Testing for vulnerabilities

Web vulnerabilities

INTRODUCTION

- Most significant vulnerabilities occur in web applications
- Web vulnerabilities have a significant impact
  - Can give unauthorized access to the vulnerable server
  - Confidential information can be lost
- There are a lot of vulnerabilities specific to web applications
  - We will discuss some of them

REMOTE CODE EXECUTION

- Allow code execution on the vulnerable server
  - Usually as a limited user (www-data, apache), but it's not a rule
- They lead to the compromise of the entire server
- They are often present due to programming errors in PHP, ASP, Java, etc.
- Examples:
  - register_globals in PHP
    - Allows setting any global variable
  - XMLRPC
    - Allows passing unvalidated input to the eval() function

REMOTE CODE EXECUTION

- Local File Inclusion
  - Allow inclusion and execution of a local file
  - We can inject code into an apache log and then include it
- Remote File Inclusion
  - Allow the inclusion and execution of a remote file
- Other application-specific vulnerabilities
- Mitigations
  - Disabling remote file inclusion in PHP
  - Avoid including files based on user input
  - Disabling super-globals in PHP
  - Sanitization of the input

CROSS-SITE SCRIPTING (XSS)

- When the user's input is displayed without filtering/validating
- Exploits the trust that the client has on the server
- Scripts can be entered as input
  - For example, in the comments section of a forum
- If we run JavaScript code in the victim's browser, we can steal the cookie and hijack the session
- XSS:
  - Persistent – the injected script is stored on the server and displayed on each access
  - Reflected – the input is returned by the server in response, without validations
- Mitigations
  - Input sanitization – special attention to characters such as "<", ">", "/", etc.

CROSS-SITE REQUEST FORGERY

- One-click attack, session-riding or CSRF
- Exploits the trust that the server has in the client
- Force an authenticated user to perform unwanted operations in the web application
  - By social engineering
  - By sending emails that have embedded malicious requests
- Unlike XSS, where data theft is tracked, CSRF tracks state modification
  - Example: changed credentials, transferred money, etc.
- Mitigations
  - Validating the source of the request
  - Using a CSRF token
  - Re-authentication
  - CAPTCHA

DEFAULT LOGIN

- The web application uses default credentials
- Usually this happens due to misconfiguration
  - The admin should disable any form of default credentials
- Common strong examples:
  - admin:blank
  - admin:admin
  - admin:root
- It is easy to find out these default credentials by reading the web application manual
  - http://www.routerpasswords.com
- Mitigations
  - Disable all default accounts
  - Use of strong usernames and passwords

WEAK SESSION MANAGEMENT

- Use of a cookie or session ID that is easy to guess/predict
- Examples:
  - Session ID of only 16 bits
  - Cookie that uses only a predictable timestamp/hash
  - The attacker only needs to brute-force to discover the session ID
- Mitigations
  - Use of long IDs (128-256 bits)
  - Using random IDs

DIRECTORY TRAVERSAL

- Insufficient input validation – special characters are not filtered out
  - .., /, \
- By passing an input with such characters, we can gain access to the entire file-system
  - Whether the input is treated as a path or file name
- With such a vulnerability, we can get:
  - Info leaks (paths, existing files, etc.)
  - Code execution (local-file inclusion)
- Mitigations
  - Do not use user input in file paths
  - Use of IDs instead of explicit names in input
  - Sanitization of the input (delete unwanted characters)
  - Chroot Jails

SESSION FIXATION

- Pin session ID
  - Either by the attacker or by the server
- The attacker can send a link with the previously pinned ID to the victim
  - The session ID must be submitted in the request
- Once logged in, the session will be assigned the pinned ID
- The attacker can use the pinned ID to access the victim's account
- Mitigations
  - Re-generation of the session ID after each request
  - Randomly generate session ID, regardless of user input

User enumeration

- Some web applications explicitly specify if the user-name is wrong
  - "invalid user-name"
  - "no such user"
- The moment we find an existing user, the message changes:
  - "wrong password"
- A dictionary-based attack can be performed to enumerate web application users
- Once users are obtained, brute-force attacks can be carried out to determine their passwords
- Mitigations
  - DO NOT specify whether the user exists or not; Use generic messages
    - "Incorrect user-name or password"
  - Add a small timeout when entering invalid credentials, to avoid this attack (along with brute-force)
    - Show the error message after 5 seconds, for example

FRAME INJECTION

- It's not necessarily a web vulnerability
  - But it has web applications as a vector
  - Any web application can be compromised
- Usually, it occurs because of the compromise of the web-server
  - The attacker injects invisible frames or iframes, which load malicious pages
- It can also occur because of a local attack
  - Malware intercepts web traffic, and injects additional fields into HTML pages

Conclusions

- Web applications can present a very large number of vulnerabilities
- In the pentest process, web applications are the most interesting
- Not all vulnerabilities lead to RCE, but some can be extremely useful