Web Security

Thierry Sans

Securing the web architecture means securing ...

- The network
- The operating system
- The web server (*Apache* for instance)
- The administration server (*SSH* for instance)
- The database (*Oracle* for instance)
- The web application

Our focus here!

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

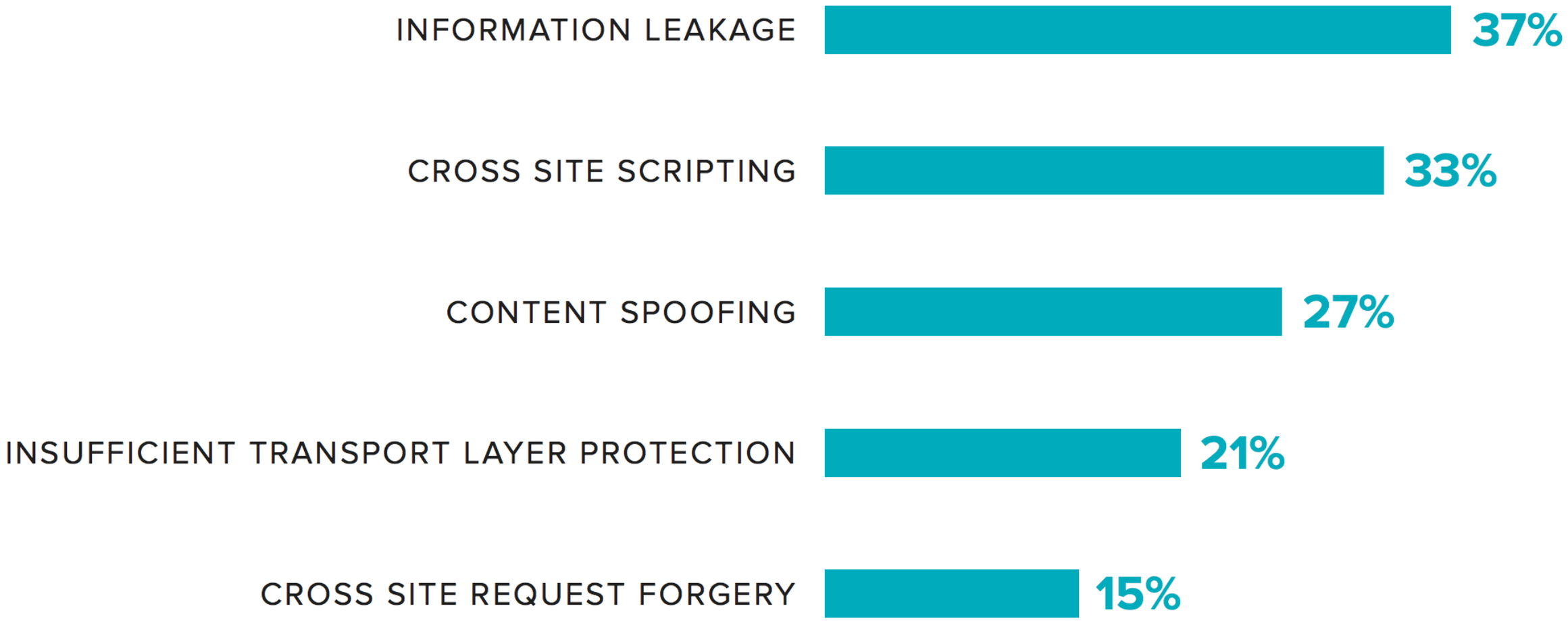
New security holes found in D-Link router

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

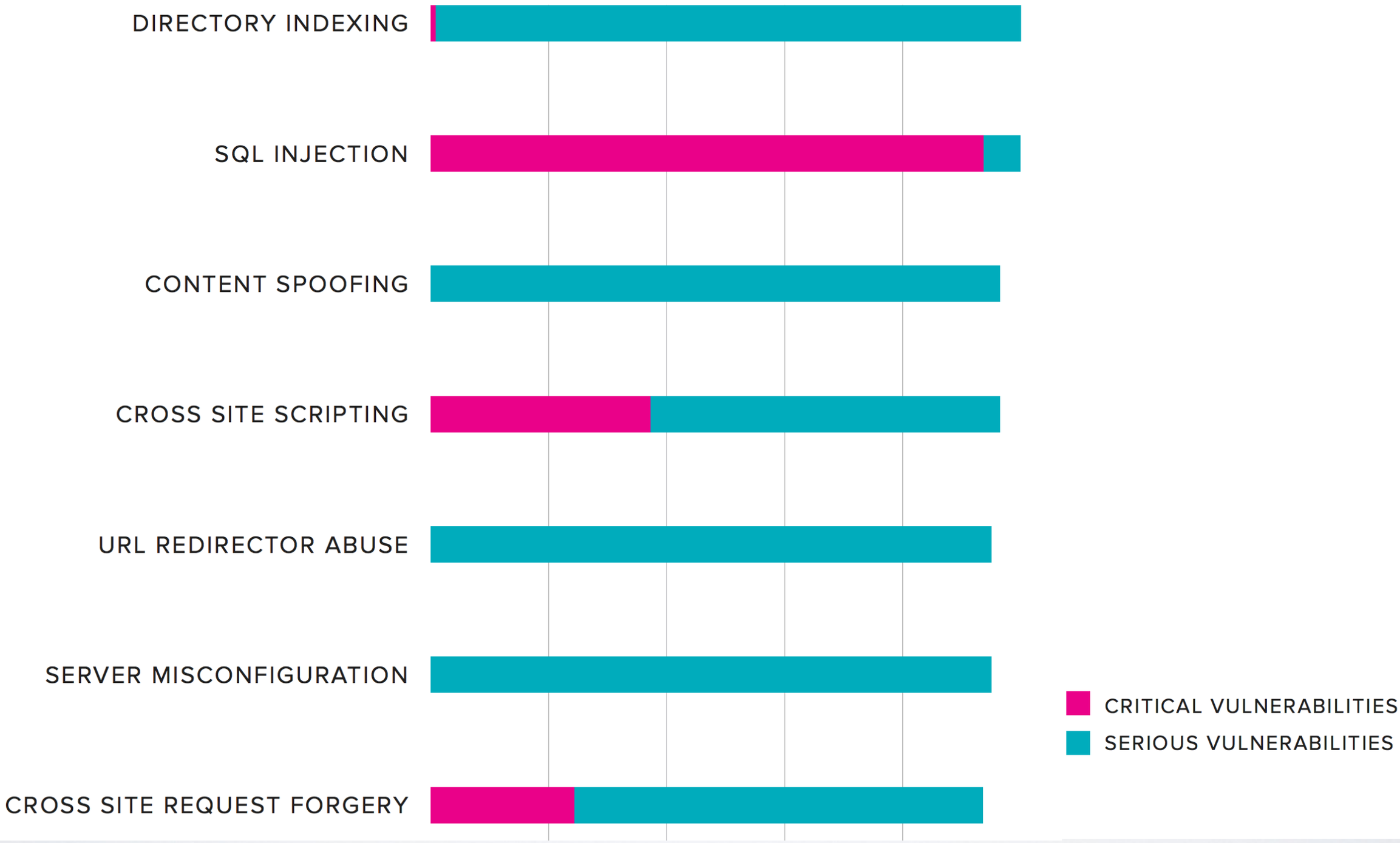
A new spate of vulnerabilities have been found in a D-Link router, a security researcher said Monday.

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**.



Vulnerability Likelihood

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



Most serious vulnerabilities by class

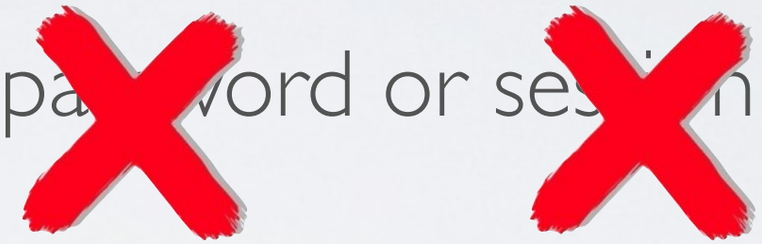
source *"WhiteHat Website Security Statistics report 2017"*
from *WhiteHat Security*

Insufficient Transport Layer Protection

a.k.a the need for HTTPs

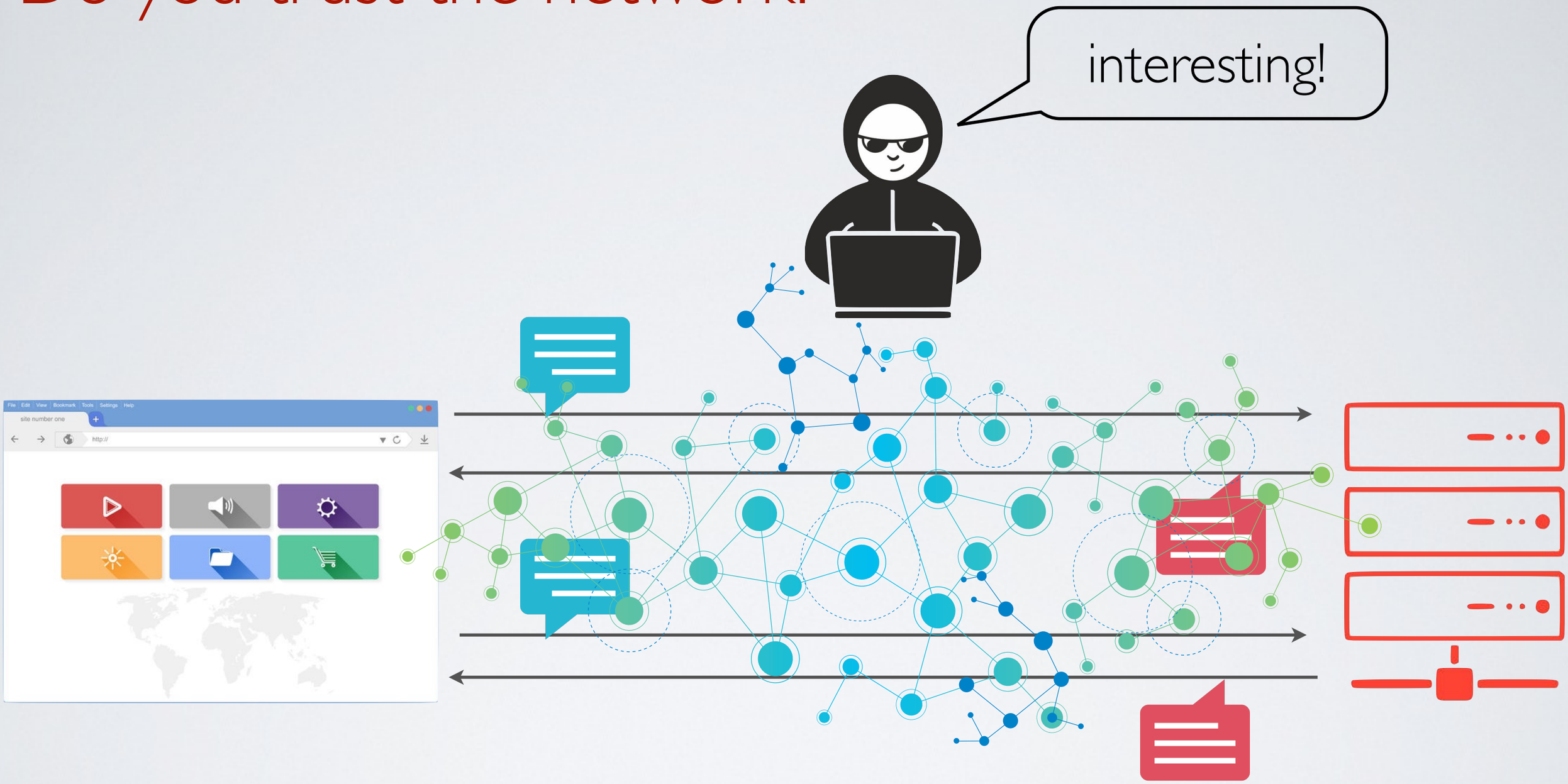
How to steal user's credentials

- Brute force the user's password or session ID



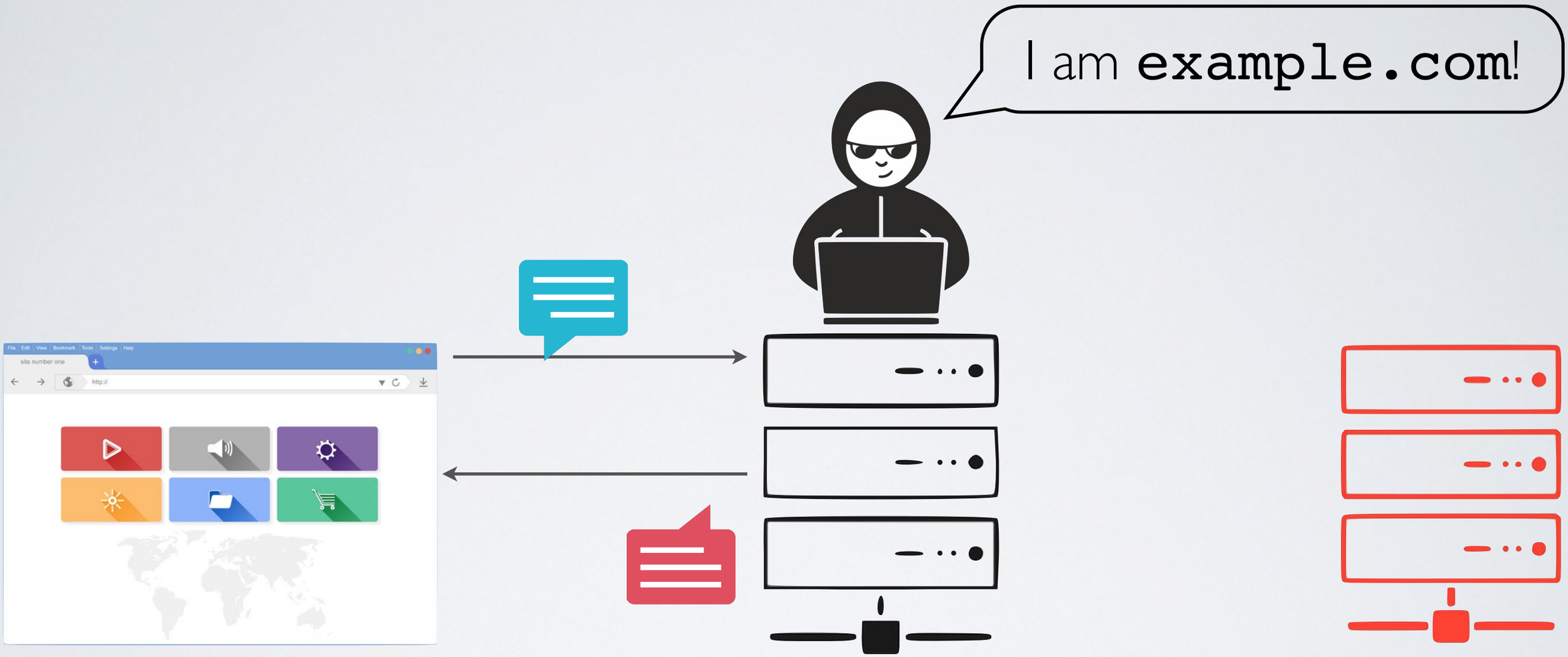
- Steal the user's password or session ID

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?

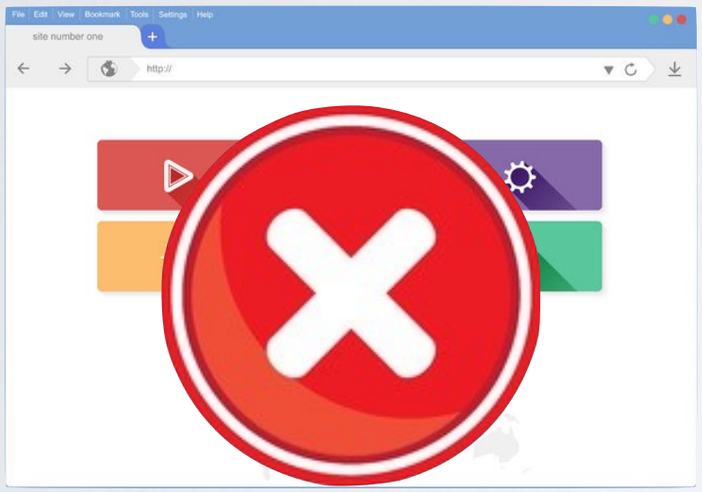
Generic solution - HTTPS

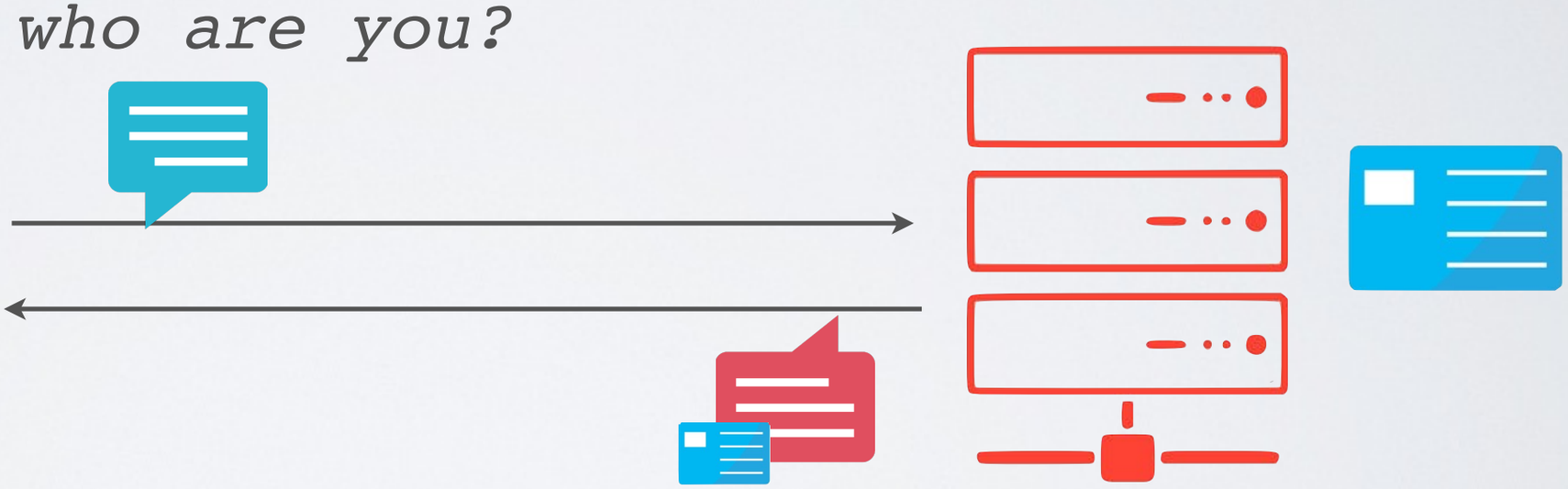
✓ HTTPS = HTTP + TLS

➡ Transport Layer Security (TLS previously known as SSL) provides

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

Generating and using (self-signed) certificates





Self-signed certificates are not trusted by your browser

This Connection is Untrusted

You have asked Firefox to connect securely to **www.domainname.tld** but we can't confirm that your connection is secure.

Normally, when you try to connect securely, sites will present trusted identification to prove that you are going to the right place. However, this site's identity can't be verified.

What Should I Do?

If you usually connect to this site without problems, this error could mean that someone is trying to impersonate the site, and you shouldn't continue.

Get me out of here!

Technical Details

I Understand the Risks

If you understand what's going on, you can tell Firefox to start trusting this site's identification. **Even if you trust the site, this error could mean that someone is tampering with your connection.**

Don't add an exception unless you know there's a good reason why this site doesn't use trusted identification.

Add Exception...

Your connection is not private

Attackers might be trying to steal your information from **bitbucket.org** (for example, passwords, messages, or credit cards).

Hide advanced

Reload

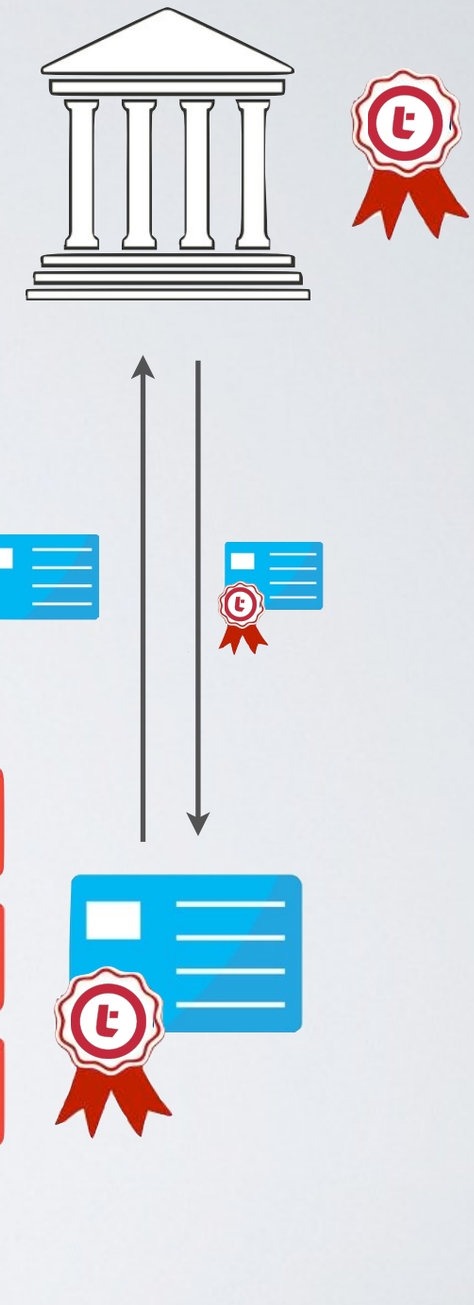
bitbucket.org normally uses encryption to protect your information. When Chrome tried to connect to bitbucket.org this time, the website sent back unusual and incorrect credentials. Either an attacker is trying to pretend to be bitbucket.org, or a Wi-Fi sign-in screen has interrupted the connection. Your information is still secure because Chrome stopped the connection before any data was exchanged.

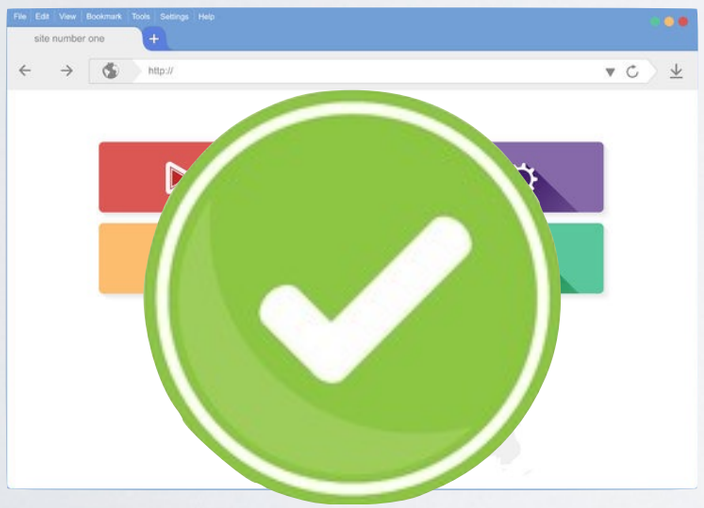
You cannot visit bitbucket.org right now because the website uses HSTS. Network errors and attacks are usually temporary, so this page will probably work later.

NET::ERR_CERT_DATE_INVALID

Signed Certificate

Certificate Authority (CA)

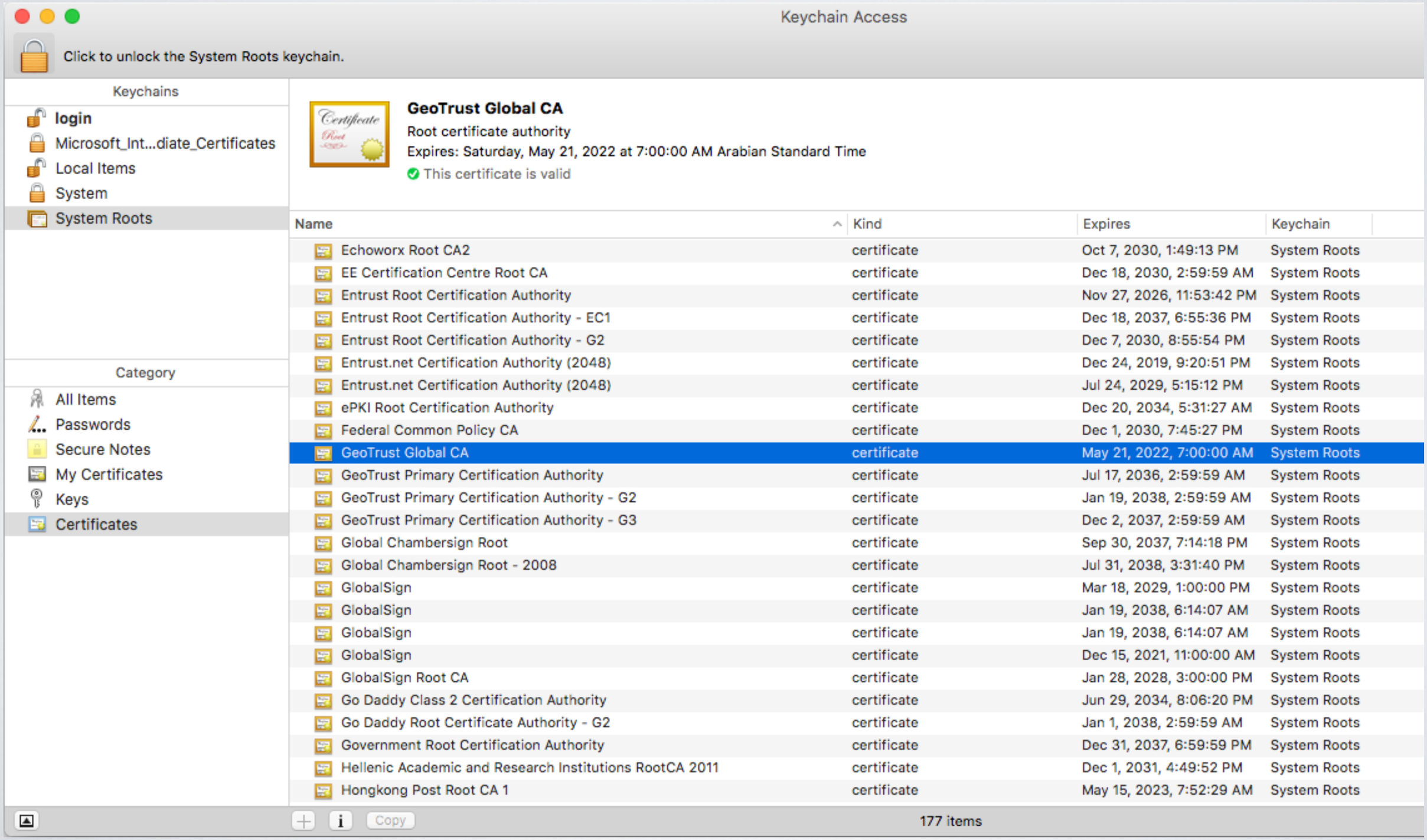




who are you?

I am example.com

Your browser trusts many CAs **by default**



Real attacks

Google Security Blog

The latest news and insights from Google on security and safety on the Internet

An update on attempted man-in-the-middle attacks

August 29, 2011

Posted by Heather Adkins, Information Security Manager

Today we received reports of attempted SSL man-in-the-middle (MITM) attacks against Google users, whereby someone tried to get between them and encrypted Google services. The people affected were primarily located in Iran. The attacker used a fraudulent SSL certificate issued by DigiNotar, a root certificate authority that should not issue certificates for Google (and has since revoked it).

Google Chrome users were protected from this attack because Chrome was able to detect the fraudulent certificate.

Google Security Blog

The latest news and insights from Google on security and safety on

Enhancing digital certificate security

January 3, 2013

Posted by Adam Langley, Software Engineer

Late on December 24, Chrome detected and blocked an unauthorized digital certificate for the "*.google.com" domain. We investigated immediately and found the certificate was issued by an intermediate certificate authority (CA) linking back to TURKTRUST, a Turkish certificate authority. Intermediate CA certificates carry the full authority of the CA, so anyone who has one can use it to create a certificate for any website they wish to impersonate.

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

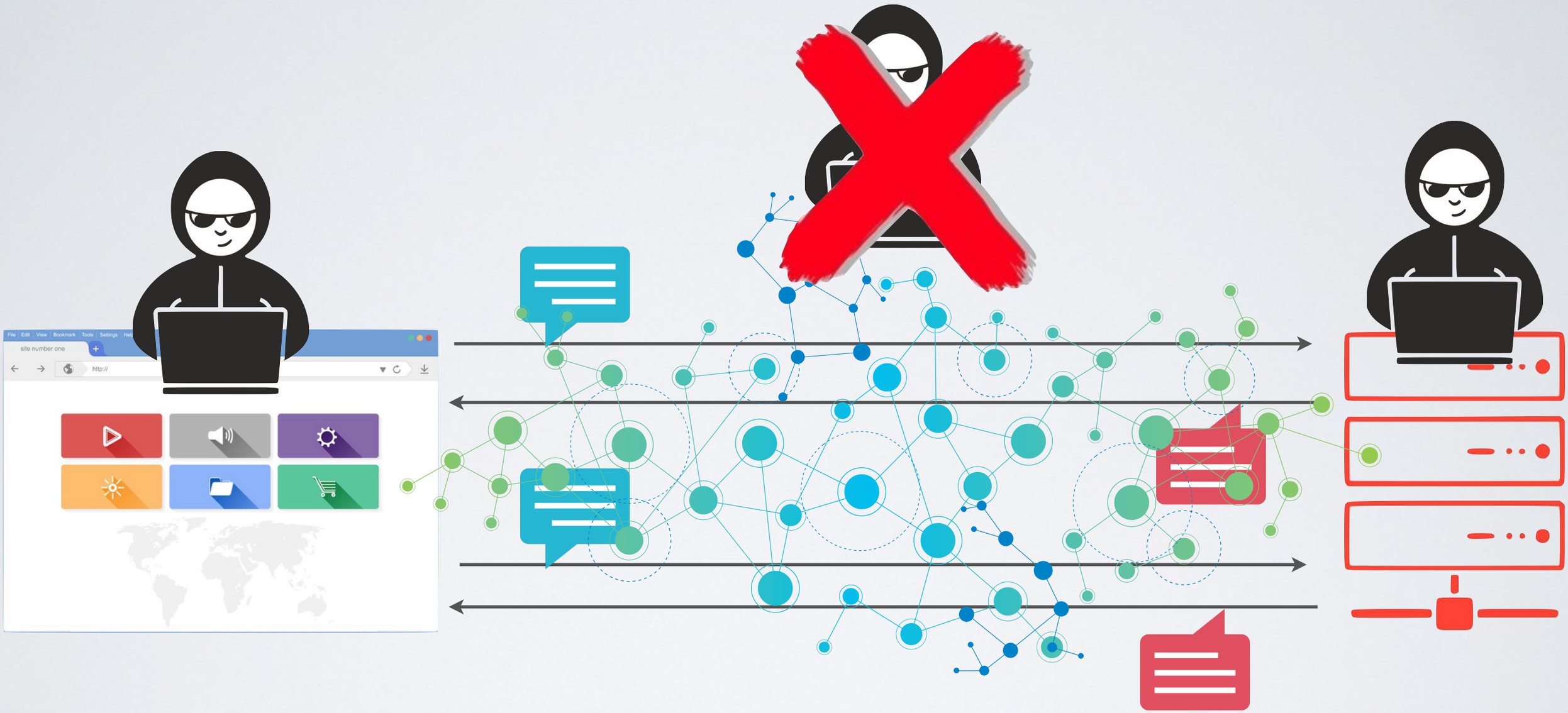
✓ The cookie will be sent over HTTPS exclusively

➡ 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

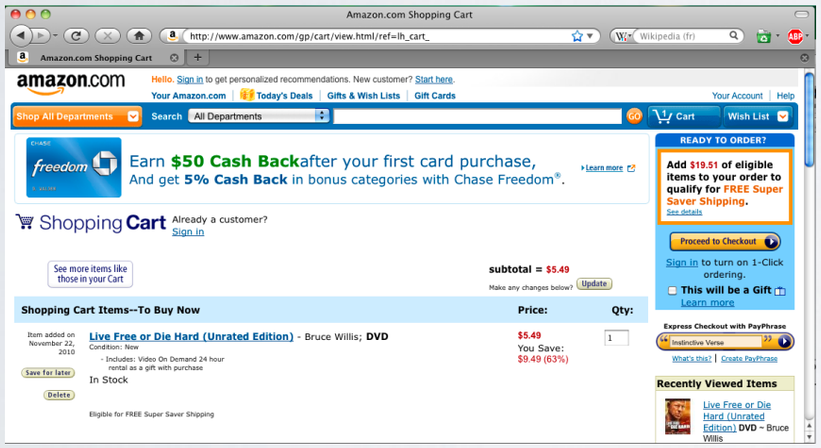


Problem

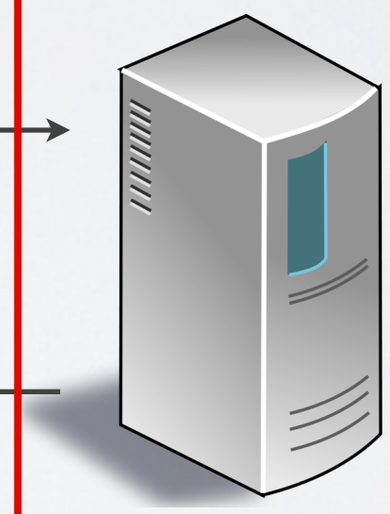
You have **absolutely no control** on the client and the network

Client Side

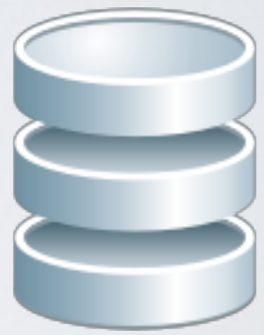
Server Side



Web Browser



Web Server



Database

Beyond HTTPS - attacking the web application

Frontend Vulnerabilities

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

Backend Vulnerabilities

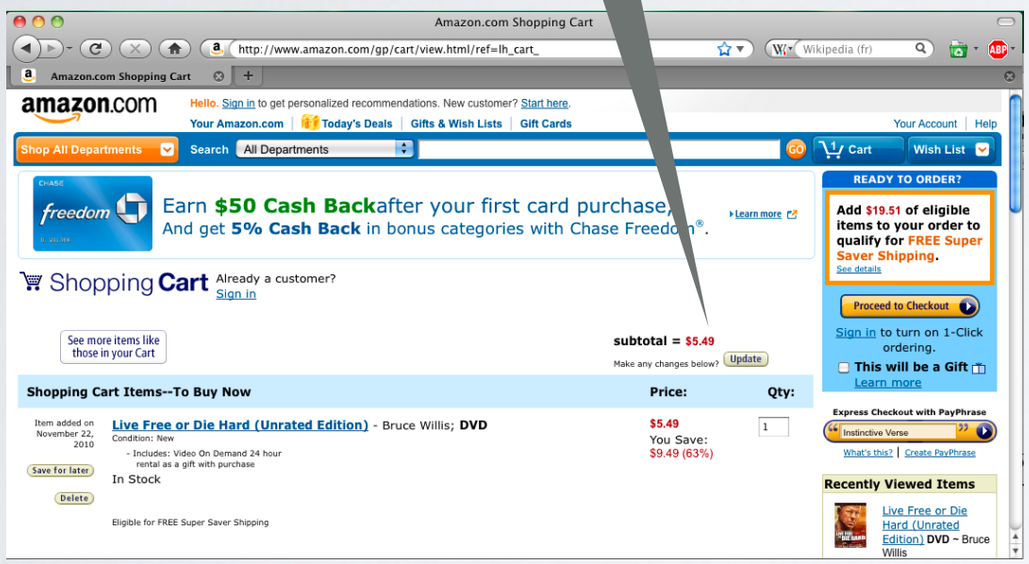
- Incomplete mediation
- Information leakage
- SQL injection

Incomplete Mediation

The Shopping Cart Attack

The total is calculated by a script on the client

The order is generated based on the request



*

amazon.com

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

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*

Solution

✓ Authentication

- Who are the authorized users?

✓ Authorization

- Who can access what and how?

SQL Injection

Problem

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

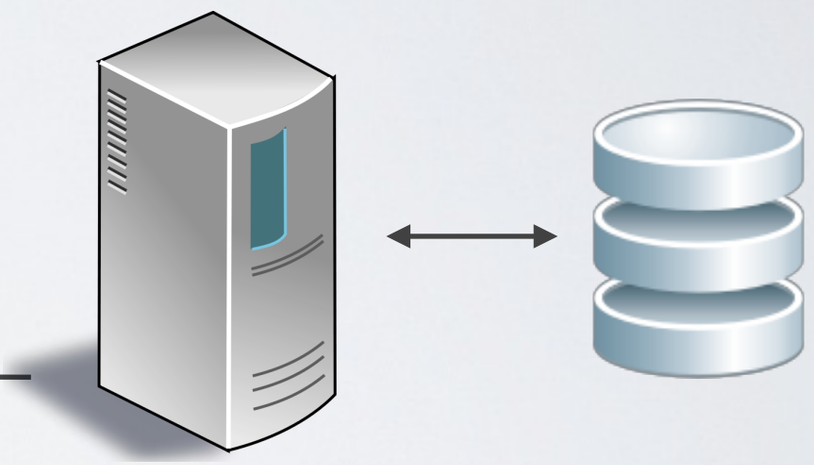
Checking password

`signin.html`

`/signin/`

name=Alice&pwd=pass4alice

Access Granted!



SQL Injection

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

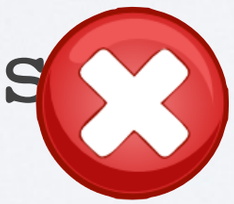
```
username: alice
password: pas lice
```

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

NoSQL Injection

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

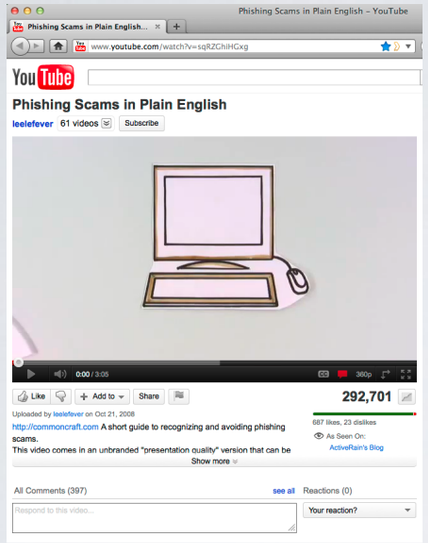
```
username: alice
password: pas lice
```



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

Content Spoofing

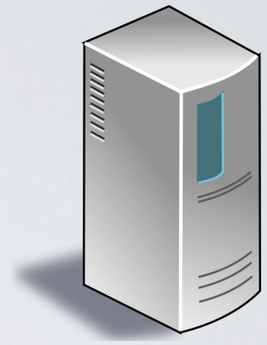
Content Spoofing

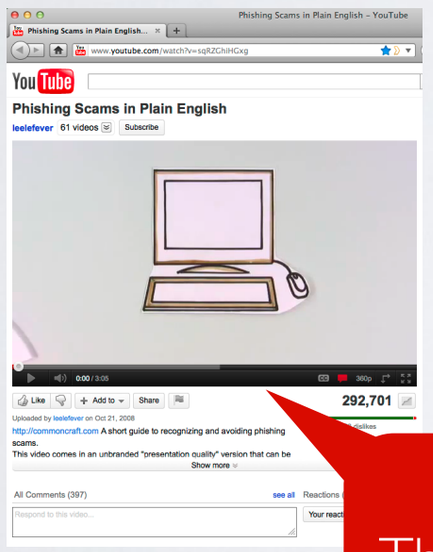


GET /?videoid=527

<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

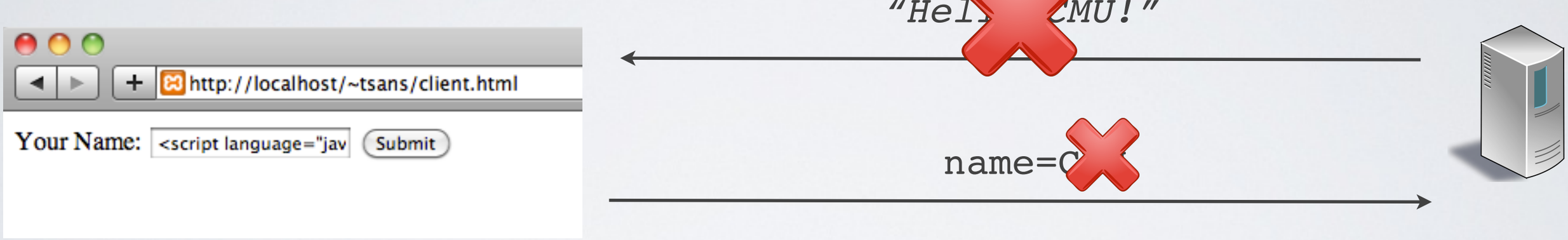
- ➡ An attacker can inject HTML tags in the page
  - Add illegitimate content to the webpage (ads most of the time)

Generic Solution

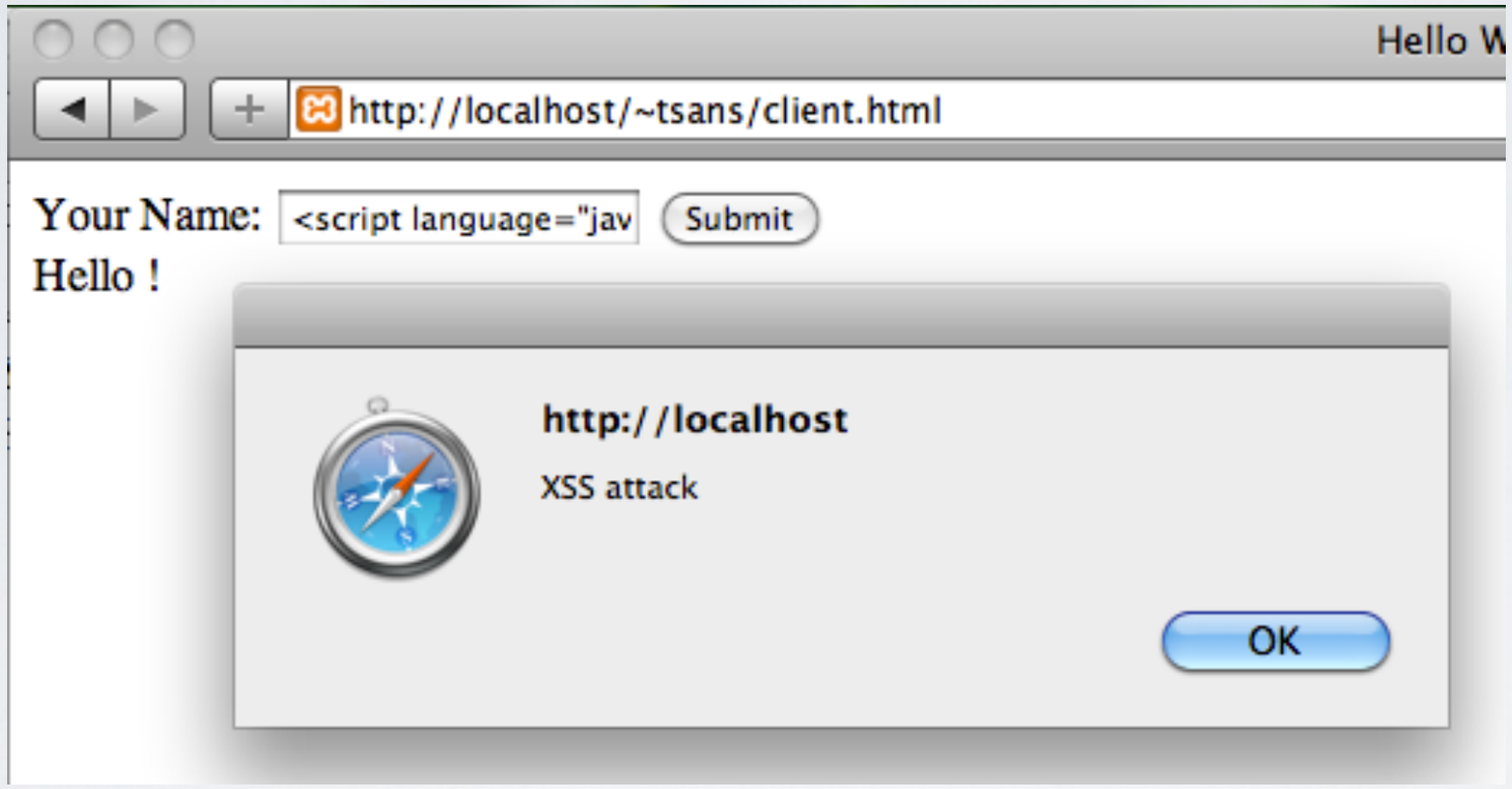
- ✓ Data inserted in the DOM must be validated

Cross-Site Scripting (XSS)

Cross-Site Scripting Attack (XSS attack)



XSS Attack = Javascript Code Injection

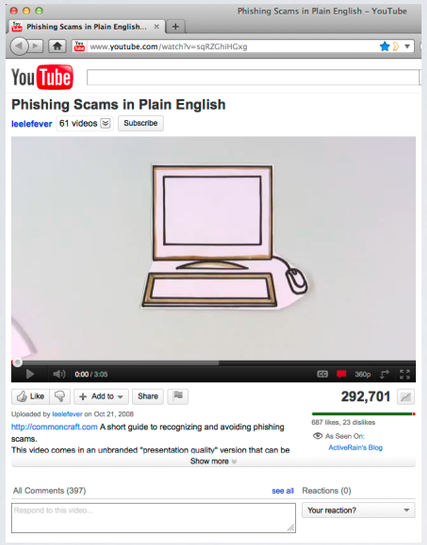


Problem

➡ 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

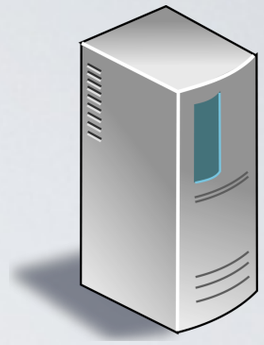
Forging a perfect scam

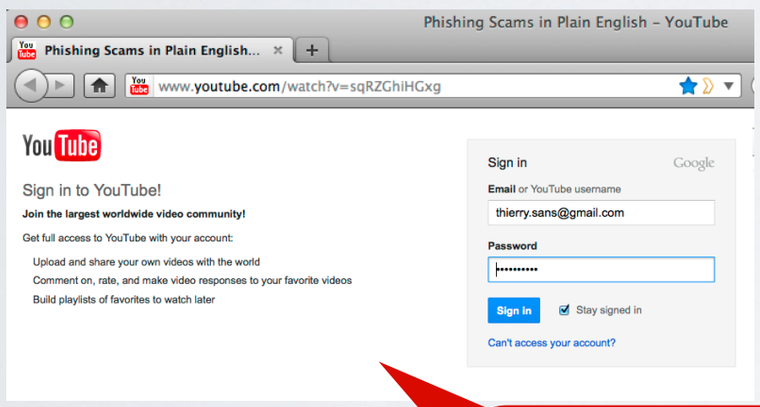


GET /?videoid=527

<html ...

comment = “<script> ...





GET /?videoid=527

<html ...

login=Alice&password=123456

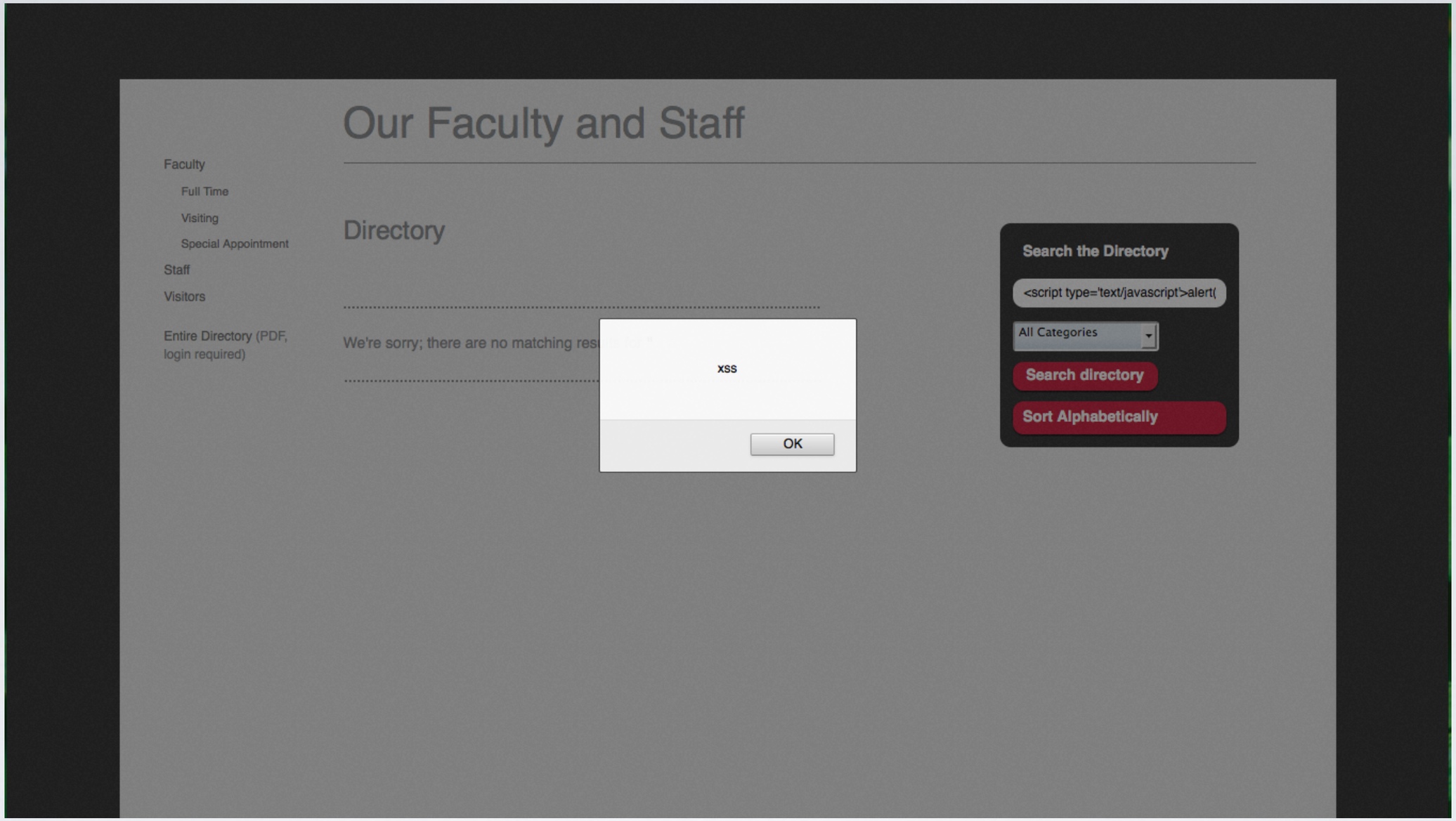
The script contained in the comments modifies the page to look like the login page!

** 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

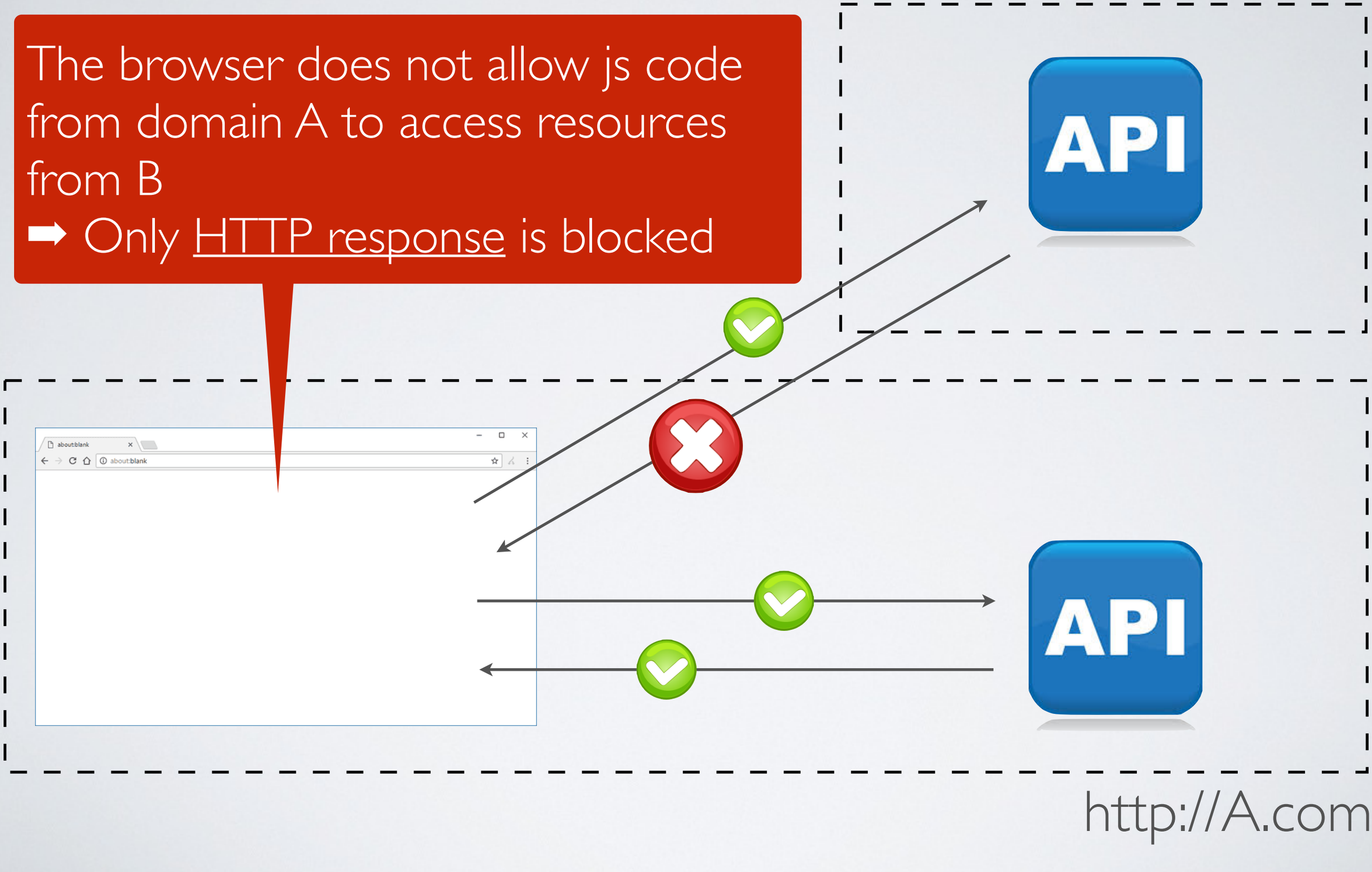
➡ Prevents the authentication cookie from being leaked when an XSS attack (cross-site scripting) occurs

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)

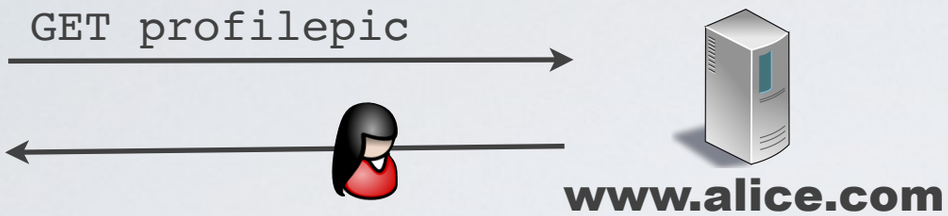
Problem

➡ 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

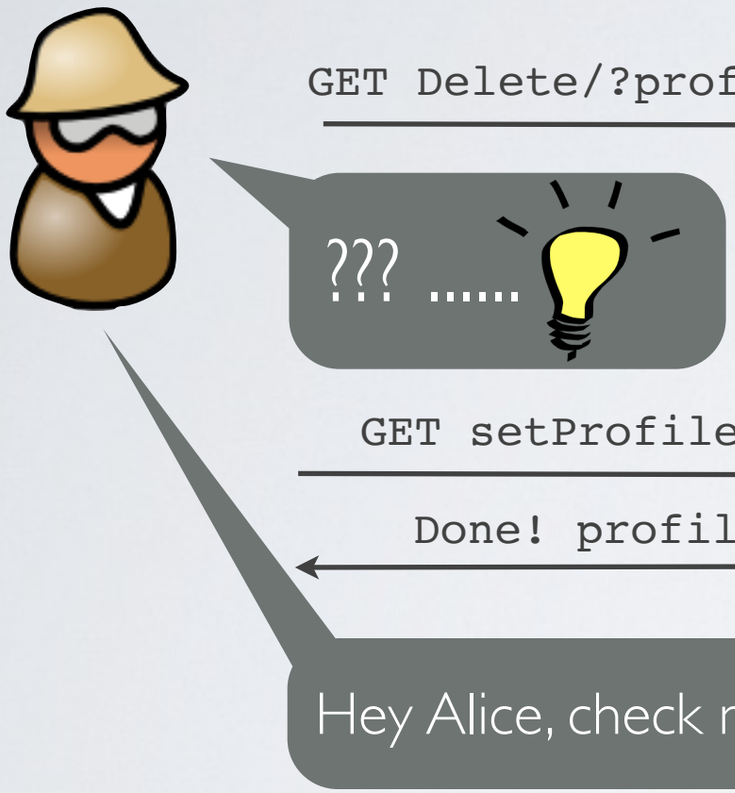
GET View/?profileid=53

GET profilepic



www.alice.com

GET Delete/?profileid=53



GET setProfile/?url=Delete/?profileid=53

Done! profileid=86

Hey Alice, check my profile

GET View/?profileid=86

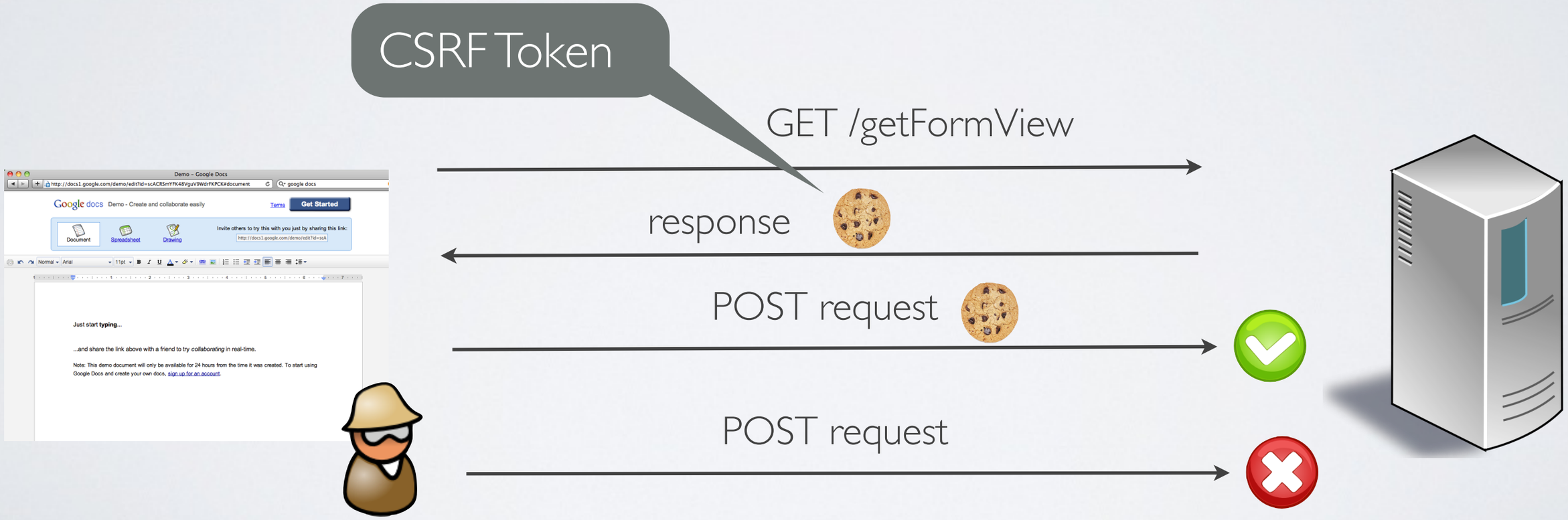
GET Delete/?profileid=53

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)