LDAPFragger: Command and Control over LDAP attributes

March 19, 2020

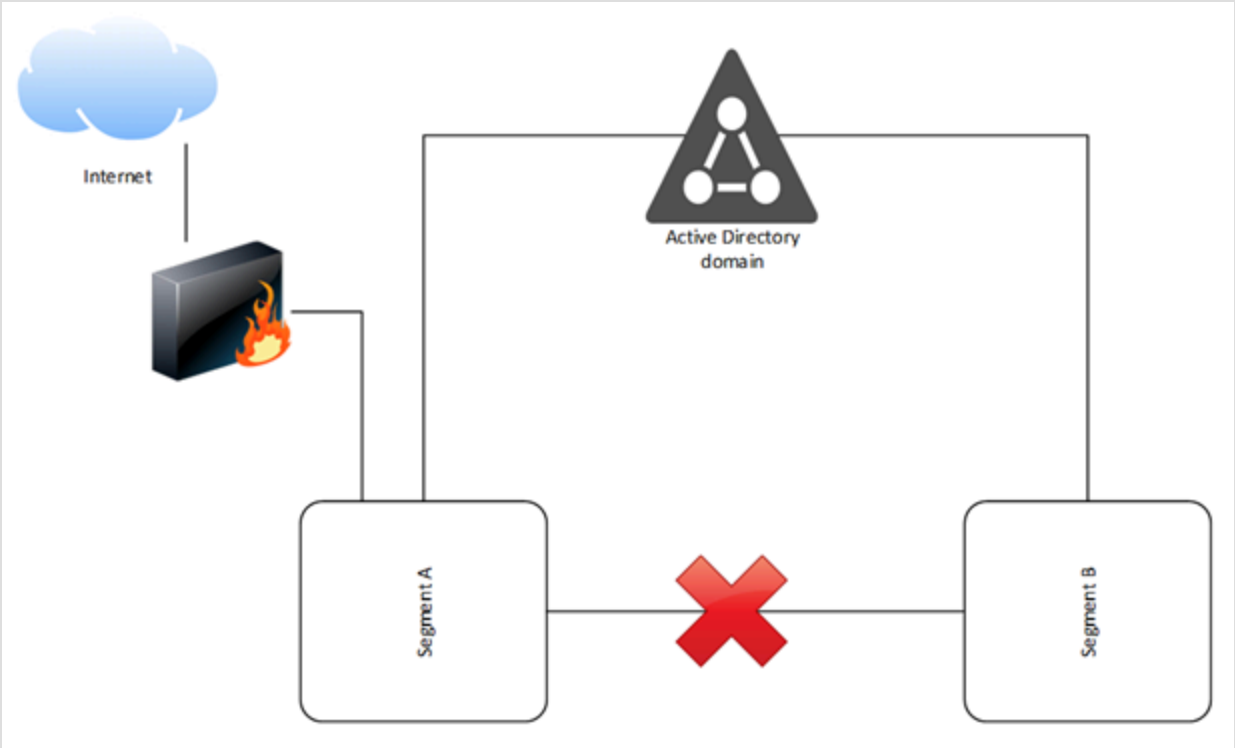
Written by Rindert Kramer

Introduction

A while back during a penetration test of an internal network, we encountered physically segmented networks. These networks contained workstations joined to the same Active Directory domain, however only one network segment could connect to the internet. To control workstations in both segments remotely with Cobalt Strike, we built a tool that uses the shared Active Directory component to build a communication channel. For this, it uses the LDAP protocol which is commonly used to manage Active Directory, effectively routing beacon data over LDAP. This blogpost will go into detail about the development process, how the tool works and provides mitigation advice.

Scenario

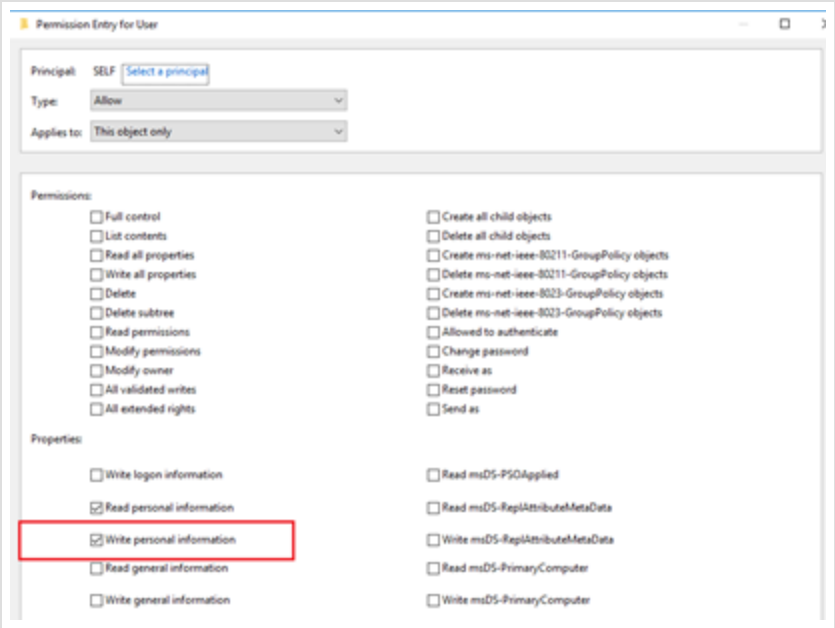
A couple of months ago, we did a network penetration test at one of our clients. This client had multiple networks that were completely firewalled, so there was no direct connection possible between these network segments. Because of cost/workload efficiency reasons, the client chose to use the same Active Directory domain between those network segments. This is what it looked like from a high-level overview.



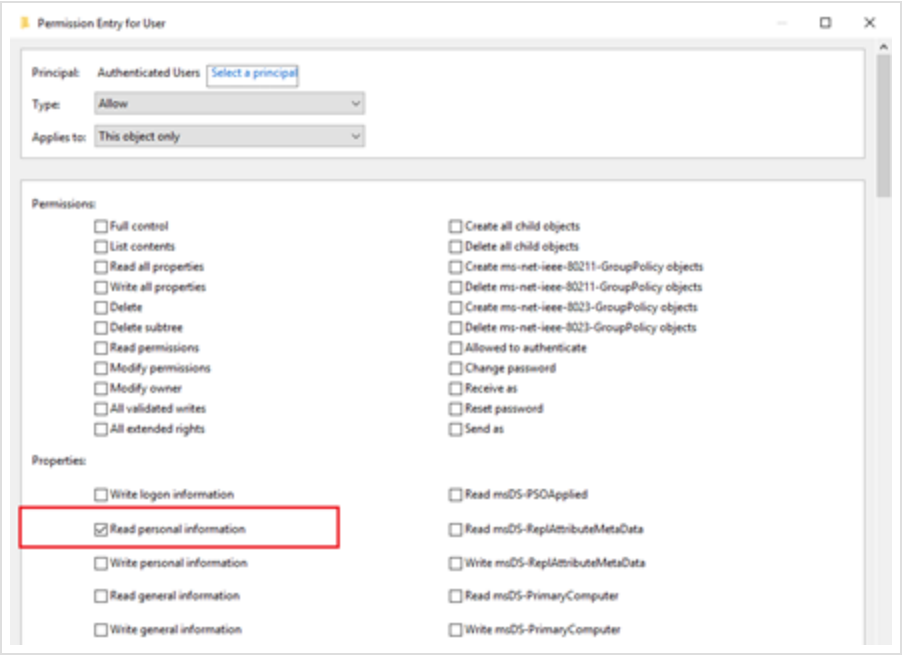
We had physical access on workstations in both segment A and segment B. In this example, workstations in segment A were able to reach the internet, while workstations in segment B could not. While we did have physical access on workstation in both network segments, we wanted to control workstations in network segment B from the internet.

Active Directory as a shared component

Both network segments were able to connect to domain controllers in the same domain and could interact with objects, authenticate users, query information and more. In Active Directory, user accounts are objects to which extra information can be added. This information is stored in attributes. By default, user accounts have write permissions on some of these attributes. For example, users can update personal information such as telephone numbers or office locations for their own account. No special privileges are needed for this, since this information is writable for the identity `SELF`, which is the account itself. This is configured in the Active Directory schema, as can be seen in the screenshot below.



Personal information, such as a telephone number or street address, is by default readable for every authenticated user in the forest. Below is a screenshot that displays the permissions for public information for the `Authenticated Users` identity.

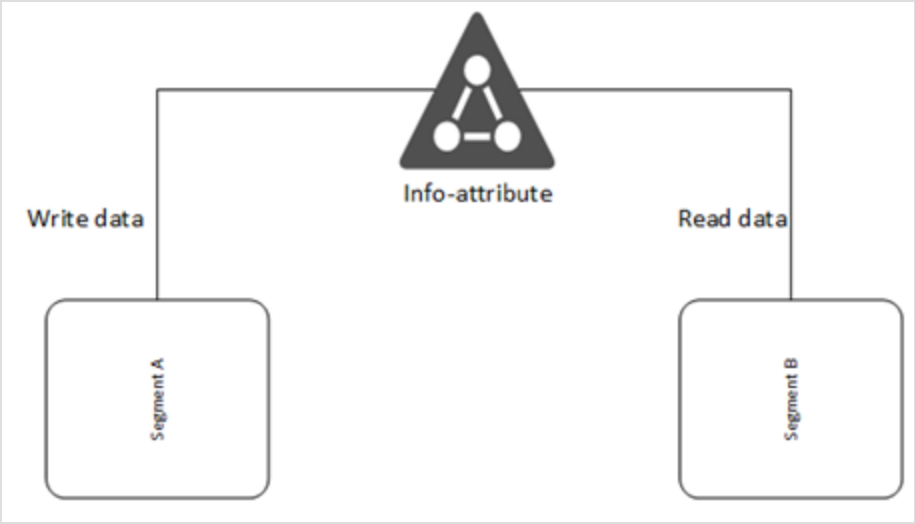


The permissions set in the screenshot above provide access to the attributes defined in the `Personal-Information` property set. This property set contains 40+ attributes that users can read from and write to. The complete list of attributes can be found in the following article: https://docs.microsoft.com/en-us/windows/win32/adschema/r-personal-information
By default, every user that has successfully been authenticated within the same forest is an ‘authenticated user’. This means we can use Active Directory as a temporary data store and exchange data between the two isolated networks by writing the data to these attributes and then reading the data from the other segment.
If we have access to a user account, we can use that user account in both network segments simultaneously to exchange data over

Active Directory. This will work, regardless of the security settings of the workstation, since the account will communicate directly to the domain controller instead of the workstation.

To route data over LDAP we need to get code execution privileges first on workstations in both segments. To achieve this, however, is up to the reader and beyond the scope of this blogpost.
To route data over LDAP, we would write data into one of the attributes and read the data from the other network segment.
In a typical scenario where we want to execute `ipconfig` on a workstation in network Segment B from a workstation in network Segment A, we would write the `ipconfig` command into an attribute, read the `ipconfig` command from network segment B, execute the command and write the results back into the attribute.

This process is visualized in the following overview:



A sample script to utilize this can be found on our GitHub page: https://github.com/fox-it/LDAPFragger/blob/master/LDAPChannel.ps1

While this works in practice to communicate between segmented networks over Active Directory, this solution is not ideal. For example, this channel depends on the replication of data between domain controllers. If you write a message to domain controller A, but read the message from domain controller B, you might have to wait for the domain controllers to replicate in order to get the data. In addition, in the example above we used to info-attribute to exchange data over Active Directory. This attribute can hold up to 1024 bytes of information. But what if the payload exceeds that size? More issues like these made this solution not an ideal one.

Lastly, people already built some proof of concepts doing the exact same thing. Harmj0y wrote an excellent blogpost about this technique: https://www.harmj0y.net/blog/powershell/command-and-control-using-active-directory/

That is why we decided to build an advanced LDAP communication channel that fixes these issues.

Building an advanced LDAP channel

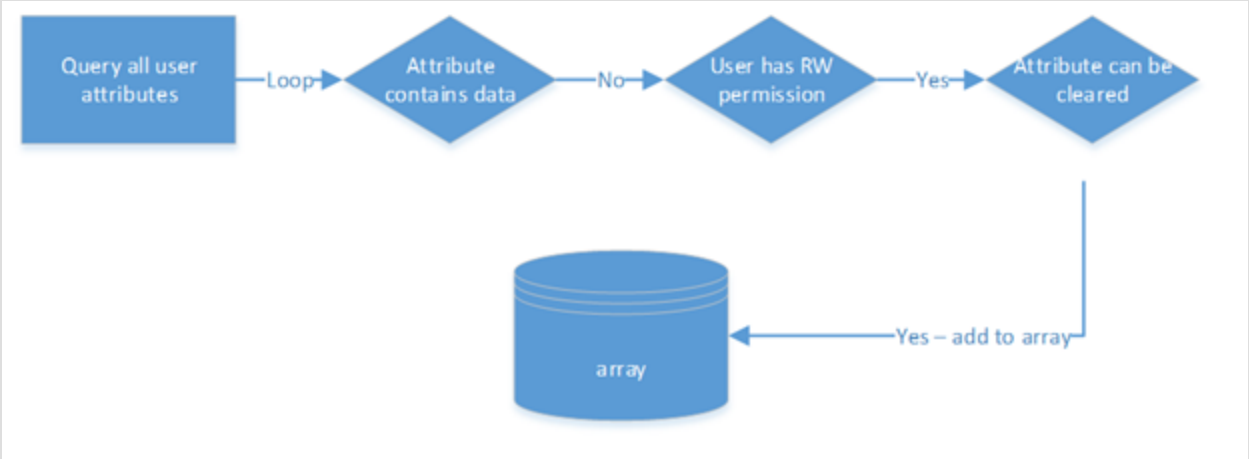
In the example above, the info-attribute is used. This is not an ideal solution, because what if the attribute already contains data or if the data ends up in a GUI somewhere?

To find other attributes, all attributes from the Active Directory schema are queried and:

- Checked if the attribute contains data;
- If the user has write permissions on it;
- If the contents can be cleared.

If this all checks out, the name and the maximum length of the attribute is stored in an array for later usage.

Visually, the process flow would look like this:



As for (payload) data not ending up somewhere in a GUI such as an address book, we did not find a reliable way to detect whether an attribute ends up in a GUI or not, so attributes such as `telephoneNumber` are added to an in-code blacklist. For now, the attribute with the highest maximum length is selected from the array with suitable attributes, for speed and efficiency purposes. We refer to this attribute as the 'data-attribute' for the rest of this blogpost.

Sharing the attribute name

Now that we selected the data-attribute, we need to find a way to share the name of this attribute from the sending network segment to the receiving side. As we want the LDAP channel to be as stealthy as possible, we did not want to share the name of the chosen attribute directly.

In order to overcome this hurdle we decided to use hashing. As mentioned, all attributes were queried in order to select a suitable attribute to exchange data over LDAP. These attributes are stored in a hashtable, together with the CRC representation of the attribute name. If this is done in both network segments, we can share the hash instead of the attribute name, since the hash will resolve to the actual name of the attribute, regardless where the tool is used in the domain.

Avoiding replication issues

Chances are that the transfer rate of the LDAP channel is higher than the replication occurrence between domain controllers. The easy fix for this is to communicate to the same domain controller.
That means that one of the clients has to select a domain controller and communicate the name of the domain controller to the other client over LDAP.

The way this is done is the same as with sharing the name of the data-attribute. When the tool is started, all domain controllers are queried and stored in a hashtable, together with the CRC representation of the fully qualified domain name (FQDN) of the domain

controller. The hash of the domain controller that has been selected is shared with the other client and resolved to the actual FQDN of the domain controller.

Initially sharing data

We now have an attribute to exchange data, we can share the name of the attribute in an obfuscated way and we can avoid replication issues by communicating to the same domain controller. All this information needs to be shared before communication can take place. Obviously, we cannot share this information if the attribute to exchange data with has not been communicated yet (sort of a chicken-egg problem).

The solution for this is to make use of some old attributes that can act as a placeholder. For the tool, we chose to make use of one the following attributes:

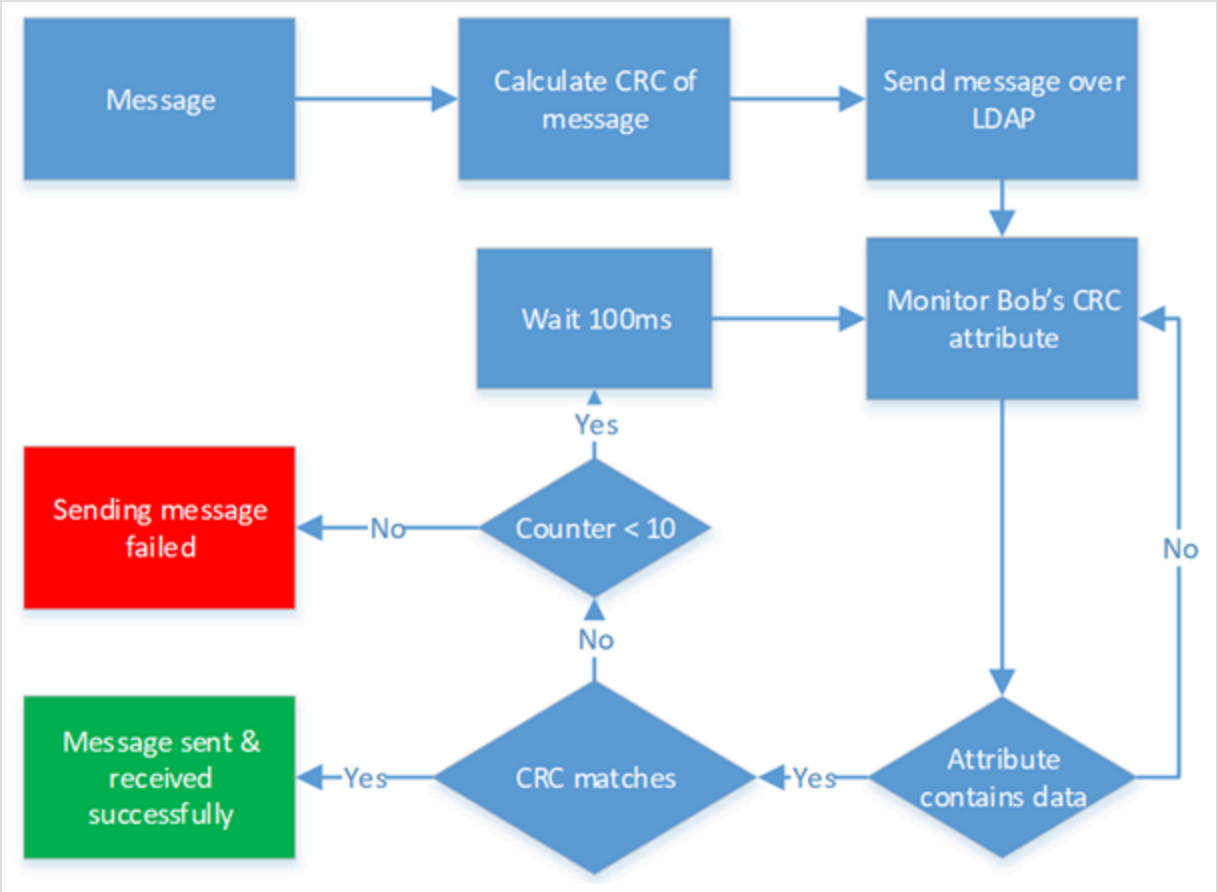
- primaryInternationalISDNNumber;
- otherFacsimileTelephoneNumber;
- primaryTelexNumber.

These attributes are part of the Personal-Information property set, and have been part of that since Windows 2000 Server. One of these attributes is selected at random to store the initial data.
We figured that the chance that people will actually use these attributes are low, but time will tell if that is really the case 😉

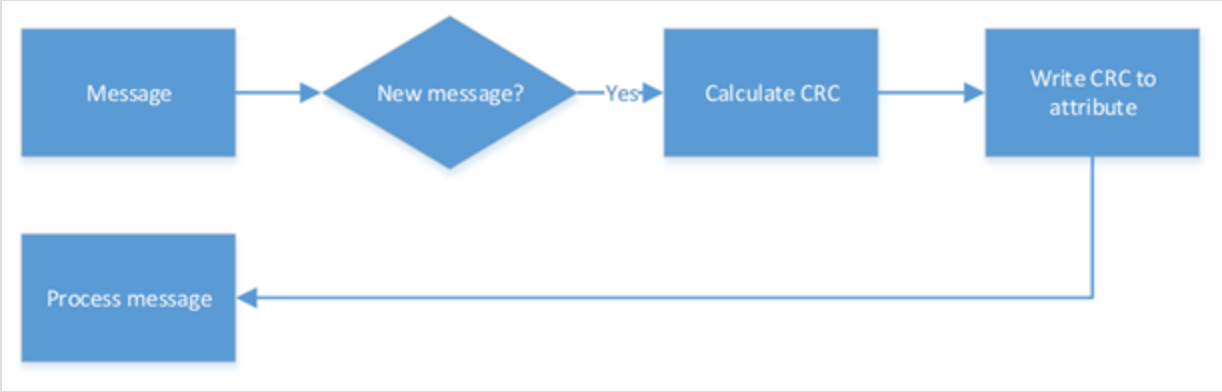
Message feedback

If we send a message over LDAP, we do not know if the message has been received correctly and if the integrity has been maintained during the transmission. To know if a message has been received correctly, another attribute will be selected – in the exact same way as the data-attribute – that is used to exchange information regarding that message. In this attribute, a CRC checksum is stored and used to verify if the correct message has been received.

In order to send a message between the two clients – Alice and Bob –, Alice would first calculate the CRC value of the message that she is about to send herself, before she sends it over to Bob over LDAP. After she sent it to Bob, Alice will monitor Bob's CRC attribute to see if it contains data. If it contains data, Alice will verify whether the data matches the CRC value that she calculated herself. If that is a match, Alice will know that the message has been received correctly.
If it does not match, Alice will wait up until 1 second in 100 millisecond intervals for Bob to post the correct CRC value.



The process on the receiving end is much simpler. After a new message has been received, the CRC is calculated and written to the CRC attribute after which the message will be processed.



Fragmentation

Another challenge that we needed to overcome is that the maximum length of the attribute will probably be smaller than the length of the message that is going to be sent over LDAP. Therefore, messages that exceed the maximum length of the attribute need to be fragmented.

The message itself contains the actual data, number of parts and a message ID for tracking purposes. This is encoded into a base64 string, which will add an additional 33% overhead.

The message is then fragmented into fragments that would fit into the attribute, but for that we need to know how much information

we can store into said attribute.
Every attribute has a different maximum length, which can be looked up in the Active Directory schema. The screenshot below displays the maximum length of the info-attribute, which is 1024.

```
PS C:\Users\Administrator> Get-ADObject -Filter {lDAPDisplayname -eq 'info'} -SearchBase 'CN=Schema,CN=Configuration,DC=chipmunk,DC=local' -Properties
rangeUpper,lDAPDisplayName | select Name, lDAPDisplayname, RangeUpper

Name    lDAPDisplayname RangeUpper
----    --------------- ----------
Comment info                  1024
```

At the start of the tool, attribute information such as the name and the maximum length of the attribute is saved. The maximum length of the attribute is used to fragment messages into the correct size, which will fit into the attribute. If the maximum length of the data-attribute is 1024 bytes, a message of 1536 will be fragmented into a message of 1024 bytes and a message of 512 bytes.
After all fragments have been received, the fragments are put back into the original message. By also using CRC, we can send big files over LDAP. Depending on the maximum length of the data-attribute that has been selected, the transfer speed of the channel can be either slow or okay.

Autodiscover

The working of the LDAP channel depends on (user) accounts. Preferably, accounts should not be statically configured, so we needed a way for clients both finding each other independently.
Our ultimate goal was to route a Cobalt Strike beacon over LDAP. Cobalt Strike has an experimental C2 interface that can be used to create your own transport channel. The external C2 server will create a DLL injectable payload upon request, which can be injected into a process, which will start a named pipe server. The name of the pipe as well as the architecture can be configured. More information about this can be read at the following location: https://www.cobaltstrike.com/help-externalc2

Until now, we have gathered the following information:

- 8 bytes – Hash of data-attribute
- 8 bytes – Hash of CRC-attribute
- 8 bytes – Hash of domain controller FQDN

Since the name of the pipe as well as the architecture are configurable, we need more information:

- 8 bytes – Hash of the system architecture
- 8 bytes – Pipe name

The hash of the system architecture is collected in the same way as the data, CRC and domain controller attribute. The name of the pipe is a randomized string of eight characters. All this information is concatenated into a string and posted into one of the placeholder attributes that we defined earlier:

- primaryInternationalISDNNumber;
- otherFacsimileTelephoneNumber;
- primaryTelexNumber.

The tool will query the Active Directory domain for accounts where one of each of these attributes contains data. If found and parsed successfully, both clients have found each other but also know which domain controller is used in the process, which attribute will contain the data, which attribute will contain the CRC checksums of the data that was received but also the additional parameters to

create a payload with Cobalt Strike's external C2 listener. After this process, the information is removed from the placeholder attribute. Until now, we have not made a distinction between clients. In order to make use of Cobalt Strike, you need a workstation that is allowed to create outbound connections. This workstation can be used to act as an implant to route the traffic over LDAP to another workstation that is not allowed to create outbound connections. Visually, it would something like this.

Let us say that we have our tool running in segment A and segment B – Alice and Bob. All information that is needed to communicate over LDAP and to generate a payload with Cobalt Strike is already shared between Alice and Bob. Alice will forward this information to Cobalt Strike and will receive a custom payload that she will transfer to Bob over LDAP. After Bob has received the payload, Bob will start a new suspended child process and injects the payload into this process, after which the named pipe server will start. Bob now connects to the named pipe server, and sends all data from the pipe server over LDAP to Alice, which on her turn will forward it to Cobalt Strike. Data from Cobalt Strike is sent to Alice, which she will forward to Bob over LDAP, and this process will continue until the named pipe server is terminated or one of the systems becomes unavailable for whatever reason. To visualize this in a nice process flow, we used the excellent format provided in the external C2 specification document.

After a new SMB beacon has been spawned in Cobalt Strike, you can interact with it just as you would normally do. For example, you can run MimiKatz to dump credentials, browse the local hard drive or start a VNC stream.
The tool has been made open source. The source code can be found here: https://github.com/fox-it/LDAPFragger

The tool is easy to use: Specifying the `cshost` and `csport` parameter will result in the tool acting as the proxy that will route data from and to Cobalt Strike. Specifying AD credentials is not necessary if integrated AD authentication is used. More information can be found on the Github page. Please do note that the default Cobalt Strike payload will get caught by modern AVs. Bypassing AVs is beyond the scope of this blogpost.

Why a C2 LDAP channel?

This solution is ideal in a situation where network segments are completely segmented and firewalled but still share the same Active Directory domain. With this channel, you can still create a reliable backdoor channel to parts of the internal network that are otherwise unreachable for other networks, if you manage to get code execution privileges on systems in those networks. Depending on the chosen attribute, speeds can be okay but still inferior to the good old reverse HTTPS channel. Furthermore, no special privileges are needed and it is hard to detect.

Remediation

In order to detect an LDAP channel like this, it would be necessary to have a baseline identified first. That means that you need to know how much traffic is considered normal, the type of traffic, et cetera. After this information has been identified, then you can filter out the anomalies, such as:

- Unusual amount of traffic from clients to a domain controller;
- A high volume of changes to AD objects, so monitor for event ID 5136 on domain controllers;
- Enable and inspect LDAP logging. For more information on how to do that, see the following article: https://support.microsoft.com/en-us/help/314980/how-to-configure-active-directory-and-lds-diagnostic-event-logging

- Use Advanced Threat Protection from Microsoft. It has some neat functionality that makes it easier to detect malicious LDAP queries. See for more information the following article: https://techcommunity.microsoft.com/t5/Microsoft-Defender-ATP/Hunting-for-reconnaissance-activities-using-LDAP-search-filters/ba-p/824726

Monitor the usage of the three static placeholders mentioned earlier in this blogpost might seem like a good tactic as well, however, that would be symptom-based prevention as it is easy for an attacker to use different attributes, rendering that remediation tactic ineffective if attackers change the attributes.

Published March 19, 2020

Hunting for beacons

In-depth analysis of the new Team9 malware family