A learning path for Broken Access Control Vulnerabilities

Broken access control was ranked as 1st in the 2021 OWASP top 10 list.
Over 500,000 applications were tested for some form of broken access control with the average incidence rate of 3.81% with a maximum of 55.97% and have the most occurrences in the contributed dataset with over 318k. [1]

Intermediate Level

Access control policies grant or restrict users' rights on the application ensuring that users cannot act outside their intended permissions.
There are various different possibilities for a server to provide or deny rights to request a certain resource.
These can be, for example, session cookies or JWT (JSON web token) token. When the user requests access to a resource, the server checks if the user can access that resource before executing the request.
Potential problems arise when we start realizing that developers have to secure every single endpoint with every possible HTTP method.
Broken access control vulnerabilities are sadly common in modern applications. One of the causes of this rising problem is in the way that applications are created. Applications become more and more complex and with a lot of different components that interact with each other. Managing this level of complexity and interaction has become more and more challenging for developers.
Common access control vulnerabilities include: [1]

1. Violation of the principle of least privilege or deny by default, where access should only be granted for particular capabilities, roles or users, but is available to everyone.

2. Bypassing access control checks by modifying the URL, internal application state, or the HTML page, or by using an attack tool to modify API requests.
3. Permitting viewing or editing someone else's account, by providing its unique identifier (insecure direct object references)
4. Accessing API with missing access controls for POST, PUT and DELETE
5. Elevation of privilege. Acting as a user without being logged in or acting as an admin when logged in as a user.
6. Force browsing to authenticated pages as an unauthenticated user or to privileged pages as a standard user.

A Cross Site Request Forgery is an attack that tricks the victim into submitting a malicious HTTP request to a target destination without their knowledge or intent, in order to perform an action disguised as the victim.
CSRF has primarily been used to perform an action against a target site using the victim's privileges or to disclose information by gaining access to the response.
For most sites, browser requests automatically include any credentials associated with the site, such as the user's session cookie, IP address and so forth. Therefore, if the user is currently authenticated to the site, the site will have no way to distinguish between the forged malicious request and the legitimate request sent by the victim.
Since the attacker has the identity of the victim, the scope of CSRF is limited only by the victim's privileges.
Sometimes it is possible to store the CSRF attack on the vulnerable site itself, this is called "Stored CSRF". It can be accomplished by storing an IMG or IFRAME tag in a field that accepts HTML. If the attack can store a CSRF in the site, the severity of the attack is amplified.

Usually, this kind of attack requires two steps: building an exploit URL or script and then finding a way to trick the victim into executing the action. This will usually happen with the help of a little bit of social engineering.
Or otherwise, an attacker can use a tag, such as the IMG tag, avoiding the step requiring the user to click on the malicious link. Suppose the attacker sends the user an email inducing him to visit a URL referring to a page containing also an IMG tag.

The tag will look something like this <img src="https://www.company.example/action" width="0" height="0">. When the browser displays this page, it will try to display the invisible (zero dimension) image as well. This results in a request being automatically sent to the web application. The request will trigger the action specified in the src field. This will happen if the browser doesn't have the image download blocked. Since usually browsers allow the image to be downloaded and displayed, this method will most likely work.

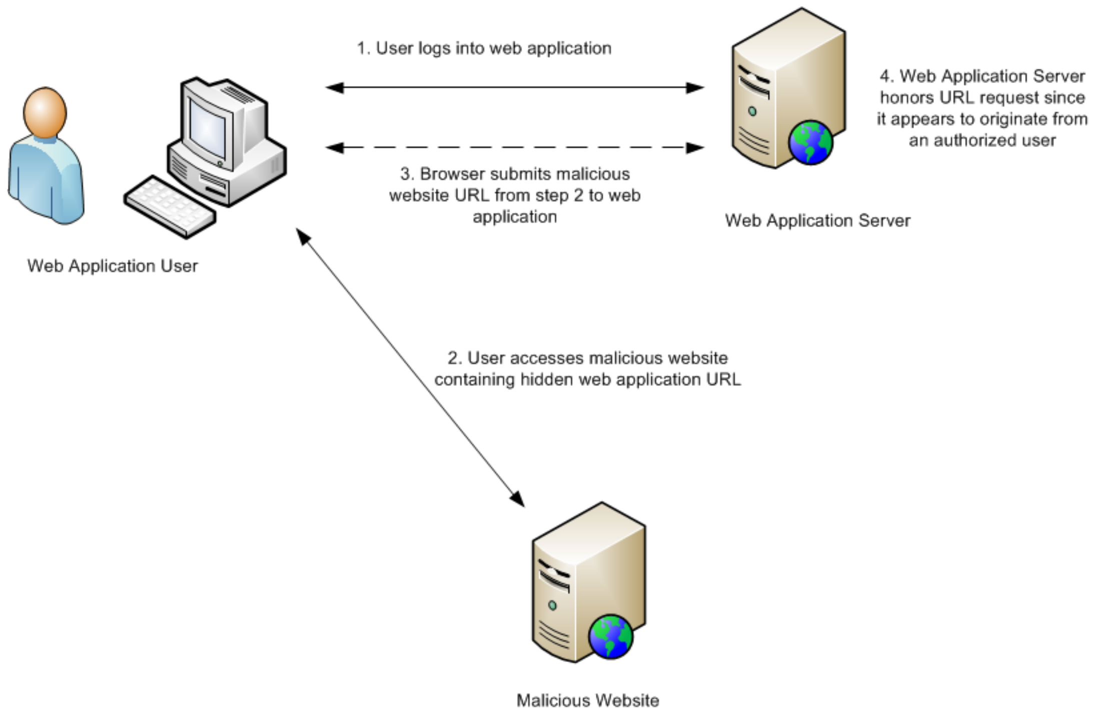
1. User logs into web application
4. Web Application Server honors URL request since it appears to originate from an authorized user
3. Browser submits malicious website URL from step 2 to web application
Web Application Server
Web Application User
2. User accesses malicious website containing hidden web application URL
Malicious Website

Fig. 2.3 CSRF Schema [2]

The attack involves the following sequence of steps:

1. The victim user logs into the trusted site using his/her username and password, and thus creates a new session.

The trusted site stores the session identifier for the session in a cookie in the victim user's web browser.

2. The victim user visits a malicious site.

3. The malicious site web page sends a request to the trusted site from the victim user's browser.

The web browser will automatically add the session cookie to the malicious request because it is targeted for the trusted site.
4. The trusted site, if vulnerable to CSRF, will process the malicious request forged by the attacker web site.

To sum up we can say that CSRF relies on:

- Web browser behaviour regarding the handling of session-related information such as cookies and HTTP authentication information. (If the victim is already authenticated, submitting another request causes the cookie to be automatically sent with it)
- An attacker's knowledge of valid web application URLs, requests or functionality.
- Application session management relying only on information known by the browser.

Existence of HTML tags whose presence cause immediate access to an HTTP resource (for example the image tag) will facilitate the exploitation.

Bibliography

[1] «A01:2021 – Broken Access Control,» [Online]. Available: https://owasp.org/Top10/A01_2021-Broken_Access_Control. [Consultato il giorno 2023].

[2] [Online]. Available: https://support.ptc.com/help/windchill/Whindchill-FHC/en/FlexPLM_Help_Center/images/WCConsiderSecureInfrastructure_CSRF.png.