OWASP Top 10 – 2017: What you need to know

BOSTON CODE CAMP 29

ROBERT HURLBUT

APRIL 7, 2018

Who am I?

Robert Hurlbut
SVP, Threat Modeling Architect / Lead
Cyber Security Technology
Bank of America

Disclaimer: The following presentation represents my views only and not those of my employer.

Boston Code Camp 29 - Thanks to our Sponsors!

Platinum

Microsoft

Gold

HealthcareSource
Quality Talent Suite™

Silver

slalom

BLUEMETAL

Robert Half Technology

Bronze

RGood Software

In-Kind Donations

GrapeCity

MANNING PUBLICATIONS

JetBrains

steema software

Agenda

OWASP Top 10: History

OWASP Top 10: Process

OWASP Top 10 - 2017

What’s next?

OWASP Top 10: History

OWASP Top 10

Goal - raise security awareness for developers and managers

Became de facto application security standard

Delivered every 3-4 years

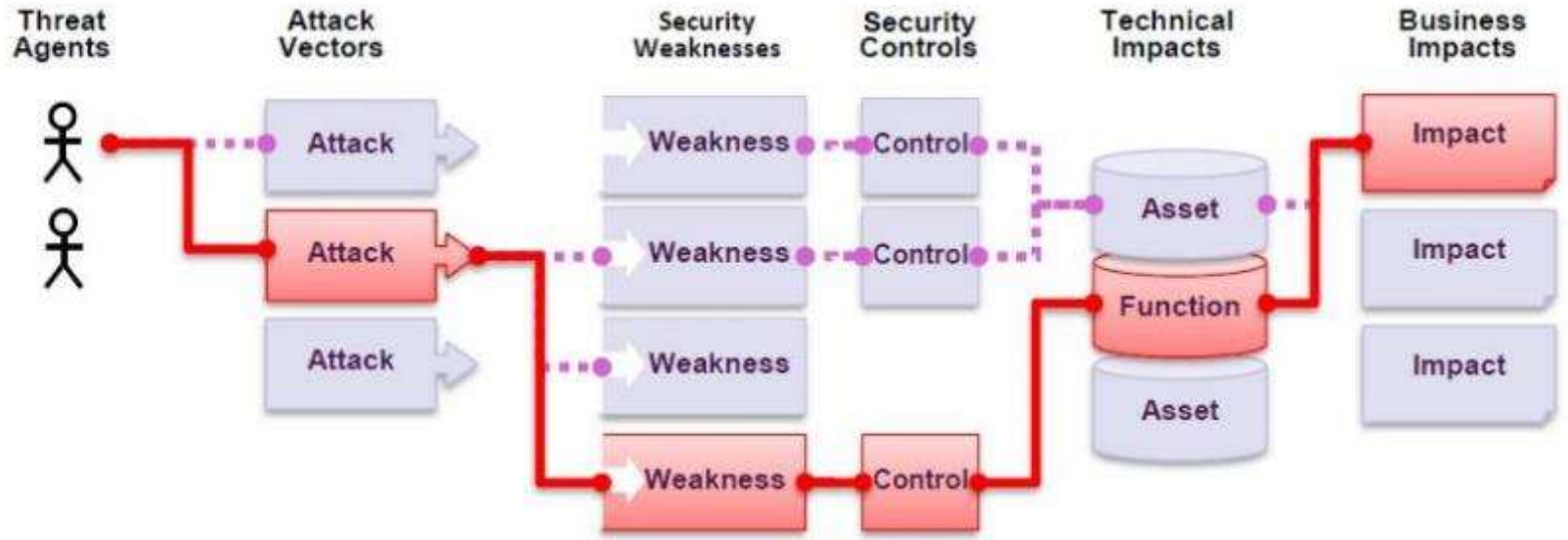
Previous – 2003, 2004, 2007, 2010, 2013

Current - 2017

OWASP Top 10: Web Application Security Risks

What Are Application Security Risks?

Attackers can potentially use many different paths through your application to do harm to your business or organization. Each of these paths represents a risk that may, or may not, be serious enough to warrant attention.



*https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf (pg 5)

Why OWASP Top 10 revised?

Risks / threats:

- Change in priority
  - Started listing by priority in the 2010 and 2013 releases
- No longer prominent
- Updated for better understanding
- Change along with industry and trends

OWASP Top 10 – 2017: History

In early 2017, a Release Candidate preview version was released

- It was proposed by Jeff Williams and Dave Wickers, long-time leaders of the OWASP Top 10 Project

OWASP Summit 2017, London in June: Andrew van der Stock became leader of OWASP Top 10 Project

- Other members: Brian Glas, Neil Smithline, Torsten Gigler

Schedule of releases in 2017:

- April, 2017 – Release Candidate 1
- September, 2017 – Release Candidate 2
- November, 2017 – Final

OWASP Top 10: Process

New approaches to OWASP Top 10

Substantial community feedback

Extensive data collected

- 40+ data submissions from firms specializing in application security
- 500+ individuals completing an industry survey

Data spans vulnerabilities gathered from hundreds of organizations and over 100,000 real-world applications and APIs

Top 10 items were selected and prioritized according to data in combination with consensus estimates of exploitability, detectability, and impact

Changes to OWASP Top 10

OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	∪	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	∪	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

*https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf (pg 4)

Changes to OWASP Top 10

A4:2013-Insecure Direct Object References and **A7:2013-Missing Function Level Access Control** merged into **A5:2017-Broken Access Control**

A8:2013-Cross-Site Request Forgery (CSRF) dropped

- Mostly covered by frameworks now – found in 5% of applications reviewed

A10:2013-Unvalidated Redirects and Forwards dropped

- Found in 8% of applications reviewed, XXE (**A4:2017-External XML Entities**) won out to be included in OWASP Top 10 - 2017

OWASP Top 10-2017: Risk Factor Summary

RISK	Threat Agents	Attack Vectors: Exploitability	Security Weakness: Prevalence	Security Weakness: Detectability	Impacts: Technical	Impacts: Business	Score
A1:2017-Injection	App Specific	EASY: 3	COMMON: 2	EASY: 3	SEVERE: 3	App Specific	8.0
A2:2017-Authentication	App Specific	EASY: 3	COMMON: 2	AVERAGE: 2	SEVERE: 3	App Specific	7.0
A3:2017-Sens. Data Exposure	App Specific	AVERAGE: 2	WIDESPREAD: 3	AVERAGE: 2	SEVERE: 3	App Specific	7.0
A4:2017-XML External Entities (XXE)	App Specific	AVERAGE: 2	COMMON: 2	EASY: 3	SEVERE: 3	App Specific	7.0
A5:2017-Broken Access Control	App Specific	AVERAGE: 2	COMMON: 2	AVERAGE: 2	SEVERE: 3	App Specific	6.0
A6:2017-Security Misconfiguration	App Specific	EASY: 3	WIDESPREAD: 3	EASY: 3	MODERATE: 2	App Specific	6.0
A7:2017-Cross-Site Scripting (XSS)	App Specific	EASY: 3	WIDESPREAD: 3	EASY: 3	MODERATE: 2	App Specific	6.0
A8:2017-Insecure Deserialization	App Specific	DIFFICULT: 1	COMMON: 2	AVERAGE: 2	SEVERE: 3	App Specific	5.0
A9:2017-Vulnerable Components	App Specific	AVERAGE: 2	WIDESPREAD: 3	AVERAGE: 2	MODERATE: 2	App Specific	4.7
A10:2017-Insufficient Logging&Monitoring	App Specific	AVERAGE: 2	WIDESPREAD: 3	DIFFICULT: 1	MODERATE: 2	App Specific	4.0

*https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf (pg 23)

OWASP Top 10 - 2017

OWASP Top 10 - 2017

Find the OWASP Top 10 2017 document at the OWASP Top 10 Project page:

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Log in Request account

Category Discussion

Read View source View history Search

Category:OWASP Top Ten Project

Help

Main | Translation Efforts | OWASP Top 10 for 2013 | OWASP Top 10 for 2010 | Project Details | Some Commercial & OWASP Uses of the Top 10

FLAGSHIP mature projects

OWASP Top 10 2017 Released

The OWASP Top 10 - 2017 is now available.

OWASP Top 10 Most Critical Web Application Security Risks

The OWASP Top 10 is a powerful awareness document for web application security. It represents a broad consensus about the most critical security risks to web applications. Project members include a variety of security experts from around the world who have shared their expertise to produce this list.

We urge all companies to adopt this awareness document within their organization and start the process of ensuring that their web applications minimize these risks. Adopting the OWASP Top 10 is perhaps the most effective first step towards changing the software development culture within your organization into one that produces secure code.

Translation Efforts

The OWASP Top 10 has been translated to many different languages by numerous volunteers. These translations are available as follows:

- OWASP Top 10 - 2017 translations are currently underway
- All versions of the OWASP Top 10 - 2013
- All versions of the OWASP Top 10 - 2010

Related Projects

- OWASP Mobile Top 10 Risks

Home
About OWASP
Acknowledgements
Advertising
AppSec Events
Books
Brand Resources
Chapters
Donate to OWASP
Downloads
Funding
Governance
Initiatives
Mailing Lists
Membership
Merchandise
News
Community portal
Presentations
Press
Projects
Video
Volunteer

Reference
Activities
Attacks
Code Snippets
Controls
Glossary
How To
Java Project
.NET Project
Principles
Technologies
Threat Agents

Quick Download

- OWASP Top 10 - 2017 - PDF
- OWASP Top 10 2013 - PDF
- OWASP Top 10 2013 - wiki
- OWASP Top 10 2013 Presentation (PPTX)

Donate to OWASP

Donate

Get Involved

- Top 10 Issues on GitHub (preferred)
- Project Email List

News and Events

- [20 Oct 2017] OWASP Top 10 2017 - RC2 Published
- [20 May 2016] OWASP Top

Covering OWASP Top 10 - 2017

What is it?

How it happens?

Why do we care?

How to prevent?

A1:2017 – Injection

What is it?

- Allowing untrusted data to be sent as part of a command or query
- Applies to SQL, NoSQL, OS, LDAP, etc. types of injection attacks

How it happens?

- Website, app, or device incorporates user input within a command
- Attacker inserts “payload” command into the input
- If input not verified, attacker “injects” and runs own commands

Why do we care?

- Attackers can make commands to control website, apps, and data

A1:2017 – Injection (Examples)

SQL Injection

Scenario #1:

An application uses untrusted data:

```
String query = "SELECT * FROM accounts WHERE
custID='" + request.getParameter("id") + "'";
```

Scenario #2:

Blind trust in frameworks may result in queries that are still vulnerable, (e.g. Hibernate Query Language (HQL)):

```
Query HQLQuery = session.createQuery("FROM accounts
WHERE custID='" + request.getParameter("id") + "'");
```

Attacker modifies '**id**' parameter value in browser to send: ' or '1'='1

```
http://example.com/app/accountView?id=' or '1'='1
```

A1:2017 – Injection

How to prevent?

- Keep data separate from commands and queries
- Use a safe API (i.e. ORMs, etc.)
- Use positive or “whitelist” server-side input validations
- For dynamic queries, escape special characters

A2:2017 – Broken Authentication

What is it?

- Incorrectly implemented authentication and session management functions

How does it happen?

- Passwords can be guessed or stolen if not protected
- With complexity, attackers find areas where user credentials or sessions not protected
- Attackers hijack user's access and potentially their data

Why do we care?

- Attackers can hijack user's or administrator's session and have access to everything available on an account, including all data from an account

A2:2017 – Broken Authentication

How to prevent?

- Implement multi-factor authentication
- Don’t deploy with default credentials
- Check for weak passwords (i.e. top 10,000 worst passwords)
- Check out NIST 800-63 B’s guidelines for passwords
- Manage Session IDs properly

A3:2017 – Sensitive Data Exposure

What is it?

- Many web technologies weren't designed to handle financial or personal data transfers

How does it happen?

- Data stored or transferred as plain text
- Older / weaker encryption still in use
- Data decrypted carelessly, allowing attackers to gain access and exploit the data

Why do we care?

- When attacker has passwords and credit card numbers, they can do real damage

A3:2017 – Sensitive Data Exposure

How to prevent?

- Classify data as processed, stored, or transmitted – determine sensitivity
- Apply controls per data classification
- Encrypt all sensitive data at rest, in transit, etc. with good secure algorithms and protocols

A4:2017 – XML External Entities (XXE)

New to OWASP Top 10

What is it?

- XML “entities” can be used to request local data or files

How does it happen?

- Application accepts XML directly or XML uploads (i.e. blog posts, etc.) especially from untrusted sources
- Attacker sends malicious data lookup values to request and display data from local file

Why do we care?

- Attackers can gain access to any data stored locally, or can further pivot to attack other internal systems

A4:2017 – XML External Entities (XXE) (Examples)

Denial of Service (DoS)

Accessing a local resource that may not return

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///dev/random" >]>
  <foo>&xxe;</foo>
```

A4:2017 – XML External Entities (XXE) (Examples)

Denial of Service (DoS)

Billion Laughs Attack

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
<!ENTITY lol "lol">
<!ENTITY lol2 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
<!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
<!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
<!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
<!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
<!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
<!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
<!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

A4:2017 – XML External Entities (XXE) (Examples)

Remote Code Execution

PHP: If "expect" module is loaded, we can get RCE.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <!DOCTYPE foo [ <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "expect://id" >]>
   <creds>
     <user>&xxe;</user>
     <pass>mypass</pass>
   </creds>
```

The response from the server will be something like:

```
You have logged in as user uid=0(root) gid=0(root)
groups=0(root)
```

A4:2017 – XML External Entities (XXE) (Examples)

Revealing Files

Disclosing /etc/passwd or other targeted files:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
  <foo>&xxe;</foo>

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///c:/boot.ini" >]>
  <foo>&xxe;</foo>
```

A4:2017 – XML External Entities (XXE)

How to prevent?

- Train developers about XXE and its dangers
- Use less complex data formats such as JSON and avoid serialization of sensitive data
- Patch XML processors and libraries in use
- Disable XML external entity and DTD processing in all XML parsers (see OWASP Cheat Sheet ‘XXE Prevention’)

A5:2017 – Broken Access Control

Previously **A4:2013-Insecure Direct Object Reference** and **A7:2013-Missing Function Level Access Control** (merged)

What is it?

- Improper enforcement of what authenticated users are allowed to do

How does it happen?

- Gaining unauthorized access because access checks not done or inadequate

Why do we care?

- Attackers can gain access to (and modify) data, accounts, and functions they shouldn't

A5:2017 – Broken Access Control (Example)

Access as Admin

Scenario:

An attacker simply force browses to target URLs.
Admin rights are required for access to the admin page.

http://example.com/app/getappInfo
http://example.com/app/admin_getappInfo

If an unauthenticated user can access either page, it's a flaw. If a non-admin can access the admin page, this is a flaw.

A5:2017 – Broken Access Control

How to prevent?

- Access control should be enforced in trusted server-side code or server-less API (where attackers can't modify access control check or metadata)
- Re-use access control mechanisms
- Log access control failures
- Rate limit API / controller access

A6:2017 – Security Misconfiguration

What is it?

- Manual, ad hoc, insecure, or lack of security configurations that enable unauthorized access

How does it happen?

- People get busy
- Things get missed
- Prioritization decisions are made / vulnerabilities left unchecked
- Default passwords are left in place

Why do we care?

- Novice attackers can find and access valuable systems and data

A6:2017 – Security Misconfiguration

How to prevent?

- Use repeatable hardening process to enable fast and easy deployments – Dev/QA/Prod should all be configured identically with different credentials
- Use segmented application architecture providing effective, secure separation between components or tenants
- Automate verification of effectiveness of configurations

A7:2017 – Cross-Site Scripting (XSS)

What is it?

- Web application includes untrusted data in a web page without proper validation or escaping

How does it happen?

- Web page or app utilizes user-entered content as part of resulting page without checking bad stuff
- Malicious user could enter content that includes HTML entities

Why do we care?

- Attackers can change behavior of app, can direct data to their own systems, or corrupt or overwrite existing data

A7:2017 – Cross-Site Scripting (XSS)

How to prevent?

- Separate untrusted data from active browser content
- Use frameworks that automatically escape XSS by design
- See OWASP Cheat Sheet on ‘XSS Prevention’
- Enable Content Security Policy (CSP)

A8:2017 – Insecure Deserialization

New, community supported

What is it?

- Receipt of hostile serialized objects resulting in remote code execution

How does it happen?

- Deserialized data can be modified to include malicious code

Why do we care?

- Attackers build illegitimate objects that execute commands within the infected application

A8:2017 – Insecure Deserialization (Examples)

Example Attack Scenarios

Scenario #1:
A React application calls a set of Spring Boot microservices. The application is serializing user state and passing it back and forth with each request.

An attacker notices the "R00" Java object signature, and uses the Java Serial Killer tool* to gain remote code execution on the application server.

**Java Serial Killer is a Burp extension to perform Java Deserialization Attacks*

A8:2017 – Insecure Deserialization (Examples)

Example Attack Scenarios

Scenario #2:

A PHP forum uses PHP object serialization to save a "super" cookie, containing the user's user ID, role, password hash, and other state:

```
a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";
i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

An attacker changes the serialized object to give themselves admin privileges:

```
a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";
i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

A8:2017 – Insecure Deserialization

How to prevent?

- Ideally, don’t accept serialized objects from untrusted sources
- Implement integrity checks such as digital signatures
- Enforce strict type constraints
- Isolate / run code in low privilege environments
- Log deserialization failures

A9:2017 – Using Components with Known Vulnerabilities

What is it?

- Finding and exploiting already-known vulnerabilities before being fixed

How does it happen?

- Organizations fail to keep software up-to-date
- When exploit is made public or patch released, attackers know the vulnerabilities may not be patched yet
- Attackers have a windows from days to years to find systems or applications that are still vulnerable

Why do we care?

- Public information can lead attackers to a recommended path to exploit (there is no excuse for organizations in not patching)

A9:2017 – Using Components with Known Vulnerabilities

How to prevent?

- Implement patch management
- Use patch management to remove unused dependencies and features
- Inventory versions
- Obtain components from official sources over secure links
- Ongoing plan to monitor, triage, and apply updates / configuration changes
- Look at “OWASP Dependency Check” and “OWASP Dependency Track”

A10:2017 – Insufficient Logging & Monitoring

New, community supported

What is it?

- Insufficient monitoring allows attackers to work unnoticed

How does it happen?

- Organizations are not tracking logins, transactions, traffic, and more

Why do we care?

- Attackers rely on lack of monitoring to exploit vulnerabilities before being detected

A10:2017 – Insufficient Logging & Monitoring

How to prevent?

- Log all login / access control failures / server-side input validation failures with enough information to identify suspicious or malicious accounts
- Make sure log files use generalized format for central consumption
- Establish effective monitoring and alerting
- Establishing incident response and recovery plan

What’s next?

Future OWASP Top 10

OWASP Top 10 Team still actively translating OWASP Top 10-2017 to multiple languages

Next OWASP Top 10 anticipated for 2020

- Looking for more data (including anonymous)
- Recommendation: automate collection of pen testing results

Other OWASP Resources

- OWASP Application Security Verification Standard (ASVS) for application security requirements
- OWASP Prevention Cheat Sheets for guidance on designing security in from the beginning
- OWASP Proactive Controls for standard security controls
- OWASP Software Assurance Maturity Model (SAMM) for software security strategy throughout entire organization
- OWASP Education Project for training materials such as WebGoat, Juice Shop, DevSlop, etc.

Questions?