About the OWASP Top 10

**OWASP Top 10 is an Awareness Document**

- Not a standard...

**First developed in 2003**

- Was probably 3\(^{rd}\) or 4\(^{th}\) OWASP project, after
  - Developers Guide
  - WebGoat
  - Maybe WebScarab ??

**Released**

A1: Injection

A2: Broken Authentication and Session Management

A3: Cross-Site Scripting (XSS)

A4: Insecure Direct Object References

A5: Security Misconfiguration

A6: Sensitive Data Exposure

A7: Missing Function Level Access Control

A8: Cross Site Request Forgery (CSRF)

A9: Using Known Vulnerable Components

A10: Unvalidated Redirects and Forwards
What Didn’t Change

It’s About Risks, Not Just Vulnerabilities

• Title is: “The Top 10 Most Critical Web Application Security Risks”

OWASP Top 10 Risk Rating Methodology

• Based on the OWASP Risk Rating Methodology, used to prioritize Top 10
### OWASP Top 10 Risk Rating Methodology

<table>
<thead>
<tr>
<th>Threat Agent</th>
<th>Attack Vector</th>
<th>Weakness Prevalence</th>
<th>Weakness Detectability</th>
<th>Technical Impact</th>
<th>Business Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Easy</td>
<td>Widespread</td>
<td>Easy</td>
<td>Severe</td>
<td>?</td>
</tr>
<tr>
<td>2</td>
<td>Average</td>
<td>Common</td>
<td>Average</td>
<td>Moderate</td>
<td>?</td>
</tr>
<tr>
<td>3</td>
<td>Difficult</td>
<td>Uncommon</td>
<td>Difficult</td>
<td>Minor</td>
<td>?</td>
</tr>
</tbody>
</table>

**Injection Example**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

1.66 weighted risk rating
What’s Changed?

Risks Added, Risks Merged, Risks Reordered

- Reordered: 7
- Added: 1
- Merged: 2 merged into 1
- Broadened: 1

Development Methodology For 2013

- Same as 2010, but
- Used more sources of vulnerability data
- All vulnerability data made public by each provider

Development Methodology for Next Version?

- More transparency
- Requested vulnerability data format
- Earlier community involvement
<table>
<thead>
<tr>
<th>OWASP Top 10 – 2010 (old)</th>
<th>OWASP Top 10 – 2013 (New)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010-A2 – Cross Site Scripting (XSS)</td>
<td>2013-A2 – Broken Authentication and Session Management</td>
</tr>
<tr>
<td>2010-A3 – Broken Authentication and Session Management</td>
<td>2013-A3 – Cross Site Scripting (XSS)</td>
</tr>
<tr>
<td>2010-A4 – Insecure Direct Object References</td>
<td>2013-A4 – Insecure Direct Object References</td>
</tr>
<tr>
<td>2010-A5 – Cross Site Request Forgery (CSRF)</td>
<td>2013-A5 – Security Misconfiguration</td>
</tr>
<tr>
<td>2010-A6 – Security Misconfiguration</td>
<td>2013-A6 – Sensitive Data Exposure</td>
</tr>
<tr>
<td>2010-A7 – Insecure Cryptographic Storage</td>
<td>2013-A7 – Missing Function Level Access Control</td>
</tr>
<tr>
<td>2010-A8 – Failure to Restrict URL Access</td>
<td>2013-A8 – Cross-Site Request Forgery (CSRF)</td>
</tr>
<tr>
<td>2010-A9 – Insufficient Transport Layer Protection</td>
<td>2013-A9 – Using Known Vulnerable Components (NEW)</td>
</tr>
<tr>
<td>2010-A10 – Unvalidated Redirects and Forwards (NEW)</td>
<td>2013-A10 – Unvalidated Redirects and Forwards</td>
</tr>
</tbody>
</table>

**3 Primary Changes:**

- Merged: 2010-A7 and 2010-A9 -> 2013-A6
- Added New 2013-A9: Using Known Vulnerable Components
  - 2010-A8 broadened to 2013-A7
OWASP Top Ten 2010 - A6
Security Misconfiguration

How Do I Prevent This?
The primary recommendations are to establish all of the following:

2. A process for keeping abreast of and deploying all new software updates and patches in a timely manner to each deployed environment. This needs to include all code libraries as well, which are frequently overlooked.”
The amount of custom code in an application hasn’t changed very much in the past 10 years.
80% Libraries

But library use is growing at a staggering rate
### Vulnerable Libraries

- **29 MILLION vulnerable downloads in 2011**

#### Key Statistics
- **Libraries**: 31
- **Library Versions**: 1,261
- **Organizations**: 61,807
- **Downloads**: 113,939,358

#### Chart Information
- **Safe Download**: 74%
- **Vulnerable Download**: 26%

[Chart: Everyone Uses Vulnerable Libraries]

Some vulnerable components (e.g., framework libraries) can be identified and exploited with automated tools. This expands the threat agent pool beyond targeted attackers to include chaotic actors.

Virtually every application has these issues because most development teams don’t focus on ensuring their components/libraries are up to date. In many cases, the developers don’t even know all the components they are using, never mind their versions. Component dependencies make things even worse.

Full range of weaknesses is possible, including injection, broken access control, XSS ...

The impact could range from minimal to complete host takeover and data compromise.
What Can You Do to Avoid This?

**Ideal**
- Automation checks periodically (e.g., nightly build) to see if your libraries are out of date
- Even better, automation also tells you about known vulnerabilities

**Minimum**
- By hand, periodically check to see if your libraries are out of date and upgrade those that are
- If any are out of date, but you really don’t want to upgrade, check to see if there are any known security issues with these out of date libraries
  - If so, upgrade those

**Could also**
- By hand, periodically check to see if any of your libraries have any known vulnerabilities at this time
  - Check CVE, other vuln repositories
  - If any do, update at least these
### Automation Example for Java – Use Maven 'Versions' Plugin

#### Output from the Maven Versions Plugin – Automated Analysis of Libraries’ Status against Central repository

<table>
<thead>
<tr>
<th>Status</th>
<th>Group Id</th>
<th>Artifact Id</th>
<th>Current Version</th>
<th>Scope</th>
<th>Classifier</th>
<th>Type</th>
<th>Next Version</th>
<th>Next Incremental</th>
<th>Next Minor</th>
<th>Next Major</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>com.fasterxml.jackson.core</td>
<td>jackson-annotations</td>
<td>2.0.4</td>
<td>compile</td>
<td>jar</td>
<td></td>
<td></td>
<td>2.0.5</td>
<td>2.1.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>com.fasterxml.jackson.core</td>
<td>jackson-core</td>
<td>2.0.4</td>
<td>compile</td>
<td>jar</td>
<td></td>
<td></td>
<td>2.0.5</td>
<td>2.1.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>com.fasterxml.jackson.core</td>
<td>jackson-databind</td>
<td>2.0.4</td>
<td>compile</td>
<td>jar</td>
<td></td>
<td></td>
<td>2.0.5</td>
<td>2.1.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>com.google.guava</td>
<td>guava</td>
<td>11.0</td>
<td>compile</td>
<td>jar</td>
<td></td>
<td></td>
<td>11.0.1</td>
<td>12.0-rc1</td>
<td>12.0</td>
</tr>
<tr>
<td></td>
<td>com.ibm.icu</td>
<td>icu4j</td>
<td>49.1</td>
<td>compile</td>
<td>jar</td>
<td></td>
<td></td>
<td>1.0.5</td>
<td>50.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>com.theoryinpractice</td>
<td>halbuilder</td>
<td>1.0.4</td>
<td>compile</td>
<td>jar</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>commons-codec</td>
<td>commons-codec</td>
<td>1.3</td>
<td>compile</td>
<td>jar</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>commons-logging</td>
<td>commons-logging</td>
<td>1.1.1</td>
<td>compile</td>
<td>jar</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>joda-time</td>
<td>joda-time</td>
<td>2.0</td>
<td>compile</td>
<td>jar</td>
<td></td>
<td></td>
<td>2.5.2</td>
<td>2.6.0</td>
<td>2.1</td>
</tr>
<tr>
<td></td>
<td>net.sf.ehcache</td>
<td>ehcache-core</td>
<td>2.5.1</td>
<td>compile</td>
<td>jar</td>
<td></td>
<td></td>
<td>4.1.3</td>
<td>4.2</td>
<td>4.2</td>
</tr>
<tr>
<td></td>
<td>org.apache.httpcomponents</td>
<td>httpclient</td>
<td>4.1.2</td>
<td>compile</td>
<td>jar</td>
<td></td>
<td></td>
<td>4.1.3</td>
<td>4.2</td>
<td>4.2</td>
</tr>
<tr>
<td></td>
<td>org.apache.httpcomponents</td>
<td>httpclient-cache</td>
<td>4.1.2</td>
<td>compile</td>
<td>jar</td>
<td></td>
<td></td>
<td>4.1.3</td>
<td>4.2</td>
<td>4.2</td>
</tr>
<tr>
<td></td>
<td>org.apache.httpcomponents</td>
<td>httpcore</td>
<td>4.1.2</td>
<td>compile</td>
<td>jar</td>
<td></td>
<td></td>
<td>1.1.2</td>
<td>2.0.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>org.slf4j</td>
<td>slf4j</td>
<td>1.7.2</td>
<td>provided</td>
<td>jar</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Most out of Date!**

**Details Developer Needs**

*This can automatically be run EVERY TIME software is built!!*
Two Related Topics Merged

- 2010-A7 – Insecure Cryptographic Storage
- 2010-A9 – Insufficient Transport Layer Protection
- To make room for New 2013-A9: Using Known Vulnerable Components

Storing and Transmitting Sensitive Data Insecurely

- Failure to identify all sensitive data
- Failure to identify all the places that this sensitive data gets stored
  - Databases, files, directories, log files, backups, etc.
- Failure to identify all the places that this sensitive data is sent
  - On the web, to backend databases, to business partners, internal communications
- Failure to properly protect this data in every location
Expanded 2013-A7 – Missing Function Level Access Control

Was: 2010-A8 – Failure to Restrict URL Access

• URLs are one way to access functions
• But not the only way …

Expand to Cover all Ways a Function Can Be Accessed

• URL to function directly
• URL plus parameter value(s) which indicate which function is being accessed
  • e.g., site/somedir/somepage?action=transferfunds

Typical Flaws

• Application simply doesn’t check to see if function invocation is authorized
• Application does check for authorization, but check is flawed. (This would be broken function level access control, but missing is far more common.)
Gather Vulnerability Stats

- Ask previous contributors, solicit new contributors well known to Top 10 team, include unsolicited volunteers
  - 3 New Data Contributors Included: TrustWave, Veracode, Minded Security
- New: Each provider asked to make their data public. All Did.

Analyze Stats, Produce Initial Draft, Release for Public Comment

- Draft Released to OWASP Community Feb 15, 2013
- Public Comment Period Open for 90+ days (thru May 30, 2013)

Final Release Produced

- All Constructive Comments Considered
- Full documentation of Constructive Comments and how they were addressed documented
- Released on June 12, 2013
Gather More Stats More Openly

- Issue Open Call For Vulnerability Stats Providers
- Provide Desired Stats Format (for consistency) and Require Public Reporting
- Consider all Stats Provided by Requested Deadline
- Don’t Ignore Future Looking Threats
  - Like we did with CSRF in 2007, and Vulnerable Components in 2013

Consider Other Stats if They Make Sense

- We only have Vulnerability Prevalence Stats
- What about Stats for Exploitability, Detectability, Impact?
- We tried to consider some Exploitability stats in 2013, but couldn’t find effective public stats

Expand Authoring Team

- Solicit Additional Volunteers
• Video Presentation of Each Item in OWASP Top 10 – 2010 (which is very similar)
  – Dave Wichers at OWASP AppSec DC (2009)
  – http://www.vimeo.com/9006276

• OWASP Top 10 – 2013 Presentation which goes through each item one by one

• Translations of OWASP Top 10 - 2013
  – French, Chinese, and Korean Translations complete
  – Many others are underway
  – https://www.owasp.org/index.php/Top10#tab=Translation_Efforts
Thank you

OWASP Top-10 Project