

**digg™**

Help, My Website has been  
Hacked! Now What?

Elliott White III - *Eli*

**digg™**

Oh noes!  
I can has hakkerz!

Elliott White III - *Eli*

# *What's included?*

- - Introduction
- - A definition of hacked
- - Three stages of security
- - Making sure you don't get hacked
- - Discovering that you've been hacked
- - Immediate response
- - Finding the security hole
- - Q&A

# *Introduction*

Why this talk?

Why should I give it?

# *Definition of hacked*

- Included in this talk:
  - - SQL Injection
  - - XSS
  - - CSRF
  - - Session Hijacking

Not directly included:

- - Network attacks
- - Apache/PHP/MySQL/etc bugs
- - Physical attacks

# *Three Stages of Security*

Prevention

Preparation

Panic!

# SQL Injection

A user having the ability to send data that is directly interpreted by your SQL engine.

## The Hole:

```
mysql_query("select * from users where name = '{$_GET['name']}'");
```

## The Attack:

```
$_GET['name'] = "' or 1=1; //";
```

## The Prevention:

```
$name = mysql_real_escape_string($_GET['name']);  
mysql_query("select * from users where name = '{$name}'");
```

# XSS

A user having the ability to send data that is directly interpreted as HTML (and therefore as Javascript)

## The Hole:

```
echo $_GET['name'];
```

## The Attack:

```
$_GET['name'] = "<script>alert('XSS');</script>";
```

## The Prevention:

```
$name = htmlspecialchars($_GET['name'], ENT_QUOTES, 'UTF-8');  
echo $name;
```

# CSRF

A user having the ability to forge or force a request on behalf of another user.

## The Hole:

```
if (count($_GET)) { /* Save data */ }
```

## The Attack:

```

```

## The Prevention:

```
if (count($_POST) && ($_SESSION['token'] == $_POST['token']))  
    { /* Save data */ }
```

# Session Hijacking

A user being able to use the session key of another user and thereby assume the original user's identity

## The Hole:

```
session_start();
```

## The Attack:

Setting your browser's session cookie to a known value (discovered through chance, CSRF, session fixation, etc)

## The Prevention:

```
session_regenerate(); // Upon any change of permission level  
if ($_SESSION['secret'] !== regenerate_secret()) { /* Sorry! */ }
```

# *Discovery Methods*

In descending order of usefulness to you:

- - Whitehat
- - Hacker announces it & exploits it
- - Hacker just announces it
- - User complaints
- - Physical Demarcation
- - Logs/Stats monitoring

# *Immediate Response*

Three Options: Depends completely on the situation

Let it Live

Break  
Functionality

Shutdown  
Website

# *Finding the Security Hole*

So you know you have a hole?

You need to look for the specific cause of one of the aforementioned problems.

Use all clues that you can find based upon the visible traces left behind by the hacker.

# *Hacker posted a webpage*

You have the exploit supposedly sitting on a known page

- - View Source
- - View Generated Source
- - Scan all Javascript and .js files
- - Look for iframes

# *Web Logs are your Friend*

You can browse your access log from your server to find evidence of the hack.

- - Limit your search to a known timeframe
- - Look for odd or out of sequence events
- - Scan for GET parameters that are improper
- - Scan other fields, such as Refer and UserAgent

# *Action Logs, also a Friend*

Hopefully you've been keeping logs that track actions taken by users with as much detail as possible.

- - Track who, when, IP, refer, and more
- - Gives you an absolute source to watch a tricky action

# *Failed SQL Logs are Required!*

You absolutely must be logging any SQL query that fails.

- - Most everything ends up in the database
- - They will not succeed on the first attempt
- - Malformed SQL commands are the best culprits
- - Scan for common XSS terms (script, onclick, onfocus)

# *Scan your DB for bad data*

You can also look in the database for the bad data:

- Query in user-supplied fields for common terms:  
(script, onclick, onfocus, javascript:, onload, etc)

# *Obscure Stuff*

You covered the basics, now what? Familiarize yourself with all the obscure methods of breaking your system:

- Obscure user supplied data:  
Refer, UserAgent, Cookies, etc
- Encoding issues:  
htmldecode() after escaping, encoded Javascript

# *Fix it!*

Once you've found evidence, tracking it back to the offending line(s) of code should not be difficult.

# *Digg is Hiring!*

Digg.com is hiring experienced PHP programmers!

<http://digg.com/jobs>  
[jobs@digg.com](mailto:jobs@digg.com)

# Any Questions?

- For this presentation and more:

<http://eliw.com/>

<http://digg.com/users/EliW>

The Digg logo, consisting of the word "digg" in a white, lowercase, sans-serif font, centered within a blue rectangular background.