SOP & Web Attacks

Prateek Saxena

December 2nd, 2011, 10:54 GMT · By **Eduard Kovacs**

United Nations Refugee Agency Hacked, Barack Obama's Credentials Leaked

Android Market Security Alert: Vulnerability Market allowed Hackers Unauthorized Installation of Apps

June 3rd, 2011, 07:27 GMT · By **Lucian Constantin**

Sony Pictures Hacked, Millions of Accounts Exposed

Thursday, October 13, 2005

How to Make 1 Million Friends on MySpace

Web Vulnerabilities: Threat or Menace?

Severity Rating of Software Errors (2011)

Web

Vulnerability	Score
SQL Injection	93.8
Command Injection	83.8
Buffer Overflow	79.0
Cross-site Scripting	77.7
Authentication	76.9
Missing Authorization	76.8

Source: CWE Common Weakness Enumeration
A Community-Developed Dictionary of Software Weakness Types

Web Basics

URL

https://**www.nus.edu.sg**:**443**/~prateeks/add.php?q=x& user=me

PROTOCOL (Scheme)

HOST (Domain)

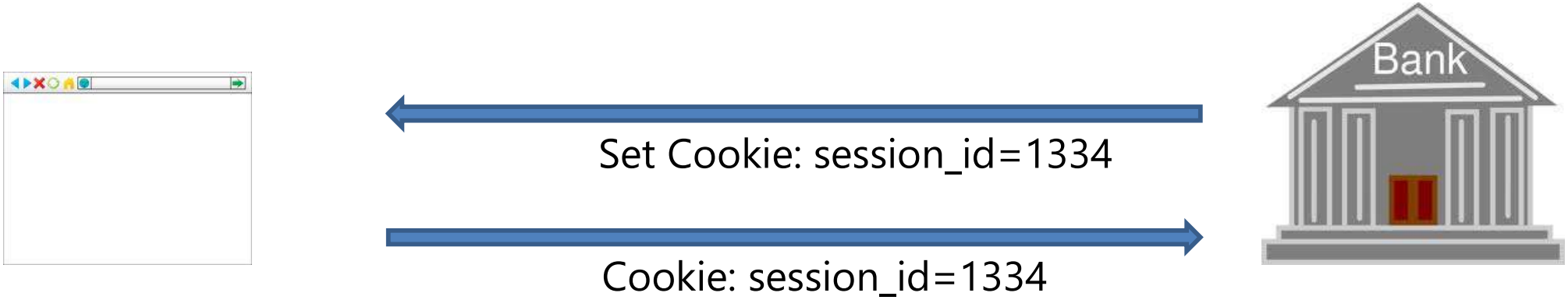
PORT

WEB ORIGIN

PATH

HTTP PARAMETERS

Cookies



Uses to persist state on the client, maintain sessions, etc…

HTML

Developer Tools - http://www.comp.nus.edu.sg/~prateeks/teaching/sp14/cs5331-sp14.html

Elements Resources Network Sources Timeline Profiles Audits Console

Name
Path

cs5331-sp14.html
/~prateeks/teaching/sp14

coursepage.css
/~prateeks/teaching/sp14

Headers Preview Response Cookies Timing

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
                "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
lang="en" xml:lang="en">
<head>
<title>CS5331 - Web Security</title>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8"/>
<link rel="stylesheet" type="text/css" href="coursepage.css" />
</head>
<body>
<div class="banner">
<h1>CS5331 - Web Security</h1>
</div>
<div id="container" style="width:50%" align="left">

<table cellpadding="5" cellspacing="10" align="center" width="100%">
  <tbody>
        <tr>
        <td><a href="#sec-1">Course Description</a></td>
            <td><a href="#sec-3">Class Logistics &amp; Grading</a></td>
            <td><a href="#sec-6">Topics</a> </td>
            <td><a href="#sec-7">Important Dates</a> </td>
            </tr>

  </tbody>
</table>

</div>

<div id="container" style="width:80%; padding:2%" align="left">

<table cellpadding="1px">
```

JavaScript

```
<!DOCTYPE html>
<html>

<body>

    <h1>My First JavaScript</h1>
    <p>Click the button to display the date.</p>
    <p id="demo"></p>

    <button type="button" onclick="myFunction()">Try it</button>

    <script>
        function myFunction() {
            document.getElementById("demo").innerHTML = Date();
        }
    </script>

    <script src="/scripts/foo.js"></script>

</body>

</html>
```

Frames / Windows

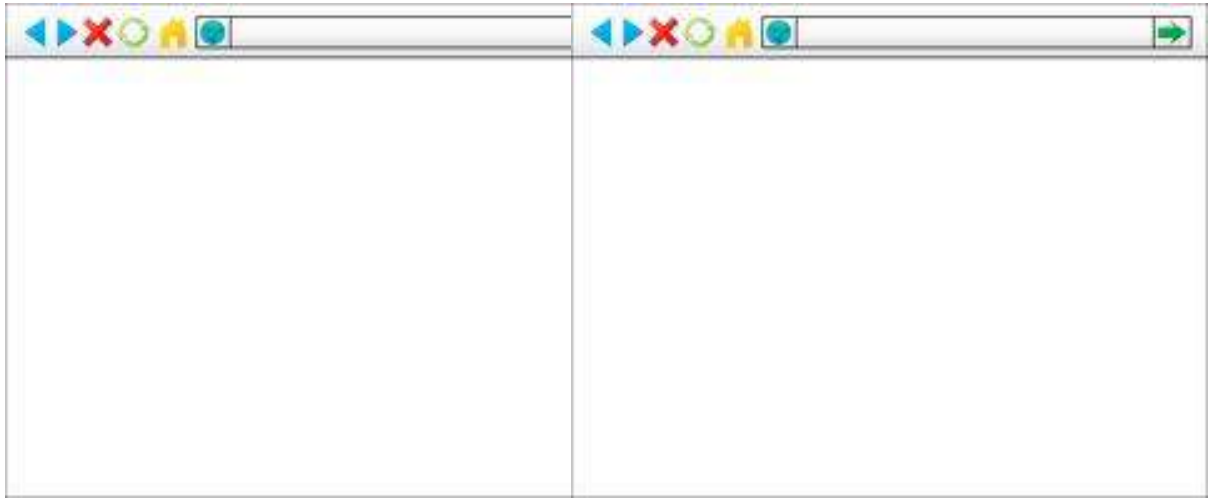
- Each window is a frame
  - A frame hosts a web origin

- Iframes: Inline frame
  - Can host a different site
    - May be hidden (0px width-ht), no borders, transparent

parent frame is unable to access the content of the site hosted inside the iframe due to same origin borders policy

The Same Origin Policy

http://evil.com http://google.com



No direct access between these frames !

https://**www.nus.edu.sg**:**443**/~prateeks/add.php?q=x

PROTOCOL HOST (Domain) PORT

WEB ORIGIN = PROTOCOL + HOST+ PORT

1. Same-origin policy [Wikipedia]
2. RFC 6454

The Same Origin Policy: Web Origins Quiz

Match following URLs to the origin
http://www.example.com/dir/page.html

Compared URL	Outcome	Reason
http://www.example.com/dir/page2.html	success - Same protocol and host	
http://www.example.com/dir2/other.html	success - Same protocol and host	
http://username:password**@www.example.com**/dir2/other.html	success - Same protocol and host	
http://www.example.com:**81**/dir/other.html	failure - Same protocol and host but different port	
https://www.example.com/dir/other.html	failure - Different protocol	
http://**en.example.com**/dir/other.html	failure - Different host	
http://**example.com**/dir/other.html	failure - Different host (exact match required)	
http://**v2.www.example.com**/dir/other.html	failure - Different host (exact match required)	
http://www.example.com:**80**/dir/other.html	Port explicit. Depends on the implementation of the browser	

The Same Origin Policy

Problem: Web Origin Isolation

Security Goals of a Web Browser

- 2 Kinds of Isolation
  - Prevent network content to access OS resources
    - E.g. Installing EXEs, Camera, GPS,…
    - *Can you think of exceptions?*
  - *Isolate Web Sites from each other*
    - *Via the "same origin policy"*

The Web Attacker Threat Model

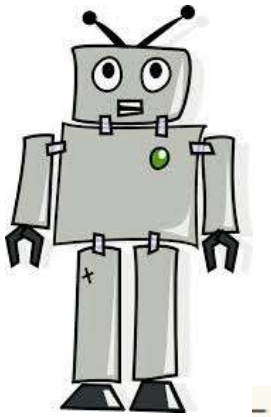
- Strictly weaker than a network attacker
- **Web Attacker (Definition)**
  - Owns a valid domain, server with an SSL certificate
  - Can entice a victim to visit his site
    - Say via "Click Here to Get a Free iPad" link
    - Or, via an advertisement (no clicks needed)
  - Can't intercept / read traffic for other sites.
- Assumptions:
  - Network channel is secure
  - Browser is secure
- Attacks: The web application is **benign-but-buggy**

Server-side Attacks:
SQL Injection

SQLI:
The Idea

Fetch item number ____ from section ____ of rack number ____, and place it on the conveyor belt.

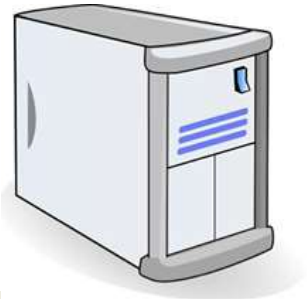
Fetch item number **1234** from section **B2** of rack number **12**, and place it on the conveyor belt.



Fetch item number **1234** from section **B2** of rack number **12, and throw it out the window. Then go back to your desk and ignore the rest of this form.** and place it on the conveyor belt.

From [StackExchange]

SQLI: Example



http://bank.com?login.php?uname=Joe&pw=ss98

```
$sql = "SELECT * FROM USERS
WHERE name='" . $uname . "'
and  passwd='". $pw . "'";
$rs = $db->execute($sql);
```

http://ba....?uname=admin%27--&pw=%20

```
SELECT * FROM USERS
WHERE name='admin'-- and passwd=' '
```

SQLI:
More Tricks

- There are (again) many attack vectors
  - SQL Injection cheat sheet
- Easier to get right than XSS
  - Beware of character set encoding
  - See rules for string literals (e.g. MYSQL)
  - Varies by database engine

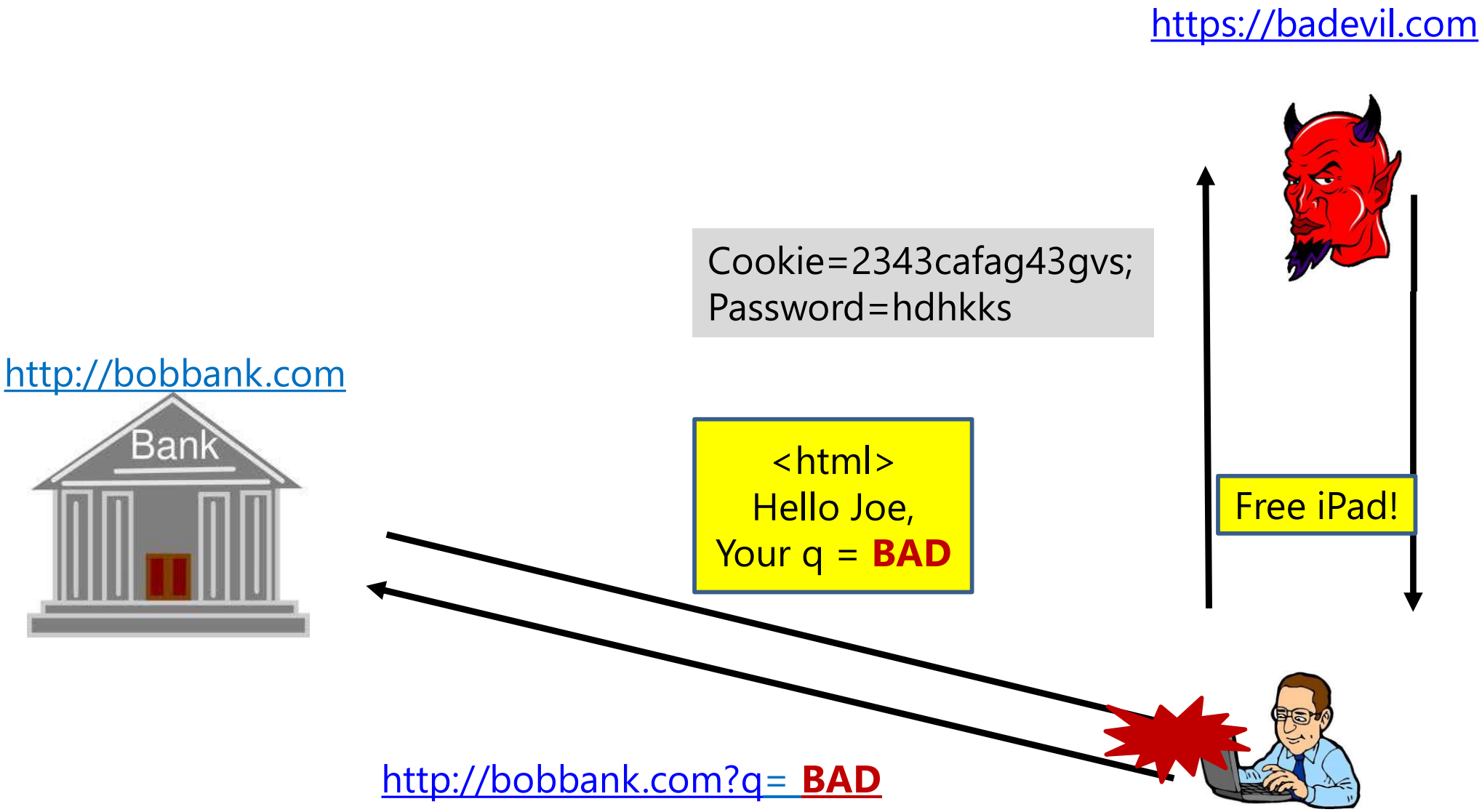
SQLI: Some Defenses

- `magic_quotes_gpc()` (on by default)
  - Runs input through `addslashes()`
    - E.g. `')- admin` becomes `\')- admin`
  - Still unsafe
    - E.g `SELECT * FROM X WHERE id=$post_id`
      - E.g. `0 or 1=1`
    - Native Character set issues
- A Good Solution: Prepared Statements

```
$stmt = $dbh->prepare("SELECT * FROM users
        WHERE USERNAME = ? AND PASSWORD = ?")
$stmt->execute(array($username, $password));
```

Script Injection Attacks:
Reflected XSS

Cross-site Scripting Attacks (I): Reflected XSS

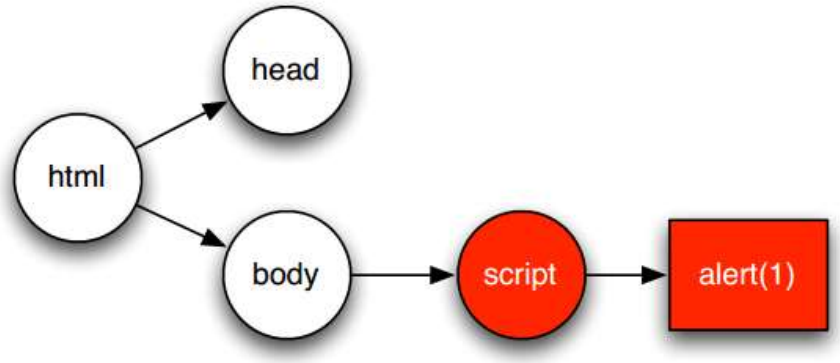


Cross-site Scripting Attacks (I): Reflected XSS

http://bobbank.com?q=<script>alert('hi')</script>

```
<html>
<head> … </head>
        <body>
        Hello Joe, Your q =
        <script>alert('hi')</script>

</body>
</html>
```



Browser's internal DOM

Defenses Against Script Injection: Sanitization & Filtering

How Can We Defeat This Attack?

Idea 1:

- Sanity check server outputs & inputs

```
String Img.RenderControl()
{
        echo Sanitize(userimg);
}
```

Sanitization

Yes, you should. But, it isn't enough! Why?

- There are too many subtle attack vectors…
- Vary a lot across browsers

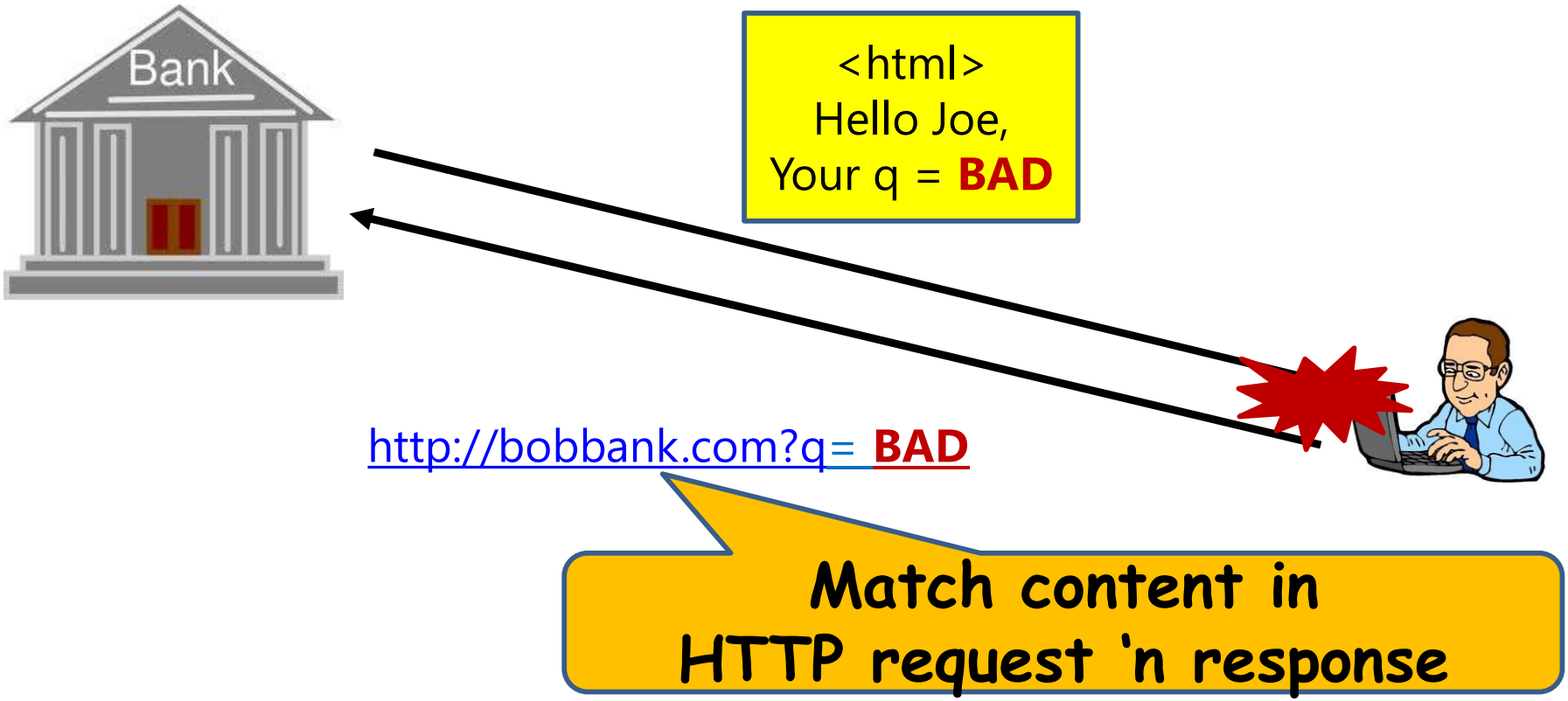
XSS Filter Evasion Cheat Sheet

HTML5 Security Cheatsheet

How Can We Defeat This Attack?

Another idea:

- Browser-side Filtering (e.g. XSS Auditor)

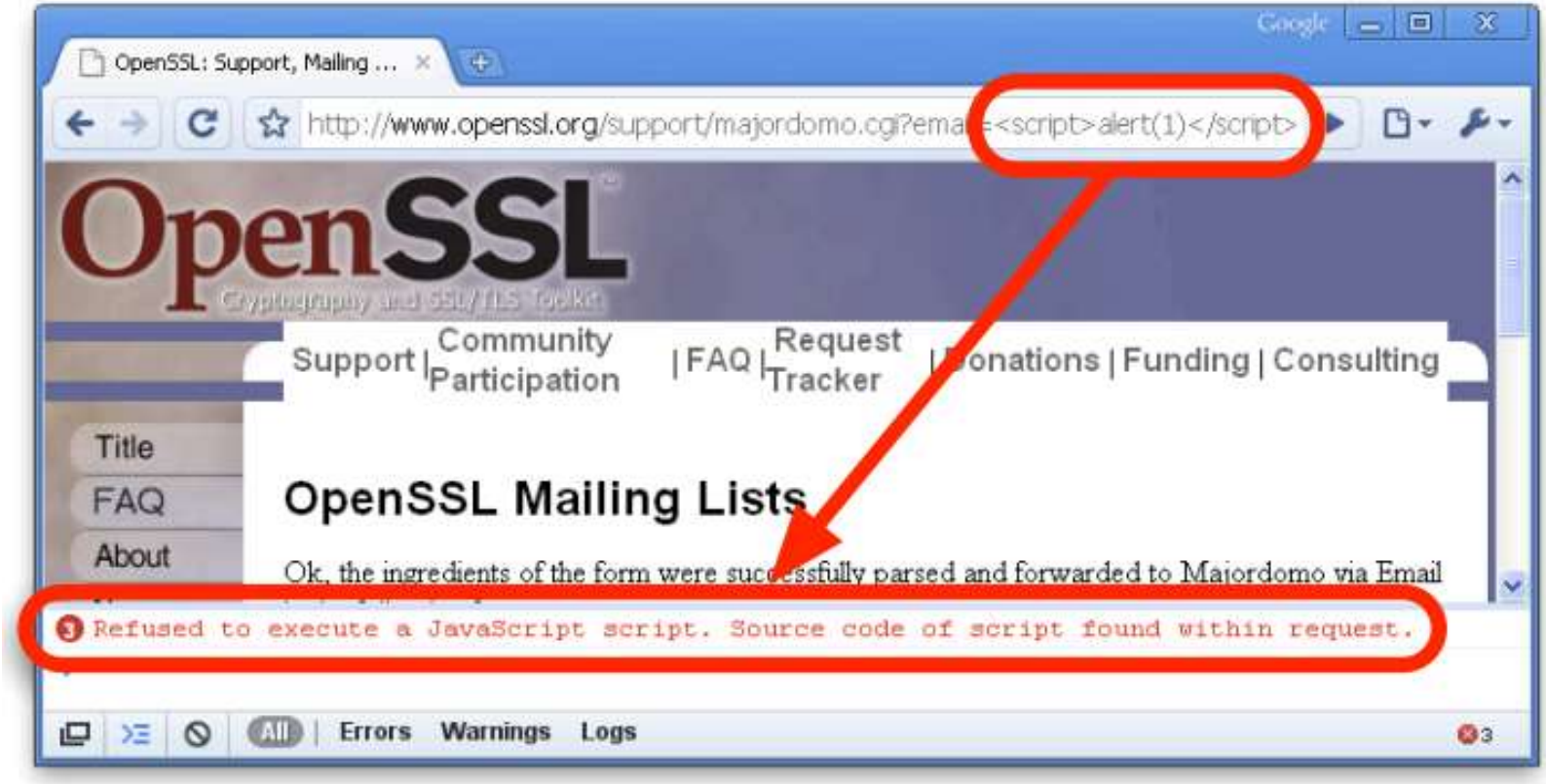


XSS Auditor: Regular Expressions Considered Harmful in Client-side XSS filters

How Can We Defeat This Attack?

Another idea:

- Browser-side Filtering (e.g. XSS Auditor)



XSS Auditor: [Regular Expressions Considered Harmful in Client-side XSS filters]

How Can We Defeat This Attack?

Another idea:

- Browser-side Filtering (e.g. XSS Auditor)
- Better to do the matching after parsing

```
00000000: 3c 68 74 6d 6c 3e 0a 3c 68 65 61 64 3e 0a 3c 2f <html>.<head>.</
00000010: 68 65 61 64 3e 0a 3c 62 6f 64 79 3e 0a 2b 41 44 head>.<body>.+AD
00000020: 77 41 63 77 42 6a 41 48 49 41 61 51 42 77 41 48 wAcwBjAHIAaQBwAH
00000030: 51 41 50 67 42 68 41 47 77 41 5a 51 42 79 41 48 QAPgBhAGwAZQByAH
00000040: 51 41 4b 41 41 78 41 43 6b 41 50 41 41 76 41 48 QAKAAxACkAPAAvAH
00000050: 4d 41 59 77 42 79 41 47 6b 41 63 41 42 30 41 44 MAYwByAGkAcAB0AD
00000060: 34 2d 3c 2f 62 6f 64 79 3e 0a 3c 2f 68 74 6d 6c 4-</body></html>
```

Figure 3: Identifying scripts in raw responses requires understanding browser parsing behavior.

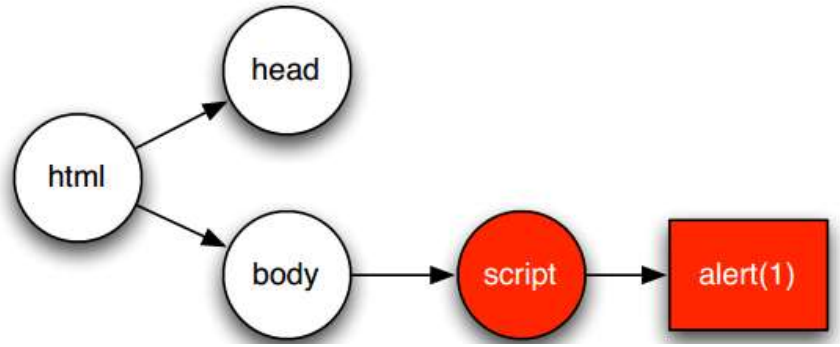
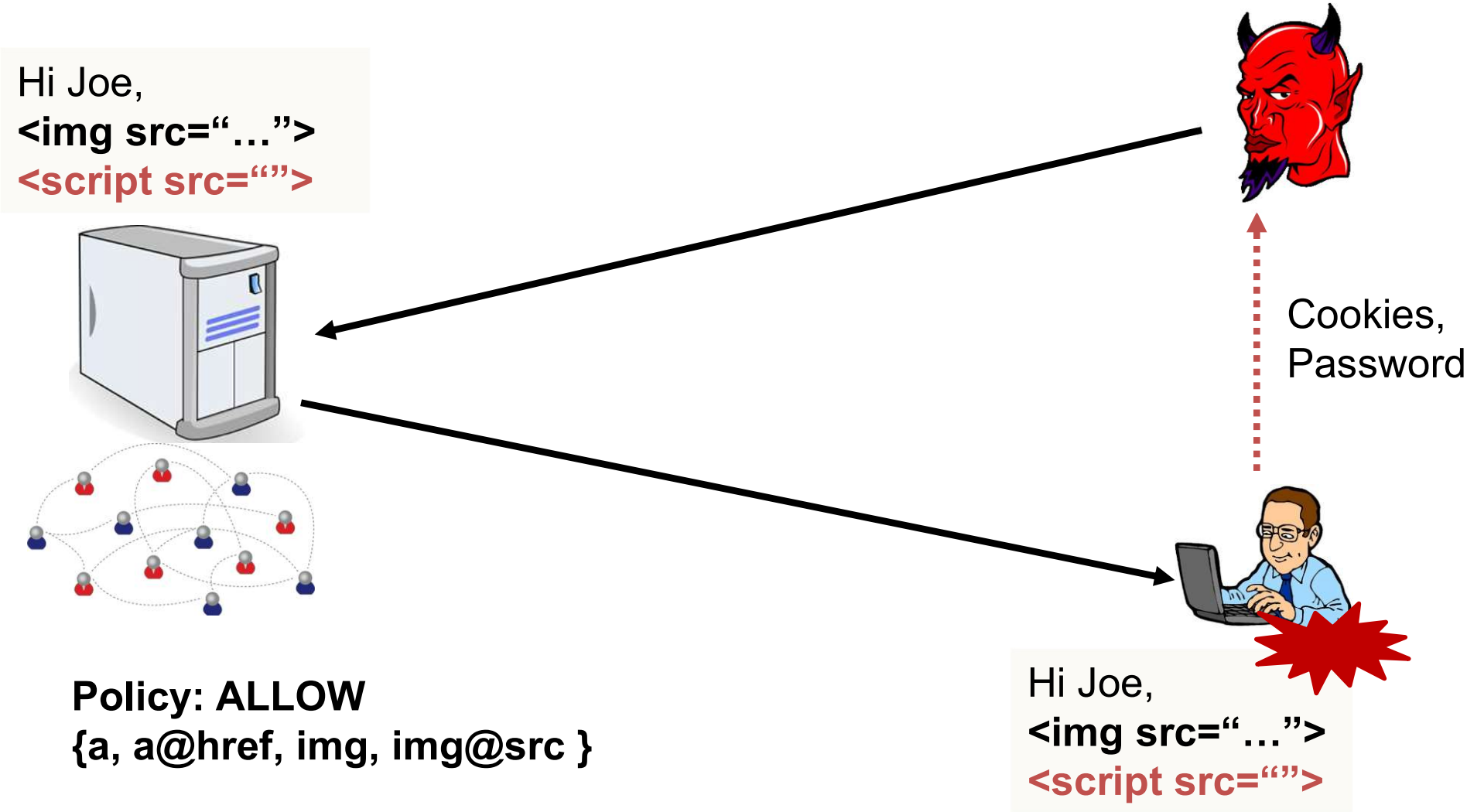


Figure 4: After the HTTP response is parsed, the script is easy to find.

XSS Auditor: [Regular Expressions Considered Harmful in Client-side XSS filters]

Script Injection Attacks:
Persistent XSS

Cross-site Scripting Attacks (II): Persistent XSS

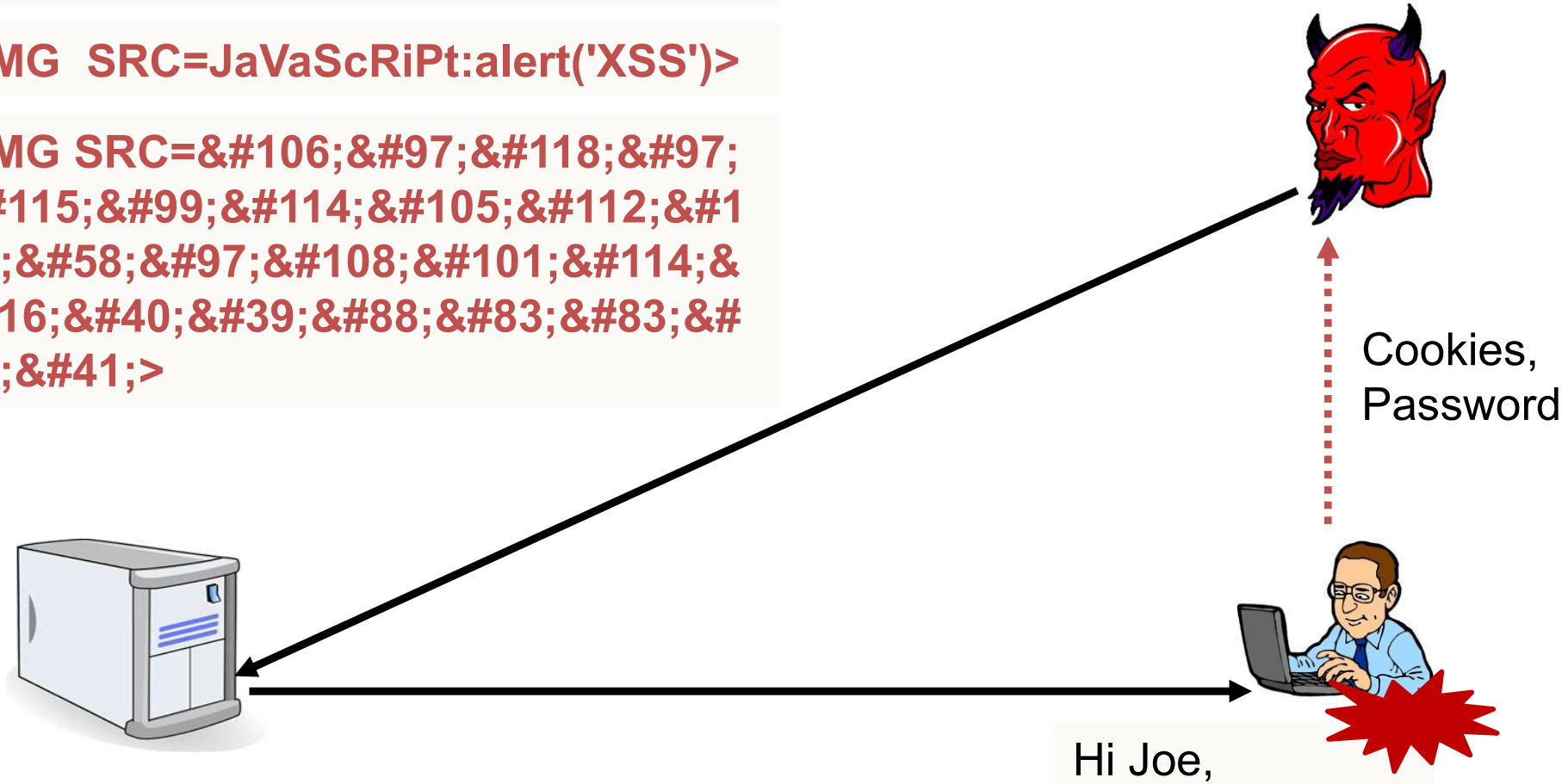


Cross-site Scripting Attacks (II): Persistent XSS

```
<IMG SRC="javascript:alert('XSS')”>
```

```
<IMG  SRC=JaVaScRiPt:alert('XSS')>
```

```
<IMG SRC=&#106;&#97;&#118;&#97;
&#115;&#99;&#114;&#105;&#112;&#1
16;&#58;&#97;&#108;&#101;&#114;&
#116;&#40;&#39;&#88;&#83;&#83;&#
39;&#41;>
```



Policy: ALLOW
{a, a@href, img, img@src }

Quiz

- Does the XSSAuditor approach defeat persistent XSS?

No, it checks the request and the response to see if there are any match in the string - which might indicate a script running

Script Injection Attacks:
DOM-based XSS

Cross-site Scripting Attacks (III): DOM-based XSS

http://twitter.com#!' onerror=bad()

’ onerror=bad()..

http://twitter.com#!

JavaScript Attribute

x = “”;

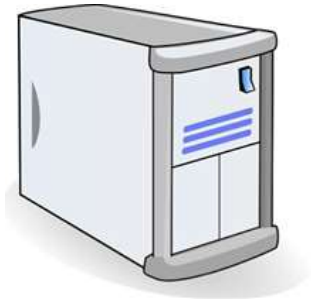
n.innerHTML = x;

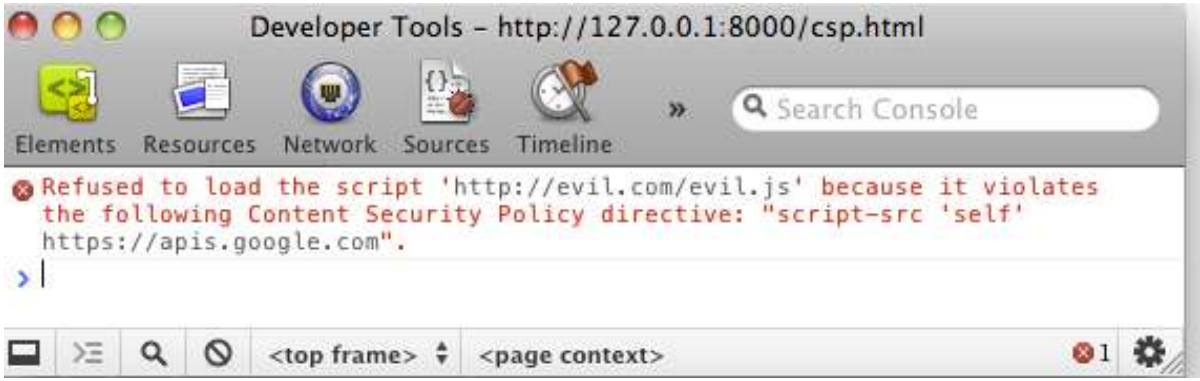
Injection in JS accessing the DOM

Defenses Against Script Injection: Content Security Policy

XSS Defenses:
CSP

```
Content-Security-Policy: script-src 'self' https://apis.google.com
```

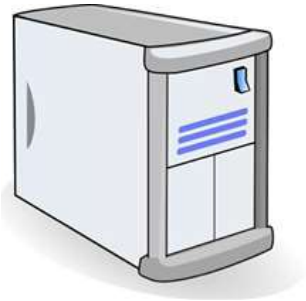




Server tells the browser the "Whitelisted" script sources. Browser denies everything outside the whitelist.

XSS Defenses: CSP

```
Content-Security-Policy: script-src 'self' https://apis.google.com
```



- `'none'`, as you might expect, matches nothing.
- `'self'` matches the current origin, but not its subdomains.
- `'unsafe-inline'` allows inline JavaScript and CSS (we'll touch on this in more detail in a bit).
- `'unsafe-eval'` allows text-to-JavaScript mechanisms like `eval` (we'll get to this too).

```
script>
 function doAmazingThings() {
   alert('YOU AM AMAZING!');
 }
/script>
button onclick='doAmazingThings();'>A
```

Disallowed

```
script src='amazing.js'></script>
button id='amazing'>Am I amazing?<
```

Allowed

Does it block all XSS attacks?
(Most people argue "yes". But, limitations have been found. No one knows fully yet!)