Why Do We Need To Secure Web Applications?
Companies use web applications for every factor of their business operations from the public websites to mission-serious business applications. With the increase in usability and vulnerabilities, web applications now become the target for the attackers to gain access. The followings are the reason why we need to secure web applications:

- Easy target, directly exposed Public Interface
- Larger Attack Surface
- Addition of Complex Design & Features, increase chances of Attacks
- Root Cause of above 80% of Security Attacks are directly or indirectly related to the Web

Why Web Security?
With a web application threats are becoming more frequent occurrence; several organizations are struggling to implement security on their web application programs; since they unaware what to do. Understanding that “Security is a process, not a product” could be an ideal solution for their searches.

Web Security Testing: Current Limitations
The followings are the reasons that stands behind the failure of web security testing:

- Organizations focus on Testing Business Functionality & Capability on Deployed Applications
- Security Testing is the last thing to consider, usually after Functional Testing
- Current trend shows an exponential increase in vulnerability list related to Web Applications

Introducing OWASP
Open Web Application Security Project (OWASP) is an International Non-Profit Charitable Open Source organization. Its participation is free and open to all. It is a technology agnostic and contributed selflessly to the security community. OWASP address Risk-based approach.

OWASP Top 10: Web Application Security Risk
This document includes the list of the 10 most web security risk in the web application. The errors listed occur most frequently in the web application and they’re dangerous since they’ll allow the hackers completely control the software and steal data. The primary aim of the list is to educate developers, designers, architects and organizations about consequences of most common web application security vulnerabilities.

OWASP Risk Rating Methodology
OWASP uses its Risk rating methodology, to analyse severity of these Risk, based on their impact, and prevalence. Let us begin with, Standard risk model:

\[ \text{Risk} = \text{Likelihood} \times \text{Impact} \]

OWASP has proposed the systematic, Risk Rating Methodology, assisting organizations to effectively analyse and manage the corresponding Web Security Risk.
Steps Involved

Step#1: Identify A Risk
Identify Security Risk that needs to be rated. It gathers information about the following aspects:
- Involved Threat Agents
- Attack used
- Vulnerability involved
- Business Impact

Step#2: Factors for Estimating Likelihood
The main goal is to estimate the likelihood of a successful attack.
- Threat agent: Estimate the likelihood of a successful attack by the group of threat agents
- Vulnerability agent: Estimate the likelihood of the certain vulnerability involved being revealed & exploited.

Step#3: Factors For Estimating Impact
Focus on the impact of the successful attack. It concentrates on the technical impact and business impact.

Step#4: Determining Severity Of Risk
Proceeding with the following steps:
- Find Likelihood & Impact based on Score
- Determine Severity

Step#5: Deciding What To Fix
Once the risks are classified, then prioritize list to determine what to fix.

Step#6: Customizing Your Risk Rating Model
Three ways to customize the model are:
- Adding Factors
- Customizing Options
- Weighting Factors
OWASP Top 10: How Each Risk Is Analysed

The table illustrates how each risk is analysed in the OWASP Top 10 document:

<table>
<thead>
<tr>
<th>Threat Agents</th>
<th>Exploitability</th>
<th>Weakness Prevalence</th>
<th>Weakness Detectability</th>
<th>Technical Impacts</th>
<th>Business Impacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>App Specific</td>
<td>EASY (3)</td>
<td>WIDESPREAD (3)</td>
<td>EASY (3)</td>
<td>SEvere (3)</td>
<td>App / Business Specific</td>
</tr>
<tr>
<td></td>
<td>AVERAGE (2)</td>
<td>COMMON (2)</td>
<td>AVERAGE (2)</td>
<td>MODERATE (2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DIFFICULT (1)</td>
<td>UNCOMMON (1)</td>
<td>DIFFICULT (1)</td>
<td>MINOR (1)</td>
<td></td>
</tr>
</tbody>
</table>


The threat environment for the API and web application continually changes. To appear up-to-date, OWASP Top 10 periodically updates their list with the recent dangerous security vulnerabilities. Recently, it announced the release of OWASP Top 10 Critical Web Application Security Risks.

Here is the comparison of OWASP Top 10 - 2013 (Previous Version and OWASP Top 10 - 2017 (Current Version)

As shown in the above illustration:
- The vulnerabilities A4 – Insecure Direct Object Reference and A7 - Missing Function Level Access Control in the 2013 list are merged and listed as A4-Broken Access Control in the 2017 list.
- Moreover, in the OWAPS 2017, three new risks called A4:2017-XML External Entities (XXE) [NEW], A8:2017-Insecure Deserialization, and A10:2017-Insufficient Logging and Monitoring are added additionally.
- The risks A8 – Cross-site Request Forgery and A10 – Unvalidated Redirects and Forwards was found in the only minimum percentage of applications, both are dropped from the list of critical web application security risks.
A1-Injection

Injection flaws, including OS, SQL, and LDAP injection typically happen when the malicious input is sent to the interpreter as a piece of query or command.

**Root cause:** Injection attacks, usually occur, when interpreter, due to limitation in parsing queries, or related implementation flaw, is unable to detect difference between a command, and a query. Here, user can trick interpreter by pushing some logical sequences of characters, along with standard commands, or keywords as per parsing engine.

**Risk Analysis**

<table>
<thead>
<tr>
<th>Attack Vectors</th>
<th>Security Weaknesses</th>
<th>Impacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exploitability</td>
<td>Prevalence</td>
<td>Detectability</td>
</tr>
<tr>
<td>Easy (3)</td>
<td>Common (2)</td>
<td>Easy (3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Severe (3)</td>
</tr>
</tbody>
</table>

**Threat Agents:** “Anyone” who can send untrusted data to the system

**Exploitability:** Easy
- Any data source can be an attack vector. The attacker sends a simple text-based attacks, that can impact targeted interpreter.

**Security Weakness**
- Commonly, found in SQL, LDAP, Xpath, OS Commands, XML parsers etc
- Easy to discover from White box analysis.

**Technical Impacts:** Severe
- Data Loss/Corruption, DOS, Authorization & Accountability Concerns
- Complete System Takeover

**Example:**
Let us see how the SQL injection activity happens. The screenshots below depict the steps that attackers were taken to exploit the authentication flaws related to username:

![Example screenshot with user input](image-url)
The screen shows the generated SQL Statements & background process:

As a result, the attacker logged into the admin page:

How To Defend

- Whitelist input data. Allow only trusted data. Trust nobody without validating. Apply a standard input validation method to validate the input data for their type, length, business rules and syntax before allowing.
- Use strong parameterized SQL statements. It is best to separate the queries from the input data. Input un-trusted data as a parameter in the query, instead of passing complete statement as a query. Type casts each parameter for ensuring correct data types.
- Ensure proper database permissions. Follow the principle of least privilege while connecting to the database as well as other backend systems. Apply account segmentation.
- Avoid using simple escaping functions; since they are weak and can be easily exploitable.
- Use powerfully typed parameterized code APIs, even while calling stored procedures.
- Careful on canonicalization errors since these errors could be applied to bypass the whitelist schemes.
A2-Broken Authentication And Session Management

Weakly implemented Authentication mechanism, can be easily exploited by any user, who can send untrusted data to the system. These are commonly found due to improper implementation of Authentication related processes, like Logouts, Password Management, Timeouts etc. Broken authentication and Session management happens when credentials can’t be authenticated & session IDs can’t be started because of the absence of efficient encryption and weak session management. These flaws can result in the user or administrative account hijacking, undermine authorization & accountability controls, cause privacy violations and even risk the company network & systems.

Risk Analysis

<table>
<thead>
<tr>
<th>Attack Vectors</th>
<th>Security Weaknesses</th>
<th>Impacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exploitability</td>
<td>Prevalence</td>
<td>Detectability</td>
</tr>
<tr>
<td>Easy (3)</td>
<td>Common (2)</td>
<td>Moderate (2)</td>
</tr>
</tbody>
</table>

Threat Agents: “Anyone” who can send untrusted Data to the system

Exploitability: Easy
- With the automated brute force and dictionary-based attacks, attackers can now be able to access the targeted database easily. In addition, they are at ease with session management attacks, especially associated with tokens.

Security Weakness
- The occurrence of Broken authentication weakness is widespread. Session management weakness can be found in password management, timeouts, Logout, etc.
- It can be discoverable with both manual and automated methods.

Technical Impacts: Severe
- Privilege Escalation, Data Leakage, Account Hijacking

How Session Can Be Hijacked:
The sessions can be compromised in different ways. The three most common ways are:

1. **Authentication Cookie Theft**
   - **Exploit via XSS** – It involves the method of exploiting the XSS weakness and steal the authentication cookie from the target who opened the infected link. Attackers will use the stolen cookie to compromise the target account. In exploit via XSS method, the hacker sends a link with the malicious JavaScript to the victim to steal the cookies of his browser. As soon as the victim opens the link, the malicious JavaScript will start to execute the instructions crafted by the hacker.
   - **Sniff it over the network/insecure connection** – This method involves two steps to gain unauthorized access. First, the attackers capture the cookie exist over the network or insecure connection with the help of sniffer tools. Then, with the sniffed cookie or session ID, he can achieve unauthorized access even without having a password.
   - **Get it from a victim’s PC** - To achieve this the hacker prompt the victim’s computer to run a malicious Java script, which includes the action of sending the HTTP request (along with cookies attached) to the hacker.

2. **Stealing Session ID**
   - **Retrieve it from the URL** – There are some situations where the application fetches JS files or image from same domain by using HTTP. The issue is when sending a request to an already signed in the domain, the session ID would automatically go. Even if a single URL passes to the server without the HTTP, the session ID would append along with it. This gives the attackers great opportunity to steal the ID.
- **Get it from response header** – This attack occurs when the script embeds the session ID in the HTTP response header. In case the attacker receives the response instead of the corresponding user, they can gain the details of the session ID.
- **Collect it from logs** – Consider that an organization logs the session ID. An internal attacker of the organization can hijack the entire session by simply obtaining access to an application log file.

3. **Account Management Attack**
   - **Brute force/Dictionary attack** – Brute force/Dictionary attack generally involves a number of attempts to guess the required information such as username and password. Similarly, when it comes to stealing the session ID, once they understand the pattern of the session ID, they can hijack the account by predicting the session details of the logged users.
   - **Weak Password Policies** – Setting password based on weak password policies includes the risk of being overwritten or changed by the attackers. With their own password, the attackers can effortlessly steal the session ID to take the control of the user’s session.

**Example:**
Let us explore how Improper Cookie Management, results Into Privilege Escalation of a User.
If an attacker gets the cookie information as shown in the screen shot:

They can escalate the privilege by changing the uid value and send a request to the server.
How To Defend

- Enhance Cookie Protection. Use ‘HttpOnly’ cookie flag. Ensure to have ‘secure’ cookie flag set. Set the Path and Domain parameters for the cookies accurately. Avoid application code for manipulating the session cookies.
- Set session expiry time, if it is idle. The session should be expired automatically when it remains idle for a particular time period. This will support to limit the chances a hacker has to steal the session ID or perform unwanted actions.
- Ensure Strong Passwords. The password should include the restricted complexity. The complexity typically needs the combination of numeric, alphabetic or non-alphanumeric characters. Users should not be allowed to reuse the previous passwords. In addition, imply effective password policy.
- Ensure the session IDs remain secure by protecting the session ID tokens with SSL.
- Session IDs should be generated, controlled as well as secured centrally only by the authorization and authentication mechanism instead of under the control of either users or client.
- Session IDs shouldn’t be exposed in the URLs, which client machines are using to connect to the web server to avoid being copied by the attackers.

A3- Sensitive Data Exposure

This flaw happens when an application doesn’t adequately safeguard sensitive details such as session tokens, passwords, user’s private information, credit card data and more, from being exposed to illegitimate users. Usually this risk prevails along with other risk or may act as input for some other security attack. In order to exploit, attacker needs to be aware of security testing methods like crafting man in the middle attack, sniffing etc.

Risk Analysis

<table>
<thead>
<tr>
<th>Attack Vectors</th>
<th>Security Weaknesses</th>
<th>Impacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exploitability</td>
<td>Prevalence</td>
<td>Detectability</td>
</tr>
<tr>
<td>Moderate (2)</td>
<td>Widespread (3)</td>
<td>Moderate (2)</td>
</tr>
</tbody>
</table>

Threat Agents: Anyone who can have access to sensitive data or any backup of same.

Exploitability: Moderate
- Can’t be directly attacked and impacted, instead attacker uses other mechanism like Stealing Keys, MitM etc to assist data leakage

Security Weakness
- Can occur if sensitive data is not properly encrypted and managed
- Easy to detect when data is in transit, but hard when data at rest.

Technical Impacts: Severe
- Compromises entire sensitive data such as personal data, credit card details, and health records.

Examples

1. **Insufficient implementation of SSL/TLS**, which comes in following patterns
   - No ‘Secure’ flag for cookies
   - Login or sensitive data transaction is not on HTTPS channel
   - Using “Mixed mode”

2. **Bad Crypto usage and poor management**, which results from
   - Using Weak Algorithms
   - Poor protection of keys
   - Weak password storage mechanism

3. **Other ways**
   - Data leakage via logs
   - Cookie disclosure via URL
   - Browser auto complete
An Example Scenario:
Let us consider a scenario, where some sensitive information is leaked in the publicly available file, say robots dot txt file, deployed for Web crawlers to direct them about what to index and what not. Analysing this file reveals one of the sensitive location like that of password.

Now, if the server is not configured properly, we can directly browse through this location and get sensitive data.

Hence, sensitive data exposure can happen if we have not implemented proper security controls to sensitive places.
How To Defend

- Try to minimize sensitive data collection. It is better to reduce the window of storage. Discard sensitive data if they are no more required.
- Use HTTPS everywhere instead of HTTP
- Use strong crypto storage with proper access control and store passwords with a strong encryption or hash format.
- Turn-off auto-complete and caching of pages
- Classify all Data (processed, stored or transmitted) and ensure security controls according to Business and regulatory requirements

A4 - XML External Entities (XXE)

XXE is a certain type of SSRF (Server-side Request Forgery) attack and this attack occurs against the applications, which parses XML input by abusing the rarely used and weakly configured XML parser. With this attack, the attackers can be competent to cause the confidential data disclosure, Denial of Service (DoS), SSRF, Port Scanning and access remote or local files.

Risk Analysis

<table>
<thead>
<tr>
<th>Attack Vectors</th>
<th>Security Weaknesses</th>
<th>Impacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exploitability</td>
<td>Prevalence</td>
<td>Detectability</td>
</tr>
<tr>
<td>Moderate (2)</td>
<td>Common (2)</td>
<td>Easy (3)</td>
</tr>
</tbody>
</table>

**Threat Agents:** Attackers who can target vulnerable XML processors

**Exploitability:** Moderate
- Can be exploited by uploading XML or adding malicious content into an XML document.

**Security Weakness**
- Several older XML processors involve an external entity specification.
- Easy to detect with manual tests and tools like SAST & DAST.

**Technical Impacts:** Severe
- In addition to a DoS attack, these flaws can cause other attacks, extract data, and scan internal systems.

**Example:**

```
<?xml version="1.0"?>
<!DOCTYPE change-log [ <!ENTITY systemEntitySYSTEM "robots.txt"> ]>
<change-log> <text>&systemEntity;</text> </change-log>
```

Here,
External Entity is defined as, `<! ENTITY systemEntitySYSTEM "robots.txt">`

Where, External Entity with name “systemEntity” is of SYSTEM type, with path of robots.txt.

Further, this External Entity is used within an XML. Now, on Parsing, this will lead to call of robots.txt file. Then, the attacker can call some sensitive file or execute script instead.
How to Prevent

- Train your developers to find and mitigate the flaws
- Employ less complex data formats like JSON & avoid sensitive data serialization
- Upgrade your entire XML processors as well as libraries
- Use Dependency checkers
- It is better to disable the external entity & DTD processing of XML in entire XML parsers
- Implement filtering, sanitization, & positive server-side input validation to avoid the injection of malicious content into the XML documents, nodes, or headers
- Ensure that XSL file upload operation evaluates the incoming XML with XSD validation

A5 (2017) – Broken Access Control

Limitations on what authenticated users are permitted to do aren’t properly prescribed. Attackers use this vulnerability to achieve access to unauthorized data and functionality such as view sensitive files, access other’s accounts, alter other’s data, change access right and more.

Risk Analysis

<table>
<thead>
<tr>
<th>Attack Vectors</th>
<th>Security Weaknesses</th>
<th>Impacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exploitability</td>
<td>Prevalence</td>
<td>Detectability</td>
</tr>
<tr>
<td>Moderate (2)</td>
<td>Common (2)</td>
<td>Moderate (2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Severe (3)</td>
</tr>
</tbody>
</table>

Threat Agents: Any Authenticated user with Less Privileges

Exploitation: Moderate

- Occurs when the attacker changes the parameter value, which directly refers to a system object for which he is unauthorized

Security Weakness

- The occurrence is common in applications and APIs where all user request privileges are not verified
- Easy to detect with manual testing, but not open to automatic dynamic or static testing

Technical Impacts: Severe

- Privilege escalation

Example:

Let us analyse this flaw from practical prospect. Consider a page, which allows you to see source code of multiple pages.
To analyse this further, here we are using Burp as a proxy server. Proxy captures the request, before it is sent to the server.

Here, we simply change the file name to something sensitive and ask parsing engine to show the contents of the same.

Now, due to lack of authorization check of user input, the server will parse this query resulting in disclosure of sensitive content.
How To Defend

- Strict and proper Authentication & Authorization check at each level
- Check for Forced browsing
  - Check for default framework resources
- Always test unprivileged roles
  - Separation of duties
  - Capture and replay privileged requests
  - Include POST requests and async calls
- Deny all by default
- Audit application/server at regular interval

A6-Security Misconfiguration

Security setting that is defined, implemented and handled, as defaults often insecure. This condition undeniably allows the attackers to compromise the entire system. If a system or component is susceptible to outbreaks because of an insecure configuration, is classified as security misconfiguration.

Risk Analysis

<table>
<thead>
<tr>
<th>Attack Vectors</th>
<th>Security Weaknesses</th>
<th>Impacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exploitation</td>
<td>Prevalence</td>
<td>Detectability</td>
</tr>
<tr>
<td>Easy (3)</td>
<td>Widespread (3)</td>
<td>Easy (3)</td>
</tr>
</tbody>
</table>

Threat Agents: “Any” anonymous External User, Inside Attacks

Exploitation: Easy
- Attacker accesses default accounts, unused pages, unpatched flaws, unprotected files and directories

Security Weakness
- Can occur at any level of app stack like platform, server, application, database, framework ... etc
- Can be detected easily with automated scanners

Technical Impacts: Moderate
- Data leakage and sometimes complete system compromise

Examples
- Running outdated software including OS, Web/App Server, DBMS, Application and all code libraries
- Having un-necessary feature enabled/installed in the system. For example, services, pages, ports, privileges, and accounts
- Default accounts and passwords are still enabled and unchanged
- Having directory-listing enabled on the system server that leaks sensitive information.
- Secure values, not set in libraries and frameworks
- Third-party applications installed on the servers
- Misconfigured Firewalls
- Missing OS security patches

How To Defend
- Take steps to update everything. It is best to design the system in such a way the software patches and updates can be automatically deployed in a regular manner
- Avoid inconsistencies by using the same configuration for production, staging and developing environments.
- Perform scans and audits regularly in order to determine future misconfigurations.
- Build a strong application architecture, which offers secure and effective separation between components.
A7-Cross Site Scripting (XSS)

Cross Site Scripting attack, is possible if attacker is able to trick client system, like browser, to execute some code or script. Client system will execute same by considering that it is the part of application and runs same in that context. Cross Site Scripting attack primarily targets browser, not Server. Two main categories of XSS are:

1. **Persistent XSS** – Happens when the malicious client-side script is inserted directly into the susceptible web page or web application.
2. **Reflected XSS** – Involves reflecting of the malicious script. Here, the malicious script is implanted into a link. The script will execute only when the victim clicked that link.

**Risk Analysis**

<table>
<thead>
<tr>
<th>Attack Vectors</th>
<th>Security Weaknesses</th>
<th>Impacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exploitability</td>
<td>Prevalence</td>
<td>Detectability</td>
</tr>
<tr>
<td>Easy (3)</td>
<td>Widespread (3)</td>
<td>Easy (3)</td>
</tr>
</tbody>
</table>

**Threat Agents:** “Anyone” who can send untrusted Data to the system

**Exploitability:** Easy
- Even with a text based script, the attacker can easily exploit interpreter in the browser.

**Security Weakness**
- Occurs when an application includes user supplied data in a page transmitted to the browser without validating the content.
- Easy to detect via both code analysis and testing

**Technical Impacts:** Moderate
- Moderate for Reflected & DOM XSS but severe for Stored XSS. Impacts like Session Hijacking, Redirection, Deface Website and Insert Hostile Content.

**Example:**

1. **Reflected Type Cross Site Scripting**

Let us explore how an attacker call a client function as a part of the script, and data accordingly. If they place `document.cookie` as script value in the input field:
It will result in session id under alert window:

2. **Persistent Or Stored Cross Site Scripting**

Instead of inserting the blog content, attacker can insert the malicious Java script as shown below:

The script will be stored as an entry at server side, and will be executed every time when the page is loaded.
How To Defend

- Whitelist the user data
- Validate insecure attributes including ID, name, background
- Use HTTP only cookie flag
- Implement content security policy
- Encode the context to convert the untrusted input into a secure form where the input appears as data instead executing as a script in the browser. One can encode the context via HTML/attributes/Java Script.
- Consider the CSP (Content Security Policy) to safeguard against XSS attacks

A8- Insecure Deserialization

Serialization & Deserialization is a mechanism utilized for storing object or transferring an object state from one end to another. Serialization is the process of converting an object into another format that is persisted to disk, whereas deserialization is the process of obtaining the data or object from some other format. The deserialization can be re-modeled for malicious action when functioning on untrusted data that attacker control and it is said to be insecure deserialization.

Risk Analysis

<table>
<thead>
<tr>
<th>Attack Vectors</th>
<th>Security Weaknesses</th>
<th>Impacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exploitability</td>
<td>Prevalence</td>
<td>Detectability</td>
</tr>
<tr>
<td>Hard (1)</td>
<td>Common (2)</td>
<td>Moderate (2)</td>
</tr>
</tbody>
</table>

Threat Agents: Authenticated user with privileges

Exploitability: Hard

- Requires manual scripts as well as tweaks to exploit

Security Weakness

- Happens when an application & APIs de-serializes the tampered objects submitted by an attacker
- Though tools can be used to detect the flaw, it often requires the manual assistance for validation

Technical Impacts: Severe

- Can cause remote code execution and data tampering attacks

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;i:5:"Alice";i:2;i:5:"admin";
i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```
How To Defend

- Don’t accept serialized objects or data from untrusted sources
- Apply integrity checks like digital signatures on the serialized objects to avoid data tampering
- Enforce strict type constraints like - accept only definable set of classes during deserialization
- Isolate and run code, which involves deserialization in the low privilege environment
- Properly log the failures and exceptions of deserializations
- Properly monitor and restrict the incoming as well as an outgoing network association from servers or containers, which deserializes
- Regularly monitor the deserialization and make sure to alert in any user deserializes continuously

A9 – Using Components With Known Vulnerabilities

The modern application now relies on the open source and third-party libraries and software. It is common to have vulnerabilities in the software and third-party libraries and they could be utilized to compromise the system security using the software. This flaw occurs when we are offering full privileges intentionally or unintentionally to the components of the applications like frameworks, libraries, etc. In such situations, hackers get an opportunity to perform a serious attack.

Risk Analysis

<table>
<thead>
<tr>
<th>Attack Vectors</th>
<th>Security Weaknesses</th>
<th>Impacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exploitability</td>
<td>Prevalence</td>
<td>Detectability</td>
</tr>
<tr>
<td>Moderate (2)</td>
<td>Widespread (3)</td>
<td>Moderate (2)</td>
</tr>
</tbody>
</table>

Threat Agents: Depends on the application. Some vulnerable components of the system can be determined and exploited using automated tools.

Exploitability: Moderate

- Attacker identifies weak component through scanning or manual analysis, modify exploit and launch it accordingly

Security Weakness

- Most applications have this issue due to lack of Update/Upgrade process in an organization
- Developer/QA doesn’t know all components used and their interdependency on other components

Technical Impacts: Moderate

- Injection, Broken Access control, XSS. Etc

How To Defend

- Identify Components and Version
  - Look for all used components and dependable
  - Keep track for all versions
- Monitor for versions
- Plan for regular component upgrades
A10 – Insufficient Logging & Monitoring

This flaw occurs when the security-critical events aren’t logged properly and the system is not monitoring the current happenings. Since logs offer the summary of what has already occurred and monitoring can offer the summary of what is presently happening. The lack of these functionalities can make the malevolent activities harder to detect and typically affect the action of incident handling when an attack happens.

Risk Analysis

<table>
<thead>
<tr>
<th>Attack Vectors</th>
<th>Security Weaknesses</th>
<th>Impacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exploitability</td>
<td>Prevalence</td>
<td>Detectability</td>
</tr>
<tr>
<td>Moderate (2)</td>
<td>Widespread (3)</td>
<td>Hard (1)</td>
</tr>
</tbody>
</table>

**Threat Agents:** Any authorized user who can access to Logs

**Exploitability:** Moderate

The system is not providing adequate Logs to allow event traceback

**Security Weakness**

- Applications employ several components and generally, logs are either not created or are shared with some other components
- Hard to detect since along with automated tools, manual analysis is essential for various error conditions

**Technical Impacts:** Moderate

- Severely impact accountability

**How To Defend**

- Ensure all access control failure, login & server-side input validation failures can be logged properly with adequate user context. Since it might help to determine malicious or suspicious accounts
- Ensure the logged details are held for adequate time to perform delayed forensic analysis
- Make sure that the logs are produced in an easily consumable format
- Enforce that the high-value transactions should possess an audit trail along with integrity controls in order to avoid tampering/deletion
- Establish or adopt an incident response and recovery plan

When it comes to maintaining the security environment of your business, it is essential to address, the web security risks listed above and should be prepared for the additional evolving risks. Hope the above summary of top ten critical web application security risk with the reference to OWASP Top 10-2017 support the developers and organization minimize the risk in a cost-effective manner.