Lab 3

Test Case Report

Group 1

, Abdul-Ramon Adeyemi Folarin, &

, , &

School of Information Technology Administration & Security, Seneca Polytechnic

WAS705NBB: Application Security Methodology

Professors Suranjit Paul & Mark Shtern

June 11, 2026

Student Integrity Statement: I/we declare that the attached assignment is my/our own work in accordance with the Seneca Academic Policy. No part of this assignment has been copied manually or electronically from any other source (including web sites) or distributed to other students

Table of Contents

Executive Summary 4
1 Introduction 4
2 Test Environment and Setup 5
2.1 Test Account Creation 5
2.2 API Token Generation 5
2.3 Authenticated Access Verification 6
2.4 Intercepting Proxy Setup 7
3 Identification of User Inputs 7
3.1 Input Vector Inventory 9
4 Mandatory Controls Testing 9
4.1 Authentication 9
4.2 Session Management 9
4.3 Access Control 12
4.4 Error Handling 13
4.5 HTTP Method, CORS, and Login Rate-Limiting 13
5 Fuzzing Implementation and Analysis 15
5.1 Content and Route Discovery 15
5.2 Payload-Based Value Fuzzing 16
5.3 Mutation Fuzzing 16
5.4 Header and Cookie Fuzzing 17
5.5 Target-Side Log Monitoring 18
6 Injection and Application-Logic Testing 18
6.1 SQL Injection 18
6.2 Stored XSS - Name Field 19
6.3 Stored XSS - Description Field 20
6.4 Cross-Site Request Forgery 21
6.5 Server-Side Template Injection (Twig Invoice Sandbox) 22
6.6 Mass Assignment - Preferences API 24
7 Specialized Scanners and Configuration Review 25
7.1 Web Server Misconfiguration Scan (Nikto) 25
7.2 Manual HTTP Security Header Audit 25
7.3 Clickjacking and Framing 26
7.4 Tool Selection and Justification 27
8 OWASP Top 10 2025 Coverage 28
9 Feature Coverage and Prioritization 28

10 Traceability: Lab 2 Priorities to Lab 3 Testing 29
11 Conclusion 29
12 Lab 3 Task Checklist 30
References 31

Executive Summary

This report documents the group's black-box and lightly grey-box security testing of Kimai, an open-source PHP and Symfony time-tracking application. Testing was carried out across two instances that the group deployed during the lab: version 2.58.0, tested from a low-privilege ROLE_TEAMLEAD account (carol), and version 2.59.0, tested from standard-user accounts (alice and bob) with an administrator account used only for read-back verification. Both instances are documented so the evidence matches the version each test was run against.

The assessment covered the four mandatory controls, fuzzing, injection and application-logic testing, access control, and specialized scanning, mapped to the OWASP Top 10 2025. The core security controls held. Authentication rejected missing and invalid tokens, access control returned 403 for cross-user reads and writes, object-level authorization on the timesheet API held across an enumerated range, no SQL injection was found, stored-XSS attempts were blocked by server-side validation or neutralized by output encoding, the Twig invoice sandbox blocked every code-execution and secret-access attempt, and the preferences mass-assignment weakness tracked as CVE-2026-40486 was not present in 2.59.0.

The observations were lower-severity: missing HTTP security headers (Content-Security-Policy, X-Frame-Options, X-Content-Type-Options) confirmed by both a manual audit and Nikto; reflection of an arbitrary Origin header on the 2.58.0 instance; the ability to frame public pages, with authenticated framing mitigated by the SameSite=Lax cookie; a signature-based remember-me token that is not invalidated server-side on logout (Low to Medium, conditional on possession of the token); and the Twig sandbox permitting constant(), which can disclose the PHP version (Low).

One discrepancy is flagged for the group to reconcile. The login brute-force test produced different results on the two instances: on 2.58.0 ten sequential failed logins each returned HTTP 200 with no lockout seen in that run, while on 2.59.0 repeated failed attempts triggered a lockout with a five-minute wait message. This is discussed in Section 4.5 and should be settled with a single consistent test before final submission.

1 Introduction

Target and versions. The application under test is Kimai, deployed through Docker Compose. Because the group deployed the application at different times, the Docker image tag resolved to two adjacent published builds, and testing was therefore performed against two instances: Kimai 2.58.0 (confirmed in Fig. 3) and Kimai 2.59.0 (confirmed during the 2.59.0 testing). Rather than discard either set of evidence, both versions are documented and each finding is tied to the version it was observed on. The differences between the two builds are small, and where a test was run on only one instance this is stated.

Standard and scope. Findings are mapped to the OWASP Top 10 2025, the current edition at the time of testing, in which Server-Side Request Forgery is consolidated into A01 Broken Access Control and two new categories (A03 Software Supply Chain Failures and A10 Mishandling of Exceptional Conditions) are introduced. Testing was performed across the network from a Kali Linux host against the target instances, using low-privilege accounts and their API tokens, to reflect a realistic authenticated attacker rather than a localhost administrator.

2 Test Environment and Setup

Before security testing began, a controlled test environment was prepared to ensure that all activities could be performed safely and consistently. The objective of this phase was to establish authenticated access to the application, verify communication between the testing workstation and target system, and confirm that monitoring and interception tools were functioning correctly. These steps ensured that subsequent testing activities were conducted against a fully operational and properly configured deployment of Kimai.

Table 1 Test environment and accounts across the two documented instances.

Component	Detail
Attacker host	Kali Linux (192.168.43.145)
Target host	Ubuntu Server with Docker (192.168.43.146:8920)
Instance A	Kimai 2.58.0, tested from ROLE_TEAMLEAD account carol (user ID 2)
Instance B	Kimai 2.59.0, tested from standard users alice and bob, with an admin account for read-back only
Core tooling	Burp Suite Community, ffuf, Nikto, sqlmap, curl, SecLists wordlists

2.1 Test Account Creation

A dedicated low-privilege user account was created to simulate a realistic authenticated user. Using a non-administrative account allowed security testing to focus on authorization boundaries, privilege restrictions, and potential escalation opportunities that could affect regular users of the application.

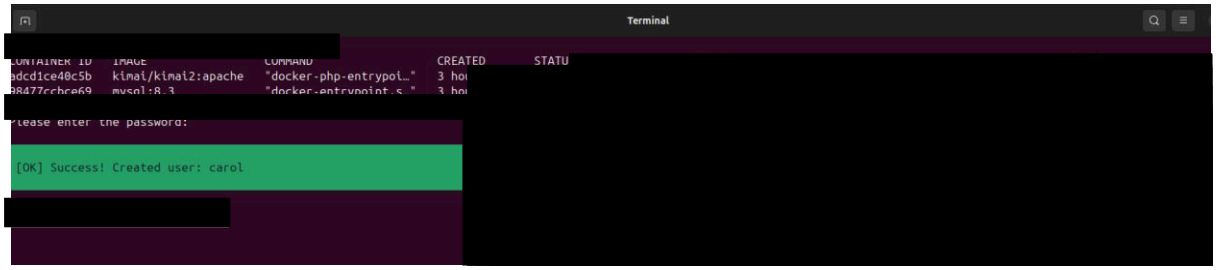


Figure 1 Creation of the ROLE_TEAMLEAD test account used as the low-privilege attacker identity.

Figure 1 shows the creation of the low-privilege account used throughout the assessment. The Docker host confirms that both the application and database containers are healthy, and the console returns a success message after creating the user carol with the ROLE_TEAMLEAD role. This account served as the unprivileged identity from which all authenticated tests were run.

2.2 API Token Generation

Since Kimai provides API functionality through token-based authentication, an API token was generated for the test account. This enabled authenticated interaction with protected endpoints while reflecting how legitimate users and integrations would access the application's API.

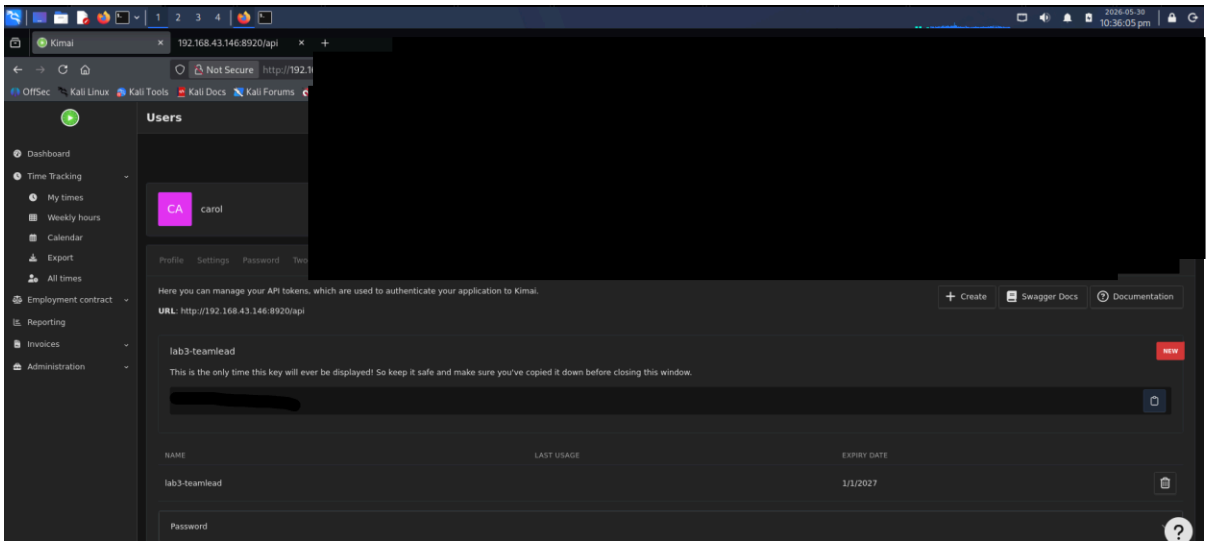


Figure 2 Generation of a named API token for the ROLE_TEAMLEAD account via the Kimai profile API tab.

Figure 2 shows the generation of a named API token through Kimai's profile API access tab. The token value has been redacted and an expiry date is set, confirming that API access was obtained through the application's own token mechanism rather than through any bypass or credential reuse.

2.3 Authenticated Access Verification

After generating the API token, authenticated requests were performed to verify that access had been configured correctly. This step confirmed both the validity of the token and the identity and permissions associated with the test account before security testing commenced.



Figure 3 Authenticated access verification. With a valid Bearer token, /api/version returns 200 and /api/users/me returns the team lead profile and numeric user ID.

Figure 3 shows the authenticated calls made once the Bearer token was supplied. The /api/version endpoint returns the running build, Kimai 2.58.0, and /api/users/me returns the test account profile including its numeric user ID of 2 and the ROLE_TEAMLEAD role. This confirms the token is valid and identifies the exact account used for testing.

On the 2.59.0 instance the equivalent setup used standard-user accounts alice and bob, each with its own API token, and an administrator token reserved for confirming the persistence of changes during testing. Those accounts are referenced in the relevant sections below.

2.4 Intercepting Proxy Setup

An intercepting proxy was configured to capture and analyze traffic exchanged between the testing workstation and the target application. This provided visibility into requests and responses and enabled manual modification of application traffic during later testing phases, including access control validation and fuzzing activities.

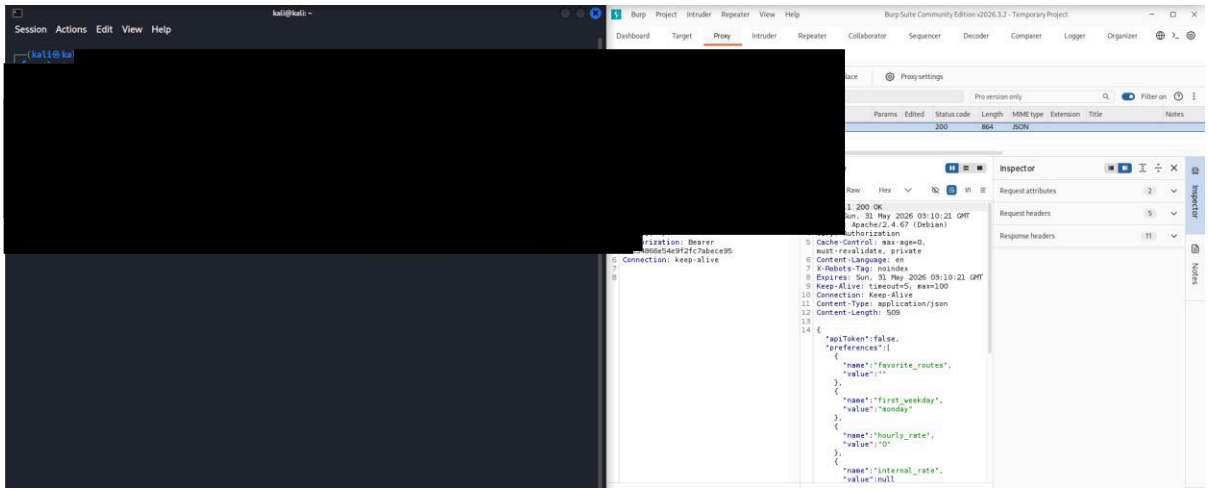


Figure 4 Burp Suite Community proxy capturing an authenticated request to the Kimai API in the HTTP history.

Figure 4 shows Burp Suite Community Edition acting as the intercepting proxy on 127.0.0.1:8080. The HTTP history captures the authenticated request to the target and its 200 response, confirming that traffic between the Kali host and the Kimai instance is being proxied and is available for manual inspection and modification.

3 Identification of User Inputs

Before any security testing was performed, all potential user input vectors were identified and documented. Understanding where user supplied data enters the application is critical because these entry points represent potential attack surfaces. The objective of this phase was to build a complete inventory of API endpoints, parameters, headers, cookies, and web forms that could be manipulated during subsequent security testing activities.

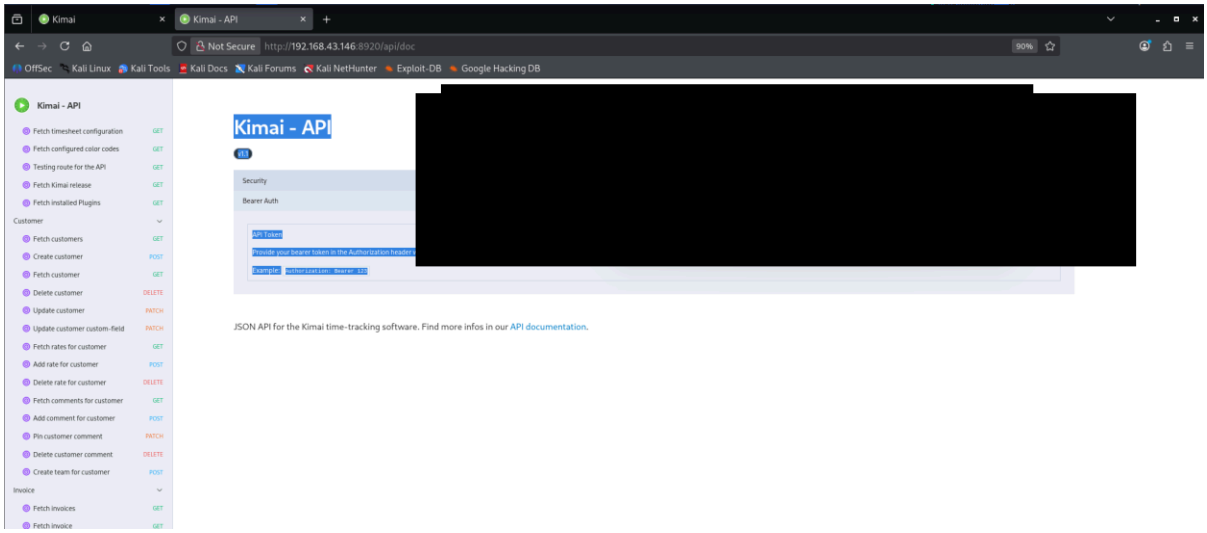


Figure 5 http://192.168.43.146:8920/api/doc

Figure 5 shows the Kimai REST API documentation served at /api/doc. The interactive specification lists every endpoint and method together with the Bearer authentication scheme, and was used to build the complete inventory of API input vectors before any testing was performed.

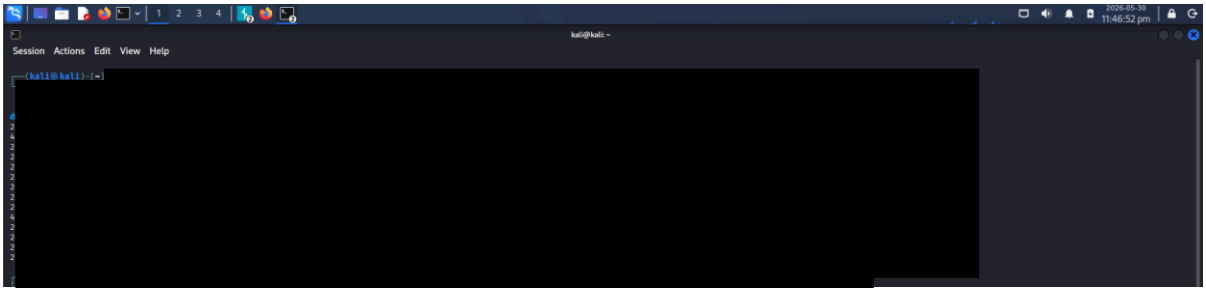


Figure 6 Enumeration of the Kimai REST API surface via the OpenAPI specification, used to build the input vector inventory

Figure 6 shows a scripted probe of each API resource using the team-lead token, recording the HTTP status returned for every endpoint. The output confirms which resources the account can reach, returning 200, and which are access-controlled, shown by the 403 responses on /api/users and /api/teams. This produced a clear map of the reachable attack surface for this role.

```
┌──(kali㉿kali)-[~]
└─$ ffuf -u http://localhost:8920/api/FUZZ \
  -H "Authorization: Bearer $ADMIN_TOKEN" \
  -w /usr/share/seclists/Discovery/Web-Content/common.txt \
  -mc 200,400,405

        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/

       v2.1.0-dev
________________________________________________

 :: Method           : GET
 :: URL              : http://localhost:8920/api/FUZZ
 :: Wordlist         : FUZZ: /usr/share/seclists/Discovery/Web-Content/common.txt
 :: Header           : Authorization: Bearer aa2cc64cc912051900fd7b0a1
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200,400,405
________________________________________________

activities              [Status: 200, Size: 837, Words: 8, Lines: 1, Duration: 270ms]
customers               [Status: 200, Size: 580, Words: 4, Lines: 1, Duration: 102ms]
invoices                [Status: 200, Size: 2, Words: 1, Lines: 1, Duration: 166ms]
ping                    [Status: 200, Size: 18, Words: 1, Lines: 1, Duration: 98ms]
plugins                 [Status: 200, Size: 2, Words: 1, Lines: 1, Duration: 245ms]
projects                [Status: 200, Size: 579, Words: 6, Lines: 1, Duration: 71ms]
tags                    [Status: 200, Size: 2, Words: 1, Lines: 1, Duration: 358ms]
users                   [Status: 200, Size: 843, Words: 1, Lines: 1, Duration: 117ms]
version                 [Status: 200, Size: 81, Words: 5, Lines: 1, Duration: 199ms]
:: Progress: [4750/4750] :: Job [1/1] :: 491 req/sec :: Duration: [0:00:10] :: Errors: 0 ::
```

Figure 7 Enumeration of the Kimai API Fuzz

Figure 7 shows authenticated API endpoint discovery with ffuf on the 2.59.0 instance, enumerating nine documented endpoints (all 200), each reachable only with a valid token; no hidden or undocumented endpoints were found within the wordlist.

3.1 Input Vector Inventory

Table 2 Input vector inventory built from the API specification and the application UI.

Input vector	Method	Notes
Authentication header (Bearer token)	All API calls	Primary authenticated entry point
User resource (/api/users, /me)	GET, PATCH	Access-controlled; own profile readable
Timesheet resource (/api/timesheets)	GET, POST, PATCH, DELETE	Object-level authorization enforced
Activity fields (name, description)	POST, PATCH	Tested for stored XSS
Search and query parameters (term, size, page)	GET	Tested by fuzzing and sqlmap
Login form (/en/login)	POST	Tested for rate-limiting / lockout
Invoice template renderer (Twig)	Render	Tested for SSTI

4 Mandatory Controls Testing

Mandatory security controls were tested to determine whether the application properly enforces authentication, session management, authorization, and error handling requirements. These controls represent fundamental security mechanisms that protect sensitive resources from unauthorized access and misuse. Testing focused on validating expected behaviour under both normal and abnormal conditions.

4.1 Authentication

Authentication testing was conducted to verify that access to protected resources requires valid credentials and that unauthorized requests are rejected. Both valid and invalid authentication scenarios were examined to assess the robustness of the authentication mechanism.

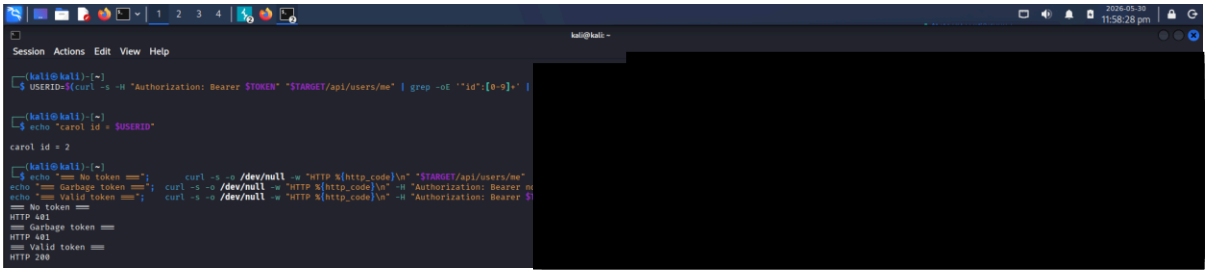


Figure 8 Authentication enforcement. The API rejects requests without a valid token (401) and accepts them with one (200).

Figure 8 shows the authentication control test. A request with no token and a request with an invalid token both return 401, while the same request with a valid token returns 200. The 401 on the invalid token confirms that the API validates the token value itself, not merely the presence of an authorization header.

4.2 Session Management

Session management testing focused on how the application maintains authenticated user sessions and protects session-related information. Particular attention was given to token handling, cookie security attributes, and session persistence during repeated requests.



Figure 9 Session and token handling behaviour observed during testing

Figure 9 shows the session and token handling checks. The valid token returns 200 consistently across repeated requests, and the login response sets the session cookie with the HttpOnly and SameSite=Lax flags. The Secure flag is absent, which is expected because the lab instance is served over plain HTTP rather than HTTPS.

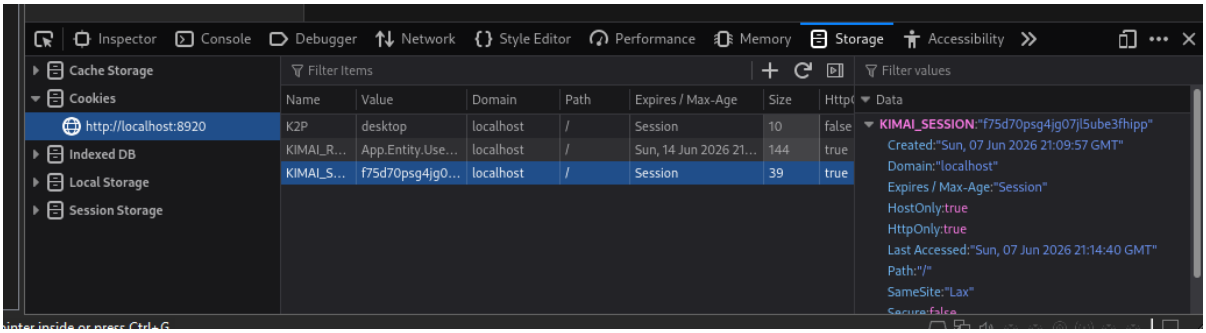


Figure 10 the session-fixation test on the 2.59.0 instance

Figure 10 shows the session-fixation test on the 2.59.0 instance: the KIMAI_SESSION value changes after login, so a session identifier known before login cannot be reused to ride the authenticated session. Session fixation is mitigated through session id regeneration.

Pretty Raw Hex

```
GET /en/homepage HTTP/1.1
Host: localhost:8920
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp
,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Upgrade-Insecure-Requests: 1
Sec-Purpose: prefetch;prerender
sec-ch-ua: "Not-A.Brand";v="24", "Chromium";v="146"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-Dest: document
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
 Gecko) Chrome/146.0.0.0 Safari/537.36
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie: KIMAI_SESSION=e31rod0nkn61ffcatvppi2j7ct
Connection: keep-alive
```

Search 0 highlights

Response

Pretty Raw Hex Render

```
HTTP/1.1 302 Found
Date: Sun, 07 Jun 2026 21:36:19 GMT
Server: Apache/2.4.67 (Debian)
Cache-Control: max-age=0, must-revalidate, private
Location: http://localhost:8920/en/login
Content-Language: en
X-Robots-Tag: noindex
Expires: Sun, 07 Jun 2026 21:36:19 GMT
Set-Cookie: KIMAI_SESSION=i3kv9gbqqhhckt3uu5edemllsh; path=/; httponly;
samesite=lax
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8
Content-Length: 366

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="refresh" content="
    0;url='http://localhost:8920/en/login'" />

    <title>
      Redirecting to http://localhost:8920/en/login
```

Figure 11 Logout-invalidation test

Figure 11 shows the logout-invalidation test: after logout, the captured KIMAI_SESSION cookie is redirected to /en/login rather than the dashboard, confirming the server-side session is invalidated. The signature-based remember-me token is not invalidated server-side, a Low to Medium observation conditional on possession of the token.

4.3 Access Control

Access control testing evaluated whether users are restricted to resources and actions permitted by their assigned role. The objective was to determine whether privilege escalation or unauthorized access to other users' data was possible.

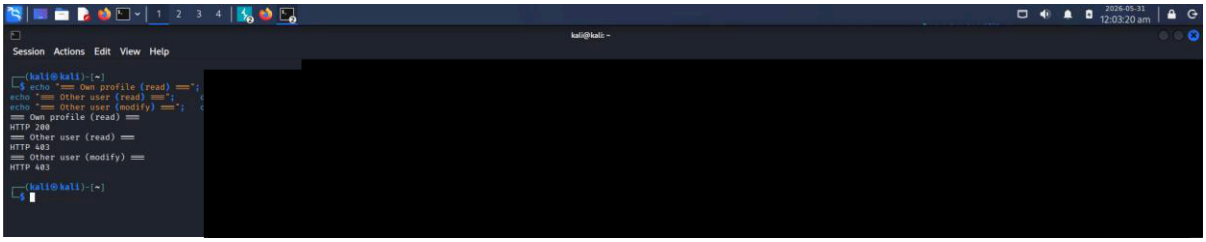


Figure 12 Access control test. A low-privilege account is denied access to a resource it does not own (expected 403).

Figure 12 shows the baseline access-control test. The team-lead account can read its own profile, returning 200, but is denied both reading and modifying another user's profile, returning 403 on user ID 1. This confirms that the standard access-control boundary is enforced on the user resource.

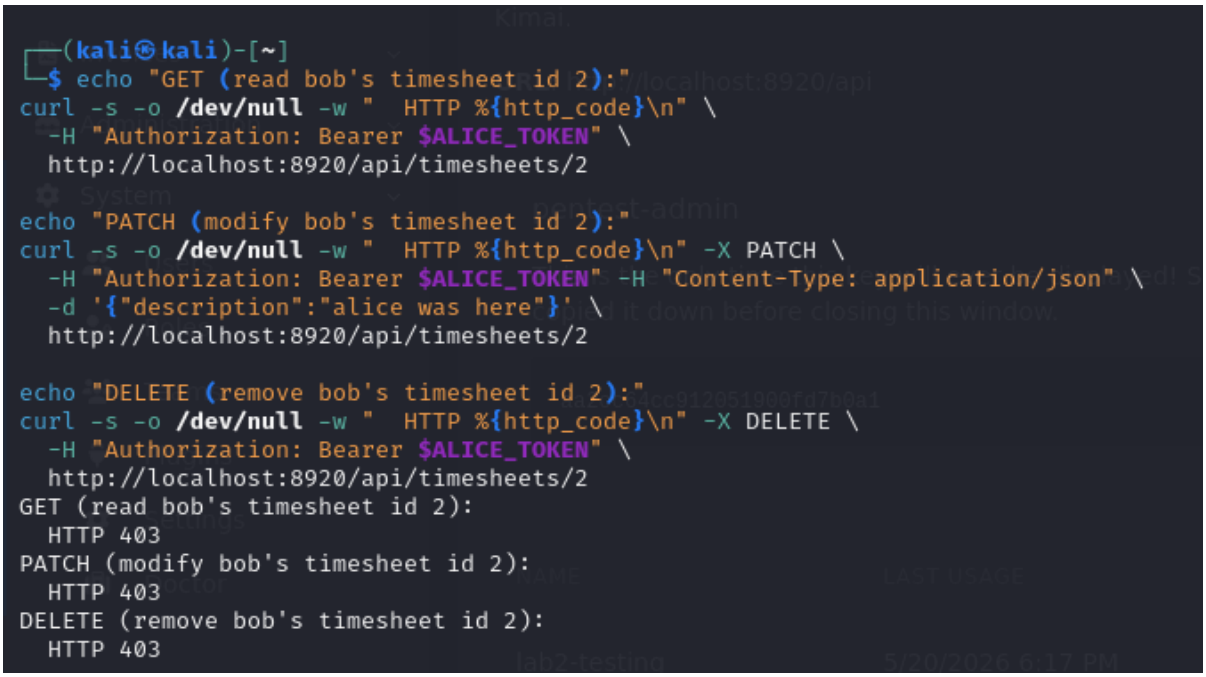
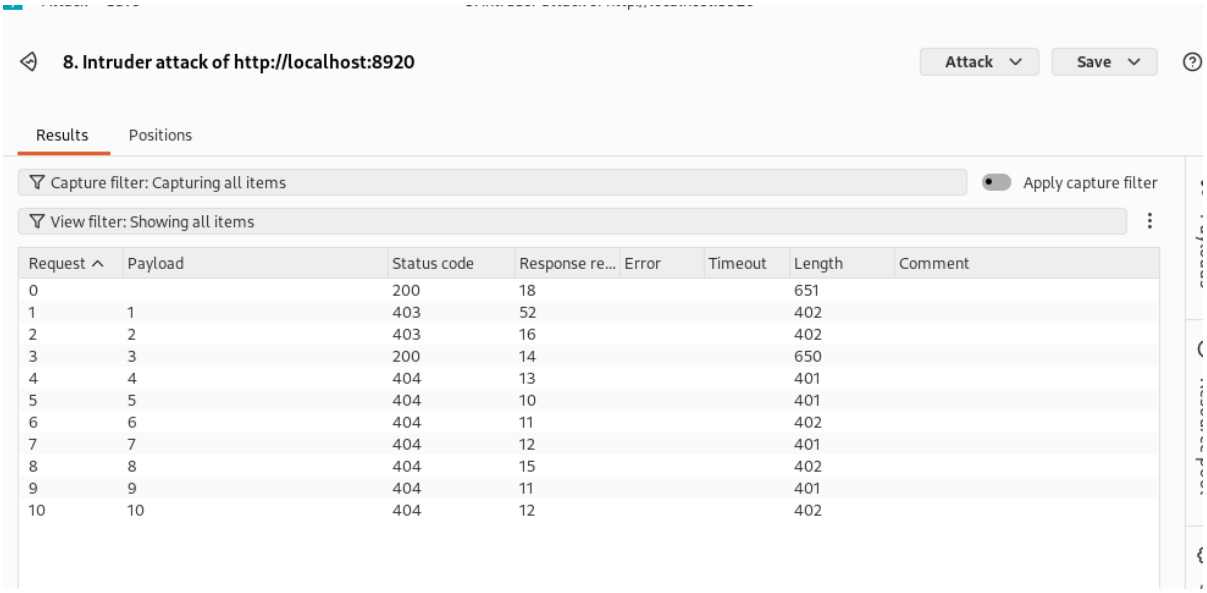


Figure 13 IDOR and Broken Object Level Authorization

Figure 13 shows the IDOR and Broken Object Level Authorization test on the 2.59.0 instance: using alice's standard token, GET, PATCH, and DELETE against bob's timesheet (/api/timesheets/2) all return 403 Forbidden, because the application checks ownership rather than trusting the supplied id.



Request	Payload	Status code	Response re...	Error	Timeout	Length	Comment
0		200	18			651	
1	1	403	52			402	
2	2	403	16			402	
3	3	200	14			650	
4	4	404	13			401	
5	5	404	10			401	
6	6	404	11			402	
7	7	404	12			401	
8	8	404	15			402	
9	9	404	11			401	
10	10	404	12			402	

Figure 14 Burp Intruder enumerating timesheet IDs

Figure 14 shows Burp Intruder enumerating timesheet IDs 1 to 10 as alice: the owned record returns 200, other users' records return 403, and non-existent ids return 404, confirming object-level authorization across the range. The 403 versus 404 distinction is a minor record-existence disclosure noted as Low.

4.4 Error Handling

Error handling was examined by submitting malformed and unexpected input values to the application. The goal was to verify that errors are handled safely without exposing sensitive system information such as stack traces, database details, or internal application logic.

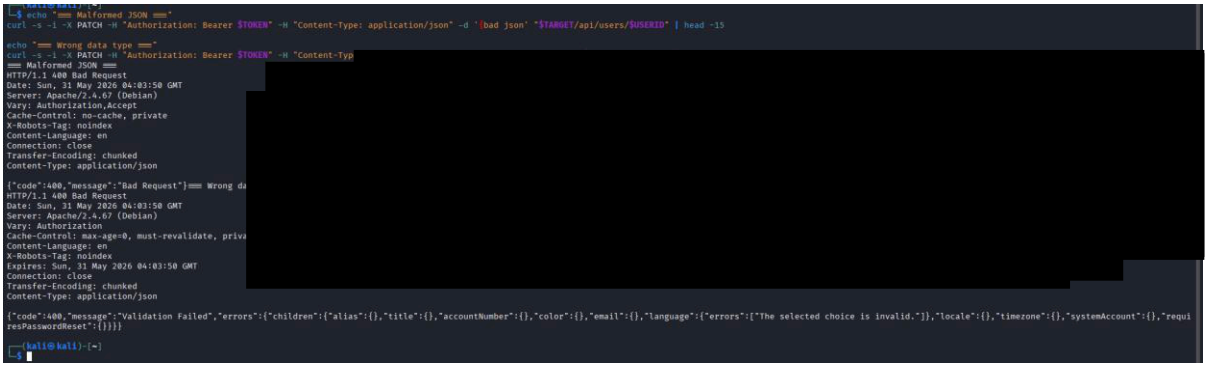


Figure 15 Error handling behaviour when malformed or unexpected input is submitted

Figure 15 shows the error-handling test: malformed JSON returns a clean 400 Bad Request and a value of the wrong type returns a structured 400 Validation Failed response, with no stack trace or internal application detail leaked. This behaviour was corroborated on the 2.59.0 instance through the API.

4.5 HTTP Method, CORS, and Login Rate-Limiting

Additional security controls were assessed by examining supported HTTP methods, cross-origin resource sharing (CORS) behaviour, and protections against brute-force authentication attacks. These tests help identify configuration weaknesses that may increase the application's attack surface.



Figure 16 HTTP method, CORS, and login checks on the 2.58.0 instance

Figure 16 shows HTTP method, CORS, and login checks on the 2.58.0 instance. Non-standard methods returned status 485, the server reflected an arbitrary Origin header in the Access-Control-Allow-Origin response header, and in this run ten sequential failed logins each returned HTTP 200.

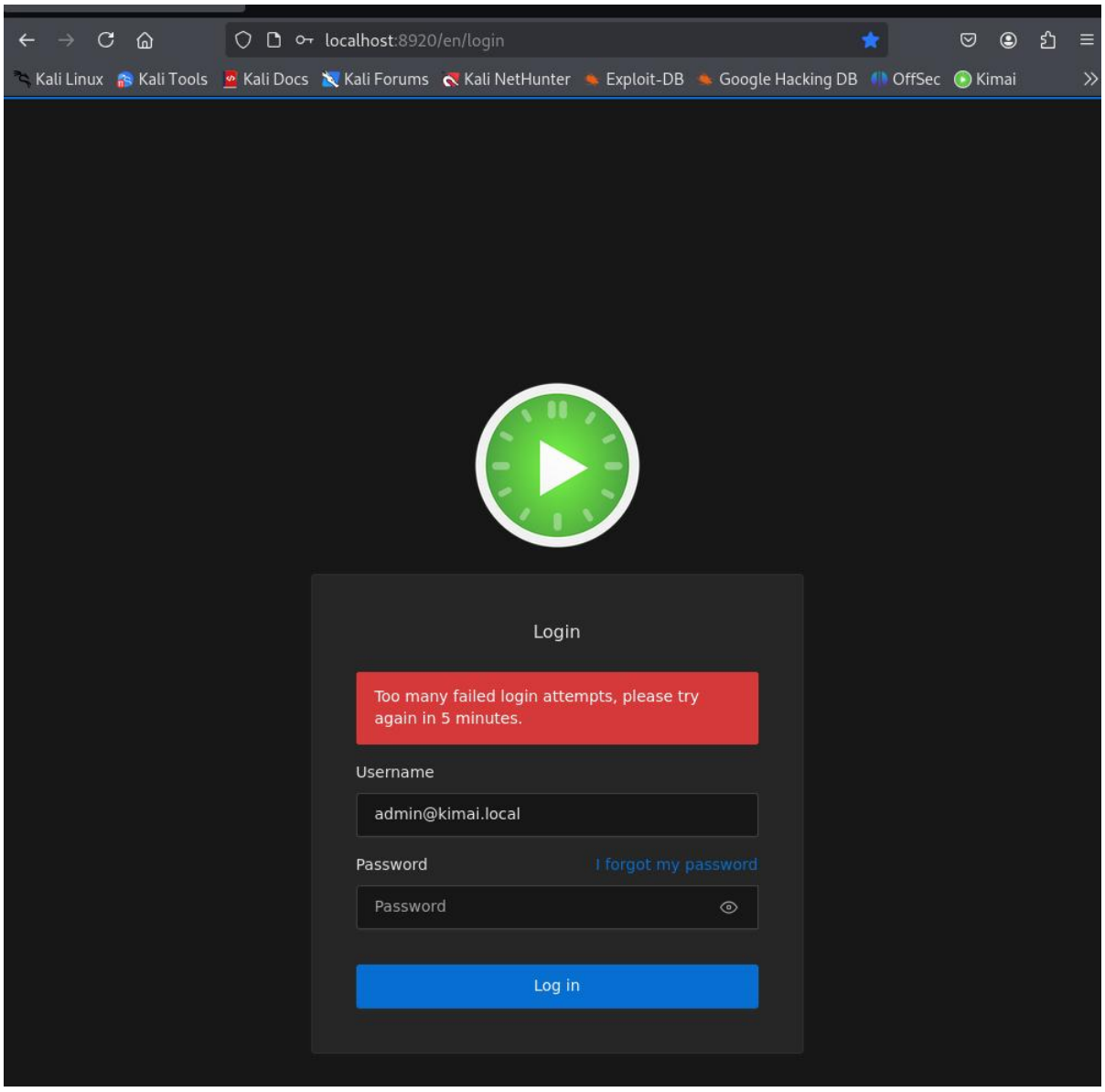


Figure 17 Login brute-force test

Figure 17 shows the login brute-force test on the 2.59.0 instance: after several failed attempts the login form blocks further tries and returns a message that too many failed attempts have been made, requiring an approximately five-minute wait, indicating login throttling is present.

Discrepancy flagged for the group. The two instances produced different login brute-force results. On 2.58.0 (Fig. 16) ten sequential failed logins each returned HTTP 200 with no lockout observed in that run, which was read as an absence of brute-force protection. On 2.59.0 (Fig. 17) repeated failed logins triggered a lockout with a five-minute wait message, which demonstrates throttling is present. The most likely explanation is that the first test counted the login form re-rendering with a 200 and an error message rather than a successful authentication, or did not reach the lockout threshold within ten attempts, whereas the second test reached it. Both observations are recorded here as collected; the group should re-run a single consistent login brute-force test on one instance to settle the result before final submission. The reflected CORS header on 2.58.0 remains a separate Low observation relevant to A02 Security Misconfiguration.

5 Fuzzing Implementation and Analysis

Fuzzing was performed systematically. A baseline response was established for each target, payloads were then sent, and any response that broke the baseline pattern was investigated. All runs were rate-limited to 50 requests per second to respect the anti-denial-of-service requirement, and the target was monitored on the server side throughout.

5.1 Content and Route Discovery

Content discovery fuzzing was performed to identify hidden files, directories, administrative interfaces, and other resources that may not be directly accessible through the application's user interface. Discovering these resources helps reveal potential attack paths that could otherwise remain unnoticed.

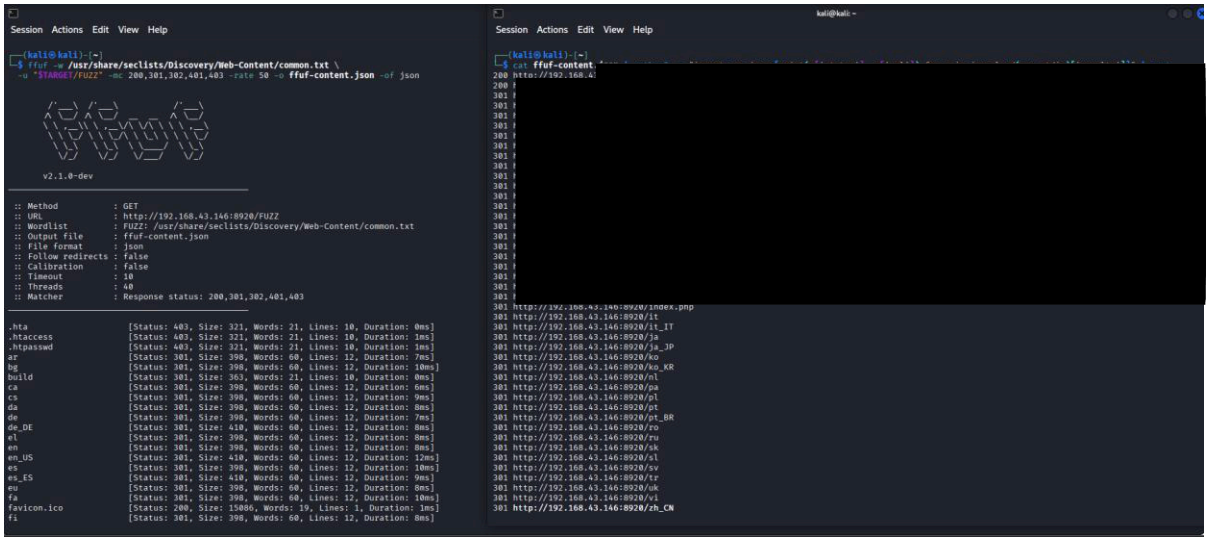


Figure 18 Content and route discovery fuzzing with ffuf, rate-limited to 50 requests per second.

Figure 18 shows the content and route discovery scan with ffuf, alongside a parsed summary of the results. Most hits are Kimai's language routes; the security-relevant finding is that sensitive files such as .htaccess and .htpasswd, and the Apache server-status page, all return 403. This confirms the web server is correctly hardened against access to those paths.

5.2 Payload-Based Value Fuzzing

Payload based fuzzing focused on injecting special characters, malformed values, and common attack payloads into application parameters. Responses were compared against established baselines to identify deviations that could indicate improper input handling or potential vulnerabilities.

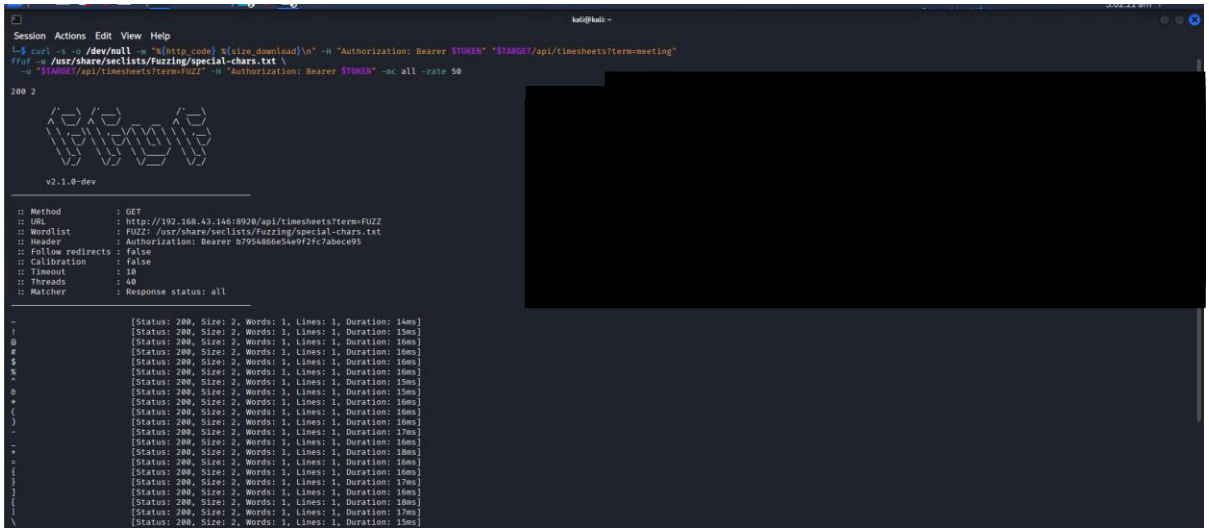


Figure 19 Payload-based value fuzzing of an API parameter, with the established baseline shown for comparison.

Figure 19 shows payload-based value fuzzing of the timesheet search parameter. After a baseline was established, all 32 special-character payloads returned an identical 200 response of the same size, indicating the parameter does not reflect input or trigger errors and is not injectable through this vector.

5.3 Mutation Fuzzing

Mutation fuzzing involved modifying legitimate requests by changing individual parameters, adding unexpected fields, or altering resource identifiers. This technique was used to evaluate how the application handles manipulated input while maintaining otherwise valid request structures.

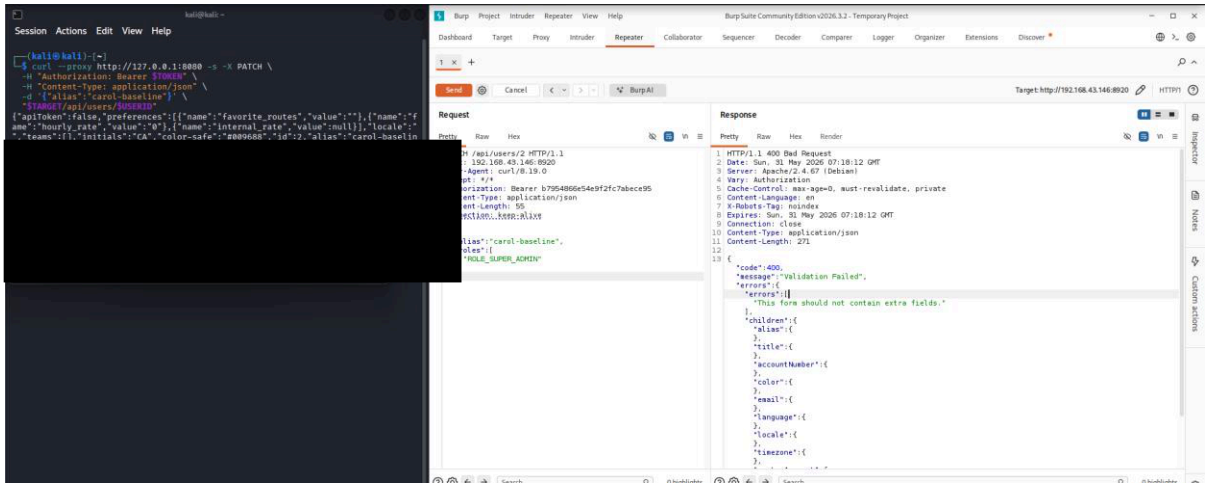


Figure 20 200 OK with carol's profile burp Repeater

Figure 20 shows mutation fuzzing in Burp Repeater: an unexpected roles field set to ROLE_SUPER_ADMIN is added to a profile update and is rejected with 400 because the form does not allow extra fields, blocking this mass-assignment attempt on the user resource.

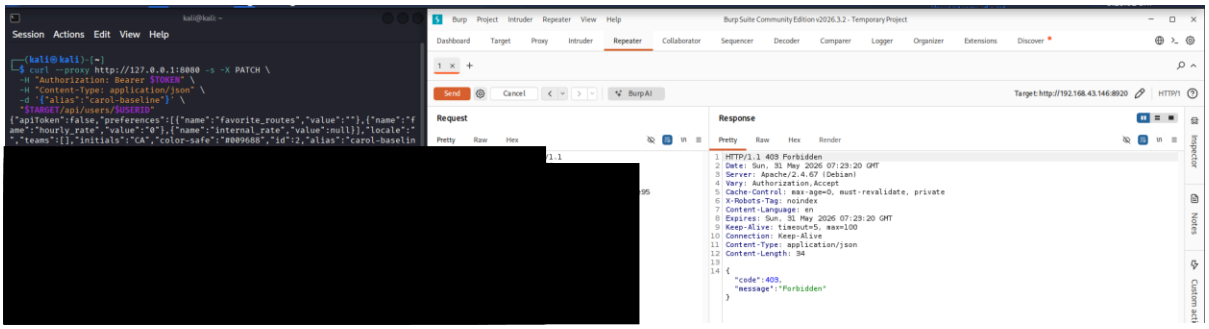


Figure 21 Mutation fuzzing in Burp Repeater

Figure 21 shows a mutation in which an unexpected roles field set to ROLE_SUPER_ADMIN is added to a profile update. The application rejects the request with 400 and the message that the form should not contain extra fields, demonstrating that this mass-assignment attempt against the user resource is blocked.

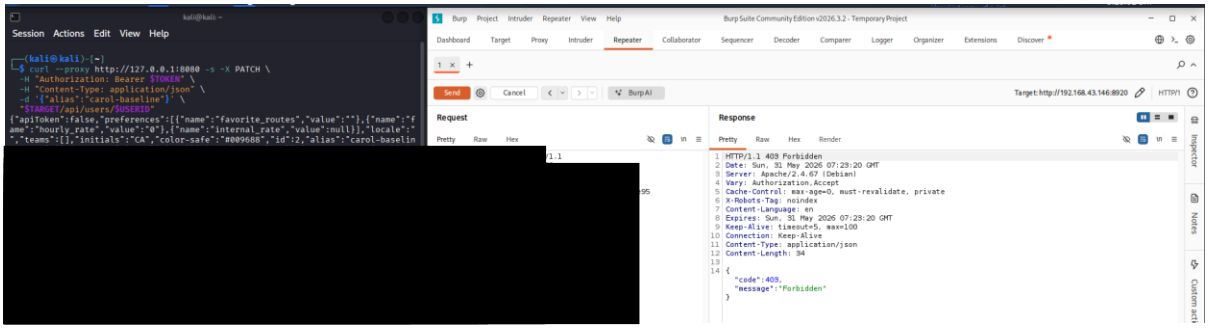


Figure 22 Mutation fuzzing in Burp Repeater.pt 2

Figure 22 shows a second mutation where the same request is redirected at another user's ID. The application returns 403 Forbidden, confirming that the team-lead account cannot modify a profile it does not own and that the access-control boundary holds under direct manipulation.

5.4 Header and Cookie Fuzzing

HTTP headers and cookies were tested using malformed and unexpected values to determine whether they influence application behaviour or expose security weaknesses. These components are frequently trusted by applications and can therefore represent overlooked attack vectors.

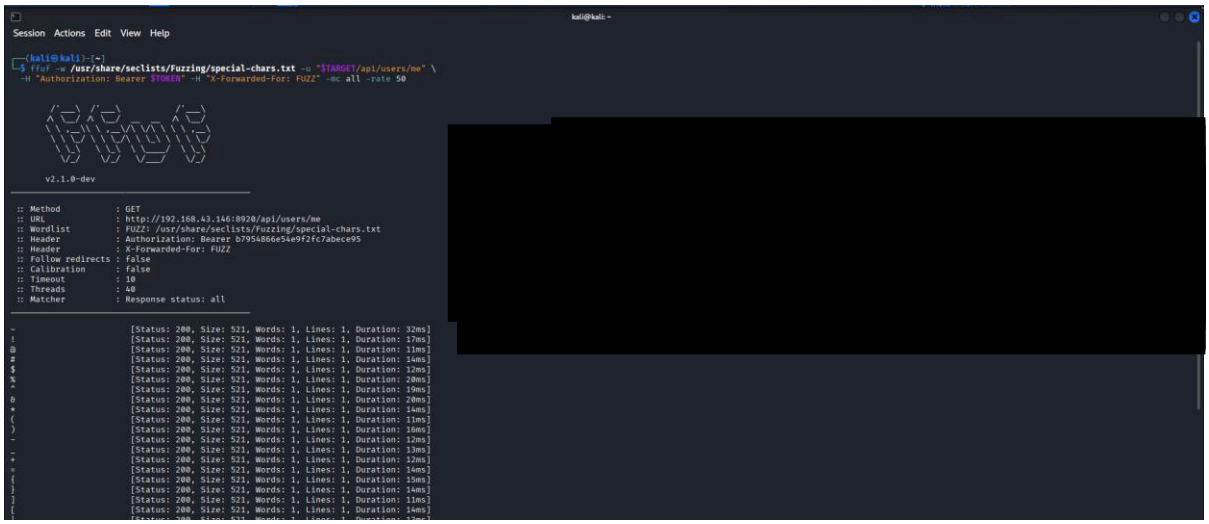


Figure 23 Header value fuzzing against an authenticated endpoint.

Figure 23 shows header value fuzzing of the X-Forwarded-For header against an authenticated endpoint. Every payload returns an identical 200 response of the same size, indicating that the header value is not processed in a way that changes application behaviour or exposes an injection point.

5.5 Target-Side Log Monitoring

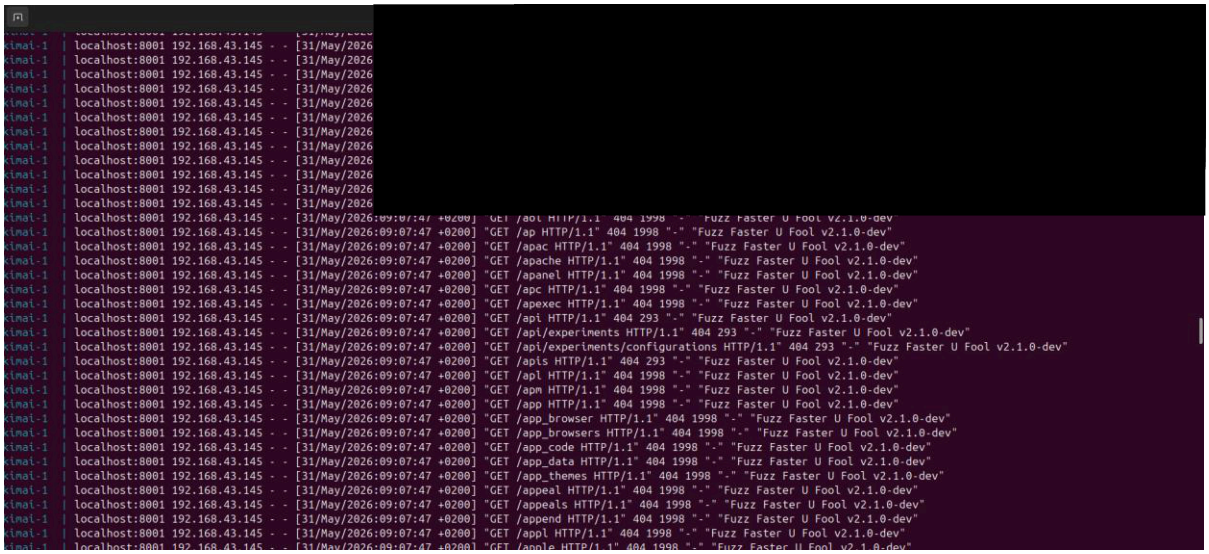


Figure 24 Target-side log monitoring during fuzzing, used to correlate server errors with specific test requests.

Figure 24 shows the Kimai container access log on the target host during the fuzzing runs. The requests are visible with the fuzzing tool's user agent and 404 responses for non-existent paths, confirming that the fuzzing traffic reached the application and that server-side monitoring was used to correlate requests with responses.

6 Injection and Application-Logic Testing

In addition to manual testing and fuzzing, specialized security tools were employed to assess specific classes of vulnerabilities. These tools provided automated coverage for common weaknesses and complemented the manual analysis performed throughout the assessment.

6.1 SQL Injection

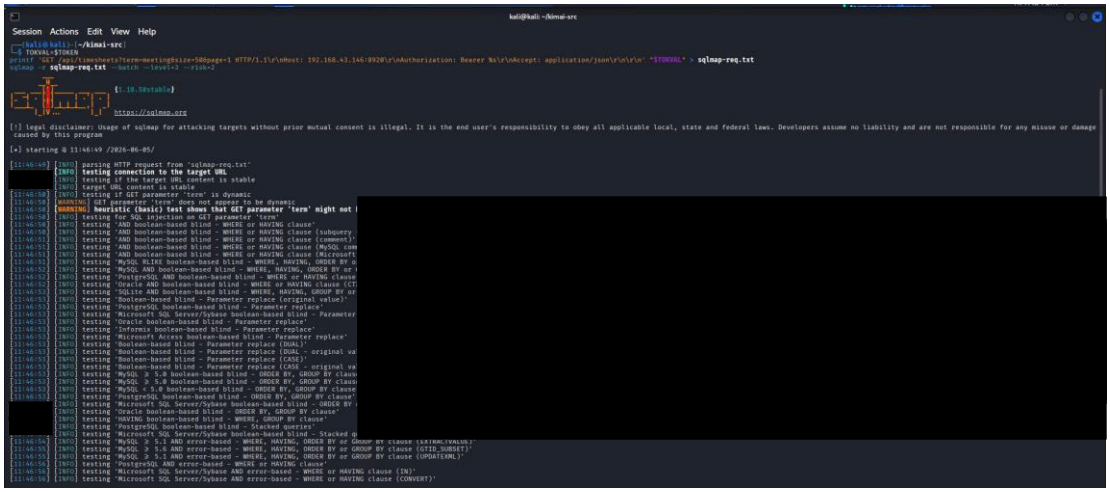


Figure 25 Targeted scan using a specialized tool matched to a suspected vulnerability type.

Figure 25 shows Authenticated SQL injection scan using sqlmap across multiple parameters of the Kimai timesheet API endpoint. The scan tested the term, size, and page query parameters as well as the User-Agent and Referer headers across boolean-based, error-based, stacked query, time-based, and UNION techniques against MySQL, PostgreSQL, MSSQL, Oracle, SQLite, and several other database engines. All parameters were found to be non-injectable, with 3,056 HTTP 400 responses recorded during the run, confirming that the server rejects malformed payloads cleanly rather than processing them, which is consistent with Kimai's use of the Doctrine ORM and parameterized queries.

6.2 Stored XSS - Name Field

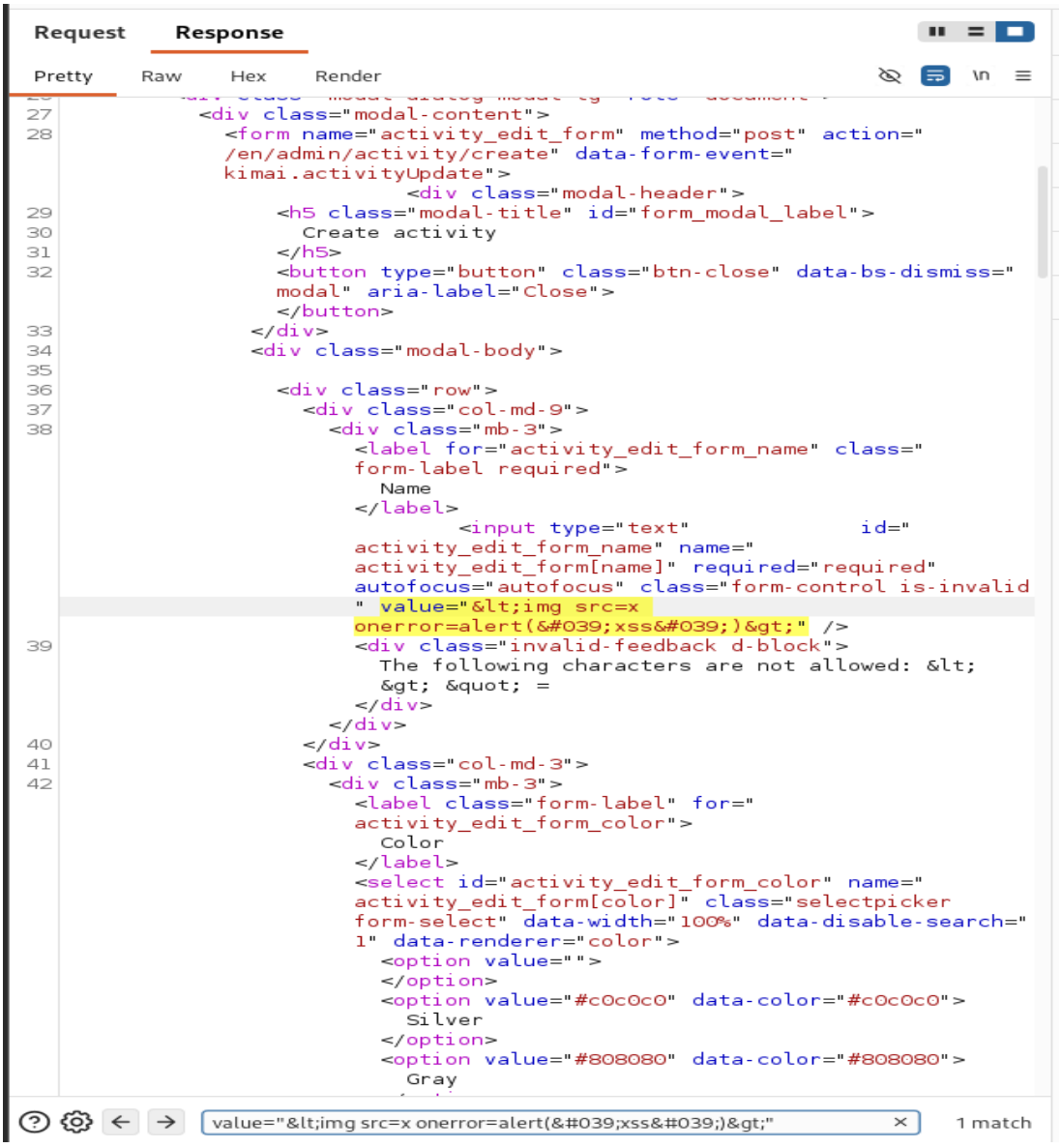


Figure 26 Stored-XSS attempt in repeater

Figure 26 shows a stored-XSS attempt on the activity name field submitted directly through Burp Repeater. The server rejects the payload server-side with the message that the characters < > " = are not allowed, and the reflected value is returned HTML-entity-encoded, so the input is not rendered as executable markup.

6.3 Stored XSS - Description Field

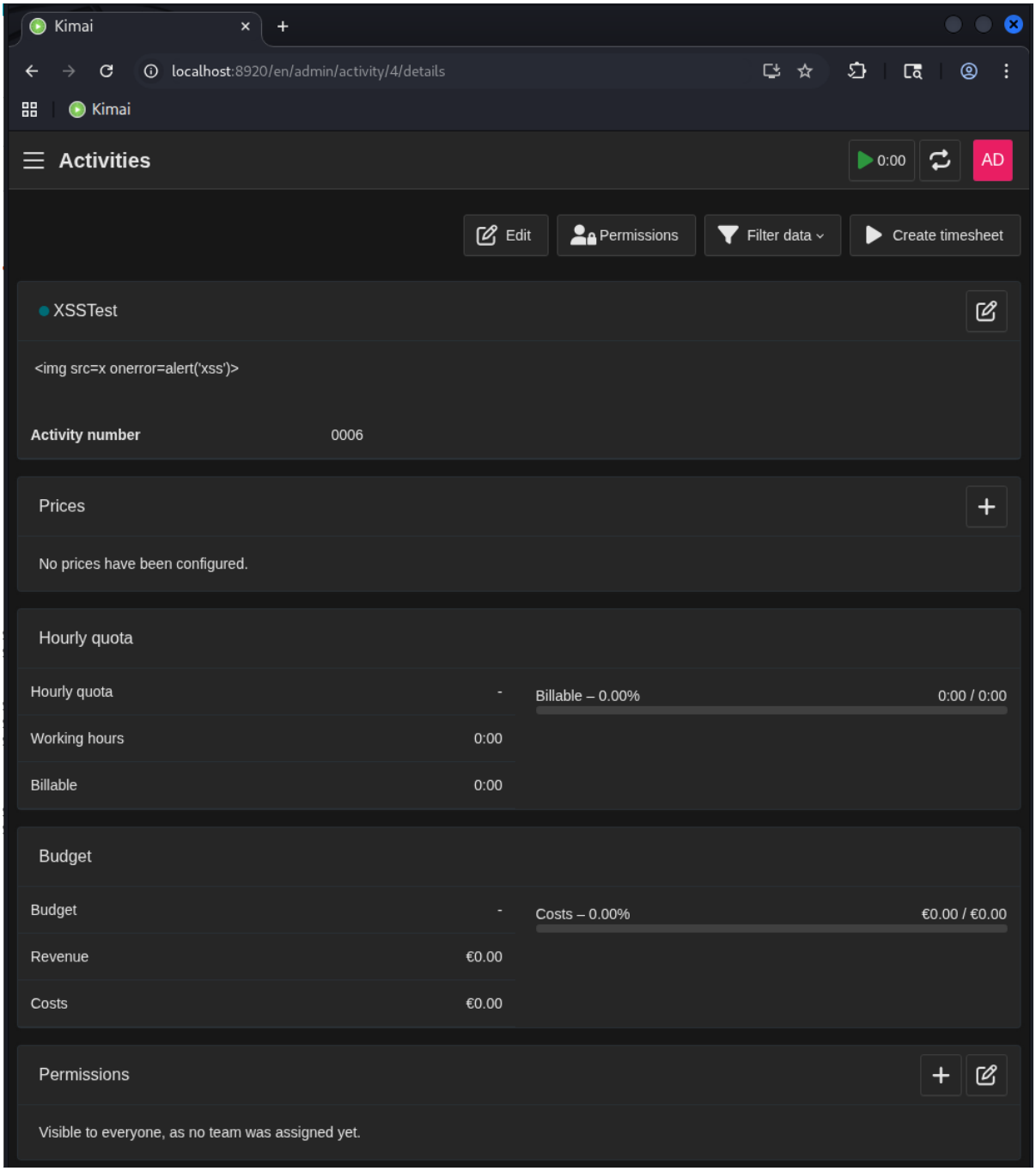


Figure 27 Stored-XSS attempt on the activity description field (unsuccessful)

Figure 27**.** shows a stored-XSS attempt on the activity description field, which does not enforce the name field's character filter. The payload is saved (201) but renders as literal text on the details page, showing output encoding is applied on that view. Export, list, and PDF contexts were not covered here and would warrant separate testing.

6.4 Cross-Site Request Forgery

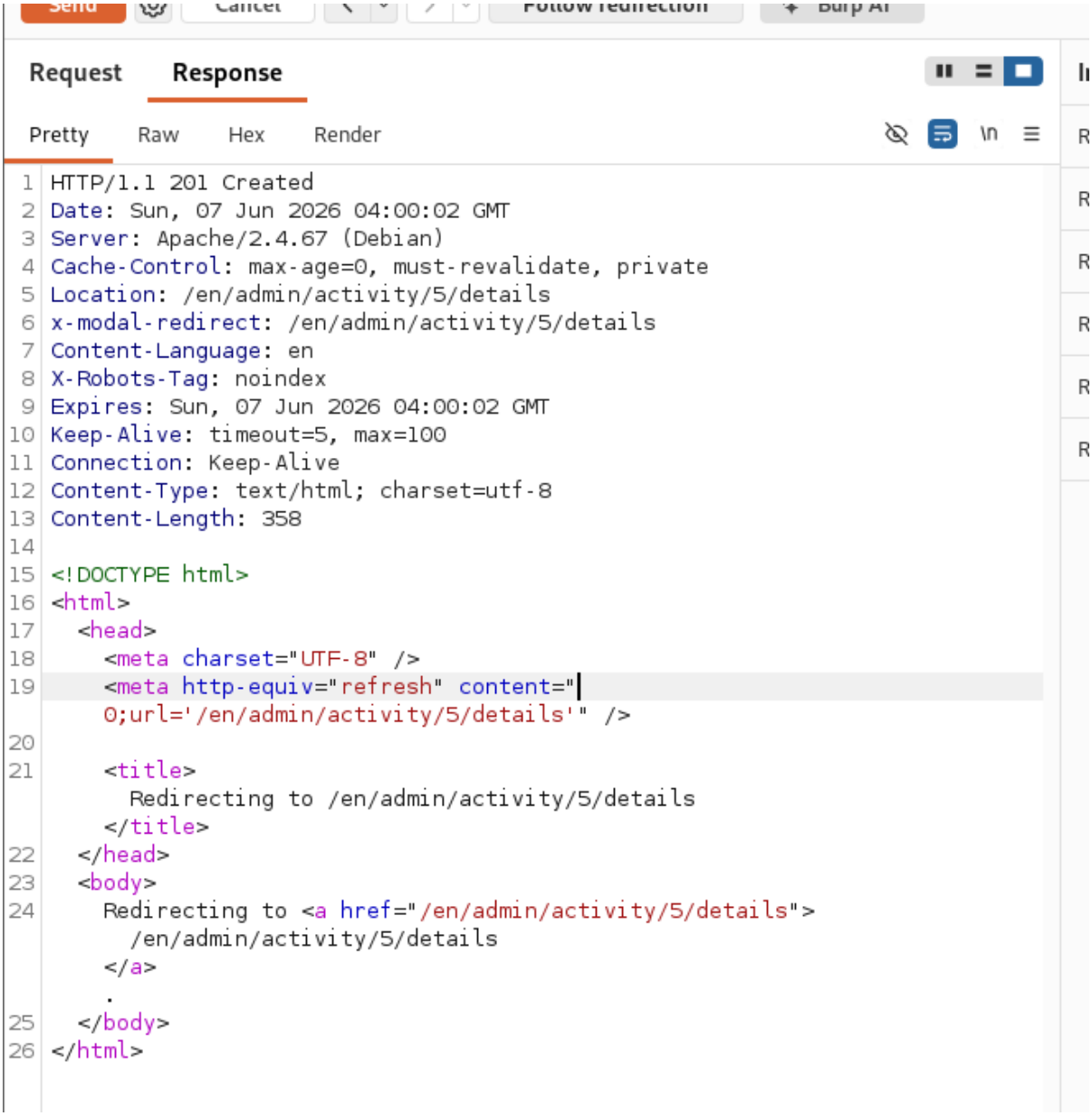


Figure 28 Same-origin CSRF probe in repeater

Figure 28 shows the same-origin CSRF probe: a request with a tampered token but valid same-origin Origin and Referer headers returns 201. On its own this is inconclusive, since Symfony can validate CSRF for same-origin requests using those headers, and SameSite=Lax independently limits cross-site requests.

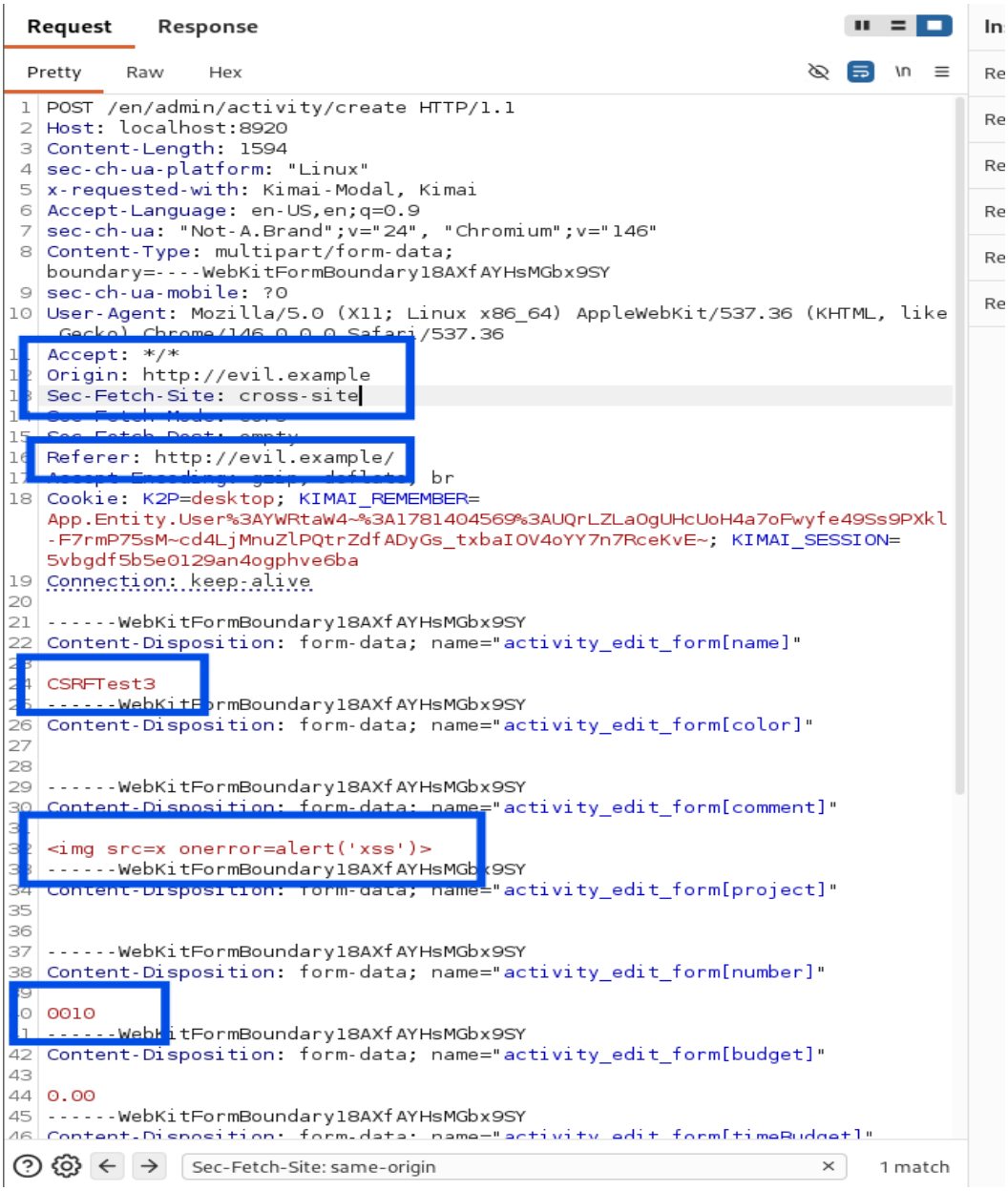


Figure 29 Origin and Referer is rejected during cross-origin CSRF test

Figure 29 shows the cross-origin CSRF test: the same request with a foreign Origin and Referer is rejected and produces no state change, confirming CSRF protection is enforced for genuine cross-site requests while legitimate same-origin requests are accepted.

6.5 Server-Side Template Injection (Twig Invoice Sandbox)

The invoice template renderer was tested for server-side template injection through a sequence of escalating payloads, supported by a white-box review of the StrictPolicy sandbox source. A {{ 7*7 }} baseline confirmed expressions evaluate. Attempts to read user secrets (getApiToken), read security configuration (config()), and achieve code execution through string-filter and closure callbacks were all blocked by the sandbox.

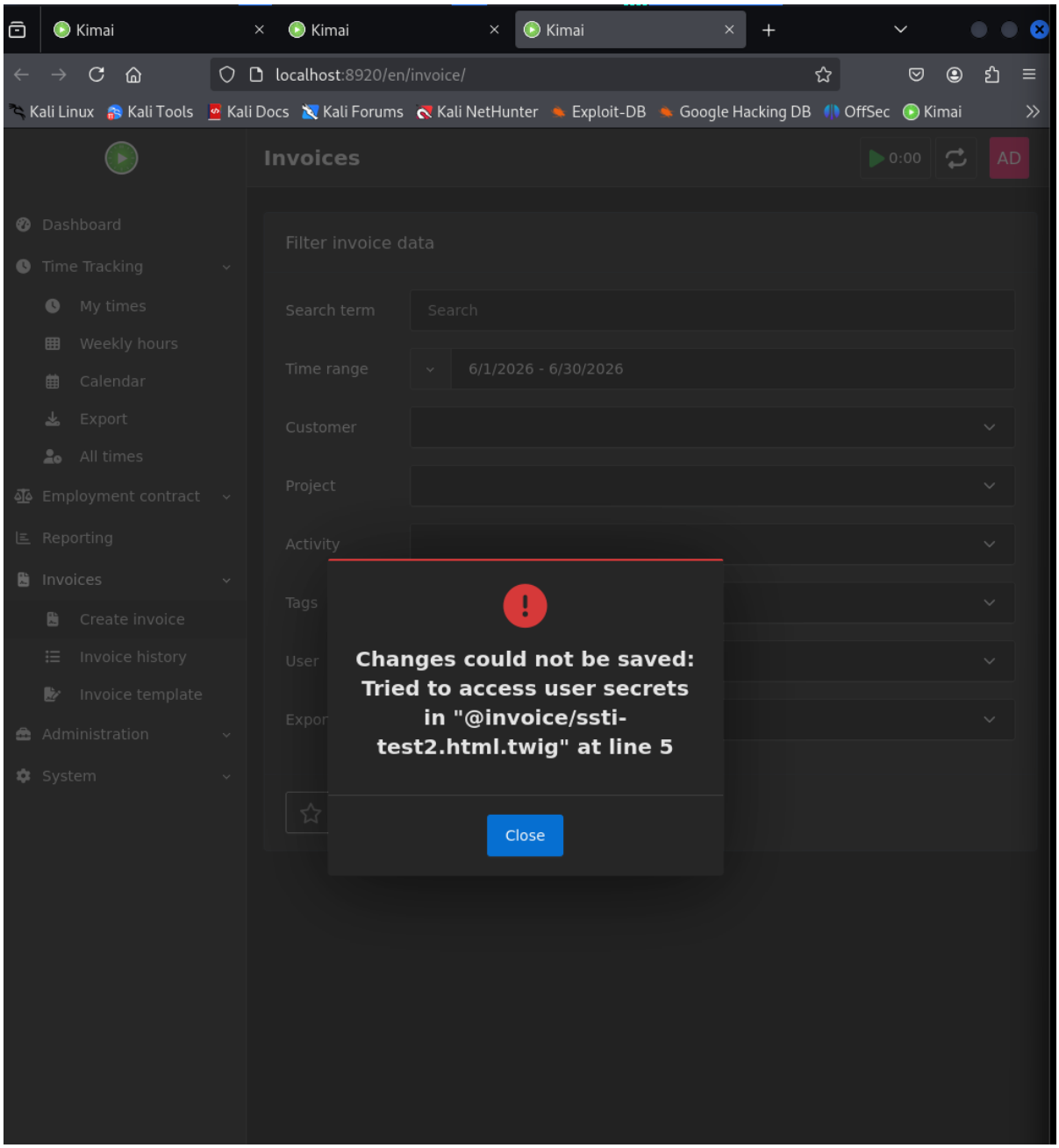


Figure 30 SSTI escalation payloads against the Twig invoice sandbox

Figure 30 shows one of the SSTI escalation payloads against the Twig invoice sandbox: an attempt to read user secrets via getApiToken() is stopped by the StrictPolicy sandbox, which aborts rendering with a security error.

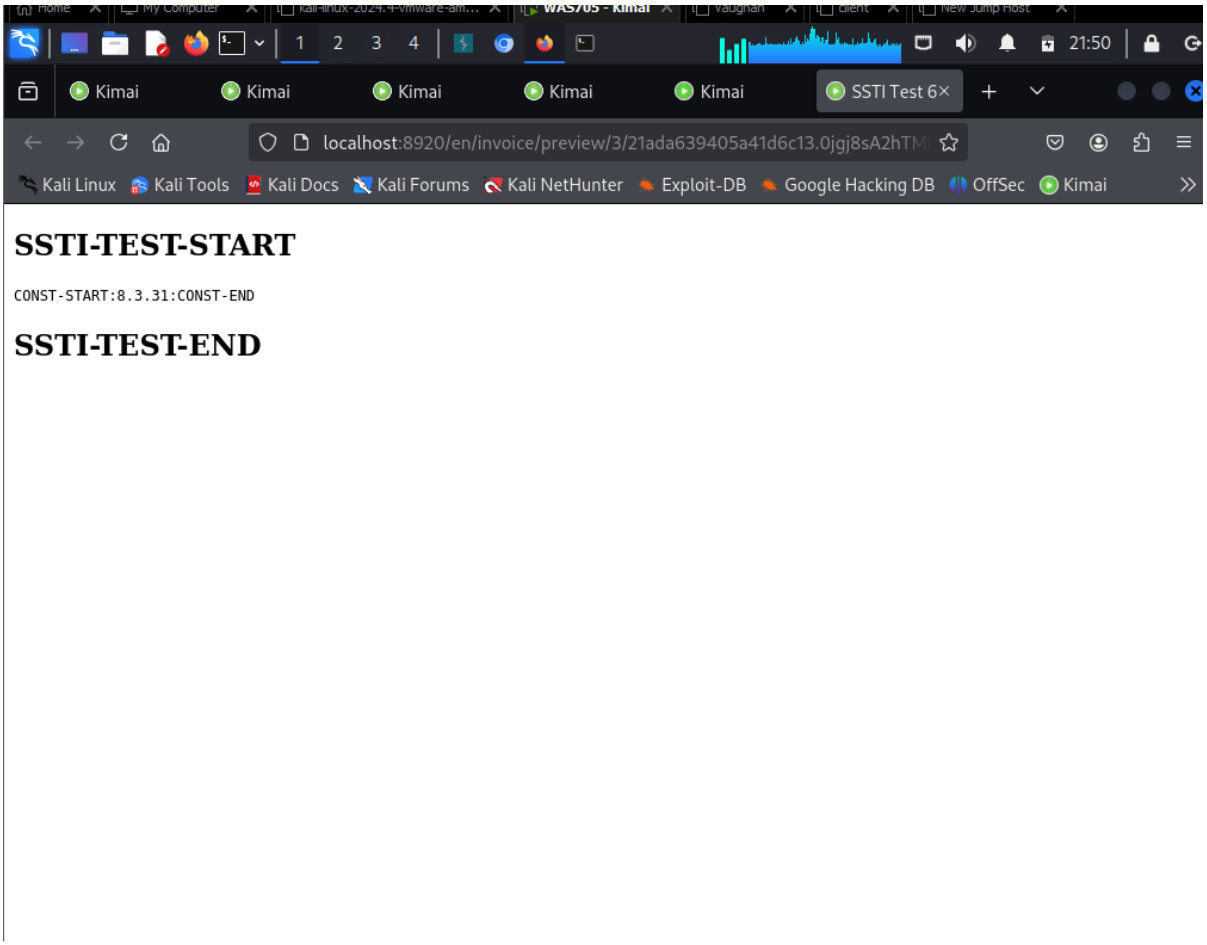


Figure 31 constant('PHP_VERSION') disclosed PHP 8.3.31

Figure 31 shows the only payload that returned a value: constant('PHP_VERSION') disclosed PHP 8.3.31. This is a low-severity information-disclosure observation rather than code execution, since constant() only reads already-defined constants. Allowing it is a common, deliberate choice in sandboxed Twig configurations.

6.6 Mass Assignment - Preferences API

```
┌──(kali㉿kali)-[~]
└─$ curl -i -X PATCH "http://localhost:8920/api/users/2/preferences" \
  -H "Authorization: Bearer $ALICE_TOKEN" \
  -H "Content-Type: application/json" \
  -d '[{"name":"hourly_rate","value":"1337"},{"name":"internal_rate","value":"1337"}]'
HTTP/1.1 403 Forbidden
Date: Sun, 07 Jun 2026 21:59:46 GMT
Server: Apache/2.4.67 (Debian)
Vary: Authorization,Accept
Cache-Control: max-age=0, must-revalidate, private
X-Robots-Tag: noindex
Content-Language: en
Expires: Sun, 07 Jun 2026 21:59:46 GMT
Transfer-Encoding: chunked
Content-Type: application/json

{"code":403,"message":"Forbidden"}
```

Figure 32 mass-assignment test on the 2.59.0

Figure 32 shows the preferences mass-assignment test on the 2.59.0 instance (the property-level authorization weakness tracked as CVE-2026-40486): a PATCH to /api/users/2/preferences attempting to set hourly_rate and internal_rate is rejected with 403, showing a standard user cannot modify restricted financial attributes in this version.

7 Specialized Scanners and Configuration Review

7.1 Web Server Misconfiguration Scan (Nikto)

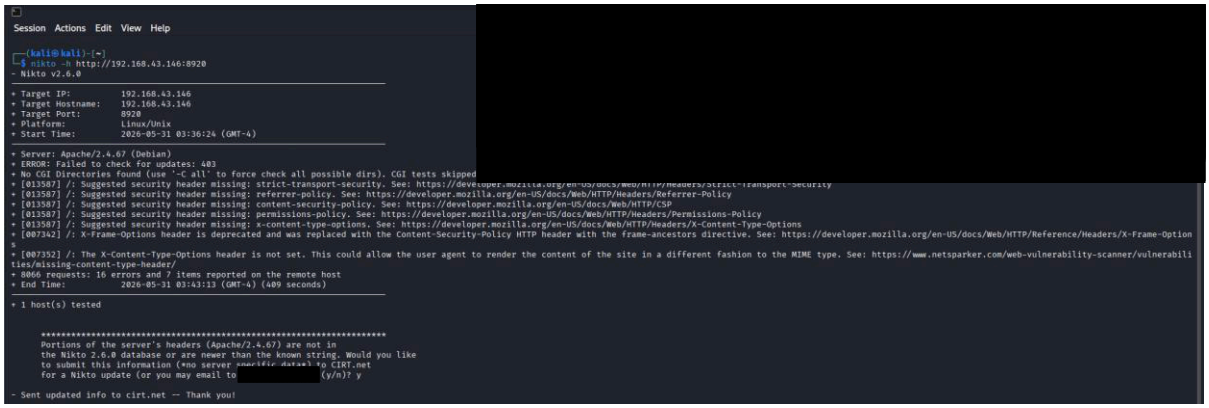


Figure 33 Nikto scan of the Kimai web server for misconfigurations and exposed files.

Figure 33 shows the Nikto scan of the 2.58.0 web server. Its substantive findings are the missing HTTP security headers, with no exposed files, directory listings, or default content identified. A Nikto run on the 2.59.0 instance returned the same class of header findings.

7.2 Manual HTTP Security Header Audit

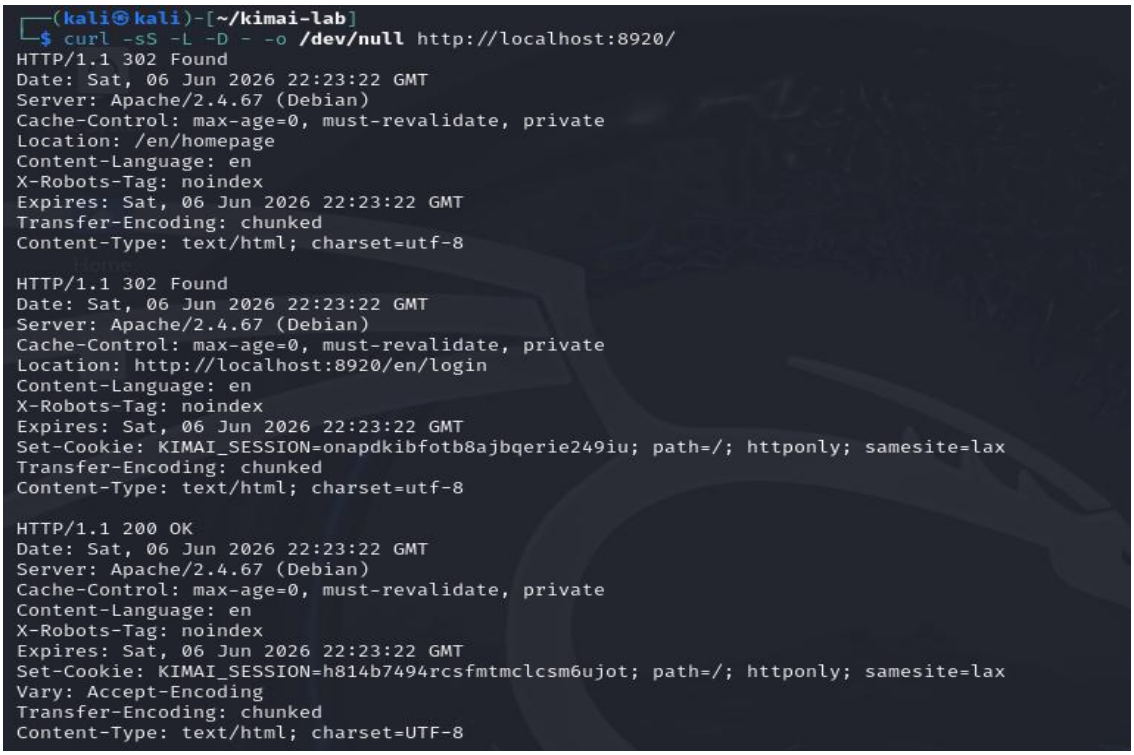


Figure 34 Manual HTTP Security Header Audit

Figure 34 shows the manual HTTP security header audit (curl) of the 2.59.0 instance. The KIMAI_SESSION cookie carries HttpOnly and SameSite=Lax, while Content-Security-Policy, X-Frame-Options, X-Content-Type-Options, Referrer-Policy, and Permissions-Policy are absent. The Server header discloses Apache/2.4.67 (Debian). This corroborates the Nikto findings.

7.3 Clickjacking and Framing

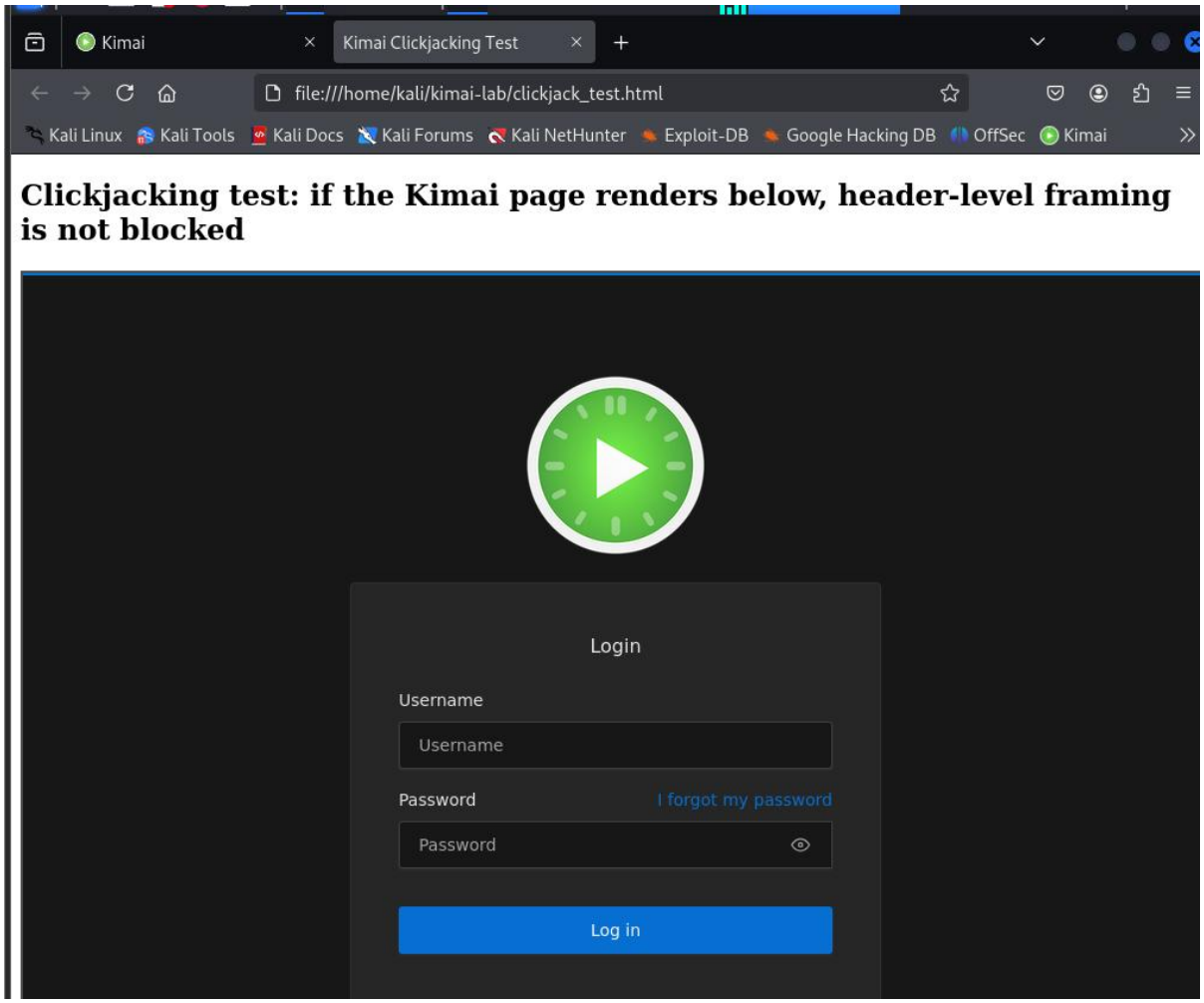


Figure 35 Cross-origin framing test

Figure 35 shows a cross-origin framing test: the Kimai login page renders fully inside an iframe served from a separate origin, confirming that public pages can be framed, which is the precondition for a clickjacking attack.

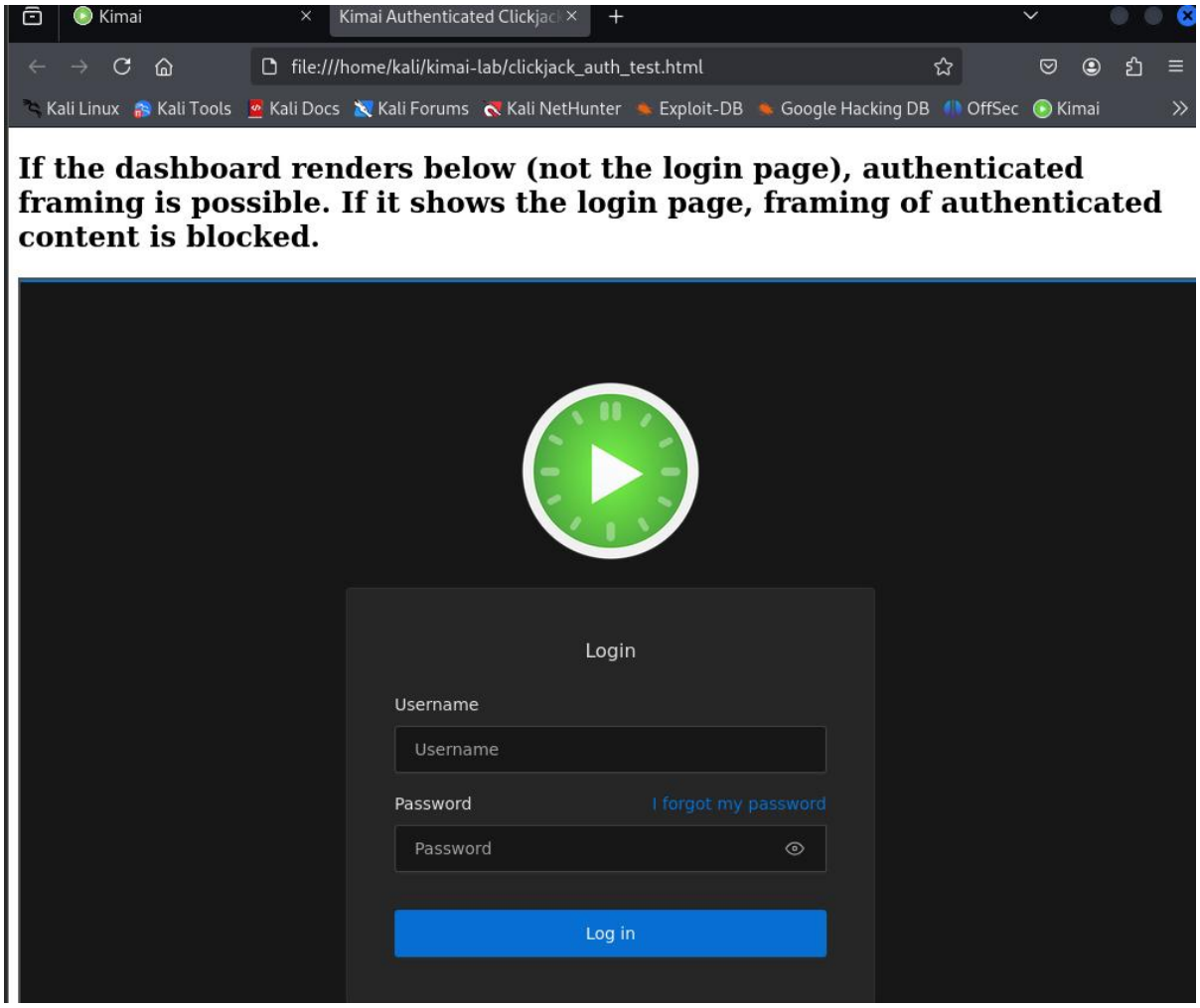


Figure 36 Authenticated framing test

Figure 36 shows the authenticated framing test: an iframe targeting /en/homepage renders the login page rather than the dashboard, because the SameSite=Lax session cookie is withheld on cross-site framed requests. Clickjacking of authenticated, state-changing actions is therefore mitigated, though an explicit X-Frame-Options or frame-ancestors policy is still recommended.

7.4 Tool Selection and Justification

Table 3 Tool Selection and Justification

Tool	Purpose	Justification
Burp Suite Community	Intercept, modify, replay, fuzz, and enumerate (Intruder)	Primary interactive proxy for manual request manipulation, mutation fuzzing, XSS, CSRF, and ID enumeration
ffuf	Content, route, and API endpoint discovery	Fast fuzzer with auto-calibration; used with SecLists wordlists, rate-limited
Nikto	Automated web server misconfiguration scan	Specialized scanner corroborating the manual header audit and checking for exposed files
sqlmap	Automated SQL injection testing	Specialized injection scanner run authenticated against multiple parameters and engines
curl	Manual request and header inspection	Lightweight verification of headers, methods, CORS, and API responses

Tool	Purpose	Justification
SecLists	Wordlists for discovery and fuzzing	Standard payload and path lists for repeatable discovery

Table 3. Tools selected for the assessment and the rationale for each.

8 OWASP Top 10 2025 Coverage

Testing activities were mapped against the OWASP Top 10 (2021) categories to demonstrate the breadth of security coverage achieved during the assessment. Each category was reviewed and linked to the corresponding evidence gathered throughout the testing process.

Table 4 Coverage of the OWASP Top 10 2025 across both instances.

Category	Coverage	Where addressed
A01 Broken Access Control	Deep	User-record access control, timesheet IDOR/BOLA, Intruder id enumeration, mutation fuzzing, CSRF, preferences mass assignment
A02 Security Misconfiguration	Deep	Nikto, manual header audit, clickjacking/framing, content discovery, CORS reflection
A03 Software Supply Chain Failures	Cross-referenced	Outdated twig/twig and Symfony components documented in the individual dependency bug report; not re-tested here
A04 Cryptographic Failures	Partial	HTTP-only deployment, absent Secure cookie flag, HttpOnly and SameSite present
A05 Injection	Deep	sqlmap (no SQLi), value and header fuzzing, stored XSS on name and description, SSTI on the Twig invoice sandbox
A06 Insecure Design	Partial	Lab 2 threat model; white-box review of the Twig StrictPolicy sandbox during SSTI testing
A07 Authentication Failures	Deep	Authentication enforcement, login rate-limiting (see Section 4.5), session fixation, logout invalidation including remember-me
A08 Software or Data Integrity Failures	Not tested	Out of scope for a runtime black-box assessment of a deployed instance
A09 Logging and Alerting Failures	Limited	Target-side access-log monitoring observed during fuzzing; no alerting access
A10 Mishandling of Exceptional Conditions	Moderate	Error-handling tests: malformed and mistyped input handled cleanly with no crashes or information leakage

9 Feature Coverage and Prioritization

Testing followed the priorities set in the Lab 2 threat model rather than testing features at random, concentrating depth on the highest-risk areas: API authorization, authentication and session management, injection, and the invoice template renderer. Of the seventeen major feature areas identified, approximately seventy percent received meaningful (deep or moderate) testing. The untested areas are concentrated in multi-format export, reporting, tagging, and the plugin system, which were deprioritized in the threat model.

Table 5 Feature coverage map; depth was concentrated on the highest-risk areas.

Feature area	Tested	Depth
Authentication / login	Yes	Deep
Session management	Yes	Deep
Access control / RBAC	Yes	Deep

Feature area	Tested	Depth
REST API	Yes	Deep
Timesheet management	Yes	Deep
HTTP transport and security headers	Yes	Deep
Input validation (form fields)	Yes	Moderate
Error handling	Yes	Moderate
Content and file exposure	Yes	Moderate
Invoice generation (Twig renderer)	Partial	Moderate
Customer / Project / Activity management	Partial	Partial
Cryptography (TLS, cookie flags, hashing)	Partial	Observed only
Multi-format export (CSV/XLSX/PDF)	No	None
Reporting, tagging, plugins, 2FA	No	None

10 Traceability: Lab 2 Priorities to Lab 3 Testing

Table 6 Traceability from the Lab 2 threat model to the Lab 3 tests.

Lab 2 threat (rank)	Severity	Addressed	How, or why deferred
IDOR on timesheet REST API	Critical	Yes	Timesheet BOLA test and Burp Intruder id enumeration as a low-privilege user
SSTI via invoice template	Critical	Yes	Escalating payloads through the renderer plus white-box sandbox review; all attempts blocked, one Low constant() observation
Stored XSS (user field)	High	Partial	Tested on activity name and description fields
Credential stuffing on login	High	Yes	Login rate-limiting test (see the Section 4.5 discrepancy)
Session hijack over HTTP	High	Partial	Session fixation and logout invalidation tested; cleartext transport noted
Non-expiring API tokens	High	Not tested	Token lifetime is a deployment configuration property; noted, not exercised
Supply-chain / dependency CVEs	Medium	Cross-referenced	Covered in the individual dependency bug report and Lab 2

11 Conclusion

In conclusion both documented instances, Kimai's mandatory controls and access-control model behaved correctly under testing. Authentication, object-level authorization, error handling, output encoding, and the Twig invoice sandbox all held, and no SQL injection, working stored XSS, or template-injection code execution was found. The residual findings are configuration and transport observations of low severity, with two items meriting follow-up: the remember-me token that is not invalidated server-side on logout, and the login brute-force discrepancy between the two instances that the group should reconcile with a single consistent test. The highest-value results in this assessment came from manual access-control and mutation testing and from the structured API authorization checks, rather than from automated scanners, which mainly corroborated the configuration findings.

12 Lab 3 Task Checklist

Table 7 Lab 3 task checklist.

Requirement	Status	Evidence
Identification of user inputs	Complete	Section 3, Figs 5–7, Table 2
Four mandatory controls tested	Complete	Section 4, Figs 8–17
Fuzzing implementation	Complete	Section 5, Figs 18–23
Injection and application-logic testing	Complete	Section 6, Figs 24–31
Specialized scanners	Complete	Section 7, Figs 32–35 (Nikto, sqlmap, header audit)
Additional tools and justification	Complete	Section 7.4, Table 3
OWASP Top 10 2025 mapping	Complete	Section 8, Table 4
Feature coverage map	Complete	Section 9, Table 5

References

Anchore. (2024). *Syft*. GitHub. https://github.com/anchore/syft

CycloneDX. (2024). *CycloneDX PHP Composer plugin*. GitHub. https://github.com/CycloneDX/cyclonedx-php-composer

Court of Justice of the European Union. (2019, May 14). *Federación de Servicios de Comisiones Obreras (CCOO) v. Deutsche Bank SAE, Case C-55/18*. https://curia.europa.eu/juris/liste.jsf?num=C-55/18

GitHub Advisory Database. (2023). *GHSA-fjhg-96cp-6fcw: Authenticated SSTI to RCE via uploaded Twig file in Kimai*. https://github.com/advisories/GHSA-fjhg-96cp-6fcw

National Vulnerability Database. (2026). *CVE-2026-40479: Stored XSS via incomplete HTML attribute escaping in Kimai*. https://nvd.nist.gov/vuln/detail/CVE-2026-40479

OWASP Foundation. (2024). *OWASP Web Security Testing Guide v4.2*. https://owasp.org/www-project-web-security-testing-guide/

OWASP Foundation. (2021). *OWASP Top 10*. https://owasp.org/Top10/

Papst, K. (2026). *Kimai: Time-tracking application*. GitHub. https://github.com/kimai/kimai

Papst, K. (2026). *Kimai bughunter documentation*. Kimai. https://www.kimai.org/documentation/bughunter.html

Papst, K. (2026). *Kimai security policy*. GitHub. https://github.com/kimai/kimai/blob/main/SECURITY.md

PortSwigger. (n.d.). *Burp Suite Community Edition*. https://portswigger.net/burp/communitydownload

ffuf. (n.d.). *ffuf: Fast web fuzzer*. GitHub. https://github.com/ffuf/ffuf

sqlmap. (n.d.). *sqlmap: Automatic SQL injection and database takeover tool*. https://sqlmap.org/

Sullo. (n.d.). *Nikto web server scanner*. GitHub. https://github.com/sullo/nikto

Daniel Miessler. (n.d.). *SecLists: Security tester's companion wordlists*. GitHub. https://github.com/danielmiessler/SecLists