

# Web Security Basics

Eli White

*Vice President — One for All Events*

@EliW



# Web Security Basics

Eli White

*Vice President — One for All Events*

@EliW



# About Security

---

Do we really need to worry about this?

A photograph of two surveillance cameras mounted on a single pole against a clear blue sky. One camera is positioned higher and angled downwards, while the other is lower and angled slightly to the right. Both cameras have white bodies and dark lenses.

# 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'.

# FIEO



## #1 Rule of Web Security!

### Filter Input, Escape Output

- 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

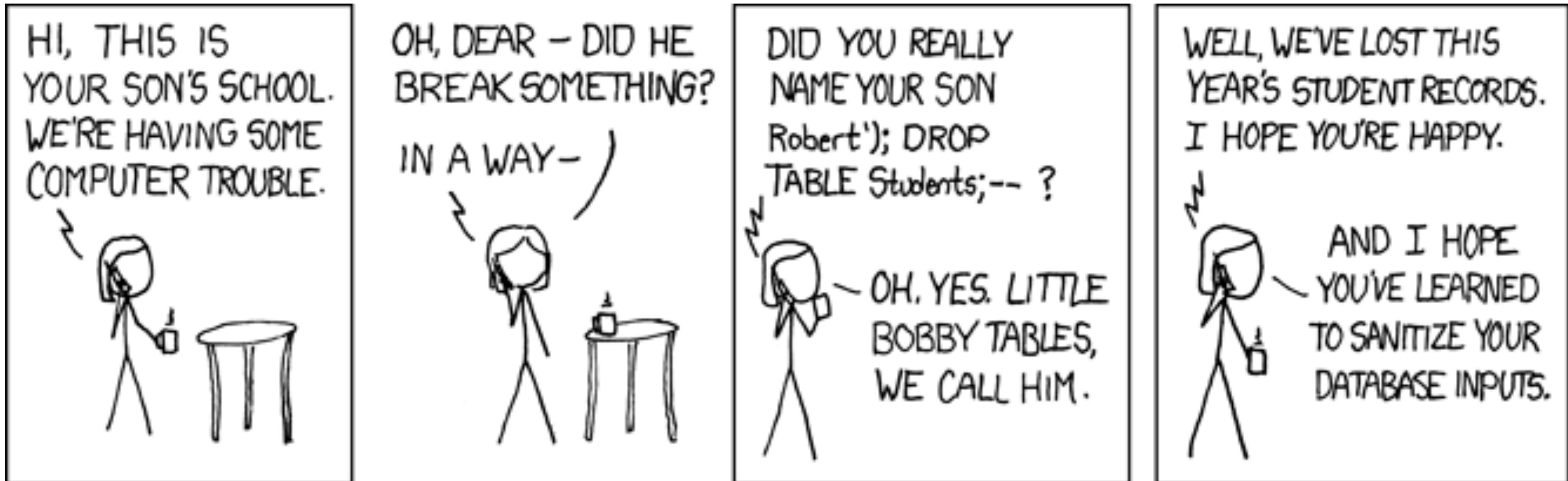
**Must** be done appropriate to context

Every type of output needs escaped **differently**

# SQL Injection



# SQL Injection



# 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:

# 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('&', '&', $_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>
```

# 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("#[\x{00}-\x{1f}#msi", ' ', 
              str_replace('&', '&', $_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 += '''; }
        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/chrisbeef/jquery-encoder/>

Datum	Einheiten	D/02.02.03
Sorte		
FLG		
Stoffverhältnis DIP / Etik.		
V - Sieb		
V - Poperoller		
Arbeitsbreite		
Stoffauflauf		
Auslaufverhältnis		
C1		
Schicht/ Datum		
Sorte		
FLG		
Stoffverhältnis DIP / Etik.		
V - Sieb		
V - Poperoller		
Arbeitsbreite		
Stoffauflauf		
Auslaufverhältnis		
Druck		
PD Innendruck		
Druckwaage / Spülung		
Lippenöffnung		
Vorderwand		
Pumpendrehzahl		
Schüttelbock Freq.		
Schüttelbock Hub		
Duoformer D		
Scimmer / Entwässerung		
1. Zone		
2. Zone		
Obersiebentwässerung		
Druck Leiste		
Einlaufwalze Duoform		
Vakuuminstellung		
1. Vakufoil		
Einlaufwalze / Naßsauger		
1. Formationszone		
2. Zone (Trockeng. Trennsauger)		
1. Formationszone		
2. Zone		
B		
Schicht/ Datum		
Sorte		
FLG		
Stoffverhältnis DIP / Etik.		
V - Sieb		
V - Poperoller		
Arbeitsbreite		
Stoffauflauf		
Auslaufverhältnis		
Druck		
PD Innendruck		
Druckwaage / Spülung		
Lippenöffnung		
Vorderwand		
FU- Stoffauflaufpumpe		
Schüttelbock Freq.		
Schüttelbock Hub		
Duoformer D		
Obersiebentwässerung		
Scimmer / Entwässerung in		
1. Zone		
2. Zone		
Druck Leiste 1 + 2		
Druck Leiste 3		
Druck Leiste 4		
Druck Leiste 5 + 6		
Druck Leiste 7 + 8		
Druck Leiste 9 + 10		
Einlaufwalze Duoformer / Spalt		
Vakuumstellungen		
1. Vakufoil		
2. Vakufoil / Naßsauger		
Doppelvakufol		
Scimmer		
R		
05.01.03		
Newspr.		
g/m²		
%		
m/min.		
m/min		
m		
1018		
10519		
2,5		
3		
415		
120		
945		
280		
25,0		
35		
32		
64		
4		
70		
70		
150		
150		
120		
defekt		
114		
-12		
-27		

# CSRF (Cross Site Request Forgery)

# 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)

The Solution (on form):

```
<?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>
```

**Protect via CSRF token**

# CSRF (Cross Site Request Forgery)

The Solution (on submission):

```
<?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;
}
?>
```

**Protect via CSRF token**

# Clickjacking



# 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

Realtime results for **don't click**

6620 more results since you started searching. [Refresh](#) to see them.

 [iboy](#): **Don't Click:** <http://tinyurl.com/amgzs6> (expand)  
less than 10 seconds ago · [Reply](#) · [View Tweet](#)

 [animealmanac](#): **Don't Click:** <http://tinyurl.com/amgzs6> (expand)  
less than 20 seconds ago · [Reply](#) · [View Tweet](#)

 [carolangrisani](#): **Don't Click:** <http://tinyurl.com/amgzs6> (expand)  
less than 20 seconds ago · [Reply](#) · [View Tweet](#)

 [petebakes](#): **Don't Click:** <http://tinyurl.com/amgzs6> (expand)  
less than 20 seconds ago · [Reply](#) · [View Tweet](#)

 [jjhall](#): **Don't Click:** <http://tinyurl.com/amgzs6> (expand)  
less than 20 seconds ago · [Reply](#) · [View Tweet](#)

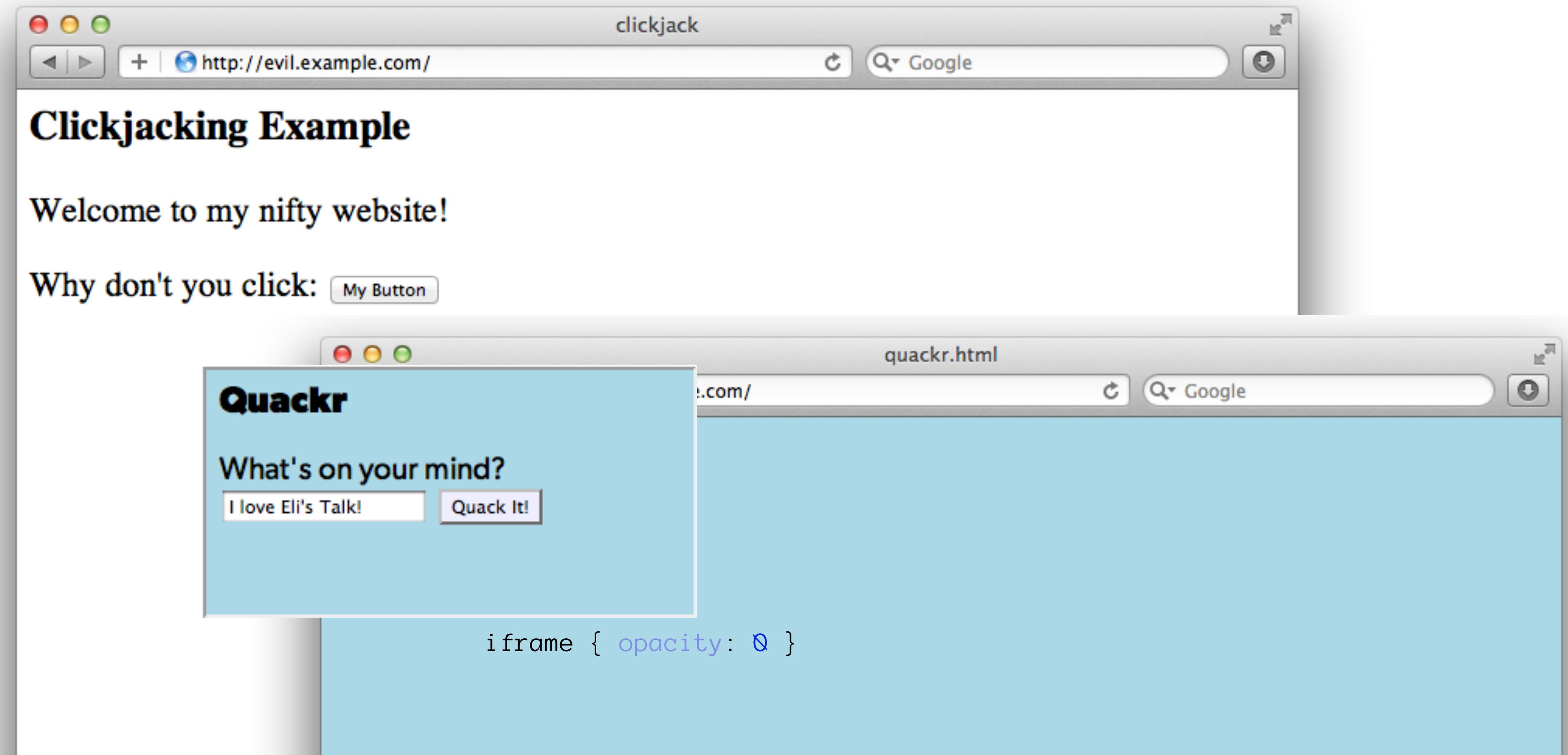
Timeline Favorites Following Followers Lists

Dawkins -> I have spent: 32.7 hours on Twitter! See how much you have:  
58 seconds ago

Dawkins --> I have spent: 32.7 hours on Twitter! See how much you have:  
53 seconds ago

Old Trafford -> I have spent: 32.7 hours on Twitter! See how much you have:  
1 minute ago

# Clickjacking



# 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

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>
```

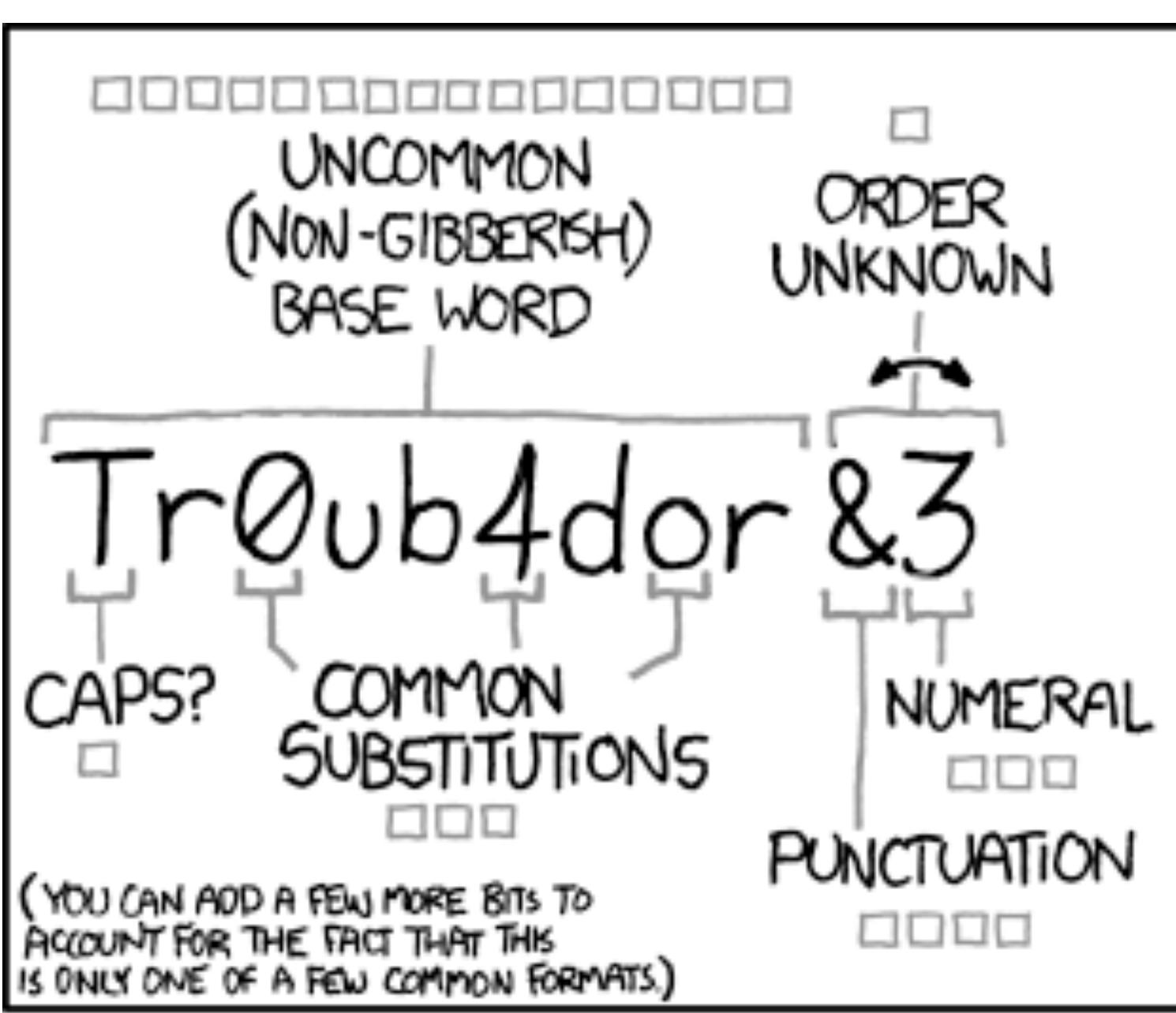
**Ensure you aren't displayed in iFrame**

# 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



~28 BITS OF ENTROPY

A grid of 15 empty square boxes arranged in three rows of five. The first two rows are fully visible, while the third row is partially visible at the bottom.

$2^{28}$  = 3 DAYS AT  
1000 GUESSES/SEC

(PLAUSIBLE ATTACK ON A WEAK REMOTE  
WEB SERVICE. YES, CRACKING A STOLEN  
HASH IS FASTER, BUT IT'S NOT WHAT THE  
AVERAGE USER SHOULD WORRY ABOUT.)

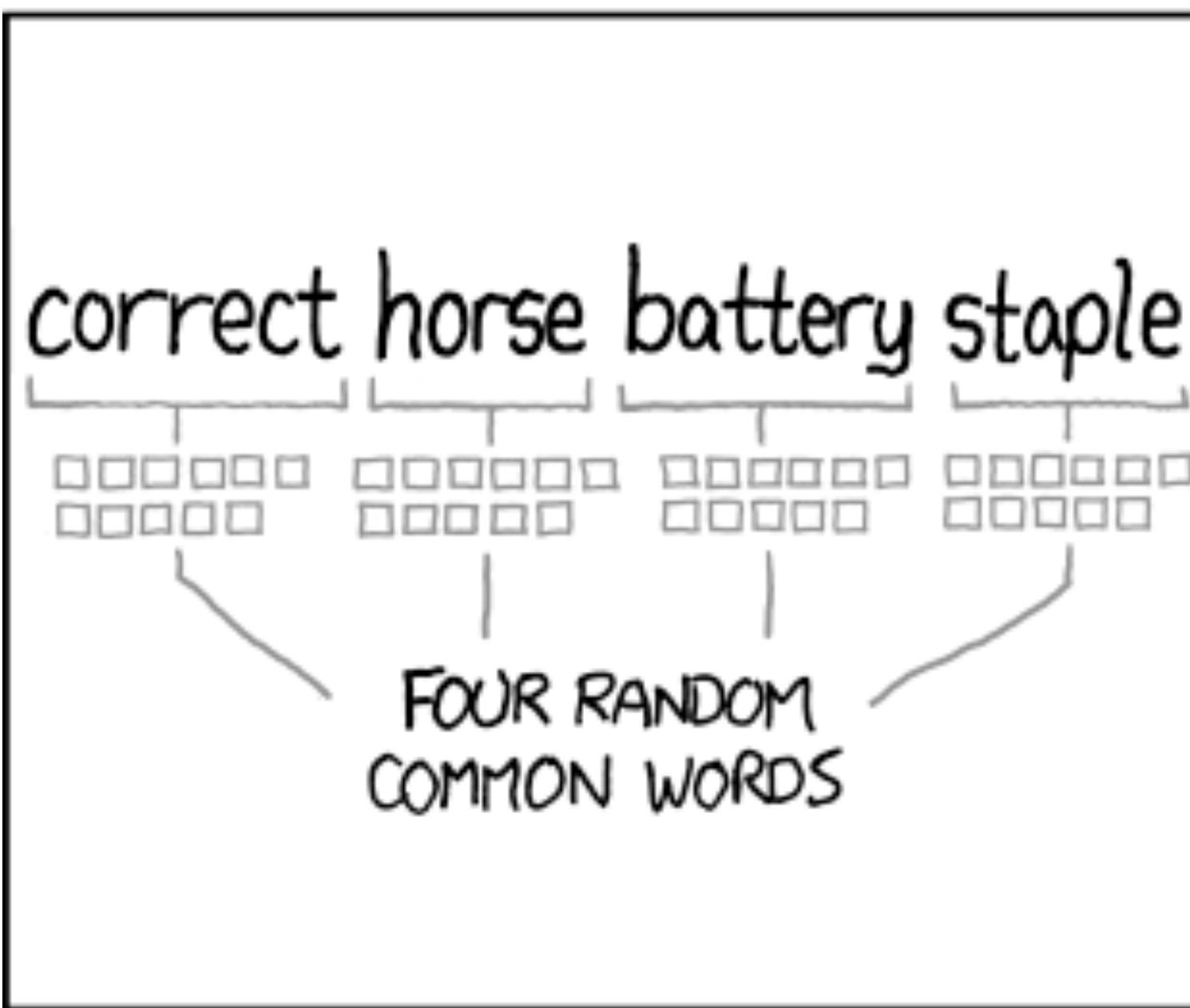
DIFFICULTY TO GUESS:  
**EASY**

WAS IT TROMBONE? NO,  
TROUBADOR. AND ONE OF  
THE Os WAS A ZERO?

AND THERE WAS  
SOME SYMBOL...



# DIFFICULTY TO REMEMBER: HARD

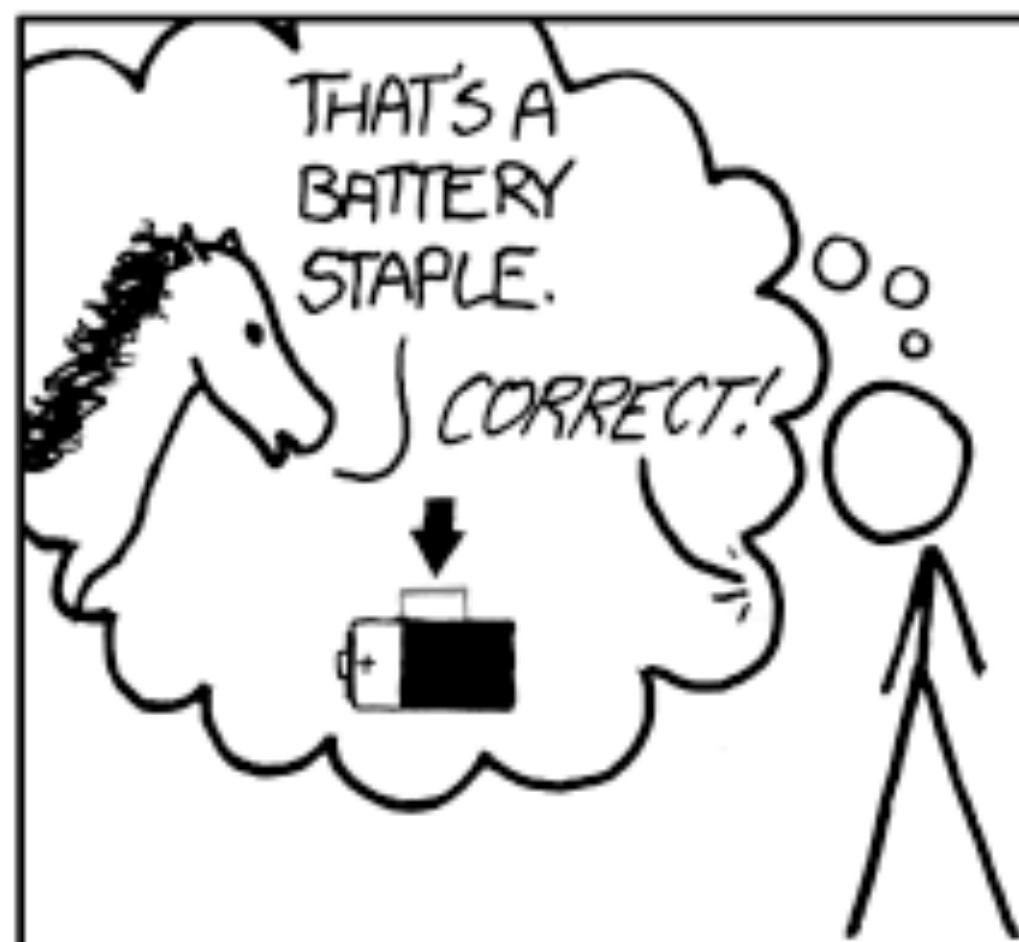


~44 BITS OF ENTROPY

A 4x10 grid of 40 empty square boxes, arranged in four rows and ten columns.

$2^{44} = 550$  YEARS AT  
1000 GUESSES/SEC

## DIFFICULTY TO GUESS: HARD



## DIFFICULTY TO REMEMBER: YOU'VE ALREADY MEMORIZED IT

# 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!



# 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.

# Thank you very much!

---

But a brief commercial interruption

**CODERCRUISE.COM**

**JULY 16-23, 2017**

**PORT OF NEW ORLEANS**

*Montego Bay  
Grand Cayman  
Cozumel*

**CODERCRUISE**

# Questions?

For this presentation & more:  
[eliw.com](http://eliw.com)

Twitter: @EliW

**One for All Events:**  
[www.oneforall.events](http://www.oneforall.events)



*One for All  
Events*

# Brute Force



# 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);
}
```

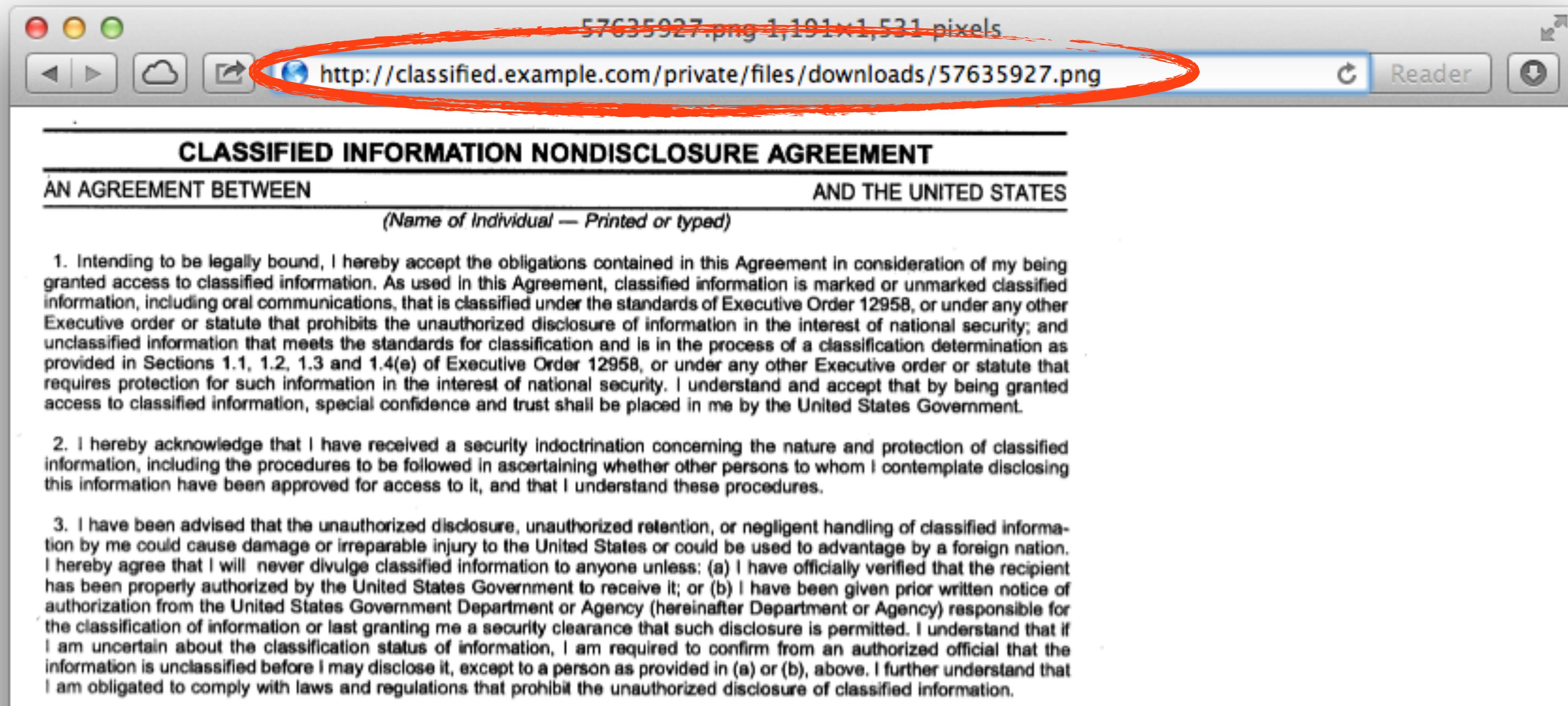
**Only allow so many fails per IP**

# Stupid Programmer

---

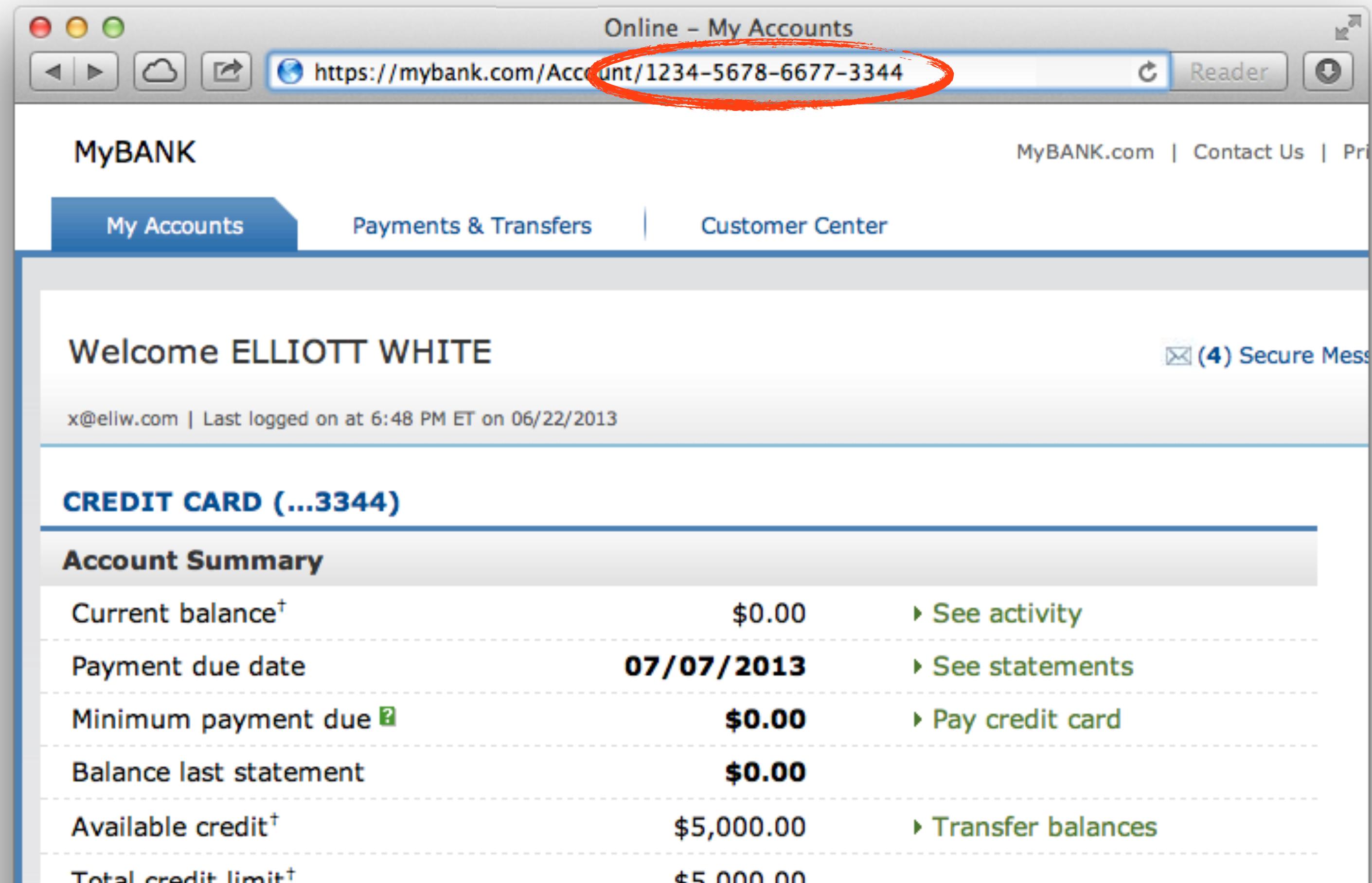
Let's clear the air on these ...

# Unchecked Permissions



# Unchecked Permissions

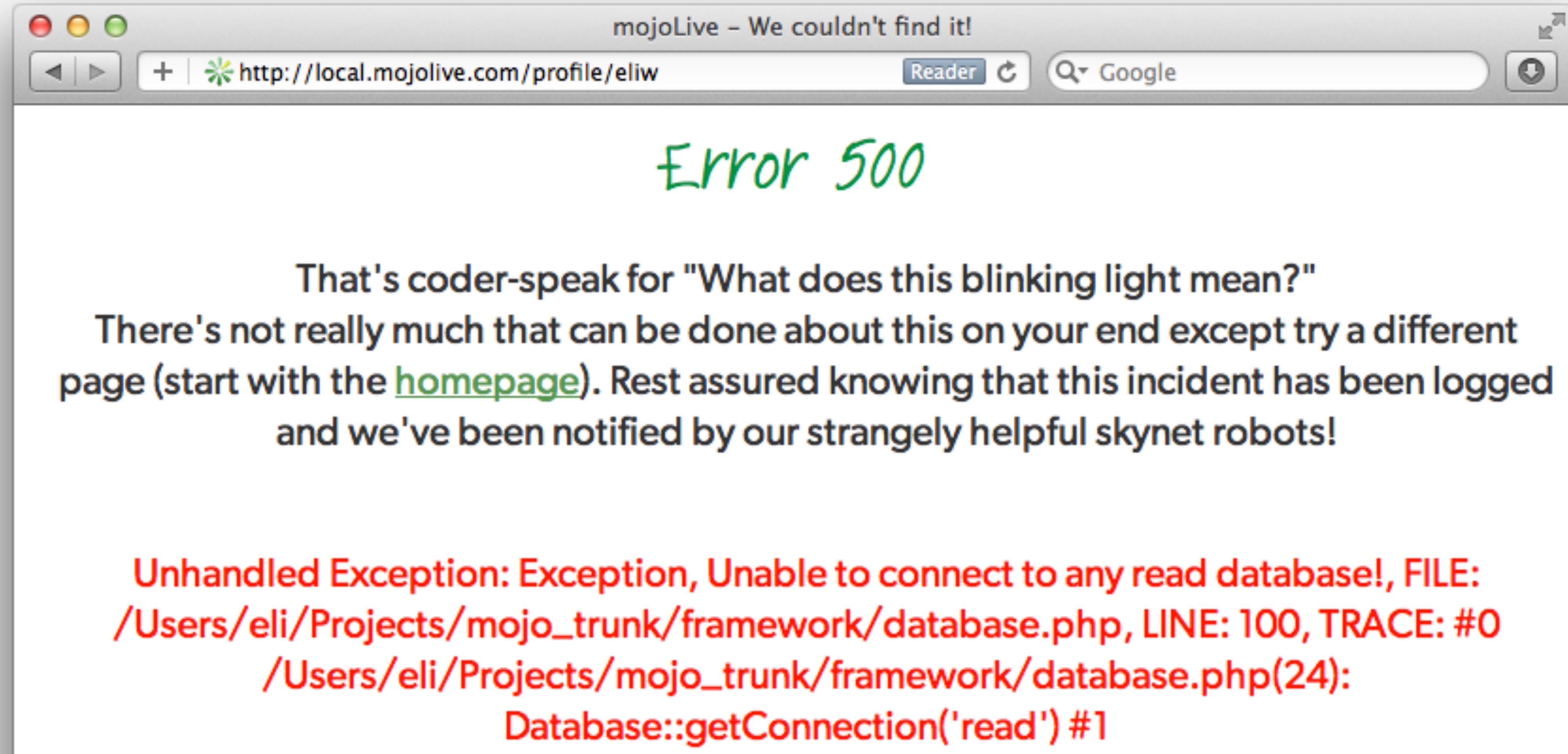
Ability to URL-hack to access unauthorized data.



The screenshot shows a web browser window for "MyBANK". The title bar says "Online - My Accounts" and the address bar shows the URL <https://mybank.com/Account/1234-5678-6677-3344>, with the account number portion circled in red. The page content includes a welcome message for "ELLIOTT WHITE", an email link "x@eliw.com", and a "CREDIT CARD (...3344)" section with an "Account Summary" table. The table lists various account details such as current balance, payment due date, and available credit.

Account Summary		
Current balance <sup>†</sup>	\$0.00	<a href="#">▶ See activity</a>
Payment due date	<b>07/07/2013</b>	<a href="#">▶ See statements</a>
Minimum payment due <small>?</small>	<b>\$0.00</b>	<a href="#">▶ Pay credit card</a>
Balance last statement	<b>\$0.00</b>	
Available credit <sup>†</sup>	<b>\$5,000.00</b>	<a href="#">▶ Transfer balances</a>
Total credit limit <sup>†</sup>	<b>\$5,000.00</b>	

# Information leaks



# Password Protection

---

Additional Information

# Password Hashing (Manually)

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

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

PHP <5.5

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.
}
```

# 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/>

# Session Protection

---

# Session Hijacking

**One user ‘becoming’ another by taking over their session via impersonation.**

Avoid “Session Fixation”  
Don’t use URL cookies for your sessions.

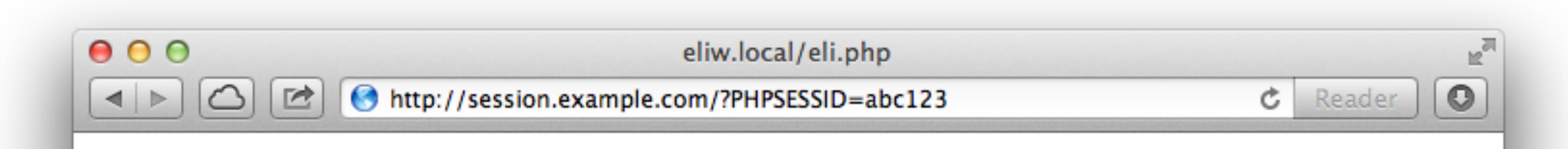
Always regenerate Session IDs on a change of access level.

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);
}
```

# Server Level Security

---

Now moving on to true 'attacks' ...

# Keep Your Stack Patched

**No excuses. Keep all your software up to date!**



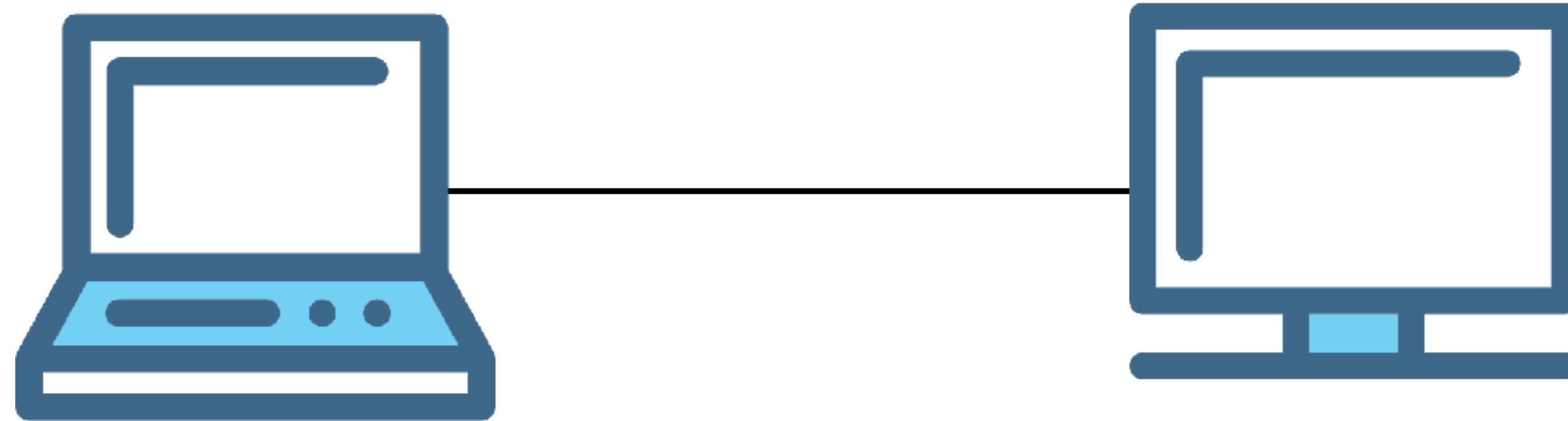
# DDOS & Similar Attacks

**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](#)

# 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

