What is OWASP?

OWASP (Open Web Application Security Project) is a non-profit organization dedicated to enhancing software security. OWASP is based on an "open community" approach, allowing anybody to engage in and contribute to projects, events, online conversations, and other activities. OWASP's guiding concept is that all resources and information on its website are free and easily accessible to anyone.

OWASP offers a variety of tools, forums, projects, and events, among other things. In a nutshell, OWASP is a one-stop-shop for everything web application security, supported by the collective wisdom and experience of its open community contributors.

What are the OWASP Top 10 Security Risks?

The OWASP Top 10 is an online publication on the OWASP website that ranks the top 10 most critical web application security vulnerabilities and gives repair assistance. The report is based on an international agreement of security professionals.

The risks are ranked based on the frequency of security flaws disclosed, the severity of the flaws, and the extent of their possible consequences. The goal of the report is to provide insight into the most common security risks so that developers and web application security professionals may adopt the research's findings and suggestions into their security procedures, reducing the prevalence of these recognized hazards in their applications.

What is the Significance of the OWASP Top 10?

OWASP Top 10 is a research effort that ranks the top 10 most dangerous web application security threats and provides repair suggestions. The study is based on a consensus reached by security experts from around the world. The risks are categorized based on the severity of the flaws, the frequency of isolated security flaws, and the magnitude of their potential consequences.

The goal of the research is to give web application security professionals and developers a better understanding of the most frequent security issues so they can incorporate the results into their security procedures. This can assist in limiting the presence of known dangers in their online apps.

OWASP has been in charge of the Top 10 list since 2003. Every 2-3 years, they update the list to reflect changes and advances in the AppSec sector. For many of the world's largest enterprises, OWASP provides actionable information and serves as a crucial checklist and internal Web application development guideline.

Auditors often interpret an organization's failure to address the OWASP Top 10 as a sign that compliance standards aren't up to par. Including the Top 10 in its software development life cycle (SDLC) demonstrates a broad appreciation for the industry's best security standards.

Top 10 Vulnerabilities by OWASP

Following are the top 10 vulnerabilities and web application security threats, as listed by OWASP −

- SQL Injection
- Broken Authentication

- Exposed Sensitive Data
- XXE Injection
- Access Control Issues
- Misconfiguration of security
- Cross-Site Scripting
- Unsafe Design
- Using Vulnerability-Known Components
- Inadequate Logging and Monitoring

Let's take a look at each of these vulnerabilities in detail.

SQL Injection

The goal of an injection attack is to inject SQL, NoSQL, OS, and LDAP data into the application. It can be done through the application's input interface as SQL queries. If SQL injection is successful, the database's sensitive data may be exposed.

SQL injection can be used to edit database data using Insert, Update, and Delete statements, as well as shut down the DBMS (Database Management System) with merely a SQL injection.

Because of the lack of input validation and data sanitization, which might directly expose input into the query, injection happens when data is entered into a program from an un trusted source. This injection vulnerability may be found on practically any website, demonstrating how serious it is. Anything that accepts parameters as input can be vulnerable to injection.

Injection can result in data loss, corruption, or disclosure to unauthorized parties, loss of accountability, or denial of access. Injection can sometimes lead to complete host takeover. The business impact depends on the needs of the application and data.

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

Broken Authentication

Broken authentication is one of the OWASP top 10 significant vulnerabilities, which attackers can employ to impersonate a valid user online.

Session management and credential management are the two locations where this vulnerability is always present. These two are classified as broken authentication since they can both be used to steal login credentials or hijack session IDs. Attackers use a variety of techniques to exploit these flaws, ranging from credential stuffing to other highly targeted methods of gaining unauthorized access to someone's credentials.

Attackers have to gain access to only a few accounts, or just one admin account to compromise the system. Depending on the domain of the application, this may allow money laundering, social security fraud, and identity theft, or disclose legally protected highly sensitive information.

Ensure registration, credential recovery, and API pathways are hardened against account enumeration attacks by using the same messages for all outcomes.

Where possible, implement multi-factor authentication to prevent automated, credential stuffing, brute force, and stolen credential re-use attacks.

Exposed Sensitive Data

This is one of the OWASP Top 10 vulnerabilities for data compromise that requires protection. This is often referred to as information disclosure or leakage. This commonly happens when a program or website unintentionally releases sensitive information to people who do not have permission to see or access it.

These are some of the details that could be released to the public, according to OWASP −

- Information about money
- Login information
- Data that is commercial or business-related
- A medical history
- Technical information about the app or website

Even if you aren't utilizing DEBUG=True, you must exercise caution while managing the configuration parameter.

Failure frequently compromises all data that should have been protected. Typically, this information includes sensitive personal information (PII) data such as health records, credentials, personal data, and credit cards, which often require protection as defined by laws.

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.

XXE Injection

XML external entity injection (also known as XXE) is a security flaw that allows a malicious person to get access to an application that processes XML data or parses XML input, according to the OWASP.

Because XML input containing a reference to an external entity is handled by an XML parse that has been configured incorrectly, this attack is always effective. An attacker can examine files on the application server and interact with any other back end or external system that the application can access if this vulnerability is successfully exploited.

Through Server Side Request Forgery (SSRF) attacks, this XXE attack can be used to compromise other back-end or underlying systems.

The vulnerability is not in the data you give to the server in XML format; rather, it is in the way the XML is parsed.

When XML parsers that support DTD retrieval do not have sufficient input validation of the XML data in place, they may be vulnerable to XXE injection, which allows an attacker to inject commands or content into an XML document.

These flaws can be used to extract data, execute a remote request from the server, scan internal systems, perform a denial-of-service attack, as well as execute other attacks. The business impact depends on the protection needs of all affected application and data.

Access Control Issues

Users cannot behave outside of their intended permissions because access control enforces policy. Failures frequently result in unauthorized information disclosure, modification, or destruction of all data, as well as the execution of a business function outside of the user's capabilities.

The technical impact is attackers acting as users or administrators, or users using privileged functions, or creating, accessing, updating or deleting every record. The business impact depends on the protection needs of the application and data.

With the exception of public resources, deny by default. • Implement access control mechanisms once and re-use them throughout the application, including minimizing CORS usage. • Model access controls should enforce record ownership, rather than accepting that the user can create, read, update, or delete any record.

Misconfiguration of Security

Security misconfiguration is also one of the Top 10 vulnerabilities that might affect an application today, according to OWASP. An attack on a web server, database, network services, platforms, application server, frameworks, custom code, virtual machines, containers, and even storage can occur at any level of an application stack.

This type of configuration issue can allow attackers to get unauthorized access to some system data or functionality, leading to a complete system compromise and shutdown.

Cross-Site Scripting

Cross-Site Scripting, also known as XSS, is a client-side code injection, according to OWASP. The attacker attempts to inject malicious script into a trustworthy website in this type of attack. This script is in the form of JavaScript code, and it can unknowingly redirect a victim from their genuine site to an attacker site.

An attacker can utilize this flaw in an application to steal cookies and user sessions, obtaining unauthorized access to the system. Cross-Site Scripting can sometimes be combined with additional vulnerabilities to create a more powerful attack on an application.

Unsafe Design

For the creation of secure software, pre-coding tasks are essential. Security needs and model threats should be collected at the design phase of your development lifecycle, and development time should be scheduled to meet these requirements.

Your team should test assumptions and conditions for expected, and failure flows as software evolves to ensure they remain accurate and desirable. Failure to do so will allow crucial information to fall into the hands of attackers, as well as a failure to foresee innovative attack routes.

Using Vulnerability-Known Components

This vulnerability arises as a result of a developer employing a component, framework, library, or some dependencies that have a known weakness that could compromise the entire system.

When such components are run with full rights and are vulnerable, an intruder can easily exploit them, potentially resulting in catastrophic data loss or server takeover.

There's also a get version function that lets you know what version of the library the app is using at all times. You can also Google a library's current version to learn about the POC and vulnerabilities.

Inadequate Logging and Monitoring

Because inappropriate logging can result in information leakage, the need to safeguard a website or application through adequate logging and monitoring cannot be overstated.

Even though there is no such thing as 100 percent security, there are techniques to keep our website or application checked on a regular basis so that if we notice something unusual, we can rapidly take action to prevent an attack. If your website does not have an effective logging and monitoring method, it is vulnerable to being exploited and can harm an application's or website's reputation. As a result, keeping an audit record is critical if we wish to know about or uncover any questionable changes to our program or website.

SQL Injection

SQL injection attacks work by taking advantage of poorly-designed or poorly-implemented SQL code. When an application receives user input, it is often incorporated directly into an SQL query without proper validation or sanitization. This can allow attackers to insert malicious code into the query, which can then be executed by the database.

For example, consider a simple login form that asks for a username and password. The application might generate an SQL query like this to verify the user's credentials −

SELECT * FROM users WHERE username='$username' AND password='$password';

In this case, the username and password variables are replaced with the user's input. If a user enters their own username and password, the query will work as intended. However, if an attacker enters malicious input, they can manipulate the query to do things like retrieve sensitive data or even delete entire tables.

For example, an attacker might enter the following as their password −

' OR 1=1; -

This would modify the query to look like this −

SELECT * FROM users WHERE username='$username' AND password='' OR 1=1;

--';

The OR 1=1 statement will always evaluate to true, so the query will return all rows from the users table. The -- at the end is an SQL comment, which tells the database to ignore everything after it. This allows the attacker to bypass the rest of the query and gain access to the entire table.

Risk Mitigation

Risk mitigation involves a systematic process in which an organization identifies, analyses, and proposes measures to effectively handle various risks that might pose threat to an organization's functions or operations. Every organization must carefully curate its risk mitigation plan.

PMBOK offers the following focus areas or procedures for an effective risk management plan −

- Plan a risk management process
- Identify potential risks
- Conduct a qualitative risk analysis
- Conduct a quantitative risk analysis
- Plan responses for risks
- Implement risk responses
- Monitor the potential risks

Various Risk Mitigation Strategies:

Depending on the organization and the type of project, the risk mitigation strategies might differ. Sometimes one strategy might be preferably used when compared to others. While in some cases, a combination of strategies can be used depending on the magnitude of the problem at hand. Here are the different types of approaches that can be followed to mitigate risks in a project −

Acceptance of Risks

This approach focuses on accepting certain risks in an organization for a certain period of time. Such an approach will help in prioritizing resources and efforts over solving risks that are deemed to have severe consequences.

This is carried out by identifying the risks and the vulnerabilities associated with those risks. Hence all the members working on a project will know that these risks exist. By bringing such acceptable risks to the business’s attention, organizations can effectively handle any setbacks.

Avoidance of Risks

This approach is particularly used when the consequences of risk are very severe. The acceptance of risks is totally not advisable because the consequences are larger than the cost of mitigation. This strategy is quite common and must be implemented earlier in any process.

Transfer of Risks

The transfer of risk approach transfers the responsibility of managing risks to various parties depending on their capacity to mitigate the risk.

Controlling Risks

This risk control strategy is usually used to mitigate risks that were identified and accepted. The main objective of such an approach is to contain the impacts caused by acceptable risks to a minimum.

Monitoring of Risks

This strategy needs to be in place so that the risks are monitored continuously to identify changes that might impact the mitigation process. Risks associated with cost, performance, and scheduling can all be monitored continuously and sometimes this forms a part of an organization's standard review plan.

Tools Used for Risk Mitigation

The risk mitigation process needs to be systematic in order to ensure proper action is taken at various stages of a project. In order to do this the following tools or techniques are being used by organizations across the world −

- Risk Assessment Framework (RAF)/Risk Assessment Criteria Matrix
- Risk Index Priority Number
- Probability and Impact Matrix
- Strength, Weakness, Opportunity, and Threats (SWOT) Analysis
- Root cause analysis

Broken user authentication

Poor implementation of API authentication allows attackers to predict other user's identities. In more general terms, broken user authentication occurs when an API having an authentication system but does not in working, or that the implemented authentication system fails in some cases, allowing attackers to project as an authenticated user.

The weaknesses present in the system, mentioned above, will be divided into two different groups, namely poor credential management and poor session management.

A. Poor credential management

Victim credentials can be collected to gain access to the system. There are various ways that the attacker can steal sensitive information, such as the following −

- **Weak passwords** − If the victim creates a weak password like '12345' or 'pass123'. The attacker can use different types of password cracking techniques like rainbow tables and dictionaries are used to brute force and to gain access to the system.
- **Weak cryptography** − Using weak decryption/encryption algorithms like base64 and weak encryption algorithms like SHA1 and MD5 make credentials vulnerable. That's why they must be stored using strong hashing algorithms that make password cracking challenging.

B. Poor session management

The application allocates a session ID to you, whenever your login and it will store all your interactions. It is through this session ID that the application make interaction with you and give responds to all your requests. If an attacker steals your session ID, then they can sign-in by impersonating your identity. This is known as session hijacking.

Below are the list of scenarios that can cause broken authentication.

- Weak usernames and passwords.
- Session fixation attacks.
- URL redirecting.
- User identity details aren't protected when stored with hashing algorithms.
- User identity details are transferred over unencrypted connections.

Broken Authentication Examples

Example #1 − Credential Stuffing

Suppose you run a online groceries store to sell groceries. To grow your business, you implemented a CRM system that collects and stores critical customer data, such as name, phone number, username, and password.

Hackers make their way to exploit CRM system to steal all the data. Then they used the same credentials (usernames and passwords) to login into the customer’s central bank's database.

In this case, hackers are trying to successfully login to the central bank's database by hoping that a consumer must be using the same credentials at both places. Such kinds of broken user authentication attacks are called as credential stuffing.

Example #2 − Application session timeouts aren't set properly.

Suppose you go to a cyber cafe and login your Gmail account from there computer. After sending the mail, you just close the browser tab and return home.

After sometime, the hacker opens your Gmail account and gains access to your sensitive information. It happens because your credentials (username and password) haven't been invalidated adequately while closing browser as the session ended.

Thus, if the application session timeouts aren't implemented properly, hackers can perform a broken authentication attack.

What is Cross-Site Scripting?

The technique of inserting malicious code on a legitimate website in order to capture user information for nefarious reasons is known as cross-site scripting (XSS).

- By inserting malicious code in a genuine web page or web application, the attacker attempts to execute harmful scripts on the victim's web browser.
- The real attack occurs when the victim visits a website or uses a web application containing malicious code. The malicious script is delivered to the user's browser via the web page or application.
- Forums, message boards, and online pages with commenting capabilities are typical targets for Cross-site Scripting attacks.

When a web page or a web application produces output that incorporates un sanitized user input, it is known as XSS. This user input must be parsed by the victim's browser.

- XSS attacks can be carried out in VBScript, ActiveX, Flash, and even CSS. They are, nevertheless, most ubiquitous in JavaScript because JavaScript is essential to most browser experiences.
- The security of that vulnerable website or online application, as well as its users, has been compromised if an attacker can exploit XSS vulnerability on a web page to run arbitrary JavaScript in a user's browser.
- Cross-site scripting can be used to deface a website. The attacker can use injected scripts to alter the website's content or even redirect the browser to another website, such as one that includes malicious code.

Types of Cross-site Scripting (XSS) Attacks

XSS attacks come in a variety of forms. They are classified into the following categories −

Persistent XSS

This sort of vulnerability, also known as stored XSS, happens when un trusted or unverified user input is stored on a target server. Message boards, comment sections, and visitor logs are all common targets for persistent XSS—any feature where other users, both authenticated and unauthenticated, would see the attacker's malicious text.

A good example of a suitable target for persistent XSS is publicly available profile pages, such as those seen on social media sites and in membership groups. When other users view the profile, their browsers will automatically execute the code entered by the attacker in the profile boxes.

Reflective XSS

Reflected or non-persistent cross-site scripting, on the other hand, entails the immediate return of user input. An attacker must mislead the user into transferring data to the target site to exploit a reflected XSS, which is generally done by deceiving the user into clicking a maliciously designed link.

Reflective XSS attacks frequently rely on phishing emails or URLs that have been truncated or otherwise hidden before being transmitted to the target user. When the victim clicks on the link, the script runs in the victim's browser.

Reflected XSS commonly targets search results and error message sites. They frequently send unaltered user input as part of the response without appropriately escaping the data so that it can be safely shown in the browser.

DOM-Based XSS

DOM-based cross-site scripting, also known as client-side XSS, is similar to reflected XSS in that it is frequently provided via a rogue URL containing a harmful script. The attack is carried out entirely within the browser rather than including the payload in a trusted site's HTTP response by changing the DOM (Document Object Model). This is aimed at legal JavaScript already on the page and fails to sanitize user input properly.

Sensitive Data Exposure

As the online applications keep flooding the internet in day by day, not all applications are secured. Many web applications do not properly protect sensitive user data such as credit cards information/Bank account info/authentication credentials. Hackers might end up stealing those weakly protected data to conduct credit card fraud, identity theft, or other crimes.

Let us understand Threat Agents, Attack Vectors, Security Weakness, Technical Impact and Business Impacts of this flaw with the help of simple diagram.

Broken Access Control

Broken access control refers to a situation in which an attacker is able to access a resource or perform an action that should be restricted. This can happen for a number of reasons. For example, a developer may have inadvertently made a security mistake, or a vulnerability may have been introduced due to a third-party library or service.

There are many different ways in which access control can be broken. For example, an attacker may be able to bypass authentication, gain access to privileged data or resources, or modify data that should be read-only. Broken access control can occur at any level of an application, from the user interface to the back-end data storage.

What are the Consequences of Broken Access Control?

The consequences of broken access control can be severe. In some cases, an attacker may be able to gain full control of a system, steal sensitive data, or delete important information. For example, an attacker might be able to bypass a login screen and gain access to a user's account. Once inside, the attacker could steal sensitive information, modify account settings, or even initiate fraudulent transactions.

In other cases, the damage caused by broken access control may be more subtle. For example, an attacker might be able to gain access to sensitive information that they are not authorized to see, leading to privacy violations or regulatory noncompliance. Broken access control can also lead to reputational damage, loss of customer trust, and other business risks.

How to Prevent Broken Access Control?

Preventing broken access control requires a multi-layered approach that involves careful design, development, and testing. In the following sections, we'll explore some of the key techniques and best practices that can help prevent broken access control.

1. Access Control Design

One of the most important steps in preventing broken access control is to design access control measures that are robust and effective. Access control design should consider the different roles and levels of access that users or entities will need, and ensure that the appropriate restrictions are in place.

For example, it's important to limit access to administrative functions to only those users who need it, and to ensure that access is granted based on the principle of least privilege. This means that users are given only the minimum level of access that they need to perform their job, and no more. This can help prevent situations where a user with too much access accidentally or intentionally causes harm to the system.

2. Authentication and Authorization

Authentication and authorization are two key components of access control. Authentication is the process of verifying a user's identity, while authorization refers to the process of determining whether a user is allowed to perform a particular action or access a particular resource.

To prevent broken access control, it's important to ensure that both authentication and authorization are implemented correctly. Authentication should be implemented using strong and reliable techniques such as multi-factor authentication or biometric authentication. Authorization should be designed to ensure that users are only given access to the resources that they are authorized to access.

3. Input Validation and Output Encoding

Another important technique for preventing broken access control is input validation and output encoding. Input validation refers to the process of checking user input to ensure that it is valid and safe. Output encoding refers to the process of encoding output data to prevent it from being misinterpreted or exploited.

4. Error Handling

Access control errors can occur due to various reasons such as incorrect user inputs, system failures, and unexpected conditions. It's important to implement proper error handling mechanisms to prevent attackers from exploiting such errors to gain unauthorized access. Error messages should not expose sensitive information and should be worded in a clear and simple manner so that users can comprehend them.

XML External Entity attack

XXE or XML External Entity attack is a web application vulnerability that affects a website which parses unsafe XML that is driven by the user. XXE attack when performed successfully can disclose local files in the file system of the website. XXE is targeted to access these sensitive local files of the website that is vulnerable to unsafe parsing.

Types of XXE Attacks

1. **File Retrieval XXE**: As the name implies, arbitrary files on the application server of a victim company can be exposed to the attacker, if there is an XXE vulnerable endpoint in the target system. This can be carried out by passing an external XML entity in the user-controlled file.

2. **Blind XXE:** It is possible that a target system doesn’t return data from the entities placed by the attacker still being insecure and vulnerable to XXE. This is done by trying out malformed user inputs. These include the input of length more than what the system

expects the wrong data type, special entities, etc. The intention is to make the system fail and check if throws out some sensitive information in the error response.

3. **XXE to SSRF:** Even if the system doesn't return the response with local file content to the attacker, the system can be still exploited in presence of an XXE attack. The entity can be pointed to a local IP of the target company which can be accessed only by its websites/network. Placing an intranet IP in XXE payload will make the target application call its local endpoint which the attacker won't have access to otherwise. This type of attack is called SSRF or Server Side Request Forgery.

XML Parsing

XML is one of the commonly used data exchange formats. Data can be transferred between User and Website in XML format. Consider a website that accepts User information in form of XML.

Cross site request forgery (CSRF) is vulnerability where an attacker performs actions while impersonating another user. For example, transferring funds to an attacker's account, changing a victim's email address, or they could even just redirect a pizza to an attacker's address!

Some form of social engineering, like phishing or spoofing, is usually required for this kind of attack to be successful. The attacker typically needs to trick the user into visiting a malicious website for the attack to take place. This malicious website would then contain a request to the targeted website. If the user is authenticated by the targeted website, the request is executed. This attack works because the user's cookies are automatically included in the modified request to a legitimate application. CSRF vulnerabilities occur when vulnerable web apps simply trust the cookies sent by web browsers without further validation.

CSRF in action

A web application is vulnerable to CSRF if it relies on session cookies to identify users, and doesn't have any other mechanism for validating requests. Additionally, the request we want to exploit needs to have predictable request parameters so that we can create our own request, like the one in this example.

Luckily for us, we've been tipped off about a vulnerable banking application called "Saturn Bank". This app simply uses session cookies to verify requests. Additionally, the session cookies don't have the Same Site attribute. We'll deep dive into what this attribute is later, in short they control how cookies are submitted in cross site requests.

Components with known vulnerabilities :

- These components can be defined as the third-party apps or software or platforms that are outdated and contain bugs that are public to all, that is- sites like https://www.exploit-db.com contain the full detail as to how to exploit the bugs to put the security of the whole website under severe threat.
- This vulnerability arises with the fact that a website finds it difficult to code everything while making its website functional like -transaction, location, chats, etc.
- So, in order to ease the process of building the website, many websites use third-party apps which do the tasks. But the main problem is that those apps can be harmful to their system if they are not updated regularly. Components with known vulnerabilities are considered to be one of the top 10 web application vulnerabilities listed by OWASP.
- Many websites security gets compromised when they use components having known vulnerabilities.

How to spot:

There are automated tools available online ex- drop scanner which display all the outdated components present in the software. These automated tools save time for security experts by directly pointing out flaws of the website.

Business impact:

When a website uses such outdated software, the hacker can misuse the flaws and get inside the database of the website and can cause serious damage to the website by stealing critical information.

Un validated Redirects and Forwards

Unvalidated Redirects and Forward Vulnerability, also sometimes referred to as URL Redirection Vulnerability, is a type of bug found in the Web Application. In this type of vulnerability, the attacker uses to manipulate the URL and sends it to the victim. As soon as the victim opens the URL, the website redirects it to a malicious website or website to which the attacker wants the user to get redirected. The attacker generally uses to exploit this type of Vulnerability with the help of manual manipulation in the URL or with the help of several tools like *Burpsuite*, which gives an attacker several types of ways due to which he can manipulate the URL to get Redirected.

How does URL Redirection work?

First of all, we need to get a brief idea about the HTTP Response Codes. So here are the response codes:

- 1×× Informational
- 2×× Success
- 3×× Redirection
- 4×× Client Error
- 5×× Server Error

The above HTTP Status code tells us about the response that we receive from a website. So for URL redirection, generally, *3xx* Codes are redirection codes that say to the user that this URL is going to get redirected to some other page. The attacker takes advantage of this and tries to inject their payloads or manipulate the URL to send the victim to their malicious website. Below is the screenshot of the Burpsuite via which an attacker can add filters and can find out specific URLs having 3xx codes.

Hackers leverage gaps in logging and monitoring by relying on the fact that security teams will take time to detect and remediate the attack to try and escalate privileges. This section explores the threats associated with insufficient logging & monitoring and the business impacts of a successful attack.

The fundamental reason for an inadequately logged system getting exploited by attack vectors are typically based on the following demerits that occur in the absence of an efficient logging and monitoring framework:

- Unlogged events and transactions
- Missing log backups
- Obscure error logging
- Missing breach escalation plans
- Poor authentication management
- Ineffective training on logging and monitoring

- Lack of exports to analyze log data
- Software misconfigurations

Threats Associated with Insufficient Logging & Monitoring

Botnet Attacks

Attackers often leverage several devices connected to the internet to inject malware into a system and coordinate a cyber attack. Such malware is *automated bots* that manipulate the application in different ways – from simple spamming operations to performing more complex attacks intended to manipulate the application.

These are also commonly supported by *botnets* that orchestrate various attacks, including Brute Force, Phishing, and Distributed Denial of Service (DDoS) attacks. Botnet attacks rely on a chain of actions running through multiple stages. In the absence of proper logging of event data, these attacks are almost impossible to detect or analyze.

An efficient monitoring system with tools like Syslog is often considered the primary first line of defense to reduce the likelihood and severity of Botnet attacks.

DNS Attacks

A Domain Name Service (DNS) offers a standard mechanism to point machine hostnames to their IP addresses. Since DNS directs network traffic towards the correct web servers and target machines, these are common vulnerable points that are often exploited by attack vectors to target the availability or stability of the DNS server as part of the overall attack strategy.

Some possible DNS attacks include:

- Cache poisoning
- Distributed Reflection DoS Attacks
- NXDOMAIN attacks
- DNS Tunneling
- Random Sub domain Attacks
- Domain lock-up attack

If DNS-based events are not logged and appropriately monitored, administrators won’t know the types of machines attackers (in the disguise of users) query and interact with. Additionally, threat actors can perpetuate malicious actions such as malware installation, credential theft, command

& control communication, network foot printing, and data theft in the absence of adequate query logging and analysis.

Insider Threats

Organizations that typically invest a fortune in securing systems from external attacks often miscalculate internal threats. Such internal threat actors continue to be a critical concern for organizations since their suspicious activities often go unchecked. In such cases, malicious or compromised insiders pose a severe threat to systems since they have access to various control and security measures. Though a situation like this sounds astonishing, the mitigation is relatively straightforward and straightforward and relies on an efficient logging mechanism.

Insufficient monitoring and log management in such instances result in untraceable user behavior patterns, thereby allowing imposters or malicious insiders to compromise the system at a much deeper level.

Some commonly known insider threats arising from insufficient logging & monitoring include:

- Malware traffic
- Ransom ware attacks
- Advanced Persistent Threat

Secure Coding Best Practices

OWASP provides a secure coding practices checklist that includes 14 areas to consider in your software development life cycle. Of those secure coding practices, we're going to focus on the top eight secure programming best practices to help you protect against vulnerabilities.

1. Security by Design
2. Password Management
3. Access Control
4. Error Handling and Logging
5. System Configuration
6. Threat Modeling
7. Cryptographic Practices
8. Input Validation and Output Encoding

Security by Design

Security needs to be a priority as you develop code, not an afterthought. Organizations may have competing priorities where software engineering and coding are concerned. Following software security best practices can conflict with optimizing for development speed. However, a "security by design" approach that puts security first tends to pay off in the long run, reducing the future cost of technical debt and risk mitigation. An analysis of your source code should be conducted throughout your software development life cycle (SDLC), and security automation should be implemented.

Password Management

Passwords are a weak point in many software systems, which is why multi-factor authentication has become so widespread. Nevertheless, passwords are the most common security credential, and following secure coding practices limits risk. You should require all passwords to be of adequate length and complexity to withstand any typical or common attacks. OWASP suggests several coding best practices for passwords, including:

- Storing only salted cryptographic hashes of passwords and never storing plain-text passwords.
- Enforcing password length and complexity requirements.
- Disable password entry after multiple incorrect login attempts.

We have also written about password expiration policies and whether they are a security best practice in a modern business environment.

Access Control

Take a "default deny" approach to sensitive data. Limit privileges and restrict access to secure data to only users who need it. Deny access to any user that cannot demonstrate authorization. Ensure that requests for sensitive information are checked to verify that the user is authorized to access it.

Learn more about access controls for remote employees and cloud access management.

Error Handling and Logging

Software errors are often indicative of bugs, many of which cause vulnerabilities. Error handling and logging are two of the most useful techniques for minimizing their impact. Error handling attempts to catch errors in the code before they result in a catastrophic failure. Logging documents errors so that developers can diagnose and mitigate their cause.

Documentation and logging of all failures, exceptions, and errors should be implemented on a trusted system to comply with secure coding standards.

System Configuration

Clear your system of any unnecessary components and ensure all working software is updated with current versions and patches. If you work in multiple environments, make sure you're managing your development and production environments securely.

Outdated software is a major source of vulnerabilities and security breaches. Software updates include patches that fix vulnerabilities, making regular updates one of the most vital, secure coding practices. A patch management system may help your business to keep on top of updates.

Threat Modeling

Document, locate, address, and validate are the four steps to threat modeling. To securely code, you need to examine your software for areas susceptible to increased threats of attack. Threat modeling is a multi-stage process that should be integrated into the software lifecycle from development, testing, and production.

The main secure design principles are the following:

a) **Economy of mechanism:** Keep the design as simple and small as possible.
b) **Fail-safe defaults:** Base access decisions on permission rather than exclusion.
c) **Complete mediation:** Every access to every object must be checked for authority (there and then).
d) **Open design:** The design (and the code) should not be considered secret. The secret is always data, like a password or a cryptographic key.
e) **Separation of privilege:** It's always safer if it takes two parties to agree on a decision than if one can do it alone.
f) **Least privilege:** Operate with the minimal set of powers needed to get the job done.
g) **Least common mechanism:** Minimize subsystems shared between or relied upon by mutually distrusting users.
h) **Psychological acceptability:** Design security systems for ease of use for humans.

The two additional secure design principles are:

i) **Work factor:** Compare the cost of circumventing the mechanism with the resources of a potential attacker.
j) **Compromise recording:** Record that a compromise of information has occurred.

Threat Modeling

Purpose of Threat Modeling

The purpose of Threat modeling is to identify, communicate, and understand threats and mitigation to the organization's stakeholders as early as possible. Documentation from this process provides system analysts and defenders with a complete analysis of probable attackers' profiles, the most likely attack vectors, and the assets most desired by the attacker**.**

Achievement of Threat Modeling

1. Defines security of application
2. Identifies and investigates potential threats and vulnerabilities
3. Results in finding architecture bugs earlier

Process of Threat Modeling

1. Aim

The target before approaching Threat Modeling must be clear within ourselves what we will achieve from Threat Modeling, that is our application must follow the CIA Triad.

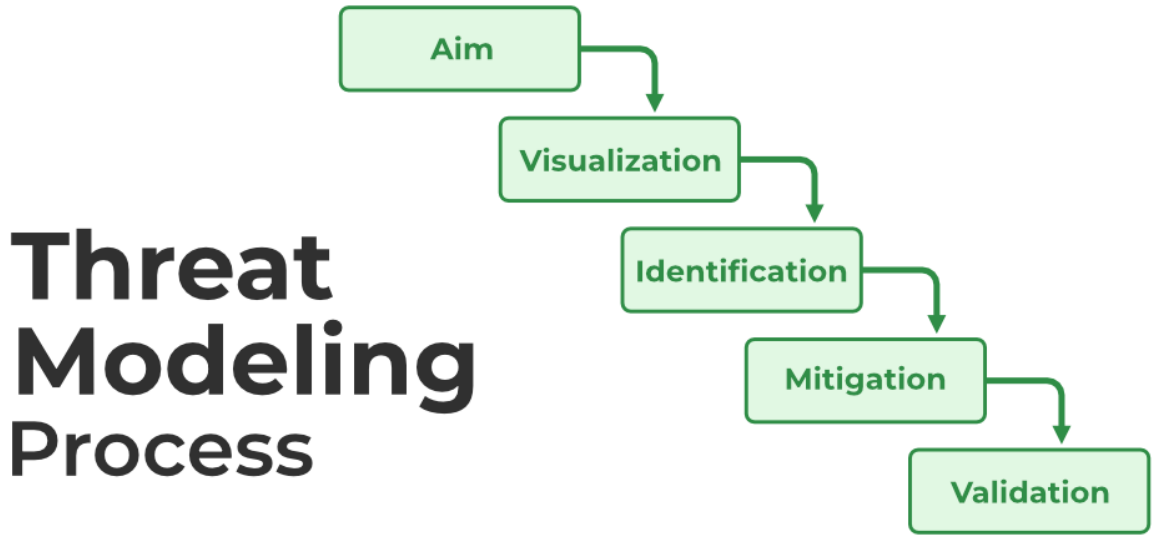
- Confidentiality: It helps in protecting data from unauthorized access.
- Integrity: It helps in preventing restricted changes.
- Availability: It helps in performing important tasks under certain attacks.

2. Visualization

Here, we will deal with what we are going to build. We must have a document overview of the application which helps in making our process easier. Here we will build diagrams that will help us in making our process easier.

It can be done in two ways:

- Data Flow Diagram: It helps in showing how the flow of data occurs in the system.
- Process Flow Diagram: It helps in finding the process of the system that from where users interact in the system, and how the system works internally.



3. Threat Identification

Here we are going to deal with how we can identify threats or what can go wrong in the process. By analyzing the images of the previous section, you have found how threats can be identified. These methods are mentioned in threat modeling methodologies..

4. Mitigation

Here, we are going to deal with what we will do about the Threats. Here we will review the layers to identify the required vulnerabilities. Mitigation involves a continuous investigation of each vulnerability so that the most effective efforts can be designed.

5. Validation

This is the final step in the process of Threat Modeling, here we are going to deal with whether we have done a good job or not. Have all the threats been mitigated or not? We will check the changes and as threat modeling is not a one-time activity, we have to regularly watch these things.