OWASP Top 10 Summary Document

Table of Contents

What is the OWASP Top 10?

The Open Web Application Security Project (OWASP) is a nonprofit cybersecurity foundation with the goal to secure the web. OWASP collects data and conducts security surveys on identified application vulnerabilities from security vendors and consultancies, bug bounties, and company/organizational contributions. This data is collated into a document called the OWASP Top 10. The OWASP Top 10 represents a broad consensus about the most critical security risks to web applications. All information in this document about OWASP and the vulnerabilities they list is gathered directly from OWASP's website, owasp.org.

Why is the OWASP Top 10 important to Time To Study?

As the Time To Study web application is developed, it is important to consider the potential vulnerabilities for a web application so the vulnerabilities can be corrected or mitigated prior to the application's launch. The OWASP Top 10 does not provide a comprehensive list of all vulnerabilities that may be present in our application, but does list the most common (more accurately, the most observed) and many of the most likely vulnerabilities to be utilized by an attacker. Therefore, the OWASP Top 10 is a good starting place for testing Time To Study for vulnerabilities, although more testing and vulnerability scanning will be necessary beyond the categories in the OWASP Top 10. Future testing and vulnerability scanning will be documented in further documentation.

A01:2021-Broken Access Control

Access control enforces an organization's permissions. Access Control failures result in users having the ability to act outside of what the organization's policy would allow them to do. For time to Study, this would include ensuring that users are unable to access or edit information that is not theirs

Common access control vulnerabilities include:

- Bypassing access control checks by modifying the URL (parameter tampering or force browsing), internal application state, or the HTML page, or by using an attack tool modifying API requests.
- Permitting viewing or editing someone else's account, by providing its unique identifier (insecure direct object references)
- Accessing API with missing access controls for POST, PUT and DELETE.
- Elevation of privilege. Acting as a user without being logged in or acting as an admin when logged in as a user.
- Metadata manipulation, such as replaying or tampering with a JSON Web Token (JWT) access control token, or a cookie or hidden field manipulated to elevate privileges or abusing JWT invalidation.
- CORS misconfiguration allows API access from unauthorized/untrusted origins.
- Force browsing to authenticated pages as an unauthenticated user or to privileged pages as a standard user.

How to Prevent:

- Except for public resources, deny by default.
- Implement access control mechanisms once and re-use them throughout the application, including minimizing Cross-Origin Resource Sharing (CORS) usage.
- Model access controls should enforce record ownership rather than accepting that the user can create, read, update, or delete any record.
- Unique application business limit requirements should be enforced by domain models.
- Disable web server directory listing and ensure file metadata (e.g., .git) and backup files are not present within web roots.
- Log access control failures, alert admins when appropriate (e.g., repeated failures).
- Rate limit API and controller access to minimize the harm from automated attack tooling.
- Stateful session identifiers should be invalidated on the server after logout. Stateless JWT tokens should rather be short-lived so that the window of opportunity for an attacker is minimized. For longer lived JWTs it's highly recommended to follow the OAuth standards to revoke access.

A02:2021-Cryptographic Failures

Cryptographic failures are the result of the failure to properly protect data at rest and in transit. This failure can both be in the determination of the protection needs of data, and the failure to properly act on that determination. In addition to protecting the privacy of the users and company data, it is also important to be compliant with relevant laws, such as GDPR and PCI DSS. For Time To Study, this means that it is important for us to both properly determine what kind of protection data requires, and properly implement that level of protection. The primary way that data will be protected is through encryption, but when implementing it, it is important to do it properly according to the previous determinations, and to ensure that any algorithms used have not been broken before. Time To Study would not be collecting data that would fall under HIPPA regulation or similar US laws, but it is important to ensure that all relevant laws are being followed, rather it be Federal, or State laws, such as the California Consumer Privacy Act (CCPA), if Time To Study operates in California. This is also relevant if we decide to operate in other countries, as if we collect data from citizens of a country in the European Union, then we will have to ensure our data collection is compliant with GDPR. In addition to the legal assurances, it is important to implement encryption in line with a user's reasonable expectation of privacy would be.

Common cryptographic failures result from a failure to properly determine the protection needs of data. Some good questions to ask to properly determine the protection needs of data are:

- Is any data transmitted in plain text?
- Are any weak or old cryptographic algorithms used?
- Are default crypto keys in use? Is proper key management/rotation missing?
- Is encryption being properly enforced?
- Is the received server certificate and the trust chain properly validated?

- Are initialization vectors ignored, reused, or not generated sufficiently secure for the cryptographic mode of operation? Is an insecure mode of operation in use?
- Are passwords being used as cryptographic keys in absence of a password base key derivation function?
- Is randomness used for cryptographic purposes that does not meet the cryptographic requirements? Does the randomness function need to be seeded by the developer, and if not, has the developer over-written the strong seeding functionality built into it with a seed that lacks sufficient entropy/unpredictability?
- Are deprecated hash functions such as MD5 or SHA1 in use, or are non-cryptographic hash functions used when cryptographic hash functions are needed?
- Are deprecated cryptographic padding methods in use?
- Are cryptographic error messages or side channel information exploitable, for example in the form of padding oracle attacks?

A03:2021-Injection

There are many kinds of injection attacks, such as SQL, NoSQL, OS command, etc., but they all are conceptually the same. An injection attack occurs when an application does not properly sanitize their user input, allowing an attacker to input, or ‘inject,’ malicious commands, queries, or code into the application through user-supplied data, such as when typing in a username or password. Since Time To Study utilizes SQL, we need to be cautious of the potential of SQL injections when creating the application. Applications are vulnerable to injection attacks when:

- User-supplied data is not validated, filtered, or sanitized by the application.
- Dynamic queries or non-parameterized calls without context-aware escaping are used directly in the interpreter.
- Hostile data is used within object-relational mapping (ORM) search parameters to extract additional, sensitive records.
- Hostile data is directly used or concatenated. The SQL or command contains the structure and malicious data in dynamic queries, commands, or stored procedures.

A04:2021-Insecure Design

In the OWASP Top Ten, the term “insecure design” is a broad category. Not to be confused with insecure implementation, insecure design is when the design of the application is flawed in a way that even with a perfect implementation, it will remain insecure. A secure design can still be made insecure by a defective implementation, but an insecure design can never be made secure, even by a perfect implementation. This is most commonly a result of not properly risk profiling the software or system, resulting in a failure to determine what level of security is necessary in the design of the implementation.

A05:2021-Security Misconfiguration

As the name suggests, security misconfiguration is the result of various failures to properly configure or maintain security features. Some ways this can happen are:

- Default accounts and passwords are still enabled and unchanged.
- Unnecessary features are enabled or installed, such as unnecessary ports, services, pages, accounts, or privileges.
- Missing appropriate security hardening across any part of the application stack or improperly configured permissions on cloud services.
- Error handling reveals overly informative error messages to users.
- The latest security features are disabled or not configured securely for upgraded systems.
- The security settings in the application servers, application frameworks (such as ASP.NET for Time To Study), libraries, databases, etc., are not set to secure values.
- The server does not send security headers or directives, or they are not set to secure values.
- The software is out of date or vulnerable (see next section for more information)

The primary way to prevent security misconfiguration is to focus on the process of installation itself, and here are some aspects that can assist that:

- A repeatable hardening process with identical configuration between different environments (with different credentials). The repeatability, and identical configurations will make it fast and easy to deploy new environments, ideally automated, which reduces the effort, and makes complying to security procedures simpler.
- Minimize the platform and remove any unnecessary features, components, documentation, and samples. Any unused features and frameworks should be removed. This will minimize unnecessary features remaining enabled or installed, and will result in an environment that is simpler to maintain.
- A task to review and update the configurations appropriate to all security notes, updates, and patches, as a part of the patch management process.
- A segmented application architecture, which provides effective and secure separation between components or tenants, with segmentation, containerization, or cloud security groups.
- Sending security directives to clients, such as with security headers
- An automated process to verify the effectiveness of the configurations and settings in all environments.

A06:2021-Vulnerable and Outdated Components

As the name suggests, these vulnerabilities are the result of vulnerable and outdated components being in use. We must ensure that Time To Study is not utilizing vulnerable or outdated components. The application if likely vulnerable if:

- You do not know the versions of all components you use both server-side and client-side. This includes both components directly used as well as nested components.
- The software is vulnerable, unsupported, or out of date. This includes the Operating System, web or application server, database management system, APIs, as well as all components, runtime environments, and libraries.
- You do not scan for vulnerabilities regularly, and don't receive bulletins related to components you use.
- You do not fix or upgrade the underlying platform, frameworks, and dependencies in a risk based, timely fashion.
- Software developers do not test compatibility of updated, upgraded, or patched libraries.
- You do not secure the components' configurations (see upper section A05).

A07:2021-Identification and Authentication Failures

Identification and Authentication failure is the failure to properly confirm a user's identity, authentication, or session management. Common weaknesses in this area include:

- The application permits automated attacks, including credential stuffing, where the attacker has a list of valid usernames and passwords.
- The application permits brute force or other automated attacks
- The application permits default, weak, or well-known passwords, such as "Password1" or "admin/admin". For some quick references for weak passwords, there is a Wikipedia page that lists the most common passwords reported by various organizations, and there are also tools such as the rockyou wordlist that are easily obtainable from GitHub by anyone and should be taken into account.
- The use of weak or ineffective credential recovery and forgot-password processes, such as "knowledge-based answers" which cannot be made safe.
- The application uses plain test, encrypted, or weakly hashed passwords data stores (see cryptographic failures section)
- The application has missing or ineffective multi-factor authentication.
- The application exposes the session identifier in the URL.
- The application reuses the session identifier after successful login.
- The application does not correctly invalidate session IDs. User sessions or authentication tokens aren't properly invalidated during logout or a period of inactivity.

A08:2021-Software and Data Integrity Failures

Software and Data integrity failures relate to code and infrastructure that does not protect against integrity violations. This can be a result of an application relying on elements from untrusted sources, repositories, and content delivery networks. This can introduce potential for unauthorized access, malicious code, or system compromise. Application auto-updating functionality can also cause this vulnerability, when updates are downloaded without sufficient integrity verification and applied to the previously trusted application. Attackers could potentially upload their own updates to be distributed and run on all installations. For Time To

Study, it is important that we are using installations and libraries from trusted sources, and verify that they have not been compromised.

Some ways to prevent this vulnerability are:

- Use digital signatures or similar mechanisms to verify the software or data is from the expected source and has not been altered.
- Ensure libraries and dependencies, such as npm or Maven, are consuming trusted repositories.
- Ensure that a software supply chain security tool, such as OWASP Dependency Check or OWASP CycloneDX, is used to verify that components do not contain known vulnerabilities.
- Ensure that there is a review process for code and configuration changes to minimize the chance that malicious code or configuration could be introduced into your software pipeline.
- Ensure that your CI/CD pipeline has proper segregation, configuration, and access control to ensure the integrity of the code flowing through the build and deploy processes.
- Ensure that unsigned or unencrypted serialized data is not sent to untrusted clients without some form of integrity check or digital signature to detect tampering or replay of the serialized data.

A09:2021-Security Logging and Monitoring Failures

Security Logging and Monitoring Failures relates more to the process in which active breaches are detected, escalated, and responded to. If proper monitoring and logging are not performed, then breaches cannot be detected. For Time To Study, this means that once our application is live, we must set up the security logging properly (including the determination on what should be logged, how long it should be kept, etc.) and properly monitoring the application so that a breach can be detected and responded to accordingly. It is also important that logging and alerting events are not visible to a user or an attacker, which would fit into the Broken Access Control vulnerability.

Some examples of insufficient logging, detection, monitoring, and active response are:

- Auditable events, including logins, failed logins, and high-value transactions, are not logged.
- Warnings and errors generate no, inadequate, or unclear log messages.
- Logs of applications and APIs are not monitored for suspicious activity.
- Logs are only stored locally.
- Appropriate alerting thresholds and response escalation processes are not in place or effective.
- Penetration testing and scans by dynamic application security testing tools do not trigger alerts.

- The application cannot detect, escalate, or alert for active attacks in real-time, or near real-time.

A10:2021-Server-Side Request Forgery (SSRF)

SSRF flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URL. This vulnerability allows attackers to make the application send a crafted response to an unexpected destination, bypassing firewalls, VPNs, or other types of network access control lists. Although SSRF is at the bottom of the list, it is becoming more common, as modern web applications provide end-users with more convenient features, fetching a URL becomes more common. This problem becomes more severe due to cloud services becoming more common, and architectures becoming more complex.

Some ways to prevent this vulnerability are to not deploy security relevant services on front systems, and control local traffic on these systems, as well as use network encryption on frontends with dedicated and manageable user groups. However, the primary ways to prevent SSRF involve the Network and Application layers. For the Network layer, remote resource access functionality must be segmented in separate networks to reduce the impact of SSRF. In addition, “deny by default” firewall policies or network access control rules need to be enforced to block all but essential intranet traffic. For the Application layer, the following should be performed:

- Sanitize and validate all client-supplied input data.
- Enforce the URL schema, port, and destination with a positive allow list.
- Do not send raw responses to clients
- Disable HTTP redirections.
- Be aware of the URL consistency to avoid attacks such as DNS rebinding and TOCTOU race conditions.
- Do not mitigate SSRF via the use of a deny list or regular expression. Attackers have payload lists, tools, and skills to bypass deny lists.

Conclusion

In conclusion, the OWASP Top Ten provides a great starting point for securing web applications including Time To Study. However, these vulnerabilities oftentimes cannot be fixed by a simple one-time solution, and instead require constant monitoring and a commitment to security. Just because they are the most observed vulnerabilities, does not mean they are the only vulnerabilities, or that a crafty attacker would be able to find other methods to accomplish their goals. Therefore, we need to be aware of the necessary actions needed to both prevent and respond to security incidents throughout the entire lifecycle of Time To Study. Although that might start with the OWASP Top Ten, it should not end there. Furthermore, placement in the list does not make the vulnerability any more or less worse, it does not matter if we tackle and mitigate the first nine vulnerabilities if our application is still vulnerable to SSRF attacks. However, increased awareness of these vulnerabilities is the first step to mitigating them, which is the purpose of the OWASP Top Ten. Now that we are aware of these vulnerabilities, we can see trends forming, and not only secure ourselves now, but potentially secure ourselves from new threats in the future.