



Web Security and You

By: Eli White

CTO & Founding Partner:
musketees.me

Managing Editor & Conference Chair:
php[architect] - phparch.com

eliw.com - @eliw



About Security

Do we really need to worry about this?

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'.

Password Protection

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 SHA-1
Don't even use SHA-512
The longer your hashing takes to run,
the longer it takes for someone to crack it!

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:

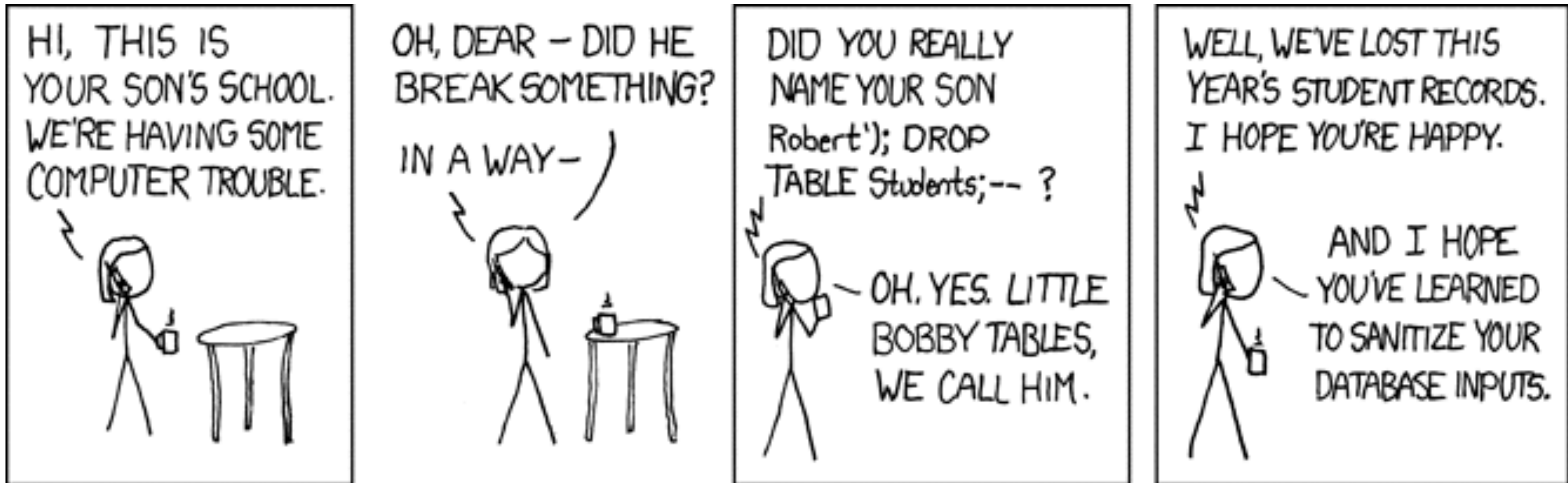
```
$hash = password_hash('MyVoiceIsMyPassport', PASSWORD_DEFAULT);  
$hash = password_hash('rootroot', PASSWORD_DEFAULT, ['cost' => 12]);
```

<http://php.net/password>

Various Attack Vectors

Now moving on to true 'attacks' ...

SQL Injection



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:

```
$_GET['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!)

Session Hijacking

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

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

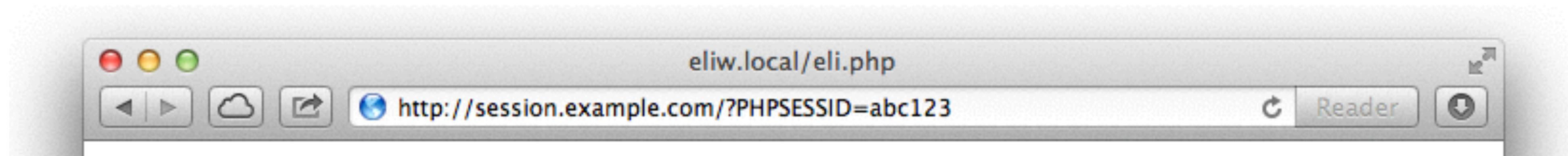
Always regenerate Session IDs on a change of access level.

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  
session.cookie_httponly = 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);
}
```


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:

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...



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>
```

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>
```

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)

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

CSRF (Cross Site Request Forgery)

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

The Attack:

```

```

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
function generateToken() {
    $token = empty($_SESSION['token']) ? false : $_SESSION['token'];
    if (!$token) {
        $token = sha1(uniqid(mt_rand(), true));
        $_SESSION['token'] = $token;
    }
    return $token;
}
?>
<form method="POST" action="">
    <input name="msg" value="" />
    <input type="hidden" name="token" value="<?= generateToken() ?>" />
    <input type="submit" />
</form>
```

CSRF (Cross Site Request Forgery)

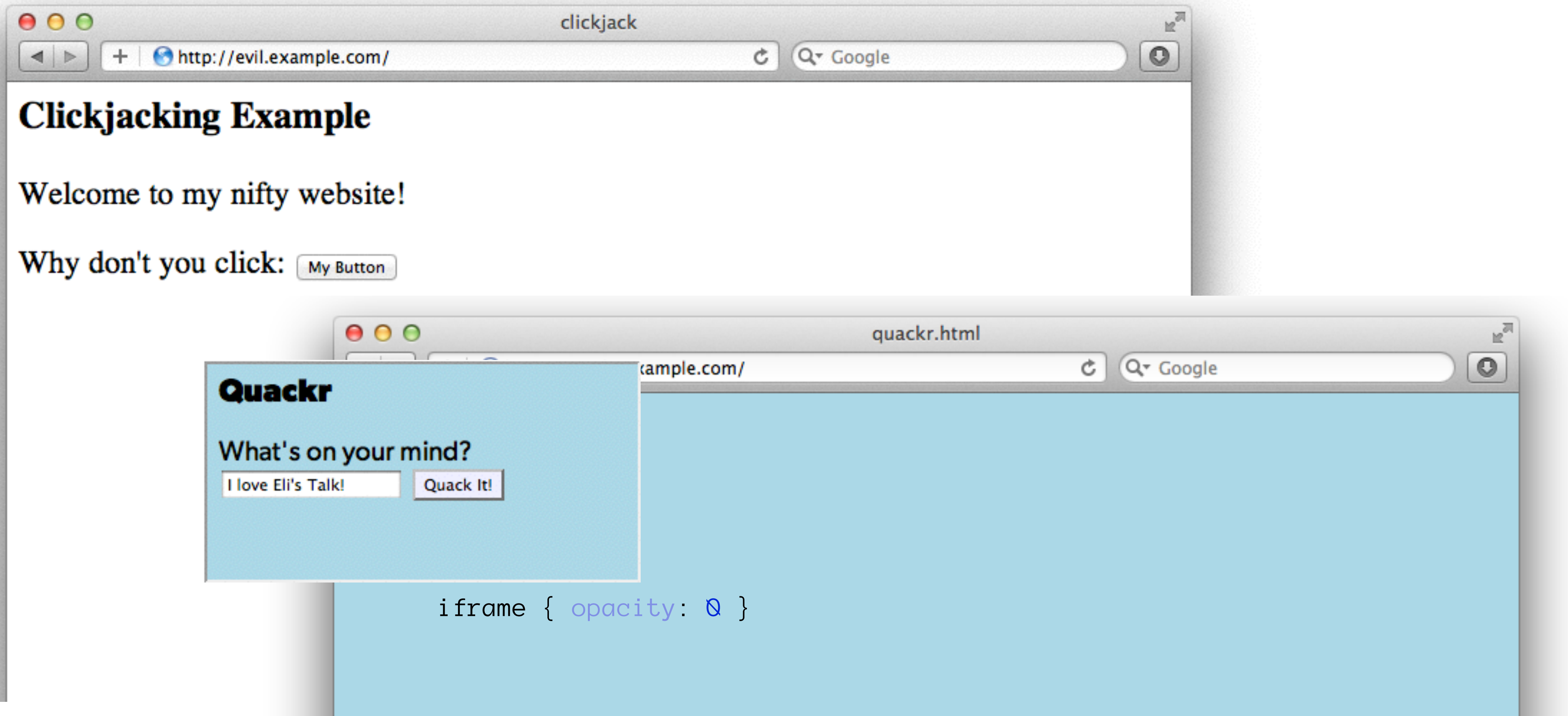
Protect via CSRF token

The Solution (on submission):

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

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

Clickjacking



Clickjacking - Solution I

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

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>
```

Ensure you aren't displayed in iFrame

Server Level Security

Now moving on to true 'attacks' ...

Keep Your Stack Patched

No excuses. Keep all your software up to date!

Linux

Apache

MySQL

PHP

Ruby

Python

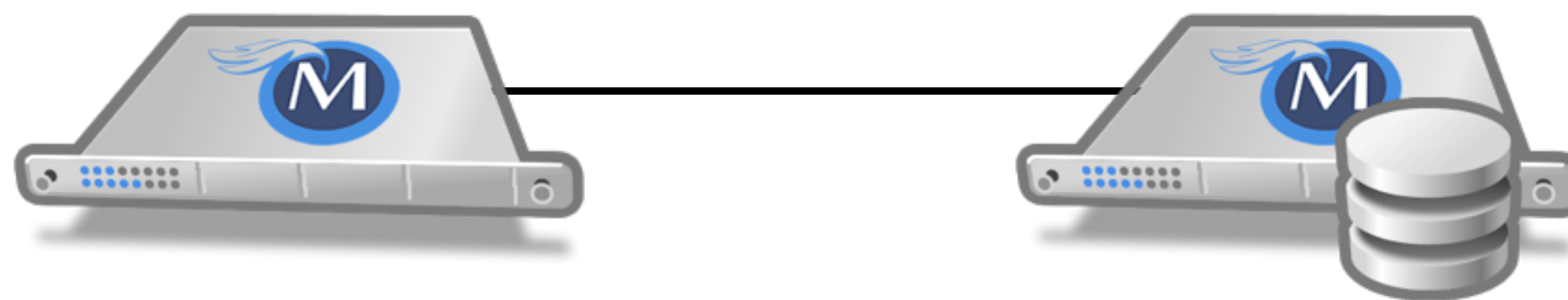
DDOS & Similar Attacks

Good luck!

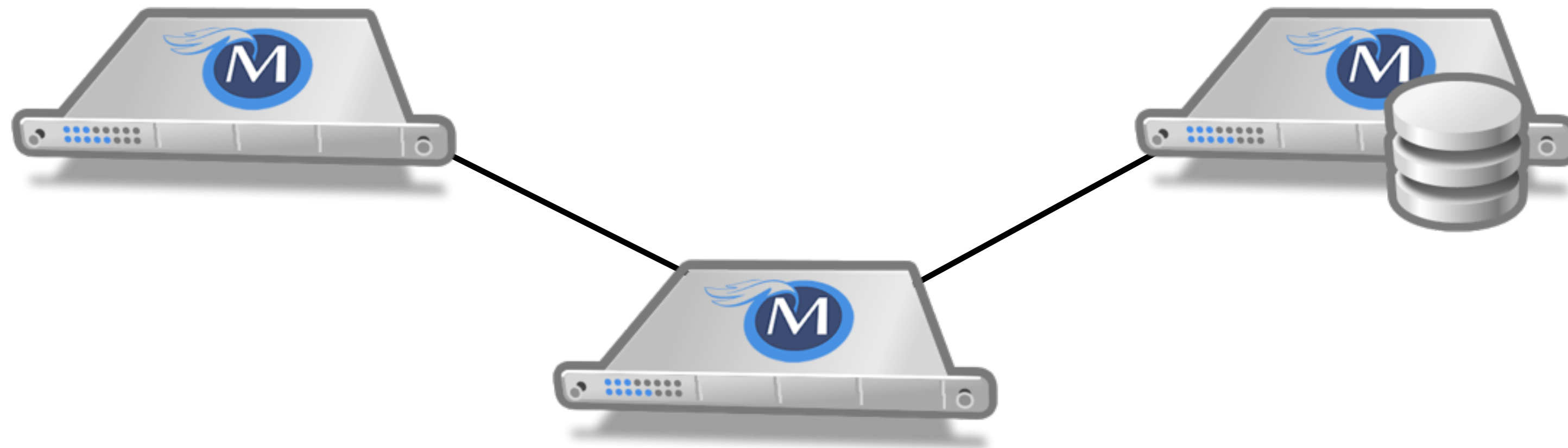
**Rely on firewall features of
your machines & hosting.**

Hire a good ops team

Man in the Middle



Man in the Middle



The Solution: Use SSL

Questions?

For this presentation & more:
<http://eliw.com/>

Twitter: @EliW

php[architect]: <https://www.phparch.com>
musketears: <http://musketears.me>

Rate this talk!
<https://joind.in/16769>



Bonus Subjects!

If Eli had too much coffee this morning...

Brute Force Attacks

Brute Force Attacks (Password)

Really only two primary defenses:



CAPTCHA



IP rate limiting

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
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)

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);
}
```

Only allow so many fails per IP

More Password Info

Password Hash Upgrades — 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.
}
```


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/>

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

Plans of Action

Be ready for a quick decision!

Let it Live

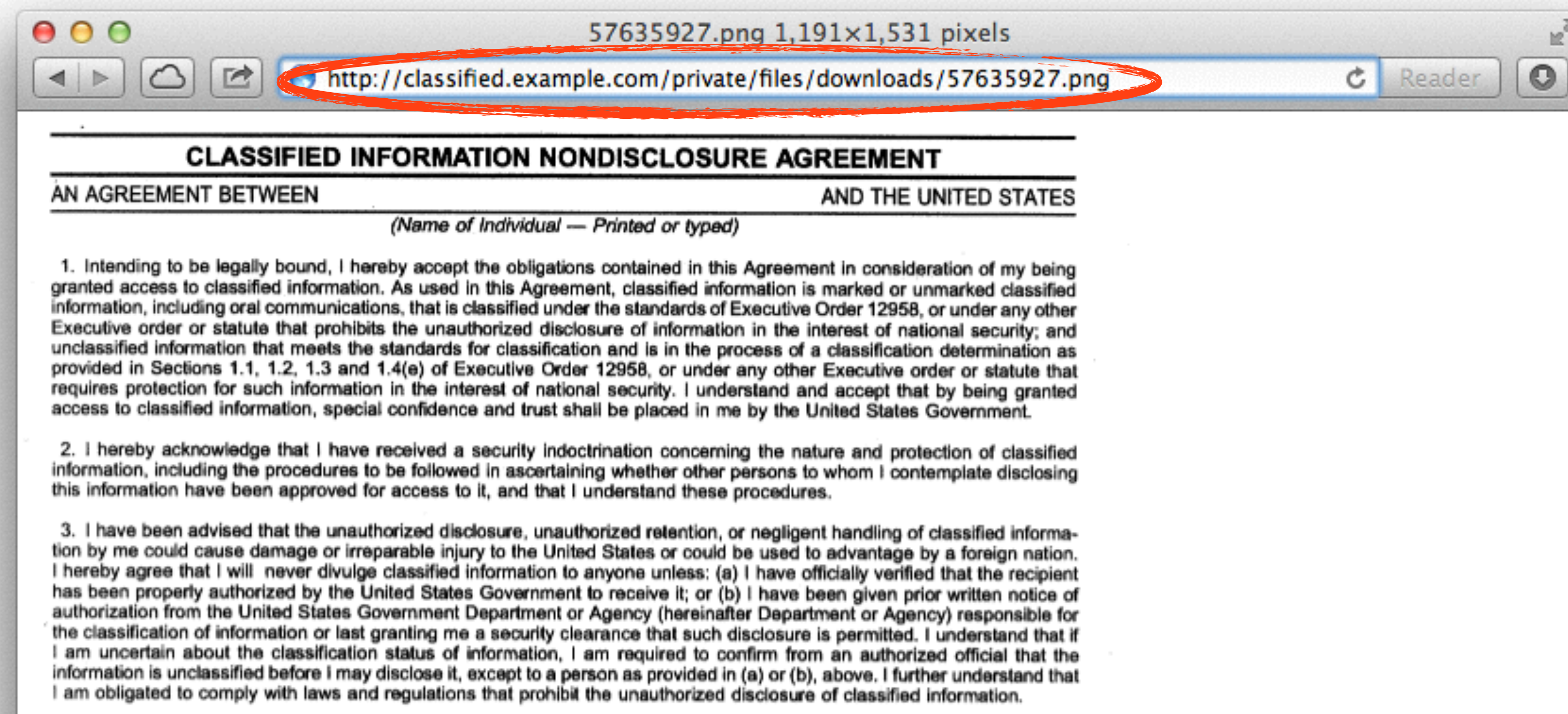
Remove Functionality

Shutdown Website

Stupid Programmer Errors

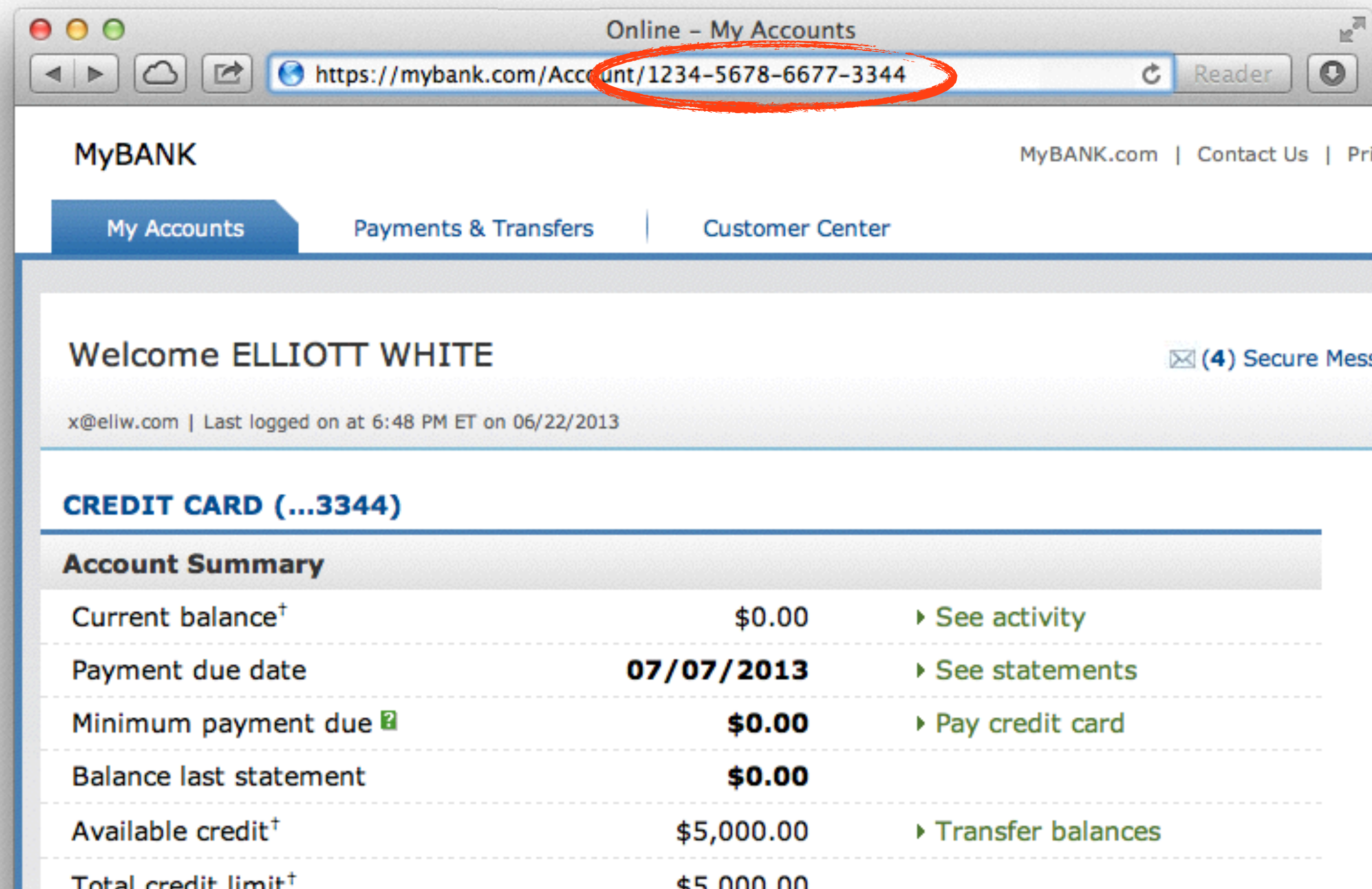
Let's clear the air on these ...

Unchecked Permissions



Unchecked Permissions

Ability to URL-hack to access unauthorized data.



Information leaks

