Web Security Project

Patrick Mutchler
cs155

New to Web Programming?

There are four different languages you need to know for this project

HTML, PHP, JS, CSS

Fortunately, there are tons of resources for learning about these technologies

HTML

Hyper**T**ext **M**arkup **L**anguage

Used to tell the browser the structure of a web page

HTML - Tags

Surrounded by angle brackets <>

- Header information: <head>
- Links to other documents: <a>
- Images: <img>
- Page regions: content here
- Input forms:

```
<form>
    <input type="text"/>
    <input type="submit" value="Submit"/>
</form>
```

HTML - Properties

Key=value pairs inside of a tag

Can represent URLs in different ways

- Relative: images/cats.jpg
- Server-relative: /images/cats.jpg
- Absolute: http://pictures.com/images/cats.jpg

Can delimit value with ' or " or nothing

HTML - iFrames

Embed HTML documents in other documents

```
<iframe name="myframe" src="http://www.google.com"/>
```

Very useful for this project

HTML - Forms

```
<form action = "/login.php">
   <input type="text" name="username">
   <input type="password" name="password>
   <input type="submit" value="Log in">
</form>
```

Use method='GET' for requests without side effects

Use method='POST' for requests with side effects

Use target='myframe' to avoid navigating away

CSS

Style information for html tags

Can set style as an html property

```
<span style="color: red"> Red text </span>
```

Can also set styles globally

```
#mynodeid {color:red;}
```

Javascript

Browser language for manipulating page content

Full of incredibly bad design choices (unrelated to this project)

Spec is poorly written and even more poorly followed

Invoking Javascript

```
<script> alert('hello world') </script>

<a href="javascript:alert('hello world')">

<button onclick="alert('hello world')">
```

Manipulating the DOM

```
document.getElementByID(id)
document.getElementsByTagName(tag)
document.write(htmltext)
document.createElement(tagname)
document.body.appendChild(node)
document.forms[index].fieldname.value = ...
document.formname.fieldname.value = ...
frame.contentDocument.getElementById(id)
someHTMLElement.innerHTML
```

Other Useful Functions

Navigation

```
document.location
document.formname.submit()
document.forms[0].submitfield.click()
```

Events

```
<script>
    var foo = document.getElementById('foo');
    foo.addEventListener('click', function(){
        alert('hello world');
    }, false);
</script>
```

Useful CSS

```
var node = document.getElementById('id');

node.style.display = 'none';
node.style.visibility = 'hidden';
node.style.position = 'absolute';

// all tags with id "mynodeid" are hidden
document.write("<style> \
    #mynodeid {visibility:hidden;} </style>"
```

PHP

Scripting language to produce HTML and code

```
<input value=<?php echo $myvalue; ?>>
```

Form data in global arrays $_GET, $_POST

Super Useful

Download Firebug for Firefox

Turn on the console for script and error logging

Allows you to manipulate the DOM manually from the client

Extremely useful for web development (and web attacks)

Attacks

Cross-Site Scripting (XSS)

Attacks 1, 4, and 5 are all variants

The most common web vulnerability

Browser executes JS controlled by the attacker in a context the attacker doesn't control

Cross-Site Scripting (XSS)

Famous example:

- Myspace Worm (2005)
- One million affected users in 20 hours

Recent example (Published last Wednesday):

- Stored XSS in Facebook chat
- Clicking a link in FB chat executes malicious Javascript

Cross-Site Scripting (XSS)

Three major types

- Reflected
- Stored
- DOM-Based

You will need to create Reflected and Stored attacks for Project 2.

Reflected XSS - How it works

```
<body> <?php
   $userid = $_GET['id'];
   echo("Your id is " . $userid);
?> </body>
```

www.vulnerable-site.com/?id=<script> ... </script>

```
<body>
   Your id is <script>...</script>
</body>
```

Stored XSS - How it works

```
<?php
   $profile = $_POST['profile'];
   DBStoreProfile('id', $profile)
   ...
   $id = $_GET['id'];
   $profile = DBReadProfile('id');
   echo("Requested profile is " . $profile);
?>
```

Requested profile is <script>...</script>

XSS - Javascript vectors

Lots more than just <script>

- onError, onLoad, etc in html tags
- javascript:
- eval (please don't ever use this)
- ...

Almost all blacklist filters can be defeated on at least some browsers

Cross-Site Request Forgery (CSRF)

Attack 2 is a CSRF attack

Executes commands on another website

Exploits the fact that the website doesn't check who is making the request

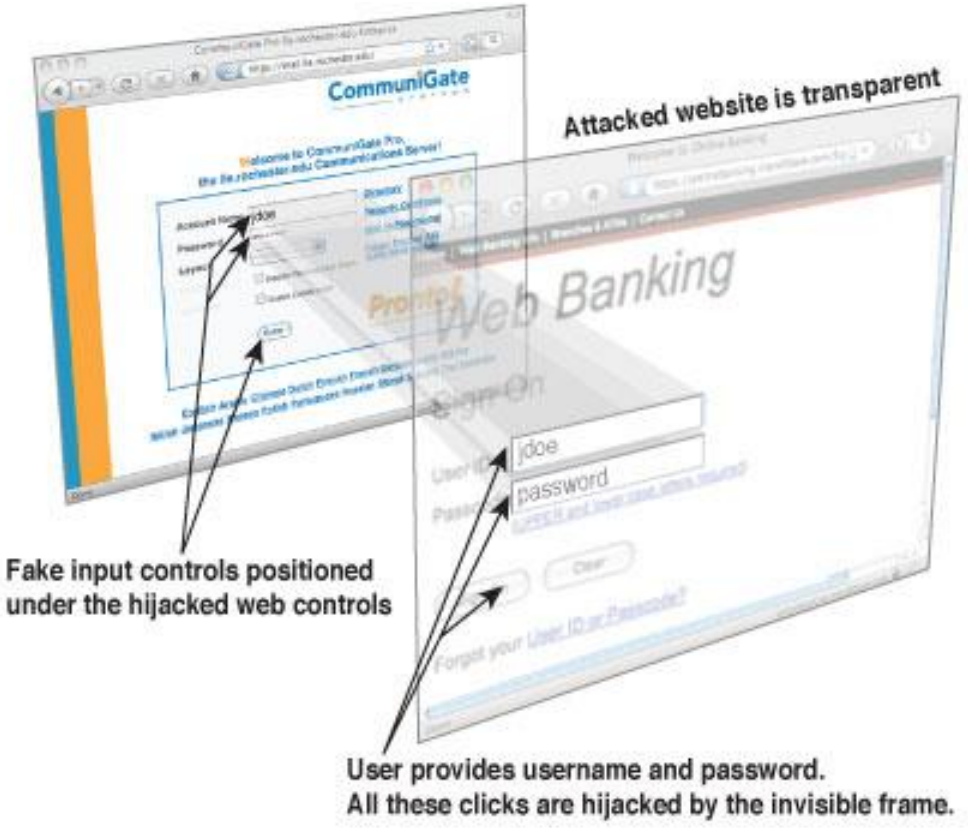
```
<img src="bank.com/withdraw?
acc=bob&amt=1000&for=eve">
```

Clickjacking

Attack 3 is a Clickjacking attack

Uses malicious styling and framing to trick users into interacting with another website

Clickjacking



Clickjacking

For attack 3 you will need to get a form to be filled out with malicious info before submitting

Several options

- Use GET and POST params
- Trick user into typing form data
- Lots of other fancy tricks
- media.blackhat.com/bh-eu-10/presentations/Stone/BlackHat-EU-2010-Stone-Next-Generation-Clickjacking-slides.pdf

Clickjacking

You will also need to defeat some basic framebusting

See link for ideas

http://seclab.stanford.edu/websec/framebusting/framebust.pdf

Tips

You have the source code! This is extremely useful for evading filters.

Firebug is your friend.

Defenses

Some Confusion

Google Chrome has a feature that prevents JS originating from the URL bar from running

Firefox has a feature called urlbar.filter that does something unrelated

I made a post explaining how to turn off this feature but this is not necessary

Attack A will work fine in Firefox without disabling this feature

Hints

Part 1 of the Web Security Project has you making GET requests that include leaked information. How do you verify that it worked?

Use XmlHttpRequests and inspect responses

- cookies.php and credentials.php reflect GET parameters

Download a Firefox plugin to monitor outbound requests

- TamperData and HttpFox are good choices, but there are others

Reflected XSS - How it works

```
<body> <?php
   $userid = $_GET['id'];
   echo("Your id is " . $userid);
?> </body>
```

www.vulnerable-site.com/?id=<script> ... </script>

```
<body>
   Your id is <script>...</script>
</body>
```

Defenses - XSS

Sanitize all user inputs using strong filters

- All meaningful HTML must be removed

PHP: htmlspecialchars(str)

```
&  =>  &amp;      "  =>  &quot;
<  =>  &lt;       >  =>  &gt;
```

```
<a href='test'>Test</a> =>
&lt;a href=&#039;test&#039;&gt;Test&lt;/a&gt;
```

Defenses - XSS

This is not trivial!

Forgetting a single sanitizer leaves the application vulnerable

Need to encode Javascript if user input is used in a JS context

Nested contexts can require nested encodings

```
<div onclick="setTimeout('do_stuff(\'user_string\')', 1000)">
```

Defenses - XSS

This is not trivial!

Forgetting a single sanitizer leaves the application vulnerable

Need to encode Javascript if user input is used in a JS context

Nested contexts can require nested encodings

```
<div onclick="setTimeout('do_stuff(\'user_string\')', 1000)">
```

Defenses - XSS

Properly encoded user input can still cause problems (attack 5)

- Think about quotes and event listeners

User input in eval() is probably a vulnerability

- (Really you shouldn't use eval at all)

Some applications want to let users include content markup

Defenses - CSRF

Need to verify that the request actually came from within the site

Secret token unique to each user

- For this project a hash of the session token is enough
- includes/auth.php has some helpful code

Defenses - Clickjacking

Framebusting code

Prevents a page from being loaded in an iframe

```
if(top.location != self.location)
   parent.location = self.location;
```

Not good enough!

http://seclab.stanford.edu/websec/framebusting/framebust.pdf

Better Defenses...?

Defenses - Frameworks

Zoobar is written using natural php

Using a web framework can make us "fail secure"

- Automatically sanitize dynamic content
- Automatically produce and check CSRF tokens
- Handle sessions safely
- Force devs to use protections against SQL Injection

But they can also introduce their own problems

- RoR mass assignment is a good example

Defenses - Browser Protections

Try attack 1 in Chrome - it doesn't work!

- Chrome removes javascript that appears in the URL bar

Devs still need to sanitize input

Actually introduces other security vulnerabilities

- Attacker can force the page to not run specified JS

Defenses - CSP

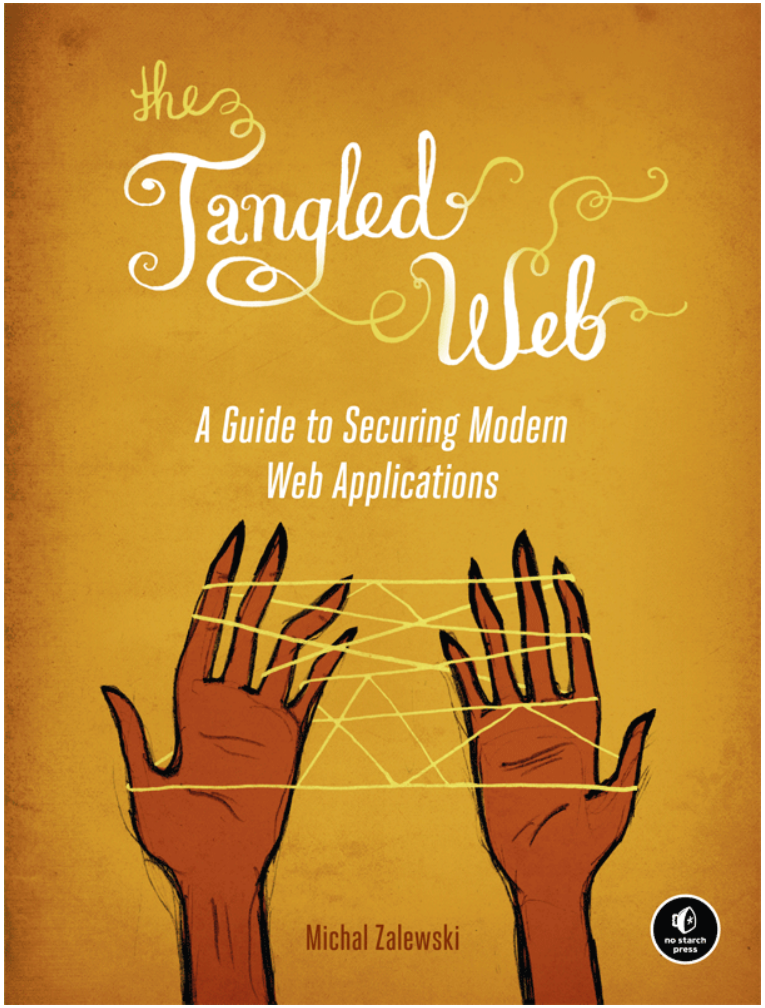
Combined efforts of browsers and developers

Developers specify a whitelist of allowable scripts and browsers prevent all other scripts from executing

...But now we can no longer have inline scripts

...Also not clear if this actually defends against XSS in all cases

Want to learn more?



Questions?