OWASP Top 10 - 2017
What’s inside?

Taras Ivashchenko, CENTR Jamboree 2018
$ whoami

- OWASP Russia chapter leader
- Mail.Ru Group product security team
- https://oxdef.info
OWASP

• The Open Web Application Security Project
• 501(c)(3) worldwide not-for-profit charitable organization and open community
• Our mission is to make software security visible, so that individuals and organizations are able to make informed decision
• https://www.owasp.org
OWASP Top 10 Project

- Simple and powerful awareness document for web application security
- The 10 most critical web application security risks
- Referenced in MITRE and PCI DSS
- [https://www.owasp.org/index.php/top10](https://www.owasp.org/index.php/top10)
From 2013 to 2017
<table>
<thead>
<tr>
<th>OWASP Top 10 - 2013</th>
<th>OWASP Top 10 - 2017</th>
</tr>
</thead>
<tbody>
<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</td>
<td>A4:2017-XML External Entities (XXE) [NEW]</td>
</tr>
<tr>
<td>A6 – Sensitive Data Exposure</td>
<td>A6:2017-Security Misconfiguration</td>
</tr>
<tr>
<td>A7 – Missing Function Level Access Contr</td>
<td>A7:2017-Cross-Site Scripting (XSS)</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>
</tbody>
</table>
What is inside?
A1:2017-Injection
Injection

- When untrusted data is sent to an interpreter as part of a command or query
- SQL, NoSQL, OS, LDAP, etc.
- The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization
Example

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

https://target.com/app/app/accountView?id=' 
or '1'='1
How to prevent

- Safe API, which avoids the use of the interpreter entirely or provides a parameterized interface
- Object Relational Mapping Tools (ORMs)
- Positive ("whitelist") **server-side** input validation
- Escape special characters using the specific escape syntax for that interpreter
A2:2017-Broken Authentication
Broken Authentication

- Application functions related to authentication and session management are often implemented incorrectly
- Allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to get into victim session
Your application is vulnerable if...

- Permits brute force or other automated attacks
- Permits default, weak, or well-known passwords, such as “Password1” or “admin/admin”
- Uses plain text, encrypted, or weakly hashed passwords
- Exposes Session IDs in the URL
- Does not properly invalidate Session IDs, etc.
How to prevent

- Implement multi-factor authentication
- Implement weak-password checks
- Do not ship or deploy with any default credentials
- Ensure registration, credential recovery, and API pathways are hardened against account enumeration attacks
- Use a server-side, secure, built-in session manager that generates a new random session ID with high entropy after login. Session IDs should not be in the URL
A3:2017-Sensitive Data Exposure
Sensitive Data Exposure

• Many web applications and APIs do not properly protect sensitive data: credit cards, healthcare and other personal data

• The most common flaw is simply not encrypting sensitive data
Example

The password database uses unsalted or simple hashes to store user's passwords. And there is an SQL injection...

Attacker uses rainbow tables of pre-calculated hashes to crack the unsalted hashes and get the passwords
How to prevent

- Classify data processed, stored or transmitted by an application and apply controls as per the classification.
- Don't store sensitive data unnecessarily!
- Make sure to encrypt all sensitive data at rest.
- Encrypt all data in transit with **secure** protocols.
- Store passwords using strong adaptive and salted hashing functions with a work factor.
A4:2017-XML External Entities (XXE)
XXE

- Many older or poorly configured XML processors evaluate external entity references within XML documents.
- External entities can be used to disclose internal files, internal port scanning, remote code execution, and denial of service attacks.
Example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]]>
<foo>&xxe;</foo>
```
How to prevent

• Whenever possible, use less complex data formats such as JSON

• Disable XML external entity and DTD processing in all XML parsers in the application

• Patch or upgrade all XML processors and libraries in use by the application or on the underlying operating system
A5:2017-Broken Access Control
Broken Access Control

• Restrictions on what authenticated users are allowed to do are often not properly enforced

• Attackers can exploit these flaws to access unauthorized functionality and/or sensitive data of other users

• Bypassing access control checks by modifying the URL

• Metadata manipulation, such as replaying or tampering with a cookie or hidden field manipulated to elevate privileges
Example

```java
pstmt.setString(1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery();
```

http://example.com/app/accountInfo?acct=notmyacct
How to prevent

- Access control is only effective if enforced in trusted server-side code or server-less API
- With the exception of public resources, deny by default
- Implement access control mechanisms once and re-use them throughout the application
- Model access controls should enforce record ownership, rather than accepting that the user can create, read, update, or delete any record
A6:2017-Security Misconfiguration
Security Misconfiguration

- Unpatched flaws in legacy systems
- Insecure default configurations
- Incomplete or ad hoc configurations
- Open cloud storage
- Misconfigured HTTP headers
- Verbose error messages containing sensitive information
Example

The application server comes with enabled sample applications into production server.

These sample applications have known security flaws attackers use to compromise the server.
How to prevent

• A repeatable (automated) hardening process that makes it fast and easy to deploy another environment that is properly locked down

• A minimal platform without any unnecessary features, components, documentation, and samples

• A segmented application architecture

• An automated process to verify the effectiveness of the configurations and settings in all environments
A7:2017-Cross-Site Scripting (XSS)
XSS

• An application includes untrusted data in a HTTP response without proper validation or escaping

• ...Or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript

• Allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites
Example

(String) page += "<input name='search' type='text' value='"; (String) page += request.getParameter("search") + "'>";

'\\<script>do_evil_things()</script>\'
How to prevent

• Frameworks that automatically escape XSS by design
• Escaping untrusted HTTP request data based on the context in the HTML output (body, attribute, JavaScript, CSS, or URL)
• Applying context-sensitive encoding when modifying the browser document on the client side
• Content Security Policy
A8:2017-Insecure Deserialization
Insecure Deserialization

• Applications and APIs will be vulnerable if they deserialize hostile or tampered objects supplied by an attacker

• Insecure deserialization *often* leads to remote code execution

• They can be also used to perform replay attacks, injection attacks and privilege escalation attacks
Example

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

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

- Do not accept serialized objects from untrusted sources
- Use serialization mediums that only permit primitive data types
- Implement integrity checks (digital signatures) on any serialized objects
- Enforce strict type constraints during deserialization before object creation
A9:2017-Using Components with Known Vulnerabilities
Using Components with Known Vulnerabilities

- Libraries, frameworks, and other software modules, run with the same privileges as the application
- If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover
- Security flaws in 3rd party components are security flaws in your application
Example

CVE-2017-5638, a Apache Struts 2 remote code execution
How to prevent

- Remove unused dependencies, unnecessary features, components, files, etc.
- **Continuously** inventory the versions of both client-side and server-side components (e.g. frameworks, libraries) and their dependencies
- **Continuously** monitor sources like CVE and NVD for vulnerabilities in the components
- Only obtain components from official sources over secure links
- Monitor for libraries and components that are unmaintained or do not create security patches for older versions
A10:2017-Insufficient Logging & Monitoring
Insufficient Logging & Monitoring

• Most breach studies show time to detect a breach is over 200 days

• Typically detected by external parties rather than internal processes or monitoring

• Exploitation of insufficient logging and monitoring is the bedrock of nearly every major incident
How to prevent

- Log all critical operation failures with sufficient user context
- Use format that can be easily consumed by a centralized log management solutions
- Ensure high-value transactions have an audit trail with integrity controls to prevent tampering or deletion
- Establish effective monitoring and alerting
- Establish or adopt an incident response and recovery plan
- Build security operation center
Don’t know how to start your security awareness program? Start it with OWASP Top 10!
Join us to stay in touch!

- https://www.owasp.org/index.php/Russia
- https://www.meetup.com/OWASP-Russia/
- @owasp_ru
Thank you!