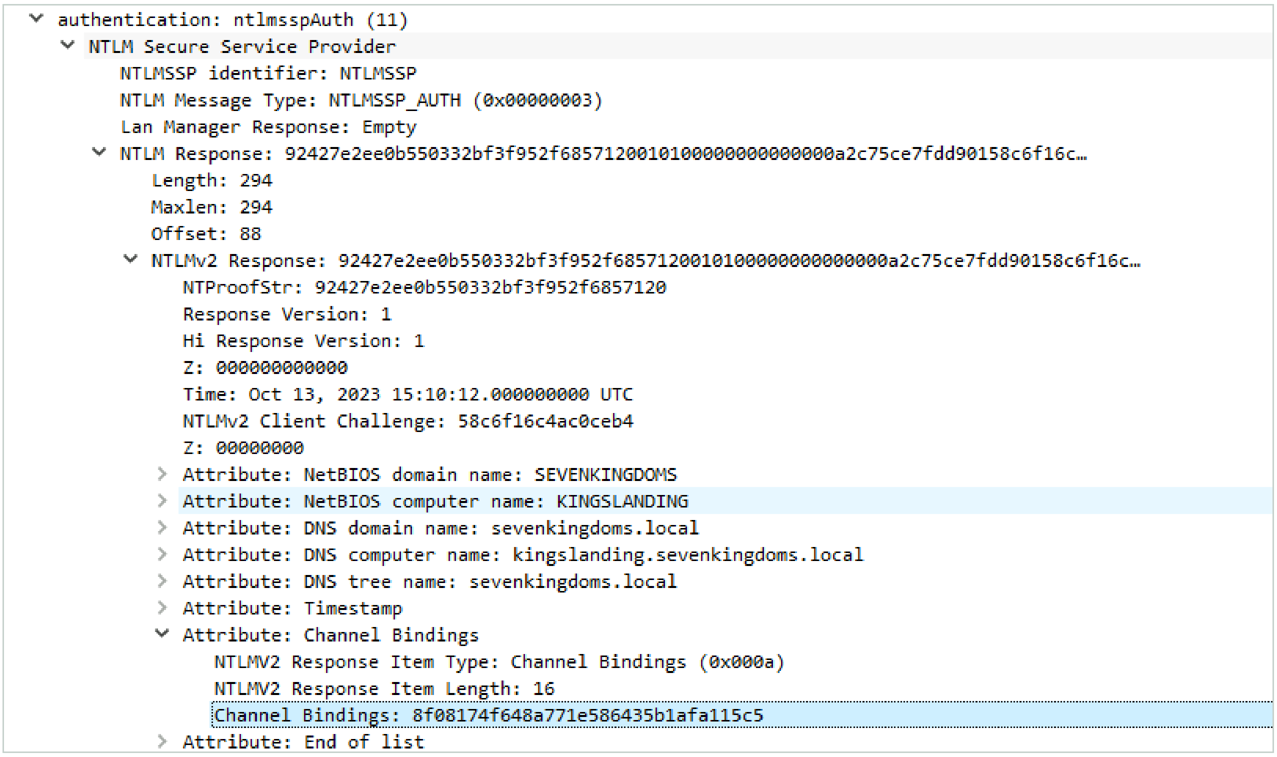
```
>>> import ldap3
>>> server = ldap3.Server('ldaps://kingslanding.sevenkingdoms.local')
>>> connection = ldap3.Connection(server, authentication=ldap3.NTLM,
user='sevenkingdoms.local\\cersei.lannister', password='il0vejaime')
>>> connection.bind()
False
>>> connection.result
{'result': 49, 'description': 'invalidCredentials', 'dn': '', 'message':
'80090346: LdapErr: DSID-0C0906B0, comment: AcceptSecurityContext error, data
80090346, v4f7c\x00', 'referrals': None, 'saslCreds': None, 'type':
'bindResponse'}
```

The server returns an `invalidCredential` error with `AcceptSecurityContext error, data 80090346` comment. This behavior is not well documented, but we can extrapolate that this error code means that the channel binding token is wrong according to Microsoft documentation on Digest authentication (see `SEC_E_BAD_BINDINGS` on this page). Tools such as LdapRelayScan use this technique to determine if channel binding is enforced or not.

Inspired by CravateRouge's work, yours truly submitted a PR to implement channel binding support in ldap3. Thus, the DC now accepts the binding:

```
>>> import ldap3
>>> server = ldap3.Server('ldaps://kingslanding.sevenkingdoms.local')
>>> connection = ldap3.Connection(server, authentication=ldap3.NTLM,
user='sevenkingdoms.local\\cersei.lannister', password='il0vejaime',
channel_binding=ldap3.TLS_CHANNEL_BINDING)
>>> connection.bind()
True
>>> connection.result
{'result': 0, 'description': 'success', 'dn': '', 'message': '', 'referrals':
None, 'saslCreds': None, 'type': 'bindResponse'}
```

Now, a new `AV_PAIR` bears the channel binding token:



```
authentication: ntlmsspAuth (11)
  NTLM Secure Service Provider
    NTLMSSP identifier: NTLMSSP
    NTLM Message Type: NTLMSSP_AUTH (0x00000003)
    Lan Manager Response: Empty
    NTLM Response: 92427e2ee0b550332bf3f952f6857120010100000000000000a2c75ce7fdd90158c6f16c…
      Length: 294
      Maxlen: 294
      Offset: 88
      NTLMv2 Response: 92427e2ee0b550332bf3f952f6857120010100000000000000a2c75ce7fdd90158c6f16c…
        NTProofStr: 92427e2ee0b550332bf3f952f6857120
        Response Version: 1
        Hi Response Version: 1
        Z: 000000000000
        Time: Oct 13, 2023 15:10:12.000000000 UTC
        NTLMv2 Client Challenge: 58c6f16c4ac0ceb4
        Z: 00000000
        Attribute: NetBIOS domain name: SEVENKINGDOMS
        Attribute: NetBIOS computer name: KINGSLANDING
        Attribute: DNS domain name: sevenkingdoms.local
        Attribute: DNS computer name: kingslanding.sevenkingdoms.local
        Attribute: DNS tree name: sevenkingdoms.local
        Attribute: Timestamp
        Attribute: Channel Bindings
          NTLMV2 Response Item Type: Channel Bindings (0x000a)
          NTLMV2 Response Item Length: 16
          Channel Bindings: 8f08174f648a771e586435b1afa115c5
        Attribute: End of list
```

Sending a wrong token raises an authentication error, which shows that the domain controller verifies it on its side too.

Side note: You can easily debug TLS protected connection with ldap3 using the `SSLKEYLOGFILE` environment variable.

```
$ export SSLKEYLOGFILE="sslkey.log"
$ python3 my_custom_ldap3_script.py
```

Then all you need to do is setting the `sslkey.log` file under the Wireshark menu `Edit-> Preferences -> Protocols -> TLS -> (Pre)-Master-Secret log filename`.

Keep in mind that both protections (channel binding and LDAP signing) are checked during the `bind` operation, thus a channel binding bypass can be performed on domain controller with LDAP signing not required.

Side quest: the curious case of NTLM signing

Interestingly, unlike DIGEST-MD5 (see associated section below), it seems that it is not possible to send “signed only” requests when authenticating with Sicily. Indeed, according to Microsoft documentation, NTLM Integrity allows to send a cleartext message and use the signing key to sign it. To enable this feature, the client needs to include the `NTLMSSP_NEGOTIATE_SIGN` flag within the `NEGOTIATE_MESSAGE` message sent by the client.

However, during testing, after sending packets like the following, the domain controller systematically returned errors "Unable to decrypt the payload":

```
[size_of_the_PDU][signature][cleartext_payload]
    4 bytes        16 bytes        X bytes
```

It seems that it tries to decrypt the cleartext payload even if the client only sent the `NTLMSSP_NEGOTIATE_SIGN` flag. It is unclear if "signed only" messages are supported with this authentication method. If you have already done some tests on this topic, feel free to contact us.

Kerberos authentication

All the examples in this section are done after this simple setup:

```
$ # cd into your impacket venv
$ getST.py sevenkingdoms.local/cersei.lannister:'il0vejaime' -dc-ip
kingslanding.sevenkingdoms.local -spn ldap/kingslanding.sevenkingdoms.local
Impacket v0.12.0.dev1+20231015.203043.419e6f24 - Copyright 2023 Fortra

[-] CCache file is not found. Skipping...
[*] Getting TGT for user
[*] Getting ST for user
[*] Saving ticket in cersei.lannister.ccache
$
$ KRB5CCNAME=cersei.lannister.ccache python3
```

ldap3 is based on python-gssapi to handle Kerberos authentication. This library needs the full hostname in the `ccache` file's SPN and the realm needs to be in uppercase, otherwise it throws an error. So, the format needs to be `ldap/kingslanding.sevenkingdoms.local@SEVENKINGDOMS.LOCAL`. For example, `ldap/kingslanding@sevenkingdoms.local` will not work. However, because the SPN is not encrypted, it is possible to change it in order to be accepted by gssapi. This modification can be done with impacket (see impacket's #1256). Because python-gssapi is a front-end to the system package gssapi, using a TGT is also possible with the same limitation about the format (it needs to be `krbtgt/kingslanding.sevenkingdoms.local@SEVENKINGDOMS.LOCAL`).

LDAP signing

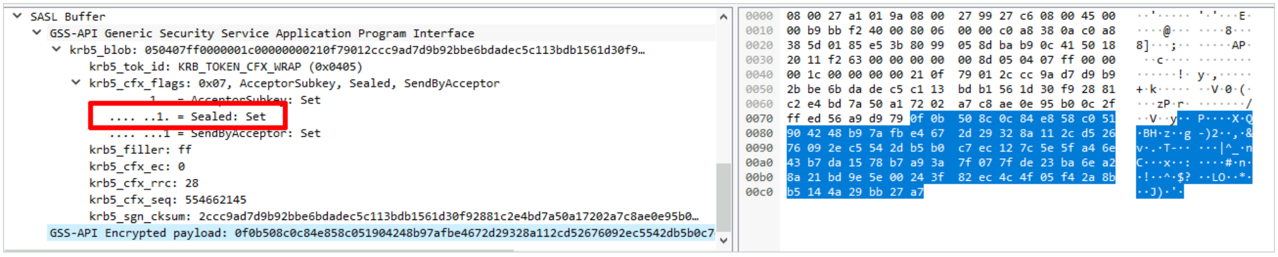
If LDAP signing is enforced by the domain controller, ldap3 is unable to bind against the DC using the LDAP scheme:

```
>>> import ldap3
>>> server = ldap3.Server('ldap://kingslanding.sevenkingdoms.local')
>>> # Remember: realm must be in uppercase
>>> connection = ldap3.Connection(server,
user='cersei.lannister@SEVENKINGDOMS.LOCAL', authentication = ldap3.SASL,
sasl_mechanism=ldap3.KERBEROS)
>>> connection.bind()
False
>>> connection.result
{'result': 8, 'description': 'strongerAuthRequired', 'dn': '', 'message':
'00002028: LdapErr: DSID-0C090259, comment: The server requires binds to turn on
integrity checking if SSL\\TLS are not already active on the connection, data 0,
v4563\x00', 'referrals': None, 'saslCreds': b'', 'type': 'bindResponse'}
```

Once again, pull request #1042 enhances the capabilities of ldap3 by implementing Kerberos payload encryption:

```
>>> import ldap3
>>> server = ldap3.Server('ldap://kingslanding.sevenkingdoms.local')
>>> connection = ldap3.Connection(server,
user='cersei.lannister@SEVENKINGDOMS.LOCAL', authentication = ldap3.SASL,
sasl_mechanism=ldap3.KERBEROS, session_security=ldap3.ENCRYPT)
>>> connection.bind()
True
>>> connection.result
{'result': 0, 'description': 'success', 'dn': '', 'message': '', 'referrals':
None, 'saslCreds': b'', 'type': 'bindResponse'}
>>> # Payloads are now encrypted, it is now possible to create computer or user as
it would be possible with LDAPS.
```

`Sealed` flag is set within the Kerberos context, which means that the payload is now encrypted and signed:



```
SASL Buffer
  GSS-API Generic Security Service Application Program Interface
    krb5_blob: 050407ff0000001c00000000210f79012ccc9ad7d9b92bbe6bdadec5c113bdb1561d30f9…
      krb5_tok_id: KRB_TOKEN_CFX_WRAP (0x0405)
      krb5_cfx_flags: 0x07, AcceptorSubkey, Sealed, SendByAcceptor
        .... .1.. = AcceptorSubkey: Set
        .... ..1. = Sealed: Set
        .... ...1 = SendByAcceptor: Set
      krb5_filler: ff
      krb5_cfx_ec: 0
      krb5_cfx_rrc: 28
      krb5_cfx_seq: 554662145
      krb5_sgn_cksum: 2ccc9ad7d9b92bbe6bdadec5c113bdb1561d30f92881c2e4bd7a50a17202a7c8ae0e95b0…
  GSS-API Encrypted payload: 0f0b508c0c84e858c051904248b97afbe4672d29328a112cd52676092ec5542db5b0c7
```

```
0000  08 00 27 a1 01 9a 08 00  27 99 27 c6 08 00 45 00   ··'·····'·'···E·
0010  00 b9 bb f2 40 00 80 06  00 00 c0 a8 38 0a c0 a8   ····@·······8···
0020  38 5d 01 85 e5 3b 80 99  05 8d ba b9 0c 41 50 18   8]···;·······AP·
0030  20 11 f2 63 00 00 00 00  00 8d 05 04 07 ff 00 00    ··c············
0040  00 1c 00 00 00 00 21 0f  79 01 2c cc 9a d7 d9 b9   ······!·y·,·····
0050  2b be 6b da de c5 c1 13  bd b1 56 1d 30 f9 28 81   +·k·······V·0·(·
0060  c2 e4 bd 7a 50 a1 72 02  a7 c8 ae 0e 95 b0 0c 2f   ···zP·r········/
0070  ff ed 56 a9 d9 79 0f 0b  50 8c 0c 84 e8 58 c0 51   ··V··y··P····X·Q
0080  90 42 48 b9 7a fb e4 67  2d 29 32 8a 11 2c d5 26   ·BH·z··g-)2··,·&
0090  76 09 2e c5 54 2d b5 b0  c7 ec 12 7c 5e 5f a4 6e   v·.·T-·····|^_·n
00a0  43 b7 da 15 78 b7 a9 3a  7f 07 7f de 23 ba 6e a2   C···x··:····#·n·
00b0  8a 21 bd 9e 5e 00 24 3f  82 ec 4c 4f 05 f4 2a 8b   ·!··^·$?··LO··*·
00c0  b5 14 4a 29 bb 27 a7                               ··J)·'·
```

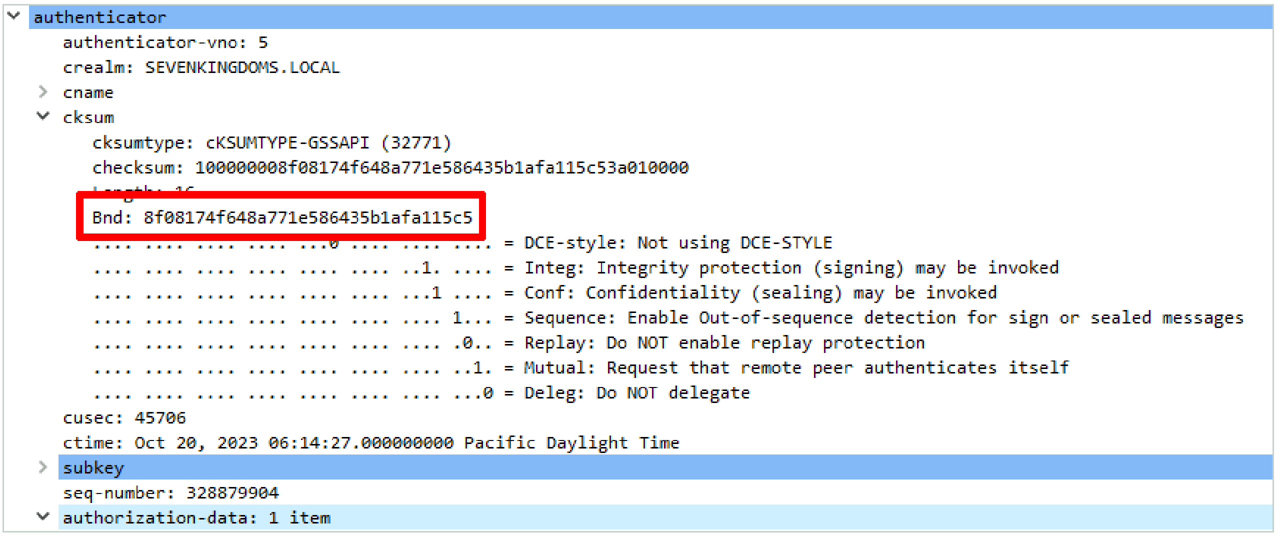
The symmetric key used to encrypt those exchanges is sent by the server in the `authenticator` part of the `AP-REP` (keytype 18 usage 22 and 24).

Channel Binding

Regarding channel binding support, python-gssapi is compatible with it, so it is possible to authenticate ldap3 against domain controller with this protection.

```
>>> import ldap3
>>> server = ldap3.Server('ldaps://kingslanding.sevenkingdoms.local')
>>> connection = ldap3.Connection(server,
user='cersei.lannister@SEVENKINGDOMS.LOCAL', authentication = ldap3.SASL,
sasl_mechanism=ldap3.KERBEROS)
>>> connection.bind()
True
>>> connection.result
{'result': 0, 'description': 'success', 'dn': '', 'message': '', 'referrals':
None, 'saslCreds': b'', 'type': 'bindResponse'}
```

The channel binding token is sent within the `authenticator` in the `AP-REQ` Kerberos message:



```
authenticator
    authenticator-vno: 5
    crealm: SEVENKINGDOMS.LOCAL
  > cname
  v cksum
        cksumtype: cKSUMTYPE-GSSAPI (32771)
        checksum: 100000008f08174f648a771e586435b1afa115c53a010000
        Length: 16
        Bnd: 8f08174f648a771e586435b1afa115c5
        .... .... .... .... ...0 .... .... .... = DCE-style: Not using DCE-STYLE
        .... .... .... .... .... .... ..1. .... = Integ: Integrity protection (signing) may be invoked
        .... .... .... .... .... .... ...1 .... = Conf: Confidentiality (sealing) may be invoked
        .... .... .... .... .... .... .... 1... = Sequence: Enable Out-of-sequence detection for sign or sealed messages
        .... .... .... .... .... .... .... .0.. = Replay: Do NOT enable replay protection
        .... .... .... .... .... .... .... ..1. = Mutual: Request that remote peer authenticates itself
        .... .... .... .... .... .... .... ...0 = Deleg: Do NOT delegate
    cusec: 45706
    ctime: Oct 20, 2023 06:14:27.000000000 Pacific Daylight Time
  > subkey
    seq-number: 328879904
  v authorization-data: 1 item
```

During our tests, we were not able to send a crafted channel binding token without patching the `libkrb5-dev` low level package. Thus, it is unclear if the domain controller checks it (more on this in the GSS-SPNEGO section).

Side note: You can debug Kerberos protected connections with Wireshark as explained here.

Simple authentication

Simple authentication is a classic user/password authentication, so it is not "vulnerable" to relay attack: if an attacker manages to trick the victim to authenticate against their machine, they don't need to relay the authentication as they got the cleartext password. This authentication method is not compatible with LDAP signing, a `strongerAuthRequired` error is returned by the server if the client tries to authenticate. However, it turns out that simple authentication is not impacted by channel binding enforcement. Thus, if the attacker has a valid account, its corresponding password (it will not work with the NT hash, so pth attack is not possible), and the DC exposes an LDAPS service, they can use it to query the DC.

```
>>> server = ldap3.Server('ldaps://192.168.56.10')
>>> connection = ldap3.Connection(server, authentication=ldap3.SIMPLE,
user='cersei.lannister@sevenkingdoms.local', password='il0vejaime')
>>> connection.bind()
True
>>> connection.result
{'result': 0, 'description': 'success', 'dn': '', 'message': '', 'referrals':
None, 'saslCreds': None, 'type': 'bindResponse'}
```

DIGEST-MD5 authentication

DIGEST-MD5 is an obsolete authentication mechanism, but it is still supported by Active Directory. It is a challenge/response authentication protocol, so it works with a shared secret between the client and the server. According to the RFC, the shared secret is based on the following elements:

```
a0 = MD5(username-value + ":" + realm-value + ":" + password)
```

However, as Active Directory does not store the password in a reversible form (unless this behavior is explicitly enabled through GPO/registry), we can ask ourselves how it is possible to compute `a0` without knowing the cleartext password.

```
$ echo -n 'cersei.lannister:sevenkingdoms.local:il0vejaime' | md5sum
532062350b784ce18e7680a2447f6fa1  -
$ echo -n 'CERSEI.LANNISTER:SEVENKINGDOMS.LOCAL:il0vejaime' | md5sum
f39ea5d591925fa63047235f63927a71  -
```

Turns out that Active Directory stores 29 MD5 hashes which represent various formats of this information. DSIntenals can retrieve those hashes:

```
PS C:\Windows\system32> Get-ADReplAccount -SamAccountName cersei.lannister -Server
sevenkingdoms.local

DistinguishedName: CN=cersei.lannister,OU=Crownlands,DC=sevenkingdoms,DC=local
Sid: S-1-5-21-3990748329-3747950281-3952183803-1115
Guid: 2090a2ab-aeb5-4770-8b9c-de268cb2fc7b
SamAccountName: cersei.lannister
[...]
Owner: S-1-5-21-3990748329-3747950281-3952183803-512
Secrets
  NTHash: c247f62516b53893c7addcf8c349954b
[...]
  SupplementalCredentials:
    [...]
    WDigest:
      Hash 01: ae9746cdc9ab4c98389c931240bfef49
      Hash 02: d134ad87efe27cb920f7261c716e071e
      Hash 03: 2cf4c83f64539f3379e14bbc6bb62f7a
      Hash 04: ae9746cdc9ab4c98389c931240bfef49
      Hash 05: d134ad87efe27cb920f7261c716e071e
      Hash 06: ee1f53e170afc2923c76ca1608e24a7c
      Hash 07: ae9746cdc9ab4c98389c931240bfef49
      Hash 08: 532062350b784ce18e7680a2447f6fa1
      Hash 09: 532062350b784ce18e7680a2447f6fa1
      Hash 10: f39ea5d591925fa63047235f63927a71
      Hash 11: 637e50611556b02b161e139f6721df9a
      Hash 12: 532062350b784ce18e7680a2447f6fa1
      Hash 13: a89bf491596edce803f45c0fe8b85c2d
      Hash 14: 637e50611556b02b161e139f6721df9a
      Hash 15: 6228c58f5f6bb52c8d292aedc6747ce2
      Hash 16: 6228c58f5f6bb52c8d292aedc6747ce2
      Hash 17: 633a4bbaf44681cc7c476633044f965a
      Hash 18: f2df7fedf3c221c684f98119dd1d143b
      Hash 19: 342b3287424f1021b1efc474694b69c4
      Hash 20: d386808d89c8360df3746b66ee3e3b46
      Hash 21: 8cf3aecb2eae565c0e91088f1168a753
      Hash 22: 8cf3aecb2eae565c0e91088f1168a753
      Hash 23: e80735b18be0aeec5ed4cd330ac9e1d3
      Hash 24: 894cc33e043d29a258037531fb1af135
      Hash 25: 894cc33e043d29a258037531fb1af135
      Hash 26: d1f74dbf597d85638150322df881678b
      Hash 27: 3214f15a66cb313eb40d94582caaacd4
      Hash 28: 51fdf783ee508745d0472b36d6e89033
      Hash 29: 4f3bba9380e6e3ad7297608c91fa99dc
[...]
```

LDAP signing

According to the RFC, DIGEST-MD5 can negotiate signing and encryption during the authentication phase. Active Directory also supports encryption and signature with DIGEST-MD5: when initiating a SASL DIGEST-MD5 authentication the server returns `qop="auth,auth-int,auth-conf"` which means:

> *qop-options*: A quoted string of one or more tokens indicating the "quality of protection" values supported by the server. The value "auth" indicates authentication; the value "auth-int" indicates authentication with integrity protection; the value "auth-conf" indicates authentication with integrity protection and encryption.

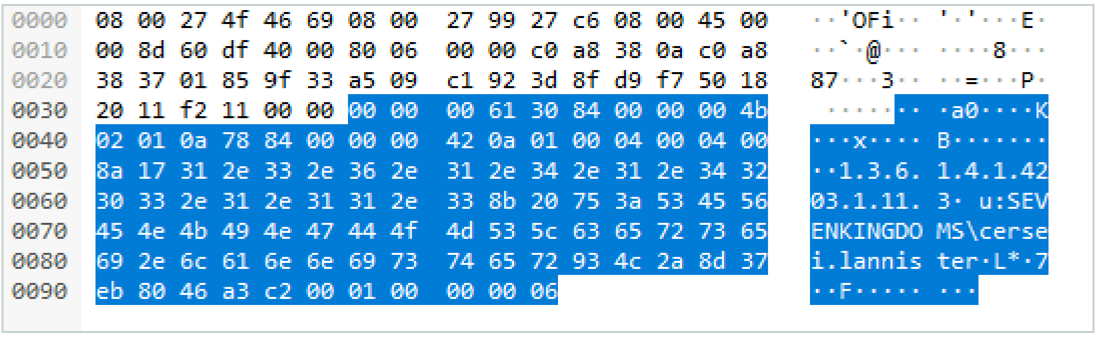
Vanilla ldap3 supports signature with DIGEST-MD5 and can be used to circumvent the lack of NTLM signing support.

```
>>> import ldap3
>>> server = ldap3.Server('ldap://kingslanding.sevenkingdoms.local') # You cannot
use an IP address here, must be a FQDN with a LDAP SPN registred
>>> connection = ldap3.Connection(server, authentication = ldap3.SASL,
sasl_mechanism = ldap3.DIGEST_MD5, sasl_credentials = (None, 'cersei.lannister',
'il0vejaime', None, None))
>>> connection.bind()
False
>>> connection.result
{'result': 8, 'description': 'strongerAuthRequired', 'dn': '', 'message':
'00002028: LdapErr: DSID-0C090254, comment: The server requires binds to turn on
integrity checking  if SSL\\TLS are not already active on the connection, data 0,
v4f7c\x00', 'referrals': None, 'saslCreds':
b'rspauth=5f33da03b10aa1b509ea9d71758279b5', 'type': 'bindResponse'}
>>> # LDAP signing is enabled on the server and ldap3's DIGEST-MD5 implementation
supports it
>>> connection = ldap3.Connection(server, authentication = ldap3.SASL,
sasl_mechanism = ldap3.DIGEST_MD5, sasl_credentials = (None, 'cersei.lannister',
'il0vejaime', None, 'sign'))
>>> connection.bind()
True
>>> connection.result
{'result': 0, 'description': 'success', 'dn': '', 'message': '', 'referrals':
None, 'saslCreds': b'rspauth=f71443de6af69750719e03445bf9ff03', 'type':
'bindResponse'}
```

Pull request #1042 also implements encryption for DIGEST-MD5.

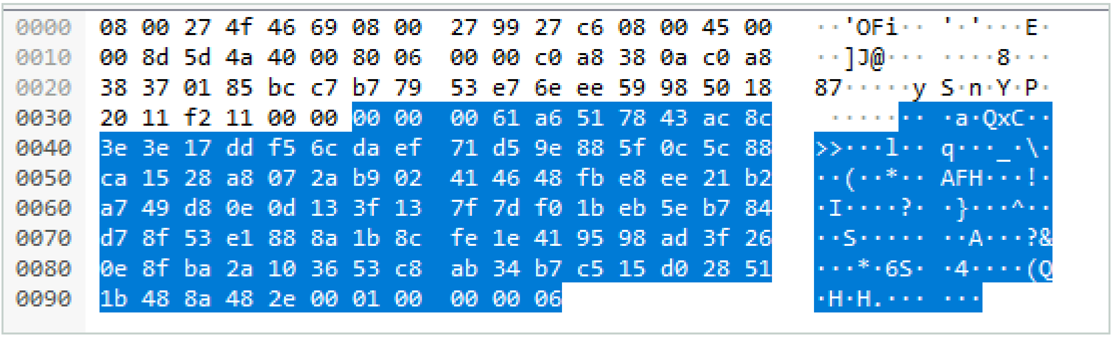
```
>>> import ldap3
>>> server = ldap3.Server('ldap://kingslanding.sevenkingdoms.local')
>>> connection = ldap3.Connection(server, authentication = ldap3.SASL,
sasl_mechanism = ldap3.DIGEST_MD5, sasl_credentials = (None, 'cersei.lannister',
'il0vejaime', None, 'ENCRYPT'))
>>> connection.bind()
True
>>> connection.result
{'result': 0, 'description': 'success', 'dn': '', 'message': '', 'referrals':
None, 'saslCreds': b'rspauth=7a9f565a4f9bf185c9eb7927bf6621ed', 'type':
'bindResponse'}
>>> # Payloads are now encrypted, it is now possible to create computer or user as
it would be possible with LDAPS.
```

Below, a signed `whoami` response:



```
0000  08 00 27 4f 46 69 08 00  27 99 27 c6 08 00 45 00   ··'OFi·· '·'···E·
0010  00 8d 60 df 40 00 80 06  00 00 c0 a8 38 0a c0 a8   ··`·@··· ····8···
0020  38 37 01 85 9f 33 a5 09  c1 92 3d 8f d9 f7 50 18   87···3·· ··=···P·
0030  20 11 f2 11 00 00 00 00  00 61 30 84 00 00 00 4b   ········ ·a0····K
0040  02 01 0a 78 84 00 00 00  42 0a 01 00 04 00 04 00   ···x···· B·······
0050  8a 17 31 2e 33 2e 36 2e  31 2e 34 2e 31 2e 34 32   ··1.3.6. 1.4.1.42
0060  30 33 2e 31 2e 31 31 2e  33 8b 20 75 3a 53 45 56   03.1.11. 3· u:SEV
0070  45 4e 4b 49 4e 47 44 4f  4d 53 5c 63 65 72 73 65   ENKINGDO MS\cerse
0080  69 2e 6c 61 6e 6e 69 73  74 65 72 93 4c 2a 8d 37   i.lannis ter·L*·7
0090  eb 80 46 a3 c2 00 01 00  00 00 06                  ··F····· ···
```

The same response encrypted:



```
0000  08 00 27 4f 46 69 08 00  27 99 27 c6 08 00 45 00   ··'OFi·· '·'···E·
0010  00 8d 5d 4a 40 00 80 06  00 00 c0 a8 38 0a c0 a8   ··]J@··· ····8···
0020  38 37 01 85 bc c7 b7 79  53 e7 6e ee 59 98 50 18   87·····y S·n·Y·P·
0030  20 11 f2 11 00 00 00 00  00 61 a6 51 78 43 ac 8c   ········ ·a·QxC··
0040  3e 3e 17 dd f5 6c da ef  71 d5 9e 88 5f 0c 5c 88   >>···l·· q···_·\·
0050  ca 15 28 a8 07 2a b9 02  41 46 48 fb e8 ee 21 b2   ··(··*·· AFH···!·
0060  a7 49 d8 0e 0d 13 3f 13  7f 7d f0 1b eb 5e b7 84   ·I····?· ·}···^··
0070  d7 8f 53 e1 88 8a 1b 8c  fe 1e 41 95 98 ad 3f 26   ··S····· ··A···?&
0080  0e 8f ba 2a 10 36 53 c8  ab 34 b7 c5 15 d0 28 51   ···*·6S· ·4····(Q
0090  1b 48 8a 48 2e 00 01 00  00 00 06                  ·H·H.··· ···
```

The format of the LDAP PDUs with DIGEST-MD5 confidentiality/integrity is similar to those with NTLM confidentiality:

```
[size_of_the_PDU][encrypted or cleartext payload][checksum][signature_version]
[sequence_number]
      4 bytes                   X bytes            10 bytes       2 bytes
4 bytes
```

Channel Binding

The drawback of using DIGEST-MD5 is the lack of channel binding support (even if an IETF draft exists):

```
>>> import ldap3
>>> server = ldap3.Server('ldaps://kingslanding.sevenkingdoms.local')
>>> connection = ldap3.Connection(server, authentication = ldap3.SASL,
sasl_mechanism = ldap3.DIGEST_MD5, sasl_credentials = (None, 'cersei.lannister',
'il0vejaime', None, None))
>>> connection.bind()
False
>>> connection.result
{'result': 49, 'description': 'invalidCredentials', 'dn': '', 'message':
'80090346: LdapErr: DSID-0C09058A, comment: AcceptSecurityContext error, data
80090346, v4563\x00', 'referrals': None, 'saslCreds': None, 'type':
'bindResponse'}
```

Schannel authentication

Another interesting mode is Schannel: it relies on TLS so it is, by design, not subject to channel binding, as the authentication is borne by TLS itself.

```
>>> import ldap3, ssl
>>> tls = ldap3.Tls(local_private_key_file='jaime.key',
local_certificate_file='jaime.crt', validate=ssl.CERT_NONE)
>>> server = ldap3.Server('ldaps://kingslanding.sevenkingdoms.local', tls=tls)
>>> connection = ldap3.Connection(server)
>>> connection.open()
>>> print(connection.extend.standard.who_am_i())
u:SEVENKINGDOMS\jaime.lannister
```

Schannel is not subject to LDAP signing either as the `bind` is performed after a StartTLS command when used on the LDAP TCP port.

```
>>> import ldap3, ssl
>>> tls = ldap3.Tls(local_private_key_file='jaime.key',
local_certificate_file='jaime.crt', validate=ssl.CERT_NONE)
>>> server = ldap3.Server('ldap://kingslanding.sevenkingdoms.local', tls=tls)
>>> connection = ldap3.Connection(server, authentication=ldap3.SASL,
sasl_mechanism=ldap3.EXTERNAL)
>>> connection.open()
>>> connection.start_tls()
True
>>> connection.bind()
True
>>> connection.result
{'result': 0, 'description': 'success', 'dn': '', 'message': '', 'referrals':
None, 'saslCreds': None, 'type': 'bindResponse'}
```

PassTheCert and Certipy use this authentication mechanism to perform various attacks.

impacket.ldap

The impacket library also implements the LDAP RFC, however, its implementation is minimalistic and not as powerful as ldap3 (developers urge users to use third party libraries for more complex use cases).

Based on our experience, the main limitations of `impacket.ldap` are related to bad Unicode handling ("Utilisateurs du Bureau **à** distance" 🥖🇫🇷) and bugs when using complex search filters containing special characters (if your DN is like `CN=John Doe (admin account)` for example). However, it is still used by some of impacket' scripts and according to the code, `impacket.ldap` supports 4 (sub) authentication methods:

- Simple
- Sicily
- SASL (GSS-SPNEGO NTLM)
- SASL (GSS-SPNEGO Kerberos)

Impacket's Sicily implementation does not support LDAP signing nor channel binding.

GSS-SPNEGO NTLM

To sum up, GSS-SPNEGO NTLM encapsulates the NTLM protocol within a SASL structure, so it is pretty similar to Sicily authentication. As confidentiality, integrity and channel binding are not implemented within `impacket.ldap`, this library will not work against hardened domain controllers. However, it seems easy to implement them as impacket implements a lot of NTLM features.

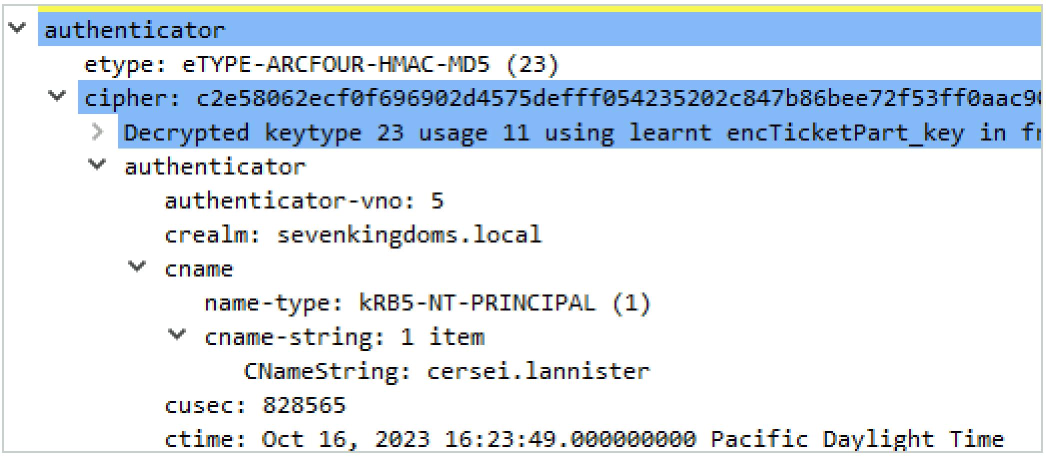
```
>>> from impacket.ldap import ldap, ldapasn1
>>> ldap_connection =
ldap.LDAPConnection('ldap://kingslanding.sevenkingdoms.local',
'dc=sevenkingdoms,dc=local', '192.168.56.10')
>>> ldap_connection.login('cersei.lannister', 'il0vejaime', 'sevenkingdoms.local',
'', '', authenticationChoice="sasl")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/user/impacket/impacket/ldap/ldap.py", line 369, in login
    raise LDAPSessionError(
impacket.ldap.ldap.LDAPSessionError: Error in bindRequest -> strongerAuthRequired:
00002028: LdapErr: DSID-0C090259, comment: The server requires binds to turn on
integrity checking if SSL\TLS are not already active on the connection, data 0,
v4563
>>>
>>> # impacket.ldap does not implement NTLM confidentiality, now let's try with
channel binding
>>>
>>> ldap_connection =
ldap.LDAPConnection('ldaps://kingslanding.sevenkingdoms.local',
'dc=sevenkingdoms,dc=local', '192.168.56.10')
>>> ldap_connection.login('cersei.lannister', 'il0vejaime', 'sevenkingdoms.local',
'', '', authenticationChoice="sasl")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/user/impacket/impacket/ldap/ldap.py", line 369, in login
    raise LDAPSessionError(
impacket.ldap.ldap.LDAPSessionError: Error in bindRequest -> invalidCredentials:
80090346: LdapErr: DSID-0C09058A, comment: AcceptSecurityContext error, data
80090346, v4563
```

GSS-SPNEGO Kerberos

As mentioned before, it is not clear if Kerberos authentication is impacted by channel binding with GSSAPI. However, we can state that, it is possible to query a hardened domain controller with impacket's ldap even if channel binding is enforced on LDAPS:

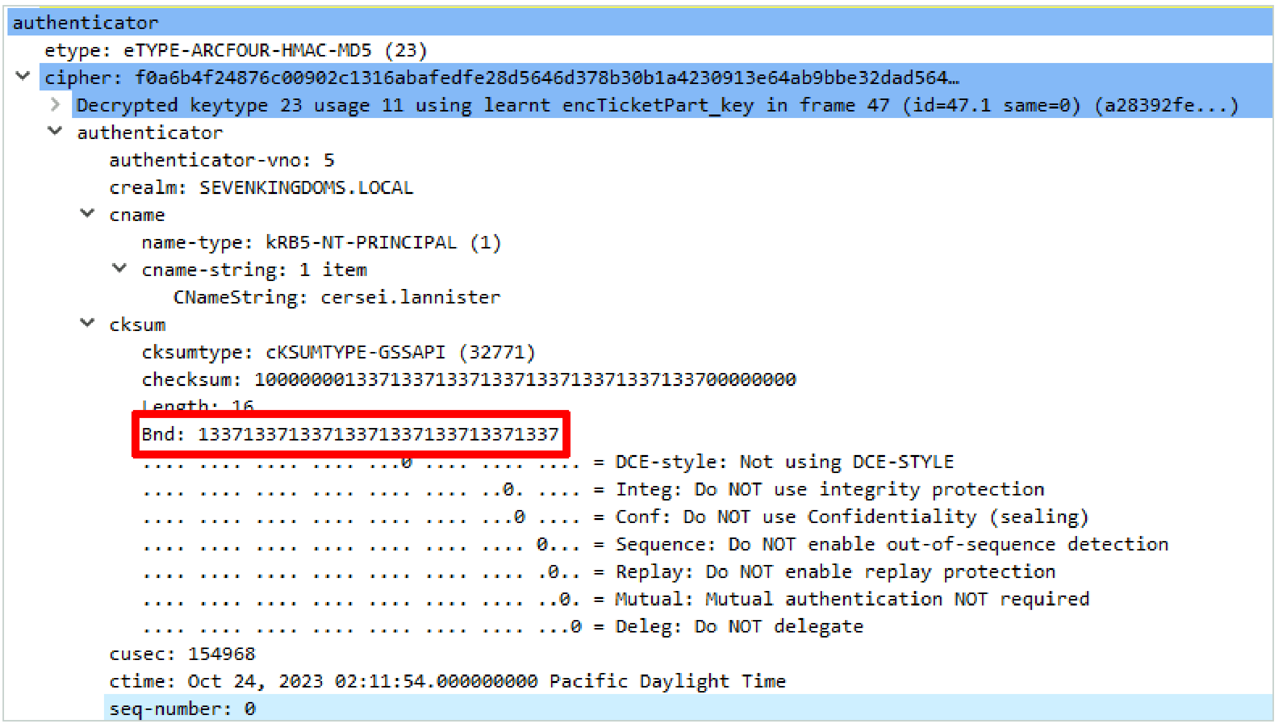
```
>>> from impacket.ldap import ldap, ldapasn1
>>> ldap_connection =
ldap.LDAPConnection('ldap://kingslanding.sevenkingdoms.local',
'dc=sevenkingdoms,dc=local', '192.168.56.10')
>>> ldap_connection.kerberosLogin('cersei.lannister', 'il0vejaime',
'sevenkingdoms.local', '', '')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/user/impacket/impacket/ldap/ldap.py", line 252, in kerberosLogin
    raise LDAPSessionError(
impacket.ldap.ldap.LDAPSessionError: Error in bindRequest -> strongerAuthRequired:
00002028: LdapErr: DSID-0C090259, comment: The server requires binds to turn on
integrity checking if SSL\TLS are not already active on the connection, data 0,
v4563
>>>
>>> # impacket.ldap does not implement Kerberos confidentiality, now let's try
with channel binding
>>>
>>> dap_connection =
ldap.LDAPConnection('ldaps://kingslanding.sevenkingdoms.local',
'dc=sevenkingdoms,dc=local', '192.168.56.10')
>>> ldap_connection.kerberosLogin('cersei.lannister', 'il0vejaime',
'sevenkingdoms.local', '', '')
True
>>> attributes=list()
>>> paged_search_control =
ldapasn1.SimplePagedResultsControl(criticality=True,size=10)
>>> search_filter = '(&(objectCategory=group)(name=Domain Admins))'
>>> search_results =
ldap_connection.search(searchFilter=search_filter,searchControls=
[paged_search_control],attributes=attributes)
```

The channel binding token is not present within the `authenticator` part of the `AP-REQ`, however the bind request is accepted:



```
authenticator
    etype: eTYPE-ARCFOUR-HMAC-MD5 (23)
    cipher: c2e58062ecf0f696902d4575defff054235202c847b86bee72f53ff0aac9
        Decrypted keytype 23 usage 11 using learnt encTicketPart_key in fr
        authenticator
            authenticator-vno: 5
            crealm: sevenkingdoms.local
            cname
                name-type: kRB5-NT-PRINCIPAL (1)
                cname-string: 1 item
                    CNameString: cersei.lannister
            cusec: 828565
            ctime: Oct 16, 2023 16:23:49.000000000 Pacific Daylight Time
```

If the token sent by the client is wrong, the bind request fails with the error code `AcceptSecurityContext error, data 80090346`:



```
authenticator
    etype: eTYPE-ARCFOUR-HMAC-MD5 (23)
  ˅ cipher: f0a6b4f24876c00902c1316abafedfe28d5646d378b30b1a4230913e64ab9bbe32dad564…
    › Decrypted keytype 23 usage 11 using learnt encTicketPart_key in frame 47 (id=47.1 same=0) (a28392fe...)
    ˅ authenticator
        authenticator-vno: 5
        crealm: SEVENKINGDOMS.LOCAL
      ˅ cname
          name-type: kRB5-NT-PRINCIPAL (1)
        ˅ cname-string: 1 item
            CNameString: cersei.lannister
      ˅ cksum
          cksumtype: cKSUMTYPE-GSSAPI (32771)
          checksum: 10000000133713371337133713371337133713371337000000000
          Length: 16
          Bnd: 13371337133713371337133713371337
          .... .... .... .... ...0 .... .... .... = DCE-style: Not using DCE-STYLE
          .... .... .... .... .... .... ..0. .... = Integ: Do NOT use integrity protection
          .... .... .... .... .... .... ...0 .... = Conf: Do NOT use Confidentiality (sealing)
          .... .... .... .... .... .... .... 0... = Sequence: Do NOT enable out-of-sequence detection
          .... .... .... .... .... .... .... .0.. = Replay: Do NOT enable replay protection
          .... .... .... .... .... .... .... ..0. = Mutual: Mutual authentication NOT required
          .... .... .... .... .... .... .... ...0 = Deleg: Do NOT delegate
        cusec: 154968
        ctime: Oct 24, 2023 02:11:54.000000000 Pacific Daylight Time
        seq-number: 0
```

Summary

	LDAP	LDAPS	LDAP + Signing	LDAPS + Channel Binding
NTLM (ldap3 vanilla)	✓	✓	❌	❌
Kerberos (ldap3 vanilla)	✓	✓	❌	✓
Schannel (ldap3 vanilla)	✓ (*)	✓	N/A	N/A
Simple Authentication (ldap3 vanilla)	✓	✓	❌	✓
DIGEST-MD5 (ldap3 vanilla)	✓	✓	✓ (**)	❌
NTLM (ldap3 patched)	✓	✓	✓	✓
Kerberos (ldap3 patched)	✓	✓	✓	✓
GSS-SPNEGO NTLM (impacket.ldap)	✓	✓	❌	❌
GSS-SPNEGO Kerberos (impacket.ldap)	✓	✓	❌	✓

* Using StartTLS

** Using the option `sign`

Conclusion

This brief overview of LDAP authentication protocols usable in Active Directory environments shows that client-side support can vary widely depending on the implementation, but can often be "hacked in" when needed. Since a lot of impacket's examples are based on ldap3, it seems easy to adapt them to work against hardened domain controllers by installing patches (as shown here by snovvcrash). A fork has been created with the 2 PRs to easily use them in tools. Certipy has recently started supporting channel binding and pywerview implements a logic to automatically detect and handle LDAP protections for you (full disclosure: the author is a pywerview maintainer). If patching ldap3 library is not an option, it is still possible to directly patch the tools to circumvent the problem (see here, and here). Lastly, an even more radical approach can be used: tools such as msldap have chosen to reimplement the authentication stack, via asysocks, asyauth or minikerberos.

Credits

The author would like to thank:

- CravateRouge for kickstarting the idea of this blogpost with their pull request.
- Giovanni Cannata and all the contributors to the ldap3 project.
- Forta, SecureAuth, Alberto Solino and all the contributors to the impacket project.
- SkelSec for their tools, sometimes better than the official Microsoft documentation.
- Mayfly / Orange-CyberDefense for the GOAD project, used in the examples.