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]

Advanced Level

Access control policies grant or restrict users' rights on the application ensuring that users cannot act outside their intended permissions.
Exploitation of access control is a core skill of attackers. Broken access control vulnerabilities are common in modern applications since the design and implementation of access control mechanisms rely on a highly complex ecosystem of multiple components and processes.
It is also due to the lack of automated detection and lack of effective functional testing by application developers.

If a user can gain access to functionality that he is not permitted to access, this is called Vertical Privilege Escalation.
At its most basic, Vertical Privilege Escalation arises where an application does not enforce any protection over sensitive functionality. Administrative functions might be directly accessible from an administrator's page but not from a user's page. However, a user might simply be able to access the administrative functions by browsing directly to the relevant admin URL.
The admin URL could be disclosed somewhere in the application, but attackers could be able to access the page anyway by brute-forcing the application with a wordlist.

Simply hiding sensitive functionalities do not provide effective access control, since users might still discover the obfuscated URLs in various ways.
Horizontal privilege escalation arises when a user is able to gain access to resources belonging to another user, instead of their own resources of that type.
Often, a horizontal privilege escalation attack can be turned into a vertical privilege escalation, by compromising a more privileged user. A horizontal escalation might allow an attacker to reset or capture the password belonging to another user. If the attacker targets an administrative user and compromises their account, then they can gain administrative access and so perform vertical privilege escalation. [2]

Remediating access control failure is a rather complex task since attackers can leverage many scenarios to exploit the vulnerability. In addition, misconfiguration of function-level access often results in security gaps used for privilege escalation by attackers.
Thus, there is not an easy solution to this problem, but some good practices can have a positive impact on the application's security.
For example:
- Except for public resources, denied 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.
- Disable web server directory listing and ensure file metadata and backup files are not present within web roots.
- Log access control failures, alert admins when appropriate.
- Rate limit API and controller access to minimize the harm form automated attack tooling.
- Stateful session identifiers should be invalidated on the server after logout. [1]

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.

This attack can happen when a web application does not sufficiently verify whether a valid request was intentionally provided by the user who submitted the request.

Actually, when a web server is designed to receive a request from a client without any mechanism for verifying that it was intentionally sent, then it might be possible for an attacker to trick a client into making an unintentional request to the web server, which will be treated as an authentic request.

This can be achieved in multiple ways and can result in exposure of data or unintended code execution.

The consequences will vary depending on the nature of the functionality that is vulnerable to CSRF. If the victim is an administrator or privileged user, the consequences may include obtaining complete control over the web application.

Different types of requests can be automatically triggered by an attacker. For example, some form fields can be hidden in a html page. If the attacker knows which parameters needs to be sent in the request, he can create a form with hidden fields with the right name and value. The form can be automatically sent via a basic script without requiring the victim user to perform any action.

An interesting fact is that these vulnerabilities can also be accessed behind a firewall. It is in fact sufficient that the attacked link is reachable by the victim and not directly by the attacker. In particular, the victim site can be any intranet web server which is unlikely to be exposed to the internet.

The most robust way to defend against CSRF attacks is to include a CSRF token within relevant requests. This token should be unpredictable with high entropy, tied to the user's session and strictly validated in every case before the relevant action is executed.

An additional defense that is partially effective against CSRF and can be used in conjunction with CSRF tokens is SameSite cookies. By setting this attribute on session cookies an application can prevent the default browser behaviour of automatically adding cookies to requests regardless of where they originate.

Bibliography

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

[2] «Access control vulerabilities and privilege escalation,» [Online]. Available: https://portswigger.net/web-security/access-control. [Consulted 2023].