Hacking Web Sites
OWASP Top 10

November 10, 2023

Emmanuel Benoist | BFH-TI

Web Security: Overview of other security risks

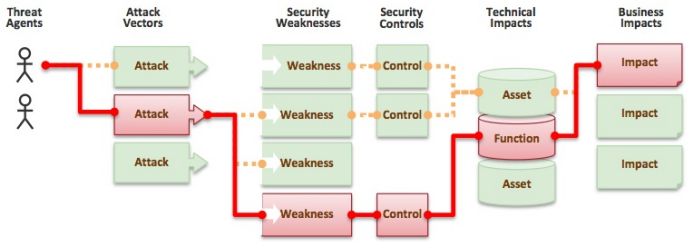
OWASP Top 10

OWASP Top 10

- **10 most critical security risks for web applications**
- **Goal**
  - Raise awareness of people about application security
- **Based on real examples**
  - 8 datasets from 7 firms specialized in application security
  - 500'000 vulnerabilities, thousands of applications
  - Sorted on the prevalence of data in combination with risks (exploitability, detectability and impact estimation)

What are application security risks?

- **Attackers can use many different paths to do harm**



Top 10 Web Security Risks

OWASP Top 10

- **Presents the 10 most critical web application security risks**
  - Produced by the Open Web Application Security Project (OWASP)
  - Available on line `www.owasp.org`
  - Updated in `2021`
- **Not Exhaustive**
  - hundreds of other issues occure in Web Security
  - But it is foccused on the most critical ones

OWASP Top 10

Version 2017

- **A1:2017 - Injection**
- **A2:2017 - Broken Authentication**
- **A3:2017 - Sensitive Data Exposure**
- **A4:2017 - XML External Entities (XXE)**
- **A5:2017 - Broken Access Control**
- **A6:2017 - Security Misconfiguration**
- **A7:2017 - Cross-Site Scripting (XSS)**
- **A8:2017 - Insecure Deserialization**
- **A9:2017 - Using components with known vulnerabilities**
- **A10:2017 - Insufficient Logging and Monitoring**

OWASP Top 10

Version 2021

- **A1:2021 - Broken Access Control** Seen
- **A2:2021 - Cryptographic Failures** Seen
- **A3:2021 - Injection** Seen
- **A4:2021 - Insecure design**
- **A5:2021 - Security Misconfiguration**
- **A6:2021 - Vulnerable and Outdated Components**
- **A7:2021 - Identification and Authentication Failures** Seen
- **A8:2021 - Software and Data integrity Failures**
- **A9:2021 - Security Logging and Monitoring Failures**
- **A10:2021 - Server-Side Request Forgery**

A4:2021 Insecure design

A4:2021 Insecure design

- **risks related to design and architectural flaws,**
  - call for more use of threat modeling,
  - secure design patterns,
  - and reference architectures.
- **Move beyond "Shift-left" in the coding space to pre-code activities**
  - Build applications "Secure by Design".
- **Notable Common Weakness Enumerations (CWEs)**
  - CWE-209: Generation of Error Message Containing Sensitive Information,
  - CWE-256: Unprotected Storage of Credentials,
  - CWE-501: Trust Boundary Violation, and
  - CWE-522: Insufficiently Protected Credentials.

Insecure design is a broad category

- **Difference between insecure design and insecure implementation.**
  - Difference between design flaws and implementation defects:
  - A secure design can still have implementation defects leading to vulnerabilities that may be exploited.
  - An insecure design cannot be fixed by a perfect implementation.
- **One common factor : the lack of business risk profiling**
  - Implies failure to determine what level of security design is required

Examples of Attack Scenarios

- **Scenario #1 : Questions and Answers**
  - Password recovery workflow includes “questions and answers”
  - Is prohibited (NIST 800-63v, OWASP ASVS, OWASP Top 10)
  - Questions and answers can not show an identity, since many people may know the answer.
  - Such code should be removed and replaced with a more secure design.
- **Scenario #2 : Cinema**
  - A cinema chain allows group booking discounts and has a maximum of fifteen attendees before requiring a deposit.
  - Attackers could book six hundred seats and all cinemas at once in a few requests, causing a massive loss of income.
- **Scenario #3 : High End Video Cards**
  - A retail chain’s e-commerce website does not have protection against bots run by scalpers buying high-end video cards to resell auction websites.
  - Normal users can not obtain cards at any price.
  - Careful anti-bot design and domain logic rules, such as purchases made within a few seconds of availability, might identify inauthentic purchases and rejected such transactions.

Secure Design

- **Requirements and Resource Management**
- **Secure Design**
- **Secure Development Lifecycle**

How to Prevent

- Establish and use a secure development lifecycle with AppSec professionals to help evaluate and design security and privacy-related controls
- Establish and use a library of secure design patterns or paved road ready to use components
- Use threat modeling for critical authentication, access control, business logic, and key flows
- Integrate security language and controls into user stories
- Integrate plausibility checks at each tier of your application (from frontend to backend)
- Write unit and integration tests to validate that all critical flows are resistant to the threat model. Compile use-cases and misuse-cases for each tier of your application.
- Segregate tier layers on the system and network layers depending on the exposure and protection needs
- Segregate tenants robustly by design throughout all tiers
- Limit resource consumption by user or service

A5:2021 Security Misconfiguration

A5:2021 Security Misconfiguration

- **90% of applications were tested for some form of misconfiguration**
  - With more shifts into highly configurable software,
  - it's not surprising to see this category move up.
- **Notable CWEs**
  - CWE-16 Configuration and
  - CWE-611 Improper Restriction of XML External Entity Reference

Description

The application might be vulnerable if the application is:

-
- Missing appropriate security hardening across any part of the application stack or improperly configured permissions on cloud services.
- Unnecessary features are enabled or installed (e.g., unnecessary ports, services, pages, accounts, or privileges).
- Default accounts and their passwords are still enabled and unchanged.
- Error handling reveals stack traces or other overly informative error messages to users.
- For upgraded systems, the latest security features are disabled or not configured securely.
- The security settings in the application servers, application frameworks (e.g., Struts, Spring, ASP.NET), 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 A06:2021-Vulnerable and Outdated Components).

Without a concerted, repeatable application security configuration process, systems are at a higher risk.

Examples of Attacks

- **Scenario #1:**
  - The application server comes with sample applications not removed from the production server.
  - These sample applications have known security flaws attackers use to compromise the server.
  - Suppose one of these applications is the admin console, and default accounts weren't changed.
  - In that case, the attacker logs in with default passwords and takes over.
- **Scenario #2:**
  - Directory listing is not disabled on the server. An attacker discovers they can simply list directories.
  - The attacker finds and downloads the compiled Java classes, which they decompile and reverse engineer to view the code.
  - The attacker then finds a severe access control flaw in the application.

Examples of Attacks (Cont.)

- **Scenario #3:**
  - The application server's configuration allows detailed error messages, e.g., stack traces, to be returned to users.
  - This potentially exposes sensitive information or underlying flaws such as component versions that are known to be vulnerable.
- **Scenario #4:**
  - A cloud service provider (CSP) has default sharing permissions open to the Internet by other CSP users.
  - This allows sensitive data stored within cloud storage to be accessed.

How to Prevent

Secure installation processes should be implemented, including:

- A repeatable hardening process makes it fast and easy to deploy another environment that is appropriately locked down. Development, QA, and production environments should all be configured identically, with different credentials used in each environment. This process should be automated to minimize the effort required to set up a new secure environment.
- A minimal platform without any unnecessary features, components, documentation, and samples. Remove or do not install unused features and frameworks.
- A task to review and update the configurations appropriate to all security notes, updates, and patches as part of the patch management process (see A06:2021-Vulnerable and Outdated Components). Review cloud storage permissions (e.g., S3 bucket permissions).
- A segmented application architecture provides effective and secure separation between components or tenants, with segmentation, containerization, or cloud security groups (ACLs).
- Sending security directives to clients, e.g., Security Headers.
- An automated process to verify the effectiveness of the configurations and settings in all environments.

A6:2021 Vulnerable and Outdated Components

A6:2021 Vulnerable and Outdated Components

- **Vulnerable Components are a known issue**
- **Notable CWEs**
  - CWE-1104 Use of Unmaintained Third Party Components
  - CWE-937 OWASP Top 10 2013: Using Components with Known Vulnerabilities
  - CWE-1035 2017 Top 10 A9: Using Components with Known Vulnerabilities

Description

You are likely vulnerable:

- If you do not know the versions of all components you use (both client-side and server-side). This includes components you directly use as well as nested dependencies.
- If the software is vulnerable, unsupported, or out of date. This includes the OS, web/application server, database management system (DBMS), applications, APIs and all components, runtime environments, and libraries.
- If you do not scan for vulnerabilities regularly and subscribe to security bulletins related to the components you use.
- If you do not fix or upgrade the underlying platform, frameworks, and dependencies in a risk-based, timely fashion. This commonly happens in environments when patching is a monthly or quarterly task under change control, leaving organizations open to days or months of unnecessary exposure to fixed vulnerabilities.
- If software developers do not test the compatibility of updated, upgraded, or patched libraries.
- If you do not secure the components' configurations (see A05:2021-Security Misconfiguration).

Example Attack Scenarios

- **Scenario #1:**
  - Components typically run with the same privileges as the application itself, so flaws in any component can result in serious impact.
    Such flaws can be accidental (e.g., coding error) or intentional (e.g., a backdoor in a component).
    Some example exploitable component vulnerabilities discovered are:
    - CVE-2017-5638, a Struts 2 remote code execution vulnerability that enables the execution of arbitrary code on the server, has been blamed for significant breaches.
    - While the internet of things (IoT) is frequently difficult or impossible to patch, the importance of patching them can be great (e.g., biomedical devices).
  - There are automated tools to help attackers find unpatched or misconfigured systems.
    For example, the Shodan IoT search engine can help you find devices that still suffer from Heartbleed vulnerability patched in April 2014.

How to prevent

There should be a patch management process in place to:

- Remove unused dependencies, unnecessary features, components, files, and documentation.
- Continuously inventory the versions of both client-side and server-side components (e.g., frameworks, libraries) and their dependencies using tools like versions, OWASP Dependency Check, retire.js, etc.
  Continuously monitor sources like Common Vulnerability and Exposures (CVE) and National Vulnerability Database (NVD) for vulnerabilities in the components.
  Use software composition analysis tools to automate the process.
  Subscribe to email alerts for security vulnerabilities related to components you use.
- Only obtain components from official sources over secure links.
  Prefer signed packages to reduce the chance of including a modified, malicious component (See A08:2021-Software and Data Integrity Failures).
- Monitor for libraries and components that are unmaintained or do not create security patches for older versions.
  If patching is not possible, consider deploying a virtual patch to monitor, detect, or protect against the discovered issue.

A8:2021 Software and Data integrity Failures

A8:2021 Software and Data integrity Failures

- **A new category for 2021 focuses on making assumptions related to software updates, critical data, and CI/CD pipelines without verifying integrity.**
  - One of the highest weighted impacts from Common Vulnerability and Exposures/Common Vulnerability Scoring System (CVE/CVSS) data.
-
- **Notable Common Weakness Enumerations (CWEs)**
  - CWE-829: Inclusion of Functionality from Untrusted Control Sphere,
  - CWE-494: Download of Code Without Integrity Check, and
  - CWE-502: Deserialization of Untrusted Data.

Descriptions

- **Software and data integrity failures relate to code and infrastructure that does not protect against integrity violations.**
- **An example of this is where an application relies upon**
  - plugins, libraries, or modules
  - from untrusted sources, repositories, and content delivery networks (CDNs).
- **An insecure CI/CD pipeline can introduce the potential for unauthorized access, malicious code, or system compromise.**
  - Lastly, many applications now include auto-update functionality, where 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.
  - Another example is where objects or data are encoded or serialized into a structure that an attacker can see and modify is vulnerable to insecure deserialization.

Example Attack Scenarios

- **Scenario #1 Update without signing:**
  - Many home routers, set-top boxes, device firmware, and others do not verify updates via signed firmware.
  - Unsigned firmware is a growing target for attackers and is expected to only get worse.
  - This is a major concern as many times there is no mechanism to remediate other than to fix in a future version and wait for previous versions to age out.
- **Scenario #2 SolarWinds malicious update:**
  - Nation-states have been known to attack update mechanisms, with a recent notable attack being the SolarWinds Orion attack.
  - The company that develops the software had secure build and update integrity processes.
  - Still, these were able to be subverted, and for several months, the firm distributed a highly targeted malicious update to more than 18,000 organizations, of which around 100 or so were affected.
  - This is one of the most far-reaching and most significant breaches of this nature in history.

Example Attack Scenarios (Cont.)

- **Scenario #3 Insecure Deserialization:**
  - A React application calls a set of Spring Boot microservices.
  - Being functional programmers, they tried to ensure that their code is immutable.
  - The solution they came up with is serializing the user state and passing it back and forth with each request.
  - An attacker notices the "rOo" Java object signature (in base64) and uses the Java Serial Killer tool to gain remote code execution on the application server.

How to Prevent

- 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. If you have a higher risk profile, consider hosting an internal known-good repository that's vetted.
- 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

A9:2021 Security Logging and Monitoring Failures

A9:2021 Security Logging and Monitoring Failures

- **Ensure all login, access control failures, and server-side input validation failures can be logged with sufficient user context**
  - Should be used to identify suspicious or malicious accounts, and held for sufficient time to allow delayed forensic analysis.
- **Ensure that logs are generated in a format that can be easily consumed by a centralized log management solutions.**
- **Ensure high-value transactions have an audit trail with integrity controls to prevent tampering or deletion, such as append-only database tables or similar.**
- **Establish effective monitoring and alerting such that suspicious activities are detected and responded to in a timely fashion.**

A10:2021 Server-Side Request Forgery

A10:2021 Server-Side Request Forgery

- **The data shows a relatively low incidence rate**
  - with above average testing coverage
  - and above-average Exploit
  - and Impact potential ratings.
- **Description**
  - SSRF flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URL.
  - It allows an attacker to coerce the application to send a crafted request to an unexpected destination,
  - even when protected by a firewall, VPN, or another type of network access control list (ACL).

Example Attack Scenarios

- **Scenario #1 : Port scan internal servers**
  - If the network architecture is unsegmented,
  - attackers can map out internal networks and determine if ports are open or closed on internal servers
  - from connection results or elapsed time to connect or reject SSRF payload connections.
- **Scenario #2 : Sensitive data exposure**
  - Attackers can access local files or internal services to gain sensitive information
  - such as file:///etc/passwd
  - and http://localhost:28017/.
- **Scenario #3 : Access metadata storage of cloud service**
  - Most cloud providers have metadata storage
  - such as http://169.254.169.254/.
  - An attacker can read the metadata to gain sensitive information.
- **Scenario #4 : Compromise internal services**
  - The attacker can abuse internal services to conduct further attacks such as Remote Code Execution (RCE) or Denial of Service (DoS).

How to Prevent

- **From Network layer**
  - Segment remote resource access functionality in separate networks to reduce the impact of SSRF
  - Enforce “deny by default” firewall policies or network access control rules to block all but essential intranet traffic.
- **From Application layer:**
  - 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 “time of check, time of use” (TOCTOU) race conditions

  Do not use Black lists.
- **Additional Measures to consider:**
  - Don’t deploy other security relevant services on front systems (e.g. OpenID). Control local traffic on these systems (e.g. localhost)
  - For frontends with dedicated and manageable user groups use network encryption (e.g. VPNs) on independent systems to consider very high protection needs

Conclusion

Conclusion

- **Web Security belongs to security**
  - Encryption,
  - Testing of inputs
  - Teaching of users
- **It is somehow different**
  - Restricted entrypoint port 80 (may be more easy to protect)
  - Open infrastructure (anybody can visit and attack)
  - International Architecture
  - No control on the client

References

- **OWASP Top 10, The ten most critical Web Application Security Vulnerabilities, 2013 Update, 2017 and 2021** http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
- **RSnake, "What is CSRF?"** http://ha.ckers.org/blog/20061030/what-is-csrf/
- **OWASP CSRF Prevention Cheat Sheet** https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29_Prevention_Cheat_Sheet
- OWASP A04:2021 Insecure Design
  https://owasp.org/Top10/A04_2021-Insecure_Design/
- OWASP A05:2021 Security Misconfiguration
  https://owasp.org/Top10/A05_2021-Security_Misconfiguration/
- OWASP A08:2021 Software and Data Integrity Failures
  https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/