Web Security Basics

Eli White

Vice President — One for All Events

@EliW

Web Security Basics

Eli White

Vice President — One for All Events

@EliW

TITANIUM SPONSORS

epiq
PAIGE TECHNOLOGIES
INTELLIGENT PAIRING. PERPETUAL SUCCESS.
ADAPTIVE™ SOLUTIONS GROUP
okta

Platinum Sponsors

VinSolutions
Make every connection count.
jack henry & ASSOCIATES INC.®
VALOREM
NAIC
National Association of Insurance Commissioners
NIPR
NATIONAL INSURANCE PRODUCER REGISTRY
TEKsystems
Our people make IT possible.
KEYHOLE SOFTWARE
smartsheet
Work Better™
Couchbase
EquipmentShare
KOCH™
PLURALSIGHT
Google Cloud
VML
mylo™
POWERED BY LOCKTON

Gold Sponsors

Advantage Tech IT Staffing & Recruiting Services
PeopleAdmin
CENTRIQ TRAINING
Cerner
KU EDWARDS CAMPUS The University of Kansas
twilio
{Track:js} JAVASCRIPT ERROR MONITORING
OBJECT PARTNERS
smg service management group®
stackify
Balance Innovations
Allied Global SERVICES
GARMIN
dsi
LRS CONSULTING SERVICES
Aviture ingenuity. engineered.
TouchNet® + Heartland
Mediware®
An affiliate of C&A Industries, Inc. AUREUS GROUP Finance | Systems | Executive
Children's Mercy KANSAS CITY
Commerce Bank
SECURITY PS STRATEGIC INFORMATION SECURITY
ProKarma

About Security

Do we really need to worry about this?

One for All Events

About Security

Security? Bah!

Whether big or small. Someone will try to hack you!

It only takes one person!

The Open Web Application Security Project

http://owasp.org/

The best online resource for learning about various attack vectors and solutions to them.

Use good judgement though, often wiki-user edited 'solutions'.

Stupid Programmer

Let’s clear the air on these …

One for All Events

Unchecked Permissions

CLASSIFIED INFORMATION NONDISCLOSURE AGREEMENT

AN AGREEMENT BETWEEN AND THE UNITED STATES

(Name of Individual — Printed or typed)

1. Intending to be legally bound, I hereby accept the obligations contained in this Agreement in consideration of my being granted access to classified information. As used in this Agreement, classified information is marked or unmarked classified information, including oral communications, that is classified under the standards of Executive Order 12958, or under any other Executive order or statute that prohibits the unauthorized disclosure of information in the interest of national security; and unclassified information that meets the standards for classification and is in the process of a classification determination as provided in Sections 1.1, 1.2, 1.3 and 1.4(e) of Executive Order 12958, or under any other Executive order or statute that requires protection for such information in the interest of national security. I understand and accept that by being granted access to classified information, special confidence and trust shall be placed in me by the United States Government.

2. I hereby acknowledge that I have received a security indoctrination concerning the nature and protection of classified information, including the procedures to be followed in ascertaining whether other persons to whom I contemplate disclosing this information have been approved for access to it, and that I understand these procedures.

3. I have been advised that the unauthorized disclosure, unauthorized retention, or negligent handling of classified information by me could cause damage or irreparable injury to the United States or could be used to advantage by a foreign nation. I hereby agree that I will never divulge classified information to anyone unless: (a) I have officially verified that the recipient has been properly authorized by the United States Government to receive it; or (b) I have been given prior written notice of authorization from the United States Government Department or Agency (hereinafter Department or Agency) responsible for the classification of information or last granting me a security clearance that such disclosure is permitted. I understand that if I am uncertain about the classification status of information, I am required to confirm from an authorized official that the information is unclassified before I may disclose it, except to a person as provided in (a) or (b), above. I further understand that I am obligated to comply with laws and regulations that prohibit the unauthorized disclosure of classified information.

Unchecked Permissions

Ability to URL-hack to access unauthorized data.

One for All Events

Information leaks

One for All Events

FIEO

FIEO

#1 Rule of Web Security!

Filter **I**nput, **E**scape **O**utput

- Filter the data that comes in, so that it's as expected
- Escape the data going out, so that it's safe to use

Filtering Input

Filtering is not directly a security measure, **but**:

- Simplifies security later
- Provides security in depth
- Makes for cleaner data

More layers of security mean less chance of exploit

Sanitize vs Validate

Validate:

- Check that the data is what was expected (an email address is an email address), and refuse if not.

Sanitize:

- Attempt to convert the data into an expected value (convert any strings to integers)

S vs V: Pros & Cons

Validation

- Drawback is refusing data that could be figured out.
  - Declining '301-555-1234' as a phone number because of dashes
  - Refusing '42 towels', when asking "How many?", because non-int

Sanitization

- Drawback is accepting incorrect data
  - Converting '4.5' to 4, when converting to integer
  - Converting 'yes' to 0, when converting to integer (in some languages)

There are appropriate times for each option

Escaping Output

Escaping is actual protection

Making the output **safe** to be used

Must be done appropriate to context

Every type of output needs escaped **differently**

SQL Injection

SQL Injection

One for All Events

SQL Injection

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

The Security Hole:

```
$pdo->query("SELECT * FROM users
             WHERE name = '{$_POST['name']}' AND pass = '{$_POST['pass']}'");
```

The Attack:

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

SQL Injection

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

The Solution:

```
$query = $pdo->prepare("SELECT * FROM users WHERE name = ? AND pass = ?");
$query->execute(array($_POST['name'], $_POST['pass']));
```

or:

```
$name = $pdo->quote($_POST['name']);
$pass = $pdo->quote($_POST['pass']);
$pdo->query("SELECT * FROM users WHERE name = {$name} AND pass = {$pass}");
```

Other Injection

Command Injection:
The user being able to inject code into a command line.

Unchecked File Uploads:
The user being allowed to upload an executable file.

Code Injection:
User being able to directly inject code. (DON'T USE EVAL!)

XSS (Cross Site Scripting)

XSS (Cross Site Scripting)

A user sending data that is executed as script

Many ways this attack can come in, but in all cases:
Everything from a user is suspect (forms, user-agent, headers, etc)
When fixing, escape to the situation (HTML, JS, XML, etc)
FIEO (Filter Input, Escape Output)

Don't forget about rewritten URL strings!

XSS - Reflected XSS

Reflected XSS

Directly echoing back content from the user

The Security Hole:

```
<p>Thank you for your submission: <?= $_POST['first_name'] ?></p>
```

The Attack:

First Name: <script>alert('XSS')</script> Submit

One for All Events

XSS - Reflected XSS

Reflected XSS
Directly echoing back content from the user

The Solution (HTML):

```
$name = htmlentities($_POST['first_name'], ENT_QUOTES, 'UTF-8', FALSE);
```

The Solution (JS):

```
$name = str_replace(array("\r\n","\r","\n"),
                    array("\n","\n","\\\n"),addslashes($_POST['first_name']));
```

The Solution (XML):

```
$name = iconv('UTF-8', 'UTF-8//IGNORE',
              preg_replace("#[\\x00-\\x1f]#msi", ' ',
              str_replace('&', '&amp;', $_POST['first_name'])));
```

Wait, why is this a problem?

The user can only hack themselves, right?

1) Users can be directed to your website via links.

2) Also, users can be talked into anything…

Wait! Built-in Protection?

XSS - Stored XSS

Stored XSS

You store the data, then later display it

The Security Hole:

```
<?php
$query = $pdo->prepare("UPDATE users SET first = ? WHERE id = 42");
$query->execute(array($_POST['first_name']));
?>

[...]

<?php
$result = $pdo->query("SELECT * FROM users WHERE id = 42");
$user = $result->fetchObject();
?>
<p>Welcome to <?= $user->first ?>'s Profile</p>
```


One for All Events

XSS - Stored XSS

Stored XSS
You store the data, then later display it

The Solution (HTML):

```
$name = htmlentities($_POST['first_name'], ENT_QUOTES, 'UTF-8', FALSE);
```

The Solution (JS):

```
$name = str_replace(array("\r\n","\r","\n"),
                    array("\n","\n","\\\n"),addslashes($_POST['first_name']));
```

The Solution (XML):

```
$name = iconv('UTF-8', 'UTF-8//IGNORE',
              preg_replace("#[\\x00-\\x1f]#msi", ' ',
              str_replace('&', '&amp;', $_POST['first_name'])));
```

The Same!

XSS - DOM XSS

DOM XSS

What happens in JavaScript, stays in JavaScript

The Security Hole:

```
<script>
$('#verify').submit(function() {
    var first = $(this).find("input[name=first]").val();
    $(body).append("<p>Thanks for the submission: " + first + "</p>");
    return false;
});
</script>
```


One for All Events

XSS - DOM XSS

DOM XSS

What happens in JavaScript, stays in JavaScript

The Solution (Simple):

```
<script>
function escapeHTML(str) {
     str = str + ""; var out = "";
     for (var i=0; i<str.length; i++) {
         if (str[i] === '<') { out += '&lt;'; }
         else if (str[i] === '>') { out += '&gt;'; }
         else if (str[i] === "'") { out += '&#39;'; }
         else if (str[i] === '"') { out += '&quot;'; }
         else { out += str[i]; }
     }
     return out;
}
</script>
```

But you have to deal with attr vs HTML vs CSS etc

So use this: https://github.com/chrisisbeef/jquery-encoder/

CSRF (*Cross Site Request Forgery*)

One for All Events

CSRF (Cross Site Request Forgery)

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

Simplistically via IMG tag or POST forms

Complicated via JavaScript

One for All Events

CSRF (Cross Site Request Forgery)

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

The Attack:

```
<img width="1" height="1"
     src="http://quackr.example.com/quackit?msg=CSRF+Attacks+Rock!" />
```

or

```
<script>
$.post({
    url: 'http://quackr.example.com/quackit',
    data: { msg: 'CSRF Attacks Rock!'}
});
</script>
```

CSRF (Cross Site Request Forgery)

Protect via CSRF token

The Solution (on form):

```php
<?php
function generateToken() {
    $token = empty($_SESSION['token']) ? false : $_SESSION['token'];
    $expires = empty($_SESSION['tExpires']) ? false : $_SESSION['tExpires'];
    if (!$token || ($expires < time())) {
        $token = md5(uniqid(mt_rand(), true));
        $_SESSION['token'] = $token;
    }
    $_SESSION['tokenExpires'] = time() + 14400;
    return $token;
}
?>
<form method="POST" action="">
    <input name="msg" value="" />
    <input type="hidden" name="token" value="<?= generateToken() ?>" />
    <input type="submit" />
</form>
```


One for All Events

CSRF (Cross Site Request Forgery)

Protect via CSRF token

The Solution (on submission):

```php
<?php
$token = empty($_SESSION['token']) ? false : $_SESSION['token'];
$expires = empty($_SESSION['tExpires']) ? false : $_SESSION['tExpires'];
$check = empty($_POST['token']) ? false : $_POST['token'];

if ($token && ($token == $check) && ($expires > time())) {
    // SUCCESS - Process the form
} else {
    // FAILURE - Block this:
    header('HTTP/1.0 403 Forbidden');
    die;
}
?>
```


One for All Events

Clickjacking

One for All Events

Clickjacking

One of the 'newer' threats

Lots of publicity when Twitter was hit

February 2009

Tricks user into physically making a click on the remote website without realizing it, getting around any CSRF protection.

Twitter Attack

Clickjacking

One for All Events

Clickjacking - Solution 1

Use specific header, to disallow site framing:

The Solution:

```
header('X-Frame-Options: DENY');
```

or

```
header('X-Frame-Options: SAMEORIGIN');
```

Doesn't work in all browsers!

Became IETF standard RFC 7034 in October 2013

Clickjacking - Solution 2

Ensure you aren't displayed in iFrame

The Solution:

```
<html>
    <head>
        <style> body { display : none;} </style>
    </head>
    <body>
        <script>
            if (self == top) {
              var theBody = document.getElementsByTagName('body')[0];
              theBody.style.display = "block";
            } else {
              top.location = self.location;
            }
        </script>
    </body>
</html>
```


One for All Events

Password Protection

Best Password Practices

Rules for Passwords:

- Don't restrict people from using letters, numbers, special characters or spaces
- OK to have a minimum length but not max
- Requiring mixed symbols can help, but makes hard to remember

One for All Events

One for All Events

Password Hashing

Do not store plain text passwords

Always 1-way hash

Do not just use MD5!
Highly vulnerable to rainbow tables

Don't even use SHA1
The longer your hashing takes to run,
the longer it takes for someone to crack it!

Secondary Measures

Typically used to thwart phishing attempts

- Showing a known photo on login
- Asking for date of birth
- Asking for first place of residence
- etc…

All have mixed effectiveness

2-factor Authentication

Having something beyond username/password to ensure authentication is valid. Requires extra information that the user must have on them.

Originally involved keyfobs, or other physical devices that had to be plugged into the computer.

Today most commonly is done as sending an SMS to the user (Facebook), or via a token generator such as Google Authenticator for smartphones.

Implementing 2FA

SMS Method:

- Use Amazon SNS to send SMS upon login
- Use Twilio API to send SMS (or phone call)

Token Based 2FA:

- Use Google Authenticator Libraries:
  - http://code.google.com/p/ga4php/
  - https://www.idontplaydarts.com/2011/07/google-totp-two-factor-authentication-for-php/

Brute Force

Brute Force Attacks (Password)

Really only two primary defenses:

CAPTCHA

IP rate limiting

One for All Events

Brute Force Attacks (CAPTCHA)

reCAPTCHA is free and easy to use

On the Form:

```
<?php require_once('recaptchalib.php'); ?>
<form method="POST" action="">
    <label>Username: <input name="user" /></label><br />
    <label>Password: <input name="pass" type="password"/></label><br />
    <?= recaptcha_get_html("YOUR-PUBLIC-KEY"); ?>
    <input type="submit" />
</form>
```

Brute Force Attacks (CAPTCHA)

On the Server:

```php
<?php
require_once('recaptchalib.php');
$check = recaptcha_check_answer(
    "YOUR-PRIVATE-KEY", $_SERVER["REMOTE_ADDR"],
    $_POST["recaptcha_challenge_field"], $_POST["recaptcha_response_field"]);

if (!$check->is_valid) {
    die("INVALID CAPTCHA");
} else {
    // Yay, it's a human!
}
?>
```

https://developers.google.com/recaptcha/docs/php

Brute Force Attacks (Rate Limit)

Only allow so many fails per IP

The Solution:

```
$blocked = false;
$cachekey = 'attempts.'.$_SERVER['REMOTE_ADDR'];
$now = new DateTime();
$attempts = $memcached->get($cachekey) ?: [];
if (count($attempts) > 4) {
    $oldest = new DateTime($attempts[0]);
    if ($oldest->modify('+5 minute') > $now) {
        $blocked = true; // Block them
    }
}
if (!$blocked && $user->login()) {
    $memcached->delete($cachekey);
} else {
    array_unshift($attempts, $now->format(DateTime::ISO8601));
    $attempts = array_slice($attempts, 0, 5);
    $memcached->set($cachekey, $attempts);
}
```


One for All Events

System Level Security

Filesystem Security

Make sure your web server does not run as 'root'

The user it runs as should only have access to the 'web' directory

Commonly ignored, but offers great security-in-depth

Database Security

Same advice:
Make sure the database user only has permissions that it needs.

Consider:
Make the normal DB user only have read access.
Use separate connections with another user for writing.

Keep Your Stack Patched

No excuses. Keep all your software up to date!

Apache

MySQL

PHP

Linux

Ruby

Python

DDOS & Similar Attacks

Good luck!

Rely on firewall features of your machines & hosting.

Hire a good ops team

Man in the Middle

Icons made by Freepik from www.flaticon.com

Man in the Middle

Icons made by Freepik from www.flaticon.com

Session Protection

One for All Events

Session Hijacking

One user 'becoming' another by taking over their session via impersonation.

Avoid "Session Fixation"
Don't use URL cookies for your sessions.

Always regenerate Session IDs on a change of access level.

Save an anti-hijack token to another cookie & session. Require it to be present & match. Salt on unique data (such as User Agent)

Session Fixation

A user being able to provide a known session ID to another user.

The Attack:

The Solution:

```
session.use_cookies = 1
session.use_only_cookies = 1
```

Don't use URL cookies for your sessions.

Session Fixation (Take 2)

Protect from more complicated fixation attacks, by regenerating sessions on change of access level.

The Solution:

```
session_start();
if ($user->login($_POST['user'], $_POST['pass'])) {
    session_regenerate_id(TRUE);
}
```

and

```
session_start()
$user->logout();
session_regenerate_id(TRUE);
```

Session Anti-Hijack Measures

Finally use anti-hijack measures to ensure user is legit

Not a few lines of code.
Store whatever unique you can about this user/browser combination and verify it hasn't changed between loads.

Note that IP changes or can be shared.
As happens with most other headers too.

Session Anti-Hijack Measures

```
private function _sessionStart() {
    session_start();
    if (!empty($_SESSION)) { // Session not empty, verify:
        $token = $this->_hijackToken();
        $sh = empty($_SESSION['hijack']) ? NULL : $_SESSION['hijack'];
        $ch = empty($_COOKIE['data']) ? NULL : $_COOKIE['data'];
        if (!$sh || !$ch || ($sh != $ch) || ($sh != $token)) { // Hijacked!
            session_write_close();
            session_id(sha1(uniqid(rand(), TRUE)));
            session_start();
            setcookie('data', 0, -172800);
            header("Location: http://www.example.com/");
        }
    } else { // Empty/new session, create tokens
        $_SESSION['started'] = date_format(new DateTime(), DateTime::ISO8601);
        $_SESSION['hijack'] = $this->_hijackToken();
        setcookie('data', $_SESSION['hijack']);
    }
}

private function _hijackToken() {
    $token = empty($_SERVER['HTTP_USER_AGENT']) ? 'N/A' : $_SERVER['HTTP_USER_AGENT'];
    $token .= '| Hijacking is Bad mmmkay? |'; // Salt
    $token .= $_SESSION['started']; // Random unique thing to this session
    return sha1($token);
}
```


One for All Events

Tips for Recovery

Wait, you just got a 2am phone call?

Logging

You can't react, if you don't know what happened!

Log everything you can:

- Failed SQL queries
- Detected hijack attempts
- Code (PHP) errors
- Failed server connections

One for All Events

Plans of Action

Be ready for a quick decision!

Let it Live

Remove Functionality

Shutdown Website

Thank you very much!

But a brief commercial interruption

One for All Events

23% OFF
$325
PHP[WORLD] 2017
PHP[WORLD] 2017 CONFERENCE
November 15-16, 2017 Washington, D.C.
world.phparch.com

Questions?

For this presentation & more:
eliw.com

Twitter: @EliW

One for All Events:
www.oneforall.events

Password Protection

Additional Information

One for All Events

Password Hashing (Manually)

PHP <5.5

Always generate & add a salt, to beat rainbow tables:

```
$str = "This is my secret data";
$hash = hash('sha512', $str);
```

Find a full list of supported algorithms via:

```
var_dump(hash_algos());
```

Use a more secure algorithm, such as sha512:

```
$password = "MyVoiceIsMyPassport";

// Simple salt:
$salt = "PHP FOR LIFE";
$hash = hash('sha512', $salt . $password);

// More fancy & Unique
$salt = hash('sha1', uniqid(rand(), TRUE));
$hash = $salt . hash('sha512', $salt . $password);
```

Password Hashing PHP 5.5

PHP 5.5 has a built in `password_hash` function, that takes care of salting, has a configurable cost, and provides mechanisms for upgrading algorithms in the future:

```
string password_hash ( string $password , integer $algo [, array $options ] )
boolean password_verify ( string $password , string $hash )
```

Sample Usage:

```
$hash = password_hash('MyVoiceIsMyPassport', PASSWORD_DEFAULT);
$hash = password_hash('rootroot', PASSWORD_DEFAULT, ['cost' => 12]);
```

http://php.net/password

Password Hashing PHP 5.5

Also allows for upgrade paths for password security via the password_needs_rehash() function:

```
$options = [ 'cost' => 12 ];
if (password_verify($password, $hash)) {
    // Success - Log them in, but also check for rehash:
    if (password_needs_rehash($hash, PASSWORD_DEFAULT, $options)) {
        // The password was old, rehash it:
        $rehash = password_hash($password, PASSWORD_DEFAULT, $options);
        // Save this password back to the database now
    }
} else {
    // Failure, do not log them in.
}
```