OWASP Top 10 2017

David Johansson

Principal Consultant, Synopsys

Presentation material contributed by Andrew van der Stock
David Johansson

Security consultant with 10 years in AppSec
Helping clients design and build secure software
Develop and deliver security training
Based in London, working for Synopsys

(@securitybits)
Road to release

OWASP Top 10 RC1
WE CAN'T STOP HERE. THIS IS BAT COUNTRY.
Criticism – valid and invalid

• “Not OWASP like”
  • The new additions *A7 Insufficient Attack Protection* and *A10 Underprotected APIs* boiled down to “failure to buy a tool”
  • From a vendor who sets the standard
  • From a vendor who owns the tool type market

• John Steven and others had ontological issues with the mix of both controls and vulnerabilities (“Define vulnerability. Is that a vulnerability?”)

• Others had problems with the data quality

• Showed us people really care about the OWASP Top 10!
The community called for change!

"Make the OWASP Top 10 Great Again"

AppSec USA 2017 Keynote by Jim Manico and John Steven
Leadership

- Dave Wichers and Jeff Williams stood down
- Handed it over to Andrew van der Stock
- Immediately appointed co-leaders
  - Neil Smithline (participated since 2004)
  - Torsten Gigler (German translator since 2010)
  - And the team added ... Brian Glas (data geek)
Data Call

- Need data for 2016
- Need qualitative survey data for two replacements for A7 and A10
- Brian Glas designed the new survey
- 500+ responses
- Obtained a great deal more data, including from HPE (Fortify), Veracode, CheckMarx, BugCrowd, and Synopsys. Over 114,000 apps form data set.
### Incidence Rate vs. CWE

<table>
<thead>
<tr>
<th>CWE Description</th>
<th>Incidence Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security Misconfiguration (CWE-2,1...)</td>
<td>42.70%</td>
</tr>
<tr>
<td>Cross-Site Scripting (XSS) (CWE-79)</td>
<td>28.09%</td>
</tr>
<tr>
<td>Cryptographic (CWEs-310/326/32...)</td>
<td>20.91%</td>
</tr>
<tr>
<td>Information Leakage/Disclosure (CWE-247)</td>
<td>17.52%</td>
</tr>
<tr>
<td>XML eXtrenal Entity Injection (XXE) (CWE-449)</td>
<td>11.10%</td>
</tr>
<tr>
<td>SQL Injection (CWE-89)</td>
<td>10.70%</td>
</tr>
<tr>
<td>Unchecked Redirect (CWE-601)</td>
<td>8.77%</td>
</tr>
<tr>
<td>Path Traversal (CWE-22)</td>
<td>8.00%</td>
</tr>
<tr>
<td>Authentication (CWE-287)</td>
<td>5.71%</td>
</tr>
<tr>
<td>Cleartext Transmission of Sensitive Information (CWE-</td>
<td>5.05%</td>
</tr>
<tr>
<td>384)</td>
<td></td>
</tr>
<tr>
<td>Session Fixation (CWE-384)</td>
<td>4.91%</td>
</tr>
<tr>
<td>Command Injection (CWE-7778)</td>
<td>4.84%</td>
</tr>
<tr>
<td>Use of Known Vuln Libraries (NEW...FSF)</td>
<td>4.51%</td>
</tr>
<tr>
<td>Insufficient Anti-automation (CWE...)</td>
<td>3.41%</td>
</tr>
<tr>
<td>Improper (Function Level) Access Control (CWE...)</td>
<td>2.64%</td>
</tr>
<tr>
<td>Error Handling (CWE-388)</td>
<td>2.64%</td>
</tr>
<tr>
<td>Cross-Site Request Forgery (CSRF) (CWE-35)</td>
<td>2.18%</td>
</tr>
<tr>
<td>Missing Authorization (CWE-285)</td>
<td>2.10%</td>
</tr>
<tr>
<td>Clickjacking Vulnerabilities Found (CWE-99)</td>
<td>1.80%</td>
</tr>
<tr>
<td>Denial of Service (DOS) (CWE-400)</td>
<td>1.57%</td>
</tr>
<tr>
<td>Server-Side Request Forgery (SSRF)</td>
<td>1.29%</td>
</tr>
<tr>
<td>Unvalidated Forward (No CWE)</td>
<td>1.28%</td>
</tr>
<tr>
<td>DOM-Based XSS (No CWE)</td>
<td>0.99%</td>
</tr>
<tr>
<td>Insecure Direct Object Reference (CWE-220)</td>
<td>0.88%</td>
</tr>
<tr>
<td>Mass Assignment (CWE-915)</td>
<td>0.85%</td>
</tr>
<tr>
<td>Cleartext Storage of Sensitive Information (CWE-329)</td>
<td>0.82%</td>
</tr>
<tr>
<td>Unrestricted Upload of File with Data (CWE-183)</td>
<td>0.74%</td>
</tr>
<tr>
<td>Hibernate Injection (CWE-564)</td>
<td>0.64%</td>
</tr>
<tr>
<td>Insufficient Security Logging (CWE-323)</td>
<td>0.64%</td>
</tr>
</tbody>
</table>

Data Call Summary
Ordering

- We ordered in risk (impact x likelihood), which means CVSS x (survey | data)
- Represents our best understanding of 2017 issues

Andrew van der Stock • @vanderaj
The @OWASPTop10 has data that suggests how would order works best for you? Pls RT

- 9% Same order as now
- 20% Impact (severity) order
- 15% Data (likelihood) order
- 56% Risk impact x likelihood

210 votes • Final results
GitHub

• Everything is in GitHub
• Open: Moved to GitHub
• Open: Data and analysis
• Traceable: Issues
• Translatable: Markdown
What’s new

OWASP Top 10 2017
<table>
<thead>
<tr>
<th>OWASP Top 10 - 2013</th>
<th>OWASP Top 10 - 2017</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1 - Injection</td>
<td>A1:2017-Injection</td>
</tr>
<tr>
<td>A2 - Broken Authentication and Session Management</td>
<td>A2:2017-Broken Authentication</td>
</tr>
<tr>
<td>A3 - Cross-Site Scripting (XSS)</td>
<td>A3:2017-Sensitive Data Exposure</td>
</tr>
<tr>
<td>A4 - Insecure Direct Object References [Merged+A7]</td>
<td>A4:2017-XML External Entities (XXE) [NEW]</td>
</tr>
<tr>
<td>A5 - Security Misconfiguration</td>
<td>A5:2017-Broken Access Control [Merged]</td>
</tr>
<tr>
<td>A6 - Sensitive Data Exposure</td>
<td>A6:2017-Security Misconfiguration</td>
</tr>
<tr>
<td>A8 - Cross-Site Request Forgery (CSRF)</td>
<td>A8:2017-Insecure Deserialization [NEW, Community]</td>
</tr>
<tr>
<td>A9 - Using Components with Known Vulnerabilities</td>
<td>A9:2017-Using Components with Known Vulnerabilities</td>
</tr>
<tr>
<td>A10 - Unvalidated Redirects and Forwards</td>
<td>A10:2017-Insufficient Logging&amp;Monitoring [NEW, Comm.]</td>
</tr>
</tbody>
</table>
### Insecure Deserialization

**Threat Agents** → **Attack Vectors** → **Security Weakness** → **Impacts**

<table>
<thead>
<tr>
<th>Exploitability: 1</th>
<th>Prevalence: 2</th>
<th>Detectability: 2</th>
<th>Technical: 3</th>
<th>Business?</th>
</tr>
</thead>
</table>

**Exploitation of deserialization is somewhat difficult, as off the shelf exploits rarely work without changes or tweaks to the underlying exploit code.**

**This issue is included in the Top 10 based on an industry survey and not on quantifiable data. Some tools can discover deserialization flaws, but human assistance is frequently needed to validate the problem. It is expected that prevalence data for deserialization flaws will increase as tooling is developed to help identify and address it.**

**The impact of deserialization flaws cannot be understated. These flaws can lead to remote code execution attacks, one of the most serious attacks possible. The business impact depends on the protection needs of the application and data.**

---

**Is the Application Vulnerable?**

Applications and APIs will be vulnerable if they deserialize hostile or tampered objects supplied by an attacker. This can result in two primary types of attacks:

**How to Prevent**

The only safe architectural pattern is not to accept serialized objects from untrusted sources or to use serialization mediums that only permit primitive data types.
Is the Application Vulnerable?
Applications and APIs will be vulnerable if they deserialize hostile or tampered objects supplied by an attacker. This can result in two primary types of attacks:

- Object and data structure related attacks where the attacker modifies application logic or achieves arbitrary remote code execution if there are classes available to the application that can change behavior during or after deserialization.
- Typical data tampering attacks, such as access-control-related attacks, where existing data structures are used but the content is changed.

Serialization may be used in applications for:
- Remote- and inter-process communication (RPC/IPC)
- Wire protocols, web services, message brokers
- Caching/Persistence
- Databases, cache servers, file systems
- HTTP cookies, HTML form parameters, API authentication tokens

How to Prevent
The only safe architectural pattern is not to accept serialized objects from untrusted sources or to use serialization mediums that only permit primitive data types.

If that is not possible, consider one of more of the following:

- Implementing integrity checks such as digital signatures on any serialized objects to prevent hostile object creation or data tampering.
- Enforcing strict type constraints during deserialization before object creation as the code typically expects a definable set of classes. Bypasses to this technique have been demonstrated, so reliance solely on this is not advisable.
- Isolating and running code that deserializes in low privilege environments when possible.
- Logging deserialization exceptions and failures, such as where the incoming type is not the expected type, or the deserialization throws exceptions.
- Restricting or monitoring incoming and outgoing network connectivity from containers or servers that deserialize.
- Monitoring deserialization, alerting if a user deserializes constantly.

Example Attack Scenarios
Scenario #1: A React application calls a set of Spring Boot services on the server, which deserialize the input object. If the input object contains malicious code, it can modify application logic or even achieve arbitrary code execution.

References
OWASP
Example Attack Scenarios

Scenario #1: A React application calls a set of Spring Boot microservices. Being functional programmers, they tried to ensure that their code is immutable. The solution they came up with 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.

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:

```php
array
    "Mallory",
    "user",
    "b6a8b3bea87fe0e05022f8f3c88bc960",

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

```php
array
    "Alice",
    "admin",
    "b6a8b3bea87fe0e05022f8f3c88bc960",

References

OWASP

- OWASP Cheat Sheet: Deserialization
- OWASP Proactive Controls: Validate All Inputs
- OWASP Application Security Verification Standard
- OWASP AppSecEU 2016: Surviving the Java Deserialization Apocalypse
- OWASP AppSecUSA 2017: Friday the 13th JSON Attacks

External

- CWE-502: Deserialization of Untrusted Data
- Java Unmarshaller Security
- OWASP AppSec Cali 2015: Marshalling Pickles
A1: 2017 Injections

- SQL injection
- NoSQL injection
- OS Command Injection
- LDAP Injection
- <insert injection here>
A2:2017 Insecure Authentication

- NIST 800-63 Alignment
- Two factor authentication
- Anti-automation
- Credential Stuffing
- Brute forcing and dictionary attacks
A3:2017 Sensitive Data Exposure

- Focuses on data breaches
- Sensitive, private, health, financial data
- Aligned with GDPR and privacy laws
• NEW FOR 2017!

• One of the most under-tested issues

• ... but only new issue that had sufficient data

• Learn, test, report, fix!
A5: 2017 Broken Access Control

- Insecure Direct Object Reference (IDOR)
- Force browsing
- Presentation layer access control
- Controller ("function") layer access control
- Model layer access control
- Domain access control – business logic
A6:2017 Security Misconfiguration

- S3 buckets, MongoDB, etc.
- Directory listings
- All the passive findings
- Risk rate as per the sensitivity!
• All your old favorites!

• Now with extra focus on DOM XSS

• Reflected XSS

• Stored XSS

A7:2017 XSS
A8:2017 Insecure deserialization

• New for 2017 (Community)
• Deserialization discovered – and this section written - by Chris Frohoff and Gabe Lawrence (!!)

• Learn, Test, Report, Fix!
A9: 2017 Known vulnerable components

• It’s still here!

• Automatically report with CI/CD dependency checkers

• Warn if outdated, break if vulnerable

• Let’s see if we can retire this in 3 years!
A10:2017 Insufficient Logging and Monitoring

- NEW for 2017 (Survey)
- Average time to discover pwnage: > 190 days
- Usually reported by external third party
- Usually costs > $1m and often a lot more

- This is a missing or ineffective control.

- Testing for this is pretty straightforward – talk to your operations team or look in your SIEM

- Detected?
- Would action be taken?
- Would escalation have occurred?

- Minimize time to detect and respond
Time to upskill and continuously improve

- OWASP Top 10 2017 is different
  - Update skills
  - Update test plans
  - Update tools
  - Update scan policies

In particular, A3, A8 and A10 are very different. No tool can adequately capture all 10 risks.
Get ready for 2020

• Get ready for the next release!

• Look at the data we collected this time around

• It’s 2018 already. Start automating that data collection!

• Please consider donating your stats to OWASP!
Thank you!

And thanks to Andrew van der Stock who contributed the presentation material.