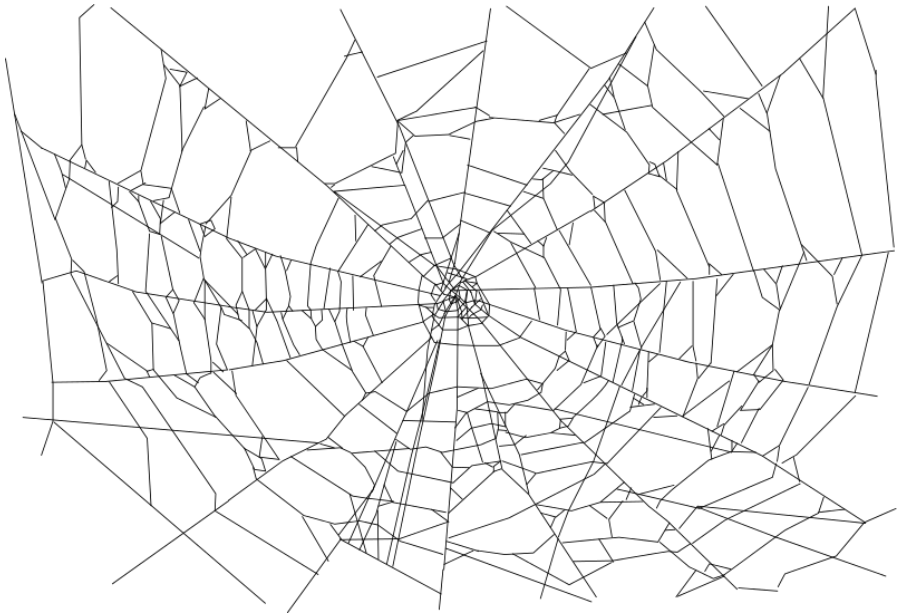
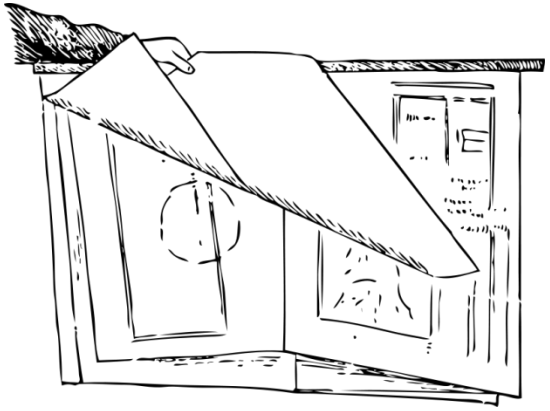
Web-application security



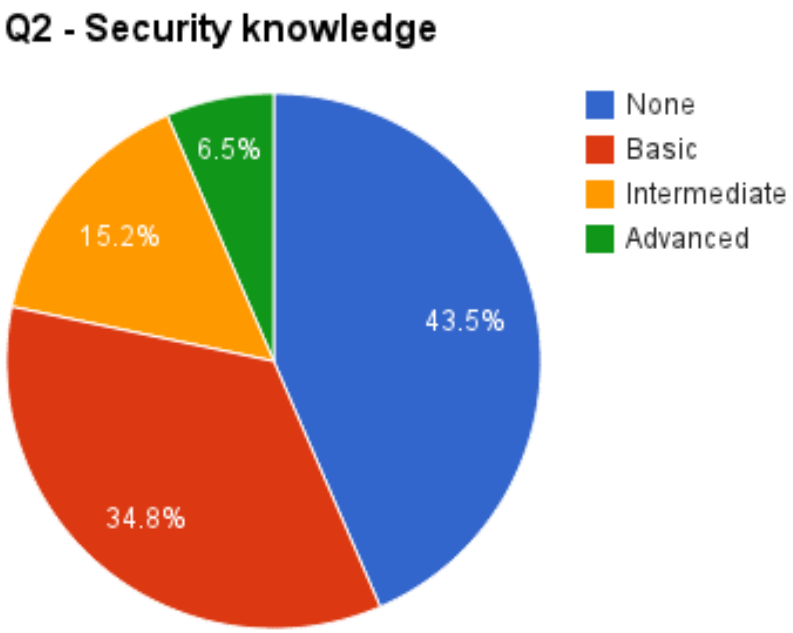
/*Secure Coding - WS 2014
LS XXII – Chair for Software Engineering
Technische Universität München*/

Lecture plan



1. Intro/Motivation
2. **Web. app. security**
3. Memory management and C/C++
4. *Presentations Phase 1 (Developing)*
5. *NO LECTURE (SVV)*
6. Countermeasures
7. *Presentations Phase 2 (Black-box testing)*
8. White-box Vulnerability detection
9. *Presentations Phase 3 (Fixing + new features)*
10. Java security
11. Reverse engineering
12. Obfuscation
13. *Presentations Phase 4 (White-Box testing/RE)*
14. Crypto / Code review / Advanced / Recap
15. *Presentations Phase 5 (Fixing + final report)*

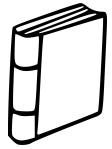
Some stats



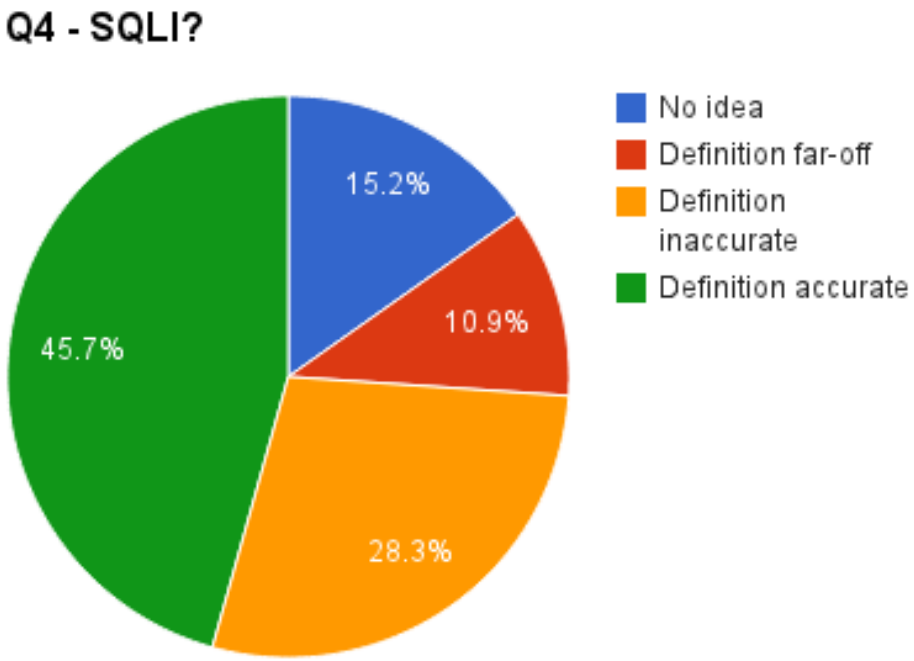
If you are not following *ask*, if you have something interesting to share, *participate*!

A recommended source for everybody:

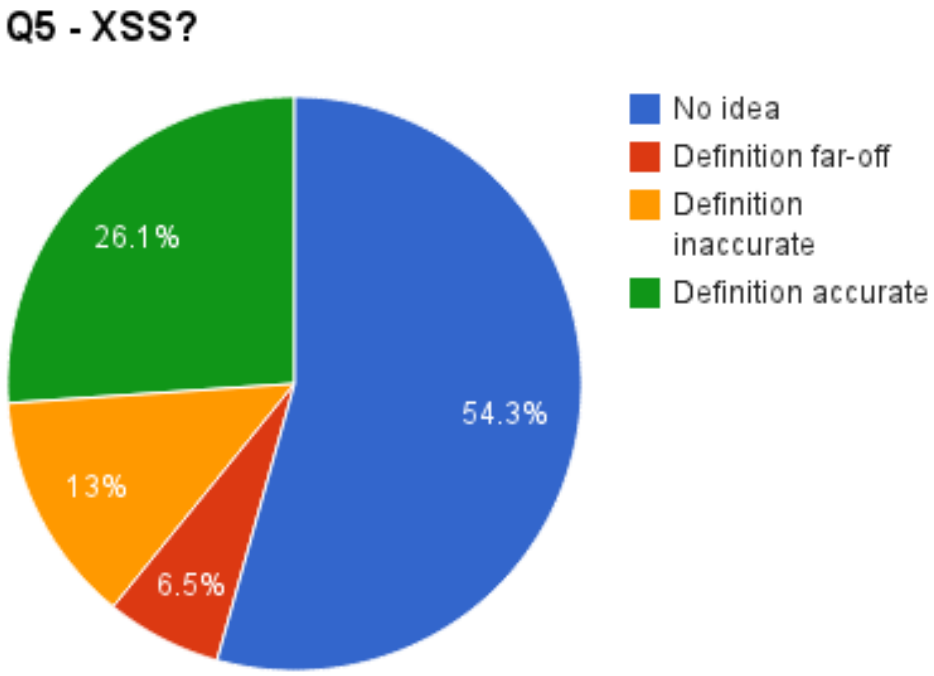
http://www.cl.cam.ac.uk/~rja14/book.html



Some stats


Q4 - SQLI?
15.2%
10.9%
28.3%
45.7%
No idea
Definition far-off
Definition inaccurate
Definition accurate

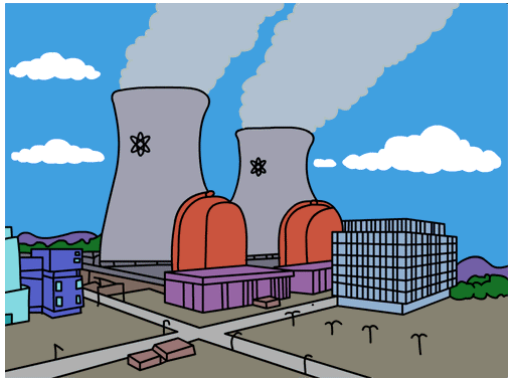
Some stats


Q5 - XSS?
No idea
Definition far-off
Definition inaccurate
Definition accurate
54.3%
6.5%
13%
26.1%

Motivation

Should be clear, right?

And remember…



HTTP basics

- Hypertext Transfer Protocol, earlier versions around 1991
- RFC 2616, request and response protocol (IETF,June 1999)
- Recent update (RFC 7230-35, 2014)
- TCP connection, no implicit HTTP state
- Client/Server architecture.
- Implementations often very tolerant to protocol nonconformities

Request

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

Response

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 131

<html>
<head>
  <title>An Example Page</title>
</head>
```

HTTP basics

- Some HTTP request methods:
  - **GET:** Access a resource (read). In theory, no side effects…
  - **POST:** Used to pass data to a specific resources (i.e. send form data).
  - **PUT:** Upload data to specific URI.
  - **DELETE:** Deletes data in specific URI.
  - **TRACE:** Echo request to client (to detect modifications on the way).

- Some response codes:
  - **2xx**: Success
  - **3xx**: Redirection
  - **4xx**: Client error
  - **5xx**: Server error

For more details read:

http://tools.ietf.org/html/rfc2616

No State - What to do?

- Lots of “work around”’s.
- Two basic ways:
  - Store state in parameter and pass it as part of a GET or POST request.

```
http://example.com/over/there?name=var&lastname=var2&…
```

  - Store state in cookie.

Cookies

Source: Wikimedia Commons

Client-side scripting

- Traditionally HTTP was meant only to retrieve and display “static” material, such as text and pictures.
- To enrich the client-side functionality, Netscape developed “JavaScript” (unrelated really to Java).
- Goal: provide basic functionality and manipulate rendered HTML without sending new requests to the server (or to do this in a transparent way to the user).

```
<meta charset="utf-8">
<title>Minimal Example</title>

<h1 id="header">This is JavaScript</h1>

<script>
    document.body.appendChild(document.createTextNode('Hello World!'));

    var h1 = document.getElementById('header'); // holds a reference to the <h1> tag
    h1 = document.getElementsByTagName('h1')[0]; // accessing the same <h1> element
</script>
```

Vulnerable applications for didactical purposes

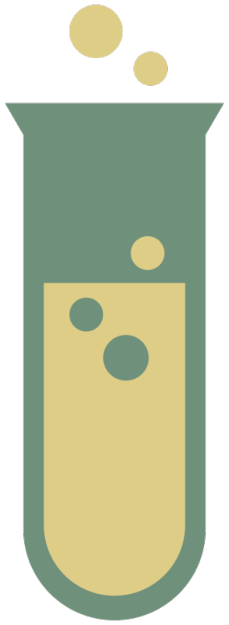
- For learning and playing around, use deliberately insecure local test applications
  - BadStore, DVWA, Hacme Casino, WebGoat, Google Gruyere, samurai dojo etc.
- We will use the Samurai Web Testing Framework (WTF)

  VM with tools and vulnerable applications

http://samurai.inguardians.com/

Warm up

- Demo!
  - HTTP and Browser basics



OWASP Top 10

- OWASP: Open Web Application Security Project
  - Community-driven
  - Hosts many projects and provides tools in the field of web security
  - Regular community events:

https://www.owasp.org/index.php/OWASP_German_Chapter_Stammtisch_Initiative/München

- OWASP Top 10
  - Last release in 2013
  - Goals: “identifying some of the most critical risks“ “to raise awareness about application security“
  - Scientifically not very accurate as a taxonomy (some overlaps). An alternative classification that subsumes this one “Seven Pernicious Kingdoms” Tsipenyuk et al. https://cwe.mitre.org/documents/sources/SevenPerniciousKingdoms.pdf

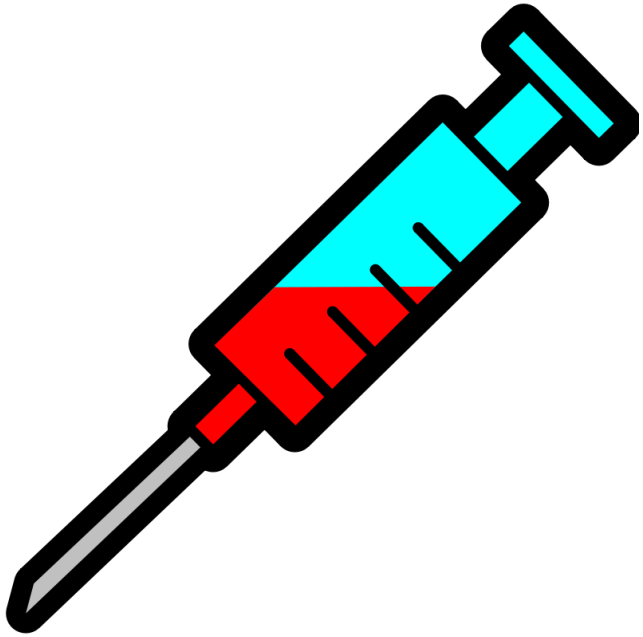
OWASP Top 10 updated on 2013

1 – Injection
2 – Broken Authentication and Session Mgmt.
3 – Cross-site scripting
4 – Insecure direct object references
5 – Security misconfiguration
6 – Sensitive data exposure
7 – Missing function level access control
8 – Cross-site request forgery
9 – Using components with known vulnerabilities
10 – Unvalidated redirects and forwards

See more on https://www.owasp.org/index.php/Top_10_2013-Top_10

OWASP Top 10 2013

1 – Injection
2 – Broken Authentication and Session Mgmt.
3 – Cross-site scripting
4 – Insecure direct object references
5 – Security misconfiguration
6 – Sensitive data exposure
7 – Missing function level access control
8 – Cross-site request forgery
9 – Using components with known vulnerabilities
10 – Unvalidated redirects and forwards



SQL Injection - The power of ‘;

Still one of the most exploited vulnerabilities in practice, despite being well known for almost a decade.

Many spectacular attacks have been performed using SQLi:

- On March 27, 2011, mysql.com, the official homepage for MySQL, was compromised by a hacker using SQL blind injection[40]
- On April 11, 2011, Barracuda Networks was compromised using an SQL injection flaw. Email addresses and usernames of employees were among the infor
- Over a period of 4 hours on April 27, 2011, an automated SQL injection attack occurred on Broadband Reports website that was able to extract 8% of the us accounts.[42][43][44]
- On June 1, 2011, "hacktivists" of the group LulzSec were accused of using SQLI to steal coupons, download keys, and passwords that were stored in plainte
- In June 2011, PBS was hacked, mostly likely through use of SQL injection; the full process used by hackers to execute SQL injections was described in this
- In May 2012, the website for *Wurm Online*, a massively multiplayer online game, was shut down from an SQL injection while the site was being updated.[48]
- In July 2012 a hacker group was reported to have stolen 450,000 login credentials from Yahoo!. The logins were stored in plain text and were allegedly taker injection technique".[49][50]
- On October 1, 2012, a hacker group called "Team GhostShell" published the personal records of students, faculty, employees, and alumni from 53 universitie pastebin.com. The hackers claimed that they were trying to “raise awareness towards the changes made in today’s education”, bemoaning changing educati
- On June 27, 2013, hacker group "RedHack" breached Istanbul Administration Site.[52] They claimed that, they've been able to erase people's debts to water, password for other citizens to log in and clear their debts early morning. They announced the news from Twitter.[53]
- On November 4, 2013, hacktivist group "RaptorSwag" allegedly compromised 71 Chinese government databases using an SQL injection attack on the Chine Anonymous.[54]
- On March 7, 2014, officials at Johns Hopkins University publicly announced that their Biomedical Engineering Servers had become victim to an SQL injectior The hackers compromised personal details of 878 students and staff, posting a press release and the leaked data on the internet.[55]
- In August 2014, Milwaukee-based computer security company Hold Security disclosed that it uncovered a theft of confidential information from nearly 420,0 check the claim.[57]

SQL Injection

What is wrong with this code?

```
$email= $_GET['email'];
$passwd=$_GET['passwd'];
...
$query="SELECT * FROM db WHERE email='$email' AND
passwd='$passwd'";
```

SQL Injection

Problem: The application does not properly separate database statements from user data

```
$email= $_GET['email'];
$passwd=$_GET['passwd'];
...
$query="SELECT * FROM db WHERE email='$email' AND
passwd='$passwd'";
```

Expected use

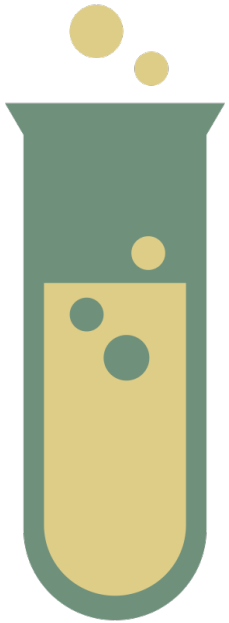
```
SELECT * FROM db WHERE email='user@host.com' AND
passwd='mypwd';
```

Example attack

```
SELECT * FROM db WHERE email='test' OR email='victim@host.com';
-- AND passwd='wtvr';
```

SQL Injection

- Demo!
  - SQLi on DVWA

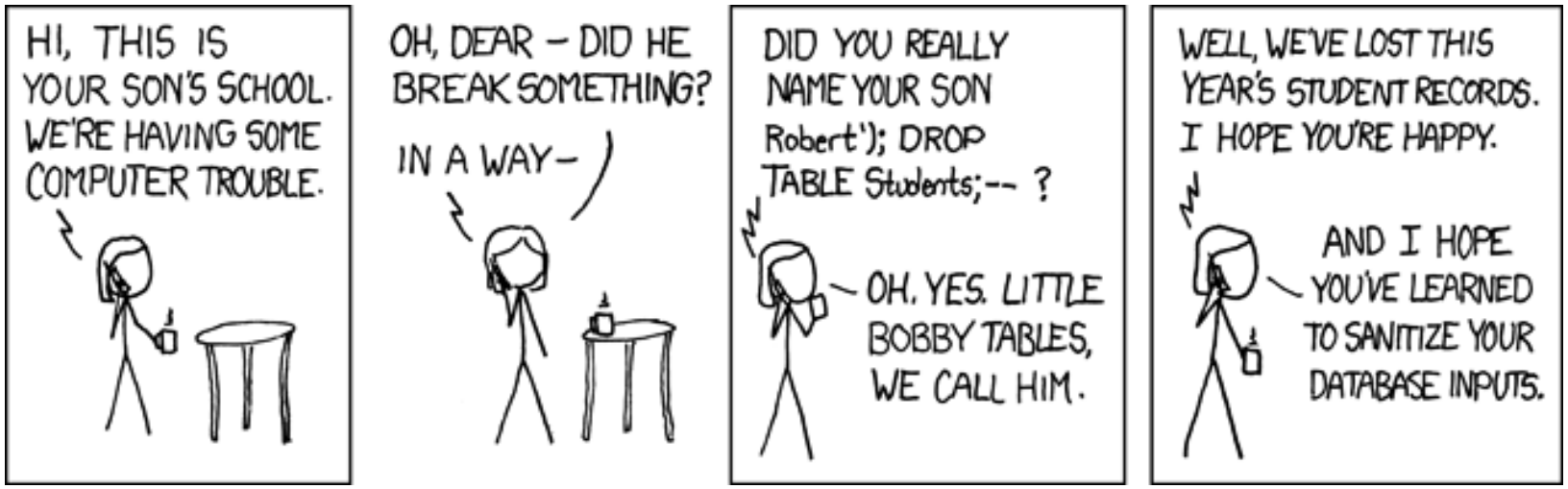


SQL Injection

- Impact
  - Read and/or write access to all database contents
    - Read user information or other data not meant to be public
    - Change information to the benefit of the attacker
    - (Run operating system commands, e.g. at MSSQL Server)
    - Sometimes an attacker can also access data of other applications on the same database server
- Useful for testing: Cheat sheets
  - For MySQL a good one is
    - http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
- Tool support? Lab!

SQL Injection

- Solutions? How to get it right?
  - Soon!



(taken from xkcd.com)

OWASP Top 10 2013

1 – Even more Injections

2 – Broken Authentication and Session Mgmt.

3 – Cross-site scripting

4 – Insecure direct object references

5 – Security misconfiguration

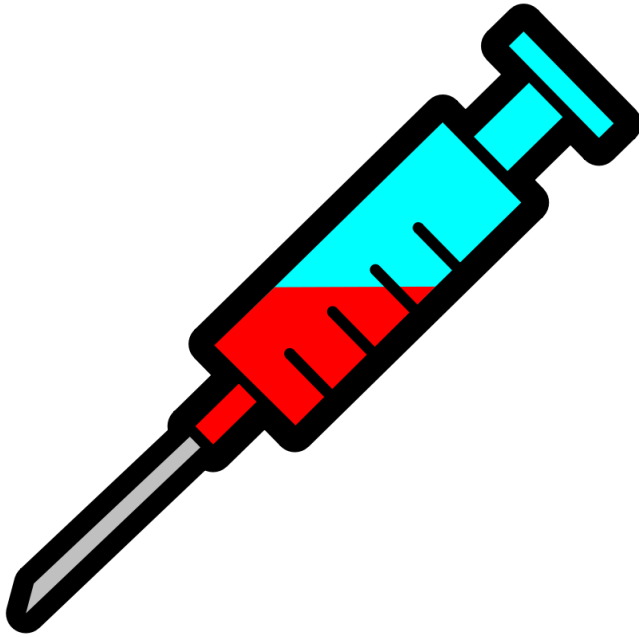
6 – Sensitive data exposure

7 – Missing function level access control

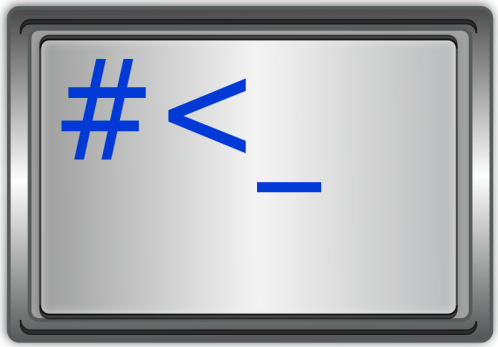
8 – Cross-site request forgery

9 – Using components with known vulnerabilities

10 – Unvalidated redirects and forwards



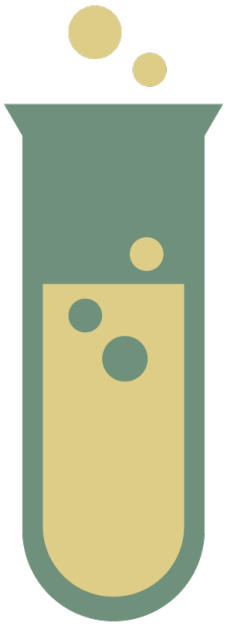
Other injections



- More interpreters to inject data
  - OS shell, LDAP, XPath, …
- Example: user data gets part of an OS command (bash, cmd.exe, etc.)
  - OS commands often used for e.g. resizing uploaded images, copying files, starting additional services, …
- Impact: Depends on the interpreter and the data the application processes
  - Most likely full system compromise in case of OS command injection

Other injections

- Demo!
  - Command injection on DVWA



OWASP Top 10 2013

1 – Even more Injections

2 – Broken Authentication and Session Mgmt.

3 – Cross-site scripting

4 – Insecure direct object references

5 – Security misconfiguration

6 – Sensitive data exposure

7 – Missing function level access control

8 – Cross-site request forgery

9 – Using components with known vulnerabilities

10 – Unvalidated redirects and forwards

OWASP Top 10 2013

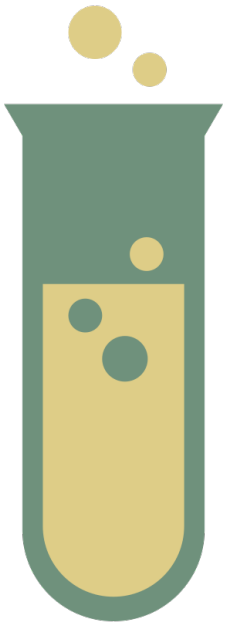
- Badly implemented access control
- For instance index.php:

- To access please
  - Login
  - Password
    - If correct => redirect to secret.php

- If no AC in place one could go directly to *secret.php* without login in!

Broken authentication

- Demo!
  - Broken authentication in WebGoat



OWASP Top 10 2013

1 – Even more Injections
2 – Broken Authentication and Session Mgmt.
3 – Cross-site scripting
4 – Insecure direct object references
5 – Security misconfiguration
6 – Sensitive data exposure
7 – Missing function level access control
8 – Cross-site request forgery
9 – Using components with known vulnerabilities
10 – Unvalidated redirects and forwards

XSS

What is wrong in this very simple PHP snippet?

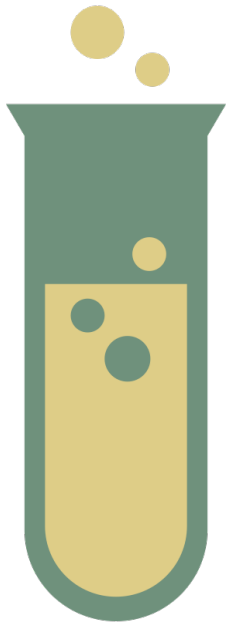
```
echo “Welcome! $_GET[‘name']";
```

XSS

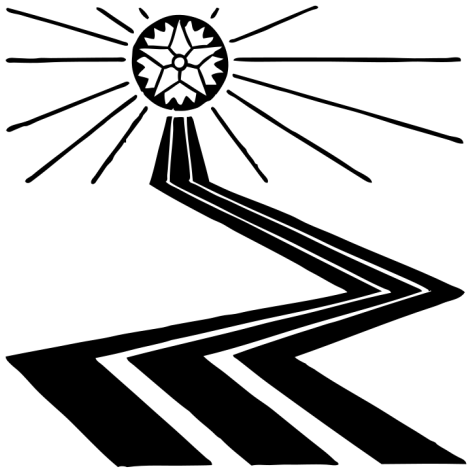
- Attacker injects JavaScript code that gets later executed in the browser of web site visitors
- Problem
  - User data that contains HTML markup is not properly escaped and included into the web servers response.
  - The browser renders the code of the attacker as part of the web page to be displayed
- Impact?

XSS

- Demo!
  - Reflected and stored XSS on DVWA



OWASP Top 10 2013



1 – Even more Injections
2 – Broken Authentication and Session Mgmt.
3 – Cross-site scripting
4 – Insecure direct object references
5 – Security misconfiguration
6 – Sensitive data exposure
7 – Missing function level access control
8 – Cross-site request forgery
9 – Using components with known vulnerabilities
10 – Unvalidated redirects and forwards

Insecure object references

- What is wrong in this code?

```
$input=$_GET['logfile'];
include ("/var/www/logs/".$input);
```

Path traversal / File disclosure

- Normal use:

```
showlog.php?logfile=log.txt
```

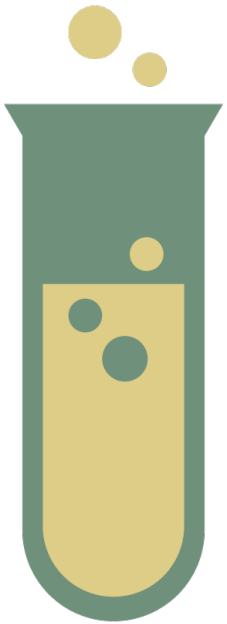
- Example attack:

```
showlog.php?logfile=../../../../../customers.xml
```

- Impact
  - Disclosure of all files the web server process has read permissions for, e.g. password files, configuration files, user data, etc.

Insecure object references

- Demo!
  - File disclosure and path traversal in DVWA



Cross-site request forgery

1 – Even more Injections
2 – Broken Authentication and Session Mgmt.
3 – Cross-site scripting
4 – Insecure direct object references
5 – Security misconfiguration
6 – Sensitive data exposure
7 – Missing function level access control
8 – Cross-site request forgery
9 – Using components with known vulnerabilities
10 – Unvalidated redirects and forwards

Cross-site request forgery

- Remember, to keep state, session variables sent in every request, usually stored in cookies.

I want to see my profile, SessionID = 123123123123

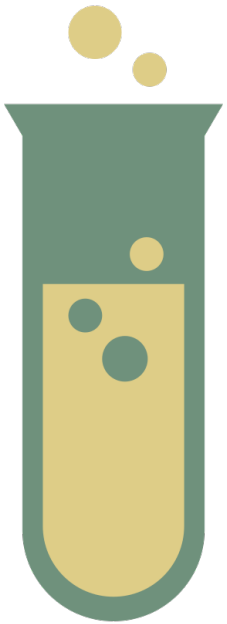
- The browser sends cookies automatically to the “owner” server.
- What happens if an authenticated in server X user gets tricked into visit a webpage containing an HTTP request to server X? The browser will send the cookies to X, since the request originated in the authenticated user's browser.

Send 100 EUR to Eve, SessionID = 123123123123

Malicious URL

CSRF

- Demo!
  - CSRF in DVWA



CSRF

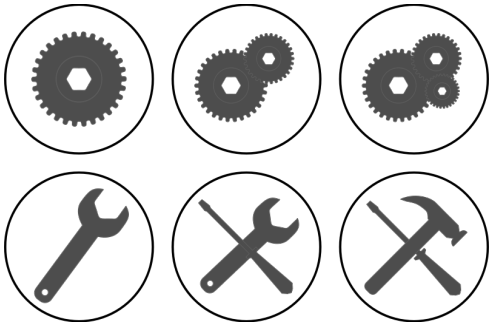
- It is possible to create proof of concept automatically from an HTTP request, for instance using Piñata

http://security-sh3ll.blogspot.com/2010/04/pinata-csrf-poc-html-generation-tool.html

- or OWASP's CSRF Tester

- You can play around with DVWA, CSRF!

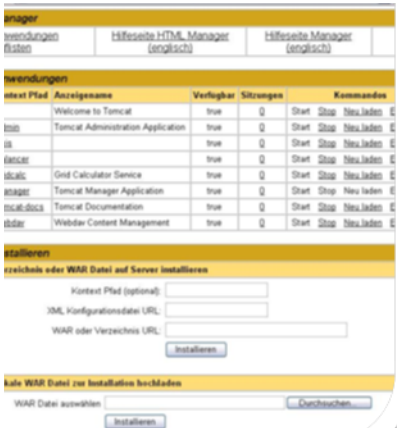
Other common issues

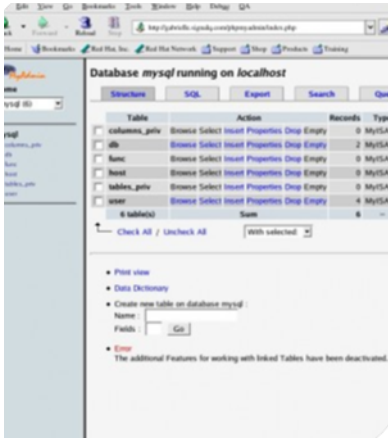


- **Security misconfiguration**
  - *Directory listings*: reveal which files are stored on a server
    - quite often debugging functionality, unreferenced content or forgotten files can be found
  - *Hidden content* e.g. backup files
    - Admins rely on that you cannot find something when there is no hyperlink to it
- **Software versions with known vulnerabilities**
    - Systems were set up once but never updated again
    - Sometimes ready-made exploit code is publicly available

Other common issues

- Freely accessible management interfaces: No need to compromise the application when the whole server can be managed directly
  - JBoss JMX Console
  - Tomcat manager
  - PHPmyadmin
  - …





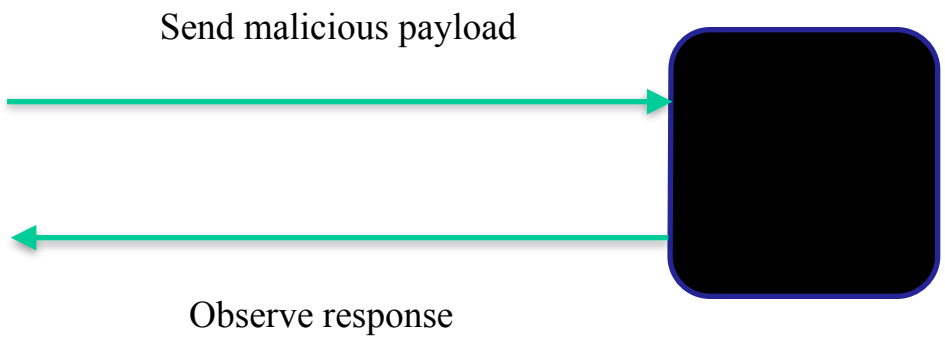
Other common issues

- User or admin accounts with weak passwords
  - Often there is no password complexity enforcement
  - Brute force known accounts with a list of common passwords
  - Tool support: THC Hydra

- Web sites that have a login function and display confidential information often do not use HTTPS
  - All data is therefore transferred in plain text
  - Information can be sniffed by an attacker on the same network

How to automatize black-box testing?

- For injections, schematically:

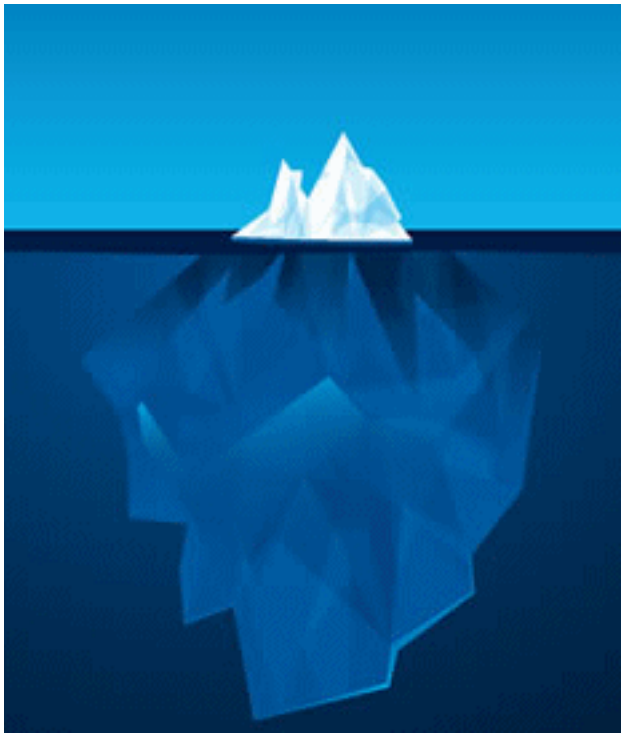


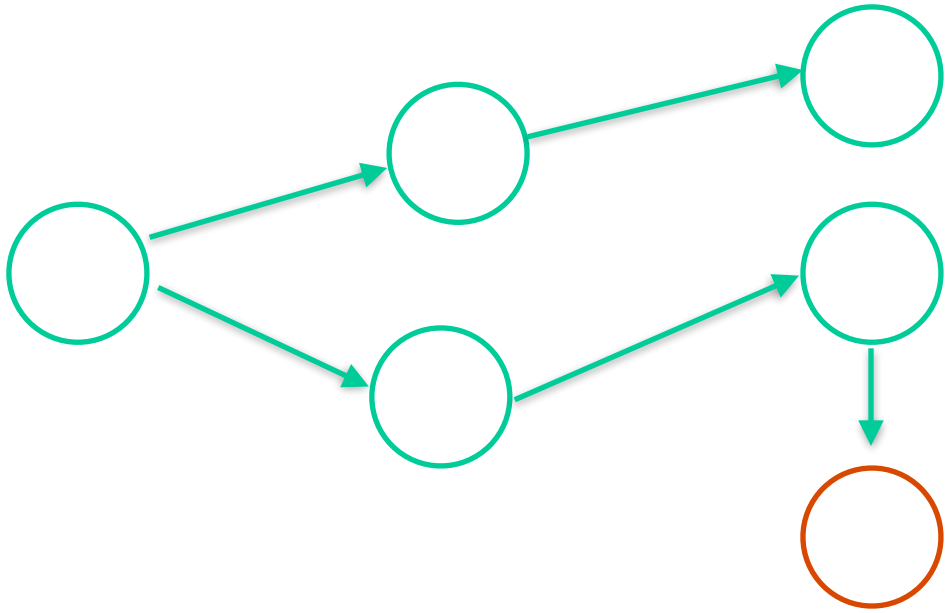
- This approach works for most of the vulnerabilities we have seen today (almost) *independently* from the application being tested.

- What about access control?

Why is automatic testing hard?

- Tools test only for known/common vulnerabilities
- Even if we are only interested in common vulnerabilities
  - What are the interesting states to test?





Is black box testing useful?

- Vendors and security consultants often claim that their product/service is better than it actually is.
- If you have some testing experience you know this is not true…

Recommended read:

Why Johnny can't Pentest: An analysis of Black-box Web Vulnerability Scanners

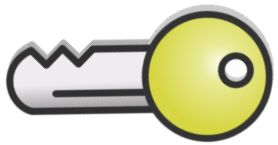
Doupé et al. DIMVA 2010

http://www.cs.bham.ac.uk/~covam/data/papers/dimva10_webscanners.pdf

They analyse popular commercial and free automatic vulnerability scanners for accuracy and detection rate showing their limitations.

- Anybody wants to share experience with vulnerability scanners from the field?

Some lessons



- The client always has full control over what it sends -> Do not trust user input!
- A secure application needs both: secure development and a secure platform it is operated on
- The problems discussed here are real!
- Security is sometimes only poorly addressed during application development and problems are only patched after testing
  - We do not advocate this in this lecture, but we think developers need to know basic testing techniques to improve the quality of their applications!
- Testing helps uncovering vulnerabilities
  - Checks adherence to security good practices
  - But can also make developers aware of typical flaws

What next?

- In the lab we will see tools that help to automatically discovery vulnerabilities in a black-box fashion

- After the presentation of Phase I we will discuss best practices to avoid common web vulnerabilities.