TRAINING KIT – CODE1 bis

Commons vulnerabilties presentation

EXCELLIUM

Your first call when it comes to IT and Security!

March 16, 2017

TLP: AMBER

Agenda

- Objective
- What is OWASP and why OWASP TOP 10 ?
- For each OWASP TOP 10 point:
  - Description
  - Vulnerability demonstration
  - How to avoid it
- Questions & Answers

Objective

This session has for objective to show to you the most common vulnerabilities meet into web application.

The presentation will also explains how to avoid them at code level.

OWASP / OWASP Top 10?

OWASP stands for Open Web Application Security Project.

It's a non-profit foundation providing open/ free resources about Security:

- Wiki / cheat sheets : https://www.owasp.org
- Tools : OWASP ZAP, OWASP ESAPI…
- Guide : OWASP ASVS, OWASP Open SAMM…

Why OWASP TOP 10?

OWASP Top 10 is a document describing the most commons vulnerabilities found into web application:

- A1 *Injection*
- A2 *Broken Authentication and Session Management*
- A3 *Cross-Site Scripting (XSS)*
- A4 *Insecure Direct Object References*
- A5 *Security Misconfiguration*
- A6 *Sensitive Data Exposure*
- A7 *Missing Function Level Access Control*
- A8 *Cross-Site Request Forgery (CSRF)*
- A9 *Using Components with Known Vulnerabilities*
- A10 *Unvalidated Redirects and Forwards*

Injection:

Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query.

The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

Demonstration:

Step 1) Perform injection using OS Command and XML.

Step 2) Analyze possible security check.

To result, how to avoid exposure to this vulnerability:

Always validate any piece of data received by a user (parameter/cookie/header) :

- Check according to a allowed list of characters (white list),
- Check the number of repetition of each character according to a list of characters allowed to appear in sequence,
- Check min and max length if the target type is a string,
- Check min and max size if the target type is a number,
- Check if input is encoded using the expected encoding used by the application,
- In case of file uploaded, verify that the file content is one of expected (not rely only on MIME type provided by the development framework used).

Injection:

Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities.

Demonstration:

Step 1) Perform authentication bypass using session side jacking

Step 2) Analyze possible security check.

To result, how to avoid exposure to this vulnerability:

- Use session management and authentication features provided by the development framework or platform used,
- Use cookie to exchange session identifier with the client,
- Configure cookie with ‘httpOnly’ and ‘secure” flags enabled,
- Restrict the attributes ‘path’ and ‘domain’ of the cookie to the most nearest possible values of the domain and context root of the application,
- Do not let user use weak password,
- Use account locking in order to prevent brute forcing,
- Do not allow (technically) user to have more than one session opened with the same account at the same time,
- Set expiration time of the session to 15 or 30 minutes max,
- Do not store authentication or authorization information in cookie,
- Do not give any failure details in case of login failure, but, trace event.

Injection:

XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

Demonstration:

Step 1) Perform XSS injection.

Step 2) Analyze possible security check.

To result, how to avoid exposure to this vulnerability:

- In complements to input data validation, data used to build response sent to client (browser) must be escaped in order to avoid execution of injected code on client side (JavaScript, VBScript, HTML, CSS,...).

- If execution of code provided by client must be executed on client side (business requirement) then filtering must be applied on code submitted in order to limit range of allowed client code functionalities.

- In general, it's recommended (and more secure) to avoid execution of code directly received from the client...

Injection:

A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.

Demonstration:

Step 1) Access data using guessing on ID.

Step 2) Analyze possible security check.

To result, how to avoid exposure to this vulnerability:

- Hide object real identifier to end user by using a random ID mapped with the object real ID. The mapping table must be stored on server side (for example in user session as hash table object),

- Random ID must not have any link into is content with the real ID (no derivation),

- UUID feature provided by most of development platforms can be used an random ID:
  - http://www.php.net/manual/en/function.uniqid.php
  - http://msdn.microsoft.com/en-us/library/system.guid.newguid.aspx
  - http://docs.oracle.com/javase/7/docs/api/java/util/UUID.html
  - http://www.ruby-doc.org/stdlib-1.9.3/libdoc/securerandom/rdoc/SecureRandom.html

Injection:

Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. Secure settings should be defined, implemented, and maintained, as defaults are often insecure. Additionally, software should be kept up to date.

Demonstration:

Step 1) Play with the parameter “template”.

Step 2) Analyze possible security check.

To result, how to avoid exposure to this vulnerability:

- Document, in details, each configuration settings of your application and infrastructure (even the default already present that will be used).
- Disable every features or services not used in your framework or your infrastructure,
- Change all default account (login + password),
- Do not let any technical information about your system (web server version, os, framework,...) reach the client side,
- Keep software up to date according to community or vendor patch ☺,
- Use only third party provided by trusted source and ask a security review to your company security folks before to use them ☺

Injection:

Many web applications do not properly protect sensitive data, such as credit cards, tax IDs, and authentication credentials. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.

Demonstration:

Step 1) Obtain the client Luxembourgish SSN.

Step 2) Analyze possible security check.

To result, how to avoid exposure to this vulnerability:

- Use cryptographic standard algorithms: Do not use your own algorithm unless you are expert in cryptography and you have invented a new killer algorithm that has been defined as new standard,
- Same remark about hashing algorithms,
- Do not forget to protect also data in transit,
- Don’t store sensitive data unnecessarily. Discard it as soon as possible,
- Disable autocomplete on forms collecting sensitive data and disable caching for pages that contain sensitive data,
- Monitor any access to sensitive information,
- Do not use encoding to protect your data: Encoding is not ciphering

Injection:

Most web applications verify function level access rights before making that functionality visible in the UI. However, applications need to perform the same access control checks on the server when each function is accessed. If requests are not verified, attackers will be able to forge requests in order to access functionality without proper authorization.

Demonstration:

Step 1) Run demo page.

Step 2) Analyze possible security check.

To result, how to avoid exposure to this vulnerability:

- Apply authorization on server side, do not rely on client side to apply them,
- Take authorization decision based on server side information,
- The enforcement mechanism(s) should deny all access by default, requiring explicit grants to specific roles for access to every function,
- It's not because the link/url to a feature is not displayed on client that the user cannot discover and/or invoke it ☺
- Check for authorization before to enter into feature (already seen during security code review ☺),
- Deny access if error occur (business or technical error),
- Trace any access denied decision taken.

Injection:

A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.

Demonstration:

Step 1) Run demo page.

Step 2) Analyze possible security check.

To result, how to avoid exposure to this vulnerability:

- In HTML form, you can use a hidden parameter with a random name and value both unpredictable (this parameter is called a Token):
- Token must be valid only one time and for a limited period of time,
- Token must be kept in user session in order to validate is presence at next HTTP request made by user,
- Name and value must be alphanumeric with a min size to 50 characters,
- Prefer use of this token as a form of hidden HTML form parameter (request body) instead of query string parameter (request url) in order to avoid that this token be exposed in logs of all network elements taking part of the network communication,
- If token use is not possible then requiring the user to reauthenticate, or prove they are a user (e.g., via a CAPTCHA) can also protect against CSRF.

Injection:

Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts.

Examples:

http://seclists.org/fulldisclosure/2016/Feb/81

https://www.exploit-db.com/exploits/39919/

To result, how to avoid exposure to this vulnerability:

- Keep third party components, server, API, Framework… up to date in term of security patch ☺
- Analyze them regularly using tools like “OWASP Dependency Check”
- Consult sites below to check if a third party item contains known vulnerabilities:
  - https://web.nvd.nist.gov/view/vuln/search?execution=e2s1
  - http://seclists.org/fulldisclosure/

Injection:

Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

Demonstration:

Step 1) Run demo page.

Step 2) Analyze possible security check.

To result, how to avoid exposure to this vulnerability:

- Validate input like for A1 vulnerability,
- Simply avoid using redirects and forwards,
- If destination parameters can't be avoided, ensure that the supplied value is valid, and authorized for the user. It is recommended that any such destination parameters be a mapping value, rather than the actual URL or portion of the URL, and that server side code translate this mapping to the target URL.
- If it is required that the user enter a portion of url or path then validate that the canonical version of the url/path received is into the expected domain/folder,
- For portion of path, in order to limit attacker action scope in case of vulnerability :
- For Unix / Linux OS use "chroot" feature,
- For Windows OS, store site files on another drive than System drive.

Q&R