

# LDAP Injection & Blind LDAP Injection

---

## In Web Applications

**Authors:** Chema Alonso, Rodolfo Bordón, Antonio Guzmán y Marta Beltrán

**Speakers:** Chema Alonso & José Parada Gimeno

**Abstract.** LDAP Services are a key component in companies. The information stored in them is used for corporate applications. If one of these applications accepts input from a client and execute it without first validating it, attackers have the potential to execute their own queries and thereby extract sensitive information from the LDAP directory. In this paper a deep analysis of the LDAP injection techniques is presented including Blind attacks.

## Index

Section	Page
1. Introduction	02
2. LDAP Overview	02
3. Common LDAP environments	03
4. LDAP Injection in Web Applications	04
4.1. AND LDAP Injection	06
4.1.1. Example 1: Access Control Bypass	06
4.1.2. Example 2: Elevation of Privileges	07
4.2. OR LDAP Injection	08
4.2.1. Example 1: Information Disclosure	09
5. Blind LDAP Injection	10
5.1. AND Blind LDAP Injection	10
5.2. OR Blind LDAP Injection	11
5.3. Exploitation Example	11
5.3.1. Discovering Attributes	11
5.3.2. Booleanization	13
5.3.3. Charset Reduction	15
6. Securing Applications against	
Blind LDAP Injection & LDAP Injection attacks	16
7. References	16

Greetings: RoMaNSoFt, Palako, Raul@Apache, Al Cutter, Mandingo, Chico Maravillas, Alejandro Martín, Dani Kachakil, Pedro Laguna, Silverhack, Rubén Alonso, David Cervigón, Marty Wilson, Inmaculada Bravo & S@m Pikesley

## 1. Introduction

The amount of data stored in organizational databases has increased rapidly in recent years due to the rapid advancement of information technologies. A high percentage of these data is sensitive, private and critical to the organizations, their clients and partners.

Therefore, databases are usually installed behind internal firewalls, protected with intrusion detection mechanisms and accessed only by applications. To access a database, users have to connect to one of these applications and submit queries through them to the database. The threat to databases arises when these applications do not behave properly and construct these queries without sanitizing user inputs first.

Over 50% of web application vulnerabilities are input validation related, which allows the exploitation of code injection techniques. These attacks have proliferated in recent years causing severe security problems in systems and applications. The SQL injection techniques are the most widely used and studied but there are other injection techniques associated with other languages or protocols such as XPath or LDAP.

Preventing the consequences of these kinds of attacks, lies in studying the different code injection possibilities and in making them public and well known for all programmers and administrators. In this paper the LDAP injection techniques are analyzed in depth, because all the web applications based on LDAP trees might be vulnerable to these kinds of attacks.

The key to exploiting injection techniques with LDAP is to manipulate the filters used to search in the directory services. Using these techniques, an attacker may obtain direct access to the database underlying an LDAP tree, and thereby to important corporate information.

This can be even more critical because the security of many applications and services relies on single sign-on environments based on LDAP directories.

Although the vulnerabilities that lead to these consequences are easy to understand and fix, they persist because of the lack of information about these attacks and their effects. Though previous references to the exploitation of this kind of vulnerability exist the presented techniques don't apply to the vast majority of modern LDAP service implementations. The main contribution of this paper is the presentation and deep analysis of new LDAP injection techniques which can be used to exploit these vulnerabilities.

This paper is organized as follows: sections 2 and 3 explain the LDAP fundamentals needed to understand the techniques presented in the following sections. Section 4 presents the two typical environments where LDAP injection techniques can be used and exemplify these techniques with illustrative cases. Section 5 describes how BLIND LDAP Injection attacks can be done with more examples. Finally, in Section 6, some recommendations for securing systems against this kind of attack are given.

## 2. LDAP Overview

The Lightweight Directory Access Protocol is a protocol for querying and modifying directory services running over TCP/IP. The most widely used implementations of LDAP services are Microsoft ADAM (Active Directory Application Mode) and OpenLDAP.

LDAP directory services are software applications that store and organize information sharing certain common attributes; the information is structured based on a tree of directory entries, and the server provides powerful browsing and search capabilities, etcetera. LDAP is object-oriented, therefore every entry in an LDAP directory services is an instance of an object and must correspond to the rules fixed for the attributes of that object.

Due to the hierarchical nature of LDAP directory services read-based queries are optimized to the detriment of write-based queries.

LDAP is also based on the client/server model. The most frequent operation is to search for directory entries using filters. Clients send queries to the server and the server responds with the directory entries matching these filters.

LDAP filters are defined in the RFC 4515. The structure of these filters can be summarized as:

```
Filter = ( filtercomp )
Filtercomp = and / or / not / item
And = & filterlist
Or = |filterlist
Not = ! filter
Filterlist = 1*filter
Item= simple / present / substring
Simple = attr filtertype assertionvalue
Filtertype = "=" / "~=" / ">=" / "<="
Present = attr = *
Substring = attr "=" [initial] * [final]
Initial = assertionvalue
Final = assertionvalue
```

All filters must be in brackets, only a reduced set of logical (AND, OR and NOT) and relational (=,>,<,<=,>=) operators are available to construct them. The special character "\*" can be used to replace one or more characters in the construction of the filters.

Apart from being logic operators, RFC 4256 allows the use of the following standalone symbols as two special constants:

- (&) -> Absolute TRUE
- (|) -> Absolute FALSE

### 3. Common LDAP environments

LDAP services are a key component for the daily operation in many companies and institutions. Directory Services such as Microsoft Active Directory, Novell E-Directory and RedHat Directory Services are based on the LDAP protocol. But there are other applications and services taking advantage of the LDAP services.

These applications and services used to require different directories (with separate authentication) to work. For example, a directory was required for the domain, a separate directory for mailboxes and distribution lists, and more directories for remote access, databases or web applications. New directories based on LDAP services are multi-purpose,

working as centralized information repositories for user authentication and enabling single sign-on environments.

This new scenario increases the productivity by reducing the administration complexity and by improving security and fault tolerance. In almost every environment, the applications based on LDAP services use the directory for one of the following purposes:

- Access control (user/password pair verification, user certificates management).
- Privilege management.
- Resource management.

Due to the importance of the LDAP services for the corporate networks, the LDAP servers are usually placed in the backend with the rest of the database servers. Figure 1 shows the typical scenario deployed for corporate networks, and it is important to keep this scenario in mind in order to understand the implications of the injection techniques exposed in following sections.

#### 4. LDAP Injection in Web Applications

LDAP injection attacks are based on similar techniques to SQL injection attacks. Therefore, the underlying concept is to take advantage of the parameters introduced by the user to generate the LDAP query. A secure Web application should sanitize the parameters introduced by the user before constructing and sending the query to the server. In a vulnerable environment these parameters are not properly filtered and the attacker can inject malicious code.

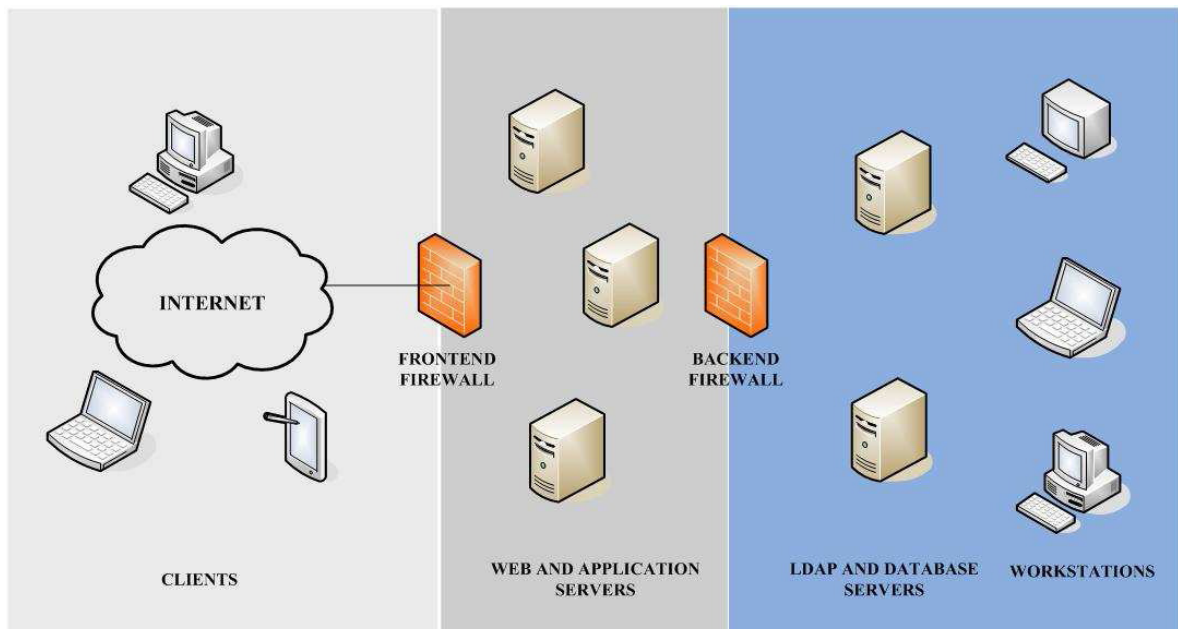


Fig. 1. Typical scenario for an LDAP-based Web application

Taking into consideration the structure of the LDAP filters explained in section 2 and the implementations of the most widely used LDAP: ADAM and OpenLDAP, the following conclusions can be drawn about the code injection. (The following filters are crafted using as value a non sanitized input from the user):

– **(attribute=value)**: If the filter used to construct the query lacks a logic operator (OR or AND), an injection like **"value)(injected\_filter"** will result in two filters: **(attribute=value)(injected\_filter)**. In the OpenLDAP implementations the second filter will be ignored, only the first one being executed. In ADAM, a query with two filters isn't allowed. Therefore, the injection is useless.

- **(|(attribute=value)(second\_filter))** or **(&(attribute=value)(second\_filter))**: If the filter used to construct the query has a logic operator (OR or AND), an injection like **"value)(injected\_filter"** will result in the following filter: **(&(attribute=value)(injected\_filter))(second\_filter)**. Though the filter is not even syntactically correct, OpenLDAP will start processing it left to right ignoring any character after the first filter is closed. Some LDAP Client web components will ignore the second filter, sending to ADAM and OpenLDAP only the first complete one, therefore allowing the injection.

Some application frameworks will check the filter for correctness before sending it to the LDAP server. Should this be the case, the filter has to be syntactically correct, which can be achieved with an injection like:

**"value)(injected\_filter))&(1=0"**. This will result in two different filters, the second being ignored: **(&(attribute=value)(injected\_filter))&(1=0)(second\_filter)**.

As the second filter is going to be ignored by the LDAP Server, some components won't allow an LDAP query with two filters. In these cases a special injection must be crafted in order to obtain a single-filter LDAP query. An injection like: **"value)(injected\_filter"** will result in the following filter: **(&(attribute=value)(injected\_filter))(second\_filter)**.

The typical test to know if an application is vulnerable to code injection consists of sending to the server a query that generates an invalid input. Therefore, if the server returns an error message, it is clear for the attacker that the server has executed his query and that he can exploit the code injection techniques. Taking into account the previous discussion, two kinds of environments can be distinguished: AND injection environments and OR injection environments.

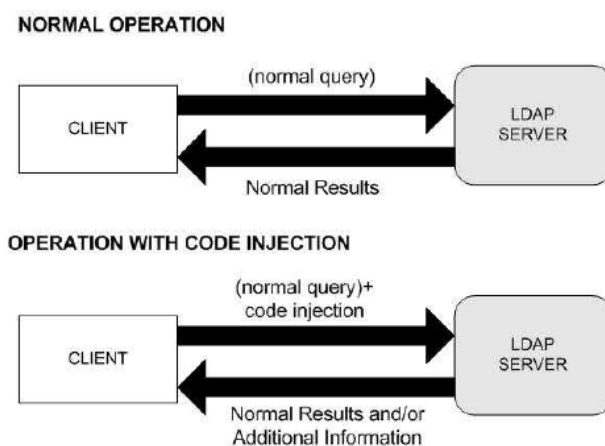


Fig. 2. LDAP injection

#### 4.1. AND LDAP Injection

In this case the application constructs the normal query to search in the LDAP directory with the “&” operator and one or more parameters introduced by the user. For example:

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

Where *value1* and *value2* are the values used to perform the search in the LDAP directory. The attacker can inject code, maintaining a correct filter construction but using the query to achieve his own objectives.

##### 4.1.1. Example 1: Access Control Bypass

A login page has two text box fields for entering user name and password (figure 3). *Uname* and *Pwd* are the user inputs for USER and PASSWORD. To verify the existence of the user/password pair supplied by a client, an LDAP search filter is constructed and sent to the LDAP server:

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

If an attacker enters a valid username, for example, *slisberger*, and injects the appropriate sequence following this name, the password check can be bypassed.

Making *Uname=slisberger)(&))* and introducing any string as the *Pwd* value, the following query is constructed and sent to the server:

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

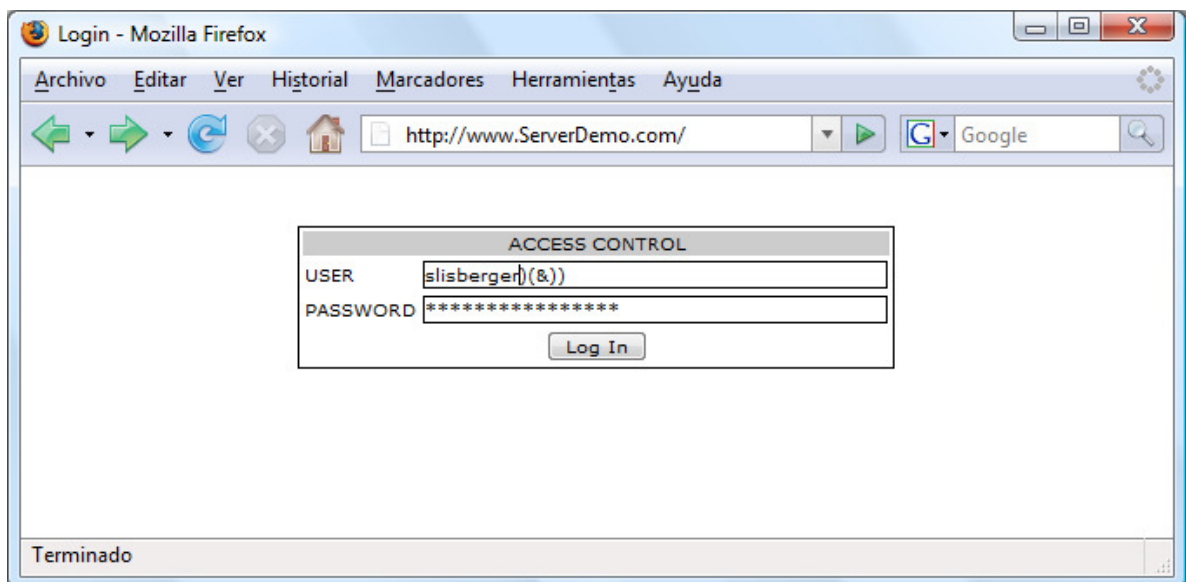


Fig. 3. Login page with LDAP injection

Only the first filter is processed by the LDAP server, that is, only the query *(&(USER=slisberger)(&))* is processed. This query is always true, so the attacker gains access to the system without having a valid password.

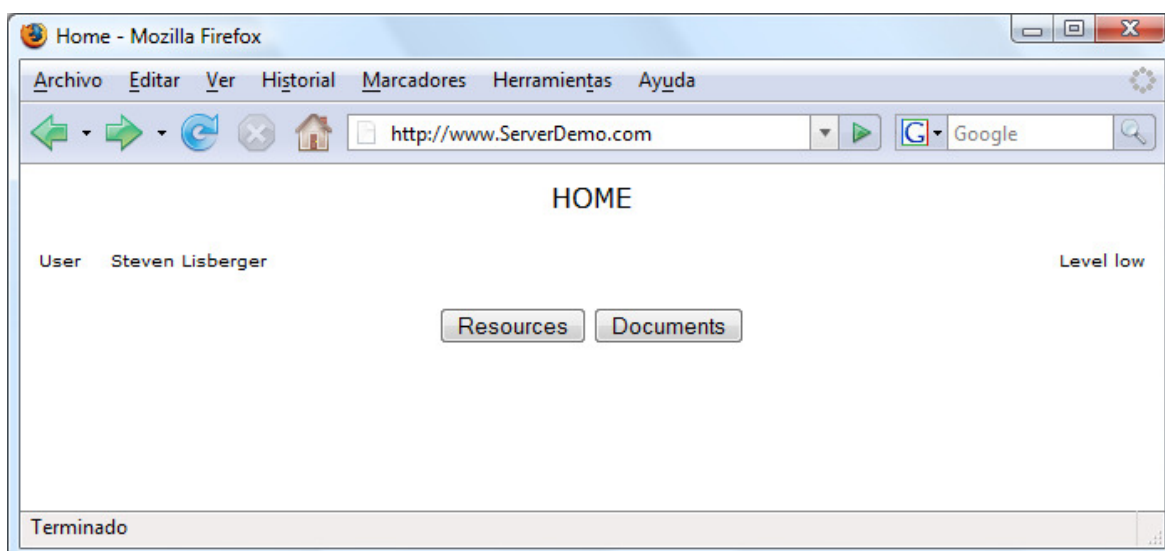


Fig. 4. Home page shown to the attacker after avoiding the access control

#### 4.1.2. Example 2: Elevation of Privileges

For example, suppose that the following query lists all the documents visible for the users with a low security level:

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

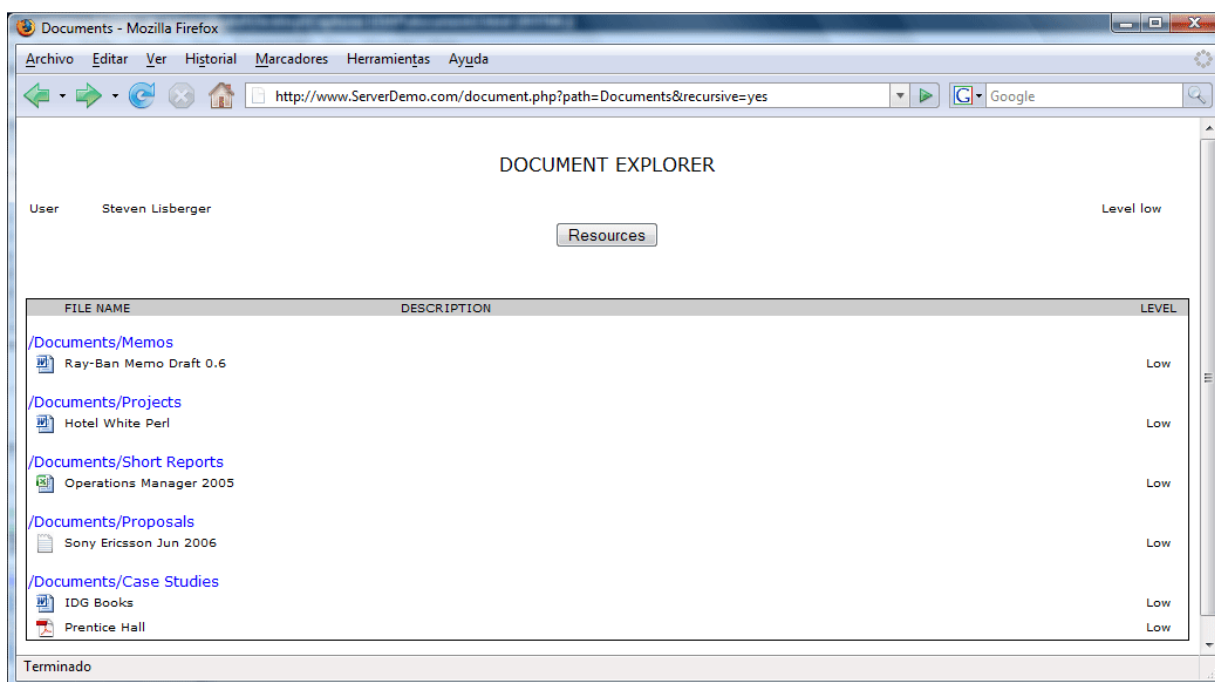


Fig. 5. Low security level documents.

Where “documents” is the user entry for the first parameter and low is the value for the second. If the attacker wants to list all the documents visible for the high security level, he can use an injection like “documents)(security\_level=\*)(&(directory=documents” resulting in the following filter:

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



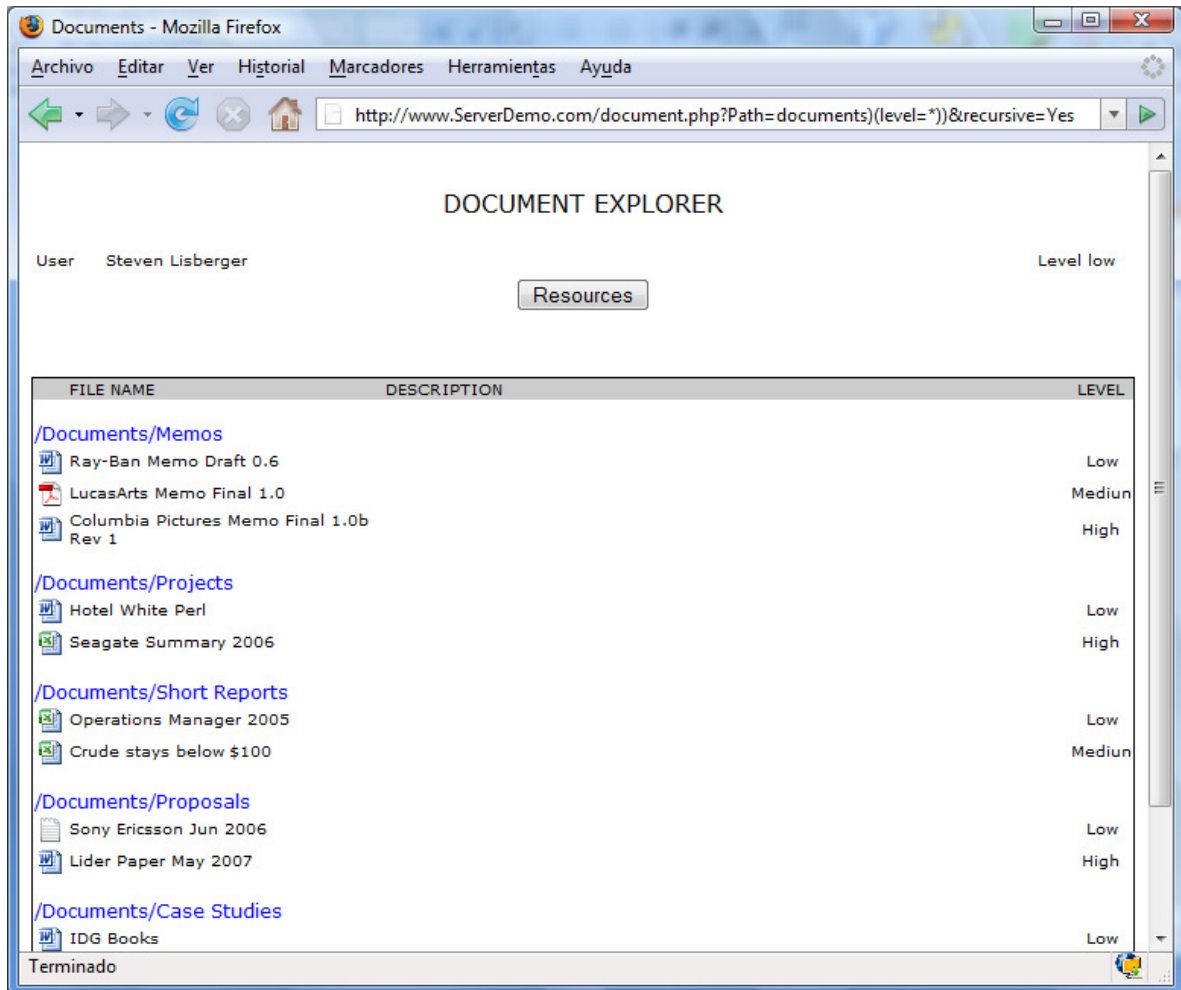


Fig. 6. All security levels documents

The LDAP server will only process the first filter ignoring the second one, therefore, only the following query will be processed:  $(&(directory=documents)(security\ level=*))$ , while  $(&(directory=documents)(security\ level=low))$  will be ignored.

As a result, a list with all the documents available for the users with all security levels will be displayed for the attacker although he doesn't have privileges to see them.

#### 4.2. OR LDAP Injection

In this case the application constructs the normal query to search in the LDAP directory with the "|" operator and one or more parameters introduced by the user. For example:

$$((parameter1=value1)(parameter2=value2))$$

Where *value1* and *value2* are the values used to perform the search in the LDAP directory. The attacker can inject code, maintaining a correct filter construction but using the query to achieve his own objectives.

#### 4.2.1. Example 1: Information Disclosure

Suppose a resources explorer allows users to know the resources available in the system (printers, scanners, storage systems, etc...). This is a typical OR LDAP Injection case, because the query used to show the available resources is:

`((type=Rsc1)(type=Rsc2))`

*Rsc1* and *Rsc2* represent the different kinds of resources in the system. In figure 7, *Rsc1=printer* and *Rsc2=scanner* to show all the available printers and scanners in the system.

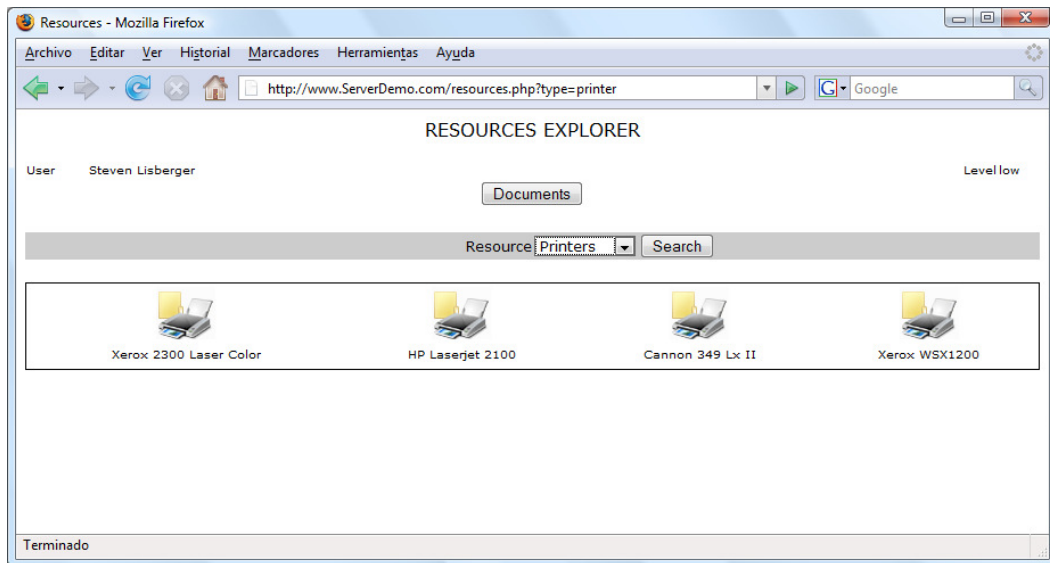


Fig. 7. Resources available to the user from the Resources Consoles Management

If the attacker enters *Rsc1=printer(uid=\*)*, the following query is sent to the server:

`((type=printer)(uid=*)) (type=scanner)`

The LDAP server responds with all the printer and user objects.

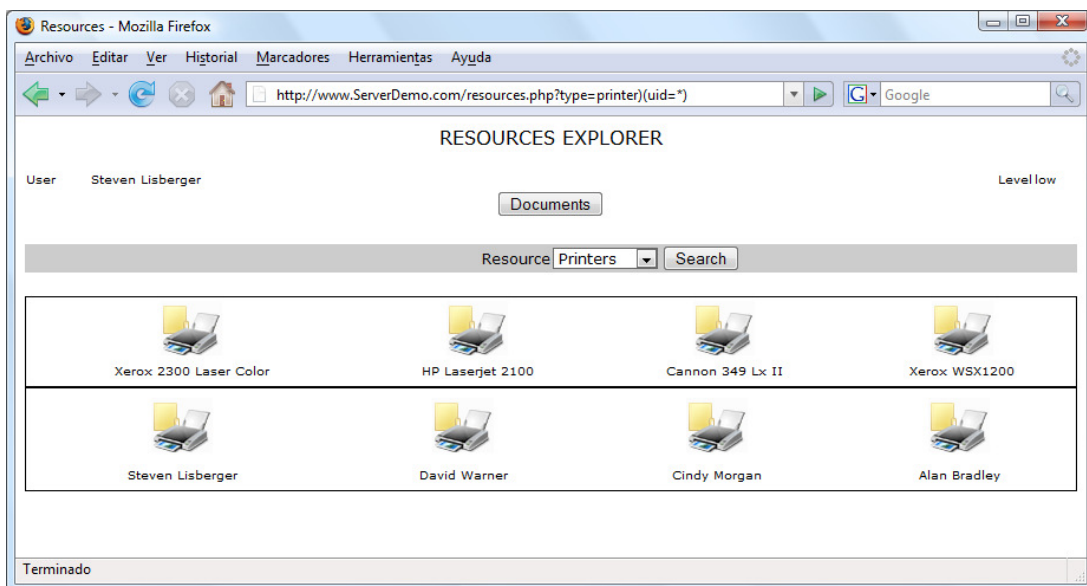


Fig. 8. Information available to the attacker after the LDAP injection

## 5. Blind LDAP Injection

Suppose that an attacker can infer from the server responses, although the application does not show error messages, the code injected in the LDAP filter generates a valid response (true result) or an error (false result). The attacker could use this behavior to ask the server true or false questions. These types of attacks are named “Blind Attacks”. Blind LDAP Injection attacks are slower than classic ones but they can be easily implemented, since they are based on binary logic, and they let the attacker extract information from the LDAP Directory.

### 5.1. AND Blind LDAP Injection

Suppose a web application wants to list all available Epson printers from an LDP directory where error messages are not returned. The application sends the following LDAP search filter:

```
(& (objectClass=printer)(type=Epson*))
```

With this query, if there are any Epson printers available, icons are shown to the client, otherwise no icon is shown. If the attacker performs a Blind LDAP injection attack injecting “\*)(objectClass=\*)(& (objectClass=void”, the web application will construct the following LDAP query:

```
(& (objectClass=*)(objectClass=*)(&(objectClass=void)(type=Epson*))
```

Only the first complete LDAP filter will process:

```
(&(objectClass=*)(objectClass=*))
```

As a result, the printer icon must be shown to the client, because this query always obtains results: the filter *objectClass=\** always returns an object. When an icon is shown the response is true, otherwise the response is false.

From this point, it is easy to use blind injection techniques. For example, the following injections can be constructed:

```
(&(objectClass=*)(objectClass=users))(&(objectClass=foo)(type=Epson*))
```

```
(&(objectClass=*)(objectClass=resources))(&(objectClass=foo)(type=Epson*))
```

This set of code injections allows the attacker to infer the different *objectClass* values possible in the LDAP directory service. When the response web page contains at least one printer icon, the *objectClass* value exists (TRUE), on the other hand the *objectClass* value does not exist or there is no access to it, and so no icon, the *objectclass* value does not exist(FALSE).

Blind LDAP injection techniques allow the attacker access to all information using TRUE/FALSE questions.

## 5.2. OR Blind LDAP Injection

In this case, the logic used to infer the desired information is the opposite, due to the presence of the OR logical operator. Following with the same example, the injection in an OR environment should be:

```
((objectClass=void)(objectClass=void))(&(objectClass=void)(type=Epson*))
```

This LDAP query obtains no objects from the LDAP directory service, therefore the printer icon is not shown to the client (FALSE). If any icon is shown in the response web page then, it is a TRUE response. Thus, an attacker could inject the following LDAP filters for gathering information:

```
((objectClass=void)(objectClass=users))(&(objectClass=void)(type=Epson*))
```

```
((objectClass=void)(objectClass=resources))(&(objectClass=void)(type=Epson*))
```

## 5.3. Exploitation example

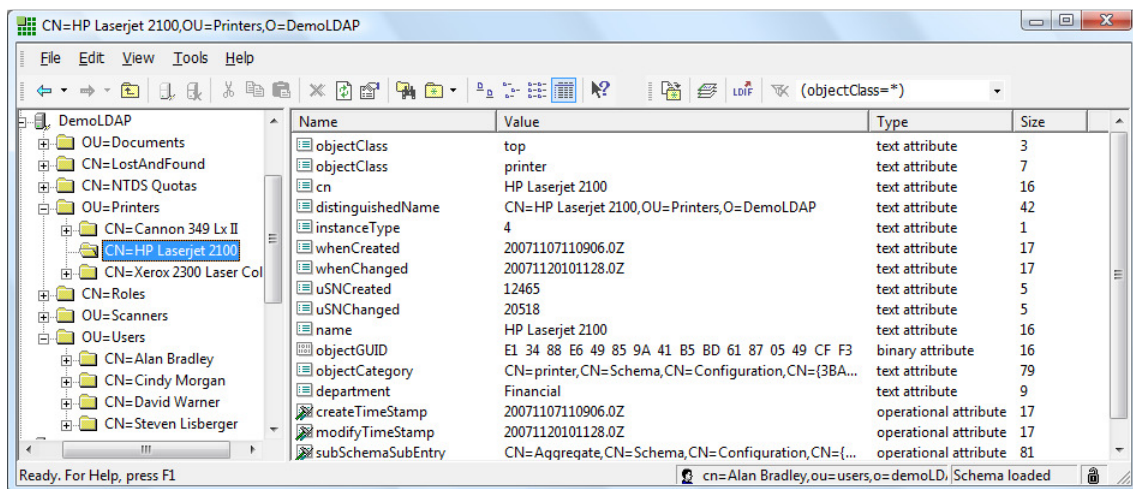
In this section, an LDAP environment has been implemented to show the use of the injection techniques explained above and also to describe the possible effects of the exploitation of these vulnerabilities and the important impact of these attacks in current systems security.

In this example the page *printerstatus.php* receives a parameter *idprinter* to construct the following LDAP search filter:

```
(&(idprinter=Value1)(objectclass=printer))
```

### 5.3.1. Discovering Attributes

Blind LDAP Injection techniques can be used to obtain sensitive information from the LDAP directory services by taking advantage of the AND operator at the beginning of the LDAP search filter built into the web application. For example, given the attributes defined for the printer object shown in figure 9 and the response web page of this LDAP query in figure 10 for *Value1=HPLaserJet2100*,



The screenshot shows the Active Directory Explorer window with the following table of attributes for the printer object:

Name	Value	Type	Size
objectClass	top	text attribute	3
objectClass	printer	text attribute	7
cn	HP Laserjet 2100	text attribute	16
distinguishedName	CN=HP Laserjet 2100,OU=Printers,O=DemoLDAP	text attribute	42
instanceType	4	text attribute	1
whenCreated	20071107110906.0Z	text attribute	17
whenChanged	20071120101128.0Z	text attribute	17
uSNCreated	12465	text attribute	5
uSNChanged	20518	text attribute	5
name	HP Laserjet 2100	text attribute	16
objectGUID	E1 34 88 E6 49 85 9A 41 B5 BD 61 87 05 49 CF F3	binary attribute	16
objectCategory	CN=printer,CN=Schema,CN=Configuration,CN={3BA...	text attribute	79
department	Financial	text attribute	9
createTimeStamp	20071107110906.0Z	operational attribute	17
modifyTimeStamp	20071120101128.0Z	operational attribute	17
subSchemaSubEntry	CN=Aggregate,CN=Schema,CN=Configuration,CN={...	operational attribute	81

Fig. 9. Attributes defined for the printer object

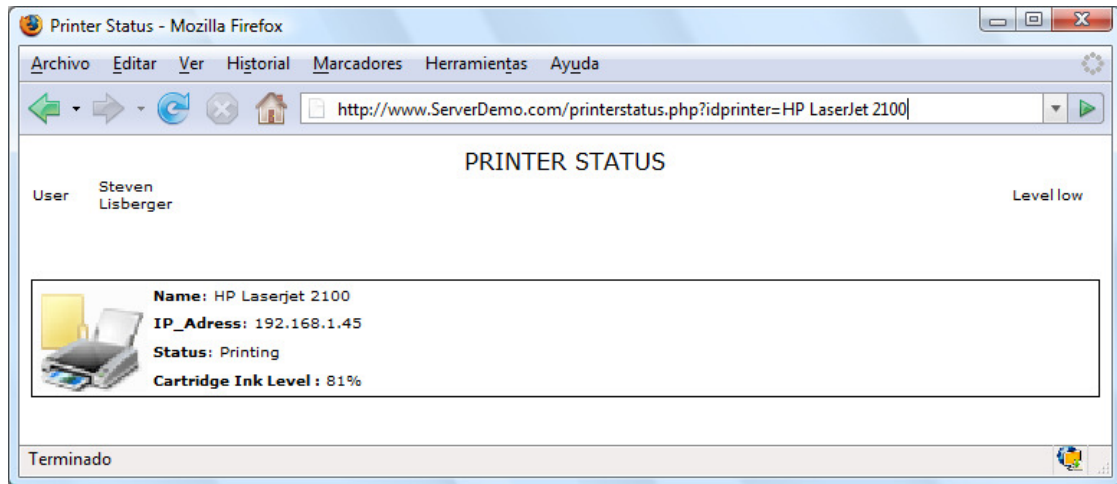


Fig. 10. Normal behavior of the application

an attribute discovering attack can be performed by making these following LDAP injections:

***(&(idprinter=HPLaserJet2100)(ipaddress=\*)) (objectclass=printer)***

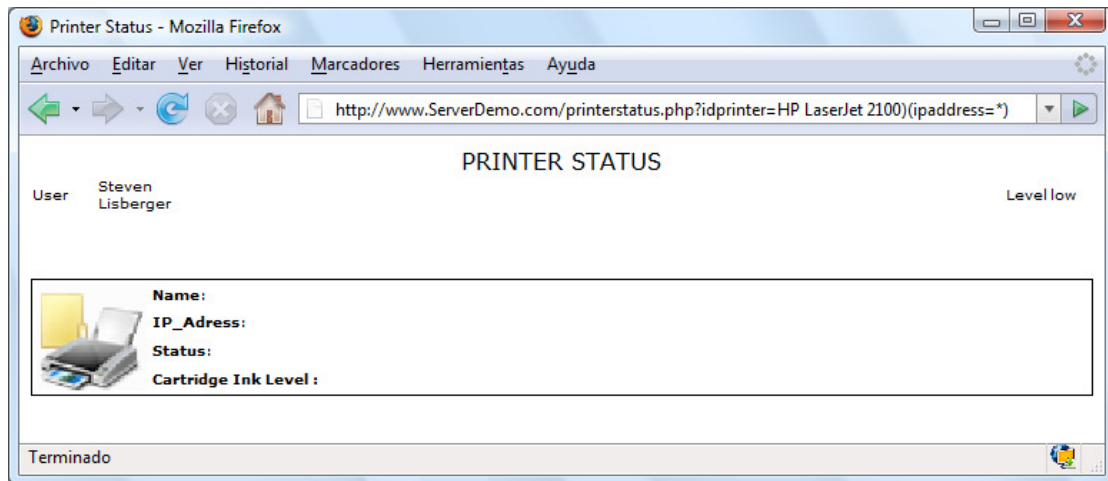


Fig 11: Response web page when the attribute does not exist

***( & (idprinter=HPLaserJet2100)(department=\*)) (objectclass=printer)***

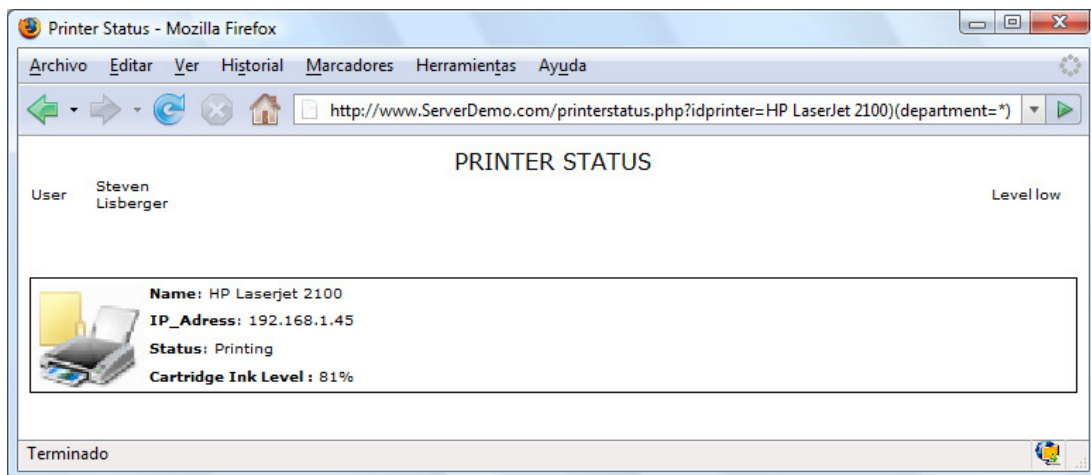


Fig 12: Response web page when the attribute exists

Obviously, the attacker can infer from these results which attributes exist and which do not. In the first case, the information about the printer is not given by the application because the attribute *ipaddress* does not exist or it is not accessible (FALSE). On the other hand, in the second case, the response web page shows the printer status and therefore, the attribute *department* exists in the LDAP directory and it is possible access to it.

Furthermore, with blind LDAP injection attacks the values of some of these attributes can be obtained. For example, suppose that the attacker wants to know the value of the department attribute: he can use booleanization and charset reduction techniques, explained in the next sections, to infer it.

### 5.3.2. Booleanization

An attacker can extract the value from attributes using alphabetic or numeric search. The crux of the idea is to transform a complex value (e.g. a string or a date) into a list of TRUE/FALSE questions. This mechanism, usually called booleanization, is summarized in figure 13 and can be applied in many different ways.

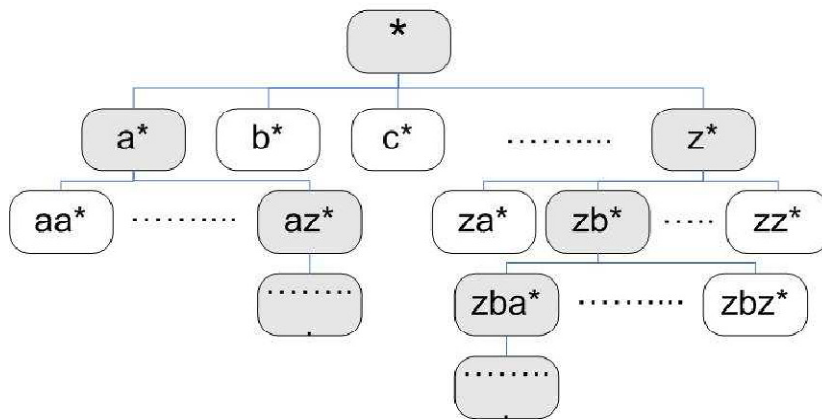


Fig. 13. Booleanization

Suppose that the attacker wants to know the value of the *department* attribute. The process would be the following:

```
(&(idprinter=HPLaserJet2100)(department=a*)(objectclass=printer))
```

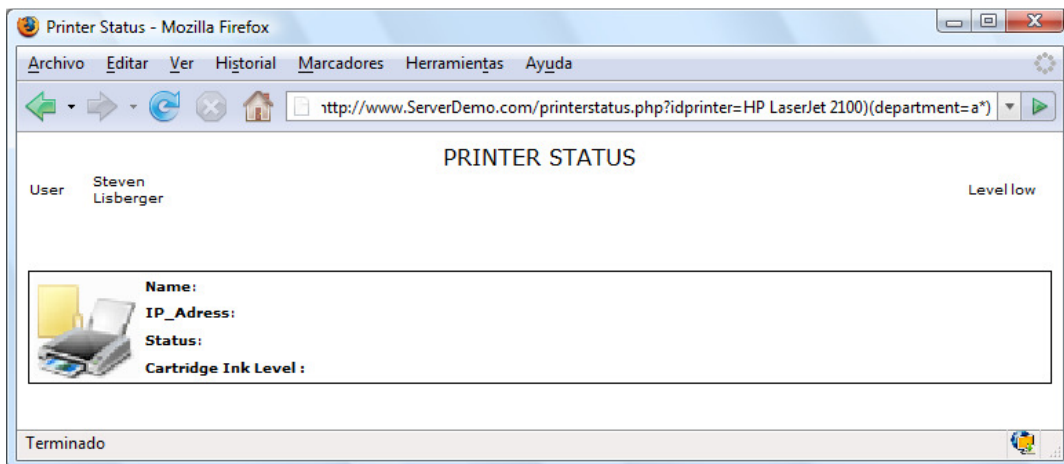


Fig. 14. FALSE. Value doesn't start with 'a'



`(&(idprinter=HPLaserJet2100)(department=f*))(objectclass=printer))`

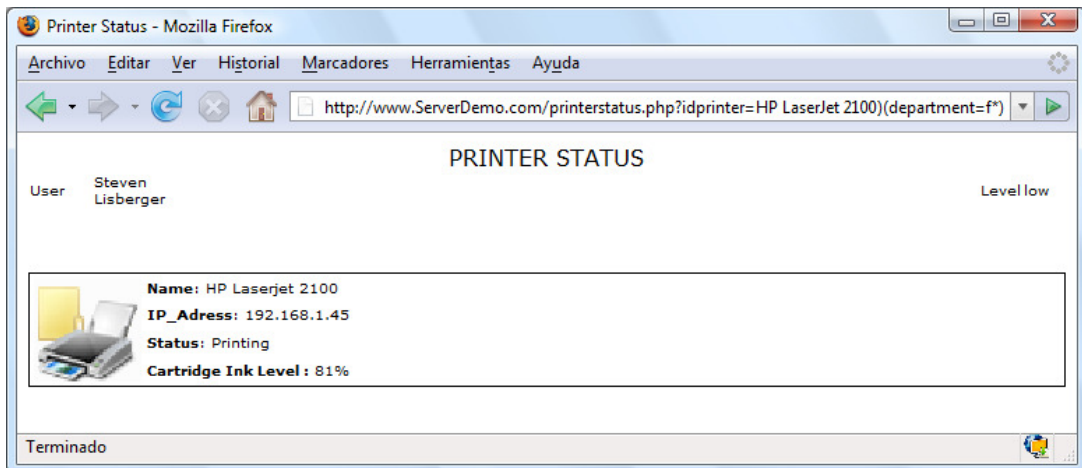


Fig. 15. TRUE. Value starts with 'f'

`(&(idprinter=HPLaserJet2100)(department=fa*))(objectclass=printer))`

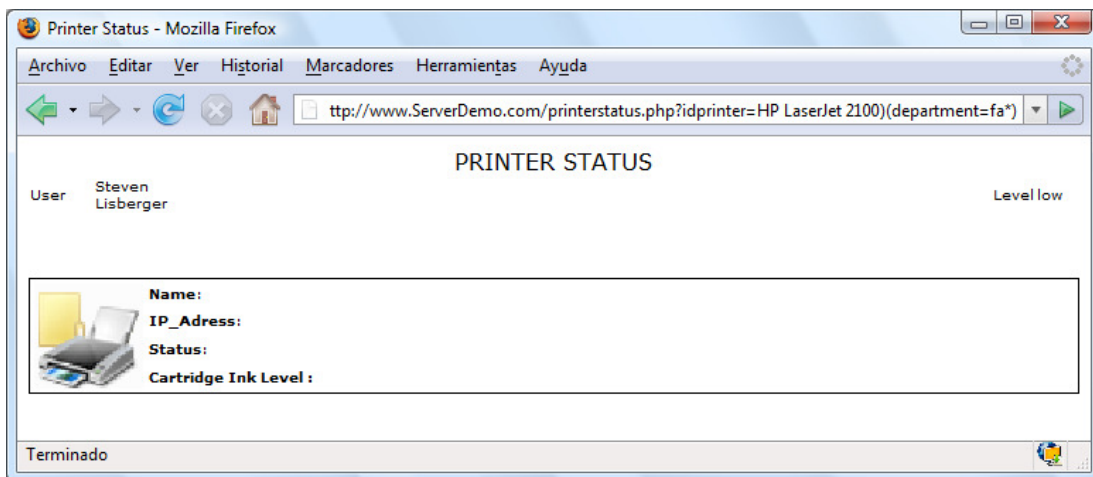


Fig. 16. FALSE. Value doesn't start with 'fa'

`(&(idprinter=HPLaserJet2100)(department=fi*))(objectclass=printer))`

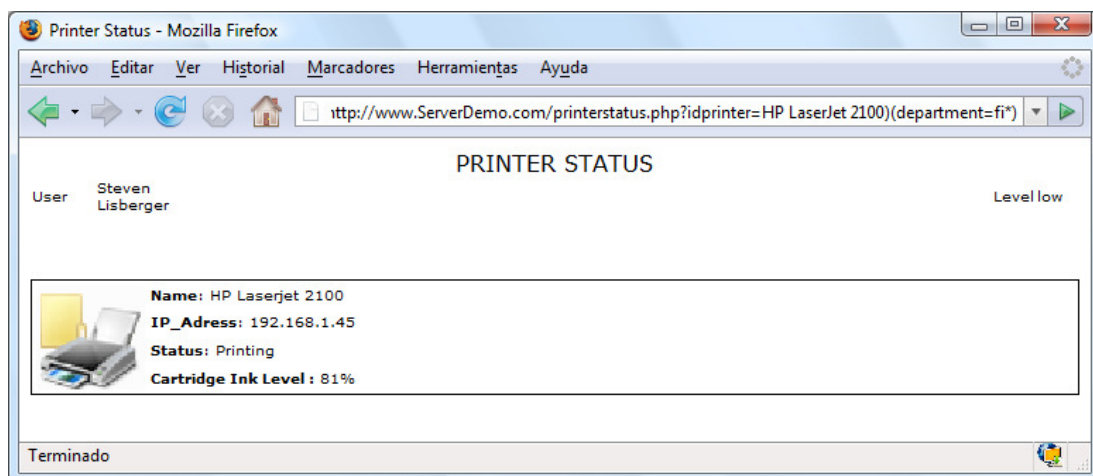


Fig. 17. TRUE. Value starts with 'fi'

As shown in figure 9, the *department* value in this example is *financiam*. The first try with the character 'a' does not obtain any printer information (figure 14) therefore, the first character is not an 'a'. After testing with the rest of the characters, the only one that obtains the normal behavior from the application is 'f' (figure 15). Regarding the second character, the only one that results in the normal operation of the application is 'i' (figure 17) and so on. Following the process, the department value can be obtained.

This algorithm can be also used for numeric values. In order to perform this, the booleanization process should use 'greater than or equal to' (>=) and 'less than or equal to' (<=) operators.

### 5.3.3 Charset Reduction

An attacker can use charset reduction to decrease the number of requests needed for obtain the information. In order to accomplish this, he uses wildcards to test if the given character is present *\*anywhere\** in the value, e.g.:

`(&(idprinter=HPLaserJet2100)(department=*b*))(objectclass=printer))`

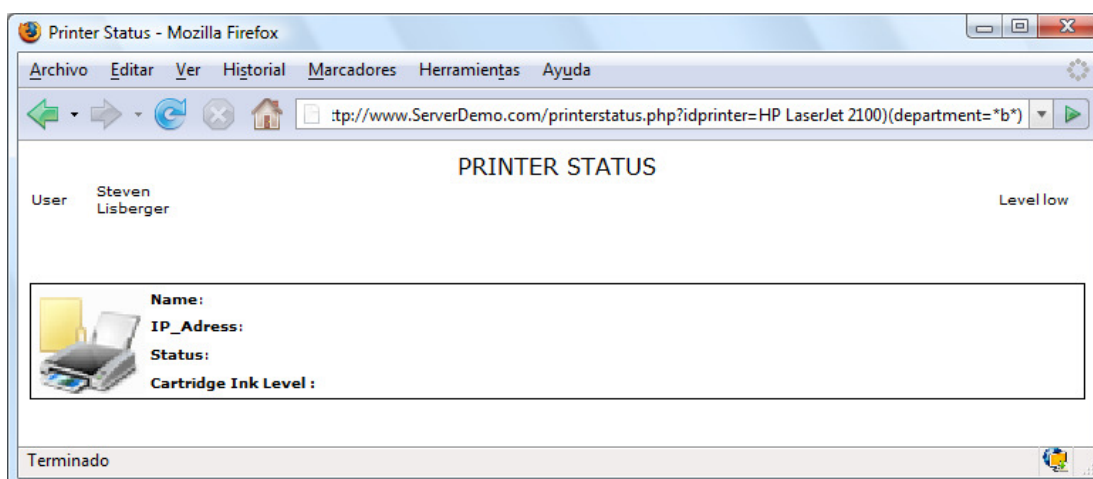


Fig. 18. FALSE. Character 'b' is not in the department value

`(&(idprinter=HPLaserJet2100)(department=*n*))(objectclass=printer))`

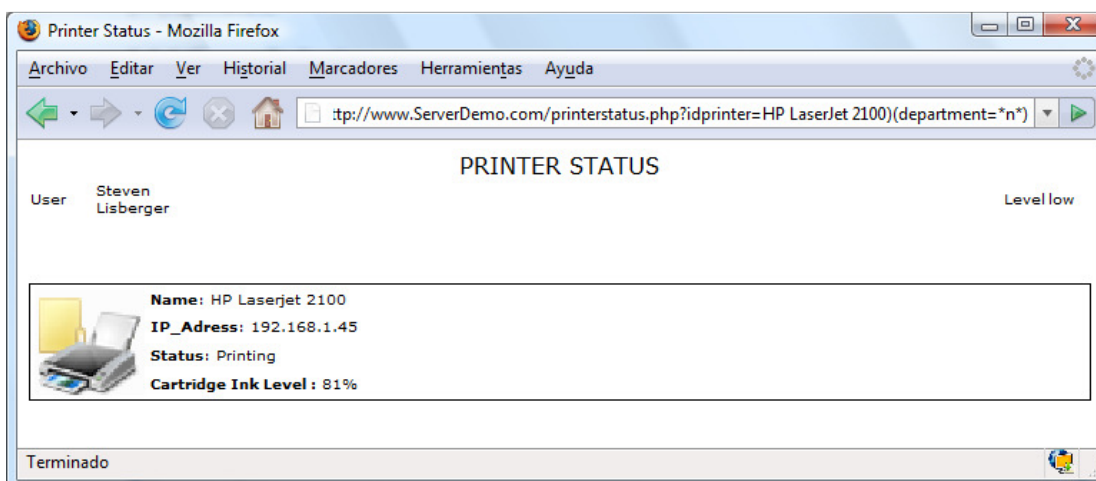


Fig. 19. TRUE. Character 'n' is in the department value



Figure 18 shows the response web page when the character 'b' is tested: no results are sent from the LDAP directory service so no letter 'b' is present, but in figure 19 a normal response web page is shown, meaning that the character 'n' is in the *department* value.

Through this process, the set of characters comprising the department value can be obtained. Once the charset reduction is done, only the characters discovered will be used in the booleanization process, thereby decreasing the number of requests needed.

## 6. Securing Applications against Blind LDAP Injection & LDAP Injection attacks

The attacks presented in the previous sections are performed on the application layer, therefore firewalls and intrusion detection mechanisms on the network layer have no effect on preventing any of these LDAP injections. However, general security recommendations for LDAP directory services can mitigate these vulnerabilities or minimize their impact by applying minimum exposure point and minimum privileges principles.

Mechanisms used to prevent code injection techniques include defensive programming, sophisticated input validation, dynamic checks and static source code analysis. The work on mitigating LDAP injections must involve similar techniques.

It has been demonstrated in the previous sections that LDAP injection attacks are performed by including special characters in the parameters sent from the client to the server. It is clear therefore that it is very important to check and sanitize the variables used to construct the LDAP filters before sending the queries to the server.

In conclusion, we see that parentheses, asterisks, logical (AND "&", OR "|" and NOT "!") and relational (=,>,<,<=,>=) operators must be filtered at the application layer.

Whenever possible, values used to construct the LDAP search filter must be checked against a list of valid values in the Application Layer before sending the query to the LDAP server.

## 7. References

- "LDAP Injection: Are your Web applications Vulnerable?", Sacha Faust. SPI Dynamics  
URL: <http://www.spidynamics.com/support/whitepapers/LDAPinjection.pdf>  
URL2: <http://www.networkdls.com/Articles/LDAPinjection.pdf>
- "Blind SQL Injection Automation Techniques", Cameron Hotchkies, BlackHat Conferences 2004.  
URL:<https://www.blackhat.com/presentations/bh-usa-04/bh-us-04-hotchkies/bh-us-04-hotchkies.pdf>
- "Blind XPath Injection", Amit Klein. SANCTUM.  
URL:[http://www.packetstormsecurity.org/papers/bypass/Blind\\_XPath\\_Injection\\_20040518.pdf](http://www.packetstormsecurity.org/papers/bypass/Blind_XPath_Injection_20040518.pdf)
- "Lightweight Directory Access Protocol (LDAP): The Protocol", J. Sermersheim, RFC 4511.  
URL: <http://www.rfc-editor.org/rfc/rfc4511.txt>
- "LDAP Search Filters", M. Smith & T. Howes, RFC 4515.  
URL: <http://www.rfc-editor.org/rfc/rfc4515.txt>
- "Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map", K. Zeilenga, Networking Working Group, LDAP Foundation, RFC, The OpenGroup.  
URL: <http://www.rfc-editor.org/rfc/rfc4510.txt>

- "Understanding LDAP - Design and Implementation", Steven Tuttle, Ami Ehlenberger, Ramakrishna Gorthi, Jay Leiserson, Richard Macbeth, Nathan Owen, Sunil Ranahandola, Michael Storrs & Chunhui Yang, IBM.  
URL: <http://www.redbooks.ibm.com/redbooks/pdfs/sg244986.pdf>
- "OpenLDAP admin Guide", OpenLDAP.org.  
URL: <http://www.openldap.org/doc/admin23/>
- "Microsoft Active Directory Technologies", Microsoft Corporation.  
URL: <http://www.microsoft.com/windowsserver2003/technologies/directory/activedirectory/default.aspx>
- "Novell E-Directory", Novell Corporation.  
URL: <http://www.novell.com/products/edirectory/>
- "RedHat Directory Services. RedHat Deployment Guide", RedHat.  
URL: <http://www.redhat.com/docs/manuals/dir-server/deploy/7.1/intro.html>
- "Single Sign-On", The OpenGroup.  
URL: <http://www.opengroup.org/security/sso/>
- "Oracle Internet Directory: Documentation", Oracle.  
URL: <http://www.oracle.com/technology/documentation/oid.html>
- "Time-Based Blind SQL Injection using heavy queries", Chema Alonso, Daniel Kachakil, Rodolfo Bordón y Antonio Guzmán.  
URL: <http://www.microsoft.com/technet/community/columns/secmvp/sv0907.aspx>

**Contact information:**

- Chema Alonso  
[chema@informatica64.com](mailto:chema@informatica64.com)  
<http://elladodelmal.blogspot.com>