WEB APPLICATION PENETRATION TESTING REPORT for Microweber

Prepared by:

GrozdniyAndy of XSS.is

20/11/2023

Table Of Contents

Table Of Contents

Executive Summary

This report presents the results of the Black Box penetration testing for the Microweber Open-Source CMS. The recommendations provided herein are structured to facilitate the remediation of the identified security risks. This document serves as a formal letter of attestation for the recent Microweber CMS penetration testing. Evaluation ratings compare information gathered during the engagement to "best in class" criteria for security standards. We believe that the statements made in this document provide an accurate assessment of Microweber's current security as it relates to Microweber data. We highly recommend reviewing the Summary section for business risks and High-Level Recommendations to better understand the risks and discovered security issues.

Grading Criteria

The penetration test for Microweber utilized the Common Weakness Enumeration (CWE) and the OWASP Top 10 as key grading criteria. The CWE framework was employed to identify and categorize specific vulnerabilities, while the OWASP Top 10 served as a guide to focus on critical web application security risks. The Common Vulnerability Scoring System (CVSSv3) was applied to quantify the severity of identified vulnerabilities, aiding in prioritizing remediation efforts based on potential impact.

Project Objectives

The overarching goal of the penetration test was to assess the security posture of Microweber, uncovering vulnerabilities in line with the CWE and OWASP Top 10 frameworks. Specific objectives included the identification and validation of security weaknesses, the quantification of risk using CVSSv3, and the provision of actionable recommendations to enhance the overall security of Microweber.

Methodology

The penetration test followed a black-box approach. The methodology included:

1. Vulnerability Assessment: Applying automated tools and manual techniques based on CWE and OWASP Top 10 to discover and validate vulnerabilities.
2. Exploitation: Ethical exploitation of identified vulnerabilities to assess impact, with severity quantified using CVSSv3.
3. Post-Exploitation: Analyzing the extent of compromise and determining potential risks associated with successful exploitation.
4. Reporting: Documenting findings, including identified vulnerabilities, their CWE categorization, OWASP Top 10 relevance, and CVSSv3 scores, with actionable recommendations for remediation.

Summary of Findings

Vulnerability #	Score
1	5.3 AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
2	5.3 AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
3	3.8 AV:N/AC:L/PR:H/UI:N/S:U/C:L/I:L/A:N
4	8.1 AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:N
5	4.6 AV:N/AC:L/PR:L/UI:R/S:U/C:L/I:L/A:N
6	3.5 AV:N/AC:L/PR:L/UI:R/S:U/C:N/I:L/A:N

Findings

2. Broken Access Control

Broken Access Control refers to a security vulnerability that occurs when an application or system fails to properly enforce restrictions on what authenticated users are allowed to do. This vulnerability arises when access controls, such as permissions and restrictions, are not effectively implemented or are easily bypassed. It allows unauthorized users to access sensitive data, perform actions they shouldn't be allowed to, or exploit functionalities that should be restricted.

The consequences of Broken Access Control can be severe, as it can lead to unauthorized access to confidential information, unauthorized modification of data, and other security breaches. Common examples include a user being able to view or modify another user's data, gaining access to privileged functionalities without proper authorization, or manipulating URLs or parameters to access restricted areas of a website or application.

Reference: https://owasp.org/Top10/A01_2021-Broken_Access_Control/

2.1 Files or Directories Accessible to External Parties

Description:
The product makes files or directories accessible to unauthorized actors, even though they should not be.

Location:
http://microweber.local/userfiles/install_log.txt
http://microweber.local/api/image-generate-tn-request/{Number}?saved

Impact:
Some files are accessible to unauthenticated users and this may lead to sensitive data leakage.

Reference: https://cwe.mitre.org/data/definitions/552.html

Vulnerability Detail [1]
File install_log.txt has logs stored and they can be viewed with anyone who knows the right value of parameter ‘_’. Those logs store installed modules and other debug information which may be considered as sensitive.

Proof of Vulnerability [1]
Install the microweber and check both requests (client-side) and request logs (server-side).

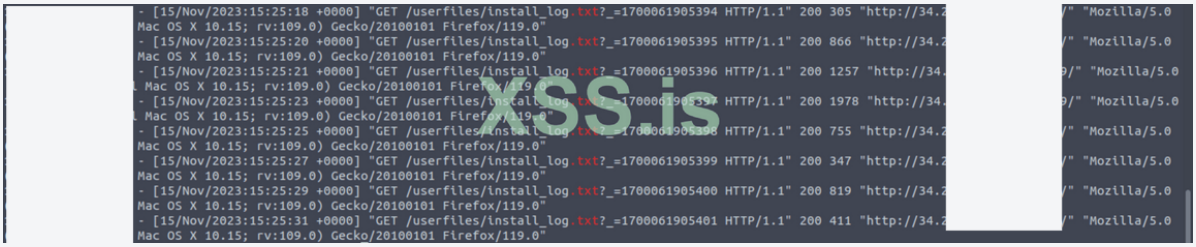

Image [1]

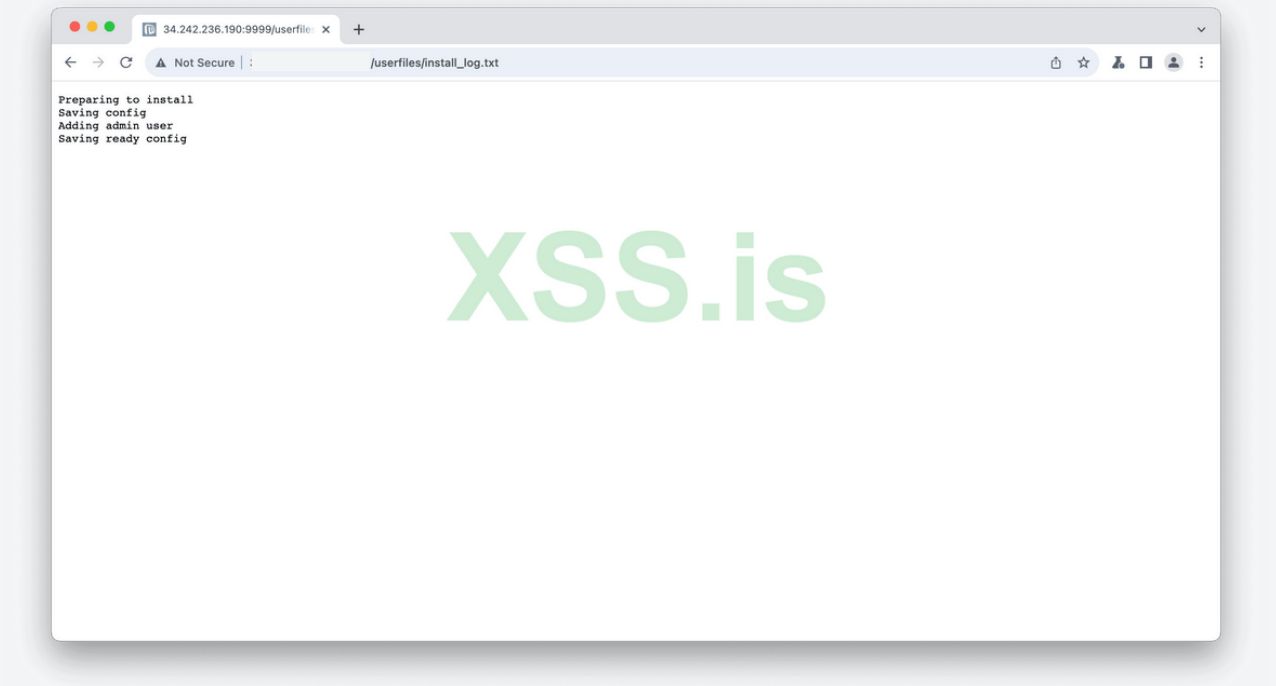

Image [2]

Image number 2 shows only partial logs, full logs to be seen we should open URL as in Image [1]. This file is stored in the server and not deleted. When I googled inurl:install_log.txt

Image [3]

Vulnerability Detail [2]
It is possible to view images uploaded by administration but not used.

Proof of Vulnerability [2]
Upload any image in any way, but don’t use it anywhere. To see uploaded images you can use Media Libary.

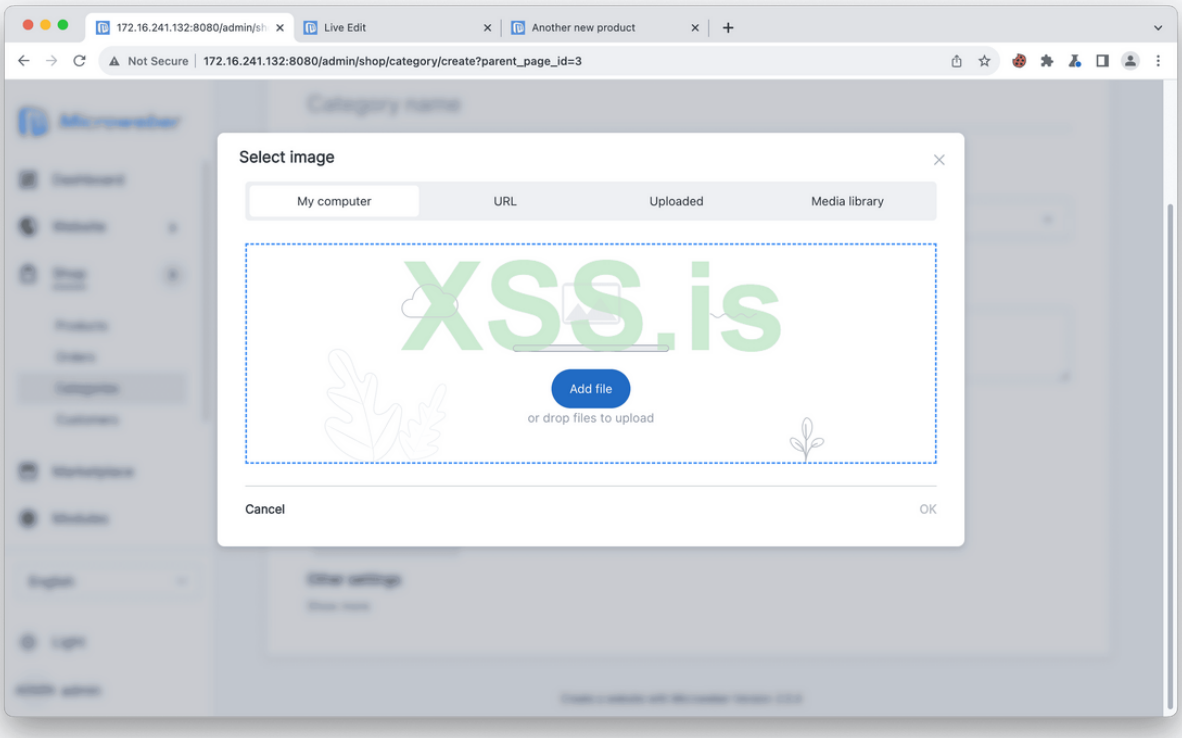

Image [4]

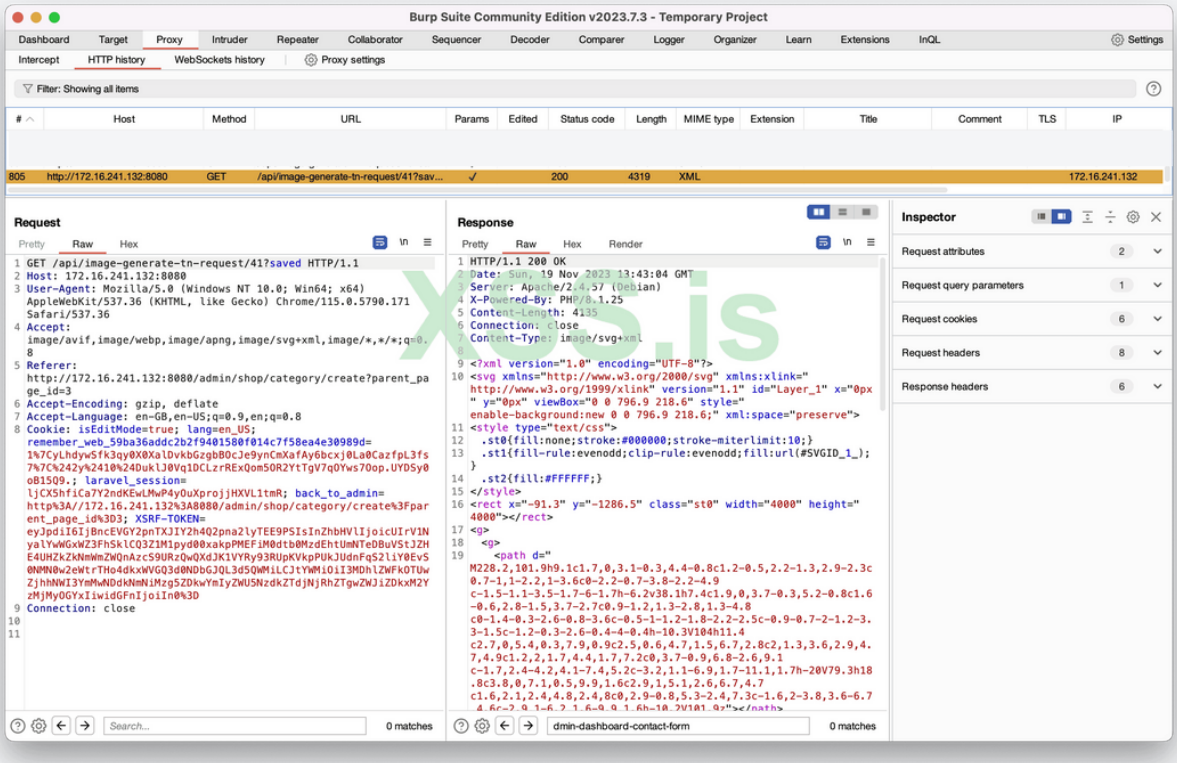

Image [5]

Public
This report is made as effort to increase security in open source projects.

I was able to view image 46 as unauthenticated user even though I shouldn't be.

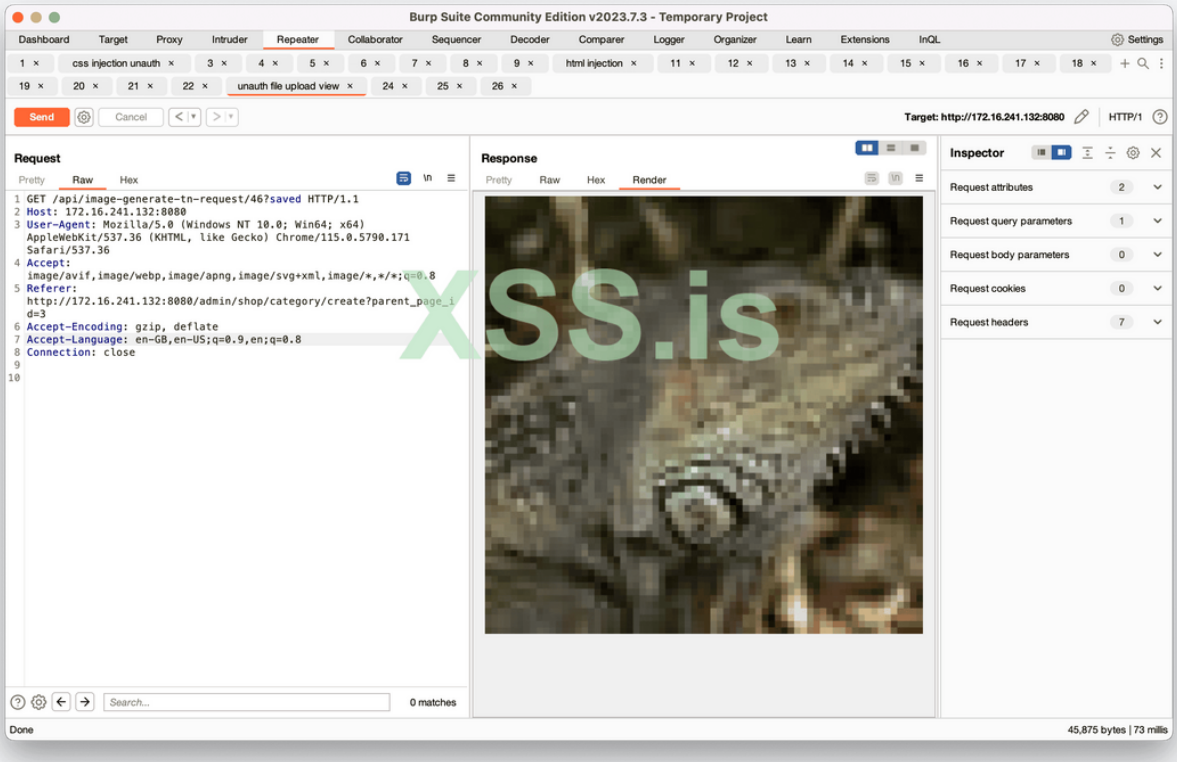

Image [6]

Remediation
remove the log file after installation and restrict access to view images only to authorised users.

2.2 Cross-Site Request Forgery (CSRF)

Description:
The web application does not, or can not, sufficiently verify whether a well-formed, valid, consistent request was intentionally provided by the user who submitted the request.

Location:
http://microweber.local/livewire/message/admin::edit-user.update-profile-form

Impact:
The consequences will vary depending on the nature of the functionality that is vulnerable to CSRF. An attacker could effectively perform any operations as the victim. If the victim is an administrator or privileged user, the consequences may include obtaining complete control over the web application - deleting or stealing data, uninstalling the product, or using it to launch other attacks against all of the product's users. Because the attacker has the identity of the victim, the scope of CSRF is limited only by the victim's privileges.

Reference: https://owasp.org/www-community/attacks/csrf

Vulnerability Detail [3]
When uploading an image, for example a profile photo, it gets stored in the server and when saving it the full URL is used to save it. Attacker can put URL from external source. It may be intended when adding a category image, but not when uploading a profile photo. Vulnerability exists in uploading profile photo. Also there is a check for extension, which can be bypassed

Proof of Vulnerability [3]
In Image [7] we put external URL, for example some random image from github and in the Image [8] we can see that it is in source attribute. Application would allow me to add URLs that end only with .jpg, so I used #.jpg, it will comment .jpg and I would be able to add any URL I want, which will allow me to make anyone viewing the profile send any malicious request I want. As you can see in Image [9], I added # before the .jpg and in Image [10], the request was sent without #.jpg, as it is a comment. This way we bypassed the security implemented.

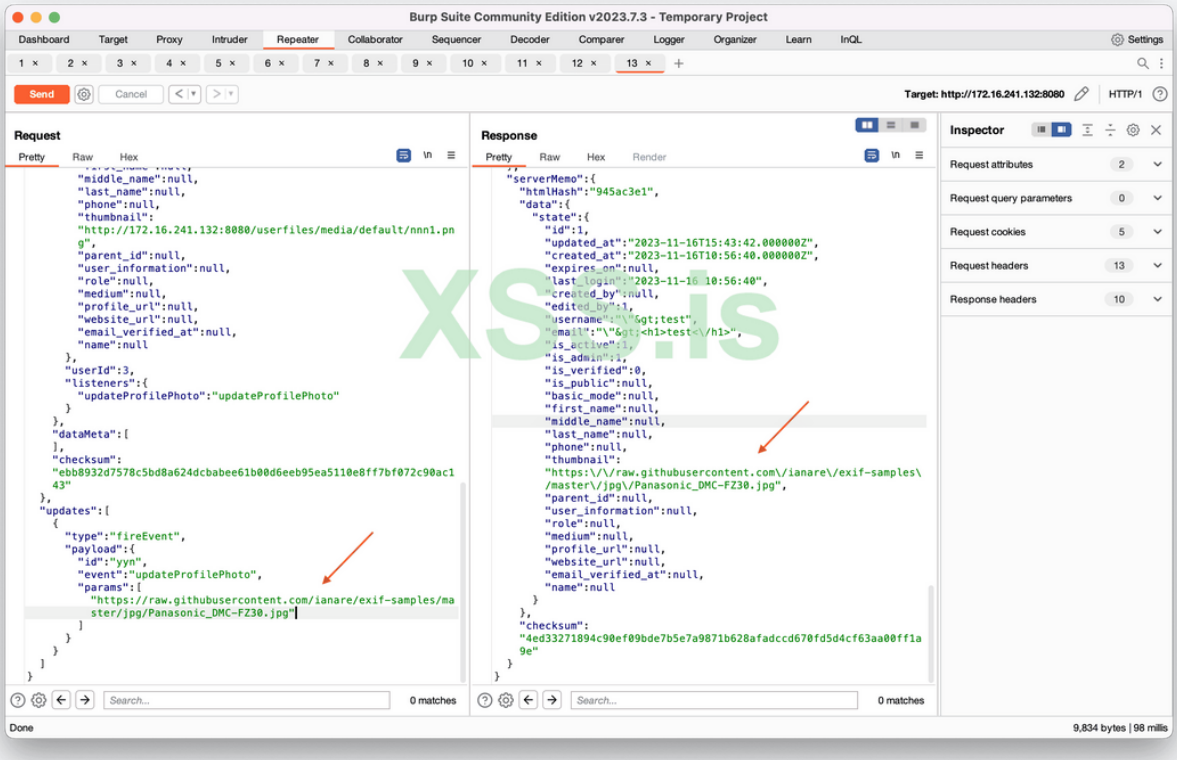

Image [7]

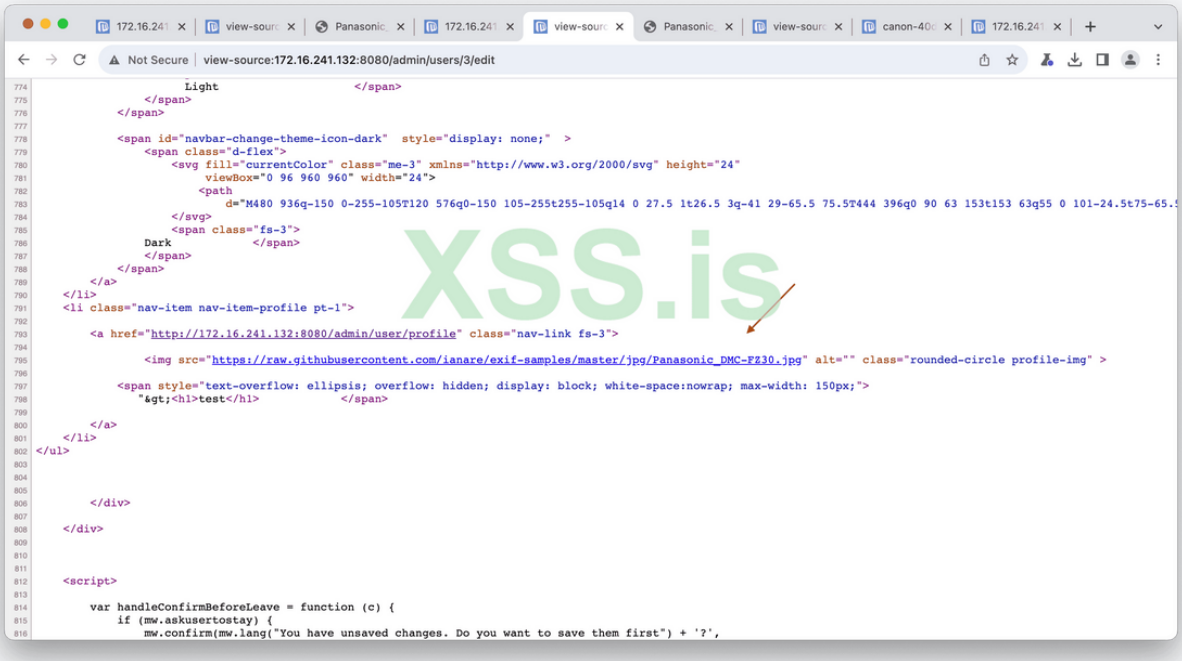

Image [8]

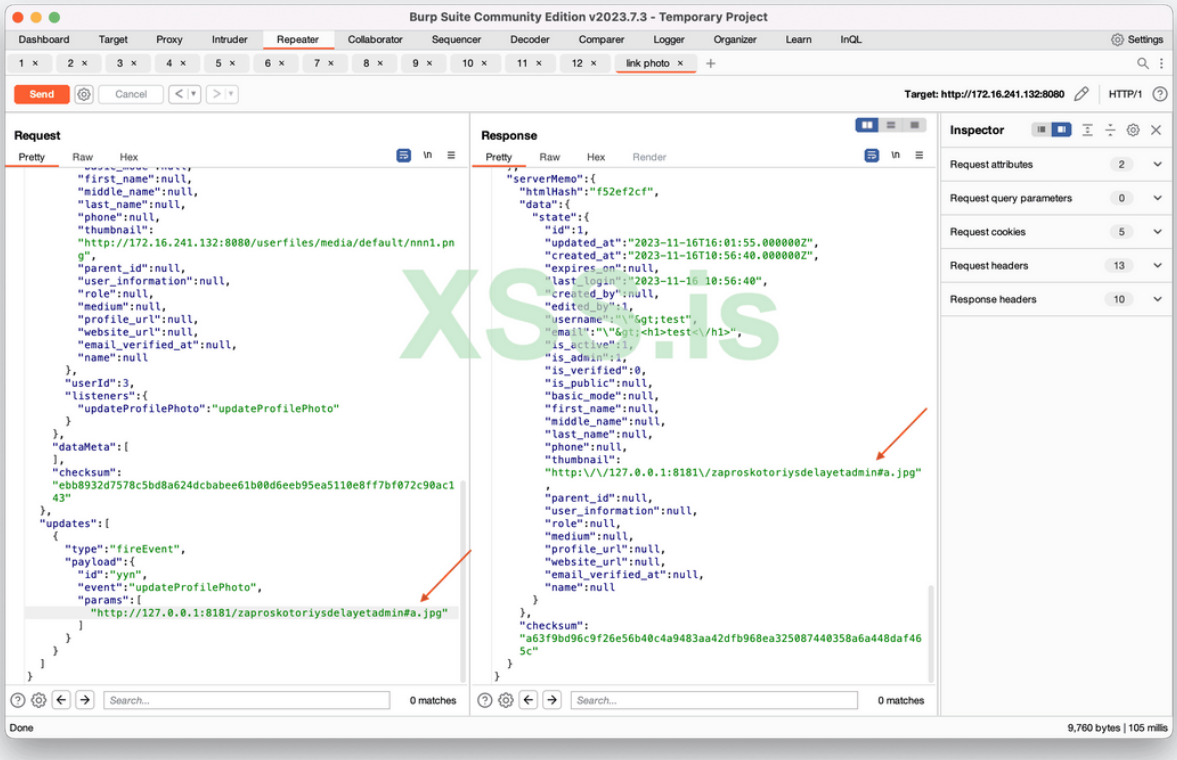

Image [9]

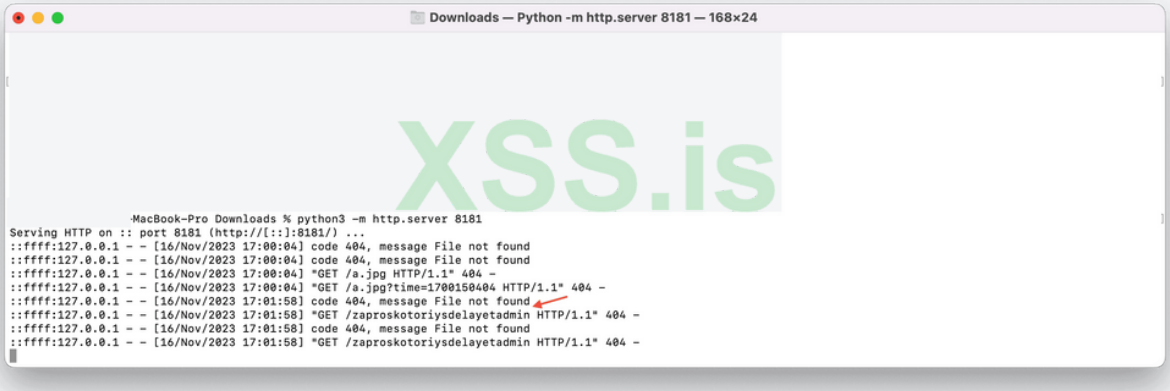

Image [10]

Remediation
Use relative paths when adding an image instead of using full URLs.

3. Injection

An application is vulnerable to attack when:

- User-supplied data is not validated, filtered, or sanitized by the application.
- Dynamic queries or non-parameterized calls without context-aware escaping are used directly in the interpreter.
- Hostile data is used within object-relational mapping (ORM) search parameters to extract additional, sensitive records.
- Hostile data is directly used or concatenated. The SQL or command contains the structure and malicious data in dynamic queries, commands, or stored procedures.

Reference: https://owasp.org/Top10/A03_2021-Injection/

3.1 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

Description:
The product does not neutralize or incorrectly neutralizes user-controllable input before it is placed in output that is used as a web page that is served to other users.

Location:
http://microweber.local/login?username={payload}&password={payload}
http://microweber.local/module
http://microweber.local/api/save_option

Impact:
In some circumstances it may be possible to run arbitrary code on a victim's computer when cross-site scripting is combined with other flaws.

Reference: https://cwe.mitre.org/data/definitions/79.html

Vulnerability Detail [4]
There is XSS vulnerability in the login. The vulnerability arises from the ability of inserting credentials in GET request, which then allows attacker to bypass the security measures implemented and execute javascript code.

Proof of Vulnerability [4]
I simply used request to add username and password and then checked where it gets reflected in the source code, Image [11]. Then instead of escaping the input tag, I executed the javascript code inside it Image [12], Image [13]. Same can be done for password parameter, Image [14].

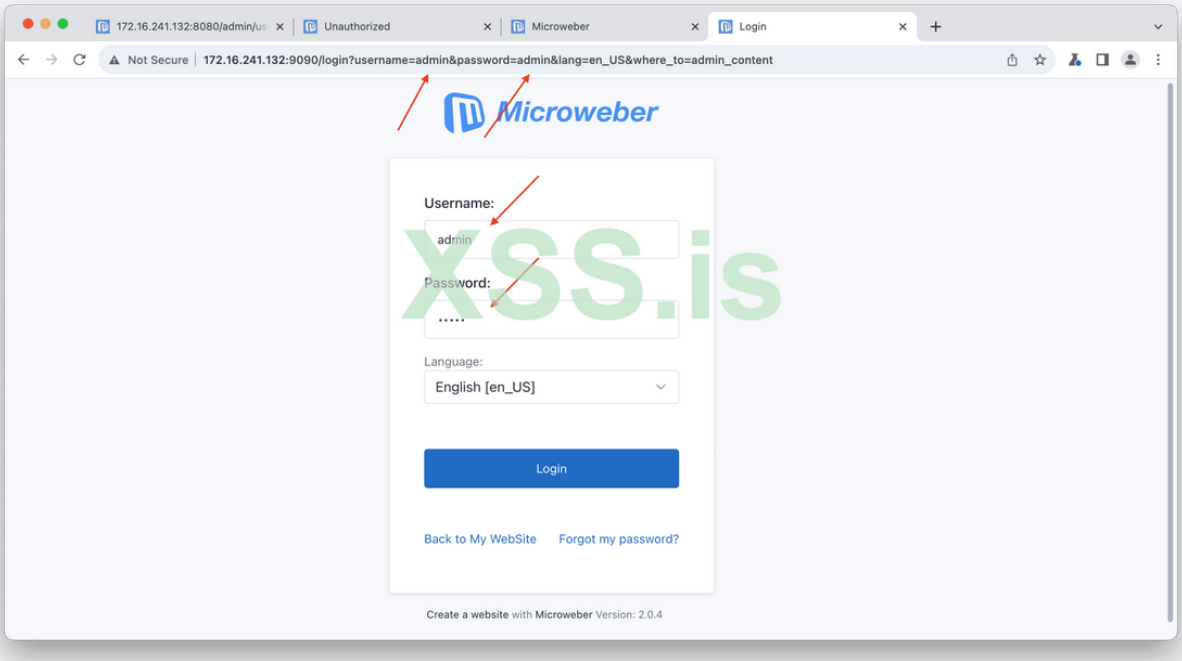

Image [11]

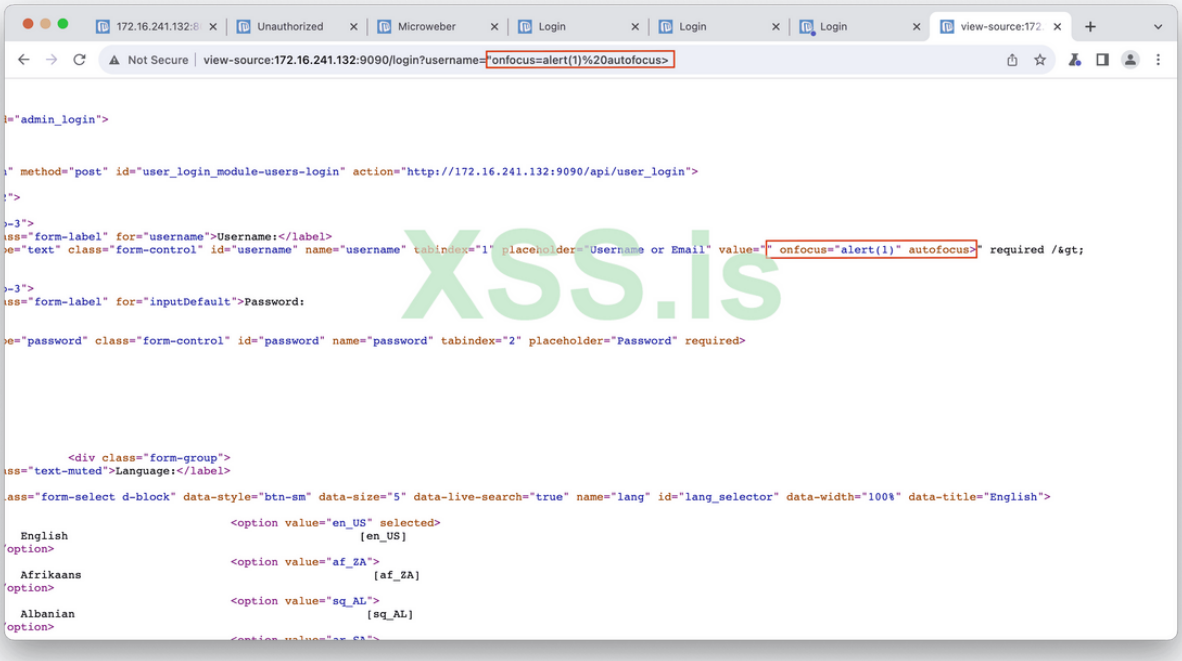

Image [12]

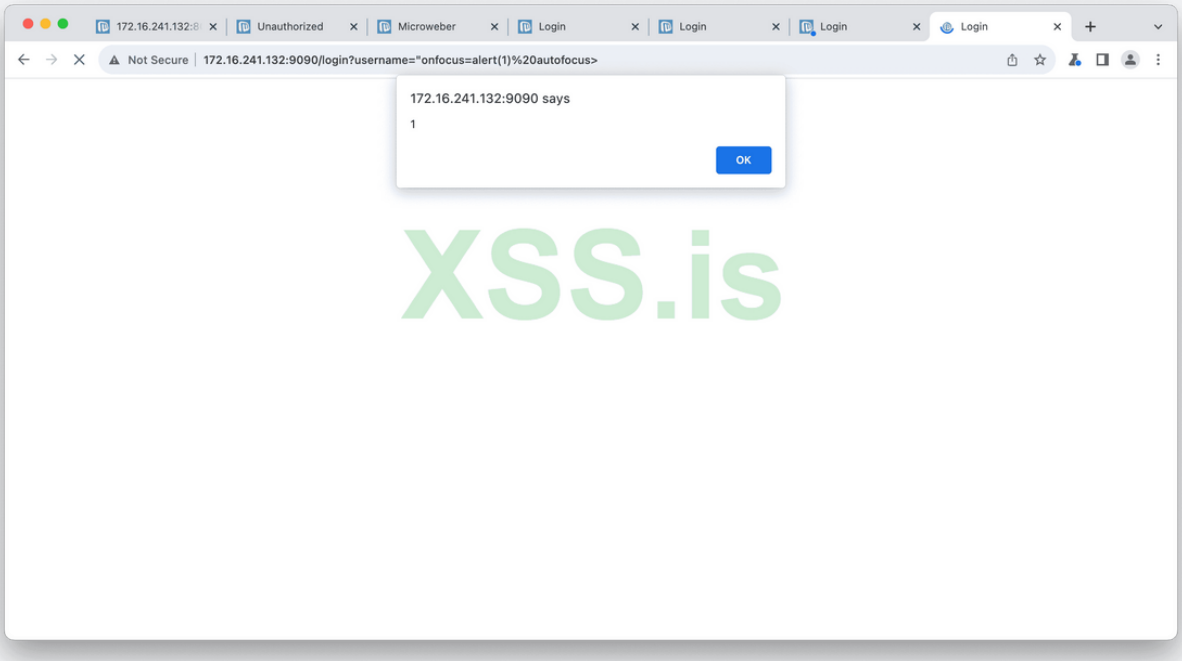

Image [13]

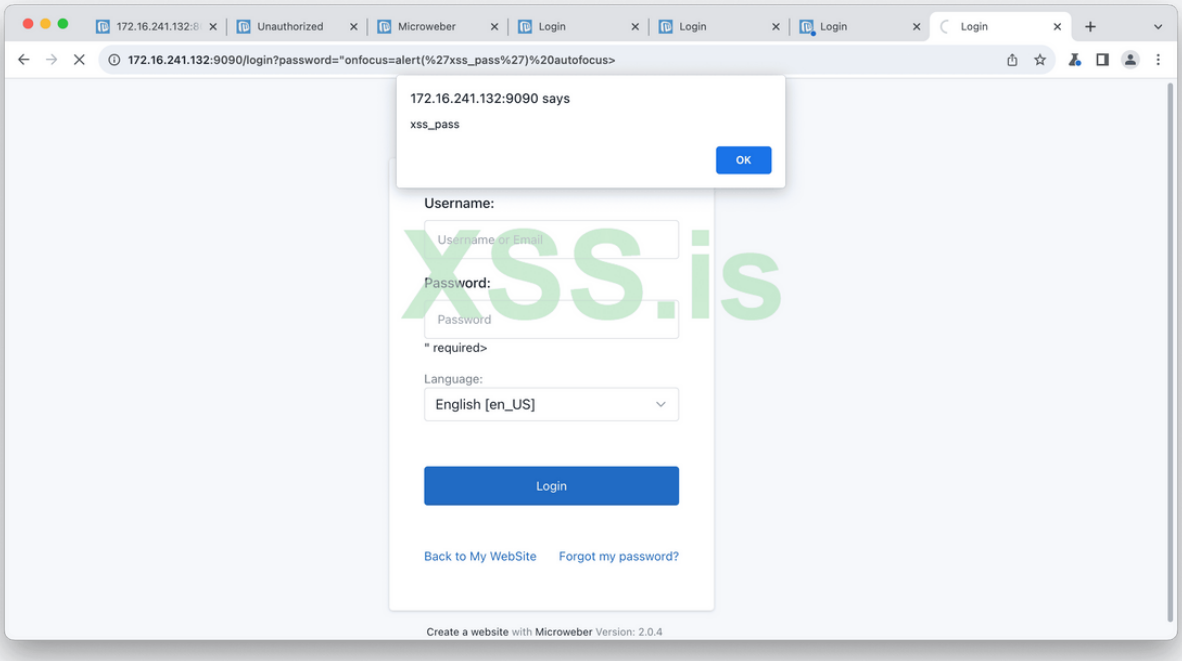

Image [14]

Vulnerability Detail [5]
There is a CSS Injection while sending request to /module/. Module adds parameters inside div which can be misused.

Proof of Vulnerability [5]
In Image [15] I added background URL and it sent a request to my server, Image [16]. In this case the style attribute was injected with external url. More information about CSS injection: https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/11-Client_Side_Testing/05-Testing_for_CSS_Injection

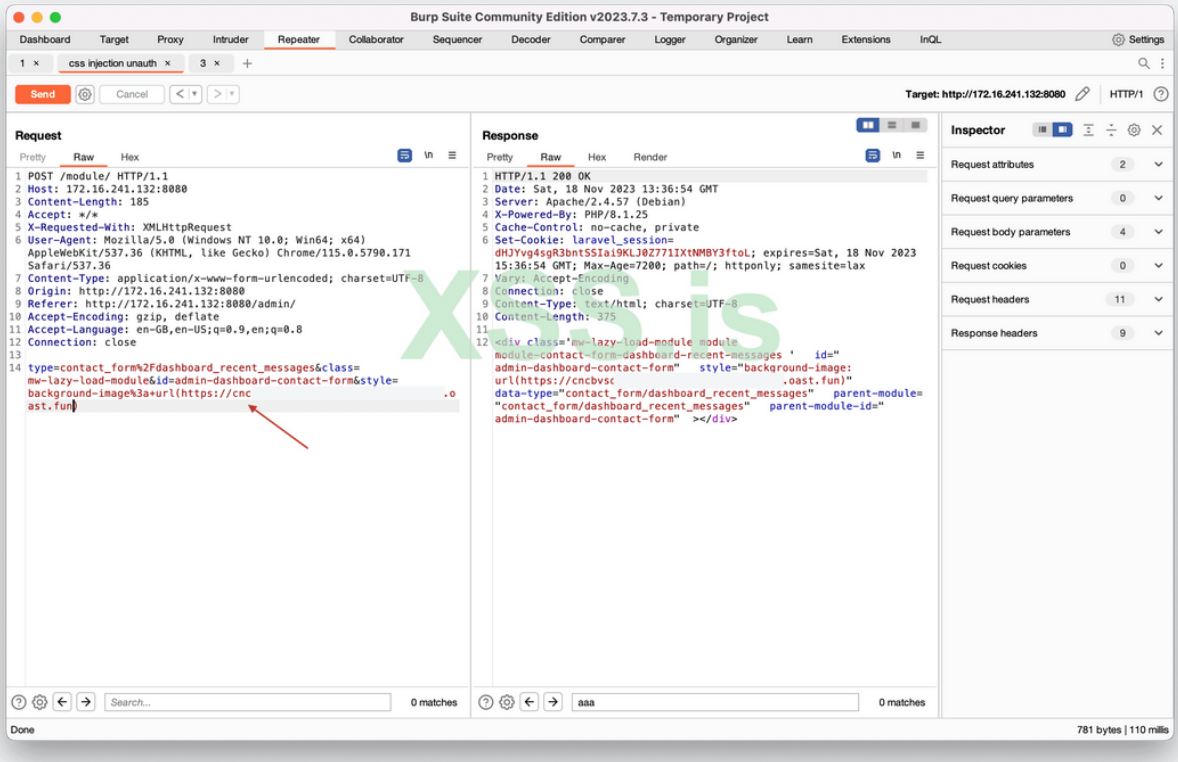

Image [15]

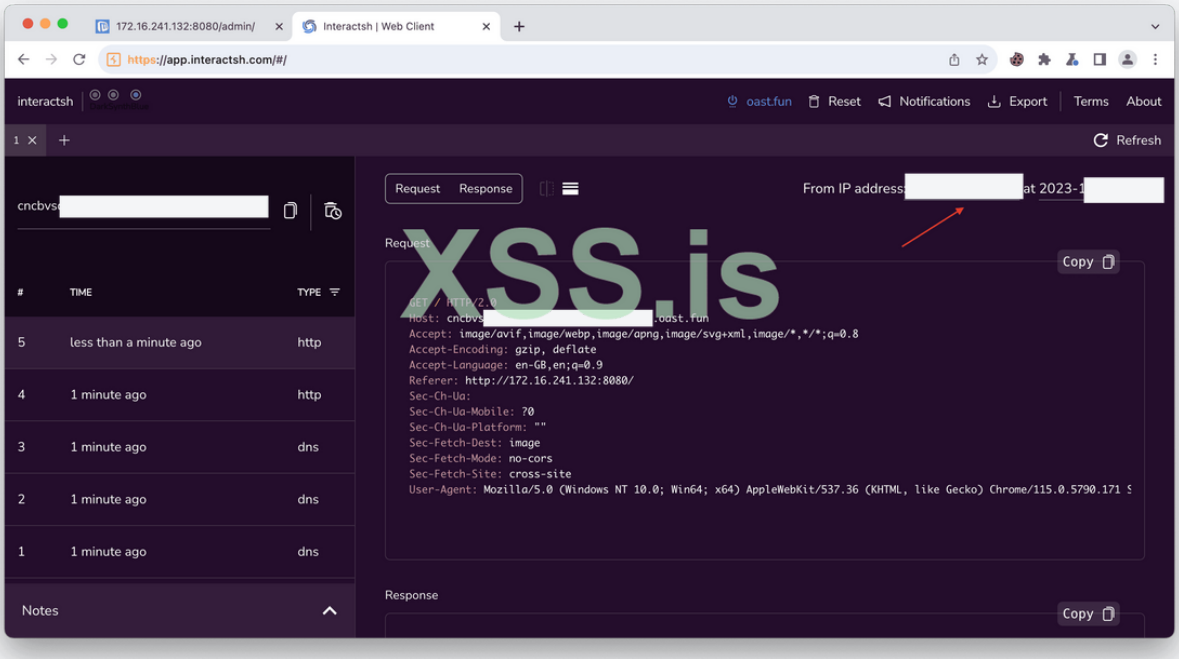

Image [16]

Vulnerability Detail [6]
There is HTML Injection in the /api/save_option. The vulnerability appears while editing email.

Proof of Vulnerability [6]
In Image [17] I edit email sender name and use a payload. In Image [18] we can see that our payload worked and it is in all the fields which can be seen in the picture.

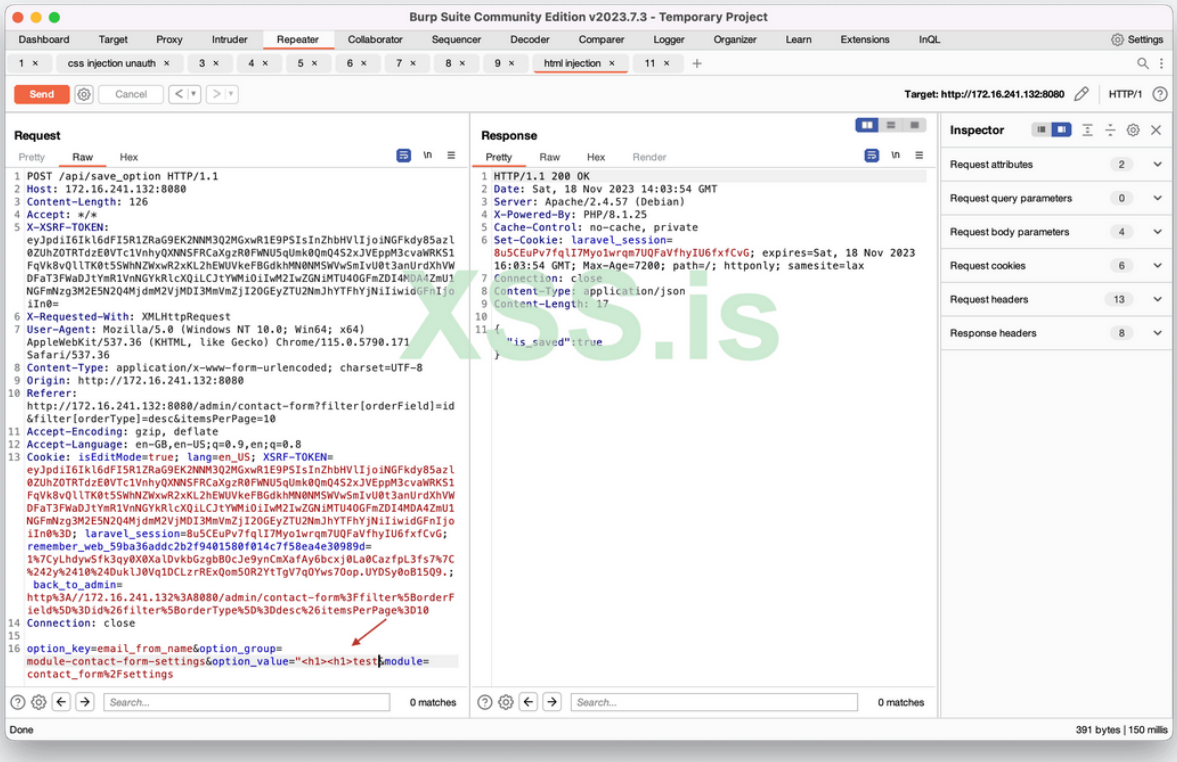

Image [17]

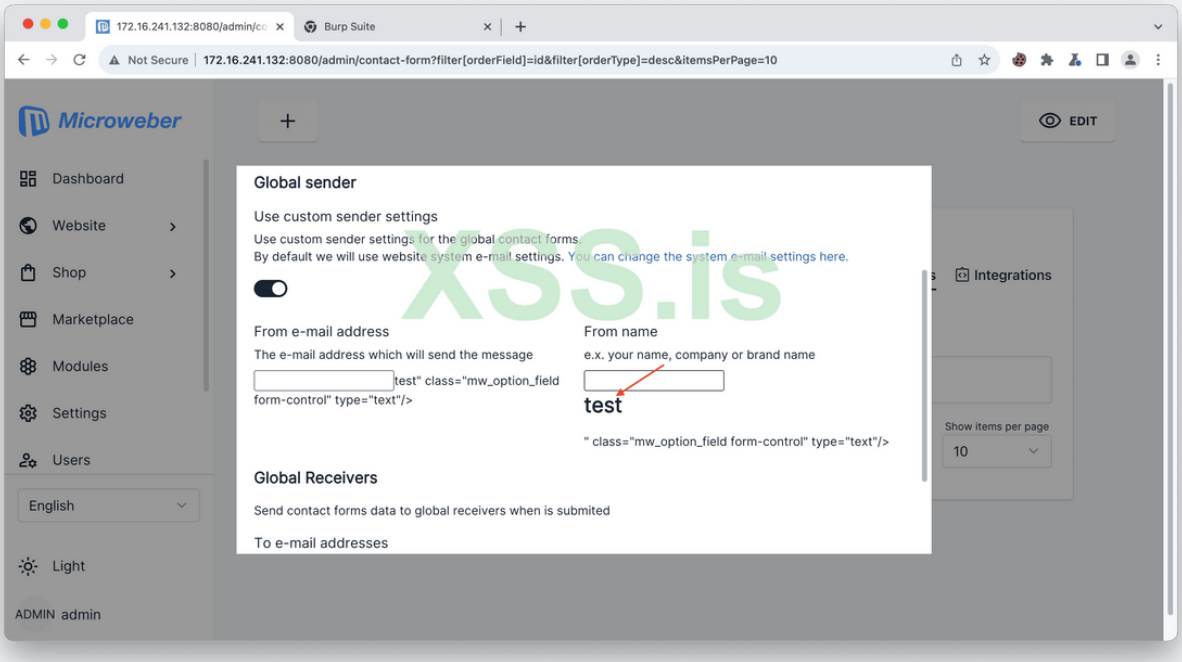

Image [18]