Web Application Vulnerabilities: OWASP Top 10 Revisited

Igino Corona
igino.corona AT diee.unica.it

Computer Security April 5th, 2018

What is a web application?

Distributed Program
- across client and server systems
- so many interpreters involved
- data exchanged through the HTTP(s) protocol

Each interpreter adds functionalities but also attack surface and complexity

Flash Silverlight ... PDF Reader
JavaScript
Images CSS HTML
HTTP(S) Client

Application Database

http://pralab.diee.unica.it
Open Web Application Security Project

International non-profit project to make web applications (web services) more secure
- i.e., towards confidentiality, integrity, availability of systems and data

Independent, reputable source

Key goals:
- Awareness: knowledge of the major/common threats
- Testing: methodologies and tools to detect known vulnerabilities
- Training: how to address known vulnerabilities

http://pralab.diee.unica.it

OWASP Top 10 Project

One of the most important outputs of OWASP is the TOP 10 Project

The OWASP Top 10 is an awareness document that focuses on the (ten) most serious threats for web applications
- “based primarily on data submissions from firms that specialize in application security and an industry survey that was completed by individuals”

New version each 3 years

It may be considered as a starting point to identify the threats for your web applications
- First step of the risk evaluation

The latest version has been produced in 2017

http://pralab.diee.unica.it
TOP 10 History


<table>
<thead>
<tr>
<th>OWASP Top Ten Entries (Unordered)</th>
<th>2003</th>
<th>2004</th>
<th>2007</th>
<th>2010</th>
<th>2013</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unvalidated Input</td>
<td>A1</td>
<td>A1</td>
<td>*</td>
<td>*</td>
<td>A4</td>
</tr>
<tr>
<td>Buffer Overflows</td>
<td>A5</td>
<td>A5</td>
<td>*</td>
<td>*</td>
<td>A7</td>
</tr>
<tr>
<td>Denial of Service</td>
<td>A9</td>
<td>A9</td>
<td>*</td>
<td>*</td>
<td>A2</td>
</tr>
<tr>
<td>Injection</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
<td>A5</td>
<td>A10</td>
</tr>
<tr>
<td>Cross Site Scripting (XSS)</td>
<td>A3</td>
<td>A3</td>
<td>A2</td>
<td>A7</td>
<td>A3</td>
</tr>
<tr>
<td>Broken Authentication and Session Management</td>
<td>A7</td>
<td>A7</td>
<td>A7</td>
<td>A7</td>
<td>A7</td>
</tr>
<tr>
<td>Insecure Direct Object Reference</td>
<td>A4</td>
<td>A4</td>
<td>A4</td>
<td>A4</td>
<td>A4</td>
</tr>
<tr>
<td>Cross Site Request Forgery (CSRF)</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
</tr>
<tr>
<td>Security Misconfiguration</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
</tr>
<tr>
<td>Missing Functional Level Access Control</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
</tr>
<tr>
<td>Unvalidated Redirects and Forwards</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
</tr>
<tr>
<td>Information Leakage and Improper Error Handling</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
</tr>
<tr>
<td>Malicious File Execution</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
</tr>
<tr>
<td>Sensitive Data Exposure</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
</tr>
<tr>
<td>Insecure Communications</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
</tr>
<tr>
<td>Remote Administration Flaws</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
</tr>
<tr>
<td>Using Known Vulnerable Components</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
<td>A2</td>
</tr>
</tbody>
</table>

Prepared by: christian.heisig@owasp.org

http://pralab.diee.unica.it

TOP 10 - 2017 vs 2013

• **New categories**
  – A4:2017 - XML External Entities (XXE)
  – A8:2017 - Insecure Deserialization
  – A10:2017 - Insufficient Logging & Monitoring

• **Merged**
  – A5:2017 - Broken Access Control
    • A4:2013 - Insecure Direct Object References
    • A7:2013 - Missing Function Level Access Control

• **Retired**
  – A8:2013 - Cross-Site Request Forgery (CSRF)
  – A10:2013 - Unvalidated Redirects and Forwards

In the following we present all TOP 10 - 2017 threats

- Examples and mitigation mechanisms as described in the official OWASP website
- We also highlight the main points for the mitigation of issues/threat vectors

**OWASP TOP 10**

**A1:2017 Injection**

Injection: the quote (') is a meta-character: it is internally used to separate data from (SQL) instructions

Example Attack Scenarios

**Scenario #1:** An application uses untrusted data in the construction of the following **vulnerable** SQL call:

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

**Scenario #2:** Similarly, an application's blind trust in frameworks may result in queries that are still vulnerable, (e.g. Hibernate Query Language (HQL)):

```java
Query HQLQuery = "custID='" + id + "'";
```

In both cases, the attacker modifies the 'id' parameter value in their browser to send: ' or '1'=1. For example:

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

This changes the meaning of both queries to return all the records from the accounts table. More dangerous attacks could modify or delete data, or even invoke stored procedures.
A1:2017 Injection

How to Prevent

Preventing injection requires keeping data separate from commands and queries.

- The preferred option is to use a safe API, which avoids the use of the interpreter entirely or provides a parameterized interface, or migrate to use Object Relational Mapping Tools (ORMs).

Note: Even when parameterized input is accepted, the use of PL/SQL or T-SQL commands like EXEC IMMEDIATE or executing external commands must be carefully avoided.

- Use positive or 'white list' input validation, and limit the input data sources
- Many applications require special characters, such as text areas or APIs for mobile applications.
- For any residual dynamic queries, escape special characters using the specific escape syntax for that interpreter.

Note: SQL structure such as table names, column names, and so on cannot be escaped, and thus user-supplied structure names are dangerous. This is a common issue in report-writing software.

- Use LIMIT and other SQL controls within queries to prevent mass disclosure of records in case of SQL injection.

Input data must be assumed as untrusted. A safe API must always keep data separate from code, and take into account the specifics of the involved interpreter(s) - meta-characters.

A2:2017 – Broken Authentication

Example Attack Scenarios

Scenario #1: Credential stuffing, the use of lists of known passwords is a common attack. If an application does not implement automated threat or credential stuffing protections, the application can be used to determine if the credentials are valid.

Scenario #2: Password Achilles’ heel – brute force attacks involve the use of lists of passwords and are viewed as a way to break into a system and reuse, weak passwords. Organizations are recommended to stop these practices per NIST 800-63 and use multi-factor authentication.

Scenario #3: Application session timeouts aren't set properly. A user uses a public computer to access an application. Instead of selecting “logout” the user simply closes the browser tab and walks away. An attacker uses the same browser an hour later, and the user is still authenticated.

Brute force attack – password guessing

Weak authentication credentials – e.g. password easy to guess and reused

Session identifiers effectively substitute authentication credentials during login. They may be stolen and used by an attacker.
A2:2017 – Broken Authentication

Strong authentication credentials: default, multi-factor, complexity, expiration, recovery

Protection against user impersonation attacks – e.g., due to credentials theft or CSRF

Protection against information gathering (e.g., account enumeration)

Logging

Brute force attack protection

Key point (not highlighted in this slide): Unique, Safe, Server-side Authentication API triggered for any request

Example Attack Scenarios

Scenario #1: An application encrypts credit card numbers in a database using automatic database encryption. However, this data is automatically decrypted when retrieved, allowing an SQL injection flaw to retrieve credit card numbers in clear text.

Scenario #2: A site doesn’t use or enforce TLS for all pages of the site. The attacker can access network traffic (e.g. at an insecure wireless network), downgrades connections from HTTPS to HTTP, intercepts requests, and steals the user’s session cookie. The attacker then replays this cookie and hijacks the user’s (authenticated) session accessing or modifying the user’s private data. Instead of the application transported data, e.g. the recipient of a money transfer.

Scenario #3: The password database uses unsalted or simple hashes to store everyone’s passwords. A file upload flaw allows an attacker to retrieve the password database. All the unsalted hashes can be exposed with a rainbow table of pre-calculated hashes. Hashes generated by simple or fast hash functions may be cracked by GPUs, even if they were salted.
A3:2017 – Sensitive Data Exposure

- Classify data processed, stored or transmitted by an application. Identify which data is sensitive according to privacy laws, regulatory requirements, or business needs.
- Apply controls as per the classification.
- Don’t store sensitive data unnecessarily. Discard it as soon as possible or use PCI DSS compliant tokenization or even truncation. Data that is not retained cannot be stolen.
- Make sure to encrypt all sensitive data at rest.
- Ensure up-to-date and strong standard algorithms, protocols, and keys are in place; use proper key management.
- Encrypt all data in transit with secure protocols such as TLS with pre-shared keys, cipher prioritization by the server, and secure protocols, which require the use of SSL/TLS to ensure data integrity.
- Use directories like HTTP Strict Transport Security (HSTS).
- Disable caching for response that contain sensitive data.
- Store passwords using strong adaptive and salted hashing functions with a work factor (delay factor), such as Argon2, scrypt, bcrypt or PBKDF2.
- Verify independently the effectiveness of configuration and settings.

http://pralab.diee.unica.it

A4:2017 - XML External Entities (XXE)

Custom Document Type Definition (DTD)

Scenario #1: The attacker attempts to extract data from the server by changing the DTD:

```
<!DOCTYPE foo [<ENTITY xxe SYSTEM "https://192.168.1.1/private" >]> 
<foo>&xxe;</foo>
```

Scenario #2: An attacker probes the server's private network by changing the line to:

```
<ENTITY xxe SYSTEM "https://192.168.1.1/private" >]
```

Scenario #3: An attacker attempts a denial-of-service attack by including a potentially endless file:

```
<ENTITY xxe SYSTEM "file:///dev/random" >]
```

DTD has an entity provided in another file

The entity is printed out: content of /etc/passwd file
Other XML DTD attack (Billion Laughs)

```
<?xml version="1.0"?>
<!DOCTYPE lolz [ 
<!ENTITY lol "lol"> 
<!ELEMENT lolz (#PCDATA)>
<!ENTITY lol1 &lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
<!ENTITY lol2 &lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
<!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 lol15 &lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
<!ENTITY lol16 &lol15;&lol15;&lol15;&lol15;&lol15;&lol15;&lol15;&lol15;&lol15;&lol15;">
<!ENTITY lol17 &lol16;&lol16;&lol16;&lol16;&lol16;&lol16;&lol16;&lol16;&lol16;&lol16;">
<!ENTITY lol18 &lol17;&lol17;&lol17;&lol17;&lol17;&lol17;&lol17;&lol17;&lol17;&lol17;">
<!ENTITY lol19 &lol18;&lol18;&lol18;&lol18;&lol18;&lol18;&lol18;&lol18;&lol18;&lol18;&lol18;">
]>
<lolz>&lol19;</lolz>
```

https://en.wikipedia.org/wiki/Billion_laughs_attack

A4:2017 - XML External Entities (XXE)

- Input data must be assumed as untrusted. A safe API must always keep data separate from code, and take into account the specifics of the involved interpreter(s).
- Use a less powerful API!

- Whenever possible, use less complex data formats such as JSON and avoiding serialization of sensitive data.
- Patch or upgrade all XML processors and libraries in use by the application.
- Use dependency checkers and make sure to keep your operating system, libraries, and tools up-to-date.
- Implement positive (whitelisting) server-side input validation, filtering, or sanitization to prevent hostile data within XML documents, headers, or nodes.
- Verify that XML or XSL file upload functionality validates incoming XML using XSD validation or similar.
- SAST tools can help detect XXE in source code, although manual code review is the best alternative in large, complex applications with many integrations.
- If these controls are not possible, consider using virtual patching, API security gateways, or Web Application Firewalls (WAFs) to detect, monitor, and block XXE attacks.

Go to main table
In both cases, **data access** depends on **unauthenticated data**.

Default: deny

Unique, Safe, Server-side Access Control API triggered for *any* request

- With the exception of public resources, access control should be enforced by domain models.
- Implement access control mechanisms once and re-use them throughout the system, including minimizing CORS usage.
- Model access controls should enforce record ownership, rather than just the capability to view records. A user can create, read, update, or delete any record.
- Unique application business limit requirements should be enforced by domain models.
- Disable web server directory listing and ensure file metadata (e.g., .git) and backup files are not present within web roots.
- Log access control failures, alert admins when appropriate (e.g., repeated failures).
- Rate limit API and controller access to minimize the harm from automated attack tooling.
- JWT tokens should be invalidated on the server after logout.

Developers and QA staff should include functional access control unit and integration tests.

If possible do not use Cross-Origin Resource Sharing (CORS) - a relaxed version of Same-origin policy

**Logging**
A6:2017 – Security Misconfiguration

Example Attack Scenarios

Scenario #1: The application server comes with sample applications that are not removed from the production server. These sample applications have known security flaws attackers use to compromise the system. In one specific case, the admin console, and default accounts weren’t changed the attacker logs in with.

Scenario #2: Directory listing is not disabled on the server. An attacker discovers sensitive files and downloads them. The attacker finds and downloads the compiled Java classes, which they decompile and reverse engineer to view the code. The attacker then finds a serious access control flaw in the application.

Scenario #3: The application server’s configuration allows detailed error messages, e.g. stack traces, to be returned to users. This potentially exposes sensitive information or underlying flaws such as component versions that are known to be vulnerable.

Scenario #4: A cloud service provider has default sharing permissions open to the Internet by other CSP users. This allows sensitive data stored within cloud storage to be accessed.

A6:2017 – Security Misconfiguration

Systematic Hardening Process

Security by default

Minimal functionalities (less powerful APIs)

Security-oriented Architecture

How to Harden

Secure installation processes should be implemented, including:

- 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, with different credentials used in each environment. This process should be automated to minimize the effort required to setup a new secure environment.
- A minimal platform without any unnecessary features, components, documentation, and samples. Remove or do not install unused features and frameworks.
- A task to review and update the configurations appropriate to all security notes, updates a management process (see A9:2017-Using Components with Known Vulnerabilities). Permissions (e.g. S3 bucket permissions).
- A segmented application architecture that provides effective, secure separation between components or tenants, with segmentation, containerization, or cloud security groups (ACLS).
- Sending security directives to clients, e.g. Security Headers.
- An automated process to verify the effectiveness of the configurations and settings in all environments.
Injection: different meta-characters (e.g., < >) used by the browser to separate data from (HTML/Javascript) instructions

Input data must be assumed as untrusted. A safe API must always keep data separate from code, and take into account the specifics of the involved interpreter(s) - meta-characters

Be aware and enforce document encoding!

Browser mitigation: allow script code only from specific domain names/paths (or completely disallow it)
Input data is parsed to dynamically generate code (data structure/classes): e.g. java and php objects

Well.. Actually this is an access control issue not a serialization one... wrong example

Do not use serialization. In general, it is a too powerful API. Keep data separate from code (natively)
Web applications are more and more complex. They may use hundreds of different libraries and third-party components. A vulnerability in any component may be automatically found and exploited. E.g. struts, other (trusted) devices.

There are automated tools to help attackers find unpatched or misconfigured systems. For example, the Shodan IoT search engine can help you find devices that still suffer from Heartbleed vulnerability that was patched in April 2014.

Dependency/vulnerability checks must be automatized and become integral part of the software development life cycle.

- Remove unused dependencies, unnecessary features, components, files, and documentation.
- Continuously inventory the versions of both client-side and server-side components (e.g. frameworks, libraries) and their dependencies using tools like versions, DependencyCheck, retire.js, etc. Continuously monitor sources like CVE and NVD for vulnerabilities in the components and use analysis tools to automate the process. Subscribe to security lists and discussion forums related to components you use.
- Only obtain components from official sources over secure links. Prefer signed packages to reduce the chance of including a modified, malicious component.
- Monitor for libraries and components that are unmaintained, abandoned, or that are no longer supported. Use patches for older versions. If patching is not possible, consider monitoring, detecting, or protecting against the discovered issue.

Every organization must ensure that there is an ongoing plan for monitoring, triaging, and applying updates or configuration changes for the lifetime of the application or portfolio.

Remove not used components

Always verify the source of components

Use actively mantained components
**A10:2017 - Insufficient Logging&Monitoring**

**Example Attack Scenarios**

**Scenario #1:** An open source project forum software run by a small team was hacked using a flaw in its software. The attackers managed to wipe out the internal source code repository containing the next version, and all of the forum contents. Although source could be recovered, the lack of monitoring, logging or alerting led to a far worse breach. The forum software project is no longer active as a result of this issue.

**Scenario #2:** An attacker uses scans for users using a common password. They can take over all accounts using this password. For all other users, this scan leaves only one false login behind. After some days, this may be repeated with a different password.

**Scenario #3:** A major US retailer reportedly had an internal malware analysis sandbox analyzing attachments. The sandbox software had detected potentially unwanted software, but no one responded to this detection. The sandbox had been producing warnings for some time before the breach was detected due to fraudulent card transactions by an external bank.

**How to Prevent**

- Ensure that logs are generated in a format that can be easily consumed by a centralized log management solutions.
- Ensure high-value transactions have an audit trail with integrity, tampering or deletion, such as append-only database tables or similar.
- Establish effective monitoring and alerting such that suspicious activity is suspected and responded to in a timely manner.
- Establish or adopt an incident response plan, such as NIST 800-61 rev 2 or later.

There are commercial and open source application protection frameworks such as OWASP AppSensor, web application firewalls such as ModSecurity with the OWASP ModSecurity Core Rule Set, and log correlation software with custom dashboards and alerting.

- **Insufficient logging/monitoring leads to worse breaches (higher attack impact)**
- **Logging relevant context allows for reducing the impact of security issues (response/remediation)**
- **Humans must be in the loop**
- **Ensure Log integrity (append only)**

Logging and monitoring is just part of an Incident Response Plan.
• **Issues**
  – The OWASP TOP 10 is very useful, but somewhat confusing

• **In the following we try to group TOP 10 entries according to their**
  – *root causes*
  – *security violations*

• **This allows to get a high-level view of common problems**
  – foresee and protect against attacks with the *same root causes*
    • New threats
    • Threats more relevant to your web applications

---

**Information System Security**

**Main Security Goals**

- **Confidentiality**
  - ensure that (sensitive) information is disclosed to *authorized* parties only

- **Integrity**
  - prevent *unauthorized* modification of data (*data integrity*), including system code and (ab)use of system functionalities (*system integrity*)

- **Availability**
  - guarantee that data and services can be accessed (in a reasonable time) by *authorized* parties when requested

---

**key aspects:**

- authentication
- access control

---

**NOTE:** Violations in one category may enable violations in any other category!

Examples:

- Password theft (confidentiality violation) may allow attackers to perform unauthorized modifications of user data (data integrity violation)
- A buffer overflow attack (system integrity violation) may allow attackers to gather private data (confidentiality violation)
Information System Security

Meta-Security Goals
mitigate the impact/risk of security violations

- **Monitoring**
  - keep track of security-relevant events, such as authentications, accesses, data modifications, system performance, detected attacks/errors/anomalies.

- **Response**
  - *counteract* against detected security violations and *remediate* (incident response plan)

*Remember*: Security is a risk management process and Humans are always in the loop!

Information System Security - Objectives

- INTEGRITY
- CONFIDENTIALITY
- AVAILABILITY
- AUTHENTICATION
- ACCESS CONTROL

**TOP 10 Classification**
## TOP 10 Threats and Security Violations

<table>
<thead>
<tr>
<th>TOP 10 Threat</th>
<th>Security Violation</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1:2017 – Injection</td>
<td></td>
</tr>
<tr>
<td>A2:2017 - Broken Authentication</td>
<td></td>
</tr>
<tr>
<td>A3:2017 – Sensitive Data Exposure</td>
<td></td>
</tr>
<tr>
<td>A4:2017 - XML External Entities (XXE)</td>
<td></td>
</tr>
<tr>
<td>A5:2017 - Broken Access Control</td>
<td></td>
</tr>
<tr>
<td>A6:2017 - Security Misconfiguration</td>
<td></td>
</tr>
<tr>
<td>A7:2017 - Cross-Site Scripting (XSS)</td>
<td></td>
</tr>
<tr>
<td>A8:2017 - Insecure Deserialization</td>
<td></td>
</tr>
<tr>
<td>A9:2017 - Using Components with Known Vulnerabilities</td>
<td></td>
</tr>
<tr>
<td>A10:2017 - Insufficient Logging &amp; Monitoring</td>
<td></td>
</tr>
</tbody>
</table>

## OWASP TOP 10 – Integrity Violations

**THREAT VECTORS**
- Data → Instruction
- A1:2017 – Injection
- A4:2017 - XML External Entities (XXE)
- A7:2017 - Cross-Site Scripting (XSS)
- A8:2017 - Insecure Deserialization
- A10:2013 - Unvalidated Redirects and Forwards
- A6:2010 - Malicious File Execution
- A5:2004 - Buffer Overflows

**RESPONSE**

**CONCEPTUAL PAGE**
- Authentication
- Access Control

**CONTENT PAGE**
- Integrity
- Confidentiality
- Availability

**NAVIGATION**
- Back to Security Objectives
• **Unauthorized modification of data and (ab)use of system functionalities**
  – Manipulating data, the attacker can exploit unexpected system functionalities or abuse expected ones
  – The attacker (ab)uses system functionalities. Unexpected functionalities are often raised by exploiting *interpreter* vulnerabilities, so that
    • input *data* is *erroneously* *interpreted as code*
    • data and code are carried in the *same channel!*
  – In this category fall the following TOP 10 attacks:
    • A1:2017 – Injection
    • A4:2017 - XML External Entities (XXE)
    • A7:2017 - Cross-Site Scripting (XSS)
    • A8:2017 - Insecure Deserialization
    • A10:2013 - Unvalidated Redirects and Forwards
    • A6:2010 - Malicious File Execution
    • A5:2004 - Buffer Overflows

---

**TOP 10 OWASP - Integrity Violations**

**Input Data is wrongly/arbitrarily interpreted as**

– Database Query (typical), OS command, LDAP, others
  • A1:2017 Injection

– **XML instruction**
  • A4:2017 - XML External Entities (XXE)

– **JavaScript instruction**
  • A7:2017 Cross-site Scripting (XSS)

– **Ad-hoc interpreter**
  • A8:2017 Insecure Deserialization

– **Browser instruction**
  • A10:2013 Unvalidated Redirects and Forwards

– **Assembly instruction**
  • A5:2004 Buffer Overflows

– **Web Application Interpreter**
  • A6:2010 Malicious File Execution

---

Preventing data→code attacks requires keeping data separate from instructions (safe API)
Interpreters

- *widespread* across client and server systems
- may be really *complex*
- may *interact to each other*

**Each interpreter adds attack surface and complexity**

From Data to Code - Integrity Violations

**OWASP TOP 10 – Generic Threat Vectors**

Generic Threat Vectors
A9:2017 - Using Components with Known Vulnerabilities
A6:2017 - Security Misconfiguration
OWASP TOP 10 – Authentication

- A8:2013-Cross-site Request Forgeries
- A2:2017-Broken Authentication

OWASP TOP 10 – Confidentiality

- A3:2017 – Sensitive Data Exposure
OWASP TOP 10 – Access Control

A5: 2017 - Broken Access Control

MONITORING

CONFIDENTIALITY

INTEGRITY

AVAILABILITY

AUTHENTICATION

ACCESS CONTROL

RESPONSE

--

OWASP TOP 10 – Availability

A9: 2004 - Denial Of Service

MONITORING

CONFIDENTIALITY

INTEGRITY

AVAILABILITY

AUTHENTICATION

ACCESS CONTROL

RESPONSE

--

http://pralab.diee.unica.it
In the next lessons

- We will dig into some of the most popular and threatening web attack techniques
  - Most of them (but not all) are in the TOP10 OWASP 2017
  - Practical sessions with
    - Real-world web services
    - OWASP BWA
      1. Download Virtualbox [https://www.virtualbox.org/](https://www.virtualbox.org/)
      2. Download the OVA archive [https://sourceforge.net/projects/owaspbwa/files/1.2/](https://sourceforge.net/projects/owaspbwa/files/1.2/)
      3. Import the OVA archive into VirtualBox