



# OWASP Top-10 2013

Dave Wichers

OWASP Top 10 Project Lead

OWASP Board Member

COO/Cofounder, Aspect Security



## OWASP

The Open Web Application Security Project



## OWASP

The Open Web Application Security Project

### OWASP Top 10 is an Awareness Document

- **Not a standard...**

### First developed in 2003

- **Was probably 3<sup>rd</sup> or 4<sup>th</sup> OWASP project, after**
  - **Developers Guide**
  - **WebGoat**
  - **Maybe WebScarab ??**

### Released

- **2003, 2004, 2007, 2010, 2013**

# OWASP Top Ten (2013 Edition)



**OWASP**

The Open Web Application Security Project

**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**



### It's About Risks, Not Just Vulnerabilities

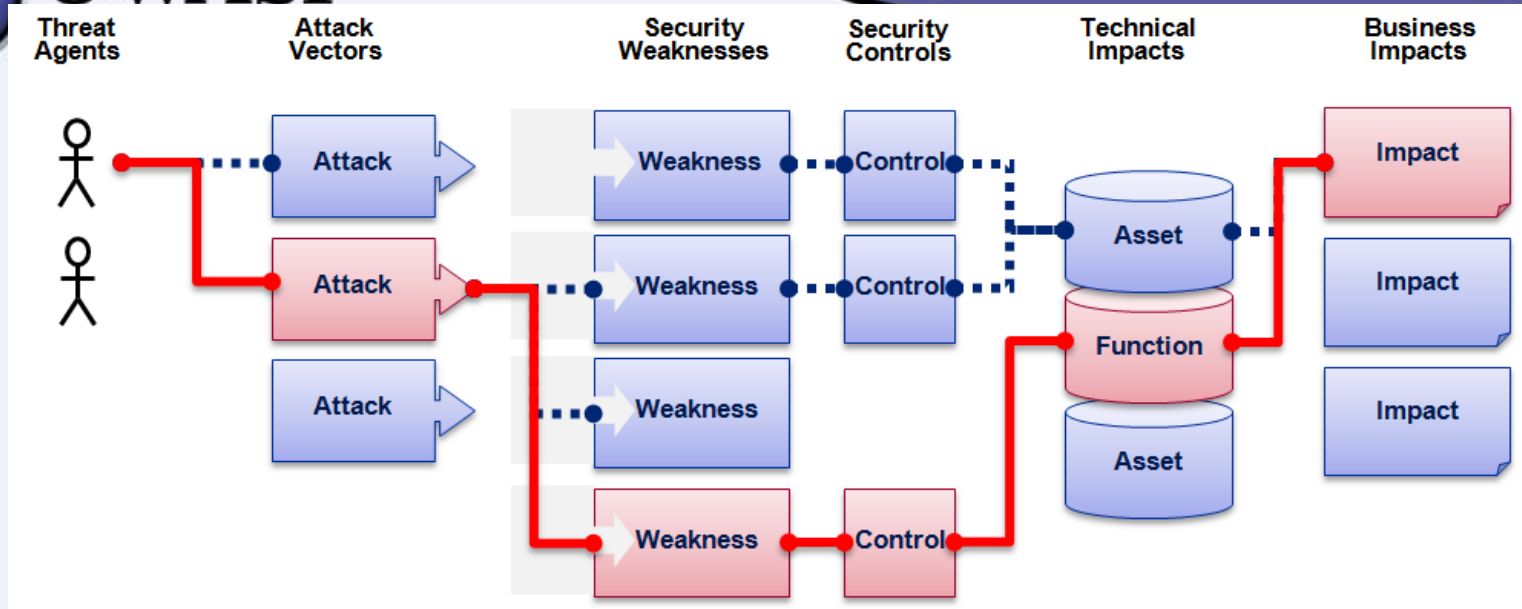
- Title is: “The Top 10 Most Critical Web Application Security Risks”

### OWASP Top 10 Risk Rating Methodology

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



# OWASP Top 10 Risk Rating Methodology



Threat Agent	Attack Vector	Weakness Prevalence	Weakness Detectability	Technical Impact	Business Impact
?	1 Easy	Widespread	Easy	Severe	?
	2 Average	Common	Average	Moderate	
	3 Difficult	Uncommon	Difficult	Minor	
	1	2	2	1	
Injection Example		1.66		*	1

1.66 weighted risk rating

# What's Changed?



**OWASP**

The Open Web Application Security Project

## 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

# Mapping from 2010 to 2013 Top 10



OWASP Top 10 – 2010 (old)	OWASP Top 10 – 2013 (New)
2010-A1 – Injection	2013-A1 – Injection
2010-A2 – Cross Site Scripting (XSS)	2013-A2 – Broken Authentication and Session Management
2010-A3 – Broken Authentication and Session Management	2013-A3 – Cross Site Scripting (XSS)
2010-A4 – Insecure Direct Object References	2013-A4 – Insecure Direct Object References
2010-A5 – Cross Site Request Forgery (CSRF)	2013-A5 – Security Misconfiguration
2010-A6 – Security Misconfiguration	2013-A6 – Sensitive Data Exposure
2010-A7 – Insecure Cryptographic Storage	2013-A7 – Missing Function Level Access Control
2010-A8 – Failure to Restrict URL Access	2013-A8 – Cross-Site Request Forgery (CSRF)
2010-A9 – Insufficient Transport Layer Protection	2013-A9 – Using Known Vulnerable Components (NEW)
2010-A10 – Unvalidated Redirects and Forwards (NEW)	2013-A10 – Unvalidated Redirects and Forwards
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



## OWASP

The Open Web Application Security Project



## OWASP

The Open Web Application Security Project

### OWASP Top 10 - 2010

The Ten Most Critical Web Application Security Risks

## A6 Security Misconfiguration

Threat Agents	Attack Vectors	Security Weakness	Technical Impacts	Business Impacts
<b>Exploitability EASY</b>	<b>Prevalence COMMON</b>	<b>Detectability EASY</b>	<b>Impact MODERATE</b>	
Consider anonymous external attackers as well as users with their own accounts that may attempt to compromise the system. Also consider insiders wanting to disguise their actions.	Attacker accesses default accounts, unused pages, unprotected files and directories, etc. to gain unauthorized access to or knowledge of the system.	Security misconfiguration can happen at any level of an application stack, including the platform, web server, application server, framework, and custom code. Developers and network administrators need to work together to ensure that the entire stack is configured properly. Automated scanners are useful for detecting missing patches, misconfigurations, use of default accounts, unnecessary services, etc.	Such flaws frequently give attackers unauthorized access to some system data or functionality. Occasionally, such flaws result in a complete system compromise.	The system could be completely compromised without you knowing it. All your data could be stolen or modified slowly over time. Recovery costs could be expensive.

### Am I Vulnerable?

Have you performed the proper security hardening across the entire application stack?

- Do you have a process for keeping all your software up to date? This includes the OS, Web/App Server, DBMS, applications, and all code libraries.
- Is everything unnecessary disabled, removed, or not installed (e.g. ports, services, pages, accounts, privileges)?
- Are default account passwords changed or disabled?
- Is your error handling set up to prevent stack traces and other overly informative error messages from leaking?
- Are the security settings in your development frameworks (e.g., Struts, Spring, ASP.NET) and libraries understood and configured properly?

A concerted, repeatable process is required to develop and maintain a proper application security configuration.

### How Do I Prevent This?

The primary recommendations are to establish all of the following:

- A repeatable hardening process that makes it fast and easy to deploy another environment that is properly locked down. Development, QA, and production environments should all be configured identically. This process should be automated to minimize the effort required to setup a new secure environment.
- 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.
- A strong application architecture that provides good separation and security between components.
- Consider running scans and doing audits periodically to help detect future misconfigurations or missing patches.

### Example Attack Scenarios

**Scenario #1:** Your application relies on a powerful framework like Struts or Spring. XSS flaws are found in these framework components you rely on. An update is released to fix these flaws but you don't update your libraries. Until you do, attackers can easily find and exploit these flaws in your app.

**Scenario #2:** The app server admin console is automatically installed and not removed. Default accounts aren't changed. Attacker discovers the standard admin pages are on your server, logs in with default passwords, and takes over.

**Scenario #3:** Directory listing is not disabled on your server. Attacker discovers she can simply list directories to find any file. Attacker finds and downloads all your compiled Java classes, which she reverses to get all your custom code. She then finds a serious access control flaw in your application.

**Scenario #4:** App server configuration allows stack traces to be returned to users, potentially exposing underlying flaws. Attackers love the extra information error messages provide.

### References

OWASP

- [OWASP Development Guide: Chapter on Configuration](#)
- [OWASP Code Review Guide: Chapter on Error Handling](#)
- [OWASP Testing Guide: Configuration Management](#)
- [OWASP Testing Guide: Testing for Error Codes](#)
- [OWASP Top 10 2004 - Insecure Configuration Management](#)

For additional requirements in this area, see the [ASVS requirements area for Security Configuration \(V12\)](#).

External

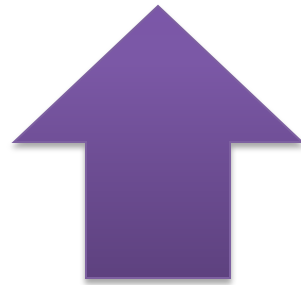
- [PC Magazine Article on Web Server Hardening](#)
- [CWE Entry 2 on Environmental Security Flaws](#)
- [CIS Security Configuration Guides/Benchmarks](#)

## 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.

An iceberg floating in the ocean. The tip of the iceberg, which is above the water line, is labeled '20% Custom Code' with a yellow arrow pointing to it. The much larger part of the iceberg, which is submerged below the water line, is labeled '80% Libraries' with a red arrow pointing to it. The background is a blue sky with light clouds and a dark blue ocean.

20% Custom Code

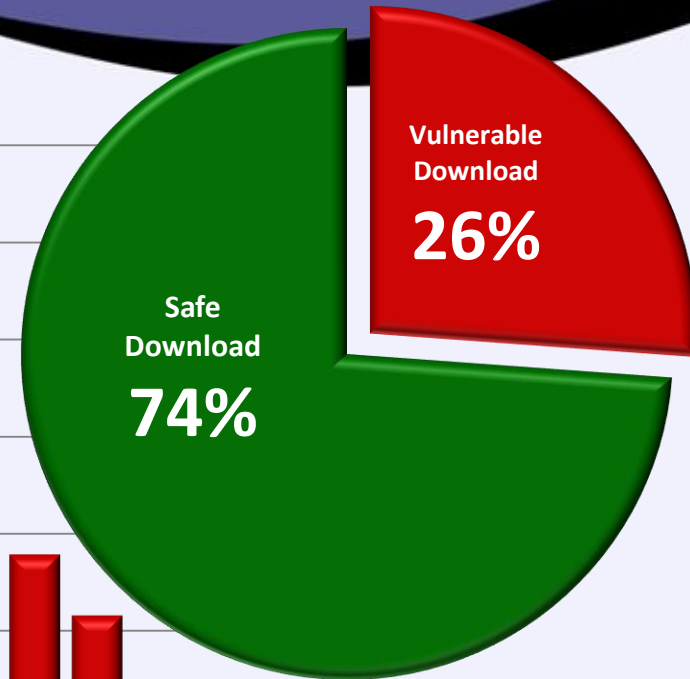
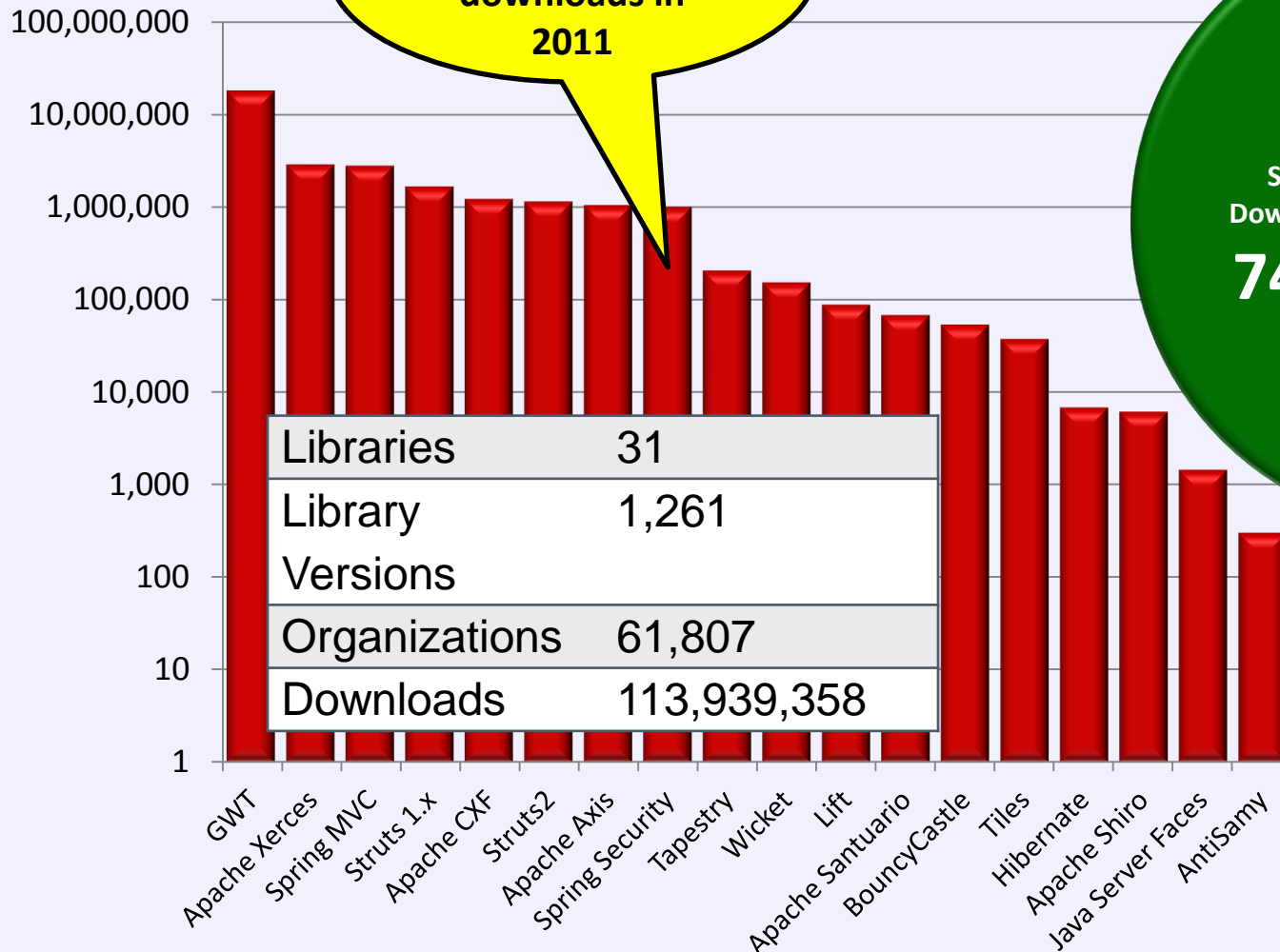
**80% Libraries**

But library use is  
growing at a  
staggering rate

# Everyone Uses Vulnerable Libraries



**29 MILLION  
vulnerable  
downloads in  
2011**





### Vulnerable Components Are Common

- 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

### Widespread

- 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

### Typical Impact

- 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?



## OWASP

The Open Web Application Security Project

### 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 data 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



**OWASP**

The Open Web Application Security Project

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

### Dependencies

Status	Group Id	Artifact Id	Current Version	Scope	Classifier	Type	Next Version	Next Incremental	Next Minor	Next Major
⚠	com.fasterxml.jackson.core	jackson-annotations	2.0.4	compile		jar		2.0.5	2.1.0	
⚠	com.fasterxml.jackson.core	jackson-core	2.0.4	compile		jar		2.0.5	2.1.0	
⚠	com.fasterxml.jackson.core	jackson-databind	2.0.4	compile		jar		2.0.5	2.1.0	
⚠	com.google.guava	guava	11.0	compile		jar		11.0.1	12.0-rc1	12.0
⚠	com.ibm.icu	icu4j	49.1	compile		jar				50.1
⚠	com.theoryinpractise	halbuilder	1.0.4	compile		jar		1.0.5		
⚠	commons-codec	commons-codec	1.3	compile		jar			1.4	
✅	commons-logging	commons-logging	1.1.1	compile		jar				
⚠	joda-time	joda-time	2.0	compile		jar			2.1	
⚠	net.sf.ehcache	ehcache-core	2.5.1	compile		jar		2.5.2	2.6.0	
⚠	org.apache.httpcomponents	httpclient	4.1.2	compile		jar		4.1.3	4.2	
⚠	org.apache.httpcomponents	httpclient-cache	4.1.2	compile		jar		4.1.3	4.2	
⚠	org.apache.httpcomponents	httpcore	4.1.2	compile		jar		4.1.3	4.2	
⚠	org.jdom	jdom	1.1	compile		jar		1.1.2		2.0.0
✅	org.slf4j	slf4j-api	1.7.2	provided		jar				

**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



**OWASP**

The Open Web Application Security Project

## 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.)



# OWASP Top 10 2013 Development Methodology



**OWASP**

The Open Web Application Security Project

## 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
  - [https://www.owasp.org/images/3/3d/OWASP\\_Top\\_10\\_-\\_2013\\_Final\\_Release\\_-\\_Change\\_Log.docx](https://www.owasp.org/images/3/3d/OWASP_Top_10_-_2013_Final_Release_-_Change_Log.docx)
- Released on June 12, 2013

# OWASP Top 10 Future Development Methodology Ideas



**OWASP**

The Open Web Application Security Project

## 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**
  - <https://www.owasp.org/index.php/Top10>
- **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](https://www.owasp.org/index.php/Top10#tab=Translation_Efforts)



Thank you  
OWASP Top-10 Project

