Lecture 7: Web hacking 3, SQL injection, XPath injection, Server side template injection, File inclusion

Universitetet i Oslo
Laszlo Erdödi

Standard Query Language (SQL)
Dynamic websites can use large amount of data. If a website stores e.g. the registered users then it is necessary to be able to save and access the data quickly. In order to have effective data management data are stored in different databases where they are organized and structured. One of the most popular databases is the relational database. The relational databases have tables where each column describes a characteristics and each row is a new data entry. The tables are connected to each other through the columns. Example:

Standard Query Language (SQL)
For accessing or modifying or inserting data the database query languages are used. SQL (Standard Query Language) is the most popular language to manipulate the database content. SQL has a special syntax and operates with the following main commands:

- **SELECT** - extracts data from a database
- **UPDATE** - updates data in a database
- **DELETE** - deletes data from a database
- **INSERT INTO** - inserts new data into a database
- **CREATE DATABASE** - creates a new database
- **ALTER DATABASE** - modifies a database
- **CREATE TABLE** - creates a new table
- **ALTER TABLE** - modifies a table
- **DROP TABLE** - deletes a table
- **CREATE INDEX** - creates an index (search key)
- **DROP INDEX** - deletes an index
SQL command examples

- SELECT EmployeeID, FirstName, LastName, HireDate, City FROM Employees
- SELECT * FROM Employees
- SELECT EmployeeID, FirstName, LastName, HireDate, City FROM Employees WHERE City = 'London'
- SELECT column1, column2, ... FROM table_name WHERE columnN LIKE pattern;
- SELECT column_name(s) FROM table1 UNION SELECT column_name(s) FROM table2;
- SELECT * FROM Employees limit 10 offset 80

An sql tutorial can be found here: https://www.w3schools.com/sql/default.asp

SQL functional diagram

In order to use databases a db server (e.g. mysql, postgresql, oracle) should be run that is accessible by the webserver. It can be on the same computer (the db is running on localhost or on another computer).

Since the website needs to access and modify the database, all server side script languages support database commands e.g. database connect, database query.

SQL with php example

Php uses the
- mysql_connect,
- mysql_select_db,
- mysql_query,
- mysql_num_rows
- mysql_fetch_array
Etc. commands

SQL practice: Check your sql command

The following script prints out the generated sql query (it is only for demonstration, that never happens with real websites)

```sql
SELECT * FROM Table1 Where email='admin' AND Jelszo='12345' Name: admin
Password: 12345
Submit
```

And

```sql
SELECT * FROM Table1 Where email='admin' AND Jelszo='12345'
Name: admin
Password: 12345
Submit
```

SQL syntax error
Simple sql injection exploitation

The easiest case of sql injection is when we have a direct influence on an action. Using the previous example we can modify the sql query to be true and allow the login. With the ‘ or ‘1’=’1 (note that the closing quotation mark is deliberately missing, it will be placed by the server side script before the execution) the sql engine will evaluate the whole query as true because 1 is equal to 1 (1 now is a string not a number).

Normally attackers have to face much more complex exploitation. Usually the attacker has only indirect influence on the website action.

Type of sql injection exploitations

Based on the situation how the attacker can influence the server side sql query and the sql engine settings (what is enabled by the configuration and what is not) the attacker can choose from the following methods:

- **Boolean based blind**
  
  The attacker provided an input and observes the website answer. The answer is either page 1 or page 2 (only two options). There’s no direct response to the attacker’s query but it’s possible to play a true and false game using the two different responses. The difference between the two responses can be only one byte or totally different (see example later).

- **Error based**
  
  The attacker forces syntactically wrong queries and tries to map the database using the error messages.

- **Union query**
  
  The attacker takes advantage of the sql’s union select statement. If the attacker can intervene to the sql query then he can append it with a union select and form the second query almost freely (see example later).

- **Stacked query**
  
  If the sql engine supports stacked queries (first query; second query; etc.) then in case of a vulnerable parameter the attacker closes the original query with a semicolon and writes additional queries to obtain the data.

- **Time based blind**
  
  It is the same as the boolean based, but instead of having two different web responses the difference is the response time (less trustworthy).

- **Other options**
Type of sql injection exploitations

Besides that the attacker can obtain or modify the database in case of sql injection, the vulnerability can be used for further attacks as well if the db engine settings allow that:

- **Reading local files**
  The attacker can obtain data except for the database
- **Writing local files**
  With the `select into outfile` command the attacker can write local files
- **Executing OS commands**
  In some cases the db engine has the right to execute os level commands

Blind boolean based sqli exploitation

Depending on the input the attacker can see two different answers from the server. Example:

![](image)

If we provide a non-existing user e.g. `laszlo`, the first version of the page appears. For valid users such as `admin` (The attacker doesn't necessarily has valid user for the site) the second version appears.

Since there's no input validation for the email parameter, the attacker can produce both answers:

![True False](image)

Blind boolean based sqli exploitation

Ok, we can enumerate the users in that particular case, but how can we obtain the whole database with only true or false answers?

There are special table independent queries that always work for specific database engines (general queries for mysql, postgresql, etc.). For example for mysql we can use the following queries:

- **Mysql version:** `SELECT @@version`
- **Mysql user, password:** `SELECT host, user, password FROM mysql.user;`
- **Mysql databases:** `SELECT schema_name FROM information_schema.schemata;`
- **Mysql tables:** `SELECT table_schema,table_name FROM information_schema.tables WHERE table_schema != 'mysql' AND table_schema != 'information_schema'`

In order to execute such a query we need to arrange the current query to be accepted by the server side script (syntactically should be correct):

```
http://193.225.218.118/sql3.php?email=laszlo' or here goes the query or '1'=2
```

Since the vulnerable parameter was escaped with a quotation mark, the query should end with a missing quotation mark (the server side script will place it, if there's no missing quotation mark, the query will be syntactically wrong).

The second part of the query should be boolean too, e.g.:

```
http://193.225.218.118/sql3.php?email=laszlo' or ASCII(Substr((SELECT @@VERSION),1,1))<64 or '1'=2
```

The previous query checks if the ASCII code of the first character of the response of `SELECT @@VERSION` is less than 64.

Task: Find the first character of the db version!
Exploitation with sqlmap

Several tool exists for automatic sql injection exploitation. Sqlmap is an advanced sqli tool. The first step is to check if sqlmap manages to identify the vulnerable parameters.

If sqlmap has identified the vulnerability the attacker could ask for specific data:
- --dbs: the databases in the db engine
- -D selecteddb --T selectedtable --columns: the columns in the selected table of the selected database
- -D selecteddb --T selectedtable --dump: all data in the selected table of the selected database

Writing local files with sql injection

Instead of asking for boolean result the attacker can use the `select into outfile` syntax to write a local file to the server. Since this is a new query the attacker has to chain it to the vulnerable first query (union select of stacked query exploitation). This is only possible if the following conditions are fulfilled:
- Union select or stacked queries are enabled
- With union select the attacker has to know or guess the row number and the types of the chained query (see example)
- A writable folder is needed in the webroot that later is accessible by the attacker
- The attacker has to know or guess the webroot folder in the server computer

Example:
```
http://193.225.218.118/sq13.php?email=laszlo' union select 'Imagine here's the attacking script' '0', '0', '0' into outfile '/var/www/temp/lennon.php
```
Sql injection filter evasion techniques

- White Space  `or 'a' = 'a'`
- Null Bytes  `%00' UNION SELECT password FROM Users WHERE username='admin'--`
- SQL Comments
  `/**UNION/**SELECT/**/password/**/FROM/**/Users/**/WHERE/**/name/**/LIKE/**/''admin'--`
- URL Encoding
  `%27%20UNION%20SELECT%20password%20FROM%20Users%20WHERE
  %20name%3D%27admin%27--`
- Character Encoding  `UNION SELECT password FROM Users WHERE
  name=char(114,111,111,116)--`
- String Concatenation  `EXEC('SEL' + 'ECT 1')`
- Hex Encoding  `Select user from users where name = unhex('726F6F74')`

Xpath injection

Instead of storing datasets in databases, data can be stored in xml format.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<users>
    <user>
        <name>John</name>
        <email>johndoe@example.com</email>
        <password>123456</password>
        <fullname>John Doe</fullname>
    </user>
</users>
```

Example task:


Get the admin user’s email (flag)!

Xpath query with php

Xpath can be used to make a query, e.g. finding the full name of the user
whose username is john and the password is imagine:

```php
$xml->xpath("/users/user[name='john' and password='imagine']/fullname")
```

Finding the first user in the database:

```php
$xml->xpath("/users/user[position()=1]/fullname")
```

Finding the penultimate user:

```php
$xml->xpath("/users/user[last()-1]/fullname")
```

Other xpath functions can be used as well:

`last()`, `count(node-set)`, `string()`, `contains()`, etc.

The full xpath reference is here:

https://docs.oracle.com/cd/E35413_01/doc.722/a35419/dev_xpath_functions.htm

Xpath injection

Xpath injection is possible when there’s no input validation or the
validation is inappropriate in the xpath query, e.g.

```php
$search = $this->xpath("/users/user[name='3.0_POST[username]'." and password='3.0_POST[password]'.'/fullname']");
if (count($search) == 0) {
    print("Hello ", $fullname, ");
    $sql = $this->xpath("/users/user[name='3.0_POST[username]'." and password='3.0_POST[password]'.']/email']");
    $sql->update($sql[0]);
}
```

The exploitation of the vulnerability looks like an sql injection exploitation:

```
Name: [input type='text' name='name' value='']
Password: [input type='password' name='password' value='']
Submit
```


https://media.blackhat.com/bh-eu-12/Siddharth/bh-eu-12-Siddharth-Xpath-WP.pdf
Server Side Template Injection (SSTI)

Template engines are widely used by web applications to present dynamic data via web pages. Unsafely embedding user input in templates enables Server-Side Template Injection. Example:

\[
\text{Output} = \text{Twig} -> \text{render}(\text{"Dear \(\text{first\_name}\),"}, \text{array}(\text{"first\_name\"} \Rightarrow \$\text{user\_first\_name}\));
\]

If a user input is substituted as template parameter without proper validation then the vulnerability appears:

\[
\text{Output} = \text{Twig} -> \text{render}(\$_\text{GET}[\text{"custom\_email"]}, \text{array}(\text{"first\_name\"} \Rightarrow \$\text{user\_first\_name}\));
\]

After detecting the vulnerability the next step is to identify the template engine that was used (e.g. Smarty, Twig, Jade). Each template engine has specific exploitation. In case of a successful exploitation the attacker can even execute arbitrary shell commands.

More details can be found here: https://portswigger.net/blog/server-side-template-injection

Local File Inclusion

Local file inclusion (LFI) is a vulnerability when the attacker can include a local file of the webserver using the webpage. If the server side script uses an include file type of method and the input for the method is not validated then the attacker can provide a filename that points to a local file:

**Task:** Find the flag inside the /etc/flag/index file!

Exploitation of the LFI vulnerability

Adding null character at the end of the directory sometimes works when the normal exploitation fails:

Exploitation of the LFI vulnerability

In addition to obtaining local files an additional aim is to upload attacking scripts and execute commands.

Depending on the server and the php settings executing php scripts can be possible if the local file is the: `php://input` and the php script is the posted data:

In other cases providing except as file will execute the desired OS command, e.g.: http://193.225.218.118/flag.php?COLOR=expect://ls
Exploitation of the LFI vulnerability

A php script source cannot be obtained through a browser, because the script is executed on the server side. But using encoding and php://filter as input the server side scripts can be obtained too. Since Php 5.0.0 the php://filter/convert.base64-encode/resource function is enabled. It encodes the php file with base64 and the php script source reveals.

Find the flag here: http://193.225.218.118/lfi2.php?COLOR=whatever

Exploitation of the LFI vulnerability

The most frequently used way for writing files to the server is to write the script in a local file first, then read it back through the LFI vulnerability. How can the attacker place his own attacking script in a local file?

One option is to access the /proc/self linux folder /proc/self/environ contains the current process info including the HTTP_USER_AGENT. If the attacker places the attacking script inside the user agent of the http head and the webserver has the right to access the /proc/self/environ file then he can execute any OS command in the name of the webserver application.

Note! Do not run the webserver as root! If the webserver is compromised and can be forced to execute commands then the command has the same rights as the server (the code is executed in the name of the server).

Exploitation of the LFI vulnerability

If the environ file is not accessible by the webserver then the attacker can try to find the webserver processid and access the environ file through the processid.

Exploitation of the LFI vulnerability

The attacker can also try to find the user agent by /proc/self/fd/ and brute-forcing the number (usually 12 or 14 in Apache)

/procj/self/fd/12/procj/self/fd/14%00/procj/self/fd/12/procj/self/fd/14%00/procj<apache_id>/fd/12/procj<apache_id>/fd/14%00/procj<apache_id>/fd/12%00/procj<apache_id>/fd/14%00
Exploitation of the LFI vulnerability

If the logs are accessible through the web server then the attacker can place the attacking PHP script in the logs to be executed in the same way as in the case of the `/proc/self` folder. The logs can be in various places, one option is to check `/var/log/apache2` folder:

The attacker can influence the source IP, the web method, the HTTP version, the URL and the browser data in the logs. The easiest way is to modify the browser data (type of browser), because it’s a string, so PHP functions such as `system()` or `phpinfo()` can be substituted:

Instead of `phpinfo()`, it's better to use the `system()` PHP command:

In this way the attacking script can be uploaded. If the log file is too long then the browser will not be able to display the logs.

Remote File Inclusion

If the PHP settings allow, remote files can be inserted to the page. PHP settings relevant to remote inclusion:

- `allow_url_fopen`: open file with `fopen`
- `allow_url_include`: include, `include_once`, `require` and `require_once`

If the attacker can include remote files he will be able to include attacking scripts that are stored on an attacker controlled web server.
End of lecture