Practical Assessment Report 2

INTE2101 Ethical Hacking

Name: Gaurav Jain

USN: s4036068

SCOPE

The scope of this penetration testing report is:
https://inte2102.semiregular.space/Practical_Assessment/2/%7B2a726c62-0783-421c-8282-4ad1401d6ba8%7D/login.php

Table of Contents

Executive Summary

Assessment Objective: The primary purpose of this assessment is to rigorously test and identify vulnerabilities present on the target website. This is a grey box penetration test.

Authentication Credentials Provided:
User ID: 20022196
Password: sixfolddiabolic

Server Information: Our review determined that the web server, inte2102.semiregular.space, runs on a Windows Server platform. Moreover, through a ping test, we ascertained that the server's target is 'hydra.semiregular.space.'

Website Navigation: When visiting the given URL, users are directly taken to the F-Co Internet Banking login screen. For this evaluation, we've supplied the necessary user credentials.

Key Findings
The target was evaluated for numerous security flaws. Upon inspection, I determined that its primary function, which involves displaying transactions across various user accounts, is filled with several vulnerabilities. Here are the main vulnerabilities that I found:
Indirect Object Referencing (IDOR), SQL Injection, Cookie manipulation, Weak Token signature and Malicious Requests.

The vulnerabilities presented pose significant risks and have the potential to result in substantial leaks of confidential data. If these vulnerabilities are not addressed, attackers could potentially obtain complete access to the web application.

Time Frame:
The various tests were conducted over 2 weeks.

Tools Used:
1. JWT Cracker – To crack the secret signature of JWT Tokens
2. JWT debugger.io – To decode the JWT tokens and make required changes
3. Burp Suite – To intercept, manipulate and decode the requests between the website and server.

Risk Matrix Used

Risk Factors
For each discovery in the assessment, two essential factors are employed to gauge its risk. These factors are assessed on a scale ranging from 1 (low) to 5 (high).

Effect

The Effect metric gauges the potential influence a finding might exert on technical and commercial processes. This covers a range of concerns, from potential implications on the data or system's confidentiality, integrity, and accessibility, to the possible financial or reputation damages.

Likelihood
The Likelihood factor evaluates the potential for a discovery to be exploited. It considers elements such as the skill level required of a potential attacker and the relative ease with which an exploitation could occur.

Rating	CVSS Score
None	0.0
Low	0.1-3.9
Medium	4.0-6.9
High	7.0-8.9
Critical	9.0-10.0

Severity Descriptions:

Results are divided into five unique tiers based on the assessed danger and potential consequences for your enterprise, inspired by the OWASP Risk Rating Approach.

None (0.0): Observations in this group represent a minimal to non-existent threat, posing no meaningful challenge to your business activities. They are of minor significance.

Low (0.1-3.9): Issues within this span denote a minor hazard, usually marked by a restricted potential effect on the business or a small chance of happening. They deserve consideration, but they're not top priorities.

Medium (4.0-6.9): Results within this tier suggest an intermediate danger level. These might result in a discernible effect on business or an increased chance of happening, indicating that measures should be taken.

High (7.0-8.9): Issues in this category are considered to be of significant concern. They carry a risk of major business consequences and have a notable probability of taking place. Immediate measures are recommended for these.

Critical (9.0-10.0): This highest category contains results with an unparalleled business effect and an extremely high probability of happening. They are of utmost importance and necessitate swift, decisive actions.

Summary of Recommendations

1. SQL Injection:
    - Mitigation: Utilize prepared statements, which are often referred to as parameterized or bound queries, to distinctly separate the SQL query's structure from its input data. By doing this, you essentially make it very difficult for attackers to maliciously manipulate the SQL query by injecting malicious code. This method ensures data is always treated as data and not executable code.
2. Insecure Direct Object Referencing:
    - Mitigation: Always perform rigorous validation of incoming requests to ensure they're genuine and not malicious attempts to access data. Additionally, switch to using non-sequential and non-predictable identifiers for your resources. Couple this with strong access control mechanisms, especially for static resources, to ensure only authorized users can access specific content.
3. Cookie Poisoning:
    - Mitigation: It's imperative not to let user input dictate cookie names or their values directly. When it's essential to store query string parameters within cookies, special attention should be given to filter out characters such as semicolons. Semicolons can act as delimiters and could be exploited to manipulate name/value pairs within the cookie.
4. File Upload Vulnerability:
    - Mitigation: Introduce strict measures that filter and check file types during the upload process, ensuring only safe and necessary file types are allowed. As a backup security layer, tweak server settings to disable the execution of scripts, particularly in uploaded directories. This ensures that even if a malicious file is uploaded, it cannot be executed on the server.

5. JWT Token Attacks:

- Mitigation: Always use a trusted and continuously updated library dedicated to handling JWTs. It's equally crucial to invest in training and resources to ensure your development team is well-versed in JWT mechanisms and the associated security concerns. All incoming JWTs should undergo thorough signature verification processes. This includes being wary of edge cases and potential misuse of unexpected signing algorithms.

6. Unauthorized Access in Follow Redirect:
    - Mitigation: One of the often-overlooked risks is the accidental exposure of authorization headers, especially during redirection from HTTPS to HTTP. It's essential to have mechanisms in place that ensure headers containing sensitive information are stripped or not forwarded during such redirects. This reduces the chance of sensitive data leakage during transitions between secure and non-secure channels.

Detailed Security Vulnerability Analysis

SQL Injection

SQL Injection (SQLi) is a security flaw in applications that enables malicious users to insert and execute unintended SQL commands within a database. This intrusion can lead to unauthorized data access, data modification, or even gaining comprehensive control over a system. Description - I successfully accessed private data from the database after identifying an SQLi vulnerability at the specified URL.

URL: https://inte2102.semiregular.space/Practical_Assessment/2/%7B2a726c62-0783-421c-8282-4ad1401d6ba8%7D/overview.php

Steps to get the flag -

1. Navigate to the affected web application's search page.
2. In the search input field, enter the following payload:
   1' UNION SELECT NULL, NULL, NULL, NULL, NULL, NULL,table_schema FROM information_Schema.tables--

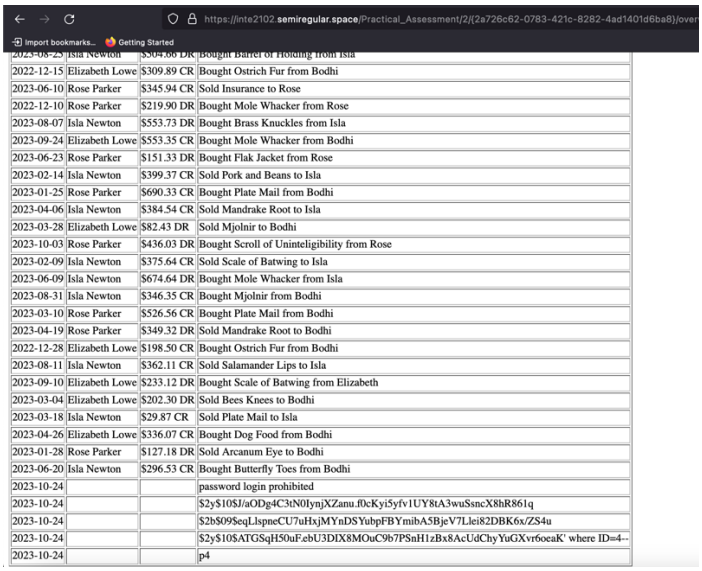
3. Upon submitting the request, the application responds with information that should normally be secure, indicating that the SQL query has been executed on the server. In our test, the injection successfully revealed all table names in the database, demonstrating the vulnerability.
4. With the knowledge of the 'it_user' table name, the following payload was crafted to enumerate columns in the table:

   'UNION SELECT NULL,NULL,NULL,NULL,NULL,NULL, COLUMN_NAME from information_schema.columns WHERE TABLE_NAME = 'it_user'--

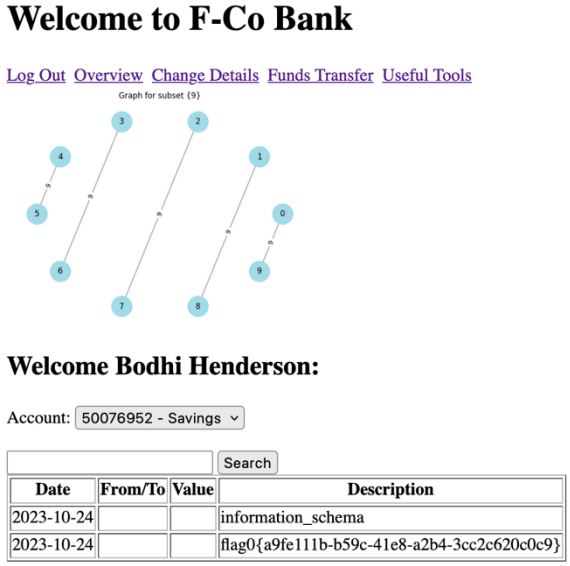
   Post the column enumeration, it was identified that the 'passwort_ik' column potentially contains sensitive information.

To confirm the data's sensitivity and validate the risk, the following payload was crafted to retrieve data from the 'passwort_ik' column:
1' UNION select NULL,NULL,NULL,NULL,NULL,NULL, passwort_ik FROM it_user--

Impact
This weakness provides malicious actors the ability to alter SQL statements within the application's database structure. Based on the permissions tied to the application's database link, someone exploiting this issue might access, modify, or even remove crucial data, undermining the data's reliability and accessibility. In some scenarios, an effective SQL injection might result in the entire system being breached.



2022-12-15	Elizabeth Lowe	$309.89 CR	Bought Ostrich Fur from Bodhi
2023-06-10	Rose Parker	$345.94 CR	Sold Insurance to Rose
2022-12-10	Rose Parker	$219.90 DR	Bought Mole Whacker from Rose
2023-08-07	Isla Newton	$553.73 DR	Bought Brass Knuckles from Isla
2023-09-24	Elizabeth Lowe	$553.35 CR	Bought Mole Whacker from Bodhi
2023-06-23	Rose Parker	$151.33 DR	Bought Flak Jacket from Rose
2023-02-14	Isla Newton	$399.37 CR	Sold Pork and Beans to Isla
2023-01-25	Rose Parker	$690.33 CR	Bought Plate Mail from Bodhi
2023-04-06	Isla Newton	$384.54 CR	Sold Mandrake Root to Isla
2023-03-28	Elizabeth Lowe	$82.43 DR	Sold Mjolnir to Bodhi
2023-10-03	Rose Parker	$436.03 DR	Bought Scroll of Unintelligibility from Rose
2023-02-09	Isla Newton	$375.64 CR	Sold Scale of Batwing to Isla
2023-06-09	Isla Newton	$674.64 DR	Bought Mole Whacker from Isla
2023-08-31	Isla Newton	$346.35 CR	Bought Mjolnir from Bodhi
2023-03-10	Rose Parker	$526.56 CR	Bought Plate Mail from Bodhi
2023-04-19	Rose Parker	$349.32 DR	Sold Mandrake Root to Bodhi
2022-12-28	Elizabeth Lowe	$198.50 CR	Bought Ostrich Fur from Bodhi
2023-08-11	Isla Newton	$362.11 CR	Sold Salamander Lips to Isla
2023-09-10	Elizabeth Lowe	$233.12 DR	Bought Scale of Batwing from Elizabeth
2023-03-04	Elizabeth Lowe	$202.30 DR	Sold Bees Knees to Bodhi
2023-03-18	Isla Newton	$29.87 CR	Sold Plate Mail to Isla
2023-04-26	Elizabeth Lowe	$336.07 CR	Bought Dog Food from Bodhi
2023-01-28	Rose Parker	$127.18 DR	Sold Arcanum Eye to Bodhi
2023-06-20	Isla Newton	$296.53 CR	Bought Butterfly Toes from Bodhi
2023-10-24			password login prohibited
2023-10-24			$2y$10$J/aODg4C3tN0lynjXZanu.f0cKyi5yfv1UY8tA3wuSsncX8hR861q
2023-10-24			$2b$09$eqLlspneCU7uHxjMYnDSYubpFBYmibA5BjeV7Llei82DBK6x/ZS4u
2023-10-24			$2y$10$ATGSqH50uF.ebU3DIX8MOuC9b7PSnH1zBx8AcUdChyYuGXvr6oeaK' where ID=4--
2023-10-24			p4



Severity – Critical
This suggests that there's a possibility for unauthorized access, alteration, or removal of data in the database, affecting its privacy, consistency, and accessibility. Recommended Actions:

1. Strengthen input verification procedures.
2. Adopt parameterized or bound queries.
3. Ensure effective error management practices.
4. Periodic scrutiny of code and security assessments.
5. Manage and review database access rights.

IDOR

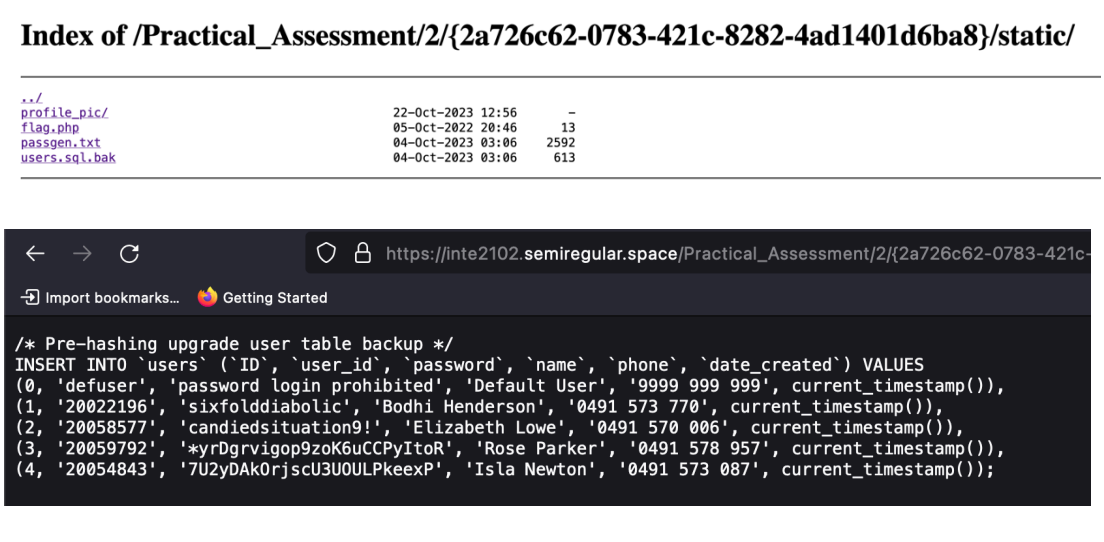
During our penetration test, we identified a security flaw referred to as Insecure Direct Object Reference (IDOR) with a particular forced browsing variation. IDOR happens when malicious actors can obtain confidential details by altering references, especially when these object references aren't secure. Unlike the standard IDOR, the forced browsing version doesn't need the initial discovery of an insecure reference; it methodically modifies parameters to achieve unauthorized entry. We noticed this flaw when reviewing the procedure for user data modification, especially during the profile picture uploads. Our

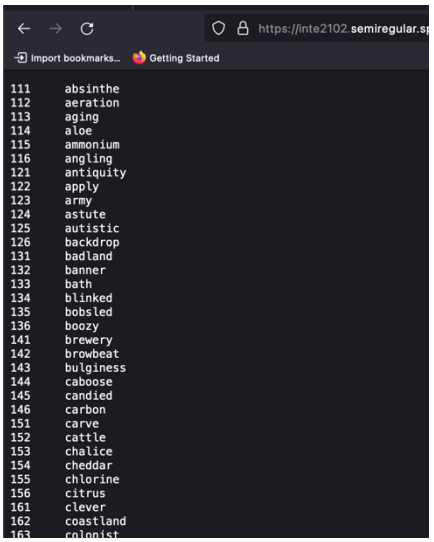
analysis showed that this feature might allow for directory navigation within the website's framework.

URL: https://inte2102.semiregular.space/Practical_Assessment/2/%7B2a726c62-0783-421c-8282-4ad1401d6ba8%7D/static

Steps to Reproduce:

1. Log in to the application and navigate to the profile details section.
2. Update the profile picture and save changes. The updated profile picture appears on the overview.php page.
3. Right-click the profile picture, select "Copy image address" (or the browser's equivalent option), and paste it into the browser's address bar.
4. In the image's URL, traverse two directories up by replacing the last two path segments with ../...
5. After performing the directory traversal, the following files were found to be accessible:
6. flag.php (access was forbidden)
7. passgen.txt
   users.sql.bak





Files that can be reached due to this flaw (namely users.sql.bak and passgen.txt) seem to house crucial confidential information. The SQL backup might hold comprehensive user data, which could encompass login details, individual specifics, or other vital data. The passgen.txt could have insights concerning password creation, possibly revealing security guidelines or direct access codes. Unpermitted entry to such information might open doors for various harmful intents, from identity breaches, monetary deception, to different cyber offenses.

Safety Measures:

1. Enhance File Protection
2. Set Up Adequate Access Barriers
3. Periodic Security Checks

4. Crisis Management

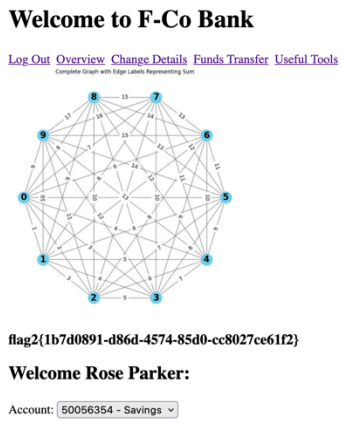
With the above information we got from users.sql.bak, we were able to see that 2 users, namely Rose Parker and Isla Newton.

Using the information provided, we can log in as Isla Newton.
User ID : 20054843
Password: 7U2yDAkOrjscU3UOULPkeexP

Looking further into the users.sql.bak page, we can understand that the user Rose Parker's password is locked. We can reset that password and change it by using the SQL vulnerability. Injecting the payload below into the new password and repeat password fields of the change details page, from any of the other users that we have access to.

$2y$10$ATGSqH50uF.ebU3DlX8MOuC9b7PSnH1zBx8AcUdChyYuGXvr6oeaK' where ID=3--

We choose this particular hashed payload since we know this is the hashed form of the password of the given user (Bodhi Henderson). Once this payload is sent, we can logout and login as Rose Parker using the following credentials:
User: 20054843
Password: sixfolddiabolic



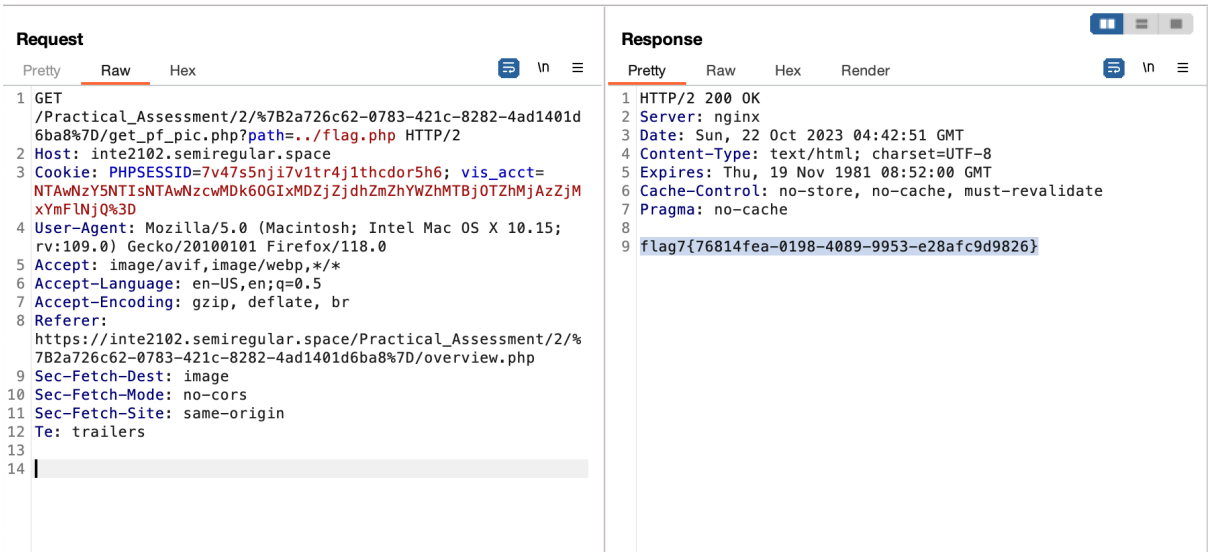
This compound exploit demonstrates how multiple lower-severity vulnerabilities can be chained together to produce a critical security issue. The ability to change a user's password through SQL injection directly undermines the confidentiality, integrity, and availability principles of security for the entire application.

Malicious requests

A vulnerability tied to malicious requests can be described as a web application's inability to ascertain if an incoming request has been tampered with. This vulnerability might pave the way for access to certain elements within the web application that should remain restricted.

I then shifted focus to the 'flag.php' which was unreachable through IDOR alone. Accessing it necessitated some form of authorization. Using Burp, I began analyzing the network requests. Interestingly, the overview page seemed to employ 'get_pf_pic.php' to fetch a user's profile picture. I postulated that a manipulated request might provide an entry point to 'flag.php'.

Through Burp, I intercepted the network traffic and noticed that on a subsequent request, the platform was attempting to obtain a profile picture via the PHP file, which had a path variable to dictate the image's location on the server directory. I then routed the request to the repeater and altered the path, awaiting the server's response. The modified request was as follows:



Access Granted! I could look at the contents of the flag.php which was not accessible using IDOR.

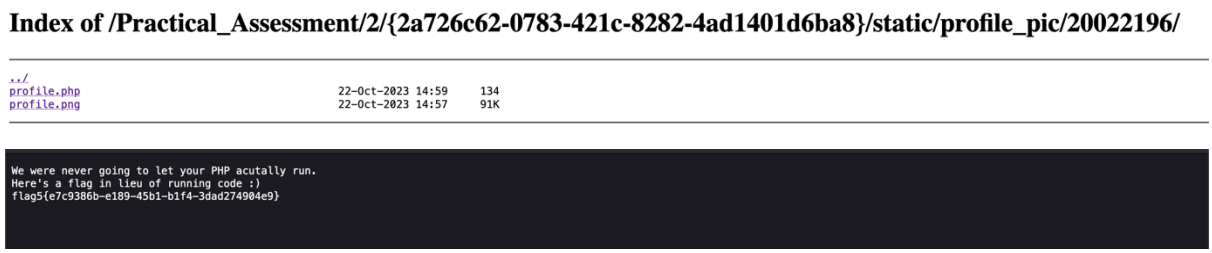
Insecure File Upload Leading to Privilege Escalation

The software displays a severe security oversight concerning the file upload process within its "change details" feature. Even though the platform tries to block unsafe file categories, it doesn't adequately handle file suffixes, allowing a sidestep of its upload safeguards. This oversight allows harmful scripts to be uploaded, camouflaged as standard file types like JPEG.
This susceptibility arises from the server's flawed assessment of file details and its dependency on file suffixes for type identification, neglecting to thoroughly inspect the file's genuine content or design. Consequently, malicious individuals could manipulate this gap to upload actionable scripts (like PHP reverse shells) that appear harmless. When the server operates these files, it could potentially offer unauthorized access or even total command, contingent on the server's setup and the script's operations.

Target URL: https://inte2102.semiregular.space/Practical_Assessment/2/%7B2a726c62-0783-421c-8282-4ad1401d6ba8%7D/change_details.php

Steps

1. Initiate a profile picture update in the "change details" section, substituting a reverse shell script (which can be found on Github) as the image file, after you upload a normal .jpeg image.
2. Confirm the script's presence on "overview.php" post-upload.
3. Employ Insecure Direct Object Reference (IDOR) to navigate up one directory level.
4. Locate and identify the malicious ".php" file, now resident on the server.



Impact:
This flaw is highly alarming because of the possibility for full system breach. Taking advantage of weak file upload mechanisms, a malicious user might elevate privileges and potentially access restricted data, carry out harmful actions, alter system setups, and instigate additional malicious activities within the infrastructure. This compromises the application's privacy, consistency, and accessibility, and presents a substantial threat to the foundational server and associated network framework.

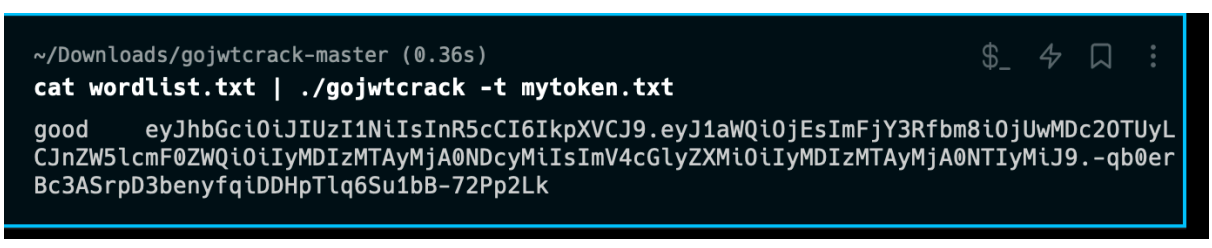
Unauthorized Access via Compromised JWT Token Attacks

Assaults focused on JWTs usually involve tweaking the token's design, leveraging gaps in the token's verification system. The main goal of such attacks is to bypass authentication processes, possibly letting an intruder pose as a verified user. This is commonly done by modifying the token's content or its validation signature, deceiving the server into providing permissions and resources meant for different users. In this scenario, the attack required the counterfeiting of the token's validation to achieve unauthorized entry, appearing as a standard user account.

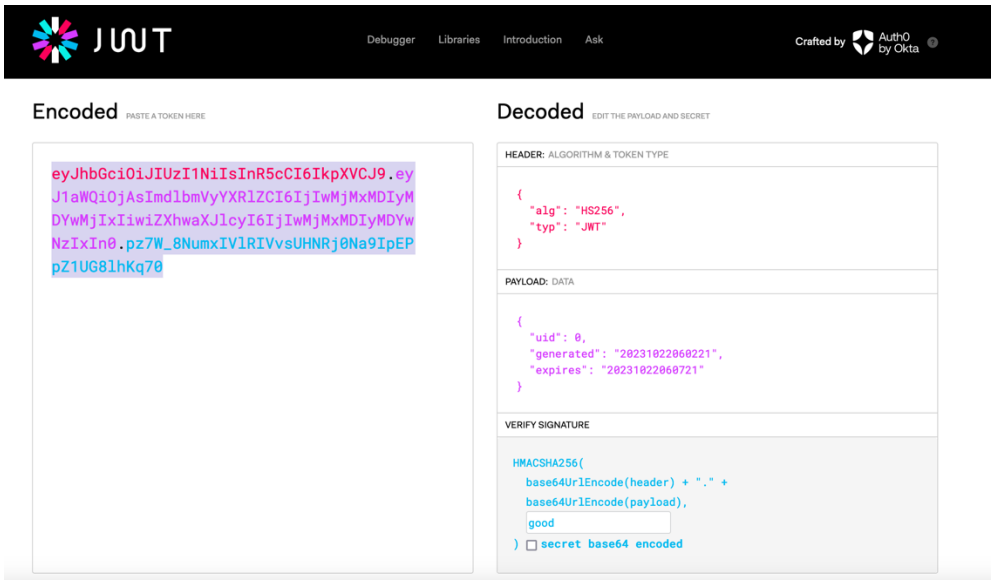
Target URL: https://inte2102.semiregular.space/Practical_Assessment/2/%7B2a726c62-0783-421c-8282-4ad1401d6ba8%7D/tools.php

Steps-

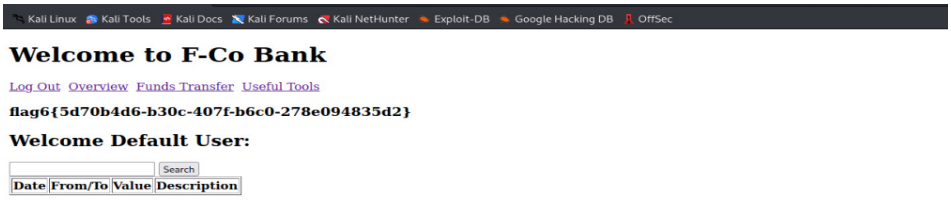
1. Utilize the link generator on tools.php to create a JWT.
2. Retrieve the secret key essential for signature verification using the 'gojwtcrack' tool, employing the 'passgen.txt' wordlist (previously obtained via IDOR) with the command:



3. Intercept the request that contains the JWT token used to login to the website.
4. Modify the JWT's user ID to '0' on ‘JWT.io’ to impersonate the default user, remove the account number details, and resign the token with the secret key we obtained.



5. Armed with this new token, we gain unauthorized access to the user with ID 0.



Impact:
The successful manipulation of a JWT can lead to several high-risk security threats, including:

1. Identity Impersonation: Intruders might adopt the persona of any user, leading to unsanctioned entry to confidential data, user profiles, individual details, or exclusive business records.
2. Elevated Access Rights: Should the infiltrated account possess admin rights, the malicious user could secure total dominance over the platform, allowing modifications, data deletions, access right shifts, or the establishment of additional account permissions for ongoing entry.
3. Information Exposure: This flaw can trigger major data exposures, translating into the disclosure of private data, potentially incurring legal and financial repercussions, tarnishing the brand's image, and violating data security norms.
4. Total System Breach: In severe scenarios, comprehensive system access might be realized, posing a threat to the broader network or host system.

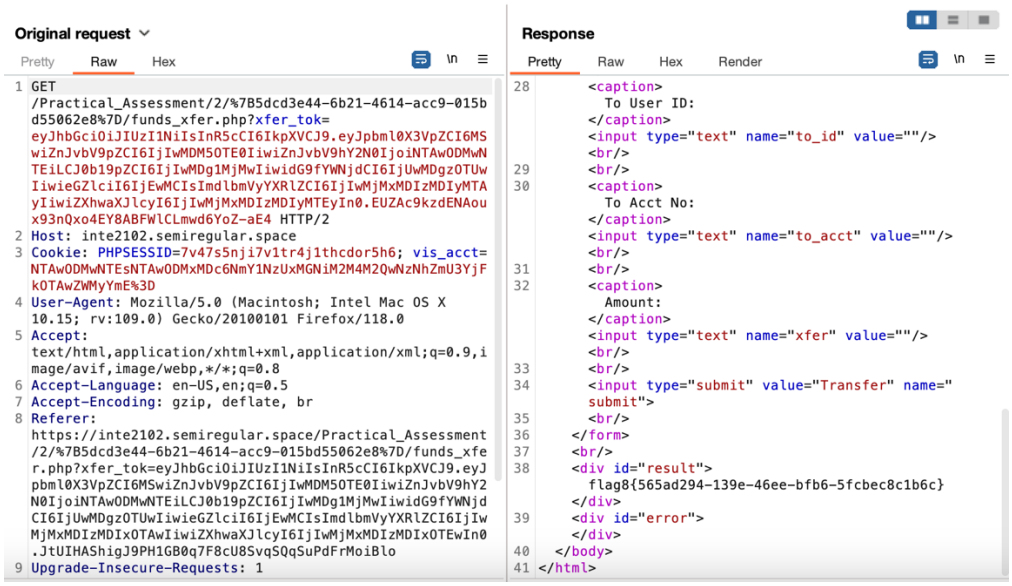
Exploiting JWT on Fund Transfer Functionality

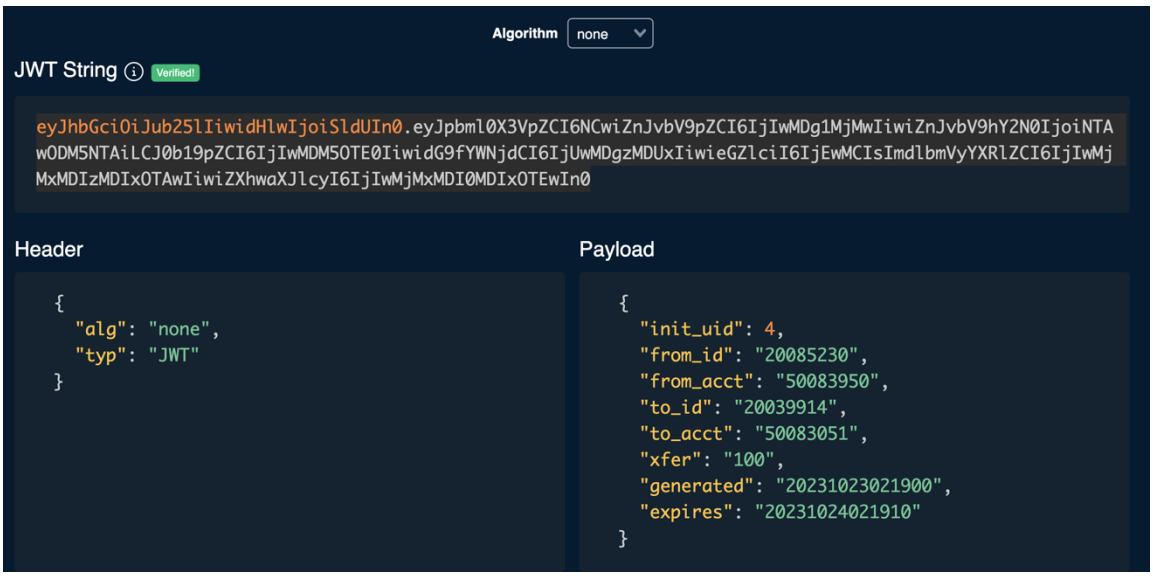
In a critical manipulation of the software's fund transfer functionality, I tweaked the JWT tokens content during the transfer phase. This involved changing the 'sender' account detail in the token content to align with the 'recipient' account, effectively setting up a movement where the money moves from the receivers account to the senders account.

Moreover, to sidestep the token's essential signature validation, I leveraged a recognized weakness in the JWT's structure: adjusting the algorithm ('alg') header to 'none.' This modification in the algorithm sector caused the system to mistakenly overlook the signature validation, presuming the token was already verified and authenticated.

This intervention enabled a transaction to go through without the necessary oversight, underlining a profound deficiency in the financial tool's protective framework. The misuse of this flaw showcased not only the transaction's facilitation on misleading grounds but also led to the successful retrieval of Flag 8, underscoring the gravity of this security gap.

Target URL: https://inte2102.semiregular.space/Practical_Assessment/2/%7B2a726c62-0783-421c-8282-4ad1401d6ba8%7D/funds_xfer.php





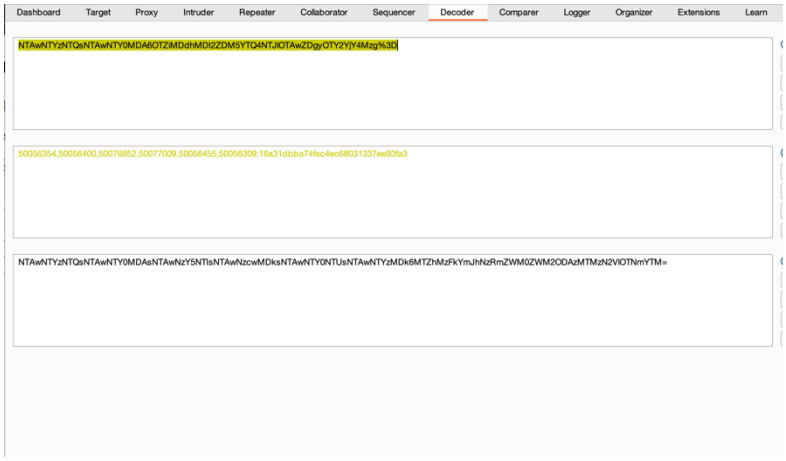
Cookie Tampering and Poisoning

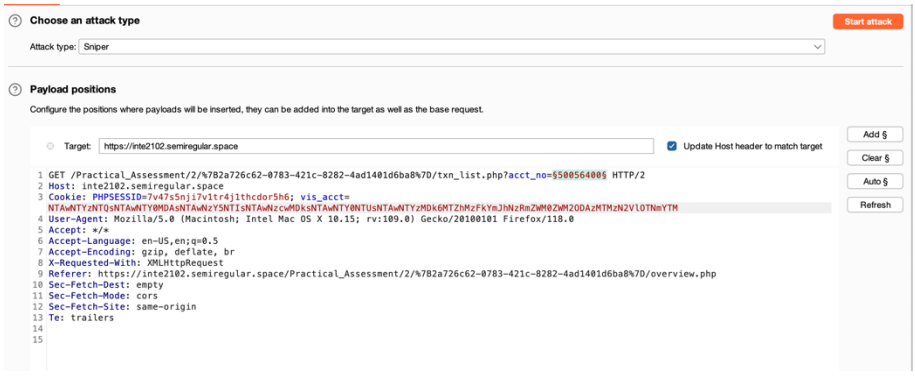
Cookies, which are compact data packets specific to a user and website, enable customized user interactions by retaining session-related information directly in a user's browser. While they offer benefits like recalling user preferences, monitoring user engagement, and supervising e-commerce actions, they also come with security concerns. Malicious actors can tamper with or forge these cookies to gain access to private data or pose as genuine users. In our evaluation, we pinpointed a vulnerability that made user account details available and editable via the cookie headers. Notably, these cookies saved and conveyed account IDs in an unprotected manner, leaving them vulnerable to unauthorized access or modification.

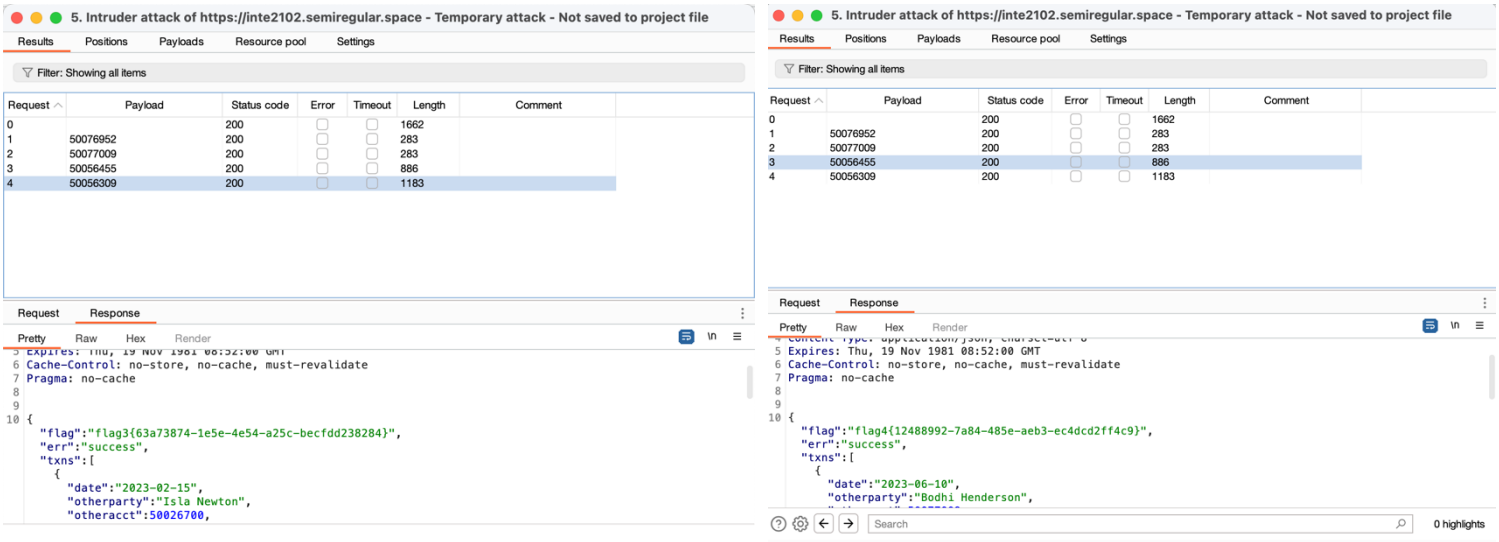
During our scrutiny of "inte2102.semiregular.space," we identified a pivotal flaw tied to mishandled session cookies, particularly the "vis_acct" cookie. This cookie, designed to oversee user session particulars related to banking operations, was observed to carry delicate details insecurely.

An irregularity came to light while examining the platform's financial transaction updates. Even though the user interface showcased only one account for each user on the main dashboard, our study unearthed the presence of multiple accounts assigned to each user. The structure of the "vis_acct" cookie was inherently insecure, with account IDs succeeded by an MD5 hash of those IDs, all shared in unencrypted form. We initiated a precision-focused tampering assault by adjusting the "vis_acct" cookie contents.

Method involved: Tweaking the cookie to incorporate extra account IDs found during our exploration. Leveraging Burp Suite's Intruder utility to systematize and intensify our approach, we employed the 'Sniper' configuration, designating the account IDs as the focal point and the freshly identified account IDs as payloads. The adjusted requests unveiled operations linked to the concealed accounts, revealing sensitive data and bringing Flags 3 and 4 to the forefront, signifying a security lapse.







Impact: This flaw potentially unveils confidential user details and allows unwarranted entry into hidden accounts, marking a critical security threat. Exploiting this vulnerability could result in monetary losses, unauthorized information access, or harmful account operations.

Conclusion: Risk Analysis

In this section we will analyze the risk put up by the vulnerabilities. We will use the same definitions and matrix mentioned in the beginning of the document (which are from AS/NZS ISO/IEC 27005:2012.)

The analysis for every vulnerability is as follows:

Insecure Direct Object Referencing

Impact – Very High, Likelihood rating – Certain
Risk Rating – Very High

IDOR is a relatively easy-to-exploit vulnerability and does not require a very deep knowledge for exploiting the vulnerability hence the likelihood of IDOR happening is very high.

SQL Injection

Impact Rating – Extreme, Likelihood Rating – High
Risk Rating – Very High
SQL injection had a very high impact during our testing therefore we gave it an extreme impact rating but it requires a good level of knowledge of database management systems hence the likelihood of that happening is not very high.

Cookie Manipulation

Impact Rating – High, Likelihood Rating – Medium
Risk Rating - Medium
Cookie Manipulation although a very trivial exploit that does not require much understanding, in our case, the cookies were double encoded, and hence we gave it a likelihood rating of Medium. But once the encodings were cracked, the impact of the vulnerability was quite high thus we gave it an impact rating of high.

File Upload vulnerability

Impact Rating – Extreme, Likelihood Rating – High
Risk Rating – Very High
File upload vulnerability is a vulnerability that allows the attacker to upload malicious files that can create a backdoor to the server. While the impact rating of it is extreme, the likelihood of that happening is not that high because of the knowledge needed to exploit that vulnerability to its full potential is not that common. Therefore, we gave it a likelihood rating of high.

Malicious Requests

Impact rating – High, Likelihood Rating – Medium
Risk Rating – Medium
Malicious Requests is a vulnerability that requires some extent of knowledge of how networking requests are made, which is not very common therefore while we gave it an impact rating of high, the likelihood of it happening is still medium.

Weak JWT Token Signature

Impact rating – Very High, Likelihood Rating – Low
Risk Rating – Medium
Weak token signatures are a very serious vulnerability, especially when the tokens are used for authentication. While the Impact of this is very high, the likelihood is still low as the attacker needs to find the secret key used to sign the token.