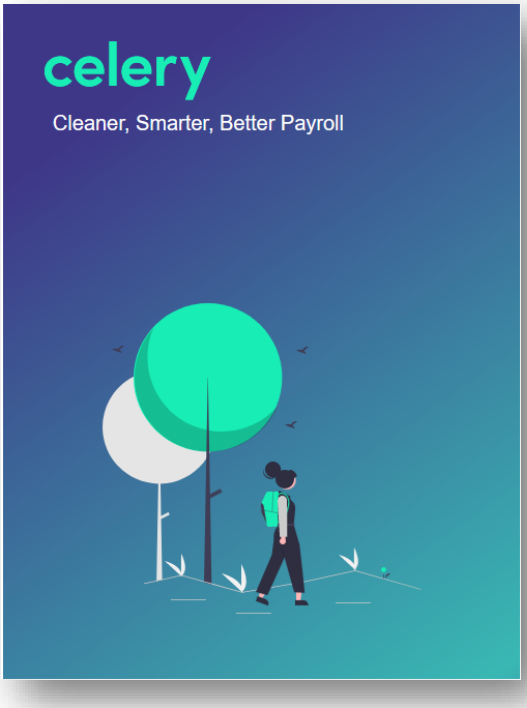
celery

https://app.celery.co.il/



Haim Mitrany

AUGUST 6, 2024

1. Introduction

The penetration testing was focused on the UI and web parts of the system.

The penetration testing was conducted in a Gray Box methodology approach that simulated attacks on the system performed by external unauthorized attackers as well as malicious internal users.

The penetration testing did not include intrusive testing or attacks that can cause actual damage to the system. Wherever feasible, the interactions with the system's external assets and providers were documented and analyzed in a passive manner only from a client-side perspective in order to identify security vulnerabilities that can affect celery users of the application.

1.1. Test details

Automatic scan:

Scan with programs like nmap, nikto

Manual scans:

Scan by using PROXY like BURP,

Threat level of the vulnerabilities

The severity of the vulnerabilities detected during the tests was determined using OWASP methodologies.

All tests were performed under celery web console follows:

url: https://app.celery.co.il/

2. Summary of findings

The following table provides a concise overview of the OWASP Top 10 security vulnerabilities, highlighting the most critical web application security risks and offering a brief explanation of each one.

Num.	Vulnerability	Description	Found issues
1	*Broken Access Control*	Access controls are improperly enforced, allowing unauthorized users to view, modify, or delete sensitive data or perform actions typically restricted to privileged accounts.	No issues found
2	*Cryptographic Failures*	Inadequate encryption or hashing methods expose sensitive data like passwords, credit card information, or personal details to attackers who can intercept or manipulate the data.	No issues found
3	*Injection*	Unsanitized user input can be used to inject malicious code into application queries, such as SQL, LDAP, or NoSQL, potentially leading to data manipulation, unauthorized access, or system compromise.	No issues found
4	*Insecure Design*	Design flaws in application logic or architecture lead to exploitable security weaknesses, such as insecure authentication flows, predictable resource locations, or unchecked access.	No issues found
5	*Security Misconfiguration*	Misconfigured security settings, like leaving default credentials or enabling unnecessary features, make applications vulnerable to exploitation by providing attackers with unintended access.	No issues found
6	*Vulnerable and Outdated Components*	Using outdated software components, plugins, or libraries with known vulnerabilities increases the risk of compromise if attackers exploit those weaknesses.	No issues found
7	*Identification and Authentication Failures*	Inadequate authentication or identity management practices, such as weak passwords, poor session handling, or insecure password recovery, allow unauthorized access.	No issues found
8	*Software and Data Integrity Failures*	The integrity of software updates, critical data, and CI/CD pipelines isn't verified, allowing attackers to tamper with or insert malicious software or data.	No issues found
9	*Security Logging and Monitoring Failures*	Inadequate logging and monitoring practices delay detection of suspicious activities, hindering the ability to respond promptly to potential breaches.	No issues found
10	*Server-Side Request Forgery (SSRF)*	By manipulating URLs in server-side requests, attackers can force a server to fetch or disclose internal resources or conduct malicious actions against other systems.	No issues found