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

About the OWASP Top 10

OWASP Top 10 is an Awareness Document

- Not a standard…

First developed in 2003

- Was probably 3rd or 4th 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

What Didn't Change

It's About Risks, Not Just Vulnerabilities

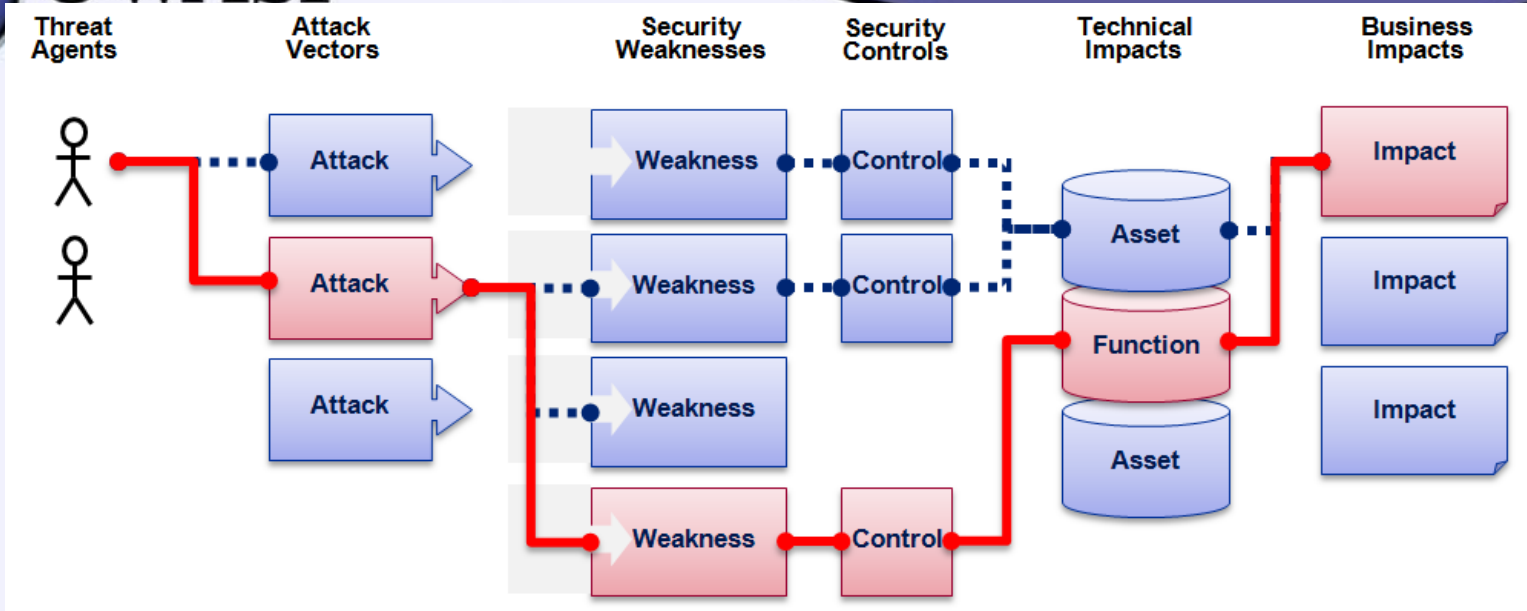
- Title is: "The Top 10 Most Critical Web Application Security Risks"

OWASP Top 10 Risk Rating Methodology

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

OWASP Top 10 Risk Rating Methodology

OWASP



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	
		1.66	*	1	

Injection Example

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

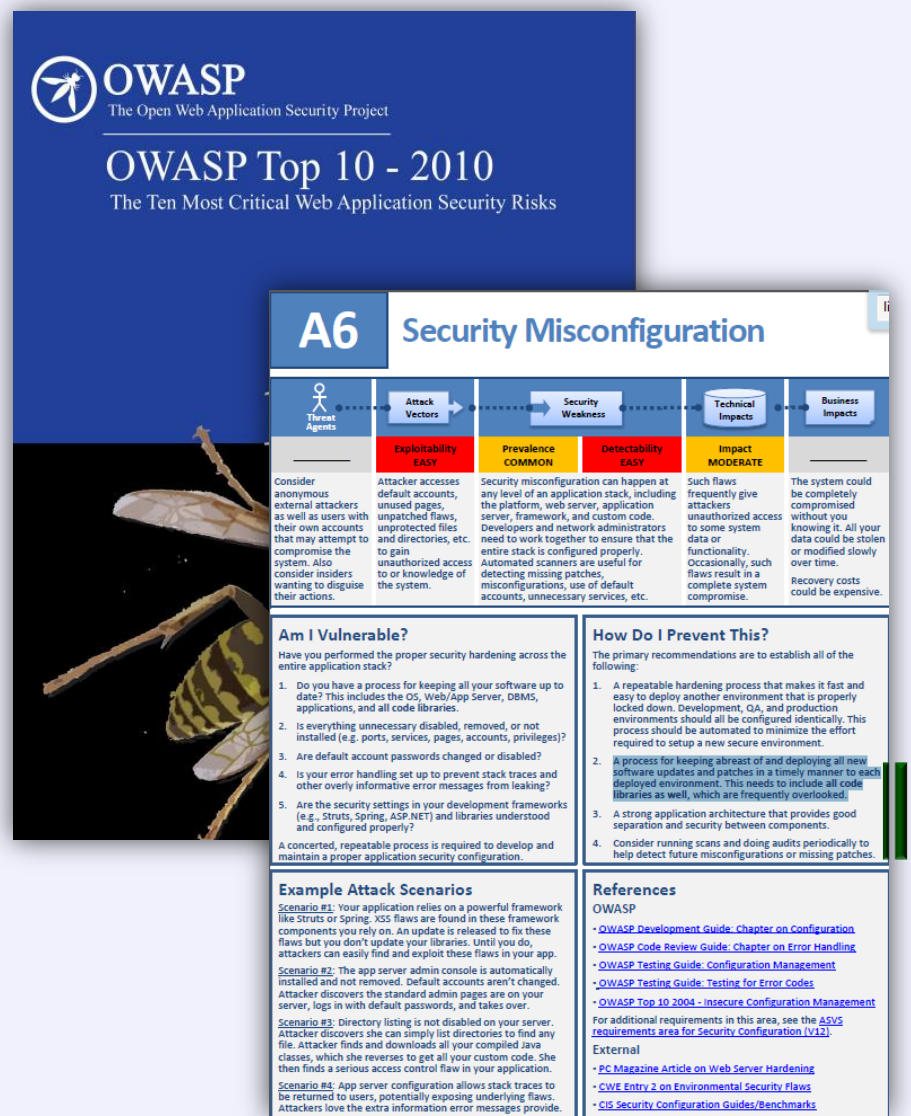
Mapping from 2010 to 2013 Top 10

OWASP
The Open Web Application Security Project

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



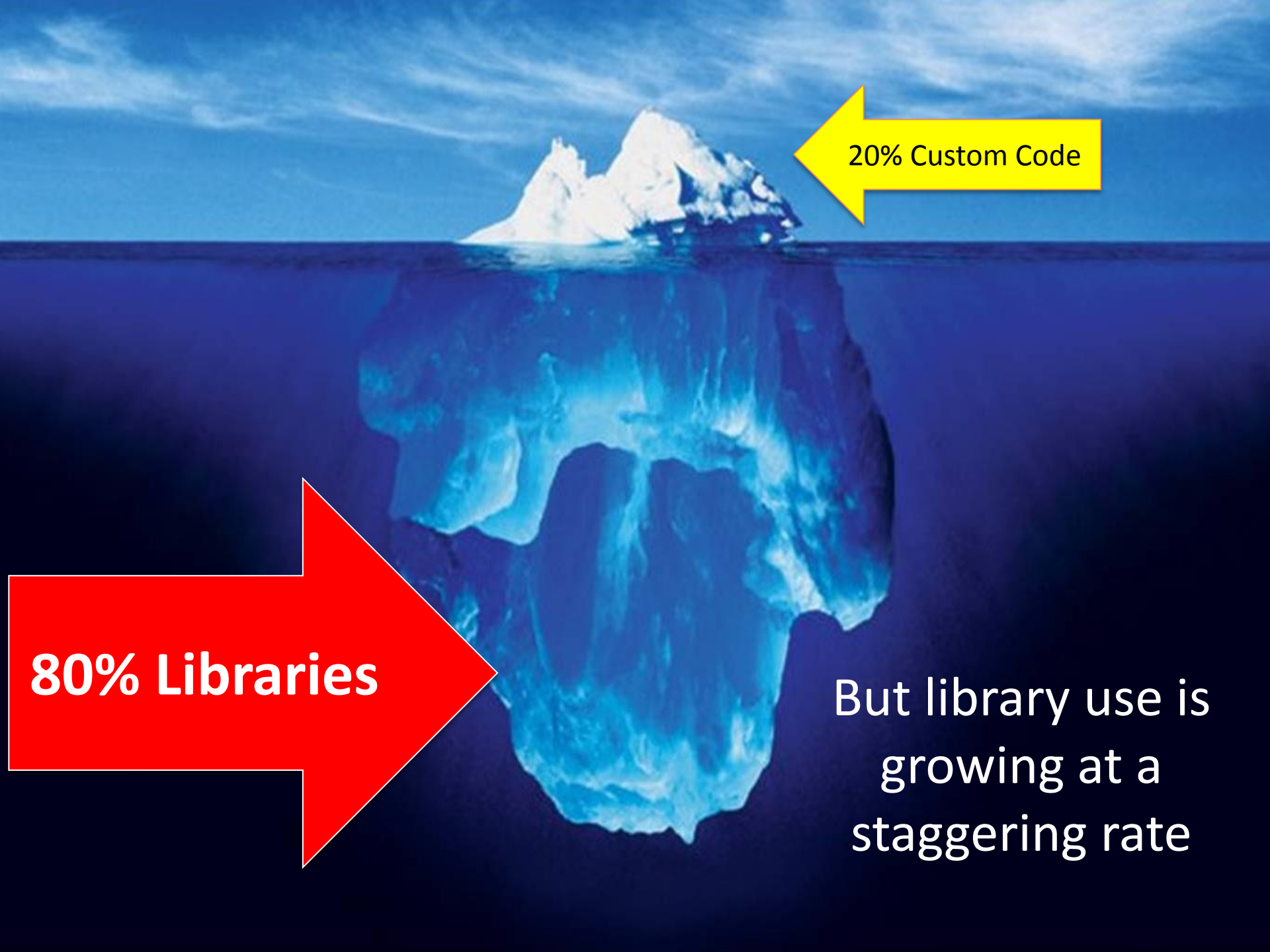
How Do I Prevent This?

The primary recommendations are to establish all of the following:

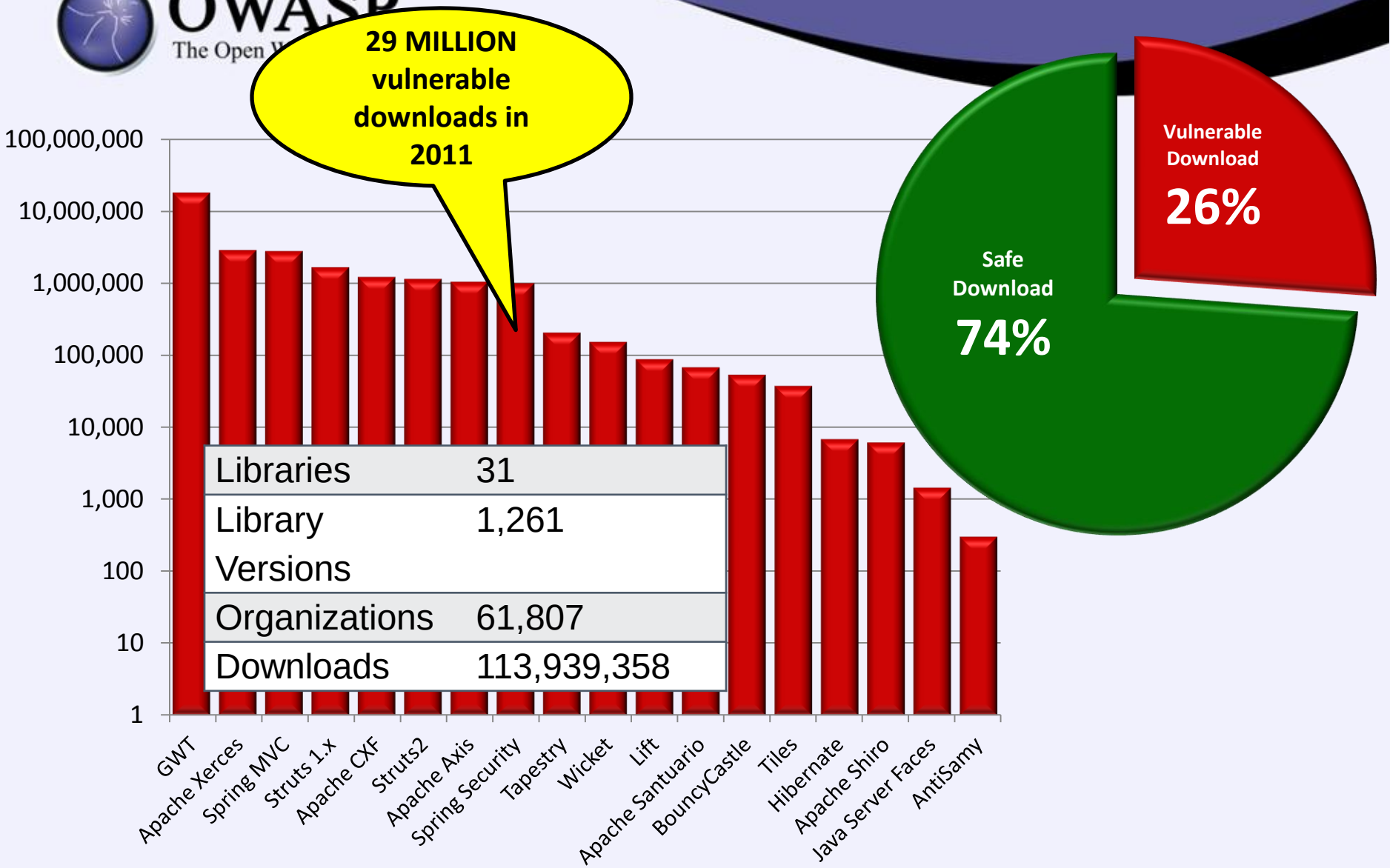
…

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.


20% Custom Code
80% Libraries
But library use is growing at a staggering rate

Everyone Uses Vulnerable Libraries



https://www.aspectsecurity.com/news/press/the-unfortunate-reality-of-insecure-libraries

2013-A9 – Using Known Vulnerable Components

OWASP
The Open Web Application Security Project

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?

Ideal

- Automation checks periodically (e.g., nightly build) to see if your libraries are out of date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

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!!

The Merged 2013-A6 – Sensitive Data Exposure

OWASP
The Open Web Application Security Project

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.)

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
- Released on June 12, 2013

OWASP Top 10 Future Development Methodology Ideas

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

OWASP Top 10 Resources

- **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**

Thank you
OWASP Top-10 Project