INTRODUCTION

The aim of this study is improve knowledge about LDAP Injection Vulnerability and LDAP subject.

DETAILS

**LDAP :** Lightweight Directory Access Protocol is a common software protocol designed to let anyone on a network to find resources such as other individuals, files and devices. Directory services such as LDAP are useful for intrantes. It can also be used to store usernames and passwords as part of a Single Sign-On (SSO) system.

What is LDAP injection?

Ldap Injection attacks are similiar to SQL Injection attacks. This attack abuse the parameters used in LDAP query. In most cases, application does not filter parameters correctly. This could let hacker to inject malicious code to the environment.

Ldap exploits can result in exposure and theft of sensitive data. Advanced LDAP Injection techniques can also execute arbitrary commands. This lets them obtain unauthorized permissions and also alter LDAP tree information. Environments that are most vulnerable to LDAP Injection attacks include ADAM and OpenLDAP.

How Do LDAP Injection Attacks Work?

Clients query an LDAP server by sending a request for a directory entry that matches a specific filter. If an entry matching the LDAP search filter is found, server returns the requested information.

Search filters used in LDAP queries follow the syntax specified in RFC 4515. Filters are constructed based on one or more LDAP attributes specified as key/value pairs in parentheses. Filters can be combined using logical and comparison operators and can contain wildcards.

Here are some examples:

* (cn=David\*) matches anything with a common name beginning with the string David (the asterisk matches any character).
* (!(cn=David\*)) matches anything where the common name does not start with the string David.
* (&(cn=D\*)(cn=\*Smith)) uses the AND logical operator, represented by the & symbol. Matches entries that start with the letter D and end with Smith.
* (|(cn=David\*)(cn=Elisa\*)) uses the OR logical operator, represented by the pipe symbol. Matches entries whose common name starts with one of the strings Dave or Elisa.

Similiar to SQL injection and code injection attacks, an LDAP injection vulnerability results when an application injects unfiltered user input directly into an LDAP statement. An attacker can use LDAP filter syntax to pass a string value, which will cause the LDAP server to execute various queries and other LDAP statements. Usually the injected command will exploit misconfiguration or inappropriate permissions set on the LDAP server.

**Types of LDAP Injection Attacks**

**Access Control Bypass**

All login pages have 2 textbox fields. One for username and other for password. The user inputs are USER(Uname) and PASSWORD(Pwd). A client supplies the user/password pair. To confirm the existence of this pair, LDAP constructs search filters and sends them to the LDAP server.

(&(USER=Uname)(PASSWORD=Pwd))

An attacker can eter a valid username (faruk.koyuncu for example) while also injecting the correct sequence after the name. With this way they successfully bypass the password check. By knowing the username any string can be introduced as the Pwd value. Then the following query gets sent to the server:

(&(USER=**john90)(&)**)(PASSWORD=Pwd))

LDAP server then processes only the first part of the filter. The query processes only the (&(USER=**john90)(&)** part of the query. Since this query is always correct, the attacker enter system with any arbitrary password.

**Elevation of Privileges**

Some queries list all documents and they’re visible to users that have a low-security level. For example /Information/Reports, /Information/UpcomingProjects, etc. files in the directory. The “Information” part is the user entry for the first parameter. All these documents have a “Low” security level. The “Low” part is the value for second parameter. This second parameter also allows hacker to Access high-security levels. In order to do that the hacker must use an injection that looks like this :

“Information)(security\_level=\*))(&(directory=documents”

This injection results in this filter:

(&(directory=Information)(security\_level=\*))(&(directory=Information)(security\_level=low))

When LDAP processes the first filter, the second filters gets ignored. And first query makes the second query become ignored compeletely. That’s how attackers see a list of documents that can actually only be accessed by users with all security levels. Even though attacker doesn’t have priveleges to see these informations.

**Information Disclosure**

Some resource explorers let a user know exatcly which resource is available in the system. For example, in a website that sells clothes, the user can look spesific shirts or pants to see if they are available. In this situation LDAP filter is something like this:

(|(type=Resource1)(type=Resource2))

Both Resource1 and Resource2 show the kinds of resources in the system. Resource1=Jeans  and Resource2=T-Shirts show all the jeans and T-Shirts that are available for purchase in the system. How do hackers exploit this? By injecting (**uid=\***) into **Resource1=Jeans**. This query then gets sent to the server:

(|(type=**Jeans)(uid=\*)**)(type=T-Shirts))

Then LDAP server then shows all jeans and objects that is kept on that server.

**LDAP Injection Examples Using Logical Operators**

An LDAP filter can b used to make a query that’s missing a logic operator (OR and AND). An injection looks like:

“value)(injected\_filter”

Results in two filters (the second gets ignored while the first one gets executed in OpenLDAP implementations)

(attribute=value)(injected\_filter)

ADAM LDAP doesn’t allow queries with two filters. This renders this injection useless. Then we have the & and | standalone symbols. Making queries with them looks like this (OpenLDAP igores second filter while first one gets executed, ADAM LDAP is does not even allow queries with 2 filters):

(&(attribute=value)(second\_filter))

(|(attribute=value)(second\_filter))

Filters that have the OR or AND logic operators can make queries in which this injection:

“value)(injected\_filter”

Results in this filter:

**(&(attribute=value)(injected\_filter))** (second\_filter)).

As you can see, this filter isn’t even syntactically correct (“.” at the end f.e.). Yet, OpenLDAP will process it regardless. It will go from left to right and ignore all characters after the first filter closes. What does that mean? Certain LDAP Cliet components ignore the second filter. The first complete one is sent to ADAM and OpenLDAP. That’s how injections bypass security.

In cases where applications have a framwork that checks th filter, it needs to be correct. An example of a synthetically correct injection looks something like:

“value)(injected\_filter))(&(1=0”

This shows two different filters where the second one gets ignored:

**(&(attribute=value)(injected\_filter))(&1=0)** (second\_filter)).

How do attackers test an application to see if it’s vulnerable to code injections? They send a query to the server that generates an invalid input(Wrong syntaxed LDAP filter or filter character like \*). If a server returns an error message, it means the server executed his query. Meaning code injection techniques are possible.

**AND LDAP Injection**

In this case, the application constructs a query with the **“&”**operator. This together with one or more parameters that are introduced by the user is used to search in the LDAP directory.

(&(parameter1=value1)(parameter2=value2))

The search uses **value1**and **value2**as values that let the search in the LDAP directory happen. Hackers can maintain a correct filter construction while also injecting their malicious code. This is how they abuse the query to pursue their own objectives.

**OR LDAP Injection**

There are cases where the application makes a normal query with the (**|**) operator. Together with one or more parameters that the user introduces. An example looks something like this:

(|(parameter=value1)(parameter2=value2))

As before, **value1**and **value2**are used for the search.

## **BLIND LDAP Injections**

Attackers can take so much information just from server’s response. The application itself may not show any error messages. Yet, the code that’s injected into the LDAP filter will generate a valid response or an error. A true result or a false result. Attackers exploit this behaviour to obtain answers to true or false questions from the server. These techniques called as **BLIND ATTACKS.** Blind attacks are work on binary logic.

**AND BLIND LDAP Injection**

Imagine an online shop that can list all Puma shirts from an LDAP directory. But the error messages are not returned. This LDAP search filter is sended:

(&(objectClass=Shirt)(type=Puma\*))

Any available Puma shirts are shown to the user as icons. If there are no Puma shirts available, the user won’t see any icons. This is where Blind LDAP Injection comes into play. “\*)objectClass=\*))(&(objectClass=void” is injected and now the application constructs an LDAP query that looks like:

(&(objectClass=**\*)(objectClass=\*))(&(objectClass=void**)(type=Puma\*))

The server process only the (&(objectClass=\*)(objectClass=\*)) part of the LDAP filter. Now the shirt icon shows to the client. How so? The objectClass=\* filter always returns an object. An icon showing means the response is true. Otherwise the response is false. Attacker now have the option of using blind injection techniques in many ways. For Example:

(&(objectClass=**\*)(objectClass=users))(&(objectClass=foo**)(type=Puma\*))

(&(objectClass=**\*)(objectClass=Resources))(&(objectClass=foo**)(type=Puma\*))

Different objectClass values can be obtained with the help of these injections. If even a single shirt icon is shown then the objectClass value exists. Otherwise, the objectClass doesn’t exist. An attacker can obtain all sorts of information by using TRUE/FALSE questions via BLIND LDAP injections.

**OR BLIND LDAP Injection**

Injection in an OR environment looks like this:

(|(objectClass=**void)(objectClass=void))(&(objectClass=void**)(type=Puma\*))

This LDAP query doesn’t obtain any objects from the LDAP directory service. The shirt icon doesn’t get shown to the client. Making it a FALSE response. If an icon is shown it is a TRUE response. In order to gather sensitive informations from the system we can inject queries like this:

(|(objectClass=**void)(objectClass=users))(&(objectClass=void**)(type=Puma\*))

(|(objectClass=**void)(objectClass=Resources))(&(objectClass=void**)(type=Puma\*))

Same thing with the AND BLIND INJECTION

**How To Protect Ourselves From The LDAP Vulnerabilities**

* Sanitize Inputs and Check Variables: AND “&”, OR “|”, NOT “!”, =, >=, <=, ~= are all operators that need to be filtered at the application layer to ensure they’re not used in Injection attacks. All values that are contained in LDAP filter should be chcked against a list of valid values in the App layer before LDAP recieves the query.
* Don’t Construct Filters by Concatenating Strings: We need to avoid creating LDAP search filters by concatenating strings, if the string contains a user input. Instead we should create filter b y using functionalities provided by the LDAP library. For example, in the Java UnboundID LDAP SDK, use this code to concatenate two strings provided by the user using an AND operator:

Filter filter = Filter.createANDFilter(  
     Filter.createEqualityFilter("cn", userInput),  
     Filter.createEqualityFilter("mail", userInput));

Creating LDAP filter programatically prevents malicious input from generating filter types that are different than expected. If LDAP library that we use does not provide any way to create our own search filters, we need to replace that library.

* Use Acces Control on LDAP Server: Follow the principle of least privilege and we need to make sure that each account only has permission to perform operations needed for the user’s role. If an application needs to process tasks on behalf of other users, you can use a proxied authentication request to ensure these tasks are handled according to the other user’s access control rights.

REFERENCES

<https://cheatsheetseries.owasp.org/cheatsheets/LDAP_Injection_Prevention_Cheat_Sheet.html>

<https://www.synopsys.com/glossary/what-is-ldap-injection.html>

<https://brightsec.com/blog/ldap-injection/>