[Car Rental]

Penetration Testing Report

Ori Hatuka
Grey Box PT

TABLE OF CONTENT

EEXECUTIVE SUMMARY

INTRODUCTION

Penetration testing of Car Rental company, which is the second test performed for the Car Rental web site; was performed to check the rectifications applied after the conclusions of the previous findings.

A grey box security audit was performed against the Car Rental subdomain of the Car Rental web site. Ori Adivi reviewed the system's ability to withstand attacks and the potential to increase the protection of the data they contain

This Penetration test was conducted during August 2022 and includes the preliminary results of the audit.

SCOPE

WEB APPLICATION

The penetration testing was limited to the Car Rental sub domain with no prior knowledge of the environment or the technologies used.

- General Injection attacks and code execution attacks on both client and server sides.
- OWASP Top 10 possible vulnerabilities including CSRF tests.
- Inspection of sensitive data handling and risk of information disclosure.
- Tests against Advance Web Application Attacks.

CONCLUSIONS

From our professional perspective, the overall security level of the system is **Medium - Critical**.

The application is vulnerable to several Brute-force attacks.

Exploiting most of these vulnerabilities requires a **Medium - Critical** technical knowledge.

IDENTIFIED VULNERABILITIES

Item	*Test Type*	*Risk Level*	*Topic*	*General Explanation*	*Status*
4.1	**Applicative**	**Critical**	**Parameter Tampering**	The Web Parameter Tampering attack is based on the manipulation of parameters exchanged between client and server in order to modify application data, such as user credentials and permissions, price and quantity of products, etc.	**Vulnerable**
4.2	**Applicative**	**Critical**	**JWT Authentication Bypass**	JSON Web Token is a proposed Internet standard for creating data with optional signature and/or optional encryption whose payload holds JSON that asserts some number of claims. The tokens are signed either using a private secret or a public/private key.	**Vulnerable**
4.3	**Applicative**	**High**	**Cross Site Request Forgery (CSRF)**	Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated.	**Vulnerable**
4.4	**Applicative**	Medium	**Cross Site Scripting (XSS)**	Cross-Site Scripting (XSS) is a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites.	**Vulnerable**

FINDING DETAILS

4.1 Parameter Tampering

Severity | **Critical** Probability | **Critical**

VULNERABILITY DESCRIPTION

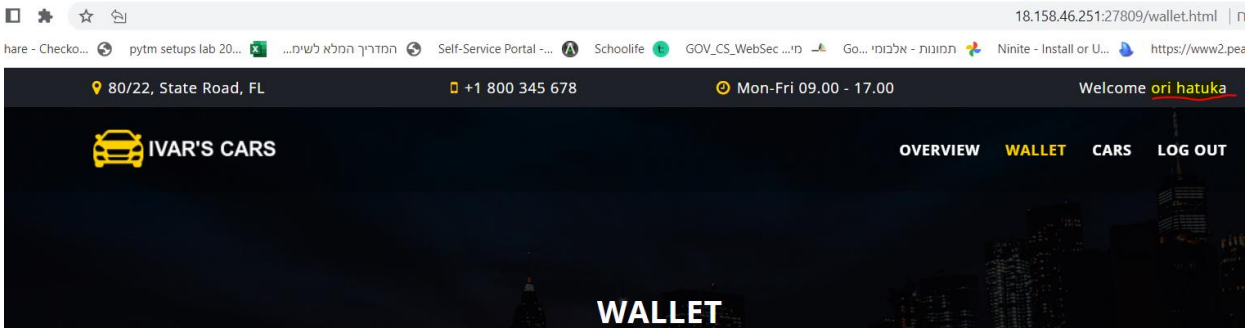
The Web Parameter Tampering attack is based on the manipulation of parameters exchanged between client and server in order to modify application data, such as user credentials and permissions, price and quantity of products, etc. Usually, this information is stored in cookies, hidden form fields, or URL Query Strings, and is used to increase application functionality and control.

VULNERABILITY DETAILS

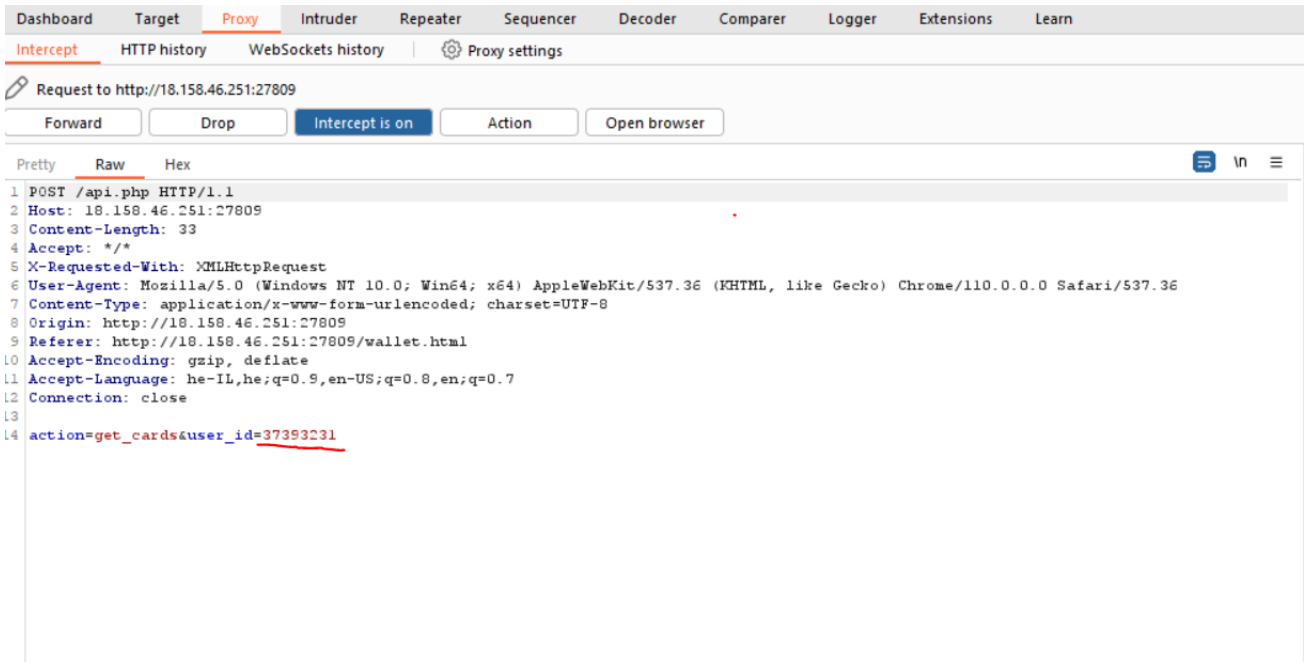
Using brute force to get the all the user id, order a car from other account and change the parameters that other user order.

EXECUTION DEMONSTRATION

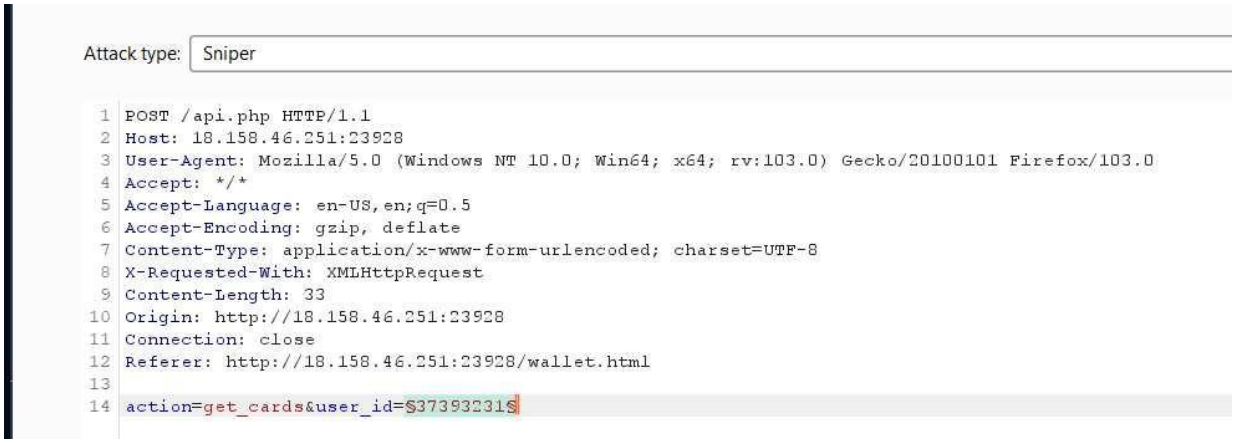
Going to wallet:



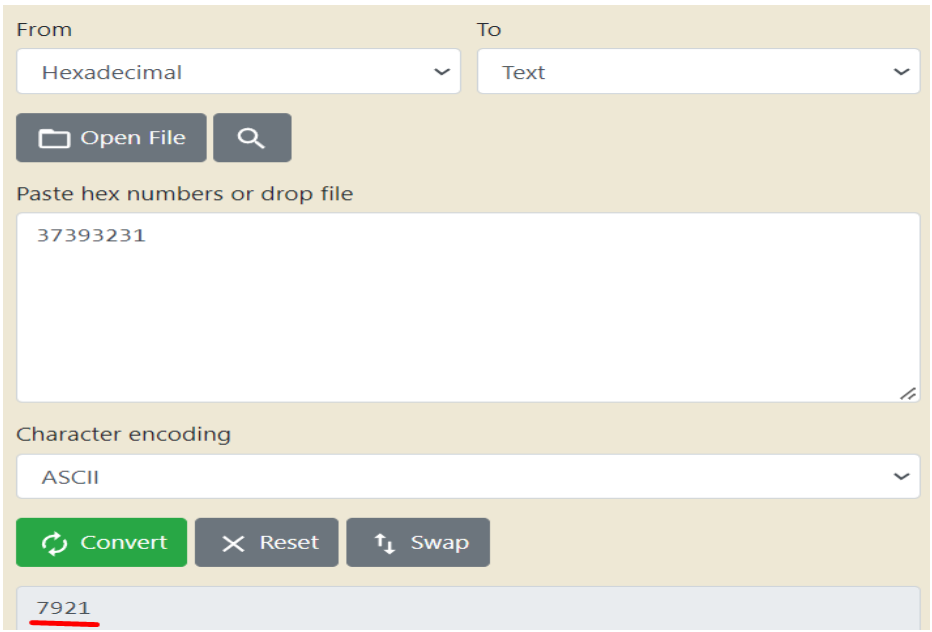
Catch the request in the burp suite:



Doing brute force to the user id

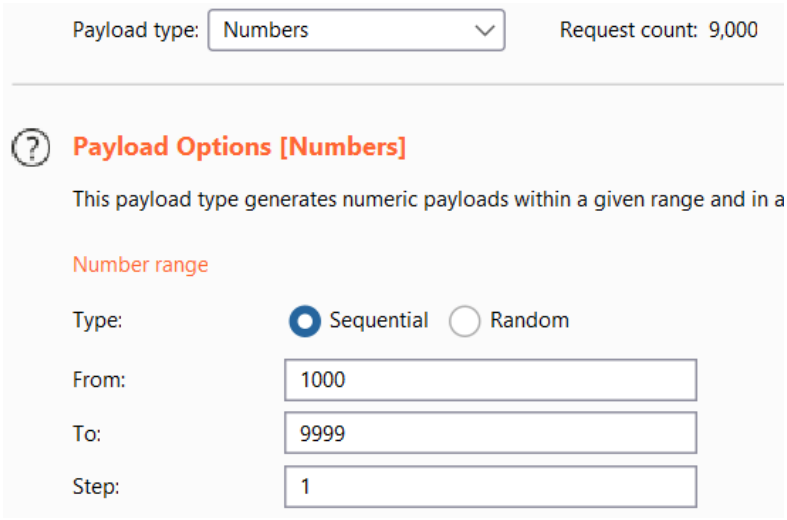


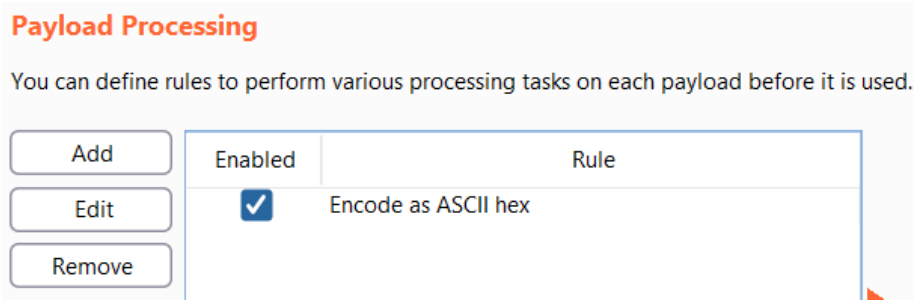
The user_id number is written in ascii, need to converter to a hexadecimal:



Set in the payload numbers 1000-9999 by step 1:

Add the payload encode as ascii hex and run:



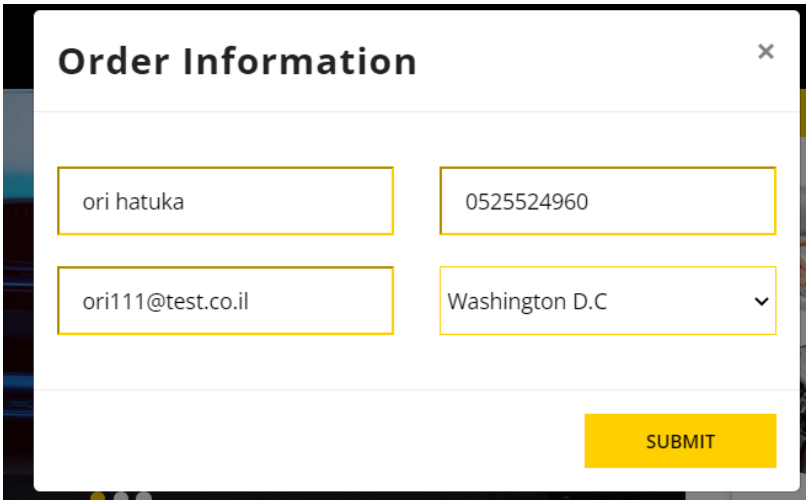


Request	Payload	Status	Error	Timeout	Length
1602	32363031	200			1908
2026	33303235	200			1901
1810	32383039	200			1879
6570	37353639	200			1055
7823	38383232	200			321
7822	38383231	200			321

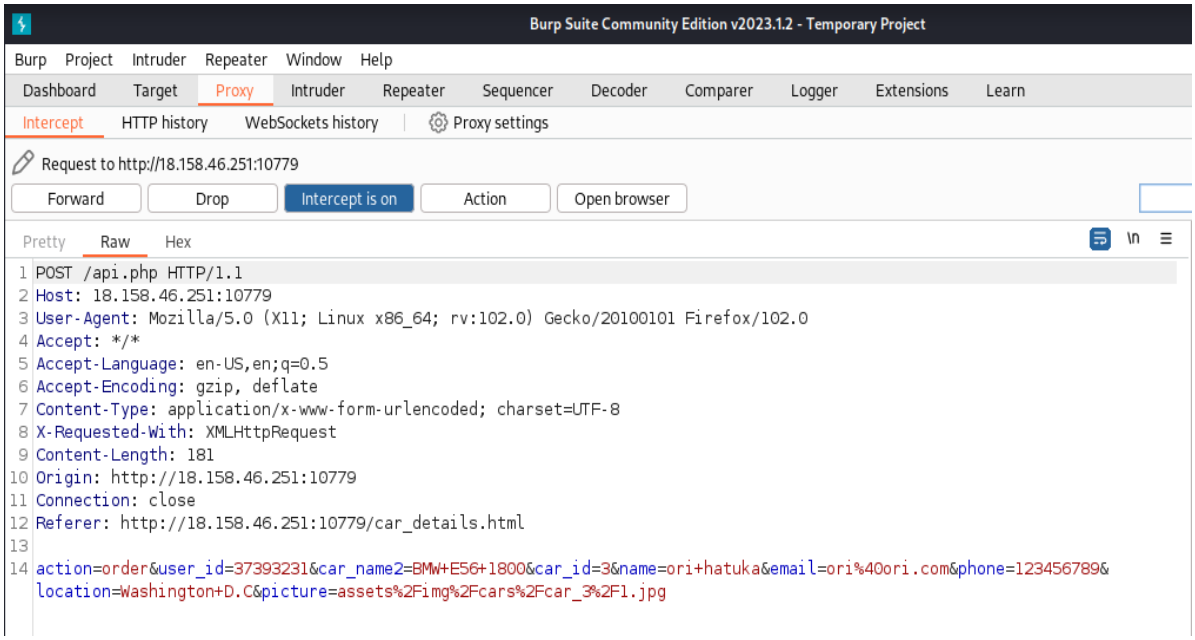
Found 4 users include us.

Order car from "ori hatuka"

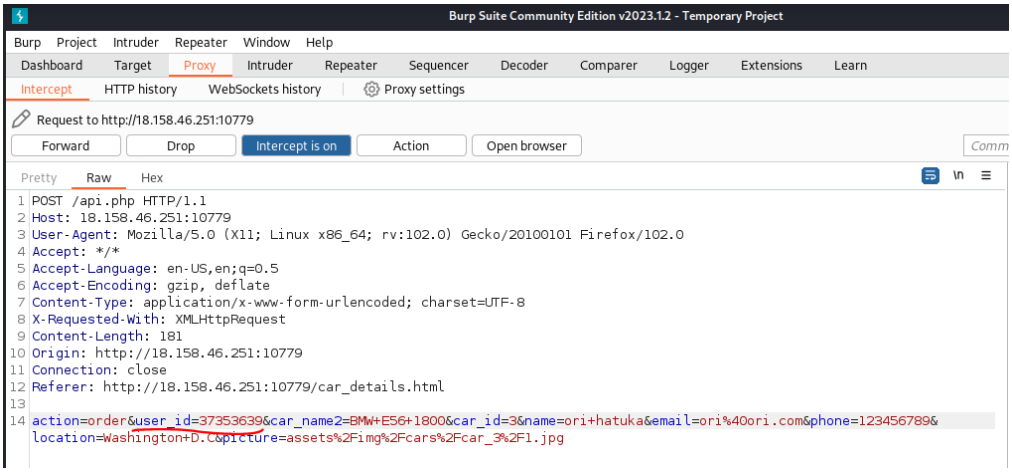
user:



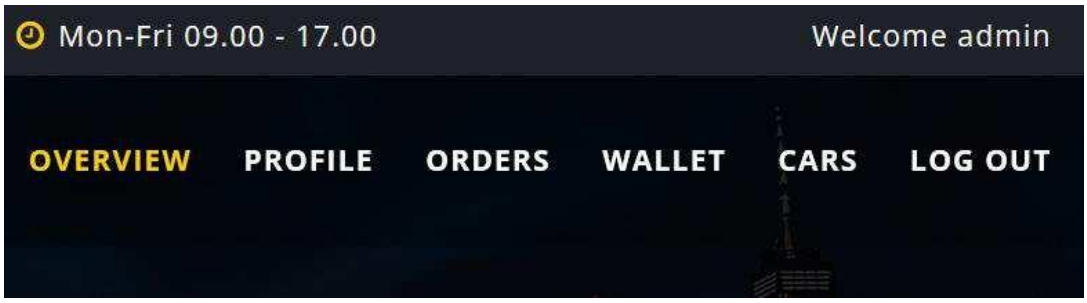
Catch packet in the burp suite:

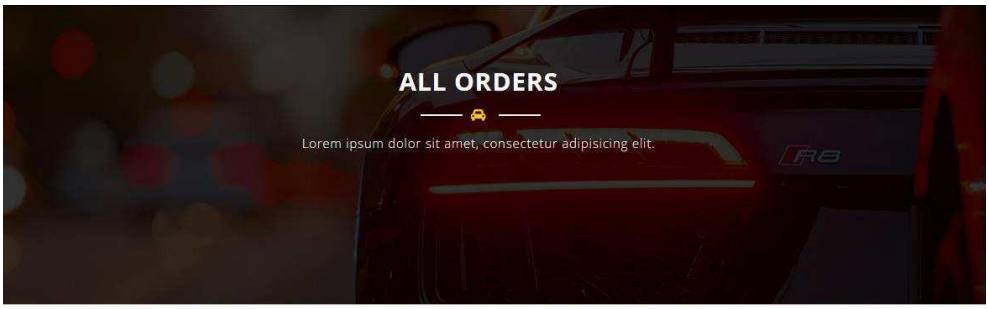


Change the parameter user id to the admin user id:



Send it and going to the user admin:





RECOMMENDED RECTIFICATION

- Sensitive parameters that should not be sent by a client -

  There are parameters whose values should not be sent from the client-side user id is similar to such a parameter and in addition it should not be stored on the client side and it is recommended to store it on the server side so that it will be protected from manipulation by the attacker, so the solution is to use session and send only the session-Id
- Control parameters with incorrect format. Assuming that a parameter is in a valid format without verifying can create serious security gaps, especially if the parameter is passed to a Structured Query Language Also, the parameter's format may be incorrect even if the parameter is normally provided by a hidden field or combo box, enabling a hacker to alter the parameter and hack into the site. For these reasons, developers should always control parameters with incorrect formats.
- Use Server-side validation compared with all inputs.
- Utilizing regex to validate or limit the data in the server side- regex is a string of text that lets you create patterns that help match, locate, and manage text.

4.2 JWT AUTHENTICATION BYPASS

Severity | **Critical** Probability | **Critical**

VULNERABILITY DESCRIPTION

JSON Web Token is a proposed Internet standard for creating data with optional signature and/or optional encryption whose payload holds JSON that asserts some number of claims. The tokens are signed either using a private secret or a public/private key.

Although JWTs can be encrypted to also provide secrecy between parties, we will focus on signed tokens. Signed tokens can verify the integrity of the claims contained within it, while encrypted tokens hide those claims from other parties

VULNERABILITY DETAILS

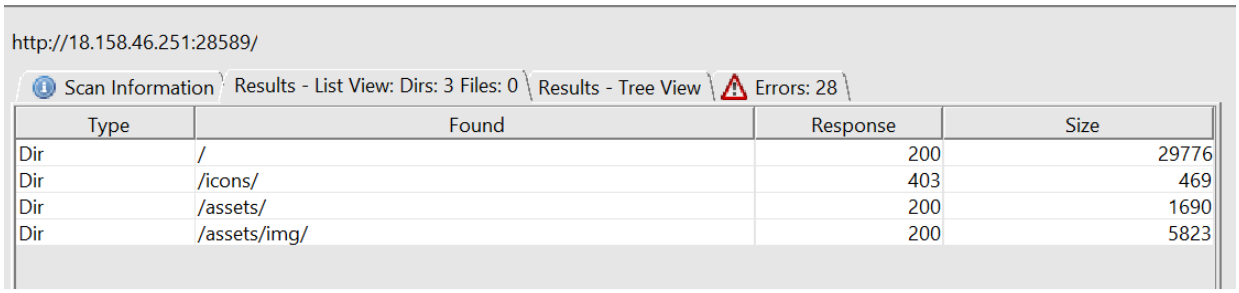
Using brute force to get the all the user id, and create admin JWT.

EXECUTION DEMONSTRATION

Found in the parameter tampering that we have 4 accounts include us and after a few times we found the user id of admin:

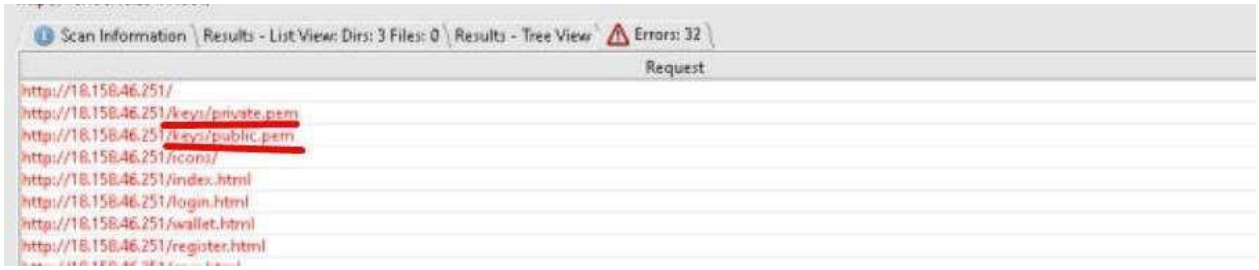
Request	Payload	Status	Error	Timeout	Length
1602	32363031	200			1908
2026	33303235	200			1901
1810	32383039	200			1879
6570	37353639	200			1055

Check with dirbuster if we have a files/dirs:



Type	Found	Response	Size
Dir	/	200	29776
Dir	/icons/	403	469
Dir	/assets/	200	1690
Dir	/assets/img/	200	5823

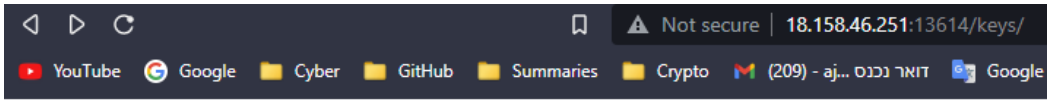
Nothing important was found but in the erros:



Also, we can use in dirb in linux and find the keys:

```
┌──(root💀kali)-[~]
└─# dirb http://18.158.46.251:28589

-----------------
DIRB v2.22
By The Dark Raver
-----------------

START_TIME: Fri Sep  9 08:39:38 2022
URL_BASE: http://18.158.46.251:28589/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

-----------------

GENERATED WORDS: 4612

---- Scanning URL: http://18.158.46.251:28589/ ----
==> DIRECTORY: http://18.158.46.251:28589/assets/
+ http://18.158.46.251:28589/favicon.ico (CODE:200|SIZE:1406)
+ http://18.158.46.251:28589/index.html (CODE:200|SIZE:29521)
==> DIRECTORY: http://18.158.46.251:28589/keys/
+ http://18.158.46.251:28589/server-status (CODE:403|SIZE:304)
==> DIRECTORY: http://18.158.46.251:28589/vendor/

---- Entering directory: http://18.158.46.251:28589/assets/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://18.158.46.251:28589/keys/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://18.158.46.251:28589/vendor/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)

-----------------
END_TIME: Fri Sep  9 08:44:49 2022
DOWNLOADED: 4612 - FOUND: 3
```

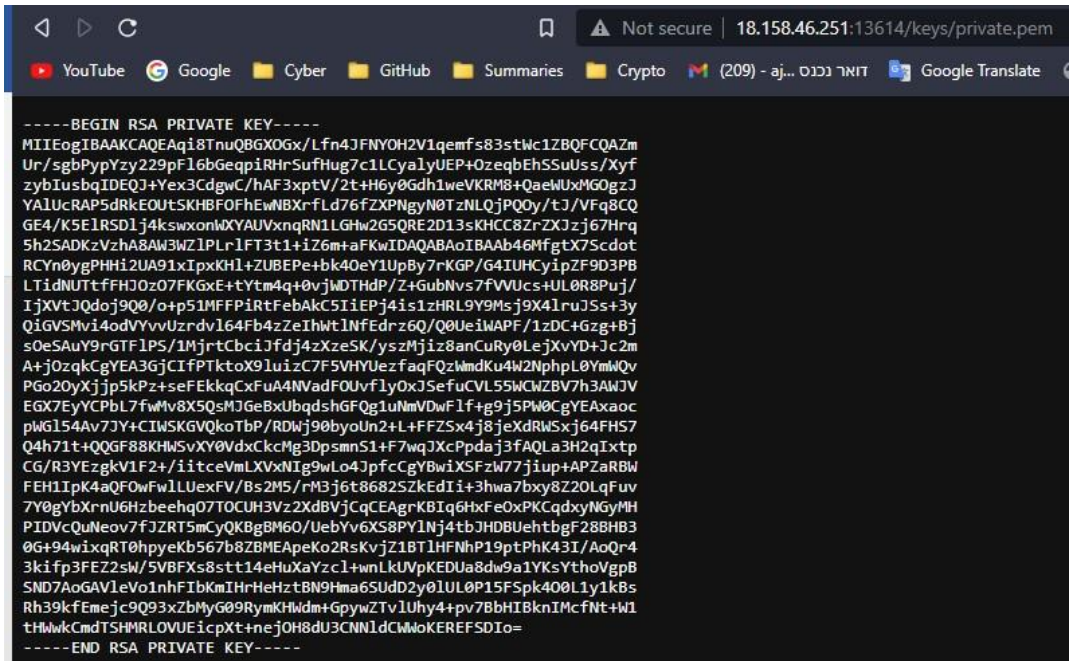
It was found that there are two keys private and public,going to the url:

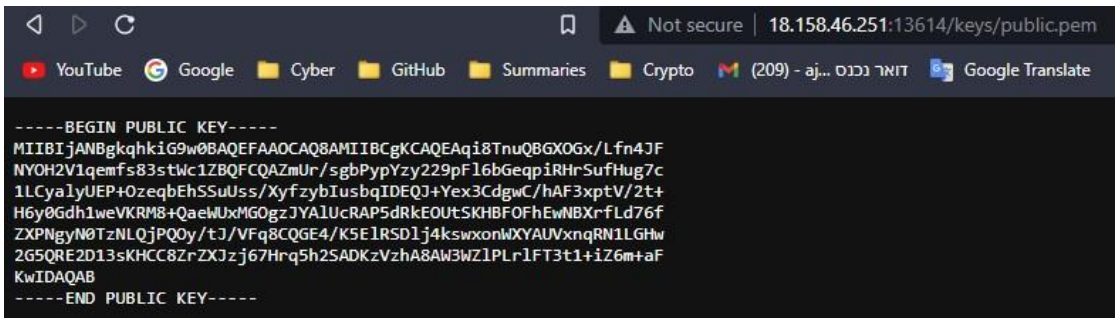


Index of /keys

Name	Last modified	Size	Description
Parent Directory		-	
private.pem	2018-07-28 13:37	1.6K	
public.pem	2018-07-28 13:37	451	

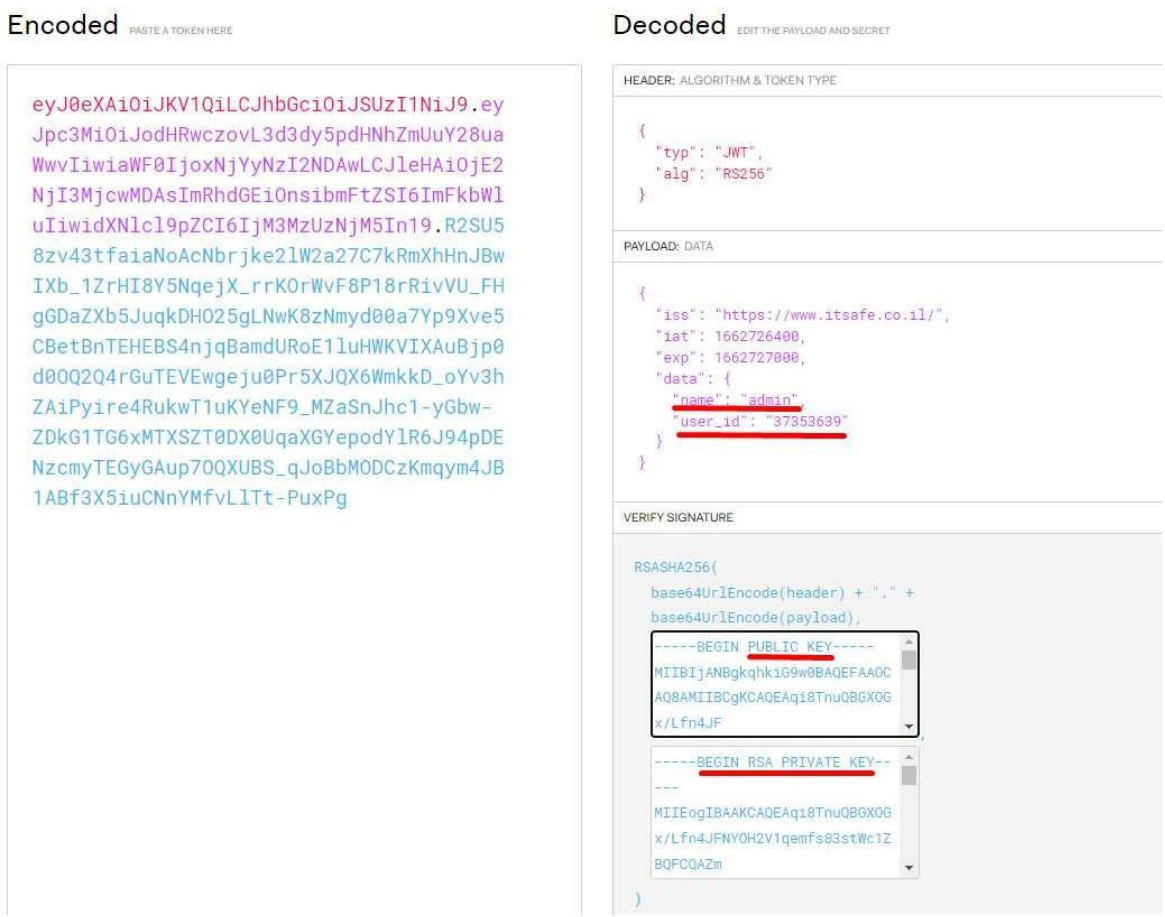
Apache/2.4.25 (Debian) Server at 18.158.46.251 Port 13614



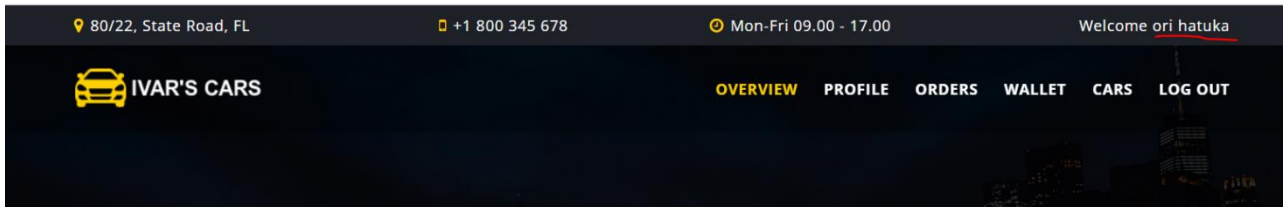


Use with jwt.io website to create a new jwt.

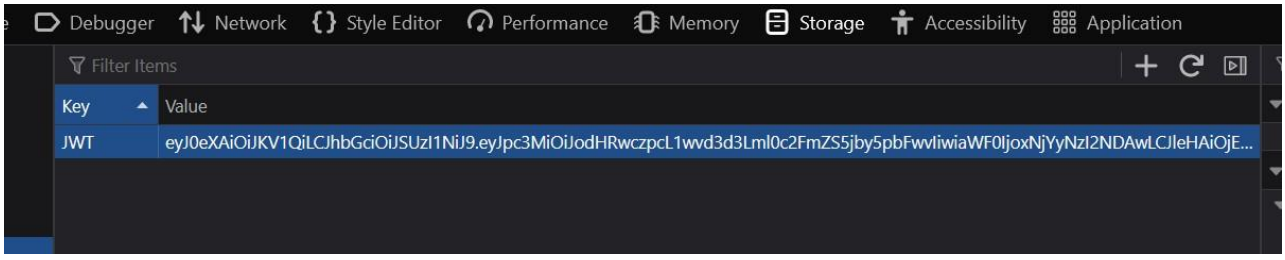
Insert the the private and public key and insert a name "admin", insert user id:



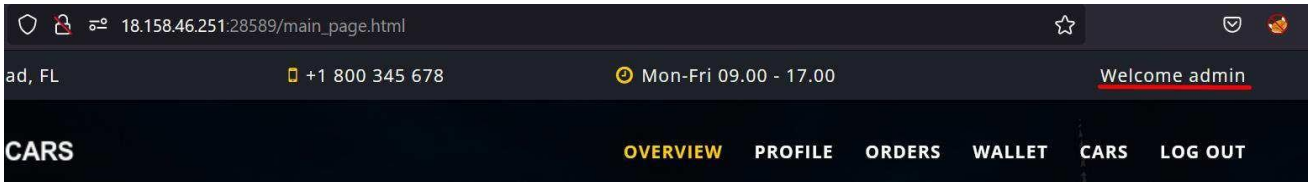
Login to the website with a regular account:



Going to inspect->Storage->Local Storage:

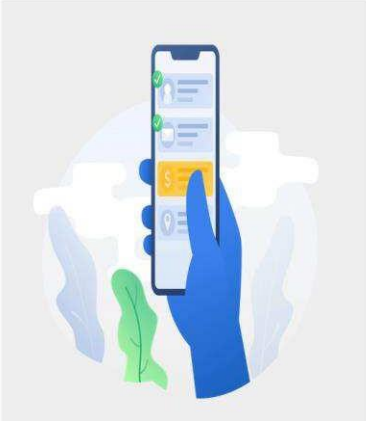


Change the JWT token and refresh the website:



We are admin!


Mastercard




FREDDIE MERCURY
Card Number
5326110330409080
Expiry Date
3/23
CVV
352
Remove Card

RECOMMENDED RECTIFICATION

- Don't expose public/private keys in the system.
- Whether the token is signed (a JWS), or encrypted (a JWE) it will contain an 'alg' claim in the header, that indicates which algorithm has been used for signing or encryption. When verifying / decrypting the token you should always check the value of this claim with a whitelist of algorithms that your system accepts. This mitigates an attack vector where someone would tamper with the token and make you use a different, probably less secure algorithm to verify the signature or decrypt the token.
- JWTs can be used as Access Tokens or ID Tokens or sometimes for other purposes it's thus important to differentiate the different types of the tokens. When validating JWTs always make sure that they are used as intended. E.g. that you won't accept an ID Token JWT as an Access Token. This can be achieved in different ways and will depend on the implementations you use

4.3 CROSS SITE REQUEST FORGERY (CSRF)

Severity | **High** Probability | **High**

VULNERABILITY DESCRIPTION

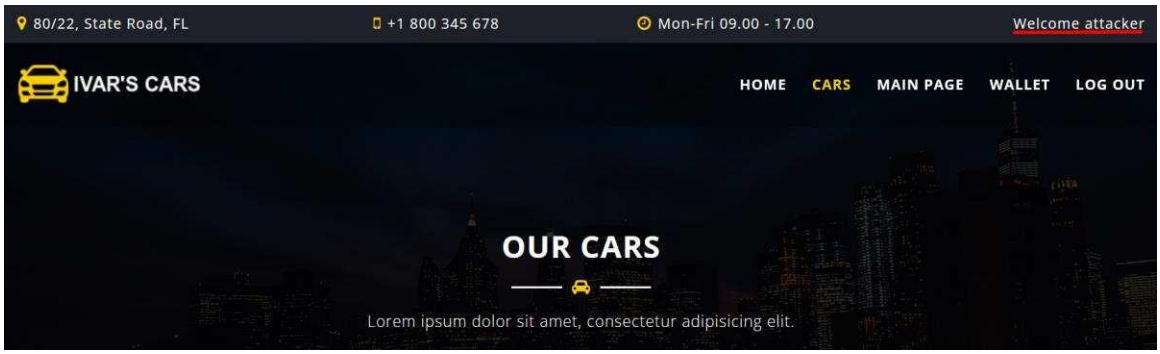
Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. with a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing. if the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. if the victim is an administrative account, CSRF can compromise the entire web application.

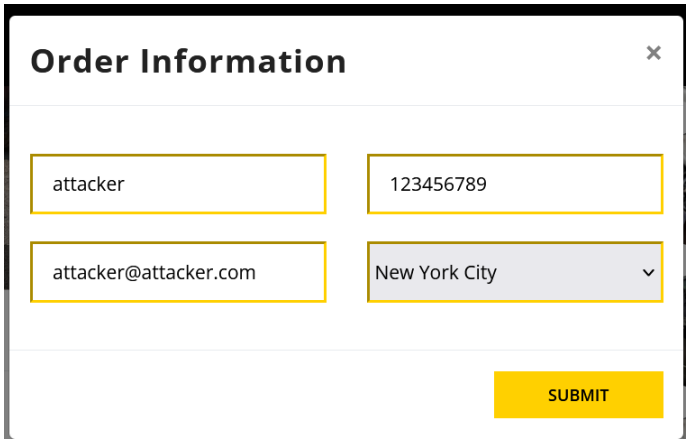
VULNERABILITY DETAILS

We can create a link that makes the user order car when he clicks on the link without, him knowing.

EXECUTION DEMONSTRATION

Order a car from the attacker account:





We can see that we ordered a car:

Catch the requset in the burp suite:

Request

Pretty | Raw | \n | Actions

```
1  POST /api.php HTTP/1.1
2  Host: 18.158.46.251:4662
3  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:103.0) Gecko/20100101 Firefox/103.0
4  Accept: */*
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Content-Type: application/x-www-form-urlencoded; charset=UTF-8
8  X-Requested-With: XMLHttpRequest
9  Content-Length: 187
10 Origin: http://18.158.46.251:4662
11 Connection: close
12 Referer: http://18.158.46.251:4662/car_details.html
13
14 action=order&user_id=38363439&car_name2=BMW+M3+2500&car_id=4&name=attacker&email=
   attacker%40attacker.com&phone=123456789&location=New+York+City&picture=
   assets%2Fimg%2Fcars%2Fcar_4%2F1.jpg
```

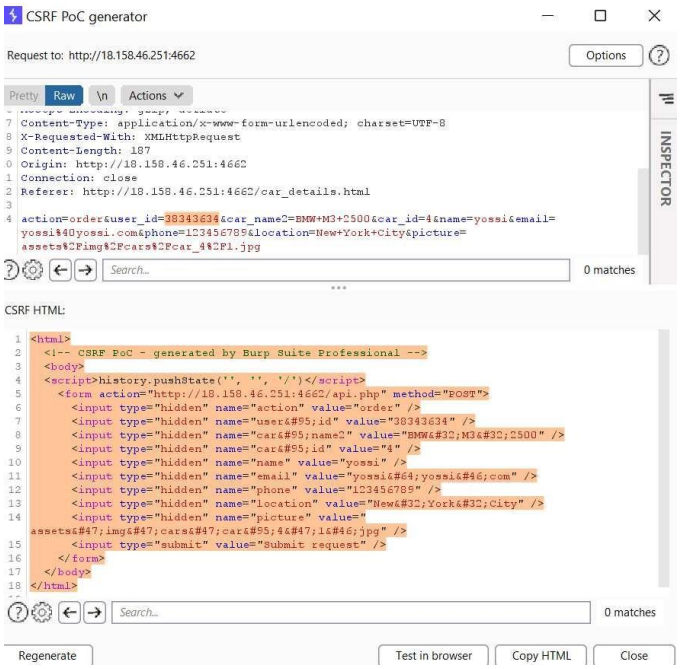
Change the parameters user_id,name,email:

Request

Pretty | Raw | \n | Actions

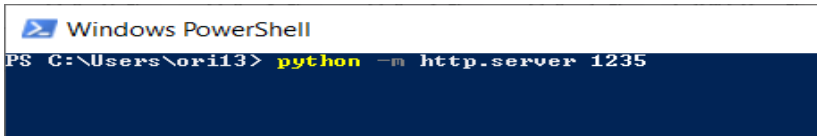
```
1  POST /api.php HTTP/1.1
2  Host: 18.158.46.251:4662
3  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:103.0) Gecko/20100101 Firefox/103.0
4  Accept: */*
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Content-Type: application/x-www-form-urlencoded; charset=UTF-8
8  X-Requested-With: XMLHttpRequest
9  Content-Length: 187
10 Origin: http://18.158.46.251:4662
11 Connection: close
12 Referer: http://18.158.46.251:4662/car_details.html
13
14 action=order&user_id=38343634&car_name2=BMW+M3+2500&car_id=4&name=yossi&email=yossi%40yossi.com&
   phone=123456789&location=New+York+City&picture=assets%2Fimg%2Fcars%2Fcar_4%2F1.jpg
```

Click on the right button->Engagement tools->Generate csrf poc:



Copy to a new file-csrf_car.html:

Share the file with python -m http.server 1235:

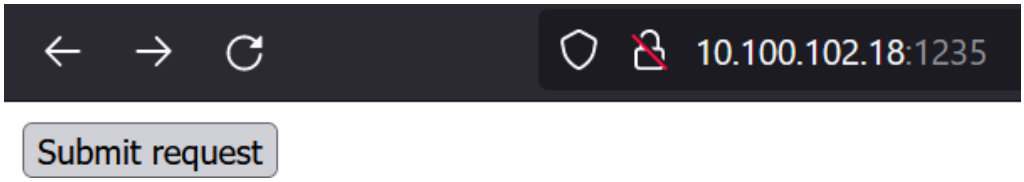


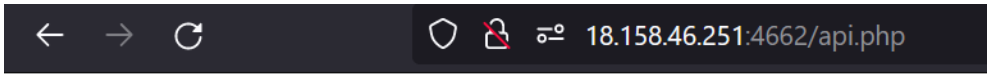
Open in the firefox with also yossi account session is conncted, Click on the "csrf_car.html":

Directory listing for /

- challanges 1-5.txt
- csrf.html
- csrf_bank.html
- csrf_car.html

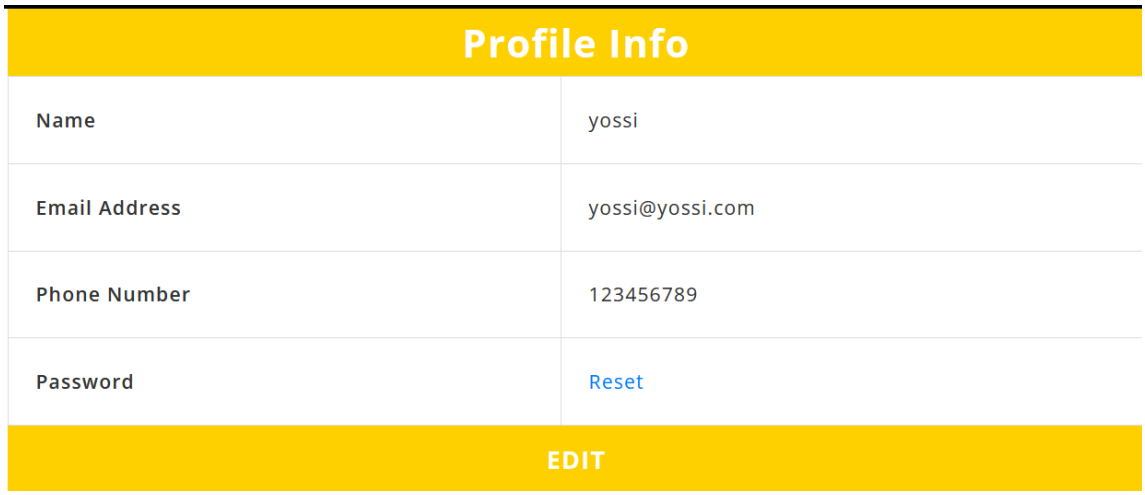
Click on "Submit request":

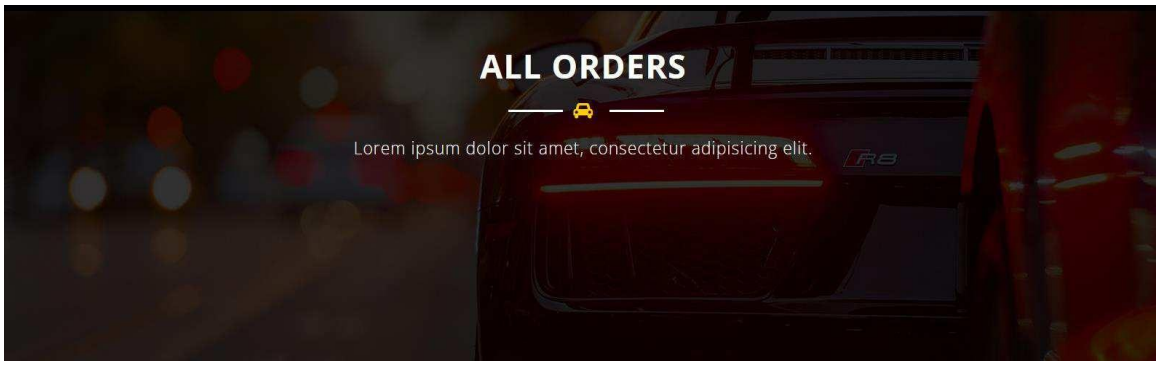


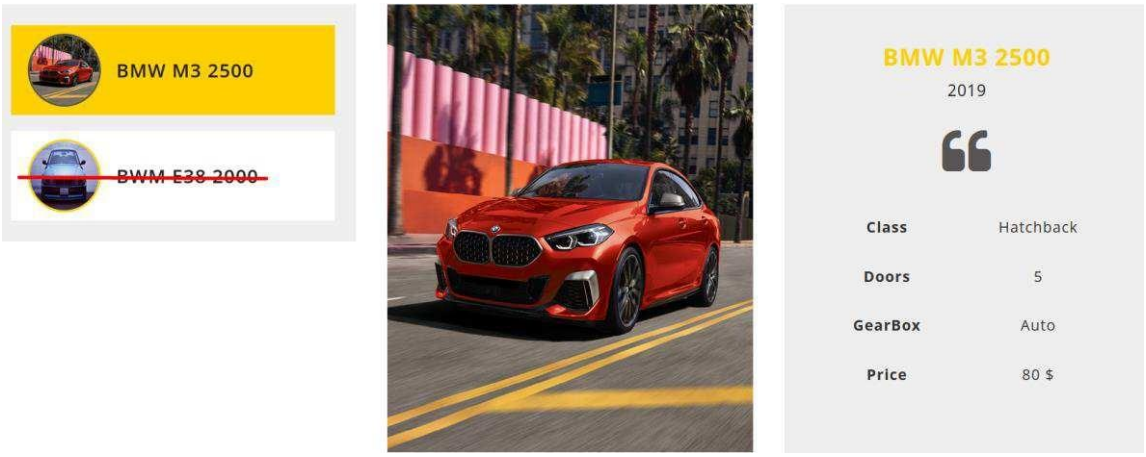


1

Go to the car rental webiste:







It's work!

RECOMMENDED RECTIFICATION

- Use cookies protection like SameSite - The SameSite attribute can be used to control whether and how cookies are submitted in cross-site requests. By setting the attribute on session cookies, an application can prevent the default browser behavior of automatically adding cookies to requests regardless of where they originate.
- CSRF token - CSRF tokens prevent CSRF because without token, attacker cannot create a valid request to the backend server.
- Double Submit Cookie - When a user authenticates to a site, the site should generate a (cryptographically strong) pseudo-random value and set it as a cookie on the user's machine separate from the session id. The server does not have to save this value in any way, that's why this pattern is sometimes also called Stateless CSRF Defense.

4.4 CROSS SITE SCRIPTING (XSS)

Severity | Medium Probability | Medium

VULNERABILITY DESCRIPTION

CROSS-SITE SCRIPTING (XSS) ATTACKS ARE A TYPE OF INJECTION, IN WHICH MALICIOUS SCRIPTS ARE INJECTED INTO OTHERWISE BENIGN AND TRUSTED WEBSITES. XSS ATTACKS OCCUR WHEN AN ATTACKER USES A WEB APPLICATION TO SEND MALICIOUS CODE, GENERALLY IN THE FORM OF A BROWSER SIDE SCRIPT, TO A DIFFERENT END USER.

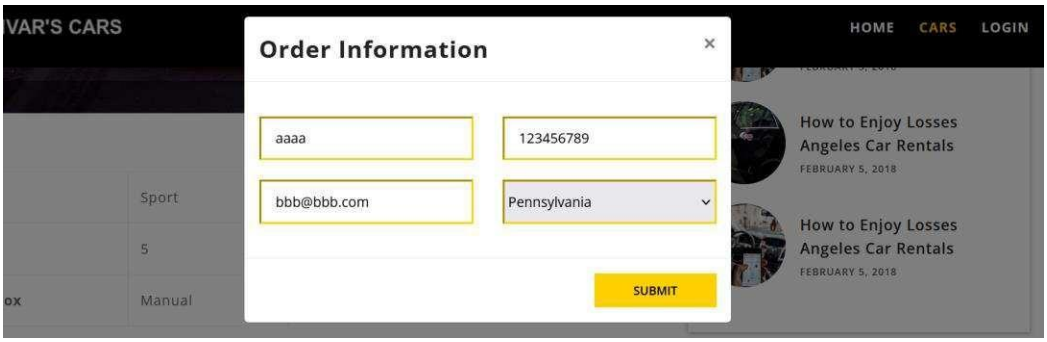
An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site.

VULNERABILITY DETAILS

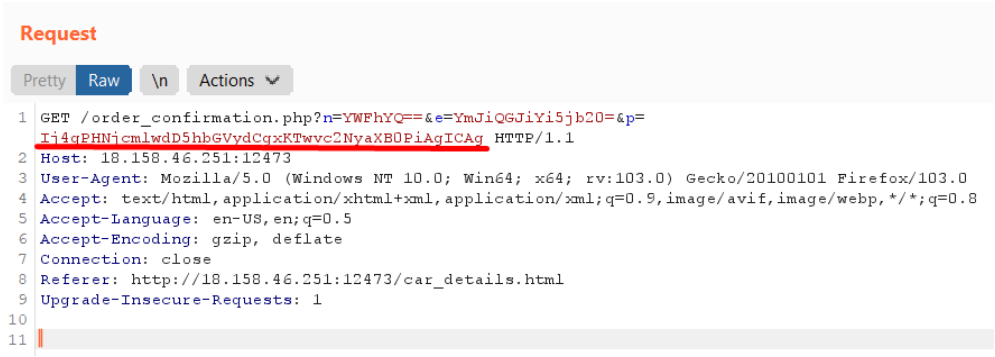
Run alert command in base64 crypted in the parameter p.

EXECUTION DEMONSTRATION

Try to order car without account:

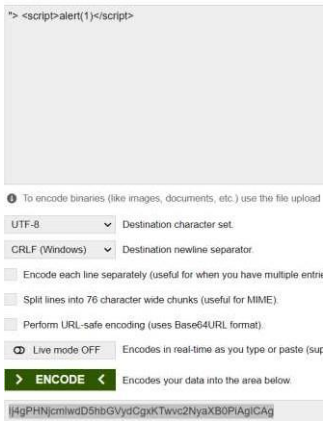


Catch the request in the burp suite:

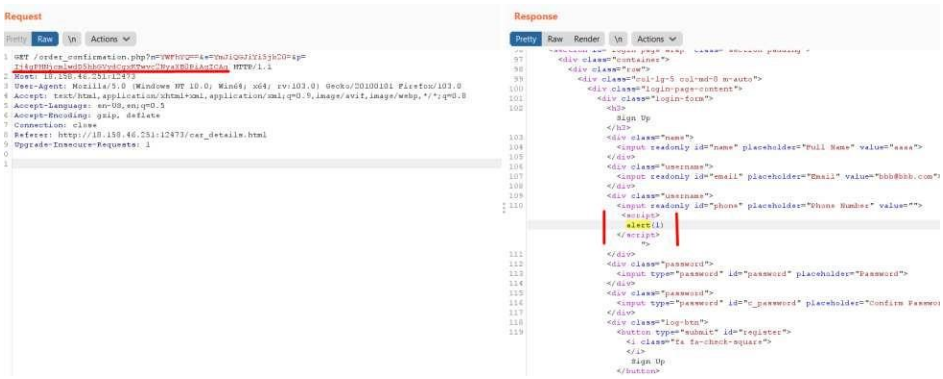


It can be seen that the data is encrypted!

Crypted with base64 website:



Paste in the parameter p:



Open in the Firefox:

Its work!

RECOMMENDED RECTIFICATION

- Use htmlentities- Convert all applicable characters to HTML entities, using this function may also lead to excessive encoding and may cause some content to display incorrectly. For example, if we use with < it changes to <
- Use appropriate response headers. To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the Content-Type and X-Content-Type-Options headers to ensure that browsers interpret the responses in the way you intend.
- Output Encoding is recommended when you need to safely display data exactly as a user typed it in. Variables should not be interpreted as code instead of text. This section covers each form of output encoding, where to use it, and where to avoid using dynamic variables entirely.

METHODOLOGY

The work methodology includes some or all of the following elements, to meet client requirements:

APPLICATION TESTS

- **Various tests to identify:**
  - Vulnerable functions.
  - Known vulnerabilities.
  - Un-sanitized Input.
  - Malformed and user manipulated output.
  - Coding errors and security holes.
  - Unhandled overload scenarios.
  - Information leakage.
- **General review and analysis (including code review tests if requested by the client). Automatic tools are used to identify security related issues in the code or the application.**
- **After an automated review, thorough manual tests are performed regarding:**
  - **Security functions:** Checking whether security functions exist, whether they operate based on a White List of a Black List, and whether they can be bypassed.
  - **Authentication mechanism:** The structure of the identification mechanism, checking the session ID's strength, securing the identification details on the client side, bypassing through the use of mechanisms for changing passwords, recovering passwords, etc.
  - **Authorization policy:** Verifying the implementation of the authorization validation procedures, whether they are implemented in all the application's interfaces, checking for a variety of problems, including forced browsing, information disclosure, directory listing, path traversal.
  - **Encryption policy:** Checking whether encryption mechanisms are implemented in the application and whether these are robust/known mechanisms or ones that were developed in-house, decoding scrambled data.
  - **Cache handling:** Checking whether relevant information is not saved in the cache memory on the client side and whether cache poisoning attacks can be executed.

- **Log off mechanism:** Checking whether users are logged off in a controlled manner after a predefined period of in activity in the application and whether information that can identify the user is saved after he has logged off.
- **Input validation:** Checking whether stringent intactness tests are performed on all the parameters received from the user, such as matching the values to the types of parameters, whether the values meet maximal and minimal length requirements, whether obligatory fields have been filled in, checking for duplication, filtering dangerous characters, SQL / Blind SQL injection.
- **Information leakage:** Checking whether essential or sensitive information about the system is not leaking through headers or error messages, comments in the code, debug functions, etc.
- **Signatures:** (with source code in case of a code review test)**:** Checking whether the code was signed in a manner that does not allow a third party to modify it.
- **Code Obfuscation:** (with source code in case of a code review test, or the case of a client-server application): Checking whether the code was encrypted in a manner that does not allow debugging or reverse engineering.
- **Administration settings:** Verifying that the connection strings are encrypted and that custom errors are used.
- **Administration files:** Verifying that the administration files are separate from the application and that they can be accessed only via a robust identification mechanism.
- **Supervision, documentation and registration functions:** Checking the documentation and logging mechanism for all the significant actions in the application, checking that the logs are saved in a secure location, where they cannot be accessed by unauthorized parties.
- **Error handling:** Checking whether the error messages that are displayed are general and do not include technical data and whether the application is operating based on the failsafe principle.

- **In-depth manual tests of application's business logic and complex scenarios.**

- **Review of possible attack scenarios, presenting exploit methods and POCs.**
- **Test results: a detailed report which summarizes the findings, including their:**
  - Description.
  - Risk level.
  - Probability of exploitation.
  - Details.
  - Mitigation recommendations.
  - Screenshots and detailed exploit methods.
- **Additional elements that may be provided if requested by the client:**
  - Providing the development team with professional support along the rectification process.
  - Repeat test (validation) including report resubmission after rectification is completed.

INFRASTRUCTURE TESTS

- **Questioning the infrastructure personnel, general architecture review.**
- **Various tests in order to identify:**
  - IP addresses, active DNS servers.
  - Active services.
  - Open ports.
  - Default passwords.
  - Known vulnerabilities.
  - Infrastructure-related information leakage.
- **General review and analysis. Automatic tools are used in order to identify security related issues in the code or the application.**
- **After an automated review, thorough manual tests are performed regarding:**
  - Vulnerable, open services.
  - Authentication mechanism.
  - Authorization policy.

- Encryption policy.
- Log off mechanism.
- Information leakage.
- Administrative settings.
- Administrative files.
- Error handling.
- Exploit of known security holes.
- Infrastructure local information leakage.
- Bypassing security systems.
- Networks separation durability.

- **In-depth manual tests of application's business logic and complex scenarios.**
- **Review of possible attack scenarios, presenting exploit methods and POCs.**
- **Test results: a detailed report which summarizes the findings, including their:**
  - Description.
  - Risk level.
  - Probability of exploitation.
  - Details.
  - Mitigation recommendations.
  - Screenshots and detailed exploit methods.
- **Additional elements that may be provided if requested by the client:**
  - Providing the development team with professional support along the rectification process.
  - Repeat test (validation) including report resubmission after rectification is completed.

FINDING CLASSIFICATION

Severity

The finding's severity relates to the impact which might be inflicted to the organization due to that finding. The severity level can be one of the following options, and is determined by the specific attack scenario:

Critical – Critical level findings are ones which may cause significant business damage to the organization, such as:

- Significant data leakage
- Denial of Service to essential systems
- Gaining control of the organization's resources (For example Servers, Routers, etc.)

High – High level findings are ones which may cause damage to the organization, such as:

- Data leakage
- Execution of unauthorized actions
- Insecure communication
- Denial of Service
- Bypassing security mechanisms
- Inflicting various business damage

Medium – Medium level findings are ones which may increase the probability of carrying out attacks, or perform a small amount of damage to the organization, such as –

- Discoveries which makes it easier to conduct other attacks
- Findings which may increase the amount of damage which an attacker can inflict, once he carries out a successful attack
- Findings which may inflict a low level of damage to the organization

Low – Low level findings are ones which may inflict a marginal cost to the organization, or assist the attacker when performing an attack, such as –

- Providing the attacker with valuable information to help plan the attack
- Findings which may inflict marginal damage to the organization
- Results which may slightly help the attacker when carrying out an attack, or remaining undetected

Informative – Informative findings are findings without any information security impact. However, they are still brought to the attention of the organization.