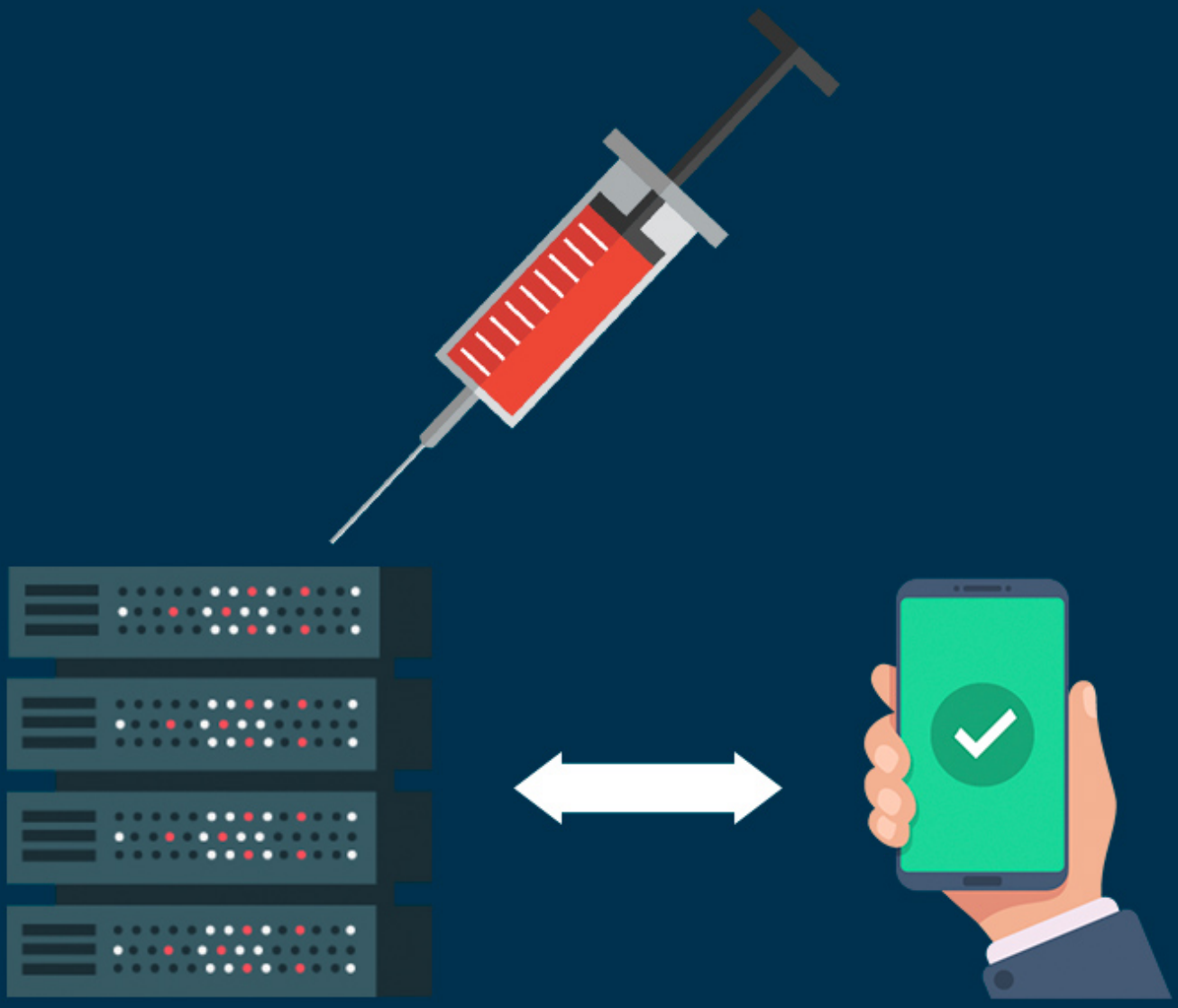
LDAP Injections



By Christophe Limpalair

August 2020

Version 1.0

CYBR

This ebook is a collection of lessons from our course, Injection Attacks: The Complete 2020 Guide. This version only contains the LDAP Injections section, and not other injection attacks for brevity. There is a separate ebook version that includes all injection attacks if you'd rather. We know some people may prefer the format of an ebook over a series of lessons online, or of written versus video lessons. In any case, we hope you enjoy it!

Table of Contents

Overview of LDAP Injections

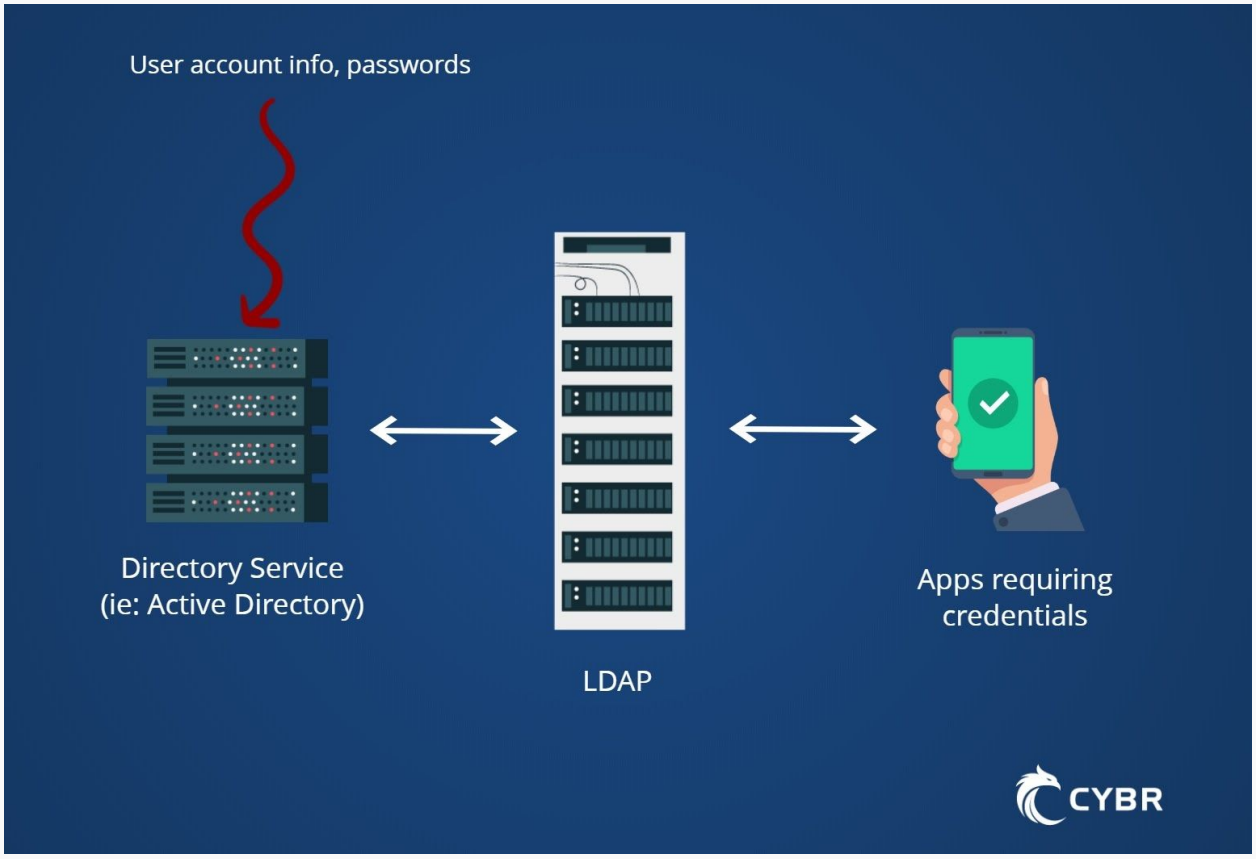
Slides used in this lesson.

In this lesson, we take a quick look at what LDAP is, and then we explore examples of LDAP injection attacks.

What is LDAP?

LDAP stands for Lightweight Directory Access Protocol. It is an open and vendor-neutral industry-standard protocol for directory services authentication. In basic terms, it's a protocol you can use to authenticate and authorize access to IT resources within your organization.

If you're familiar with Active Directory (or "AD"), LDAP essentially provides a way of talking with it, letting you manage user and group membership stored in Active Directory.



LDAP illustration

If you're not familiar with AD, think of it as a database of user and group information.

LDAP and Active Directory contain the keys to the kingdom of access, making them critical to understand in order to protect our networks and resources from unauthorized access, if we use them in our organization.

Because LDAP has been around for a long time and has been adopted by a huge number of organizations, it is still widespread today, making it a ripe target for cybercriminals.

LDAP Commands

In order for LDAP to communicate with something like AD, it sends queries to the authorization service (in our example, Active Directory), and then gets a response back.

An LDAP query is simply a command that asks the directory service for information. So if you're trying to pull information about a user's groups that they belong to, you might submit a query that looks like this:

```
(&(objectClass=user)(sAMAccountName=UserName)(memberof=CN=YourGroup,OU=Users,DC=YourDomain,DC=com))
```

LDAP Injections

Since your application will be generating these queries much like it would to access data from a database management system, we can open up our applications to vulnerabilities in a similar way to SQL injections.

As soon as we grab user-submitted inputs and throw them into an LDAP query, bad things can happen.

Let's illustrate with examples.

Let's say that we have an application that lets us search for users.

The HTTP request might look like this:

```
https://example.com/searchusers?user=Christophe
```

Which would translate into this command:

```
searchfilter="(cn="+user+")"searchfilter="(cn=Christophe)"
```

Instead, if the value of Christophe were replaced with a *:

```
https://example.com/searchusers?user=*
searchfilter="(cn=*)"
```

This would look for all objects with a 'Common Name' attribute (that's what the 'cn' stands for) that is equal to anything. And if the application is vulnerable to LDAP injection, it will display user attributes that you may not be meant to see.

Finding vulnerabilities through errors

And so with LDAP injections, much like other types of injection attacks, an attacker or tester typically tries to cause errors or alter the behavior of the application to check for vulnerabilities.

With LDAP, we could use characters like:

```
^
#
$
(
|
&
*
```

To try and cause errors or a change of behavior just like we would with an SQL injection.

By using logical operators such as | (OR) and & (AND), we can start to gather information about how queries are structured, for example.

Exploiting logical operators

Let's take a look at another attack that could be used to gain logged-in access.

If we have a login form with username and password inputs, and if we have this command being run:

```
(&(USER=Username)(PASSWORD=password))
```

And we inject:

```
christophe)(&))
```

As the username in the input field, and with whatever password we want, if it is vulnerable to LDAP injection, this is what the command would look like:

```
(& (USER=christophe)(&))(PASSWORD=whatever-you-want-here))
```


In this case, only the first filter `(& (USER=christophe)(&))` (before the PASSWORD part) would be processed by the LDAP server, which would always return true and so the attacker would gain access to the system without even having a valid password!

The end result could be granting permissions to unauthorized queries, and the attacker could modify content inside of LDAP, or view corporate information they may not be meant to see.

Blind LDAP Injections

Just like with SQL and OS Command injections, in some cases, we may have to rely on Blind Injection techniques because we're simply not getting any helpful responses back.

By using the AND as well as OR logical operators, we can try to change outputs returned to us in order to gather information about how the queries are structured and which attacks are working.

Conclusion

There are really good whitepapers on this topic that have been made available for free, so I'd recommend reading them if you have interest in exploring this topic further!

- Black Hat
- SPI Labs

Go ahead and complete this lesson, and let's move on!

Defending against LDAP Injections

Slides used in this lesson.

Now that we've covered the concepts of LDAP injections and what they look like, let's talk about some prevention techniques.

LDAP Injection Defense Techniques

The primary defense against LDAP injections is to:

- Escape all variables using the right LDAP encoding functions

Secondary defense options include:

- Using frameworks that automatically protect from LDAP injections
- Allowlist input validation
- Use least privileges

Primary Defense

Escaping

Of course, the best way to defend against injections is to avoid putting user inputs into LDAP queries, but that's not always possible. So when building LDAP queries in applications, you must escape any untrusted data that is added to those LDAP queries.

Since LDAP injections are performed with special characters, we want to remove as many of those as possible without breaking the needed functionality. For example, we'd want to filter:

- Parentheses
- Asterisks
- Logical operators (AND "&", OR "|" and NOT "!")
- Relational operators (=, <=, >=, ~=)

Things like hash signs, commas, semicolons, and others should also be considered.

Secondary defense options:

Frameworks

There are frameworks out there that will automatically escape values before handing them off to LDAP queries, helping prevent these injections from happening.

Input validation

In addition to escaping, it's usually a good idea to also perform input validation. Not only can this provide better user experiences, but it can also help detect potentially malicious input and block it from being passed to the LDAP query. Allowlists are not always effective, or if too aggressive, can prevent legitimate values, but when they are possible to implement, they are a good secondary defense option.

Least Privilege

Of course, you should always follow the principle of least privilege, and this rule applies here as well. Even if an attacker manages to find a vulnerability, they can only do as much damage as their permissions allow.

By minimizing the privileges assigned to the LDAP account in your environment, you can cut back on the damage done.

So with all of this said, here are some recommendations:

Code Review

During code review, check for any queries to LDAP, and make sure those queries are properly escaping special characters.

Automated tools

You can use automated tools to scan your code before it even goes out to production. An example of this is using OWASP ZAP modules made to detect LDAP injection issues.

Conclusion and Additional Resources

Conclusion and Additional Resources

For the video version of this ebook, including more injection attacks like SQL Injections, OS Command injections, SMTP Header injections, and XXE / XPATH injections, check out our course on Cybr.

Be sure to follow Cybr's social media accounts for more content like this:

- LinkedIn
- Facebook
- YouTube

We also have a Discord community where you can chat in real-time with me, other course authors, and other Cybr members as well as mentors. We have forums if you'd prefer asking questions there and/or finding additional resources.

I'd also love to connect on LinkedIn, where I post regularly.

Thanks again, and hope to see you soon!

- Christophe