Web Security

Thierry Sans

1991

Sir Tim Berners-Lee

info.cern.ch/hypertext/WWW/TheProject.html

World Wide Web

The WorldWideWeb (W3) is a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an executive summary of the project, Mailing lists , Policy , November's W3 news , Frequently Asked Questions .

What's out there?
Pointers to the world's online information, subjects , W3 servers, etc.
Help
on the browser you are using
Software Products
A list of W3 project components and their current state. (e.g. Line Mode ,X11 Viola , NeXTStep , Servers , Tools , Mail robot , Library)
Technical
Details of protocols, formats, program internals etc
Bibliography
Paper documentation on W3 and references.
People
A list of some people involved in the project.
History
A summary of the history of the project.
How can I help ?
If you would like to support the web..
Getting code
Getting the code by anonymous FTP , etc.

Web Portals

tumblr.

SAP

2014

salesforce

Customer Resources Management

Accounting and Billing

Insta

E-Health

coursera

Dropbox

talk

Google Apps

E-Learning

Collaboration

YouTube

LinkedIn

Content Management

Social Networks

Publishing

amazon web services™

Web security is a major concern

Facebook closes hole that let spammers auto-post to walls, friends

Social-networking site plugs a second hole that allowed spammers to automatically post to people's pages.

by **Elinor Mills** @elinormills / September 7, 2010 12:37 PM PDT / Updated: September 7, 2010 4:00 PM PDT

GhostShell claims breach of 1.6M accounts at FBI, NASA, and more

The hacktivist group says it obtained the records via SQL injection at government sites.

by **Casey Newton** @CaseyNewton / December 10, 2012 3:13 PM PST / Updated: December 10, 2012 3:19 PM PST

CNET › Security › Yahoo Mail hijacking exploit selling for $700

Yahoo Mail hijacking exploit selling for $700

XSS vulnerability allows attacks to steal and replace tracking cookies, as well as read and send e-mail from a victim's account.

by **Steven Musil** @stevenmusil / November 26, 2012 6:02 PM PST / Updated: November 27, 2012 3:32 PM PST

Researchers point out holes in McAfee's Web site

McAfee says it is working to fix three holes researchers found in its Web site.

by **Elinor Mills** @elinormills / March 28, 2011 7:28 PM PDT

Cross Site Scripting in download.mcafee.com. "In a worst case scenario this vulnerability could allow attacks that spoof the McAfee brand by presenting a URL that looks like it directs to a McAfee Web site but in fact directs elsewhere."

Researchers find security holes in NYT, YouTube, ING, MetaFilter sites

Attackers could have used vulnerabilities on several Web sites to compromise people's accounts, allowing them to steal money, harvest e-mail addresses, or pose as others online.

by **Elinor Mills** @elinormills / October 2, 2008 1:02 PM PDT / Updated: October 2, 2008 2:31 PM PDT

The vulnerability arises from a coding flaw that could allow someone to do a cross-site request forgery (CSRF) attack in which a "malicious Web site causes a user's Web browser to perform an unwanted action on a trusted site," according to the report.

Researcher finds serious Android Market bug

Google applies technical fix to bug, but Jon Oberheide says Android Market should be alerting phone owners when an app is being remotely downloaded via the Web site.

Oberheide described the XSS vulnerability as "low-hanging fruit" and said he was surprised no one had discovered it before. Such bugs are very common in Web sites.

Twitter hit by multiple variants of XSS worm

Summary: *During the weekend and early Monday, at least four separate variants of the original StalkDaily.com XSS worm hit the popular micro-blogging site Twitter, automatically hijacking accounts and advertising the author's web site by posting tweets on behalf of the account holders, by exploiting cross site scripting flaws at the site.*

By Dancho Danchev for Zero Day | April 14, 2009 -- 02:19 GMT (03:19 BST)

Follow @danchodanchev

New security holes found in D-Link router

Security researcher reveals multiple Web-based security vulnerabilities in the D-Link 2760N.

by Seth Rosenblatt @sethr / November 11, 2013 12:54 PM PST / Updated: November 12, 2013 4:54 PM PST

A new spate of vulnerabilities have been found in a D-Link router, a security researcher said Monday.

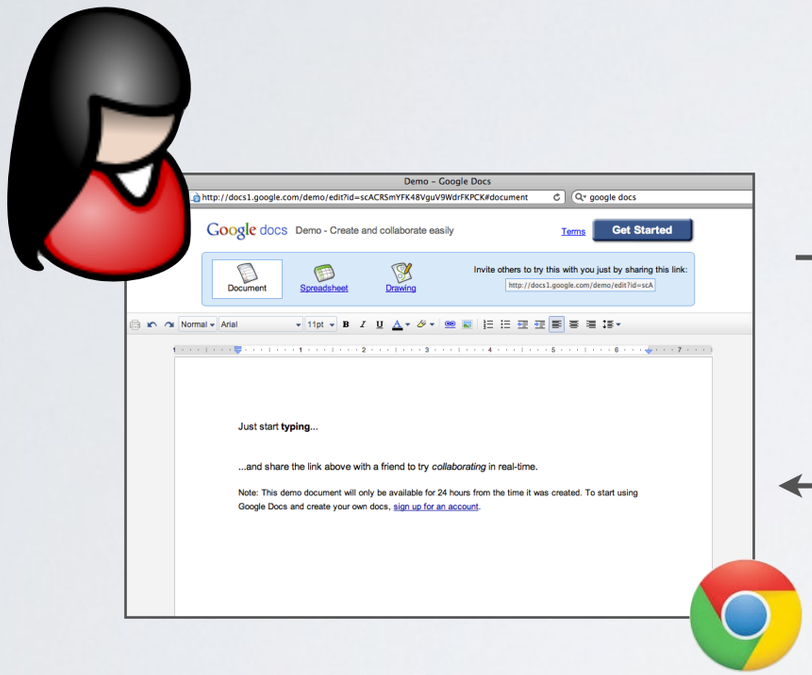
The D-Link 2760N, also known as the D-Link DSL-2760U-BN, is susceptible to **several cross-site scripting (XSS) bugs** through its Web interface, **reported ThreatPost**.

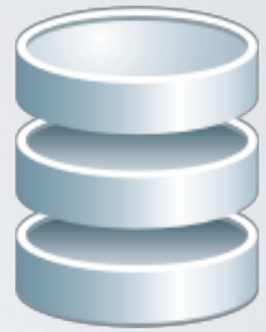
The Big Picture

The web architecture

Client Side

Server Side





Web Browser

Web Server

Database

Securing the web architecture means securing ...

- The network
- The DNS (Domain Name System)
- The web server operating system
- The web server application (*Apache* for instance)
- The database application (*Oracle* for instance)
- The web application

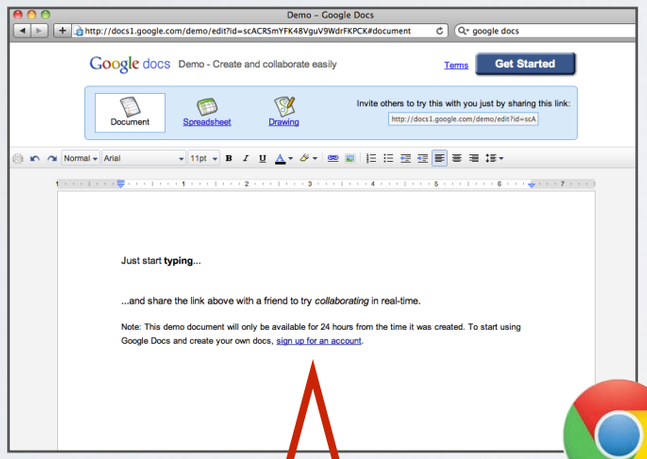
Our focus here!

What is a web application?

program running on the browser

+

program running on the server



Anatomy of a web application

The HTTP protocol

Network protocol for requesting/receiving data on the Web

- Standard TCP protocol on **port 80** (by default)
- **URI/URL** specifies what resource is being accessed
- Different **request methods**

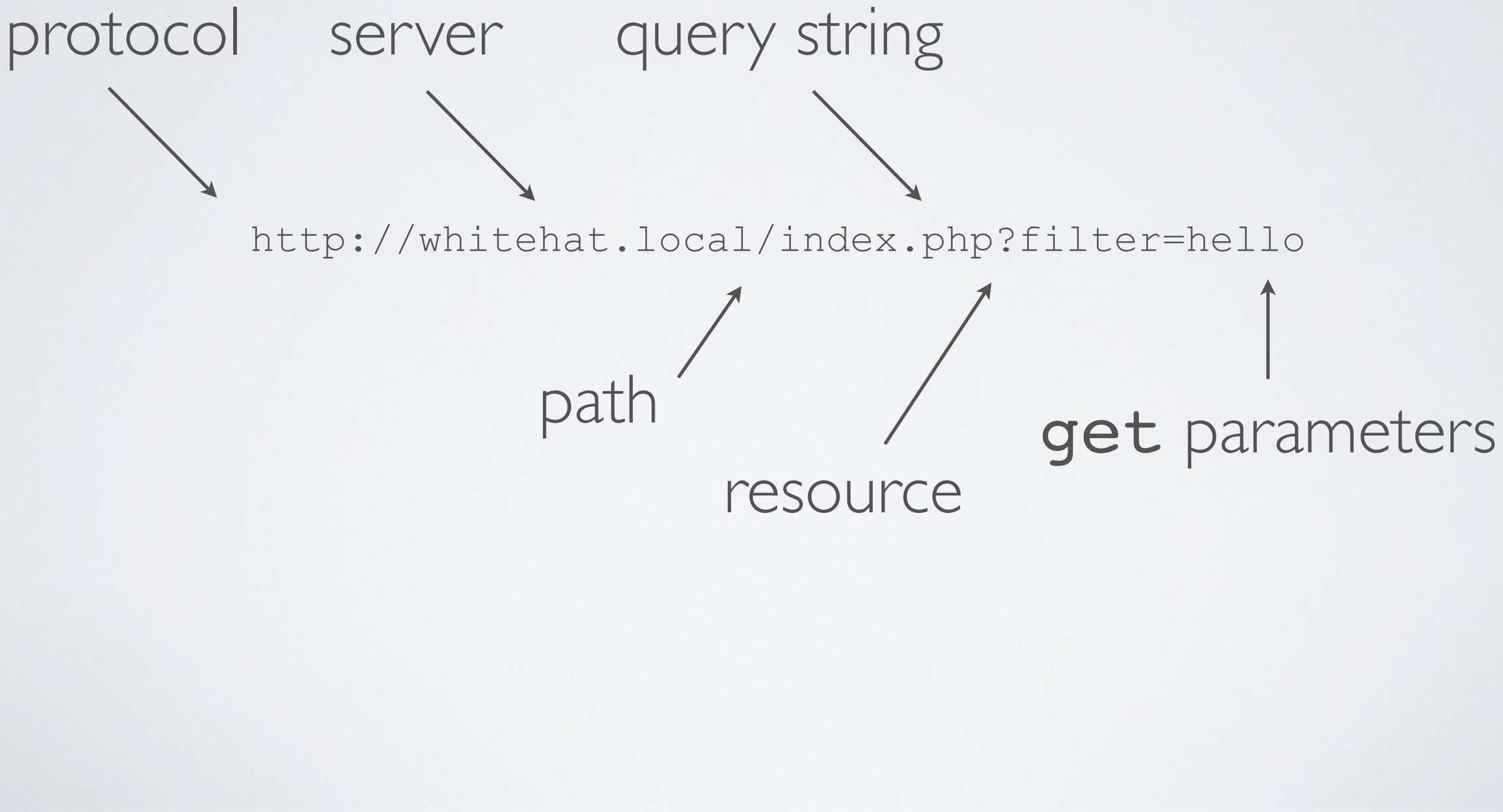
Let's look at what a web server does

telnet to a web server

```
> telnet whitehat.local 80
GET /
```

enter HTTP requests

Anatomy of a URL



Authentication and Authorization

✓ Authentication

- Who are the authorized users?

✓ Authorization

- Who can access what and how?

The simple recipe for user authentication

1. **Ask the user for a login and password** and send it to the server (HTTP/POST request)
2. **Verify the login/password** based on information stored on the server (usually in the database)
3. **Start a session** once the user has been authenticated
4. **Grant access to resources** according to the session

The concept of session

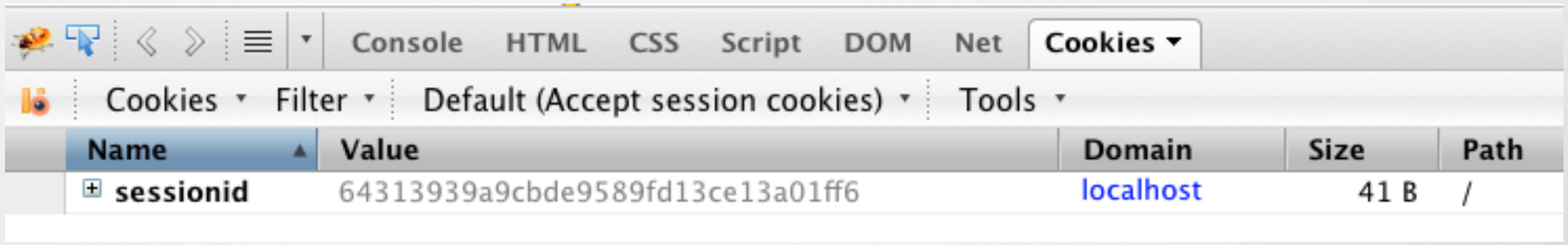
There is a **session id** (aka token)
between the browser and the web application

This session id should be **unique** and **unforgeable**
(usually a long random number or a hash)

➡ Stored in the cookie

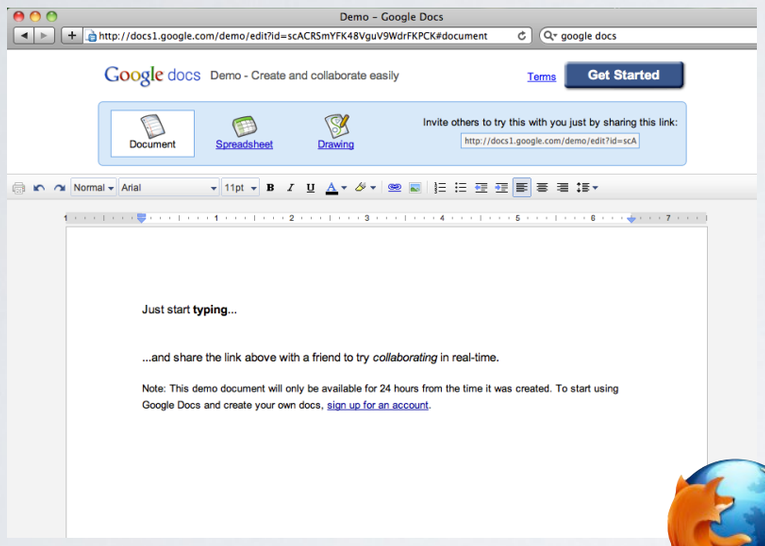
The session id is bind to **key/value pairs data**

➡ Stored on the server

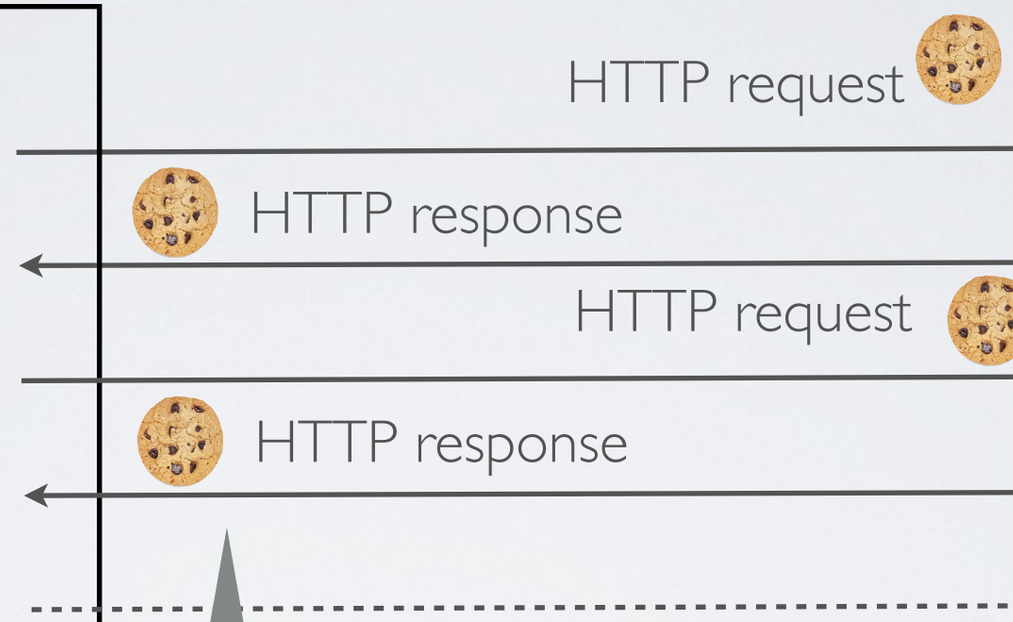


The big picture

Session : key/value pairs stored on the server



Web Browser



Web Server

Cookie : key/value pairs stored in the requests

The user can **create, modify, delete** the session ID in the cookie

But **cannot access** the key/value pairs stored on the server

Insufficient Transport Layer Protection

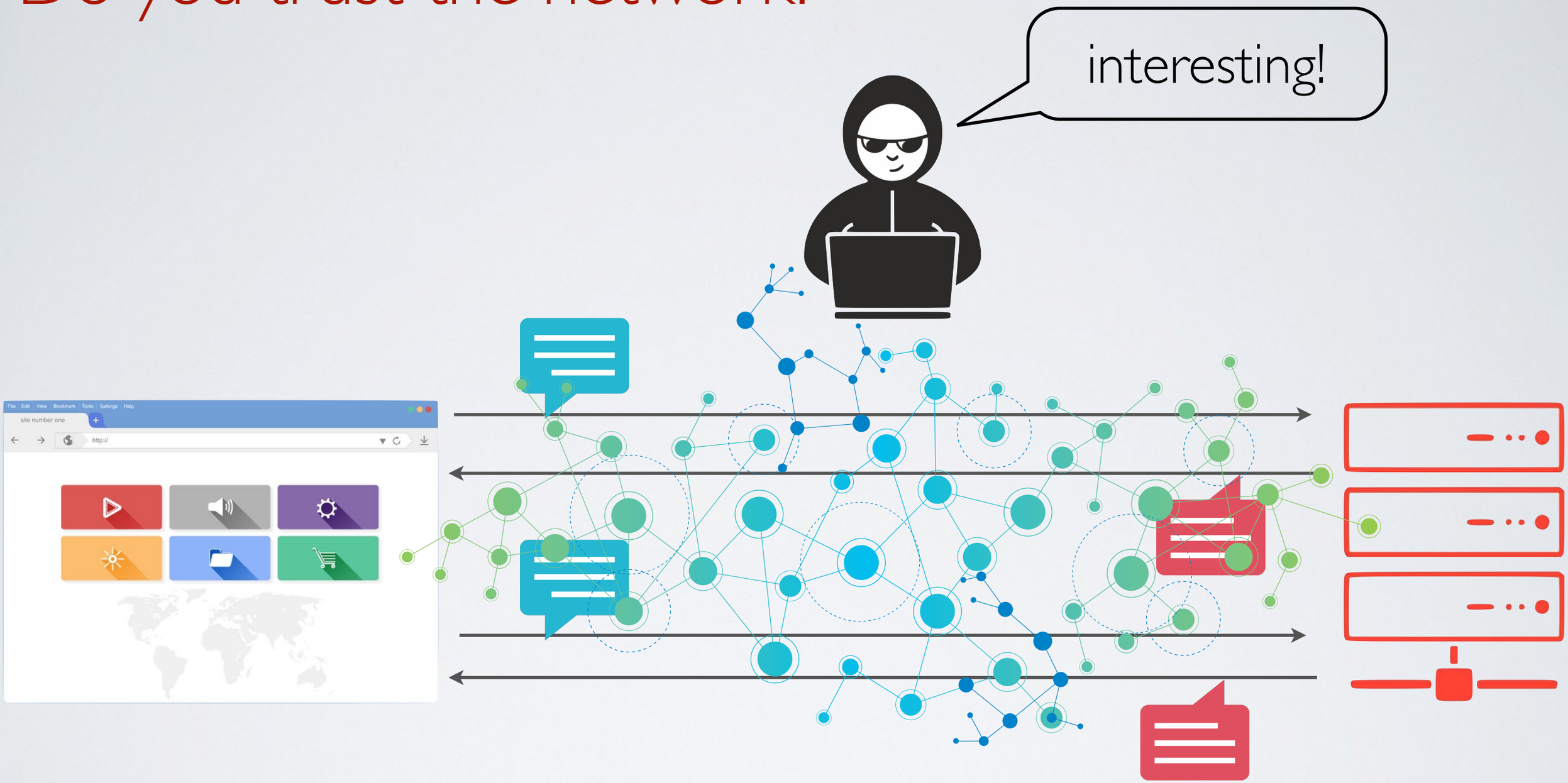
a.k.a the need for HTTPs

How to steal user's credentials

- Brute force the user's password or session ID

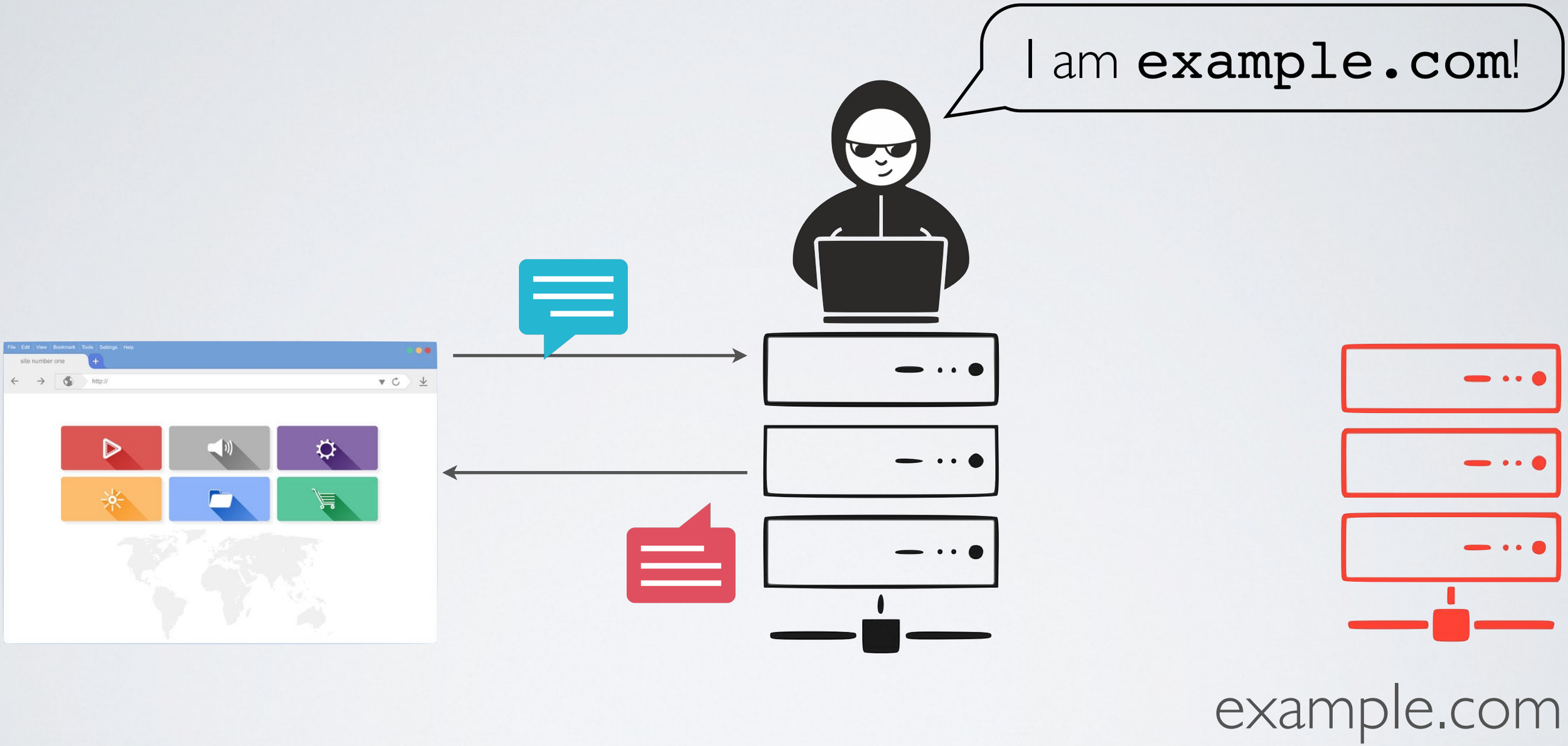
- Steal the user's password or session ID

Do you trust the network?



- Threat 1 : an attacker **can eavesdrop** messages sent back and forth

Do you *really* trust the network?



- Threat 2 : an attacker **can tamper with** messages sent back and forth

Confidentiality and Integrity

- Threat 1 : an attacker **can eavesdrop** messages sent back and forth

  Confidentiality: how do exchange information secretly?

- Threat 2 : an attacker **can tamper** messages sent back and forth

  Integrity: How do we exchange information reliably?

Why and when using HTTPS?

HTTPS = HTTP + TLS

➡ TLS provides

- confidentiality: end-to-end secure channel
- integrity: authentication handshake

➡ HTTPS protects any data send back and forth including:

- login and password
- session ID

✓ **HTTPS everywhere**
HTTPS must be used during the entire session

Be careful of mixed content

Mixed-content happens when:

1. an HTTPS page contains elements (ajax, js, image, video, css ...) served with HTTP

2. an HTTPS page transfers control to another HTTP page within the same domain

- authentication cookie will be sent over HTTP

`Secure` cookie flag

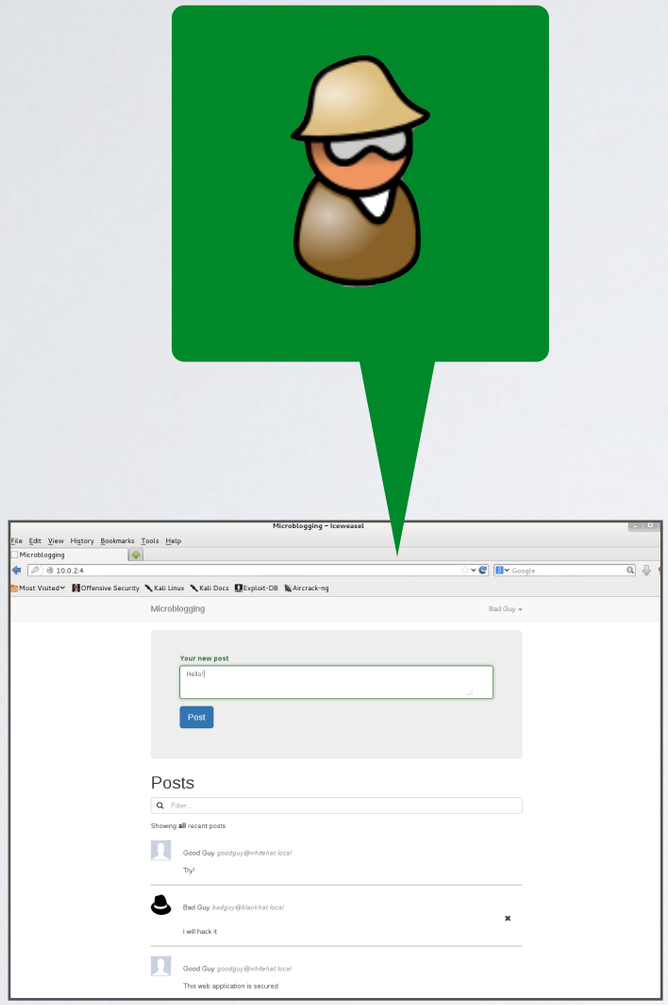
✓ The cookie will be sent over HTTPS exclusively

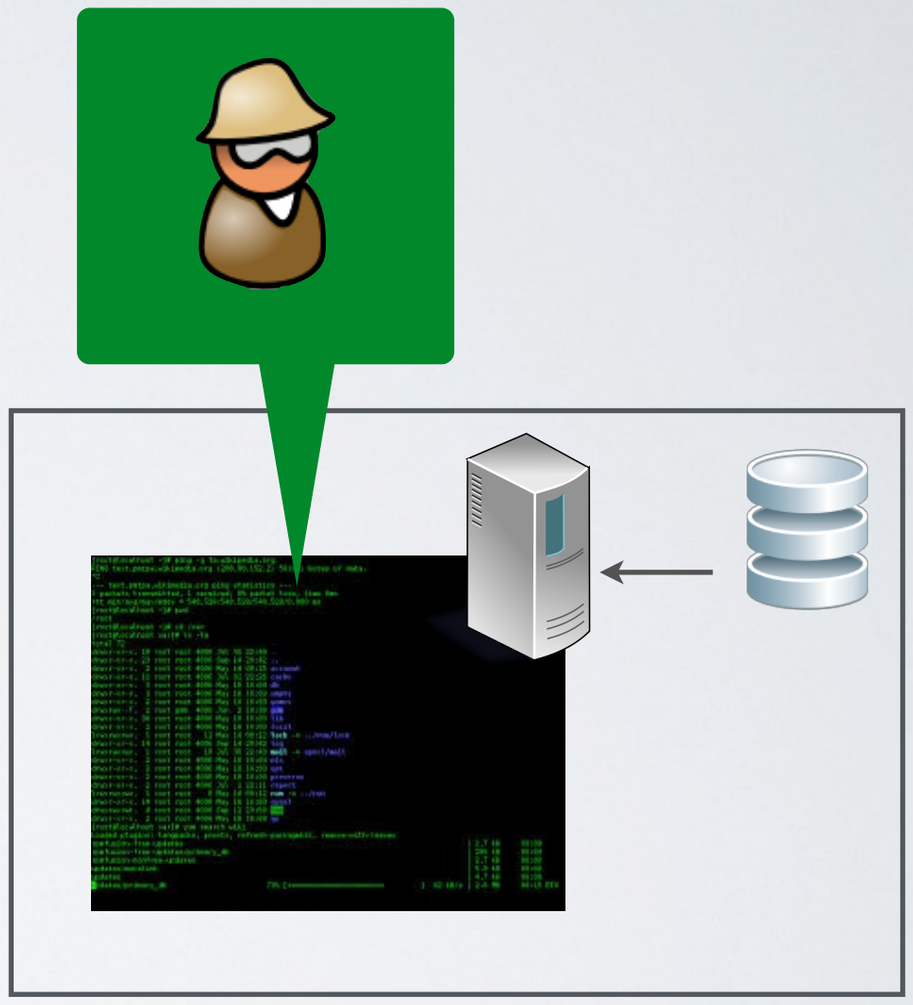
➡ Prevents authentication cookie from leaking in case of mixed-content

Do/Don't with HTTPS

- Always use HTTPS exclusively (in production)
- Always have a valid and signed certificate (no self-signed cert)
- Always avoid using absolute URL (mixed-content)
- Always use `secure` cookie flag with authentication cookie

Limitation of HTTPS



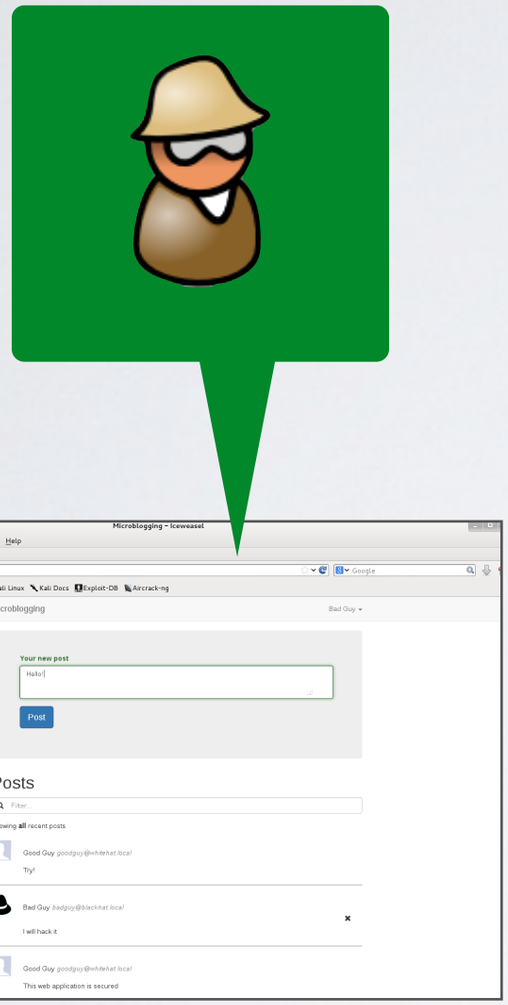


password = 123456

E#%FY7*5EZ$#G

password = 123456

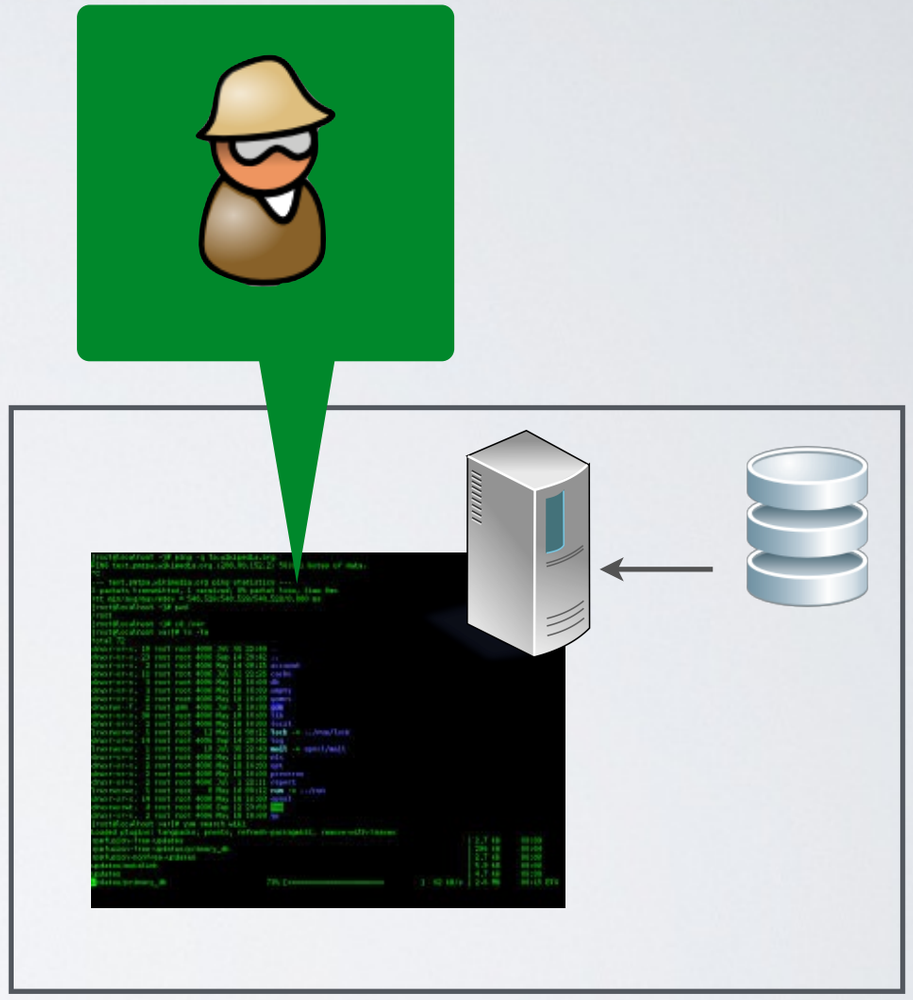
Stealing passwords from the client

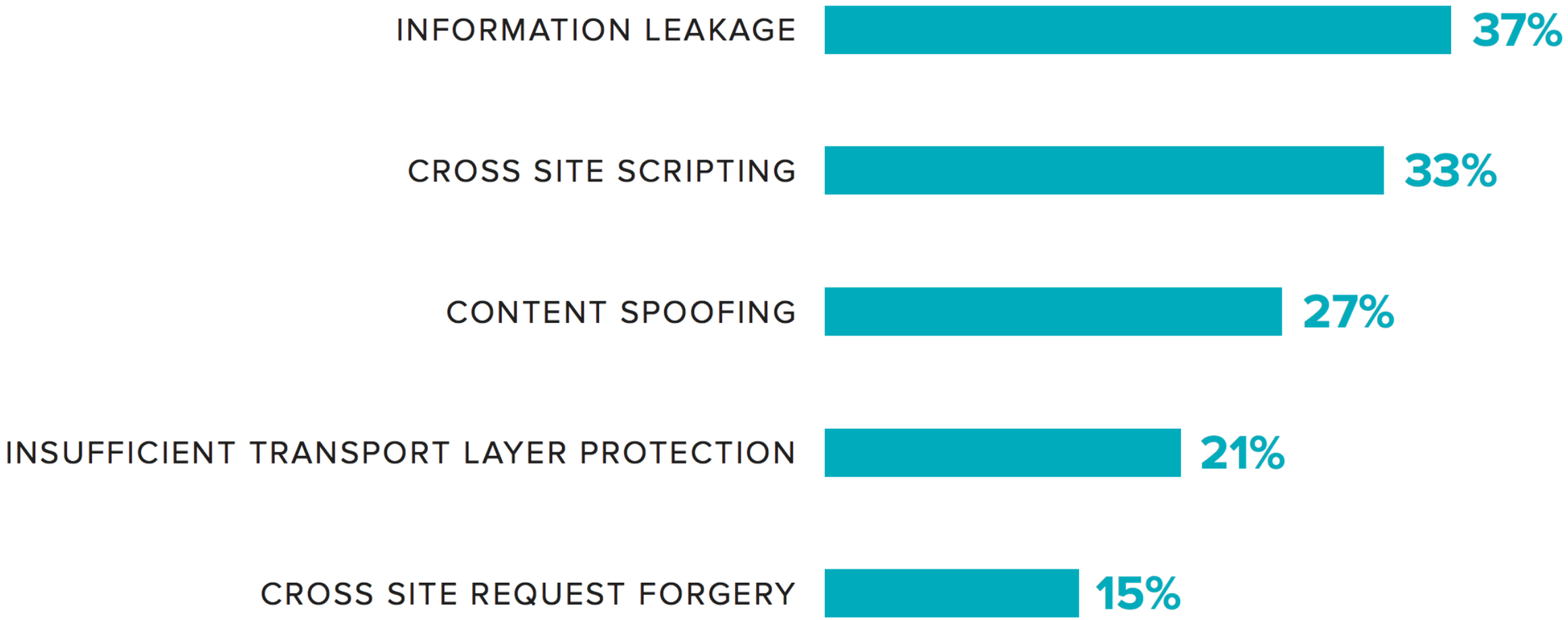


- Social engineering - Phishing
- Keyloggers (keystroke logging)
- Data mining (emails, logs)
- Hack the client's code

Stealing passwords from the server

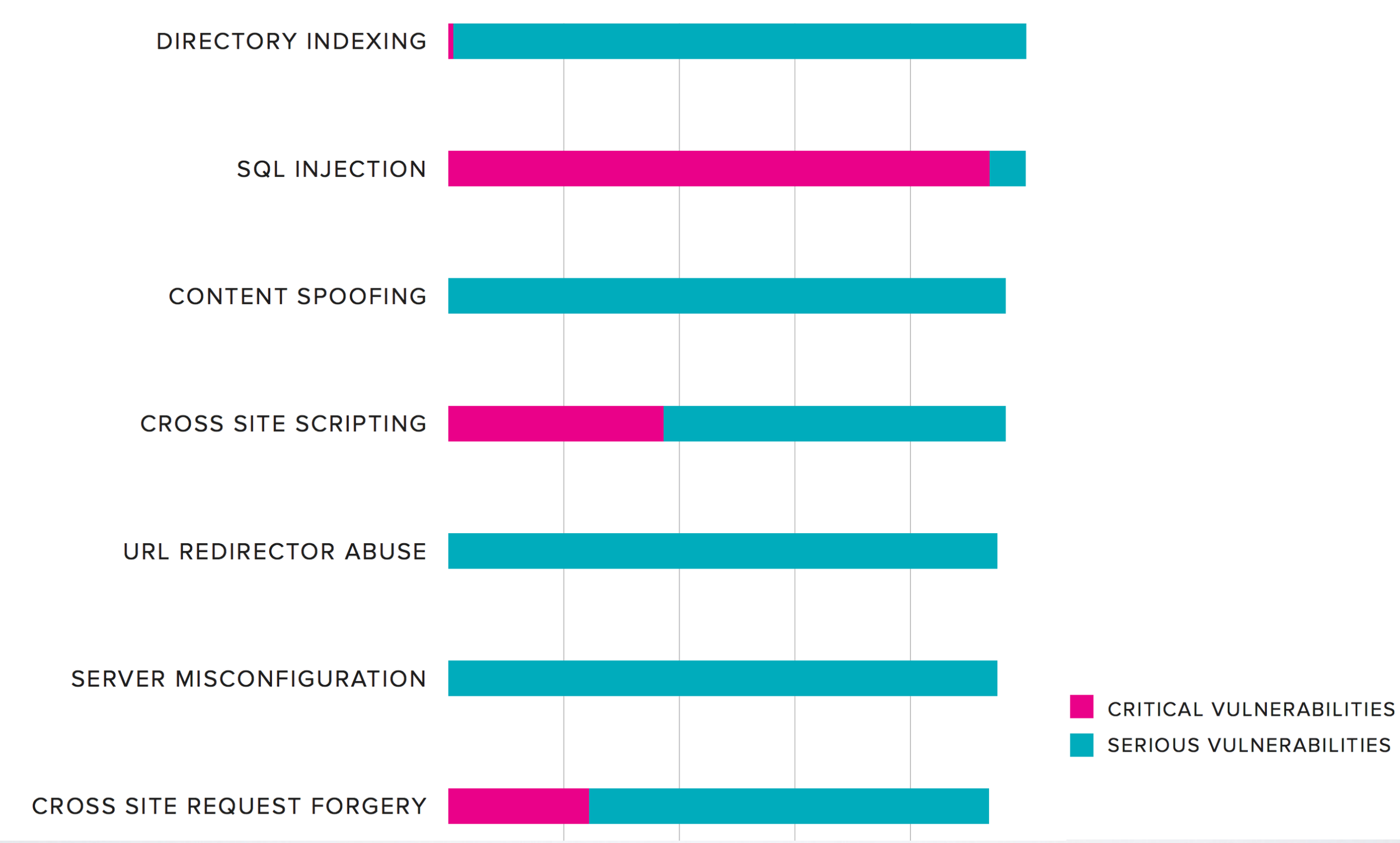
- Hack the server
- Hack the server's side code





Vulnerability Likelihood

source *"WhiteHat Website Security Statistics report 2017"*
from *WhiteHat Security*



Most serious vulnerabilities by class

source *"WhiteHat Website Security Statistics report 2017"*
from *WhiteHat Security*

Beyond HTTPS - attacking the web application

Frontend Vulnerabilities

- Content Spoofing
- Cross-Site Scripting
- Cross-site Request forgery

Backend Vulnerabilities

- Incomplete mediation
- Information leakage
- SQL injection

Backend Vulnerability

Information Leakage

Information Leakage

"AT&T Inc. apologized to Apple Inc. iPad 3G tablet computer users whose **e-mail addresses were exposed during a security breach** disclosed last week."

source *Business Week - June 14 2010*

"There's no hack, no infiltration, and no breach, **just a really poorly designed web application** that returns e-mail address when ICCID is passed to it."

source *Praetorian Prefect - June 9 2010*

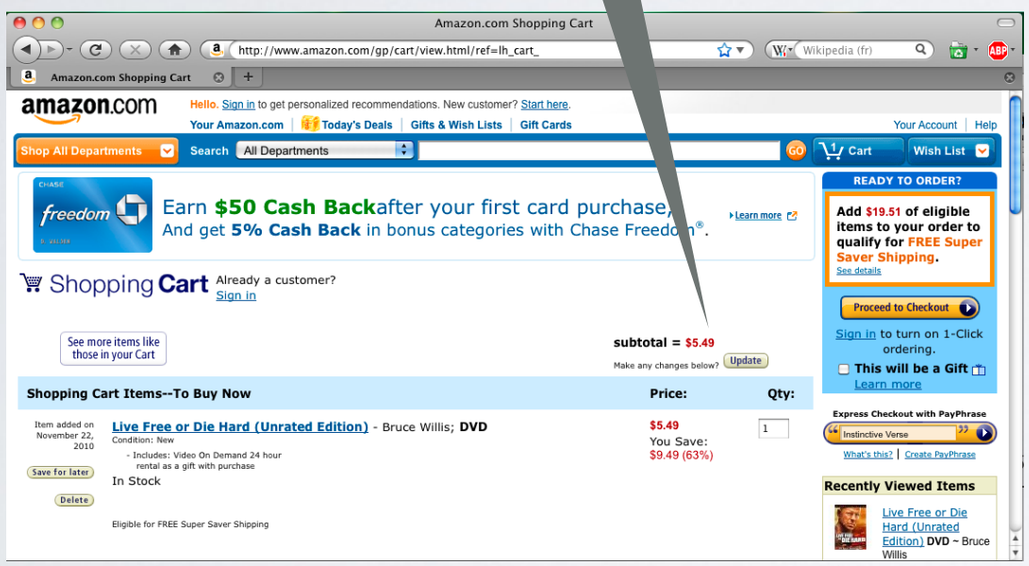
Backend Vulnerability

Incomplete Mediation

Incomplete Mediation - The Shopping Cart Attack

The total is calculated by a script on the client

The order is generated based on the request



order=(#2956,10,[redacted],10)

Thank you for your order!

Client Trusted Domain

Server Trusted Domain

* *Notice that Amazon is **not** vulnerable to this attack*

The backend is the **only trusted domain**

- ◉ Data coming from the frontend cannot be trusted

- ✓ Sensitive operations must be done on the backend

Backend Vulnerability

SQL Injection

Problem

- ➡ An attacker can inject SQL/NoSQL code
  - Retrieve, add, modify, delete information
  - Bypass authentication

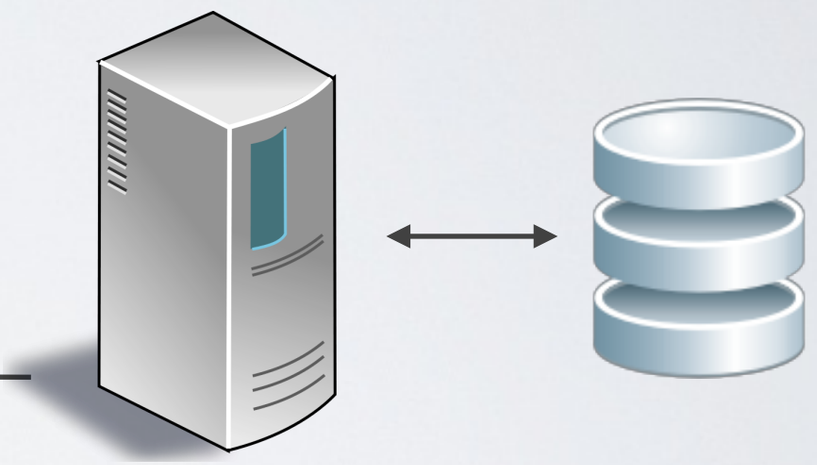
Checking password

`signin.html`

`/signin/`

name=Alice&pwd=pass4alice

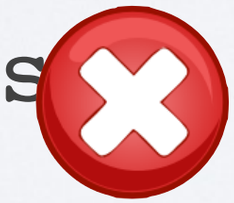
Access Granted!



Bypassing password check

```
db.run("SELECT * FROM users
WHERE USERNAME = '" + username + "'
  AND PASSWORD = '" + password + "'"
```

```
username: alice
password: pas lice
```

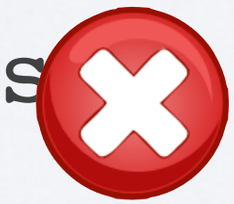


```
blah' OR '1'='1
```

NoSQL Injection

```
db.find({   username: username,
            password: password    });
```

```
username: alice
password: pas lice
```

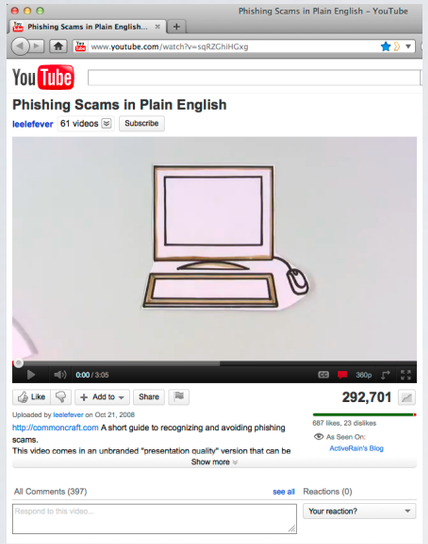


```
{gt: ""}
```

Frontend Vulnerability

Content Spoofing

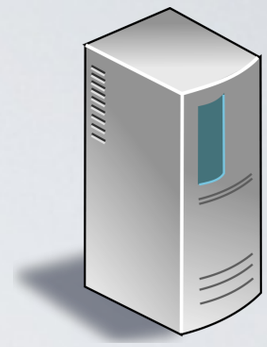
Content Spoofing

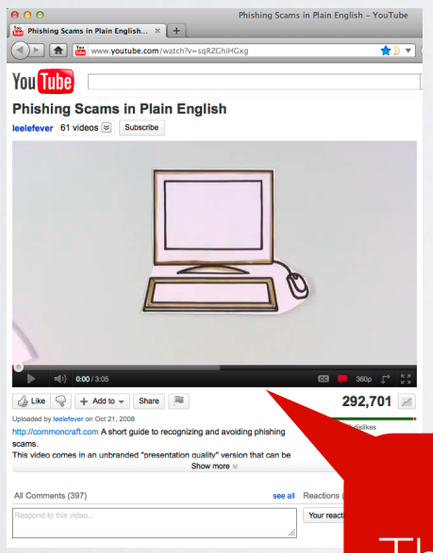


GET /?videoid=527

<html ...

comment = "<a href="myad.com">Fun stuff ...





GET /?videoid=527

<html ...

The page contains the attacker's ad.

** Notice that Youtube is **not** vulnerable to this attack*

Problem

- ➡ An attacker can inject HTML tags in the page
  - Add illegitimate content to the webpage (ads most of the time)

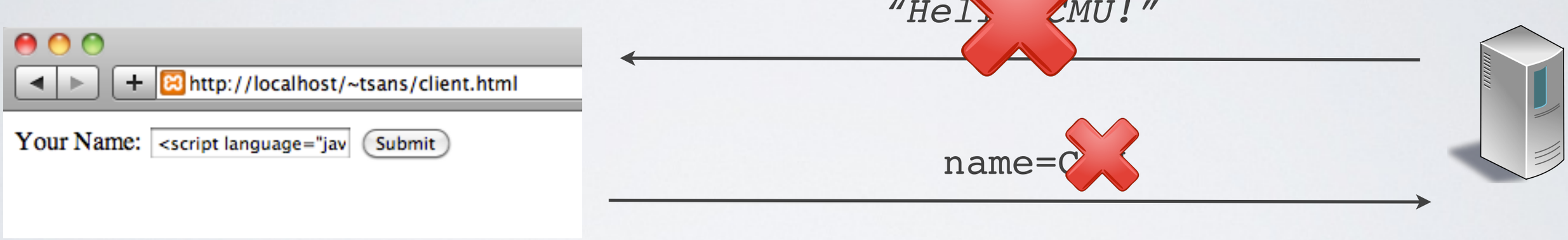
Generic Solution

- ✓ Data inserted in the DOM must be validated

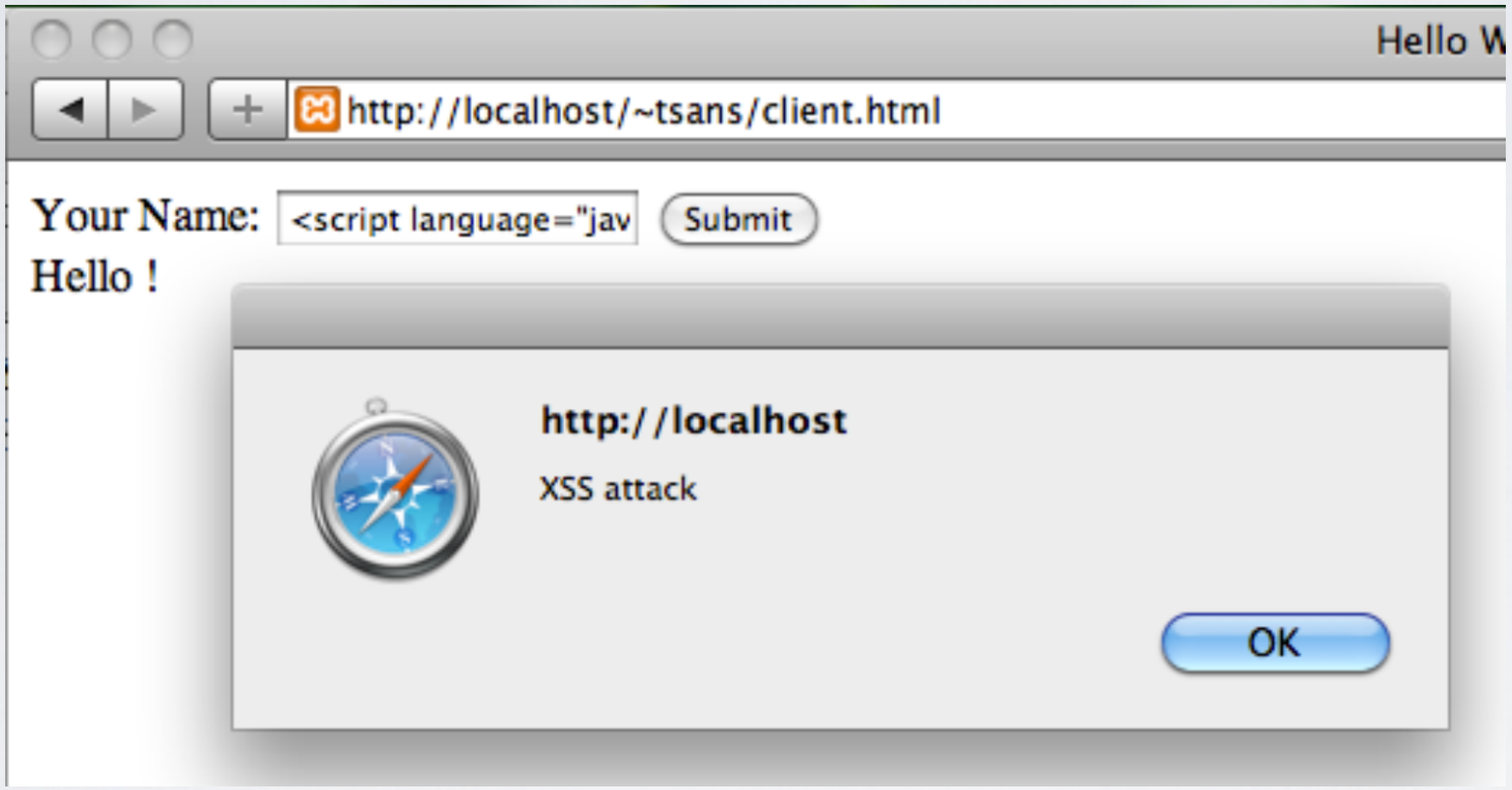
Frontend Vulnerability

Cross-Site Scripting (XSS)

Cross-Site Scripting Attack (XSS attack)



XSS Attack = Javascript Code Injection

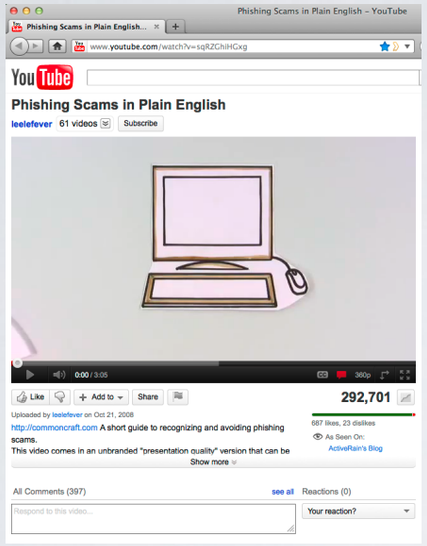


Problem

➡ An attacker can inject **arbitrary javascript code** in the page that will be executed by the browser

- **Inject illegitimate content** in the page (same as content spoofing)
- **Perform illegitimate HTTP requests** through Ajax (same as a CSRF attack)
- **Steal Session ID** from the cookie
- **Steal user's login/password** by modifying the page to forge a perfect scam

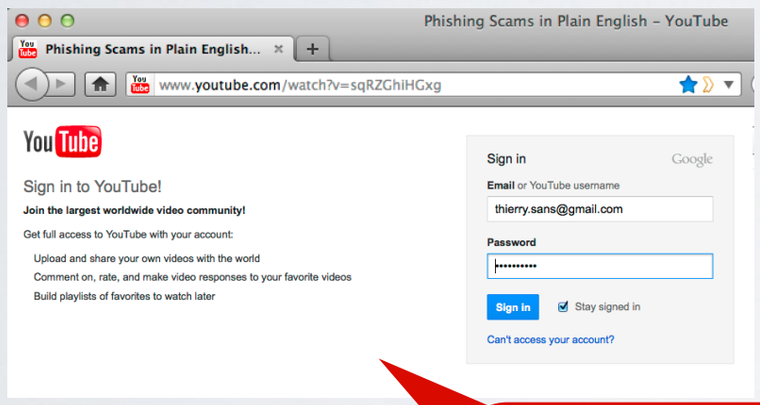
Forging a perfect scam



GET /?videoid=527

<html ...

comment = “<script> ...



GET /?videoid=527

<html ...

login=Alice&password=123456

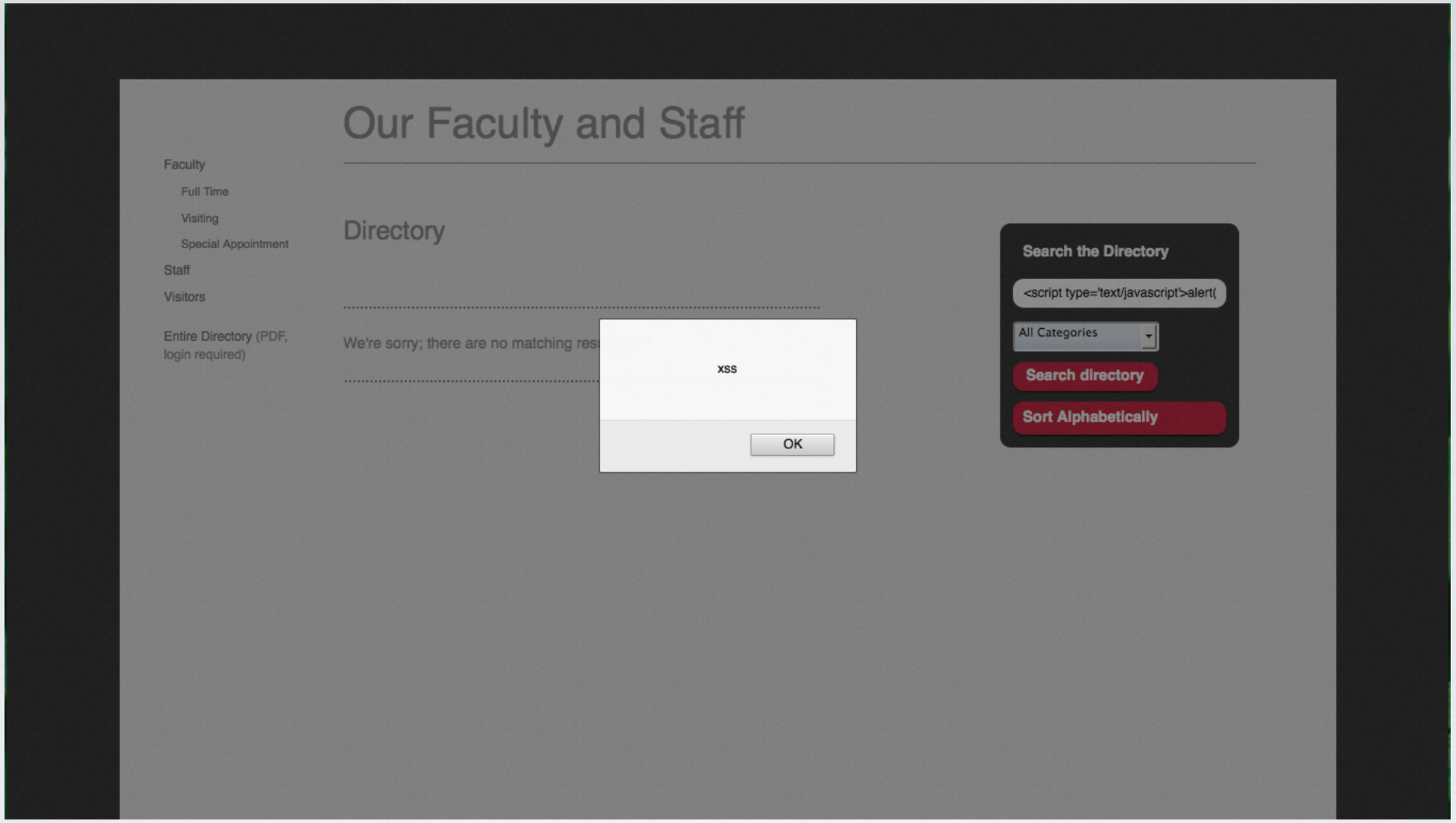
The script contained in the comments modifies the page to look like the login page!

** Notice that Youtube is **not** vulnerable to this attack*

It gets worst - XSS Worms

Spread on social networks

- Samy targeting MySpace (2005)
- JTV.worm targeting Justin.tv (2008)
- Twitter worm targeting Twitter (2010)



XSS attacks are widespread

Variations on XSS attacks

- **Reflected XSS**
  Malicious data sent to the backend are immediately sent back to the frontend to be inserted into the DOM

- **Stored XSS**
  Malicious data sent to the backend are store in the database and later-on sent back to the frontend to be inserted into the DOM

- **DOM-based attack**
  Malicious data are manipulated in the frontend (javascript) and inserted into the DOM

Solution

- ✓ Data inserted in the DOM must be validated

`HttpOnly` cookie flag

✓ The cookie is not readable/writable from the frontend

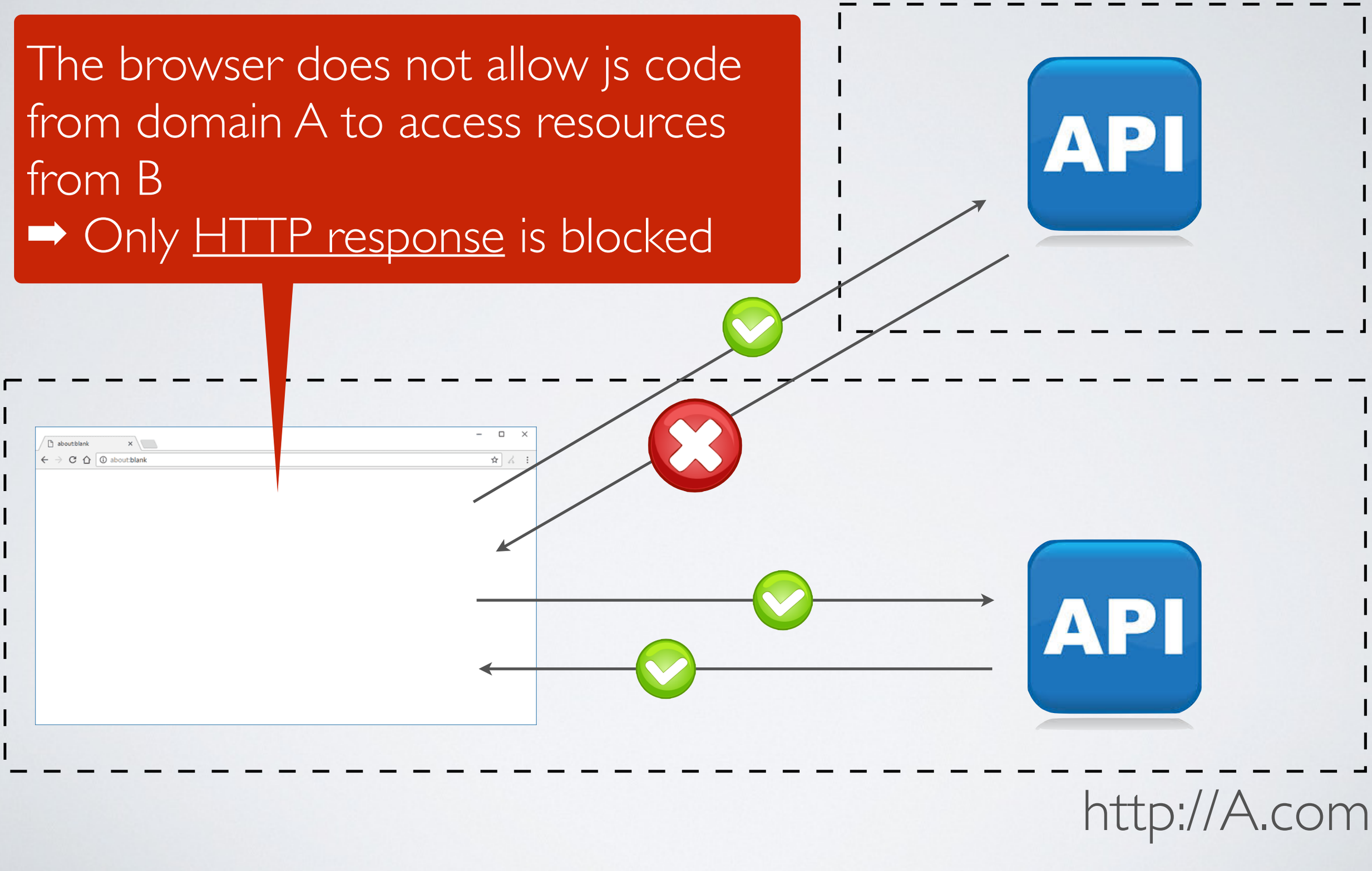
➡ Prevents the authentication cookie from being leaked when an XSS attack (cross-site scripting) occurs

Cross-Site Request Forgery

Ajax requests across domains

The browser does not allow js code from domain A to access resources from B

➡ Only HTTP response is blocked



Same origin policy

➡ **Resources must come from the same domain (protocol, host, port)**

Elements under control of the same-origin policy

- Ajax requests
- Form actions

Elements **not** under control of the same-origin policy

- Javascript scripts
- CSS
- Images, video, sound
- Plugins

Examples

	client	server
same protocol, port and host	`http://example.com`	`http://example.com`
	`http://user:pass@example.com`	`http://example.com`
top-level domain	`http://example.com`	`http://example.org`
host	`http://example.com`	`http://other.com`
sub-host	`http://www.example.com`	`http://example.com`
sub-host	`http://example.com`	`http://www.example.com`
port	`http://example.com:3000`	`http://example.com`
protocol	`http://example.com`	`https://example.com`

[digression] relaxing the same-origin policy

- Switch to the superdomain with javascript
  `www.example.com` can be relaxed to **`example.com`**
- iframe
- JSONP
- Cross-Origin Resource Sharing (CORS)

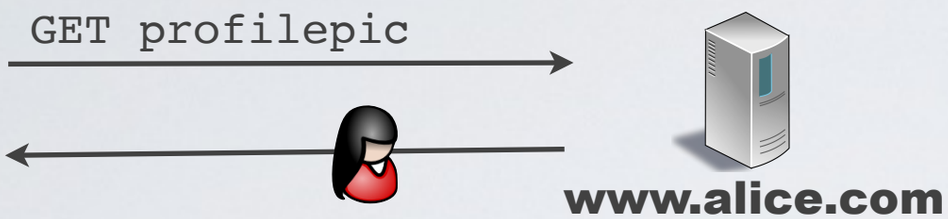
Problem

➡ An attacker can executes unwanted but yet authenticated actions on a web application by either

- setting up a malicious website with cross-origin requests
- or by injecting malicious urls into the page

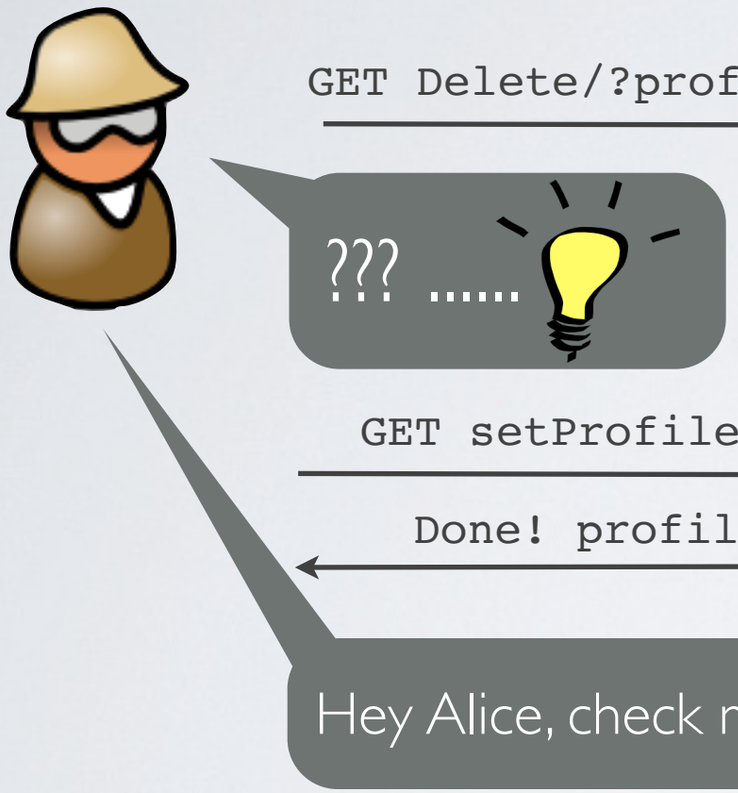
GET View/?profileid=53

GET profilepic

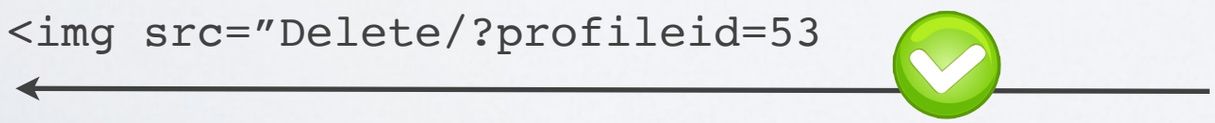


www.alice.com

GET Delete/?profileid=53



GET View/?profileid=86



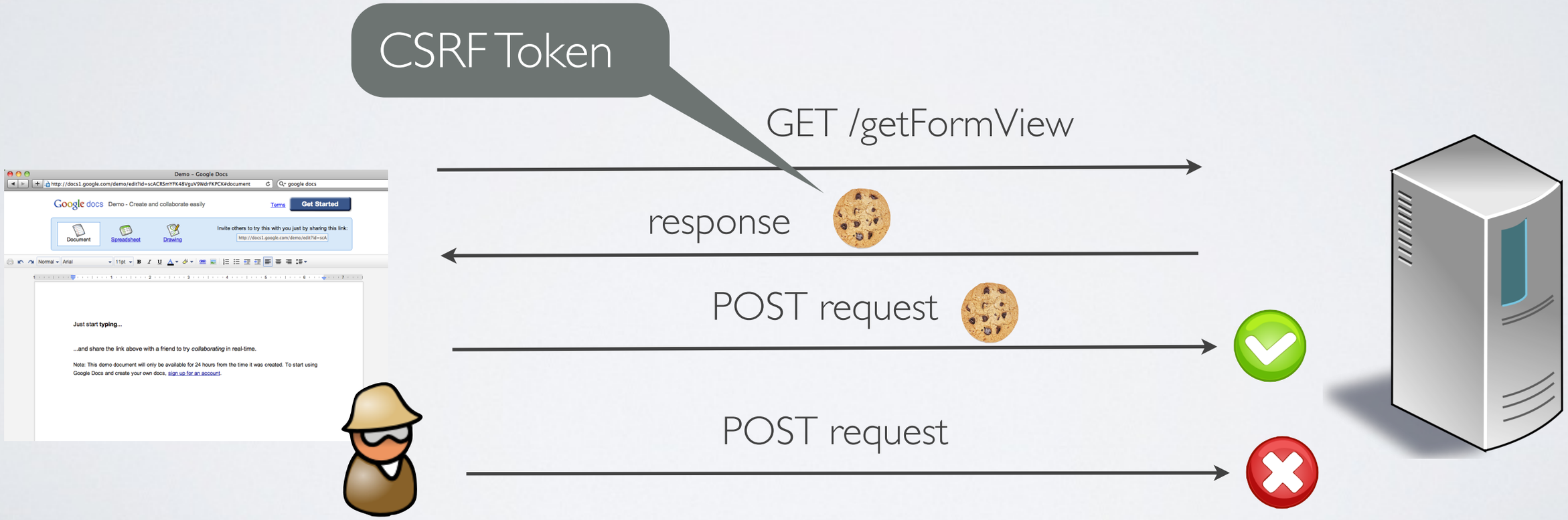
GET Delete/?profileid=53

www.badwebsite.com

id	url	name
53	www.alice.com/ profilepic	Alice
86	www.badwebsite.com/ Delete/?imageid=53	Charlie

Generic solution - CSRF tokens

✓ Protect legitimate requests with a CSRF token



`SameSite` cookie flag

✓ The cookie will be not be sent over cross-site requests

➡ Prevents forwarding the authentication cookie over cross-origin requests (cross-site request forgery)

Web Penetration Testing

Web application security tools

- Proxy mapper
- Vulnerability scanner
- Replay HTTP requests
- (Exploit tool)

Commercial

AppScan

Acunetix

Burp Suite

Nikto

W3af

Vega

Open Source

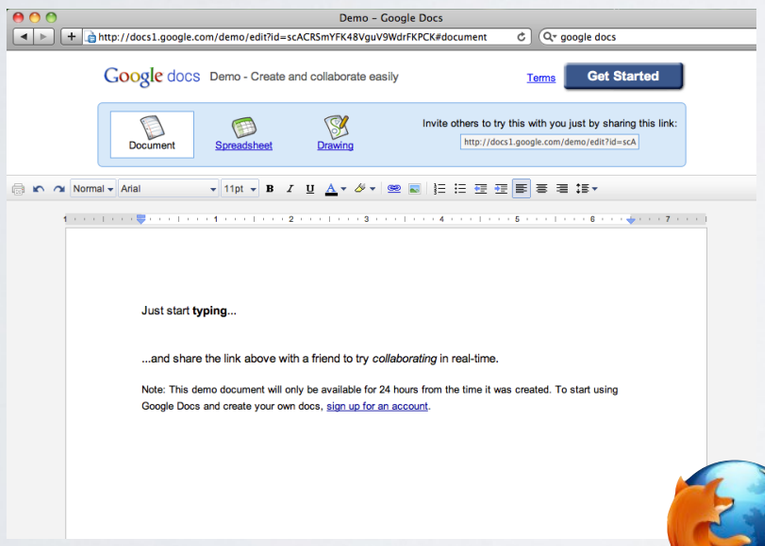
… among others

Conclusion

You have **absolutely no control** on the client

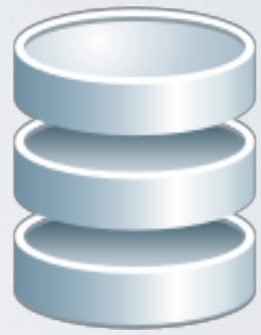
Client Side

Server Side



Web Browser

Web Server



Database

References

- Mozilla Secure Coding Guideline
  https://wiki.mozilla.org/WebAppSec/Secure_Coding_Guidelines

- Ruby on Rails Security Page
  http://guides.rubyonrails.org/security.html

- Django Security Page
  https://docs.djangoproject.com/en/dev/topics/security/

- PHP Security Pages
  http://php.net/manual/en/security.php
  http://phpsec.org/projects/guide/