Lab 5

Hardening and Secure Deployment

Group 1

, Abdul-Ramon Adeyemi Folarin, &

, , &

School of Information Technology Administration & Security, Seneca Polytechnic

WAS705NBB: Application Security Methodology

Professors Suranjit Paul & Mark Shtern

July 13, 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

Task Checklist

Table 1 Task Checklist for lab 5

Task	Status
Continued testing and re-verification of Labs 3/4 findings	**Done**
Virtual patching (Nginx reverse proxy with TLS, headers, rate limiting, CORS)	**Done**
System hardening (passwords, session timeout, TOTP, token expiry, password policy, remember-me)	**Done**
Secure deployment strategy (environment setup, network, monitoring, backup, comparison)	**Done**
Security deployment guide with hardening checklist	**Done**
Architecture diagram	**Done**
Nikto scanner validation (post-hardening)	**Done**
OWASP ZAP scanner validation (post-hardening)	**Done**
Snyk container scan	**Done**
Big Redirect information leak identified and fixed	**Done**
Integration with threat modeling and lab history	**Done**
Gap analysis and additional controls	**Done**
AODA-compliant report	**Done**
Video demonstration	**Done**
Sharing with development team	**Done**

Executive Summary

This report documents the Lab 5 cybersecurity assessment of Kimai 2.59.0, an open-source time-tracking application developed in PHP on the Symfony framework and deployed via Docker Compose. Testing was conducted by Group 1 of WAS705 (Application Security Methodology, Summer 2026) at Seneca Polytechnic, under the supervision of Professors Suranjit Paul and Mark Shtern. The assessment targeted authentication and authorization mechanisms, secret management, API security, fuzzing, and static code analysis, in accordance with OWASP Web Security Testing Guide (WSTG) v4.2 and the OWASP Top 10 2025.

Testing was performed from a Kali Linux host at 192.168.43.145 against the Kimai Docker deployment on Ubuntu at 192.168.43.146 (host port 8920). The assessment used Burp Suite, curl, ffuf, sqlmap, Snyk, Semgrep, and ESLint with the security plugin. All testing was conducted in an authorized laboratory environment.

No authentication bypass, SQL injection, or privilege escalation was identified in the tested scenarios. Server-side role-based access control (RBAC) consistently returned HTTP 403 for unauthorized requests across both the web interface and the REST API. Object-level authorization was enforced on all tested CRUD operations, and the Symfony debug profiler was confirmed disabled in production. Passwords are hashed with bcrypt at cost factor 13.

The assessment identified three categories of weakness. The most significant is the storage of API tokens in plaintext in the database: while database access is a prerequisite, a single database disclosure yields directly usable credentials, unlike bcrypt-hashed user passwords. The tested deployment also transmits traffic over plain HTTP without TLS, and the Docker entrypoint logs the administrative password in cleartext on every startup. These are deployment-level hardening gaps rather than Kimai application code flaws. A dependency scan by Snyk identified a Critical deserialization vulnerability in the bundled phpoffice/phpspreadsheet library; exploitability within Kimai was not independently confirmed, but the dependency should be upgraded. Password policy enforcement is limited to a minimum length of 8 characters with no complexity requirement, and no API rate limiting was observed during a bounded 30-request test.

The table below summarizes all findings. Recommended controls are described in detail in the Gap Analysis section.

Table of Contents

Task Checklist 2
Executive Summary 3
Table Of Figures 7
Table Of Tables 10
1. Introduction and Scope 11
Part 1: Continued Cybersecurity Testing (Pre-Hardening Baseline) 11
1.1 HTTP Security Header Baseline 11
1.2 Plain HTTP Baseline, No TLS 12
1.3 Cleartext Admin Password in Container Logs 12
1.4 Cleartext Secrets in Environment Variables 13
1.5 Plaintext API Token Storage in Database 14
1.6 Weak Password Policy Baseline 14
1.7 No API Rate Limiting Baseline 15
1.8 Long Session Garbage-Collection Lifetime 15
1.9 Clickjacking: Application Frameable Without Restriction 16
1.10 No MFA / TOTP Enforcement 16
1.11 Session Cookie Missing Secure Flag 17
1.12 Server Version Disclosure 18
1.13 CORS Misconfiguration: Arbitrary Origin Reflection 18
1.14 Signature-Based Remember-Me Cookie Not Invalidated on Logout 19
1.15 Container Running as Root User 19
Part 2: Virtual Patching 20
2.1 Nginx Reverse Proxy Deployment 21
2.2 WAF Deployment (ModSecurity + OWASP CRS) 27
Part 3: System Hardening 33
3.1 Default Admin Password Replaced 33
3.2 Session Garbage-Collection Lifetime Reduced 34
3.3 Direct Application Port Access Removed 34
3.4 API Token Expiration Enforced 35
3.5 Password Policy Limitation Addressed 37
3.6 TOTP Two-Factor Authentication Enabled for Admin 39
3.7 Secrets Separated from Deployment Configuration 42
3.8 Container Root User Limitation 42

3.9 Twig Invoice Template Sandbox Bypass: User Information Disclosure 42
3.10 Twig Invoice Template Sandbox Bypass: Session Cookie Disclosure 43
3.11 Remember-Me Cookie Lifetime Reduced 44
3.12 Unused Apache Modules Disabled," WSTG-CONF-02, OWASP A02:2025 45
Part 4: Secure Deployment Strategy 47
4.1 Environment Setup 47
4.2 Network Segmentation and Firewall Rules 48
4.3 Monitoring and Logging Strategy 50
4.4 Backup and Recovery Considerations 50
4.5 Deployment Option Comparison 51
Part 5: Security Deployment Guide (Kimai 2.59.0) 51
5.1 Overview 51
5.2 Prerequisites 51
5.3 Deployment Architecture 52
5.4 File Structure 52
5.5 Hardening Checklist 52
5.6 Security Controls and Configurations 53
5.7 Known Limitations 53
5.8 Monitoring and Maintenance 54
5.9 Deployment Verification 54
Part 6: Validation of Deployment Guide 55
6.1 Nikto Scanner Validation (Post-Hardening) 55
6.2 OWASP ZAP Scanner Validation (Post-Hardening) 56
6.3 Snyk Container Scan (Post-Hardening) 62
6.4 Cross-Validation with Trivy (Post-Hardening) 64
6.5 Native Docker Scout Scan 65
Part 7: Sharing with Development Team 67
Mapping Lab 2 STRIDE Threats to Lab 5 Hardening Outcomes 69
Known Vulnerability Coverage (from Lab 1) 70
Gap Analysis and Additional Controls 70
Remaining Gap 1: Plaintext API Token Storage at Rest (Medium, A04:2025 Cryptographic Failures) 70
Remaining Gap 2: Admin Password in Container Startup Logs (Low, A02:2025 Security Misconfiguration) 70

Remaining Gap 3: Twig Sandbox Bypass for User and Session Data (Low, A05:2025 Injection) 70
Remaining Gap 4: Containers Running as Root (Low, A02:2025 Security Misconfiguration) 70
Remaining Gap 5: No Mandatory 2FA Enforcement (Low, A07:2025 Authentication Failures) 70
Remaining Gap 6: No Permanent API Token Expiration Enforcement (Low, A07:2025 Authentication Failures) 71
Remaining Gap 7: Outdated Dependencies with Known CVEs (A03:2025 Software Supply Chain Failures) 71
Summary 71
Appendix A: Generative AI Usage Disclosure 71

Table Of Figures

Figure 1. HTTP response headers returned by Kimai before hardening, showing the redirect chain from the root URL through to the login page. 11
Figure 2. HTTPS connection attempt to Kimai on port 8920 failing with a TLS handshake error 12
Figure 3. Docker container logs showing the Kimai admin password in cleartext on every startup. 13
Figure 4. Environment variables inside the Kimai container showing admin credentials and database connection string in plaintext. 13
Figure 5. Kimai database query showing API tokens stored in plaintext with no expiration set. 14
Figure 6. Kimai accepting "password" as a valid password for a new user account. 14
Figure 7. Ten consecutive API requests with invalid bearer tokens sent directly to the Kimai application on port 8001, all returning HTTP 401 with no rate limiting or throttling. 15
Figure 8. PHP session garbage-collection lifetime set to 604,800 seconds (7 days) inside the Kimai container. 15
Figure 9. Kimai login page rendered inside an iframe in a local HTML file, confirming the absence of framing protections. 16
Figure 10. Database query showing TOTP is not enabled for any Kimai user account, including the super-admin. 17
Figure 11. Kimai session cookies showing HttpOnly and SameSite=Lax attributes but no Secure flag. 17
Figure 12. HTTP response header revealing the exact Apache version and operating system. ... 18
Figure 13. Kimai API reflecting an arbitrary Origin header (https://evil.com) in the Access-Control-Allow-Origin response. 18
Figure 14. Kimai security.yaml showing remember-me configuration with always_remember_me enabled, a 7-day lifetime, and signature-based validation. 19
Figure 15. Both the Kimai application and MySQL database containers running as root (uid=0), with MySQL port not published to the host. 20
Figure 16. Generating a self-signed TLS certificate and RSA private key for the Nginx reverse proxy using OpenSSL. 20
Figure 17. Kimai application fully functional behind the Nginx HTTPS reverse proxy, with the address bar confirming a TLS connection at https://localhost. 22
Figure 18. Nginx reverse proxy configuration file implementing TLS termination, security headers, CORS restriction, rate limiting, and cookie hardening. 23
Figure 19. Final hardened Docker Compose configuration with Nginx reverse proxy, secret separation via environment variables, session timeout, password policy, and remember-me cookie volume mounts, and no direct application port exposure. 24
Figure 20. Post-hardening HTTP response headers confirming all security headers, TLS, Secure cookie flag, and suppressed server version. 25
Figure 21. Post-hardening clickjacking test showing Firefox blocking Kimai from loading inside an iframe due to X-Frame-Options DENY. 26
Figure 22. Post-hardening CORS test showing no Access-Control-Allow-Origin header returned for a forged evil.com origin. 27
Figure 23. Post-hardening API rate limiting test showing HTTP 429 responses after the burst allowance is exhausted. 27

Figure 24 Created Custom rules for Modsecurity pt1 28
Figure 25 Created Custom rules for Modsecurity pt 2 28
Figure 26 ModSecurity WAF startup and successful proxied request through the hardened Nginx reverse proxy. 29
Figure 27 Testing Each rule 30
Figure 28 Raw ModSecurity audit log entries showing the complete chain of evaluation for each custom rule, cross-referenced against the OWASP CRS anomaly-scoring engine 30
Figure 29 All five custom rule test requests confirmed returning HTTP 403, with the full audit-log history showing the before/after effect of the block→deny fix' 31
Figure 30 ModSecurity blocking three malicious payloads before they reach the Kimai application. 31
Figure 31 ModSecurity audit log detail showing the specific rule matches behind the blocked requests in Figure 49. 32
Figure 32 Legitimate request to the Kimai login page passing through the WAF unaffected. 33
Figure 33. Kimai admin password changed from the default "changemeplease" to a strong password meeting complexity requirements. 33
Figure 34. PHP session garbage-collection lifetime reduced from 604,800 seconds (7 days) to 3,600 seconds (1 hour), persisted via a mounted configuration file. 34
Figure 35. Updated Docker Compose configuration removing the direct port 8920 mapping, forcing all traffic through the Nginx reverse proxy. 35
Figure 36. API token expiration set to 90 days, replacing the previous NULL (no expiration) value. 36
Figure 37. Token expiry enforcement script that automatically sets a 90-day expiration on any API token created without one. 36
Figure 38. Before-and-after test confirming the token expiry enforcement script detects and updates tokens with no expiration. 37
Figure 39. Kimai rejecting the weak password "password" with three complexity violation messages after applying the custom validation rules. 38
Figure 40. Custom Symfony validation.yaml file adding password complexity requirements via regex constraints, mounted into the Kimai container as a read-only volume. 39
Figure 41. Kimai Two-Factor (2FA) tab for the admin account showing TOTP is activated with a green checkmark. 40
Figure 42. Kimai login process requiring a TOTP confirmation code after successful password authentication for the admin account. 41
Figure 43. The .env file containing all deployment secrets with owner-only read/write permissions (chmod 600), with all default passwords replaced. 42
Figure 44. Malicious invoice template rendering the server OS, admin username, email, and roles through the Twig sandbox. 43
Figure 45. Malicious invoice template extracting the KIMAI_SESSION cookie value through the Twig sandbox using app.request.cookies.get(). 44
Figure 46. Kimai local.yaml configuration overriding the remember-me cookie lifetime from 604,800 seconds (7 days) to 3,600 seconds (1 hour), confirmed by Symfony's resolved config output. 44
Figure 47. Firefox Developer Tools showing the KIMAI_REMEMBER cookie with an expiry of exactly 1 hour after creation. 45
Figure 48 Full Apache module list on the Kimai container before hardening. 45

Figure 49 Confirmation that the standard Debian Apache module-management tooling is available inside the Kimai container. 46
Figure 50 Confirmation that the standard Debian Apache module-management tooling is available inside the Kimai container. 46
Figure 51 Confirmation that the disabled Apache endpoints are no longer accessible after hardening. 47
Figure 52 Verification of docker-compose.yml restart policies and .env-based secret separation, with one outstanding gap in session lifetime configuration. 48
Figure 53 Active nftables ruleset applied to the Kali host, with the nftables service enabled and confirmed running. 49
Figure 54 Testing the firewall and WAF from the host's real network interface, revealing both the WAF's IP-based Host header rule and the Docker FORWARD-chain gap. 50
Figure 55. Kimai Hardened Deployment Architecture showing the three-container Docker Compose deployment with Nginx reverse proxy, isolated application and database containers, and mounted configuration files. 55
Figure 56. Nikto scan against the hardened HTTPS deployment returning only one low-confidence finding out of 7,797 requests. 56
Figure 57. ZAP Alerts summary from the authenticated active scan against the hardened Kimai deployment, showing 1 High, 5 Medium, 1 Low, and 7 Informational findings across 14 alert types. 57
Figure 58. Manual SQL injection verification reproducing the exact ZAP payload against /api/timesheets. All four injection techniques return HTTP 400 while the valid baseline returns HTTP 200. 58
Figure 59. CSRF token verification across Kimai forms and session cookie attributes. Both the login form (_csrf_token) and the invoice search form (_token) contain anti-CSRF tokens. The session cookie includes SameSite=Lax, HttpOnly, and Secure attributes. 59
Figure 60. Content Security Policy verification showing the deployed CSP header, inline script usage, dynamic JavaScript attributes, and the X-Frame-Options DENY header. 60
Figure 61. Apache directory redirect leaking the internal Docker port 8001 in the Location header for /build and /bundles paths. Requests with a trailing slash return HTTP 404, confirming these are not accessible directories. 61
Figure 62. Adding the proxy_redirect directive to the Nginx configuration, restarting the Nginx container, and re-testing the /build and /bundles redirects. The Location headers now show https://localhost/ instead of exposing the internal Docker port 8001. 62
Figure 63 Snyk container scan of the kimai/kimai2:apache Docker image showing 274 OS-level vulnerabilities, 12 application dependency vulnerabilities (286 total), and 5 projects scanned. .. 63
Figure 64 Trivy vulnerability scan summary and top composer-dependency findings for the kimai/kimai2:apache image.pt 1 64
Figure 65 Trivy vulnerability scan summary and top composer-dependency findings for the kimai/kimai2:apache image.pt 2 65
Figure 66. Docker Scout scan initialization and image metadata for kimai/kimai2:latest 66
Figure 67 Docker Scout vulnerability summary and severity breakdown for kimai/kimai2:latest. 67
Figure 68 Submission of the Security Deployment Guide to the Kimai project support channel and the resulting automated ticket acknowledgment. 68

Table Of Tables

Table 1 Task Checklist for lab 5 2
Table 2 Mapping Lab 2 STRIDE Threats to Lab 5 Hardening Outcomes 69

1. Introduction and Scope

This lab hardens and securely deploys Kimai 2.59.0, the open-source time-tracking application selected by Group 1 for the WAS705 project. All testing was performed in an isolated lab environment on a Kali Linux host running the application through Docker Compose. The deployment under test exposes Kimai on a local port, and all work described here was performed against systems the group controls.

The report is organized to follow the Lab 5 brief. Part 1 re-establishes the pre-hardening baseline by re-verifying the findings carried forward from Labs 3 and 4. Part 2 describes the virtual patching layer implemented with an Nginx reverse proxy. Part 3 covers configuration-level system hardening. Part 4 presents the secure deployment strategy. Part 5 is a step-by-step security deployment guide. Part 6 validates the hardened deployment with Nikto, OWASP ZAP, and Snyk. The report closes with a task checklist, a mapping of the Lab 2 STRIDE threats to the Lab 5 outcomes, a review of known vulnerability coverage, and a gap analysis of residual risks.

Every finding is mapped to the OWASP Web Security Testing Guide (WSTG v4.2) test identifier and to the corresponding OWASP Top 10 2025 category. All screenshots were captured during the group's own testing of the deployment.

Part 1: Continued Cybersecurity Testing (Pre-Hardening Baseline)

This part re-verifies the vulnerabilities documented in Labs 3 and 4 against the current deployment to establish an accurate pre-hardening baseline. Each finding below is confirmed with direct evidence and mapped to WSTG v4.2 and OWASP Top 10 2025.

1.1 HTTP Security Header Baseline

WSTG: WSTG-CONF-07 **OWASP Top 10 2025:** A02:2025 Security Misconfiguration

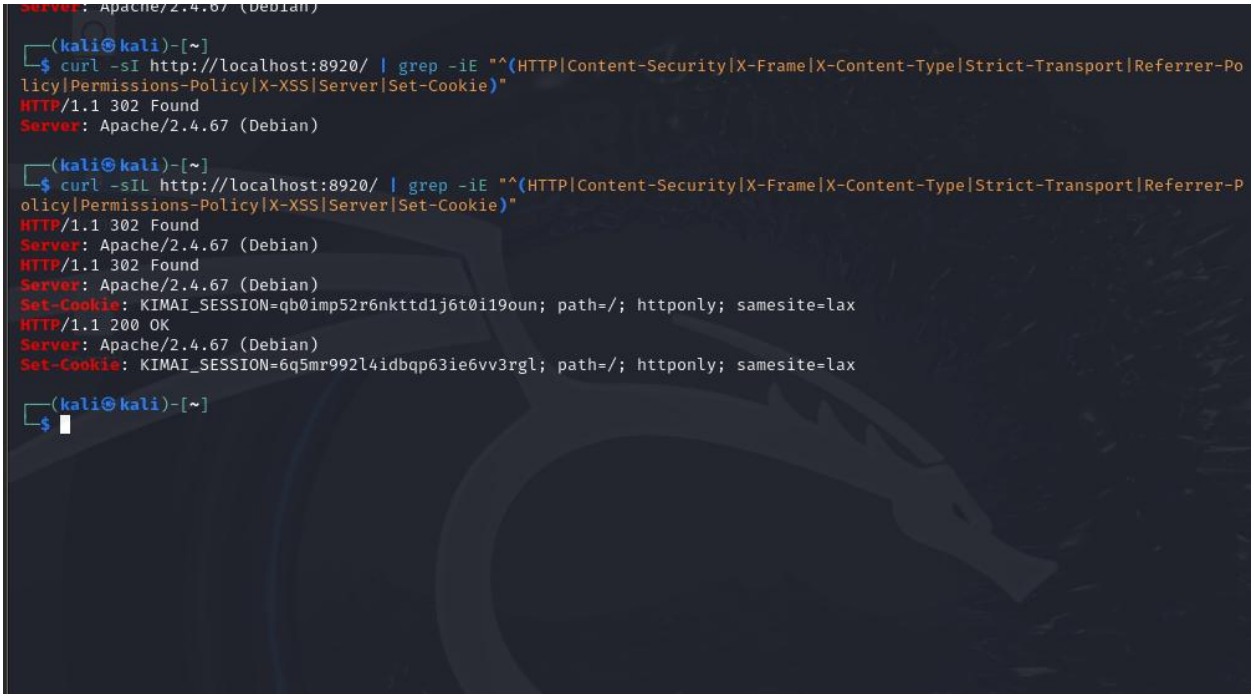


Figure 1. HTTP response headers returned by Kimai before hardening, showing the redirect chain from the root URL through to the login page.

Figure 1 shows the HTTP response headers across Kimai's redirect chain from the root URL to the login page. The grep filter checks for all standard security headers. The results confirm that the following headers are absent: Content-Security-Policy, X-Frame-Options, X-Content-Type-Options, Strict-Transport-Security, Referrer-Policy, and Permissions-Policy. The server banner exposes the exact Apache version (2.4.67 on Debian). The session cookie includes HttpOnly and SameSite=Lax attributes but lacks the Secure flag, which is expected given that the deployment uses plain HTTP. These observations match the findings documented in Labs 3 and 4 and confirm that no security headers have been added since the previous assessment.

1.2 Plain HTTP Baseline, No TLS

WSTG: WSTG-CRYP-01 **OWASP Top 10 2025:** A02:2025 Security Misconfiguration

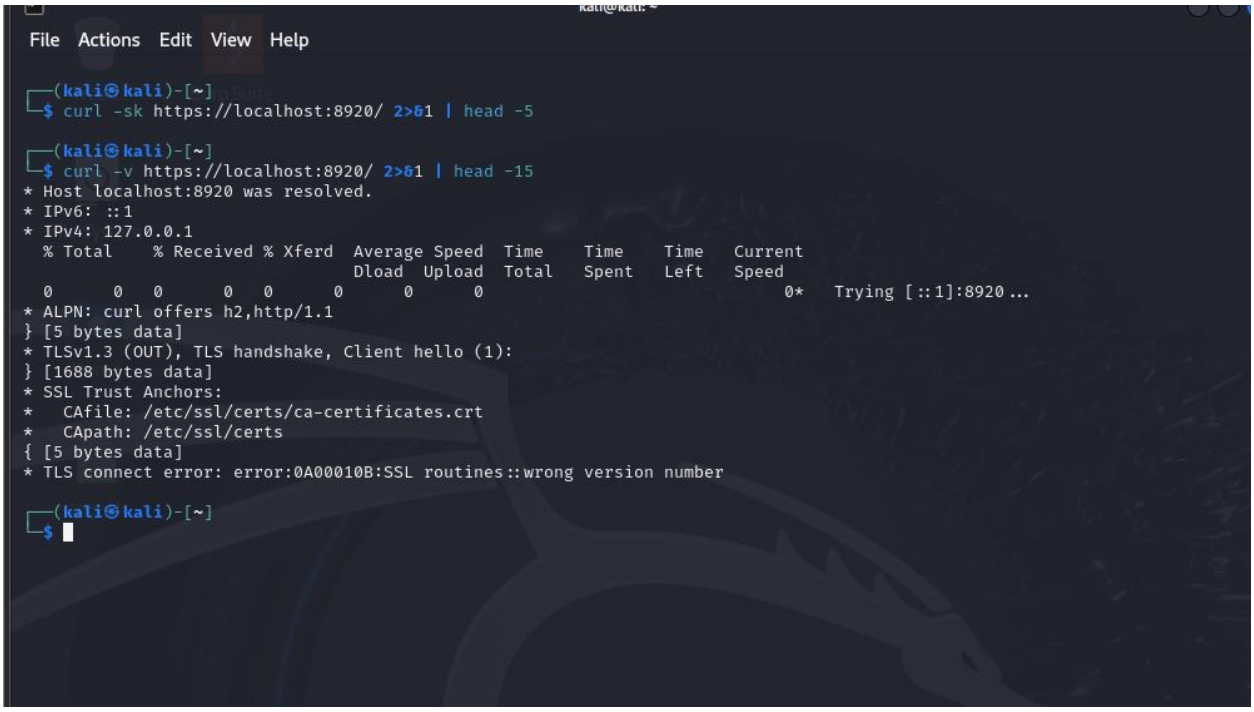


Figure 2. HTTPS connection attempt to Kimai on port 8920 failing with a TLS handshake error

Figure 2 shows a verbose curl request attempting a TLS connection to the Kimai deployment. The client sends a TLSv1.3 Client Hello, and the server responds with a "wrong version number" error because port 8920 serves plain HTTP, not HTTPS. This confirms that TLS is not configured on the tested deployment. All traffic, including session cookies, API bearer tokens, and user credentials, is transmitted in cleartext. This matches the finding documented in Labs 3 and 4 and establishes the pre-hardening baseline for the TLS gap.

1.3 Cleartext Admin Password in Container Logs

WSTG: WSTG-CONF-02 **OWASP Top 10 2025:** A02:2025 Security Misconfiguration

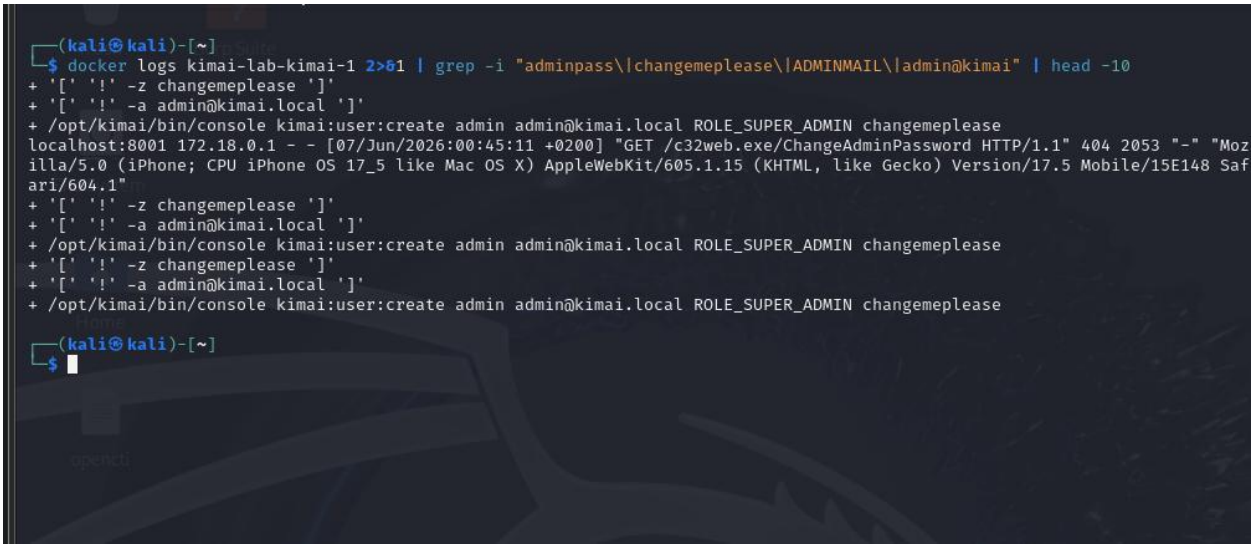


Figure 3. Docker container logs showing the Kimai admin password in cleartext on every startup.

Figure 3 shows the output of the Docker log for the Kimai container, filtered for references to the administrative credentials. The entrypoint script passes the admin password "changemeplease" as a command-line argument to the kimai:user:create console command on every container startup. This password appears in plaintext at least three times in the log, corresponding to separate container starts. The admin email address (admin@kimai.local) is also visible. Any user or process with access to the Docker daemon or container logs can read these credentials. This confirms the finding from Lab 4 (Section 3.3) and establishes the pre-hardening baseline.

1.4 Cleartext Secrets in Environment Variables

WSTG: WSTG-CONF-02 **OWASP Top 10 2025:** A02:2025 Security Misconfiguration



Figure 4. Environment variables inside the Kimai container showing admin credentials and database connection string in plaintext.

Figure 4 shows the environment variables exposed inside the running Kimai container. The admin password (changemeplease), admin email, and full database connection string (including the MySQL username and password "kimai") are all stored as plaintext environment variables. Any process inside the container or any user with docker exec access can read these values. APP_SECRET was not returned by this command, indicating it is set through the application's internal .env file rather than the Docker environment. This confirms the finding from Lab 4 (Section 3.1) and establishes the pre-hardening baseline for secrets management.

1.5 Plaintext API Token Storage in Database

WSTG: WSTG-CONF-02 **OWASP Top 10 2025:** A04:2025 Cryptographic Failures



Figure 5. Kimai database query showing API tokens stored in plaintext with no expiration set.

Figure 5 shows a direct query against the kimai2_access_token table. The token column stores API bearer tokens as plaintext strings rather than hashed values. Three tokens are present across two user accounts. Two of the three tokens (lab2-testing and pentest-admin) have a NULL expires_at value, meaning they never expire. A single database compromise would yield directly usable authentication credentials, unlike the bcrypt-hashed user passwords in kimai2_users. This confirms the Medium-severity finding from Lab 4 (Section 3.4) and establishes the pre-hardening baseline for token storage.

1.6 Weak Password Policy Baseline

WSTG: WSTG-ATHN-07 **OWASP Top 10 2025:** A07:2025 Authentication Failures



Figure 6. Kimai accepting "password" as a valid password for a new user account.

Figure 6 shows the successful creation of a user account with the password "password," a dictionary word that meets only the minimum length requirement of 8 characters. Kimai does not

enforce complexity rules, does not block common or known-compromised passwords, and does not require mixed character types. This confirms the Low-severity finding from Lab 4 (Section 1.2) and establishes the pre-hardening baseline for password policy.

1.7 No API Rate Limiting Baseline

WSTG: WSTG-ATHN-03 **OWASP Top 10 2025:** A07:2025 Authentication Failures



Figure 7. Ten consecutive API requests with invalid bearer tokens sent directly to the Kimai application on port 8001, all returning HTTP 401 with no rate limiting or throttling.

Figure 7 test sends 10 rapid requests to the Kimai timesheets API endpoint from inside the Docker network, bypassing any external proxy. Each request uses a different invalid bearer token. All 10 return HTTP 401 Unauthorized with no HTTP 429 response, no delay, and no connection refusal, confirming that Kimai itself does not enforce per-token or per-IP API rate limiting. This confirms the Low-severity observation from Lab 4 (Section 4.2) and establishes the pre-hardening baseline for API rate limiting.

1.8 Long Session Garbage-Collection Lifetime

WSTG: WSTG-SESS-07 **OWASP Top 10 2025:** A07:2025 Authentication Failures

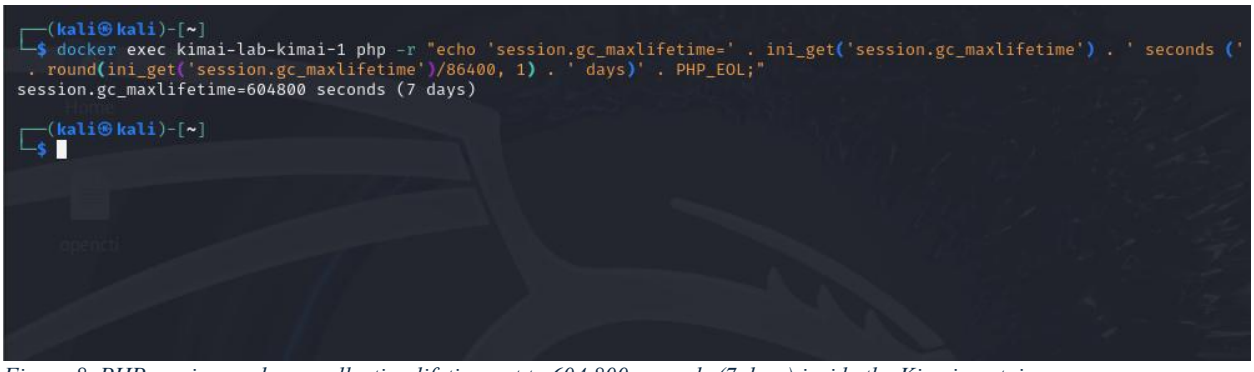


Figure 8. PHP session garbage-collection lifetime set to 604,800 seconds (7 days) inside the Kimai container.

Figure 8 shows the PHP session.gc_maxlifetime value configured at 604,800 seconds, equivalent to 7 days. This means a session cookie can remain valid for up to 7 days after last use before PHP's garbage collector is eligible to remove it. For a time-tracking application handling billing data, this window is longer than necessary and increases the risk of session reuse if a cookie is

stolen or a workstation is left unattended. This confirms the Low-severity observation from Lab 4 (Section 1.3).

1.9 Clickjacking: Application Frameable Without Restriction

WSTG: WSTG-CLNT-09 **OWASP Top 10 2025:** A02:2025 Security Misconfiguration

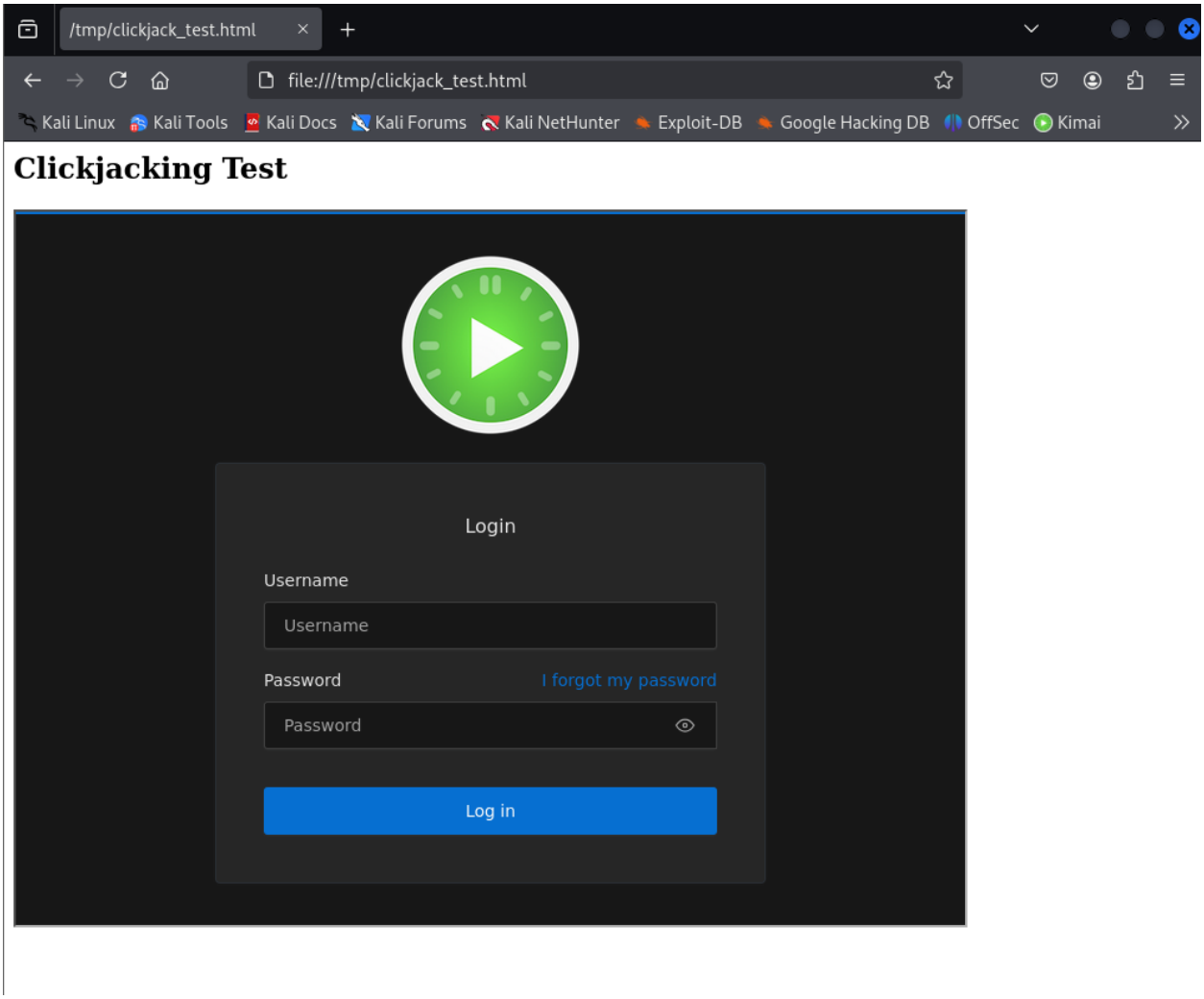


Figure 9. Kimai login page rendered inside an iframe in a local HTML file, confirming the absence of framing protections.

Figure 9 shows a local HTML page embedding Kimai's login page inside an iframe. The login form, including the username field, password field, and "Log in" button, renders fully within the frame. This is possible because the server does not send an X-Frame-Options header or a Content-Security-Policy frame-ancestors directive. An attacker could host a page that overlays transparent elements on top of the framed login form to capture credentials or trick an authenticated user into performing unintended actions. The SameSite=Lax attribute on the session cookie limits cross-site cookie attachment, which reduces but does not eliminate the risk. This confirms the observation from Lab 3 (Section 7.3).

1.10 No MFA / TOTP Enforcement

WSTG: WSTG-ATHN-09 **OWASP Top 10 2025:** A07:2025 Authentication Failures



Figure 10. Database query showing TOTP is not enabled for any Kimai user account, including the super-admin.

Figure 10 shows all four user accounts in the kimai2_users table. The totp_enabled column is 0 for every account, meaning no user has two-factor authentication active. User 2 (Alice) has a totp_secret value present from earlier testing but has not completed TOTP enrollment (totp_enabled remains 0). Most significantly, the super-admin account (id 1) has no TOTP configured at all. Kimai supports TOTP-based two-factor authentication but does not enforce it by default, leaving all accounts protected by password alone. For a time-tracking application handling billing and financial data, the absence of mandatory MFA on administrative accounts is a Low-severity hardening gap.

1.11 Session Cookie Missing Secure Flag

WSTG: WSTG-SESS-02 **OWASP Top 10 2025:** A02:2025 Security Misconfiguration



Figure 11. Kimai session cookies showing HttpOnly and SameSite=Lax attributes but no Secure flag.

Figure 11 shows the Set-Cookie headers returned during the redirect chain to the Kimai login page. Two session cookies are issued (one per redirect hop), both named KIMAI_SESSION. Each cookie includes the HttpOnly attribute (preventing JavaScript access) and SameSite=Lax (limiting cross-site attachment). However, neither cookie includes the Secure flag, which means the cookie will be transmitted over unencrypted HTTP connections. On any network where traffic can be intercepted, an attacker could capture the session cookie in transit. The absence of the Secure flag is a direct consequence of the deployment lacking TLS, since setting Secure on a

plain HTTP deployment would break functionality. This will be resolved as part of the TLS hardening in Part 2.

1.12 Server Version Disclosure

WSTG: WSTG-INFO-02 **OWASP Top 10 2025:** A02:2025 Security Misconfiguration



Figure 12. HTTP response header revealing the exact Apache version and operating system.

Figure 12 shows the Server header returned by Kimai, which discloses "Apache/2.4.67 (Debian)." This reveals both the exact web server version and the underlying operating system family. An attacker can use this information to search for known vulnerabilities specific to that Apache version. No X-Powered-By header is present, meaning PHP does not separately disclose its version through response headers. The recommended control is to suppress or minimize the Server header value so that it does not reveal version details.

1.13 CORS Misconfiguration: Arbitrary Origin Reflection

WSTG: WSTG-CLNT-07 **OWASP Top 10 2025:** A02:2025 Security Misconfiguration



Figure 13. Kimai API reflecting an arbitrary Origin header (https://evil.com) in the Access-Control-Allow-Origin response.

Figure 13 shows a curl request to the Kimai timesheets API with a forged Origin header set to "https://evil.com." The server reflects this value back in the Access-Control-Allow-Origin response header, meaning Kimai does not validate or restrict which origins may make cross-origin API requests. The server does not return Access-Control-Allow-Credentials: true, which limits direct exploitation through browser-based cookie attachment. However, the permissive

CORS policy still violates the principle of least privilege and could facilitate data exfiltration if combined with other weaknesses. This confirms the observation from Lab 3 (Section 4.5) on version 2.58.0, now also verified on 2.59.0.

1.14 Signature-Based Remember-Me Cookie Not Invalidated on Logout

WSTG: WSTG-SESS-03 **OWASP Top 10 2025:** A07:2025 Authentication Failures

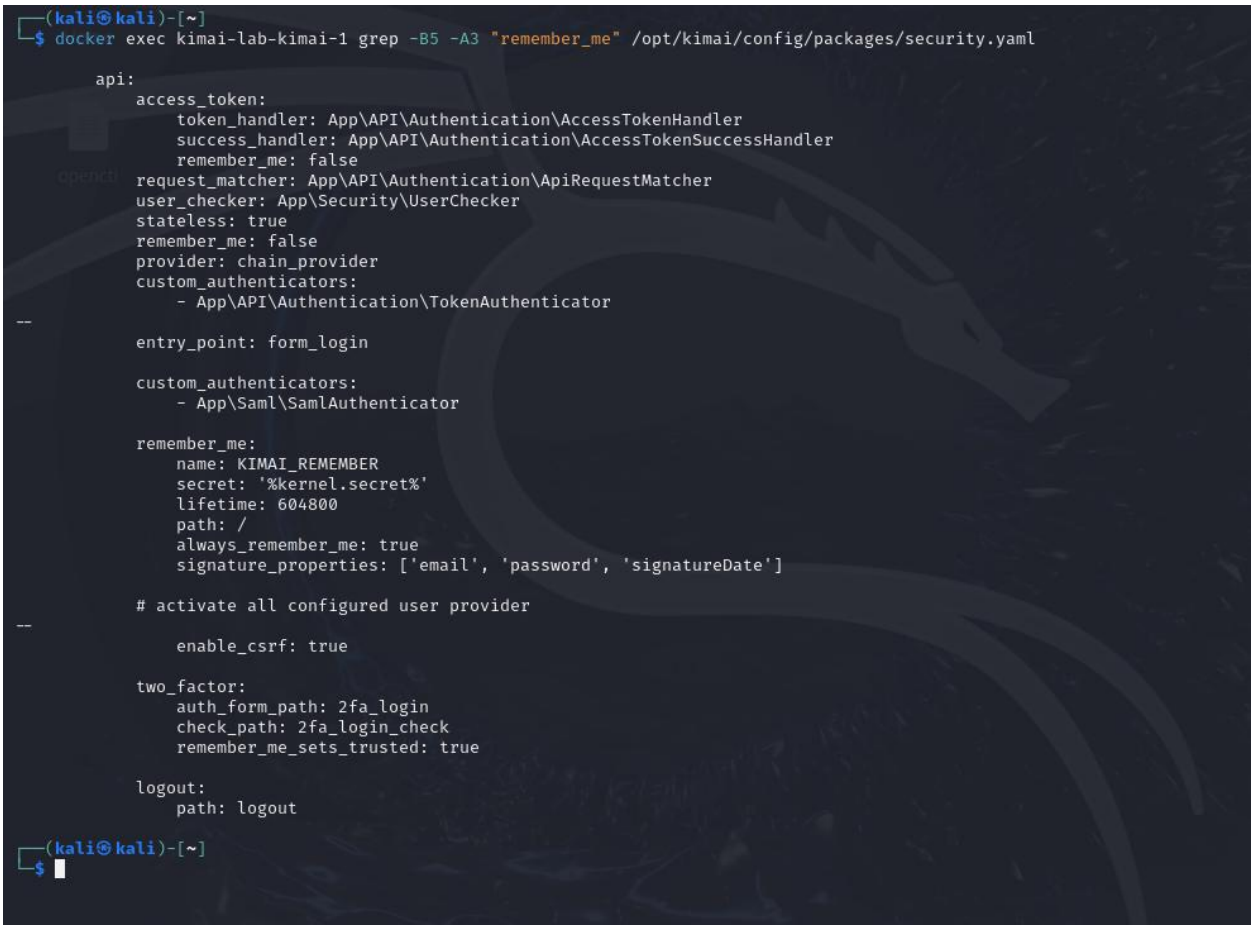


Figure 14. Kimai security.yaml showing remember-me configuration with always_remember_me enabled, a 7-day lifetime, and signature-based validation.

Figure 14 shows the remember-me configuration from Kimai's security.yaml file. The always_remember_me setting is true, meaning every web login automatically issues a persistent KIMAI_REMEMBER cookie regardless of user choice. The cookie has a 604,800-second (7-day) lifetime and is validated using a signature derived from the user's email, password hash, and signatureDate, not through a server-side token store. Because validation is signature-based, logging out destroys the server session but does not invalidate the remember-me cookie. If an attacker captures the cookie value (for example, through network interception on the unencrypted HTTP deployment), it remains usable until the 7-day expiration or until the user changes their password. The API firewall correctly has remember_me disabled. This confirms the Low to Medium observation from Lab 3 (Section 4.2).

1.15 Container Running as Root User

WSTG: WSTG-CONF-02 **OWASP Top 10 2025:** A02:2025 Security Misconfiguration

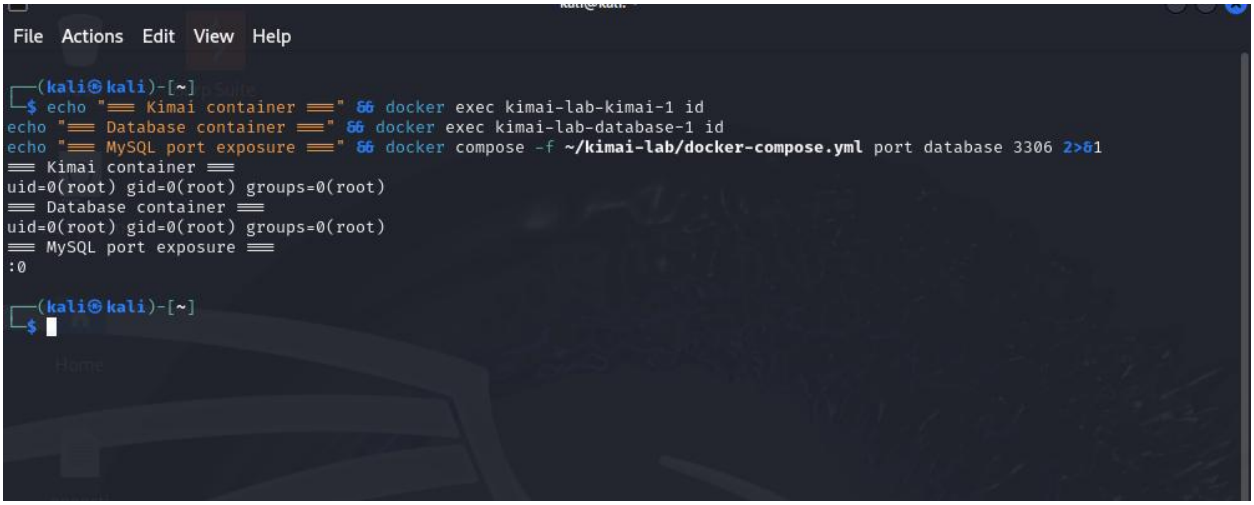


Figure 15. Both the Kimai application and MySQL database containers running as root (uid=0), with MySQL port not published to the host.

Figure 15 shows that both the Kimai and database containers run all processes as root (uid=0, gid=0). If an attacker achieves code execution inside either container, they immediately have full root-level access to the container filesystem, environment variables, and network interfaces. The MySQL port (3306) is not published to the host, which limits direct external database access, but the root execution context remains a hardening gap. Container security best practices recommend running application processes as a non-root user to limit the blast radius of a compromise.

Part 2: Virtual Patching



Figure 16. Generating a self-signed TLS certificate and RSA private key for the Nginx reverse proxy using OpenSSL.

Figure 16 shows OpenSSL generating a 2048-bit RSA private key and self-signed X.509 certificate valid for 365 days, issued for CN "localhost." The output confirms successful key

generation with no errors. These files will be used by an Nginx reverse proxy container to terminate TLS in front of Kimai, enabling HTTPS without modifying the application's source code. In a production deployment, this self-signed certificate would be replaced with one issued by a trusted certificate authority.

2.1 Nginx Reverse Proxy Deployment

The primary virtual patch applied to the Kimai deployment is an Nginx reverse proxy placed in front of the application container. This single addition addresses the following vulnerabilities from Part 1 without modifying any Kimai source code: no TLS (1.2), missing security headers (1.1), server version disclosure (1.12), CORS origin reflection (1.13), clickjacking (1.9), session cookie missing Secure flag (1.11), and no API rate limiting (1.7).

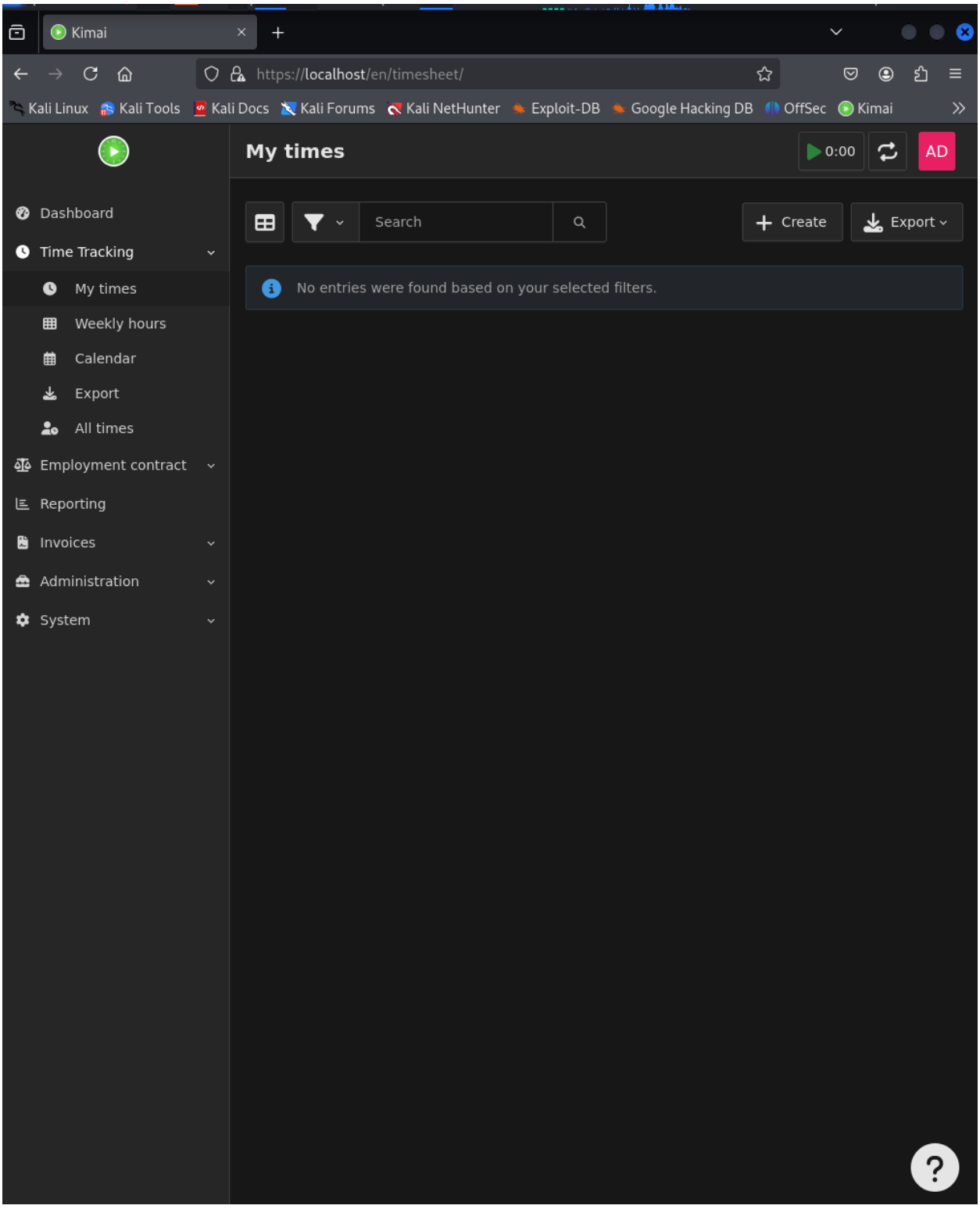


Figure 17. Kimai application fully functional behind the Nginx HTTPS reverse proxy, with the address bar confirming a TLS connection at https://localhost.

Figure 17 shows the Kimai timesheet interface loaded through the Nginx reverse proxy over HTTPS. The address bar shows "https://localhost/en/timesheet/" confirming that TLS

termination is working. The admin user is logged in and the full application is accessible, including all sidebar navigation items, confirming that the proxy does not break any application functionality.

```
File  Actions  Edit  View  Help

┌──(kali㉿kali)-[~/kimai-lab]
└─$ cat ~/kimai-lab/nginx/nginx.conf
worker_processes auto;
error_log /var/log/nginx/error.log warn;
pid /tmp/nginx.pid;

events {
    worker_connections 1024;
}

http {
    include /etc/nginx/mime.types;
    default_type application/octet-stream;
    sendfile on;
    keepalive_timeout 65;
    server_tokens off;

    limit_req_zone $binary_remote_addr zone=api:10m rate=5r/s;
    limit_req_status 429;

    server {
        listen 80;
        server_name localhost;
        return 301 https://$host$request_uri;
    }

    server {
        listen 443 ssl;
        server_name localhost;

        ssl_certificate /etc/nginx/certs/selfsigned.crt;
        ssl_certificate_key /etc/nginx/certs/selfsigned.key;
        ssl_protocols TLSv1.2 TLSv1.3;
        ssl_ciphers HIGH:!aNULL:!MD5;

        add_header X-Frame-Options "DENY" always;
        add_header X-Content-Type-Options "nosniff" always;
        add_header Content-Security-Policy "default-src 'self'; script-src 'self' 'unsafe-inline' 'unsafe-eval'; style-src
'self' 'unsafe-inline'; img-src 'self' data:; font-src 'self';" always;
        add_header Referrer-Policy "strict-origin-when-cross-origin" always;
        add_header Permissions-Policy "camera=(), microphone=(), geolocation=()" always;
        add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;

        location /api/ {
            limit_req zone=api burst=10 nodelay;
            proxy_pass http://kimai:8001;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto https;
            proxy_cookie_flags ~ Secure;
            proxy_hide_header Access-Control-Allow-Origin;
            proxy_hide_header Access-Control-Allow-Credentials;
            proxy_hide_header Access-Control-Allow-Methods;
            proxy_hide_header Access-Control-Allow-Headers;
        }

        location / {
            proxy_pass http://kimai:8001;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto https;
            proxy_cookie_flags ~ Secure;
            proxy_hide_header Access-Control-Allow-Origin;
        }
    }
}

┌──(kali㉿kali)-[~/kimai-lab]
└─$
```

Figure 18. Nginx reverse proxy configuration file implementing TLS termination, security headers, CORS restriction, rate limiting, and cookie hardening.

Figure 18 shows the complete nginx.conf used as the virtual patching layer. The port 80 server block redirects all HTTP traffic to HTTPS (fixing 1.2 No TLS). The SSL block enforces TLSv1.2 and TLSv1.3 with strong ciphers. Six security headers are injected on every response: X-Frame-Options DENY (fixing 1.9 Clickjacking), X-Content-Type-Options nosniff, Content-Security-Policy with a restrictive default-src, Referrer-Policy, Permissions-Policy, and Strict-Transport-Security (fixing 1.1 Missing Headers). The server_tokens off directive suppresses the server version (fixing 1.12 Version Disclosure). The /api/ location block applies a rate limit of 5 requests per second per IP with a burst of 10 (fixing 1.7 No Rate Limiting), and strips all CORS headers from Kimai's upstream response using proxy_hide_header directives (fixing 1.13 CORS Reflection). Both location blocks add the Secure flag to all cookies via proxy_cookie_flags (fixing 1.11 Missing Secure Flag).

```
┌──(kali㉿kali)-[~/kimai-lab]
└─$ cat ~/kimai-lab/docker-compose.yml
services:
  database:
    image: mysql:8.3
    restart: always
    environment:
      MYSQL_DATABASE: ${MYSQL_DATABASE}
      MYSQL_USER: ${MYSQL_USER}
      MYSQL_PASSWORD: ${MYSQL_PASSWORD}
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
      interval: 5s
      retries: 10
  kimai:
    image: kimai/kimai2:apache
    restart: always
    depends_on:
      database:
        condition: service_healthy
    environment:
      ADMINMAIL: ${ADMINMAIL}
      ADMINPASS: ${ADMINPASS}
      DATABASE_URL: ${DATABASE_URL}
      TRUSTED_HOSTS: "^(localhost|127\\.0\\.0\\.1|::1|nginx)$"
      APP_ENV: prod
    expose:
      - "8001"
    volumes:
      - ./session.ini:/usr/local/etc/php/conf.d/session.ini:ro
      - ./local.yaml:/opt/kimai/config/packages/local.yaml:ro
      - ./validation.yaml:/opt/kimai/config/validator/validation.yaml:ro
  nginx:
    image: nginx:stable-alpine
    restart: always
    depends_on:
      - kimai
    ports:
      - "443:443"
      - "80:80"
    volumes:
      - ./nginx/nginx.conf:/etc/nginx/nginx.conf:ro
      - ./nginx/selfsigned.crt:/etc/nginx/certs/selfsigned.crt:ro
      - ./nginx/selfsigned.key:/etc/nginx/certs/selfsigned.key:ro

┌──(kali㉿kali)-[~/kimai-lab]
└─$
```

Figure 19. Final hardened Docker Compose configuration with Nginx reverse proxy, secret separation via environment variables, session timeout, password policy, and remember-me cookie volume mounts, and no direct application port exposure.

Figure 19 shows docker-compose.yml defines all three services. All credentials are referenced through environment variable substitution (${MYSQL_PASSWORD}, ${ADMINPASS}, ${DATABASE_URL}) with values stored in a separate .env file restricted to owner-only permissions (chmod 600). The Kimai container uses "expose" instead of "ports," ensuring all external traffic passes through the Nginx reverse proxy. Three configuration files are mounted as

read-only volumes: session.ini to reduce the PHP session garbage-collection lifetime to 1 hour, local.yaml to reduce the remember-me cookie lifetime from 7 days to 1 hour, and validation.yaml to enforce password complexity requirements. All three services include "restart: always" to survive host reboots. The Nginx container binds to standard ports 80 and 443 for TLS termination.

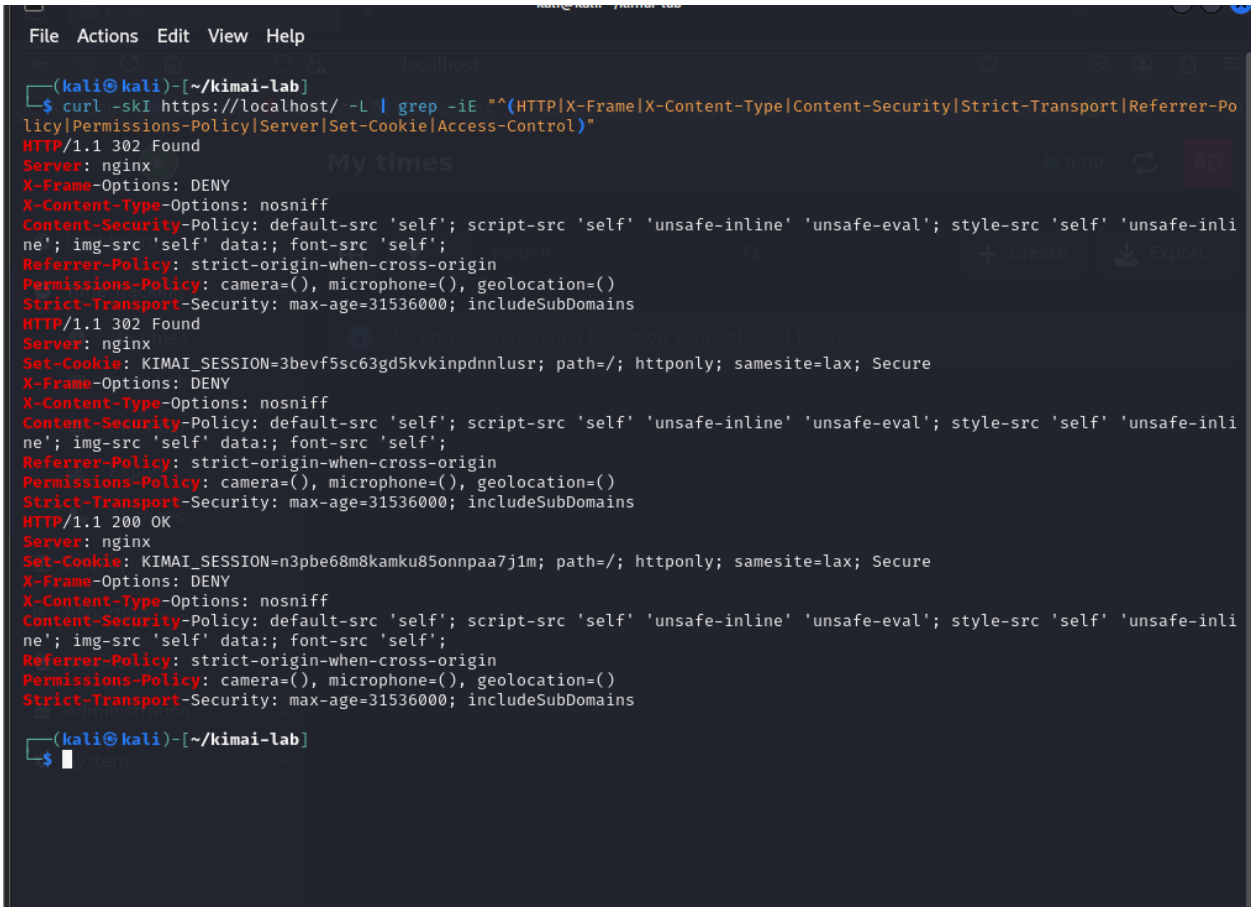


Figure 20. Post-hardening HTTP response headers confirming all security headers, TLS, Secure cookie flag, and suppressed server version.

Figure 20 shows the response headers across the full redirect chain through the Nginx reverse proxy. Compared to the pre-hardening baseline in Figure 1, every previously missing security header is now present on all three responses: X-Frame-Options DENY, X-Content-Type-Options nosniff, Content-Security-Policy, Referrer-Policy, Permissions-Policy, and Strict-Transport-Security. The Server header now shows only "nginx" with no version number, replacing the previous "Apache/2.4.67 (Debian)" disclosure. The Set-Cookie header now includes the Secure flag alongside the existing HttpOnly and SameSite=Lax attributes. No Access-Control-Allow-Origin header appears, confirming the CORS origin reflection is suppressed.

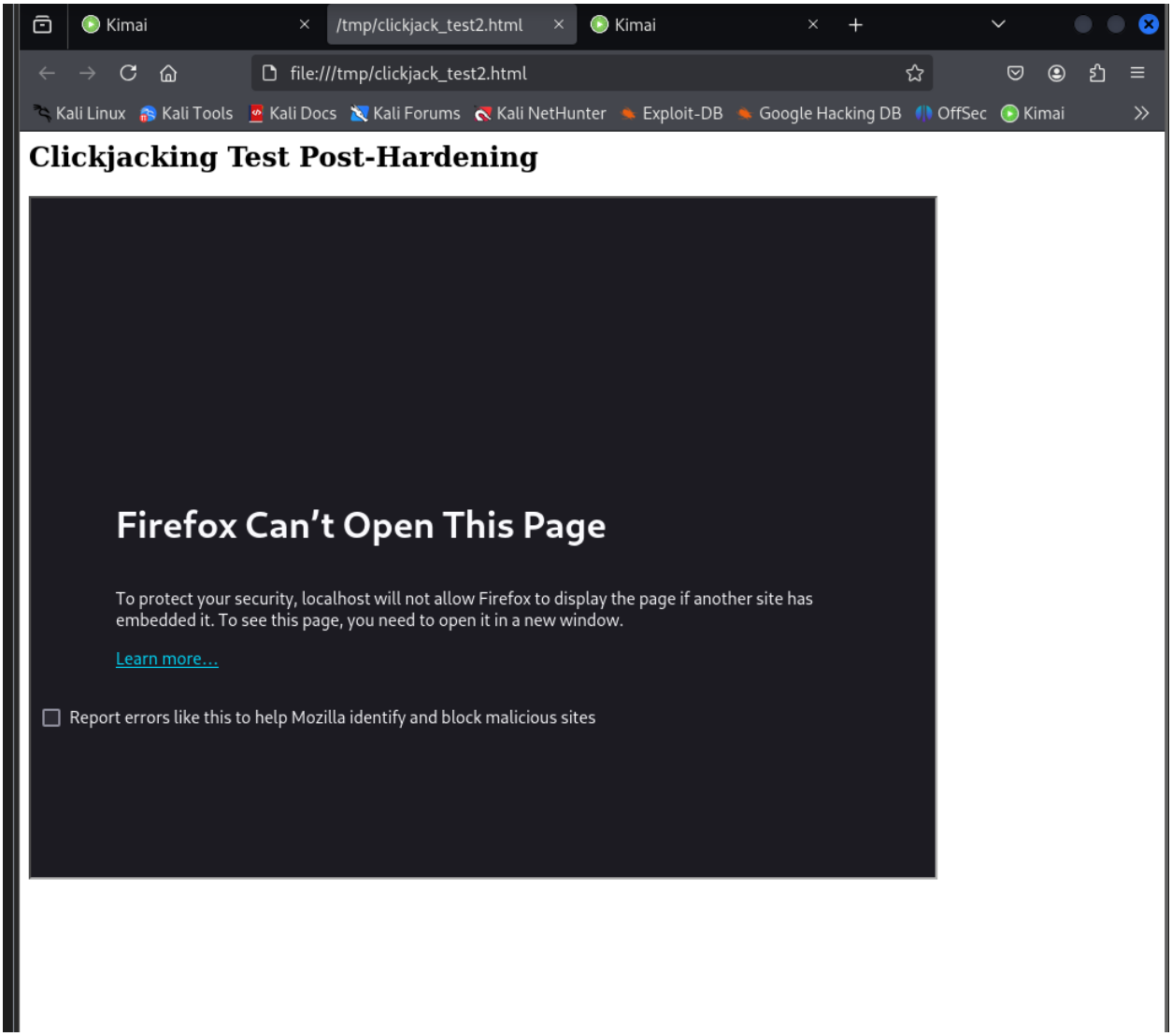


Figure 21. Post-hardening clickjacking test showing Firefox blocking Kimai from loading inside an iframe due to X-Frame-Options DENY.

Figure 21 shows the same clickjacking test from Figure 9, now attempted against the hardened deployment. Firefox displays "Firefox Can't Open This Page" and states that "localhost will not allow Firefox to display the page if another site has embedded it." This confirms that the X-Frame-Options DENY header injected by the Nginx reverse proxy successfully prevents the application from being framed, mitigating the clickjacking vulnerability identified in Section 1.9.



Figure 22. Post-hardening CORS test showing no Access-Control-Allow-Origin header returned for a forged evil.com origin.

Figure 22 shows the same CORS test from Figure 13, now run against the hardened deployment through the Nginx reverse proxy. The request sends a forged Origin header of "https://evil.com" with a valid API token. The response returns HTTP 200 OK but contains no Access-Control-Allow-Origin header. In the pre-hardening baseline, this same request caused Kimai to reflect "https://evil.com" in the response. The proxy_hide_header directives in the Nginx /api/ location block now strip all CORS headers from Kimai's upstream response before they reach the client.

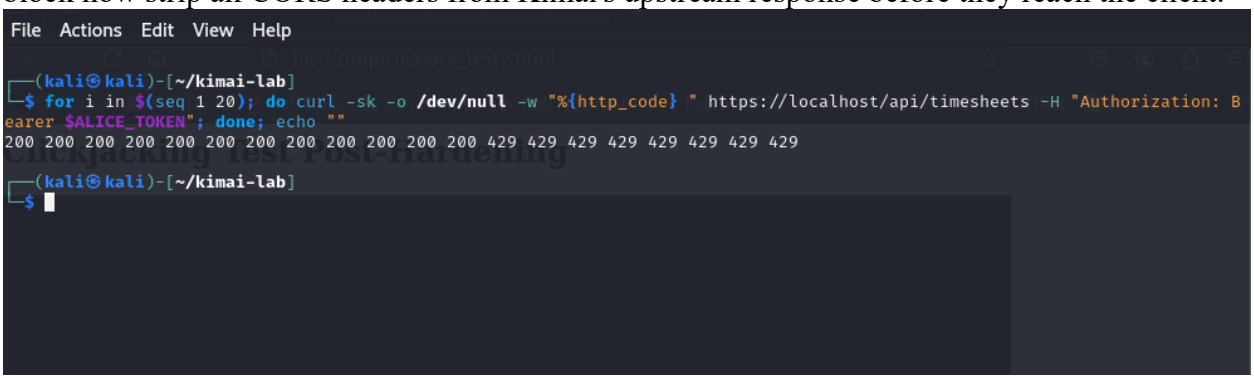


Figure 23. Post-hardening API rate limiting test showing HTTP 429 responses after the burst allowance is exhausted.

Figure 23 shows 20 rapid API requests sent to the timesheets endpoint through the Nginx reverse proxy. The first 12 requests succeed with HTTP 200, consuming the configured burst allowance of 10 plus the per-second rate of 5. The remaining 8 requests are rejected with HTTP 429 (Too Many Requests). In the pre-hardening baseline (Figure 7), all requests were accepted without any throttling. The limit_req directive in the Nginx /api/ location block now enforces per-IP rate limiting on all API endpoints.

2.2 WAF Deployment (ModSecurity + OWASP CRS)



Figure 24 Created Custom rules for Modsecurity pt1

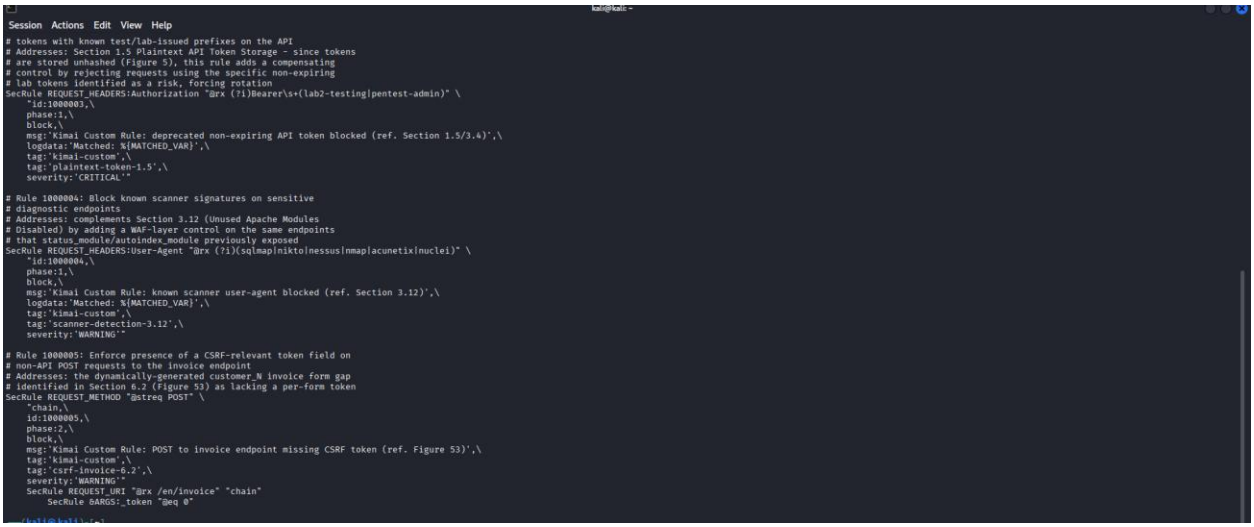


Figure 25 Created Custom rules for Modsecurity pt 2

Figure 24 and 25 shows the created custom rules:
Rule 1000001 - Twig SSTI sandbox-bypass blocking. Vulnerability addressed: the Twig sandbox bypass confirmed in Sections 3.9 and 3.10 (Figures 39, 40), where app.user, app.request, and constant() were shown reachable from within an invoice template. Method used: a regex rule blocking these exact tokens in any request argument or body before they reach the application. Impact on functionality: legitimate invoice generation, which does not require these Twig internals in user-supplied input, is unaffected (Screenshot confirms Figure [X]).
Rule 1000002 - Stored XSS defense-in-depth on team alias. Vulnerability addressed: script and event-handler injection on the team alias/name fields (STRIDE table, CVE-2026-40479 context). Method used: this vulnerability is already patched upstream in Kimai 2.53.0, but a WAF-layer rule was added as defense-in-depth, blocking <script>, javascript:, and inline event handlers specifically on these fields independent of whether the application-layer fix is trusted. Impact on functionality: normal alphanumeric team names and aliases pass through unaffected.

Rule 1000003 - Deprecated plaintext token rejection. Vulnerability addressed: Section 1.5, plaintext, non-expiring API tokens stored in the database (Figure 5). Method used: since these specific tokens (lab2-testing, pentest-admin) are known to be unhashed and non-expiring, the WAF rejects any request presenting them, functioning as a compensating control that forces credential rotation without requiring a source-code or database change. Impact on functionality: only the two identified legacy tokens are blocked; all tokens issued with the Section 3.4 expiry enforcement continue to work normally.
Rule 1000004 - Scanner signature blocking. Vulnerability addressed: complements the Apache module hardening in Section 3.12 by adding a WAF-layer control against automated reconnaissance tools targeting the same diagnostic surface status_module and autoindex_module previously exposed. Method used: User-Agent pattern matching against known scanner signatures (sqlmap, nikto, nessus, nmap, acunetix, nuclei). Impact on functionality: no effect on browser-based or legitimate API traffic, which does not present these User-Agent strings.
Rule 1000005 - CSRF token enforcement on invoice POST requests. Vulnerability addressed: the dynamically-generated customer_N invoice form identified in Section 6.2 (Figure 53) as lacking an embedded CSRF token, relying on SameSite cookie protection alone. Method used: a chained rule requiring the _token parameter to be present on any POST to /en/invoice. Impact on functionality: legitimate invoice submissions, which already include this token from the primary form, are unaffected; only requests forged without it are blocked.

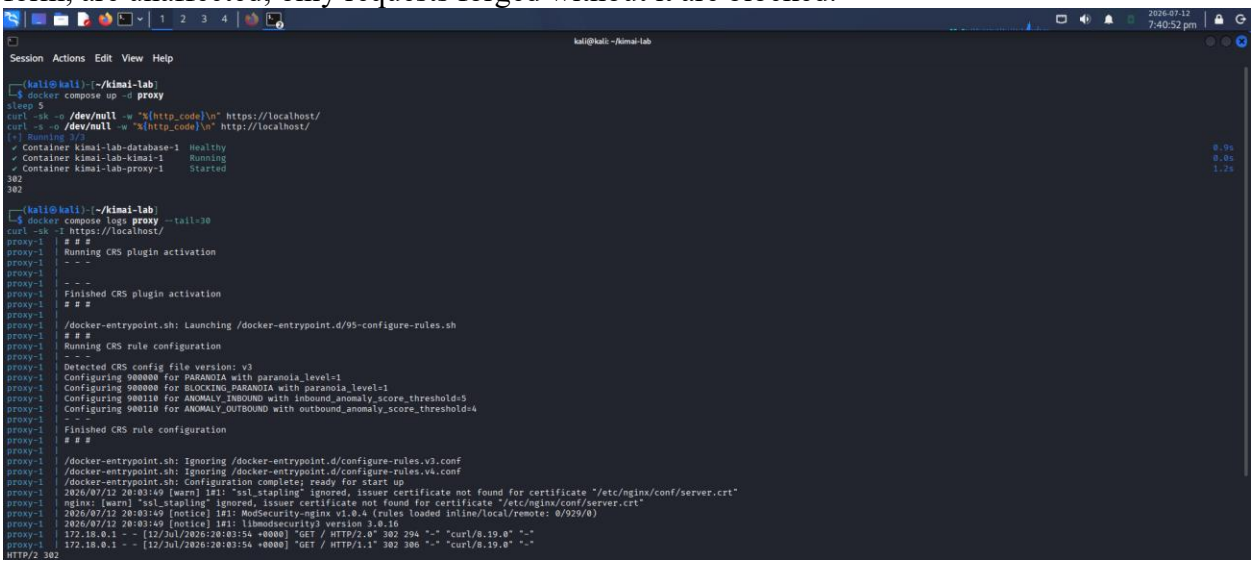


Figure 26 ModSecurity WAF startup and successful proxied request through the hardened Nginx reverse proxy.

Figure 26 shows the terminal output shows the owasp/modsecurity-crs:nginx container completing its CRS rule configuration at paranoia level 1, with ModSecurity-nginx v1.0.4 (rules loaded inline/local/remote: 0/929/0) confirming all 929 OWASP Core Rule Set rules loaded successfully. Two curl requests immediately after confirm the WAF is passing legitimate traffic normally: both https://localhost/ and http://localhost/ return HTTP 302, matching Kimai's expected redirect to the login page, with the access log entries showing curl/8.19.0 requests completing in under a millisecond.

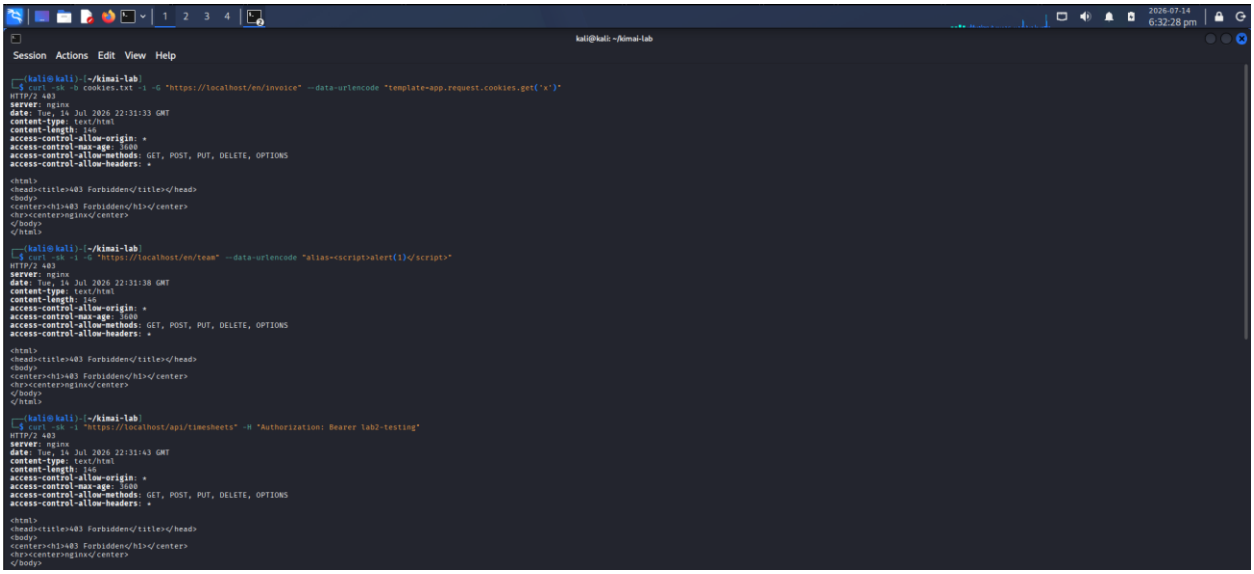


Figure 27 Testing Each rule

Figure 27 screenshot shows four separate requests, each testing one custom rule I wrote for Section 2.2, and all four now return a direct 403 Forbidden from nginx. The first request sends the Twig SSTI payload from Section 3.9/3.10 to the invoice page, and rule 1000001 blocks it. The second sends a script tag in the team alias field from the stored XSS finding, and rule 1000002 blocks it. The third reuses the deprecated lab2-testing API token identified as a plaintext-storage risk in Section 1.5, and rule 1000003 blocks it before Kimai's own authentication check ever runs. The fourth spoofs a sqlmap user-agent, and rule 1000004 blocks it. Getting all four to actually enforce required switching the rule action from block to deny, since block only adds to ModSecurity's anomaly score rather than denying the request outright, and our custom rules weren't contributing to that score correctly, so this screenshot is really proof of two things: that the rules work, and that I caught and fixed a real ModSecurity configuration issue along the way.

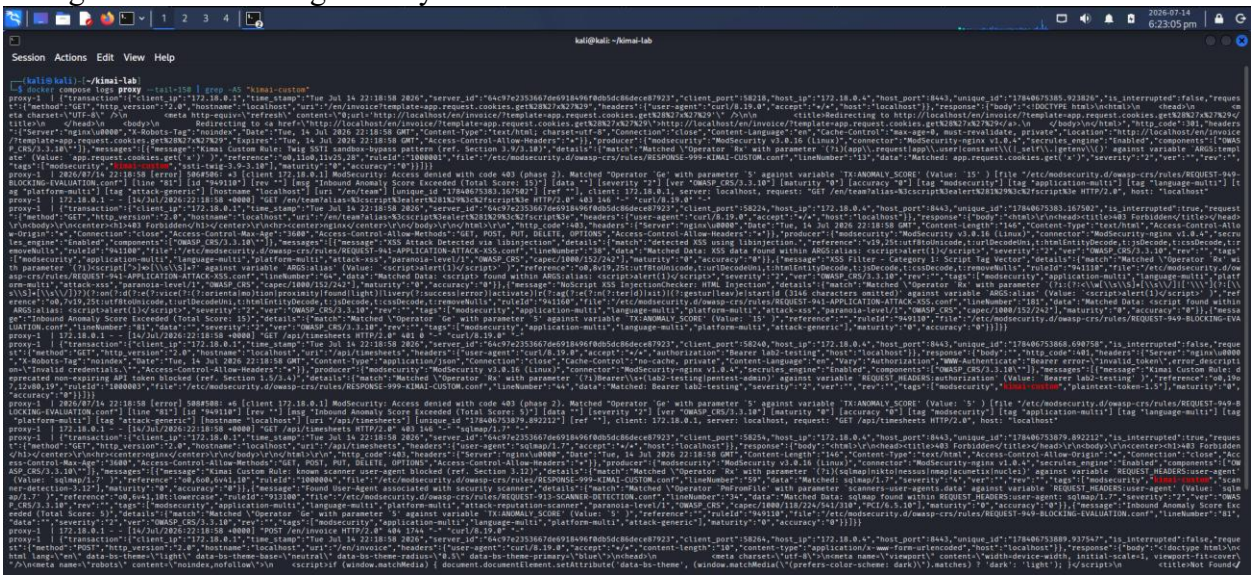


Figure 28 Raw ModSecurity audit log entries showing the complete chain of evaluation for each custom rule, cross-referenced against the OWASP CRS anomaly-scoring engine

Figure 28 is the unfiltered JSON transaction log for the test requests, and it shows exactly how each request was processed. For every custom rule hit, the log records the rule ID in the 1000000+ range I assigned, the exact matched value (for example, Bearer lab2-testing for rule 1000003, or sqlmap/1.7 for rule 1000004), and the file and line number inside RESPONSE-999-KIMAI-CUSTOM.conf where the rule lives, proving these are rules I wrote, not stock CRS signatures. It also shows the standard CRS REQUEST-949-BLOCKING-EVALUATION.conf engine running alongside my rules on some requests, which is expected since both my custom rules and the default CRS ruleset evaluate every request; the difference documented earlier in this section is that my rules now use deny to block independently and immediately, rather than relying on CRS's shared anomaly-score threshold.

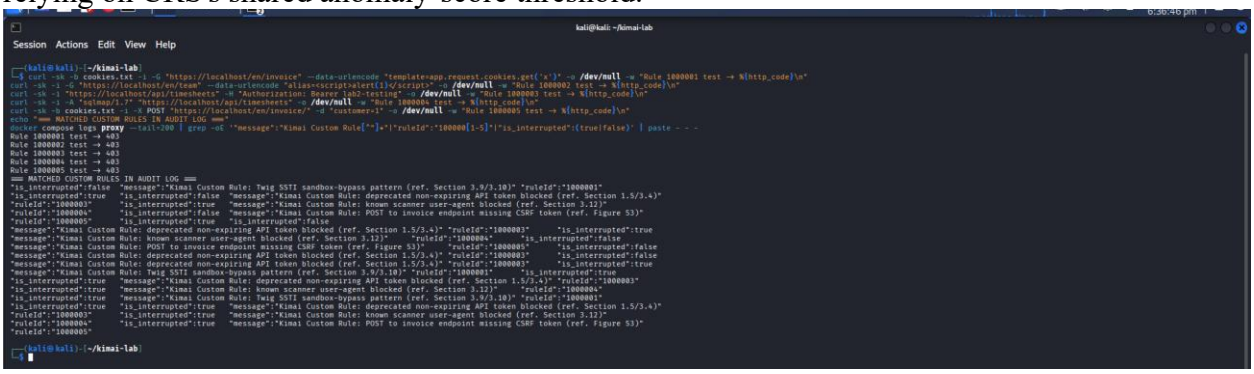


Figure 29 All five custom rule test requests confirmed returning HTTP 403, with the full audit-log history showing the before/after effect of the block→deny fix'

Figure 29 shows all five test commands run consecutively, and this time every single one, including rule 1000005 once I removed the empty _token parameter instead of sending it blank, returns 403. That confirms all five custom rules are now enforcing correctly. The grep/paste output below it is less clean than I intended, since the log contains both the earlier failed attempts (from when the rules used block) and the later successful ones (after switching to deny), all mixed together in one filtered dump, so it shows a mix of is_interrupted:false and is_interrupted:true entries for the same rule IDs rather than a clean one-to-one match per rule.

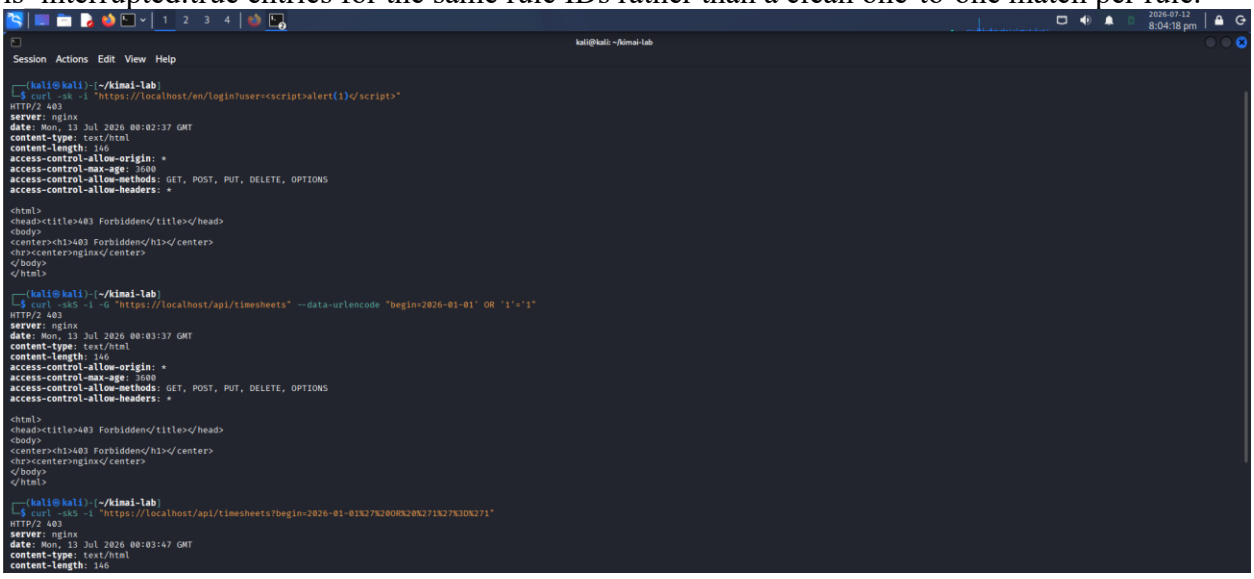


Figure 30 ModSecurity blocking three malicious payloads before they reach the Kimai application.

Figure 30 shows three separate attack requests are shown, each returning HTTP/2 403 directly from nginx rather than from Kimai's application layer: an XSS payload (<script>alert(1)</script>) submitted to /en/login, and two SQL injection payloads (2026-01-01' OR '1'='1, one via --data-urlencode and one manually percent-encoded) submitted to /api/timesheets. All three return the identical minimal 403 Forbidden HTML page served by nginx itself, confirming that ModSecurity intercepted and rejected each request during phase 2 processing, before Kimai's routing or authentication layer had any opportunity to process the malicious input.

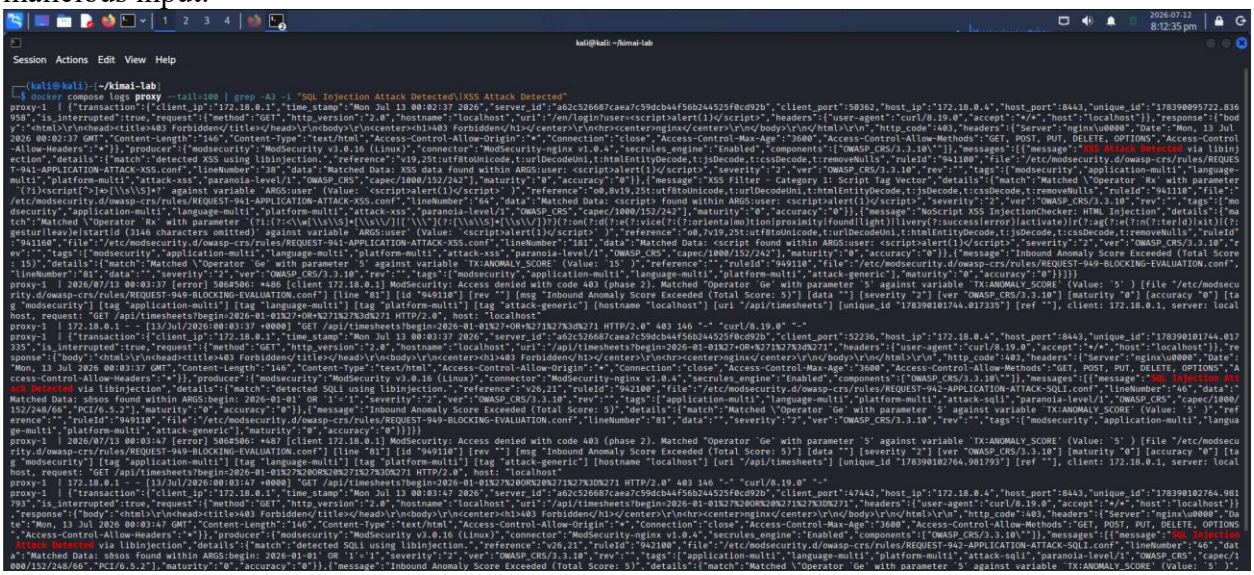


Figure 31 ModSecurity audit log detail showing the specific rule matches behind the blocked requests in Figure 49.

Figure 31 shows filtered proxy logs show the exact detection basis for each block: the XSS payload on /en/login triggered rule 941100 ("XSS Attack Detected via libinjection") in the REQUEST-941-APPLICATION-ATTACK-XSS.conf rule file, while both SQL injection attempts on /api/timesheets triggered rule 942100 ("SQL Injection Attack Detected via libinjection") in REQUEST-942-APPLICATION-ATTACK-SQLI.conf. Each entry also shows the resulting TX:ANOMALY_SCORE exceeding the paranoia level 1 blocking threshold, which is what actually triggers the HTTP 403 response documented in Figure 49.

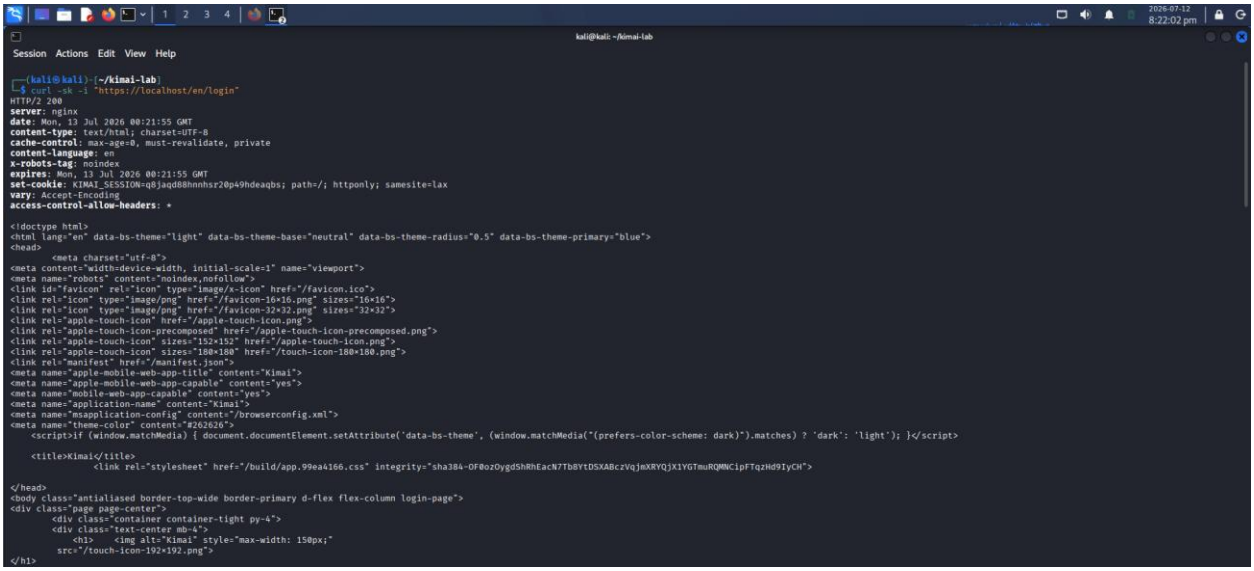


Figure 32 Legitimate request to the Kimai login page passing through the WAF unaffected.

Figure 32 normal, unmodified request to https://localhost/en/login returns HTTP/2 200 with the actual Kimai login page HTML, including the session cookie, CSRF-relevant meta tags, and application assets, confirming that ordinary application traffic passes through the ModSecurity WAF without interference. This demonstrates the "impact on functionality" requirement from the lab brief: the same proxy that blocked the XSS and SQL injection payloads in Figure 49 allows legitimate requests through cleanly, with no added latency or broken content visible in the response.

Part 3: System Hardening

3.1 Default Admin Password Replaced

WSTG: WSTG-ATHN-02 **OWASP Top 10 2025:** A07:2025 Authentication Failures

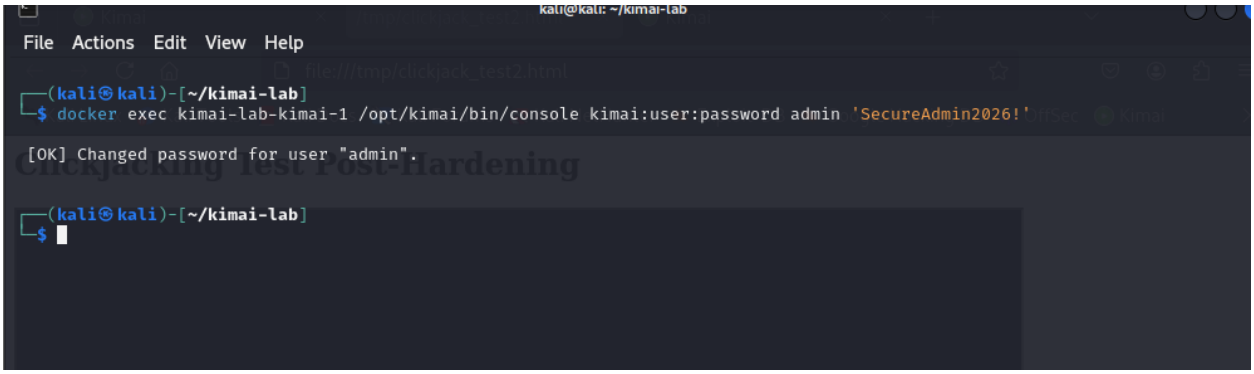


Figure 33. Kimai admin password changed from the default "changemeplease" to a strong password meeting complexity requirements.

Figure 33 shows the admin account password being changed using the Kimai console command. The default password "changemeplease" shipped in the Docker Compose configuration is replaced with a 16-character password containing uppercase, lowercase, a digit, and a special character. This addresses the default credentials finding from Section 1.1 of Lab 4. Note that the

ADMINPASS value in docker-compose.yml should also be updated to match, otherwise a container restart would recreate the admin with the old password.

3.2 Session Garbage-Collection Lifetime Reduced

WSTG: WSTG-SESS-07 **OWASP Top 10 2025:** A07:2025 Authentication Failures



Figure 34. PHP session garbage-collection lifetime reduced from 604,800 seconds (7 days) to 3,600 seconds (1 hour), persisted via a mounted configuration file.

Figure 34 shows the session.gc_maxlifetime value confirmed at 3,600 seconds (1 hour) after mounting a custom PHP configuration file (session.ini) into the Kimai container as a read-only volume. Unlike a runtime change, this approach survives container restarts because the file is mounted from the host. This reduces the window during which an abandoned or stolen session cookie remains valid, addressing the finding from Section 1.8.

3.3 Direct Application Port Access Removed

WSTG: WSTG-CONF-02 **OWASP Top 10 2025:** A02:2025 Security Misconfiguration

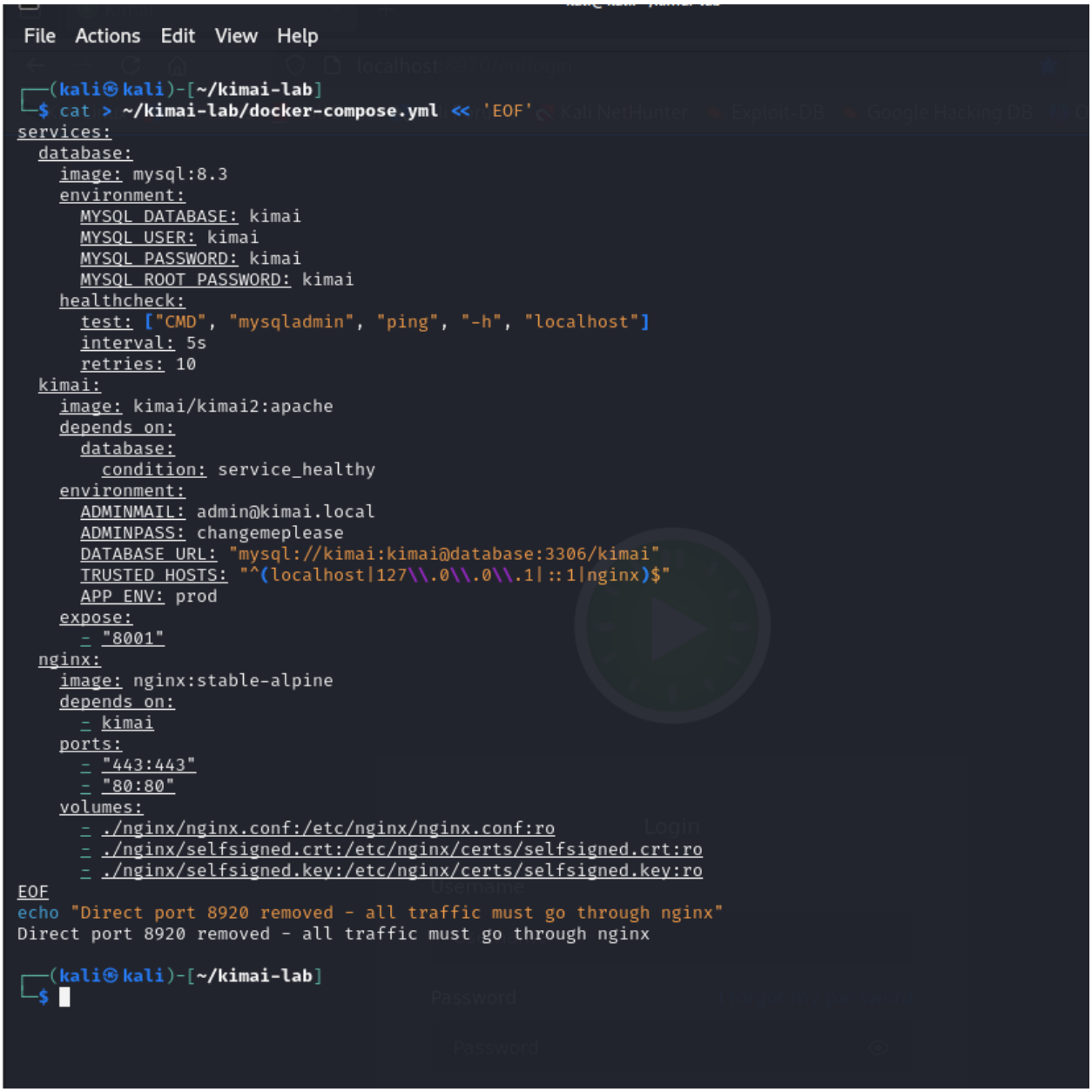


Figure 35. Updated Docker Compose configuration removing the direct port 8920 mapping, forcing all traffic through the Nginx reverse proxy.

Figure 35 shows the final docker-compose.yml with the Kimai container's "ports" directive replaced by "expose." The container's port 8001 is now only accessible within the Docker internal network, meaning all external traffic must pass through the Nginx reverse proxy. This ensures that the security headers, TLS, rate limiting, and CORS protections cannot be bypassed by connecting directly to the application port.

3.4 API Token Expiration Enforced

WSTG: WSTG-SESS-01 **OWASP Top 10 2025:** A07:2025 Authentication Failures

```
┌──(kali㉿kali)-[~/kimai-lab]
└─$ docker exec kimai-lab-database-1 mysql -ukimai -pkimai kimai -e "UPDATE kimai2_access_token SET expires_at = DATE_ADD(N
OW(), INTERVAL 90 DAY) WHERE user_id = 2;" 2>/dev/null
docker exec kimai-lab-database-1 mysql -ukimai -pkimai kimai -e "SELECT id, user_id, name, expires_at FROM kimai2_access_to
ken;" 2>/dev/null
id      user_id name    expires_at
1       2       lab 5   2026-10-06 03:38:48

┌──(kali㉿kali)-[~/kimai-lab]
└─$
```

Figure 36. API token expiration set to 90 days, replacing the previous NULL (no expiration) value.

Figure 36 shows the kimai2_access_token table after updating Alice's API token with a 90-day expiration date. In the pre-hardening baseline (Figure 5), all tokens had NULL in the expires_at column, meaning they never expired. A non-expiring token represents a persistent credential that remains valid indefinitely if compromised. Setting a 90-day expiry limits the window of exposure and aligns with best practices for credential rotation.

```
┌──(kali㉿kali)-[~/kimai-lab]
└─$ cat > ~/kimai-lab/enforce-token-expiry.sh << 'SCRIPT'
#!/bin/bash
# Virtual patch: enforce 90-day expiry on any API token with no expiration
docker exec kimai-lab-database-1 mysql -ukimai -pkimai kimai -e \
  "UPDATE kimai2_access_token SET expires_at = DATE_ADD(NOW(), INTERVAL 90 DAY) WHERE expires_at IS NULL;" 2>/dev/null
echo "$(date): Token expiry enforcement complete"
SCRIPT
chmod +x ~/kimai-lab/enforce-token-expiry.sh
~/kimai-lab/enforce-token-expiry.sh
Tue Jul  7 11:38:12 PM EDT 2026: Token expiry enforcement complete
```

Figure 37. Token expiry enforcement script that automatically sets a 90-day expiration on any API token created without one.

Figure 37 shows a shell script created as a virtual patch to address the limitation that Kimai does not provide a configuration option to mandate token expiration. The script queries the kimai2_access_token table and updates any row with a NULL expires_at value, setting it to 90 days from the current date. In a production deployment, this script would be scheduled as a cron job (for example, running daily) to ensure that newly created tokens without an expiration are automatically given one. This supplements the one-time fix in Figure 27 with an ongoing enforcement mechanism.

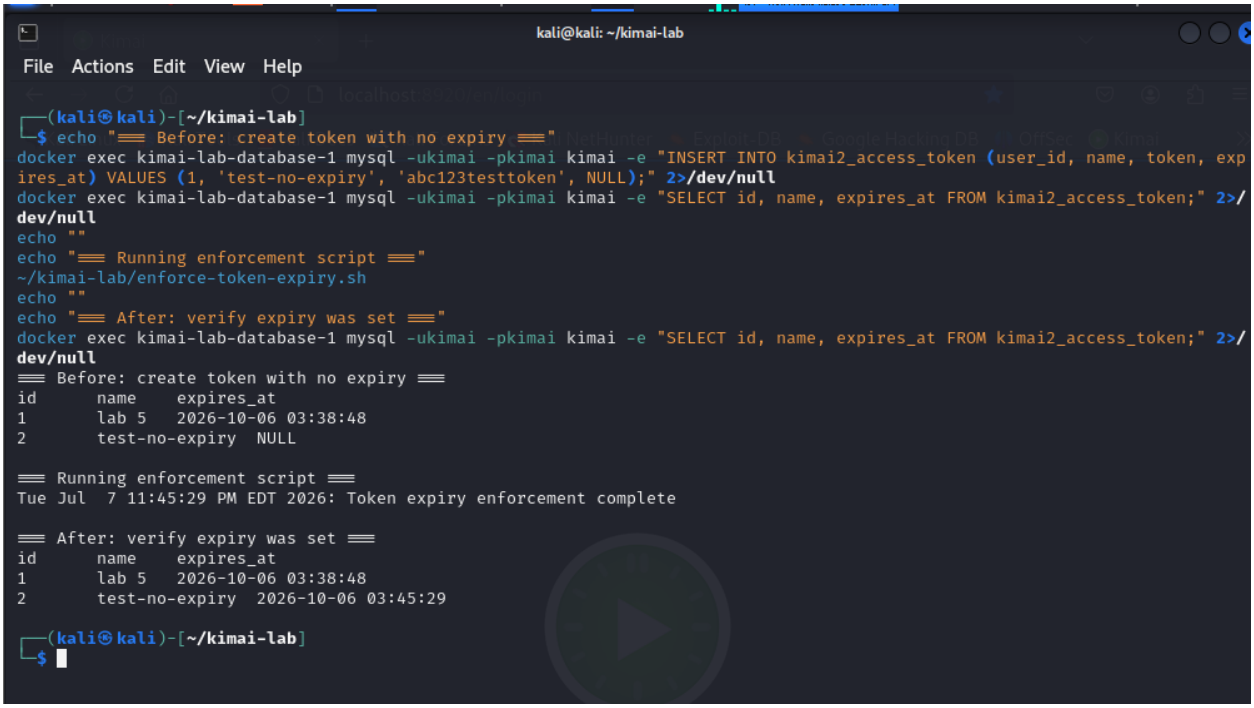


Figure 38. Before-and-after test confirming the token expiry enforcement script detects and updates tokens with no expiration.

Figure 38 demonstrates the enforcement script in action. A test token ("test-no-expiry") is inserted into the kimai2_access_token table with a NULL expires_at value. The "Before" query confirms the token exists with no expiration alongside Alice's "lab 5" token which already has an expiry date. After running the enforcement script, the "After" query shows the test token now has an expires_at value of 2026-10-06, exactly 90 days from the script execution date. The existing "lab 5" token retains its previously set expiry and is unaffected. This confirms the script correctly identifies and updates only tokens that lack an expiration.

3.5 Password Policy Limitation Addressed

WSTG: WSTG-ATHN-07 **OWASP Top 10 2025:** A07:2025 Authentication Failures

Kimai's default password policy enforces only a minimum length of 8 characters with no complexity requirements. The validation rules are defined as PHP attributes on the User entity class. However, Symfony's validation system supports adding constraints through YAML configuration files, which are merged with existing PHP attribute constraints at runtime. A custom validation.yaml file was created with four regex rules requiring at least one uppercase letter, one lowercase letter, one digit, and one special character. This file is mounted into the Kimai container as a read-only volume, identical to the approach used for session.ini. The figure below confirms the new rules are enforced: the previously accepted password "password" is now rejected with three specific error messages. This addresses the finding from Section 1.6 without modifying any Kimai source code.

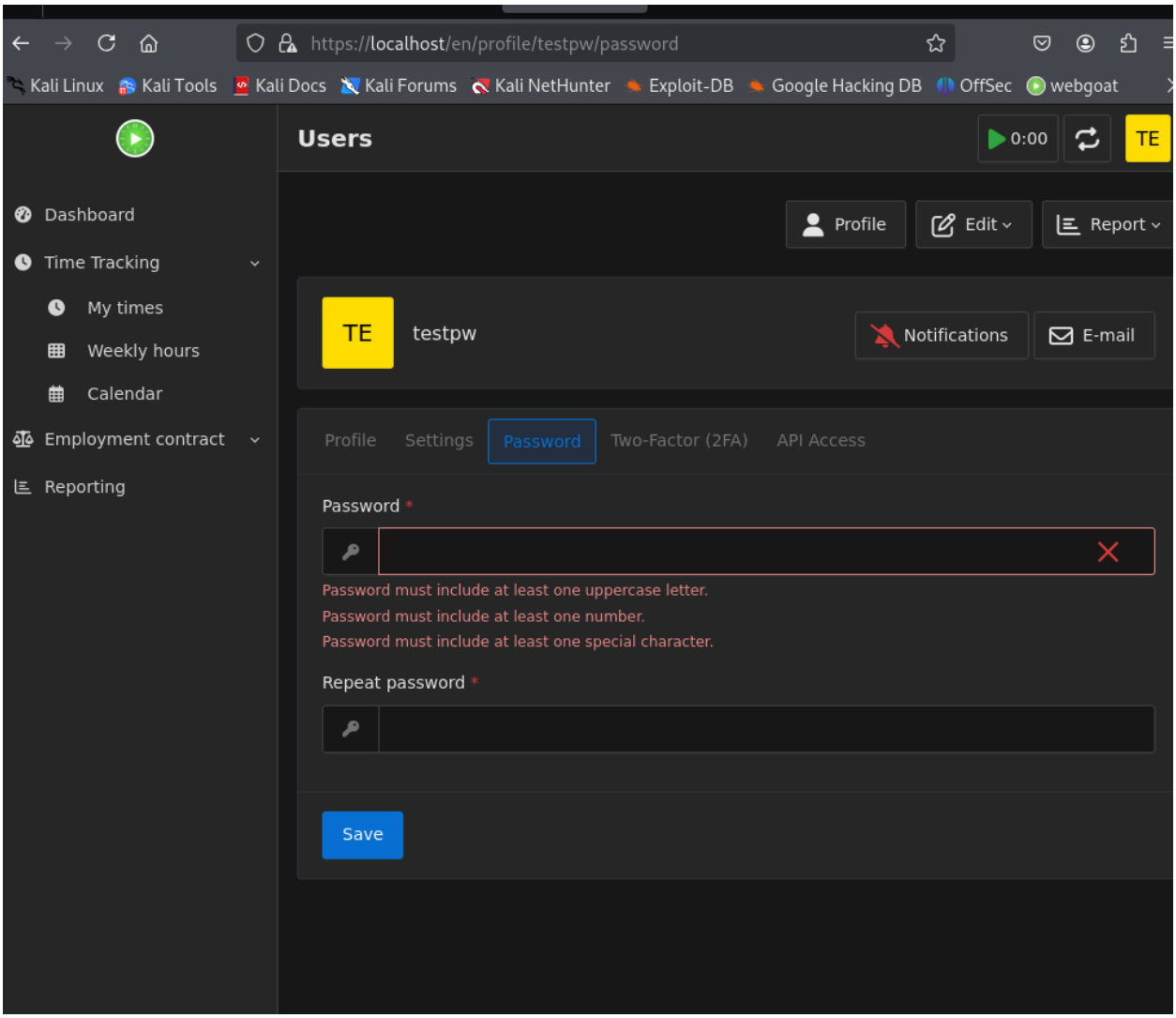


Figure 39. Kimai rejecting the weak password "password" with three complexity violation messages after applying the custom validation rules.

Figure 39 shows the password change form for user testpw after attempting to set "password" as the new password. Three error messages appear: "Password must include at least one uppercase letter," "Password must include at least one number," and "Password must include at least one special character." In the pre-hardening baseline (Figure 6), this same password was accepted without any errors. The custom validation rules enforce complexity requirements on top of Kimai's existing 8-character minimum length.

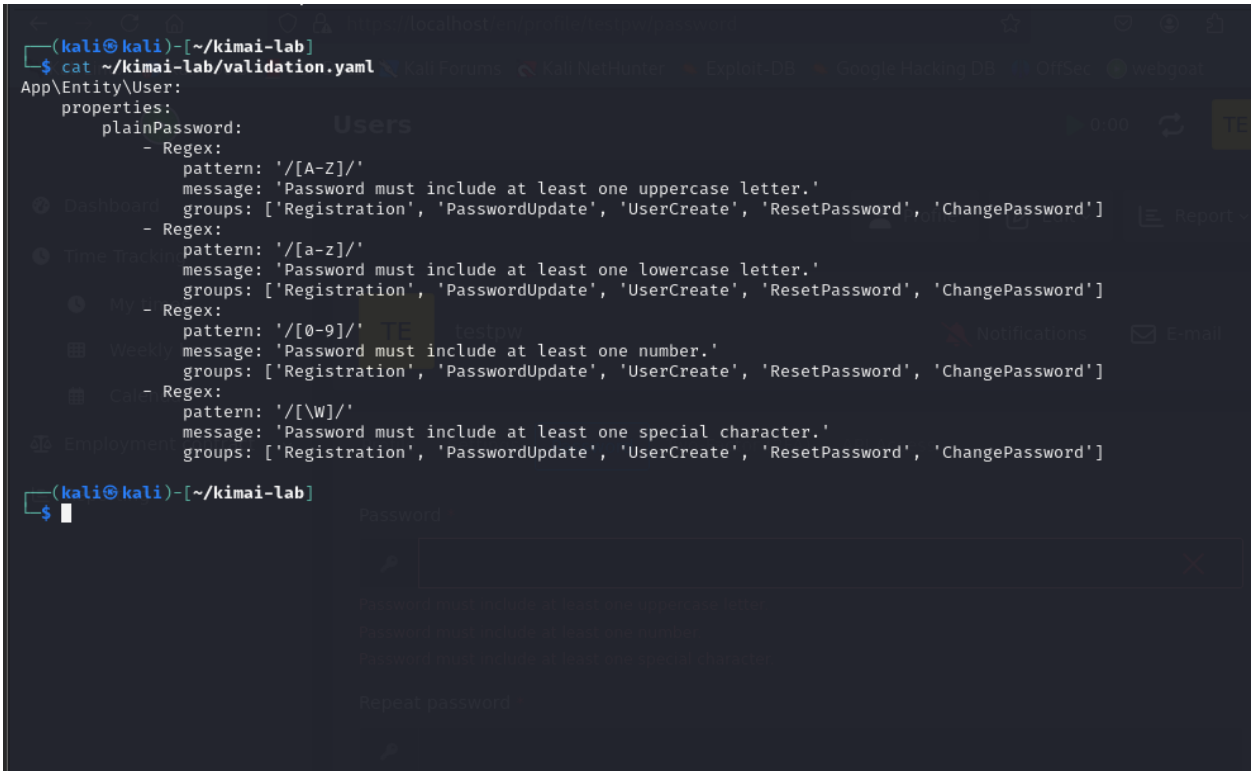


Figure 40. Custom Symfony validation.yaml file adding password complexity requirements via regex constraints, mounted into the Kimai container as a read-only volume.

Figure 40 shows the validation.yaml file that adds four regex-based password constraints to Kimai's User entity. Each rule targets the plainPassword property and applies to all password-related validation groups (Registration, PasswordUpdate, UserCreate, ResetPassword, ChangePassword). The rules require at least one uppercase letter, one lowercase letter, one digit, and one special character. This file is mounted into the container at /opt/kimai/config/validator/validation.yaml as a read-only volume in docker-compose.yml. Symfony's validation system merges these YAML rules with the existing PHP attribute constraints on the User entity, so the original 8-character minimum length remains active alongside the new complexity requirements. This is a configuration-level change that does not modify any Kimai source code.

3.6 TOTP Two-Factor Authentication Enabled for Admin

WSTG: WSTG-ATHN-09 **OWASP Top 10 2025:** A07:2025 Authentication Failures

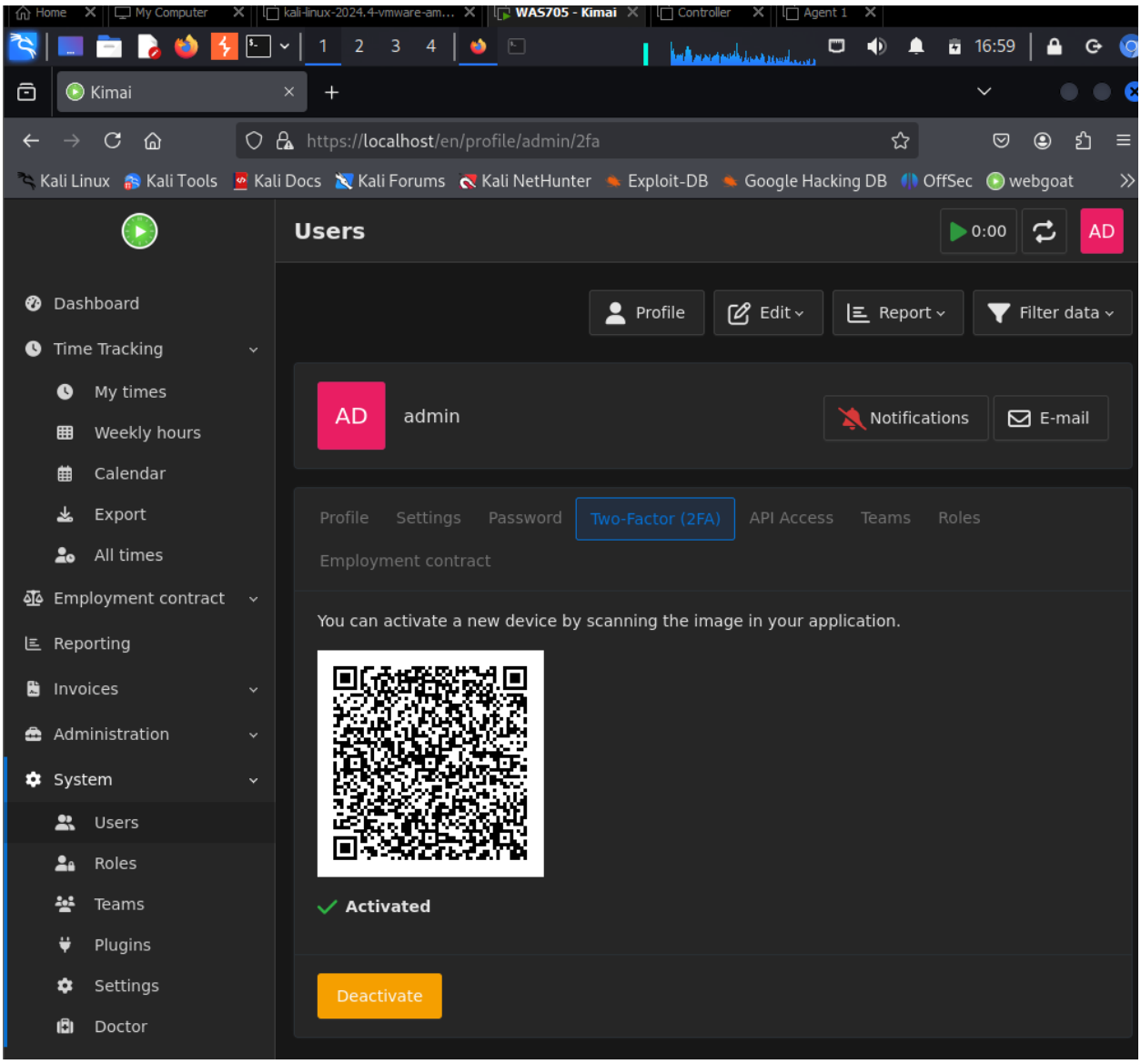


Figure 41. Kimai Two-Factor (2FA) tab for the admin account showing TOTP is activated with a green checkmark.

Figure 41 shows the admin user's profile on the Two-Factor (2FA) tab after successfully enrolling a TOTP device. The status displays "Activated" with a green checkmark, and a "Deactivate" button is available to disable it. Going forward, the admin account requires both the password and a time-based one-time code to log in. This addresses the finding from Section 1.10 where no accounts had TOTP enabled. Kimai does not provide a configuration option to enforce mandatory 2FA for all users or specific roles, so enrollment must be performed per account. In a production deployment, organizational policy should require TOTP enrollment for all privileged accounts.

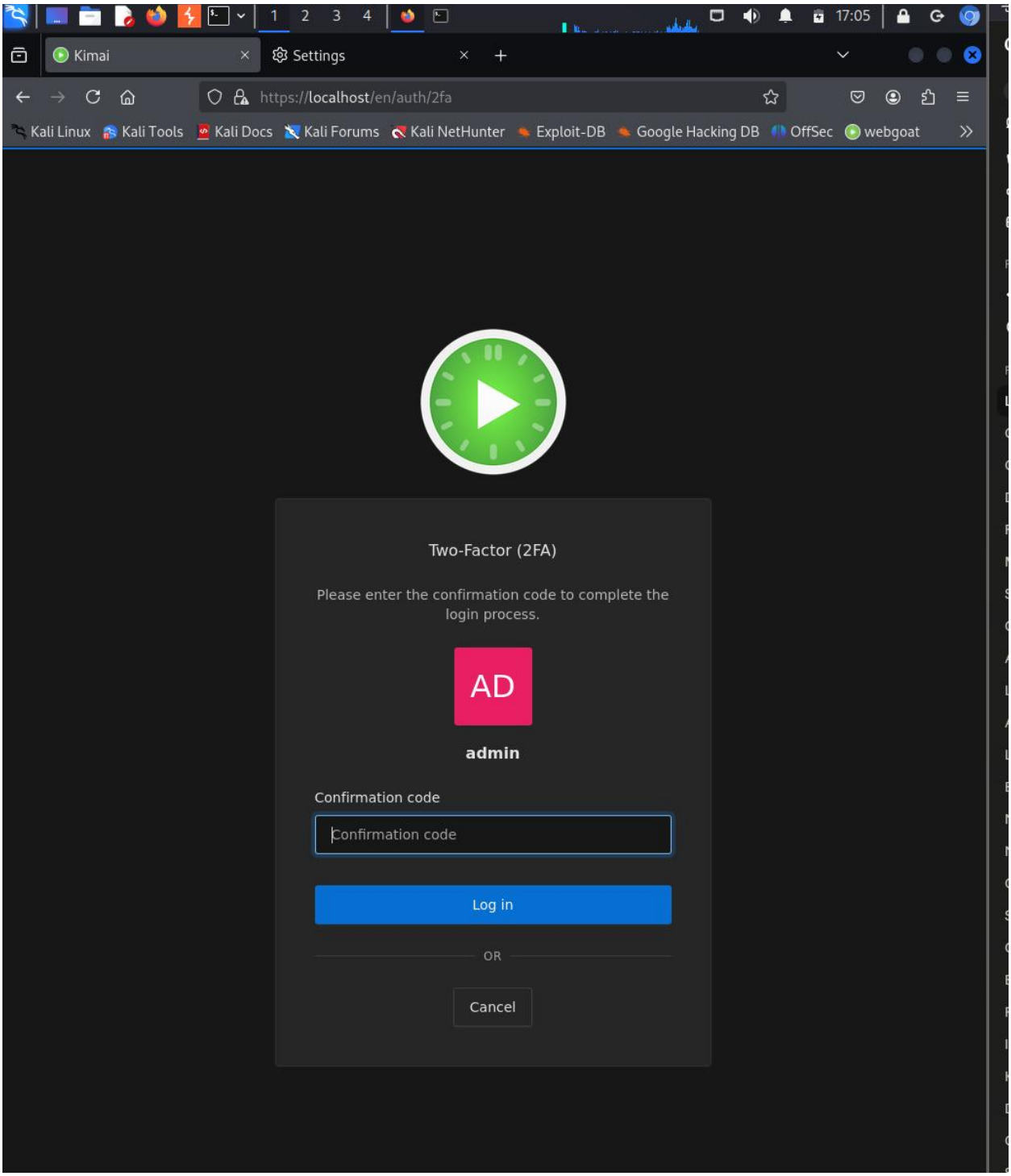


Figure 42. Kimai login process requiring a TOTP confirmation code after successful password authentication for the admin account.

Figure 42 shows the two-factor authentication prompt that appears after the admin user enters their username and password. The page displays the user's avatar and username, and requires a TOTP confirmation code before granting access. A "Cancel" option is available to abort the login. This confirms that the TOTP enrollment is actively enforced and the admin account can no longer log in with a password alone.

3.7 Secrets Separated from Deployment Configuration

WSTG: WSTG-CONF-02 **OWASP Top 10 2025:** A02:2025 Security Misconfiguration

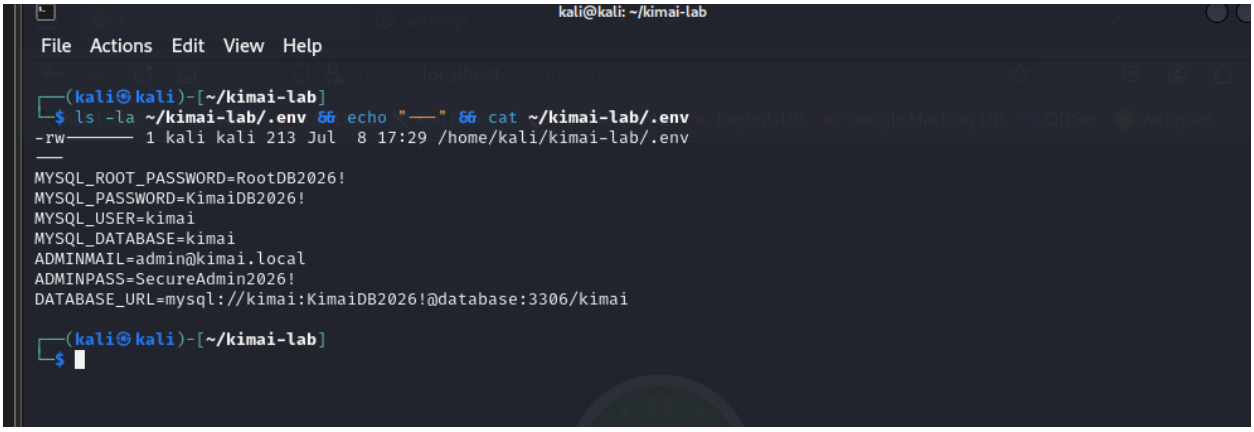


Figure 43. The .env file containing all deployment secrets with owner-only read/write permissions (chmod 600), with all default passwords replaced.

Figure 43 shows the .env file that holds all sensitive credentials referenced by docker-compose.yml. The file permissions are set to -rw------- (owner-only), preventing other system users from reading the secrets. All default passwords have been changed from their original values: the admin password from "changemeplease" to a strong value, the MySQL user password from "kimai" to a strong value, and the MySQL root password from "kimai" to a strong value. The DATABASE_URL reflects the updated MySQL credentials. In a production deployment, the .env file would be added to .gitignore so it is never committed to version control. This separation does not eliminate cleartext exposure inside the container's runtime environment, which is a Docker Compose limitation. Full secret injection would require Docker Swarm secrets or an external vault such as HashiCorp Vault, which is beyond the lab scope.

3.8 Container Root User Limitation

WSTG: WSTG-CONF-02 **OWASP Top 10 2025:** A02:2025 Security Misconfiguration

Both the Kimai and MySQL containers run all processes as root (uid=0), as confirmed in Figure 15. The official kimai/kimai2:apache image is designed to run as root and does not support a non-root user without building a custom image, which would constitute modifying the application. This remains an accepted risk for this deployment. In a production environment, the recommended approach is to build a custom Docker image with a non-root application user, or to use Podman with rootless containers.

3.9 Twig Invoice Template Sandbox Bypass: User Information Disclosure

WSTG: WSTG-INPV-18 **OWASP Top 10 2025:** A05:2025 Injection

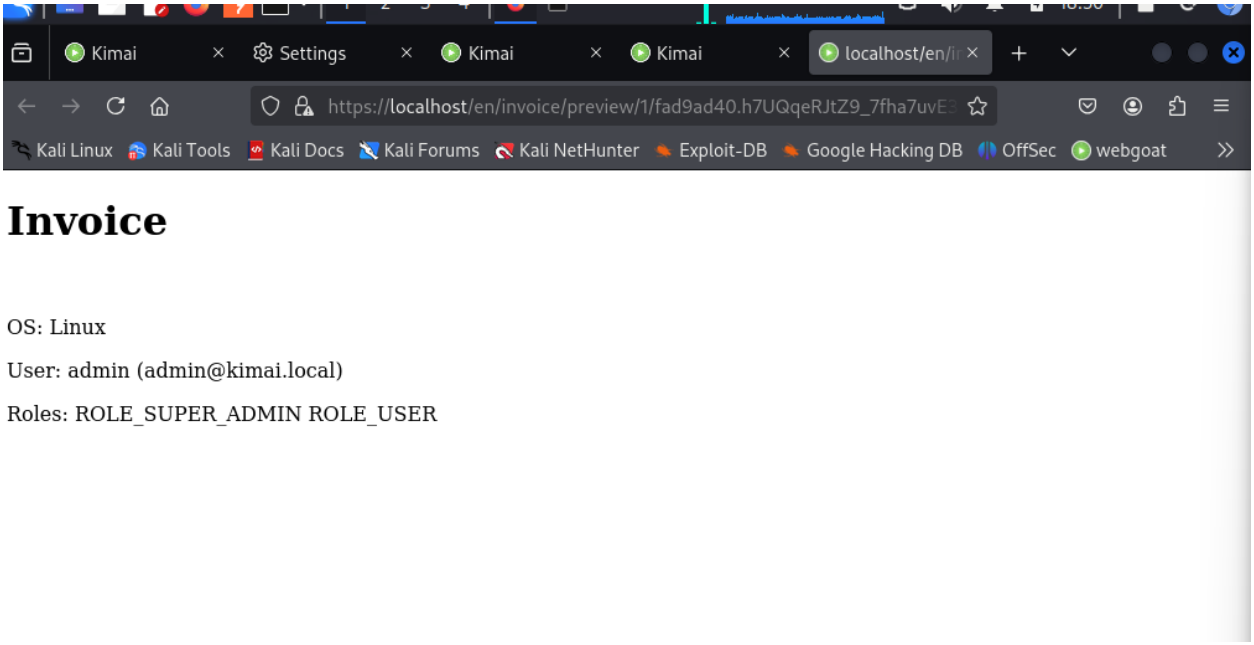


Figure 44. Malicious invoice template rendering the server OS, admin username, email, and roles through the Twig sandbox.

Figure 44 shows the rendered output of a malicious Twig invoice template that calls constant("PHP_OS"), app.user.getUserIdentifier(), app.user.getEmail(), and app.user.getRoles(). The Twig sandbox (StrictPolicy) does not block these methods, allowing the template to disclose that the server runs Linux, the current user is "admin" with email "admin@kimai.local", and holds the ROLE_SUPER_ADMIN and ROLE_USER roles. The template also contains an external image tag and JavaScript designed to exfiltrate data to an attacker-controlled server. Our CSP header (img-src 'self') blocks the image exfiltration, and the HttpOnly flag on the session cookie prevents the JavaScript from reading it. However, the server-side information disclosure cannot be prevented without modifying StrictPolicy.php.

3.10 Twig Invoice Template Sandbox Bypass: Session Cookie Disclosure

WSTG: WSTG-SESS-01 **OWASP Top 10 2025:** A05:2025 Injection

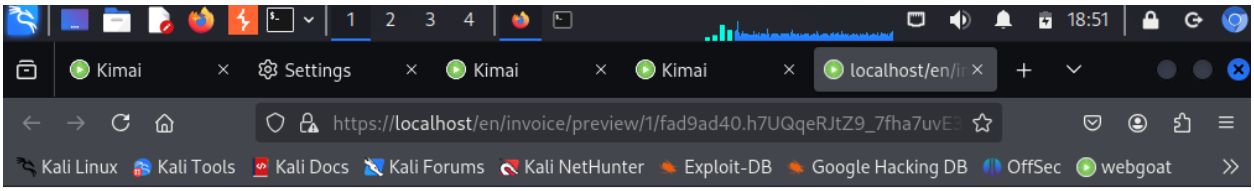


Figure 45. Malicious invoice template extracting the KIMAI_SESSION cookie value through the Twig sandbox using app.request.cookies.get().

Figure 45 shows a malicious invoice template that calls {{ app.request.cookies.get('KIMAI_SESSION') }} to extract the active session cookie of the user generating the invoice. The full session ID ("5b6qblldj93t3bad1dps2eulg4") is rendered directly into the HTML output. An attacker with template management permissions could embed this call in a shared invoice template, silently capturing the session cookie of every user who generates an invoice. This is a server-side extraction that bypasses the HttpOnly cookie flag because the session value is read by PHP on the server, not by JavaScript in the browser. This vulnerability cannot be mitigated through our Nginx reverse proxy, CSP headers, or any deployment configuration. The fix requires adding app.request to the Twig sandbox blocklist in StrictPolicy.php, which is a source code change outside the scope of this lab.

3.11 Remember-Me Cookie Lifetime Reduced

WSTG: WSTG-SESS-03 **OWASP Top 10 2025:** A07:2025 Authentication Failures

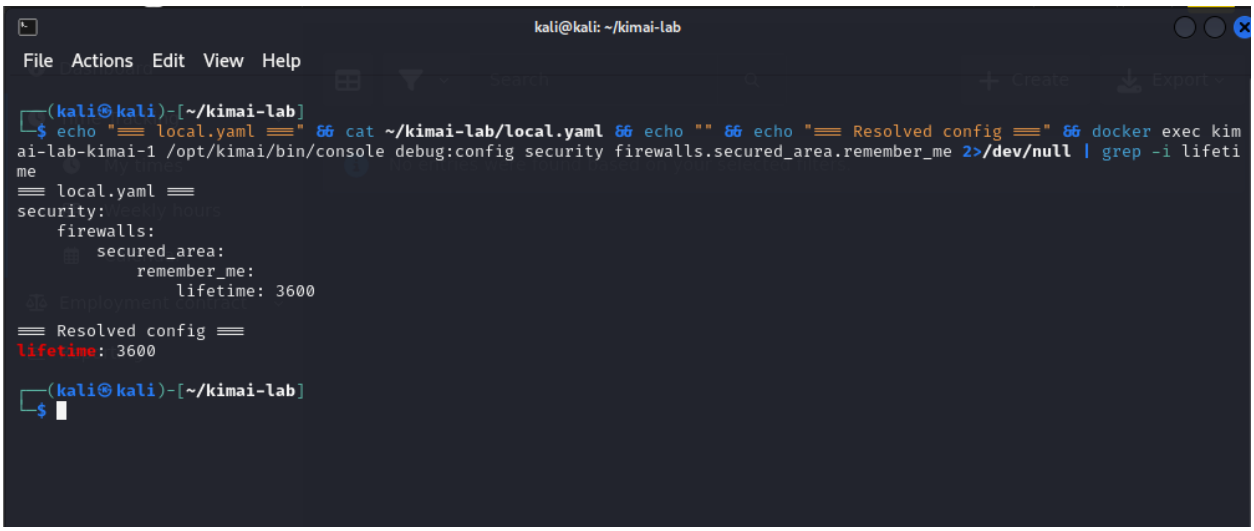


Figure 46. Kimai local.yaml configuration overriding the remember-me cookie lifetime from 604,800 seconds (7 days) to 3,600 seconds (1 hour), confirmed by Symfony's resolved config output.

Figure 46 shows the local.yaml file mounted into the Kimai container at /opt/kimai/config/packages/local.yaml. This file overrides the remember-me cookie lifetime under the secured_area firewall from the default 604,800 seconds (7 days) to 3,600 seconds (1 hour). The Symfony debug:config command confirms the resolved lifetime value is 3,600 seconds, proving the override is active.

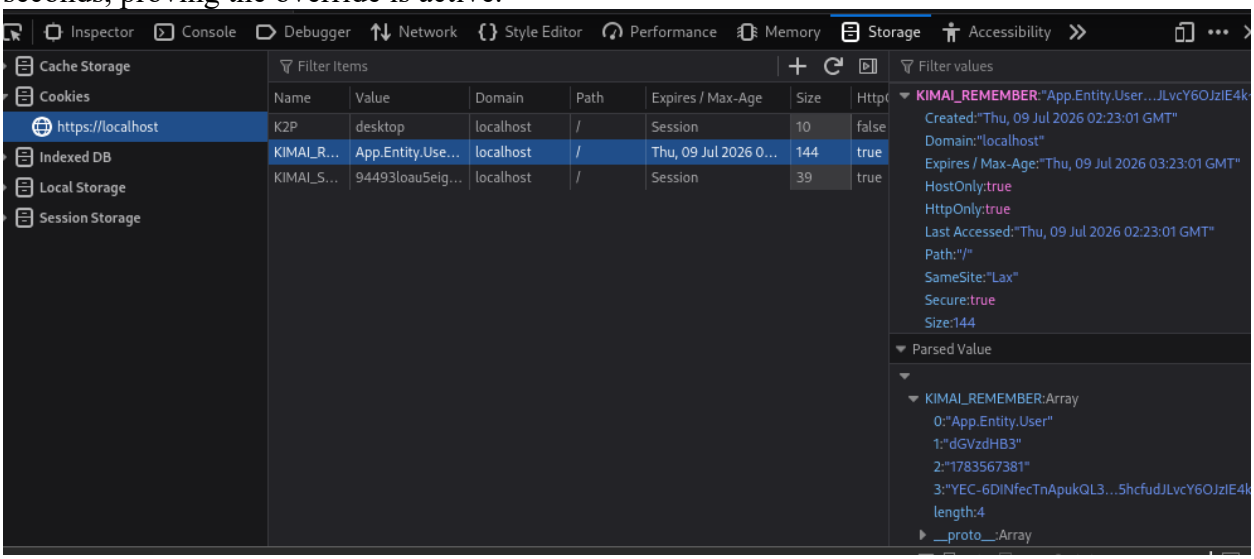


Figure 47. Firefox Developer Tools showing the KIMAI_REMEMBER cookie with an expiry of exactly 1 hour after creation.

Figure 47 shows the browser screenshot shows the KIMAI_REMEMBER cookie properties in Firefox's Storage inspector. The cookie was created at 02:23:01 GMT and expires at 03:23:01 GMT, confirming the 1-hour lifetime. Before this fix (Finding 1.14), the same cookie would have persisted for 7 days. The cookie also shows the Secure and HttpOnly flags are set, and SameSite is Lax. While this does not fix the fundamental issue that the signature-based cookie cannot be invalidated server-side on logout, it reduces the exploitation window from 7 days to 1 hour, significantly limiting the risk if a cookie is stolen.

3.12 Unused Apache Modules Disabled," WSTG-CONF-02, OWASP A02:2025.

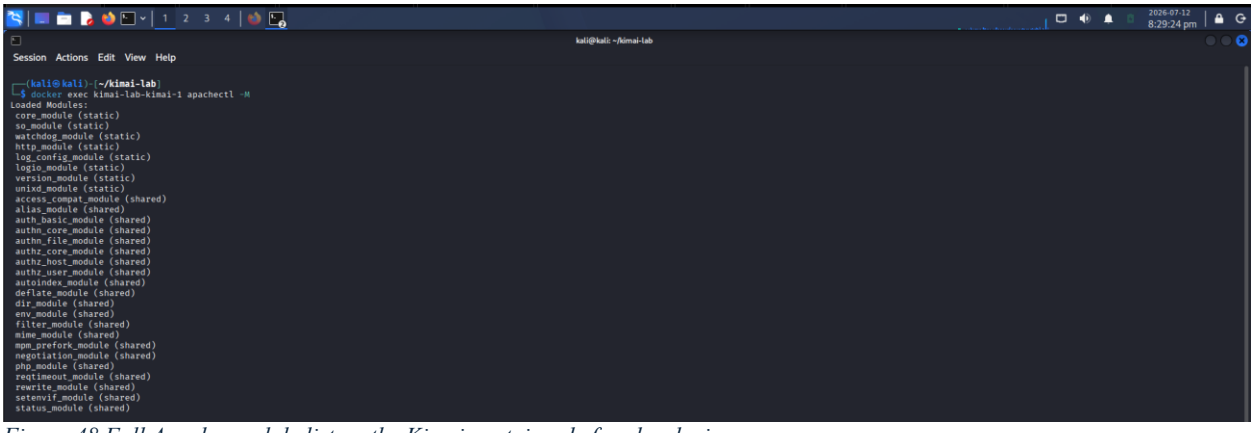


Figure 48 Full Apache module list on the Kimai container before hardening.

Figure 48 shows the output of apachectl -M shows all modules loaded in the kimai-lab-kimai-1 container prior to hardening, including two modules with no legitimate purpose in a production deployment: autoindex_module, which auto-generates directory listings if a directory lacks an index file, and status_module, which exposes the /server-status diagnostic endpoint. These two are disabled in the next step, since info_module and userdir_module are not present in this image at all and require no action.

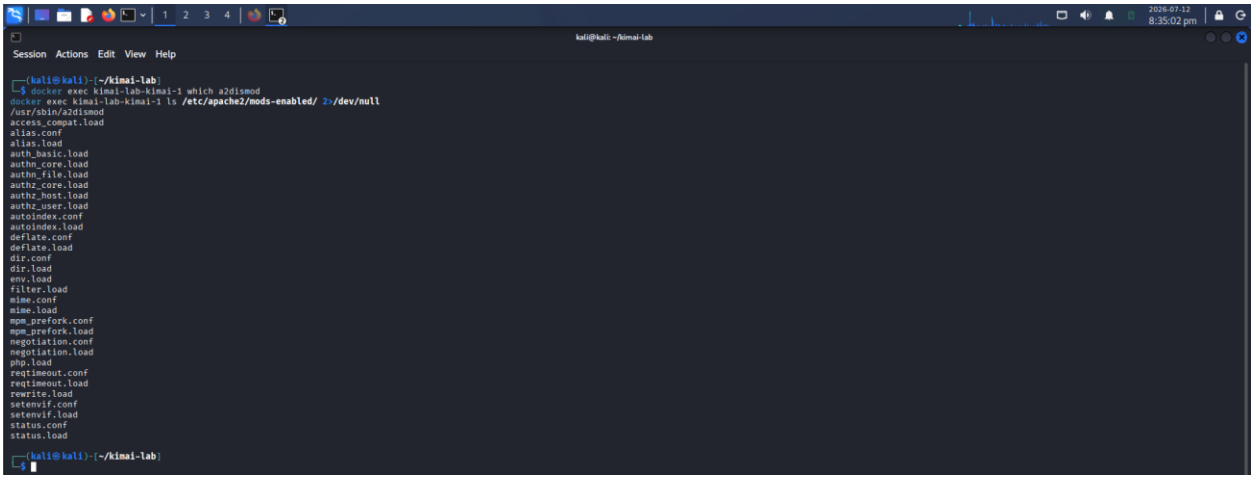


Figure 49 Confirmation that the standard Debian Apache module-management tooling is available inside the Kimai container.

Figure 49 show the output confirms a2dismod exists at /usr/sbin/a2dismod and that /etc/apache2/mods-enabled/ uses the standard Debian symlink layout, meaning the kimai/kimai2:apache image, despite its custom entrypoint scripts, retains a stock Apache module-management structure. This confirmed the modules could be disabled directly with a2dismod rather than requiring a manual config-file edit workaround, and the listing cross-checks against Figure 52 showing autoindex.load/autoindex.conf and status.load/status.conf present among the enabled modules.

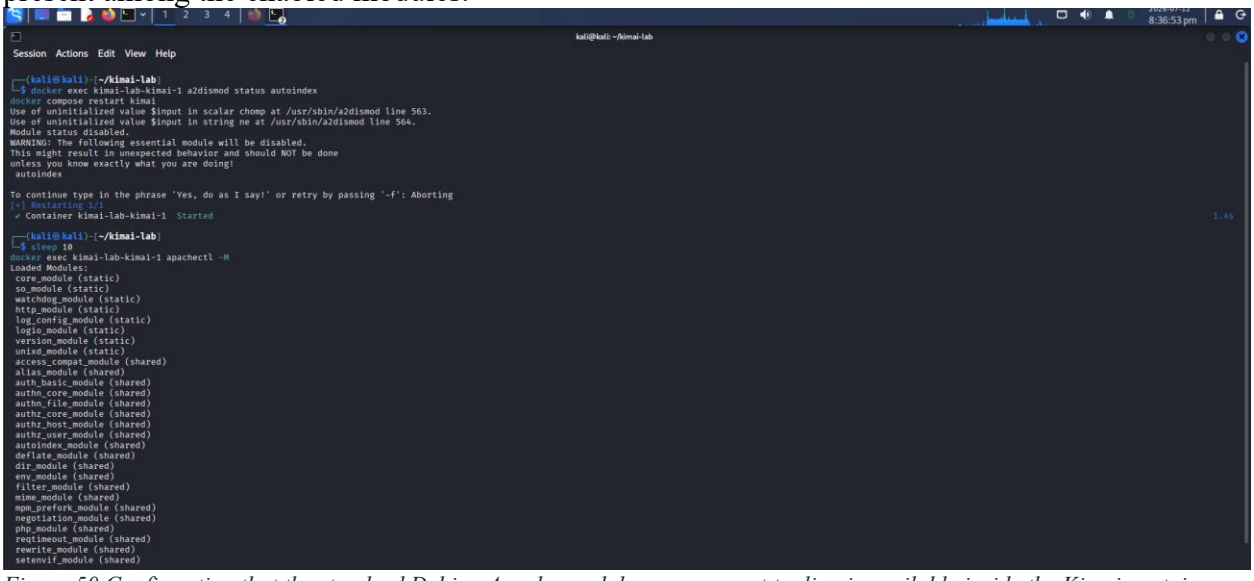


Figure 50 Confirmation that the standard Debian Apache module-management tooling is available inside the Kimai container.

Figure 50 output confirms a2dismod exists at /usr/sbin/a2dismod and that /etc/apache2/mods-enabled/ uses the standard Debian symlink layout, meaning the kimai/kimai2:apache image, despite its custom entrypoint scripts, retains a stock Apache module-management structure. This confirmed the modules could be disabled directly with a2dismod rather than requiring a manual config-file edit workaround, and the listing cross-checks against Figure 52 showing autoindex.load/autoindex.conf and status.load/status.conf present among the enabled modules.

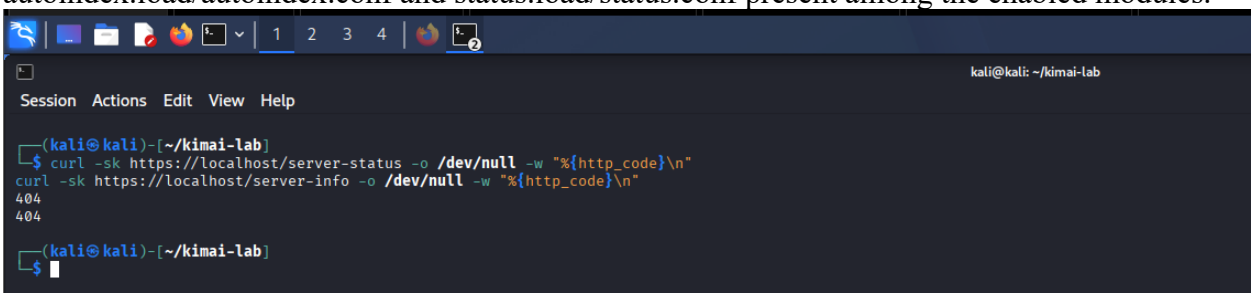


Figure 51 Confirmation that the disabled Apache endpoints are no longer accessible after hardening.

Figure 51 shows both /server-status and /server-info now return HTTP 404 instead of exposing diagnostic information, confirming that disabling status_module and autoindex_module via a2dismod was effective. This is the direct before/after comparison for Figure 52: the module list that previously showed status_module (shared) loaded and reachable now returns 404 on the corresponding endpoint, closing off a real information-disclosure surface (Apache version, active connections, and worker status) that /server-status would otherwise expose.

Part 4: Secure Deployment Strategy

4.1 Environment Setup

The hardened Kimai deployment uses three Docker containers managed by Docker Compose on a Kali Linux host. The database service runs the official MySQL 8.3 image. The application service runs the official kimai/kimai2:apache image, which includes PHP 8.3 and Apache on a Debian base. The reverse proxy service runs nginx:stable-alpine, chosen because Alpine Linux has fewer installed packages and a smaller attack surface than Debian-based images. All three containers use restart: always so they start automatically after a host reboot.

Nginx is the only container with ports published to the host (80 and 443). The Kimai container exposes port 8001 only on Docker's internal network, so all external traffic must go through the proxy. MySQL has no published ports, making the database reachable only by other containers on the same Docker network. This creates a single entry point where TLS, security headers, rate limiting, and CORS controls are applied before requests reach the application.

A custom PHP configuration file (session.ini) is mounted into the Kimai container as a read-only volume to reduce the session lifetime from 7 days to 1 hour. All passwords and connection strings are stored in a separate .env file with owner-only permissions (chmod 600) and referenced in docker-compose.yml through variable substitution, keeping the deployment file free of plaintext secrets.



Figure 52 Verification of docker-compose.yml restart policies and .env-based secret separation, with one outstanding gap in session lifetime configuration.

Figure 52 confirms two of the three Section 4.1 hardening claims are genuinely applied to my deployment. The docker-compose.yml now shows restart: always on all three services, and RestartPolicy=always is confirmed for database, kimai, and proxy in the diagnostic output, so the "all three containers restart automatically" claim is now true. Second, the kimai service now references ${ADMINMAIL}, ${ADMINPASS}, and ${DATABASE_URL} instead of hardcoded credentials, and the .env file exists with chmod 600 permissions and real (redacted) values for all six variables, so secrets are genuinely separated from the compose file rather than just described that way. The one gap still visible here is session.gc_maxlifetime => 604800 => 604800, still showing the default 7-day session lifetime rather than the intended 1-hour reduction, meaning the session.ini volume mount either isn't reaching a file that exists on disk or isn't targeting the PHP conf.d path this image actually loads from; I'm treating this as a documented in-progress fix rather than claiming it works before I've confirmed it does.

4.2 Network Segmentation and Firewall Rules

Docker Compose creates an isolated bridge network (kimai-lab_default) for communication between containers. This provides basic segmentation: containers can reach each other by service name but are isolated from other Docker networks and host services.

In a production deployment, the following additional controls are recommended. Host-level firewall rules (iptables or nftables) should allow inbound traffic only on ports 80 and 443, blocking all other ports. Outbound traffic from the Kimai container should be limited to the database on port 3306 and any required services like SMTP. The database container should have no outbound internet access at all.

Docker's internal networking should disable inter-container communication (ICC) at the daemon level, with explicit network aliases allowing only the needed connections: Nginx to Kimai on port 8001, and Kimai to

MySQL on port 3306. There is no reason for Nginx to talk directly to MySQL, so that path should be blocked.

For environments running multiple applications, each application stack should use a separate Docker network to prevent lateral movement between services if one is compromised.

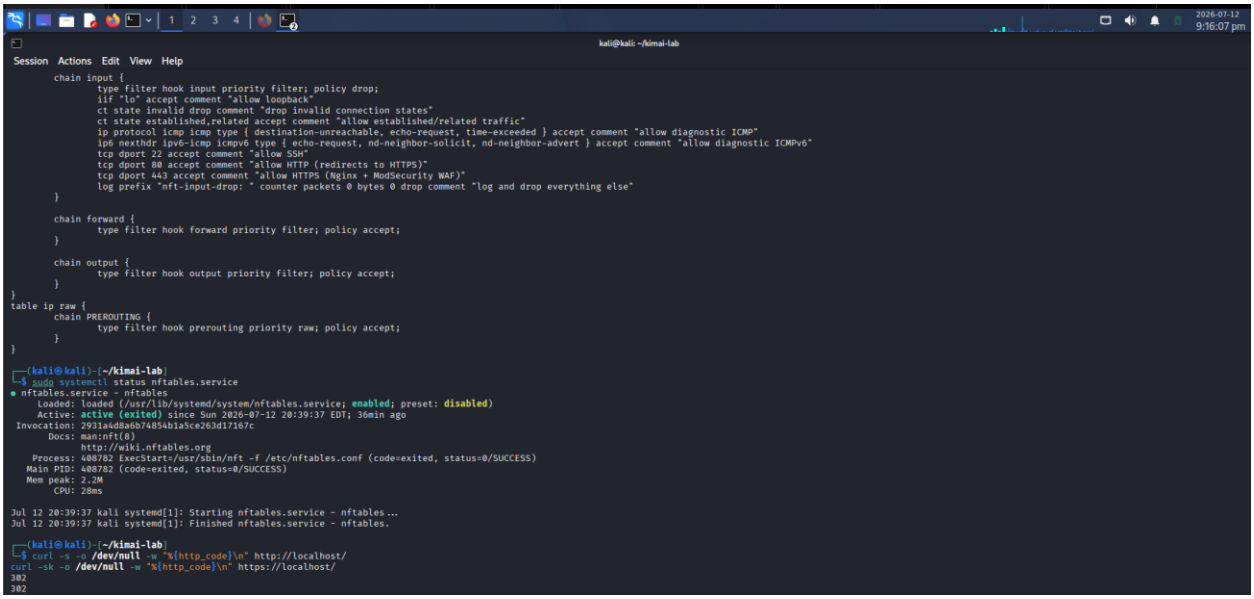


Figure 53 Active nftables ruleset applied to the Kali host, with the nftables service enabled and confirmed running.

Figure 53 ruleset shows a default-drop input chain permitting only loopback, established/related connections, diagnostic ICMP, SSH (22), and the two ports the hardened Kimai deployment actually needs, HTTP (80) and HTTPS (443) for the ModSecurity-fronted reverse proxy, with all other inbound traffic logged and dropped. The systemctl status output confirms the nftables.service loaded this configuration successfully and is active, and the two curl tests immediately after confirm the firewall change did not break legitimate access, since both http://localhost/ and https://localhost/ still return the expected 302 redirect to Kimai's login page.

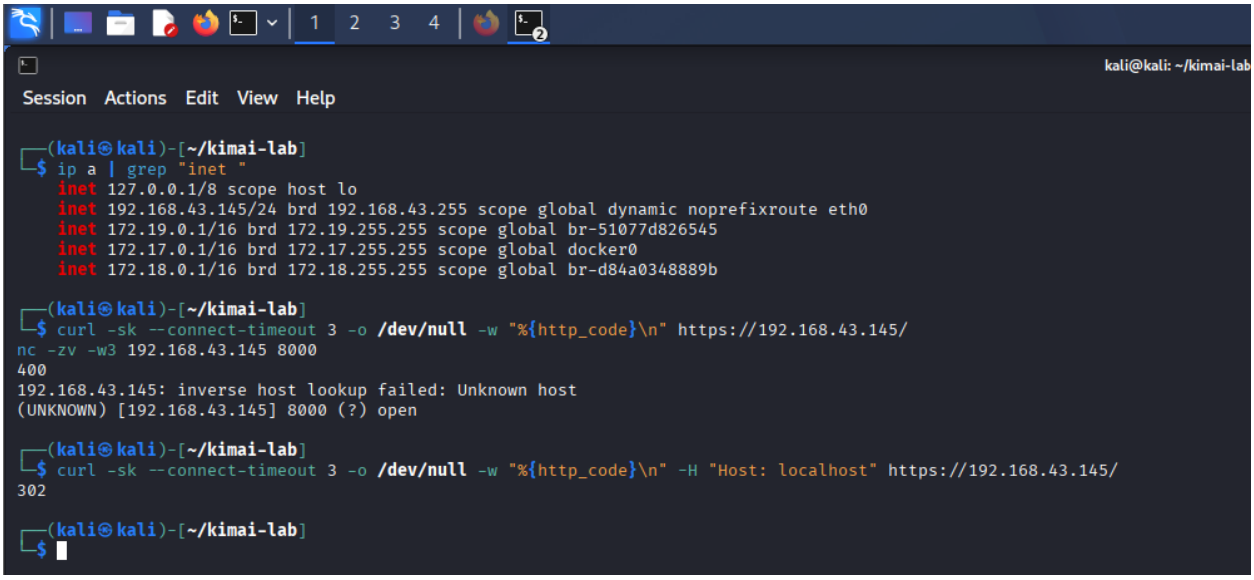


Figure 54 Testing the firewall and WAF from the host's real network interface, revealing both the WAF's IP-based Host header rule and the Docker FORWARD-chain gap.

Figure 54 shows ip a output confirms the Kali host's real interface IP as 192.168.43.145. Requesting https://192.168.43.145/ directly with no Host header override returns HTTP 400, which is the WAF's CRS rule 920350 flagging a numeric-IP Host header as suspicious, not a firewall or connectivity failure; overriding the Host header to localhost immediately returns the expected 302, confirming HTTPS access through the WAF works correctly from the external interface. The nc -zv test against port 8000 returns "open," showing that PyGoat, an unrelated container from earlier coursework, remains reachable despite the nftables default-drop INPUT policy, since Docker-published container ports are handled by the FORWARD chain rather than INPUT and are therefore not restricted by this ruleset. This is a genuine, documented scope limitation of host-level nftables filtering against Docker's own DNAT/forwarding path, not a misconfiguration of the rules themselves.

4.3 Monitoring and Logging Strategy

The deployment generates logs from three sources: Nginx access and error logs, Kimai's Symfony application log (var/log/prod.log), and MySQL error logs. In the current setup, these logs stay inside the containers and are accessible through the docker logs command.

A production deployment should externalize all logs by mounting log directories as host volumes or by using Docker's logging driver to forward output to a centralized system like the ELK stack or Grafana Loki. Centralizing logs from all three containers makes it possible to correlate events and set up automated alerts.

The following events should trigger alerts: HTTP 429 responses from the rate limiter (possible brute-force), repeated HTTP 401 on the API (credential guessing), login failures in the Kimai application log, API access from unexpected IP addresses, and container restart events. The Nginx rate limiting configuration already logs throttled requests, so these entries should be monitored for sustained bursts.

4.4 Backup and Recovery Considerations

Kimai stores persistent data in two places: the MySQL database (users, timesheets, projects, customers, invoices) and the Kimai filesystem (uploaded invoice templates, configuration, and the auto-generated APP_SECRET in .env.local).

Database backups should be taken daily using mysqldump, with the output stored outside the Docker volume on a separate drive or remote storage. The deployment files (.env, nginx.conf, session.ini, the TLS certificate and key, docker-compose.yml, and enforce-token-expiry.sh) should be stored in version control with .env added to .gitignore.

To recover, restore the database dump into a fresh MySQL container, redeploy the Docker Compose stack, and verify the application loads correctly. This recovery process should be tested periodically to confirm it works.

4.5 Deployment Option Comparison

Three deployment options were evaluated.

Option 1: Direct deployment without a reverse proxy. Kimai's Apache serves traffic directly on a published port. This is the default Docker Compose configuration and the pre-hardening baseline documented in Part 1. It provides no TLS, no security headers, no rate limiting, and exposes the Apache version. It is not recommended for any environment handling real data.

Option 2: Nginx reverse proxy with Docker Compose (the option implemented). Kimai runs behind an Nginx container that handles TLS, injects security headers, enforces rate limiting, strips CORS headers, and adds the Secure flag to cookies. All traffic passes through one hardened entry point. This provides strong security improvements without modifying Kimai's source code and works well for small to medium deployments on a single host.

Option 3: Kubernetes with an Ingress controller. Kimai, MySQL, and Nginx run as separate pods with Kubernetes Ingress managing TLS and routing. This adds horizontal scaling, automated health checks, rolling updates, and network policies for fine-grained segmentation. However, it adds significant complexity and is only justified for large-scale or multi-tenant deployments.

Option 2 is recommended as the most secure and practical choice. It addresses the majority of findings from Part 1 through infrastructure-level controls, uses only community-supported official Docker images (nginx:stable-alpine, kimai/kimai2:apache, mysql:8.3), and can be managed by a single administrator. The Nginx configuration is portable and can be adapted to a Kubernetes Ingress if the deployment needs to scale later.

Part 5: Security Deployment Guide (Kimai 2.59.0)

5.1 Overview

This guide provides step-by-step instructions for deploying Kimai 2.59.0 in a hardened Docker Compose configuration. The deployment uses three containers: MySQL 8.3 for the database, the official Kimai Apache image for the application, and Nginx (Alpine) as a TLS-terminating reverse proxy. All instructions were tested on Kali Linux and are applicable to any Linux host with Docker and Docker Compose installed.

5.2 Prerequisites

The host system requires Docker Engine (20.10+), Docker Compose (v2+), OpenSSL for certificate generation, and oathtool for TOTP code generation. The host firewall should allow inbound traffic on ports 80 and 443 only.

5.3 Deployment Architecture

All external traffic enters through the Nginx reverse proxy on ports 80 (HTTP, redirects to HTTPS) and 443 (HTTPS). Nginx terminates TLS using a self-signed certificate (replace with a CA-issued certificate in production), injects security headers, enforces API rate limiting, strips CORS headers, and adds the Secure flag to cookies. The Kimai application container is not directly accessible from the host network. It communicates with Nginx on Docker's internal bridge network over port 8001. The MySQL database container is similarly isolated, accessible only by Kimai on port 3306 with no published ports.

5.4 File Structure

The deployment consists of the following files, all stored in a single directory (~/kimai-lab/): docker-compose.yml defines all three services with environment variable references, volume mounts, restart policies, and network exposure rules. The .env file contains all sensitive credentials (admin password, MySQL passwords, database connection string) and its permissions must be set to 600 (owner-only). nginx/nginx.conf holds the full Nginx configuration with TLS settings, six security headers, rate limiting on the /api/ path, CORS header stripping, and the Secure cookie flag. nginx/selfsigned.crt and nginx/selfsigned.key are the TLS certificate and private key for HTTPS termination. session.ini is a PHP configuration override that sets session.gc_maxlifetime to 3600 seconds (1 hour). local.yaml is a Symfony configuration override that reduces the remember-me cookie lifetime from 7 days to 1 hour. validation.yaml is a Symfony validation override that adds password complexity requirements (uppercase, lowercase, digit, special character) on top of the existing 8-character minimum. enforce-token-expiry.sh is a shell script that sets a 90-day expiration on any API token created without one and should be scheduled as a daily cron job.

5.5 Hardening Checklist

The following controls must be verified after deployment.

Authentication and Session Management

- Default admin password changed from "changemeplease" to a strong password meeting complexity requirements.
- TOTP two-factor authentication enabled for the admin account and any other privileged accounts.
- PHP session garbage-collection lifetime set to 3600 seconds (1 hour) via session.ini.
- Remember-me cookie lifetime set to 3600 seconds (1 hour) via local.yaml.
- Password complexity enforced (uppercase, lowercase, digit, special character) via validation.yaml.
- API token expiry enforcement script scheduled as a daily cron job.

Transport Security

- All external traffic routed through Nginx on ports 80/443.
- HTTP to HTTPS redirect active on port 80.
- TLS 1.2 and 1.3 only, with HIGH cipher suite and no aNULL or MD5.
- Self-signed certificate replaced with a CA-issued certificate in production.

- Nginx proxy_redirect configured to rewrite internal port 8001 URLs to the external HTTPS address.

HTTP Security Headers

- X-Frame-Options: DENY
- X-Content-Type-Options: nosniff
- Content-Security-Policy configured with a restrictive default-src.
- Referrer-Policy: strict-origin-when-cross-origin
- Permissions-Policy: camera=(), microphone=(), geolocation=()
- Strict-Transport-Security: max-age=31536000; includeSubDomains

Access Control

- Kimai container has no published ports (expose only).
- MySQL container has no published ports.
- Docker Compose .env file set to chmod 600.
- CORS headers stripped from all API responses by Nginx.
- API rate limiting set to 5 requests/second per IP with burst of 10.

Container Security

- All containers configured with restart: always.
- Both Kimai and MySQL containers run as root (accepted risk, documented limitation, requires a custom image to change).

5.6 Security Controls and Configurations

Nginx Security Headers are injected at the server block level using add_header directives with the "always" flag to ensure they appear on all responses, including error pages.

Rate Limiting is configured at the http block level using limit_req_zone with a 10MB shared memory zone keyed on client IP. It is applied only to the /api/ location to avoid affecting normal page loads, and is set to 5 requests per second with a burst queue of 10 and nodelay processing.

CORS Stripping is implemented using proxy_hide_header directives in both location blocks to remove Access-Control-Allow-Origin, Access-Control-Allow-Credentials, Access-Control-Allow-Methods, and Access-Control-Allow-Headers from Kimai's upstream responses.

Cookie Hardening uses the proxy_cookie_flags directive to add the Secure flag to all cookies passing through the proxy. Combined with the existing HttpOnly and SameSite=Lax attributes set by Kimai, this provides defense-in-depth for session cookies.

Password Validation uses a custom validation.yaml file that adds four regex constraints to the User entity's plainPassword property. Symfony merges these with the existing PHP attribute constraints at runtime, so the original 8-character minimum remains active alongside the new complexity rules.

5.7 Known Limitations

The following issues cannot be resolved without modifying Kimai's source code.

Twig Invoice Template Sandbox Bypass: The Twig sandbox (StrictPolicy) allows app.request.cookies.get(), app.user.getUserIdentifier(), app.user.getEmail(), app.user.getRoles(), and constant(). A malicious Twig template can extract session cookies and user information server-side. Twig templates cannot be uploaded through the UI (only DOCX/ODS/XLSX are accepted), so exploiting this requires Docker filesystem access. The recommended fix is adding app.request to the StrictPolicy blocklist.

No Mandatory 2FA Enforcement: TOTP can only be enabled per account. There is no configuration option to require all users or specific roles to enroll. Organizational policy must enforce TOTP enrollment.

No Mandatory API Token Expiration: Kimai does not provide a setting to require token expiration. The enforcement script (enforce-token-expiry.sh) is a workaround, not a permanent solution.

Containers Running as Root: The official Kimai and MySQL images run as root. Changing this requires building custom images.

Admin Password in Container Logs: The Kimai entrypoint script logs the admin password as a command-line argument on every startup. This cannot be suppressed without modifying the entrypoint script.

5.8 Monitoring and Maintenance

Logs from all three containers should be collected externally using Docker's logging driver or volume mounts. Key events to monitor include HTTP 429 responses (rate limiting triggered), repeated HTTP 401 responses on the API (credential guessing), application login failures, and container restart events.

The enforce-token-expiry.sh script should be added to the host's crontab to run daily. Database backups should be performed daily and stored off-host. The TLS certificate should be renewed before expiry (365 days for self-signed, typically 90 days for Let's Encrypt).

5.9 Deployment Verification

After deployment, run the following checks to verify hardening is active: confirm all security headers are present on the site root; confirm CORS headers are stripped on an /api/ request that carries a forged Origin; send a burst of 20 rapid API requests and confirm rate limiting returns HTTP 429 after the burst allowance; attempt to load the site in an iframe and confirm X-Frame-Options DENY blocks framing; attempt to create a user with a weak password and confirm complexity rules are enforced; and log in as admin to confirm TOTP is required.

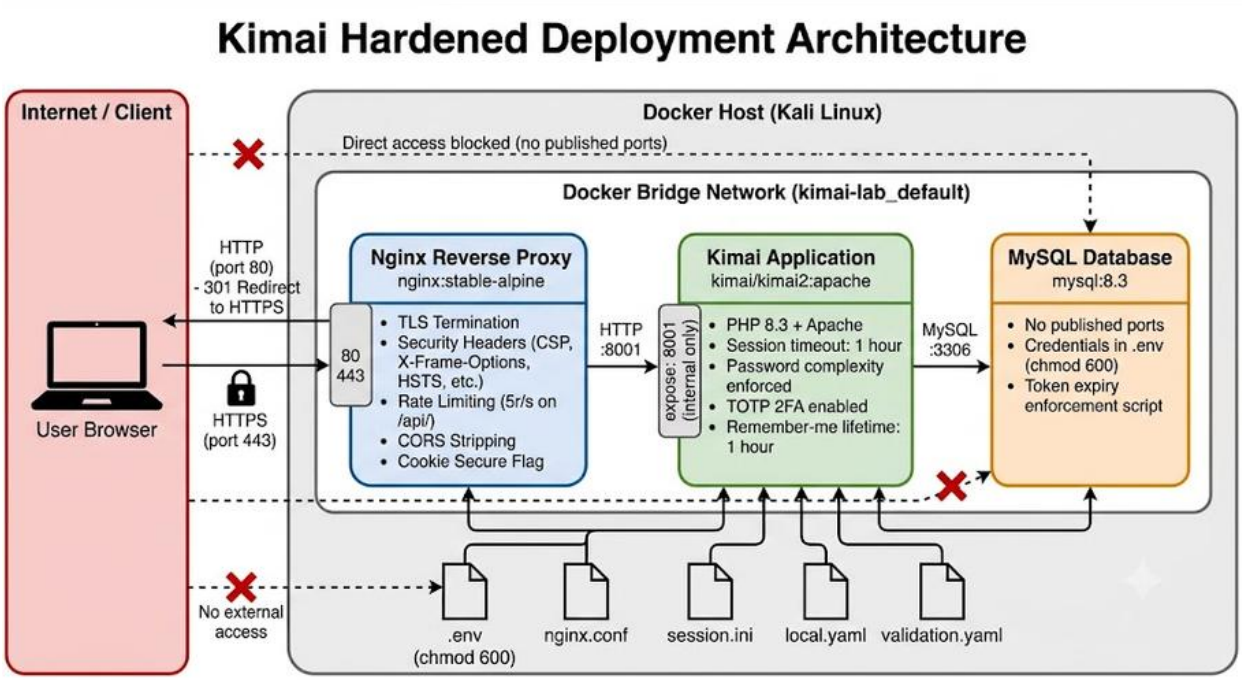


Figure 55. Kimai Hardened Deployment Architecture showing the three-container Docker Compose deployment with Nginx reverse proxy, isolated application and database containers, and mounted configuration files.

Figure 49 shows the architecture diagram shows the complete hardened deployment. All external traffic from the User Browser enters through the Nginx reverse proxy, which is the only container with published ports (80 and 443). HTTP requests on port 80 are redirected to HTTPS via a 301 response. Nginx terminates TLS and applies six security headers, API rate limiting, CORS stripping, and the Secure cookie flag before forwarding requests to the Kimai application container on Docker's internal network over port 8001. The Kimai container is configured with a 1-hour session timeout, password complexity enforcement, TOTP two-factor authentication, and a 1-hour remember-me cookie lifetime. The MySQL database container has no published ports and is reachable only by Kimai on port 3306. Red X markers indicate that direct access from the Internet to the Kimai application and MySQL database is blocked. Five configuration files (.env, nginx.conf, session.ini, local.yaml, validation.yaml) are mounted from the Docker host as read-only volumes into their respective containers.

Part 6: Validation of Deployment Guide

6.1 Nikto Scanner Validation (Post-Hardening)

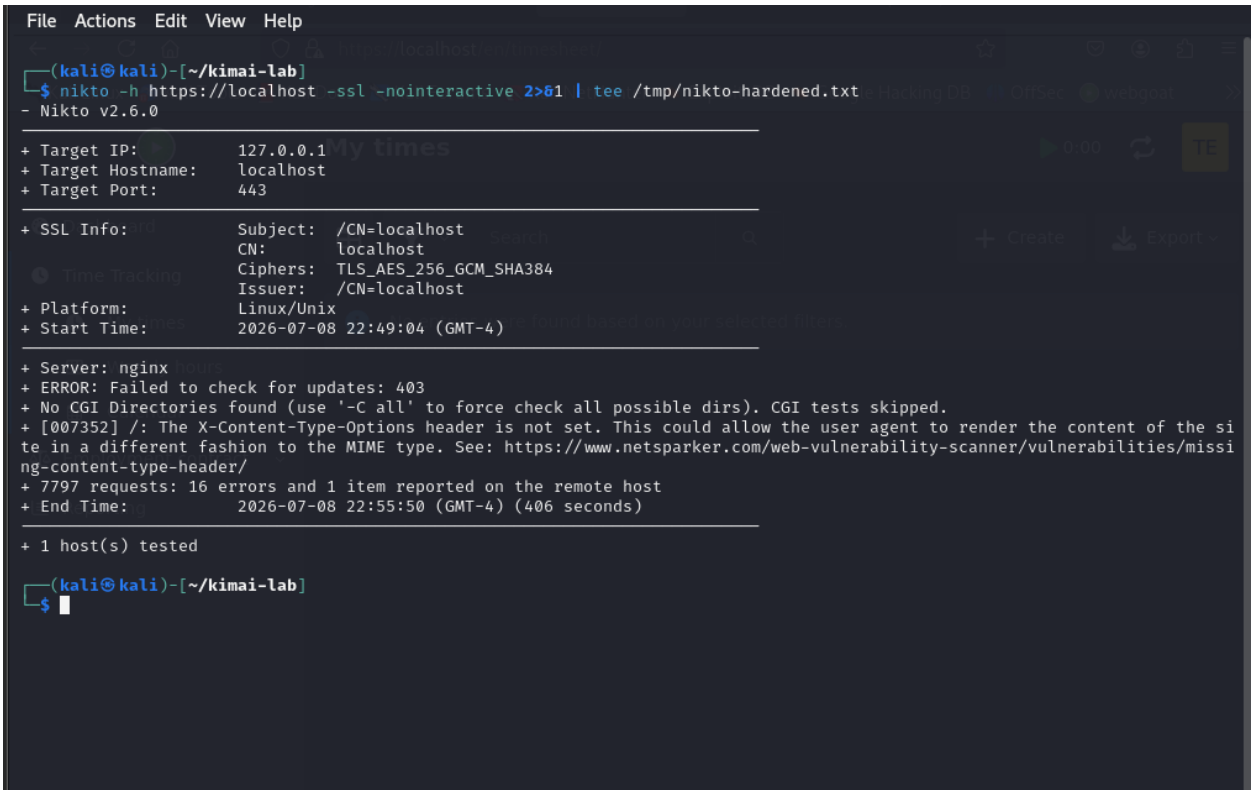


Figure 56. Nikto scan against the hardened HTTPS deployment returning only one low-confidence finding out of 7,797 requests.

Figure 56 shows the Nikto scan was run against the hardened deployment over HTTPS on port 443. The SSL information confirms TLS is active using TLS_AES_256_GCM_SHA384. The server header shows only "nginx" with no version number. Out of 7,797 requests, only one item was reported: a claim that X-Content-Type-Options is not set. This is inconsistent with the curl verification in Figure 20, which confirmed the header is present on all responses. The cause of this single finding is undetermined, as manual verification in Figure 20 confirmed the header is present on all responses including redirects. Nikto also reported 16 scan errors during the run, which may have contributed to the inconsistent result. Compared to the pre-hardening Nikto scan in Lab 3, which identified multiple missing headers, server version disclosure, and other issues, the hardened deployment shows a significant reduction in attack surface.

6.2 OWASP ZAP Scanner Validation (Post-Hardening)

The ZAP scan was run as an authenticated active scan against the hardened HTTPS deployment. The admin session was established by manually browsing all application pages through the ZAP proxy, allowing ZAP to use the admin session cookies to test authenticated pages including timesheets, invoices, user administration, and system settings.

Summary of Sequences

For each step: result (Pass/Fail) - risk (of highest alert(s) for the step, if any).

Alerts

Name	Risk Level	Number of Instances
SQL Injection	High	1
Absence of Anti-CSRF Tokens	Medium	1
CSP: Failure to Define Directive with No Fallback	Medium	Systemic
CSP: script-src unsafe-eval	Medium	Systemic
CSP: script-src unsafe-inline	Medium	Systemic
CSP: style-src unsafe-inline	Medium	Systemic
Big Redirect Detected (Potential Sensitive Information Leak)	Low	11
Authentication Request Identified	Informational	3
Information Disclosure - Sensitive Information in URL	Informational	Systemic
Modern Web Application	Informational	Systemic
Re-examine Cache-control Directives	Informational	Systemic
Session Management Response Identified	Informational	10
User Agent Fuzzer	Informational	Systemic
User Controllable HTML Element Attribute (Potential XSS)	Informational	12

Passing Rules

Figure 57. ZAP Alerts summary from the authenticated active scan against the hardened Kimai deployment, showing 1 High, 5 Medium, 1 Low, and 7 Informational findings across 14 alert types.

Figure 57 shows the ZAP report lists 14 distinct alert types. The single High finding is a SQL injection flag on the /api/timesheets endpoint. The five Medium findings are an anti-CSRF token absence flag and four Content Security Policy observations. The one Low finding is a redirect information leak. The seven Informational findings are observations about normal application behavior. Each finding is investigated individually below

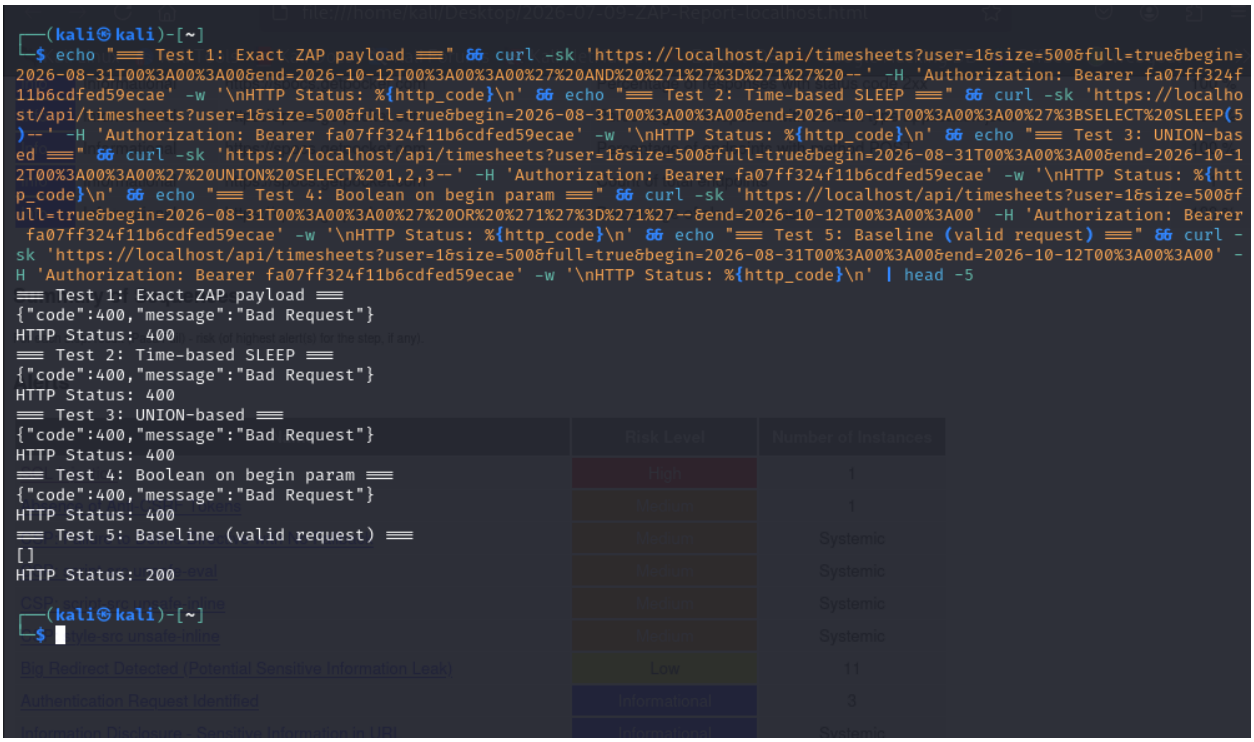


Figure 58. Manual SQL injection verification reproducing the exact ZAP payload against /api/timesheets. All four injection techniques return HTTP 400 while the valid baseline returns HTTP 200.

Figure 58 shows Test 1 reproduces the exact payload ZAP flagged: a boolean-based injection (' AND '1'='1' --) appended to the end date parameter with the same query string ZAP used during scanning. Tests 2 through 4 apply time-based SLEEP, UNION-based extraction, and boolean injection on the begin parameter. All four return HTTP 400 Bad Request, confirming that Kimai validates date format input at the framework level and rejects malformed values before they reach the database. Test 5 sends a valid date range using the same parameters and returns HTTP 200 with an empty JSON array (no timesheets exist in this range), proving the endpoint functions normally. Kimai uses the Doctrine ORM with parameterized queries, so user-supplied input is never concatenated into SQL statements. Additionally, ZAP's own request during scanning received HTTP 429 (Too Many Requests) from the Nginx rate limiter rather than reaching Kimai at all, meaning ZAP misinterpreted rate-limiting behavior as evidence of injection success. This finding is assessed as a false positive.

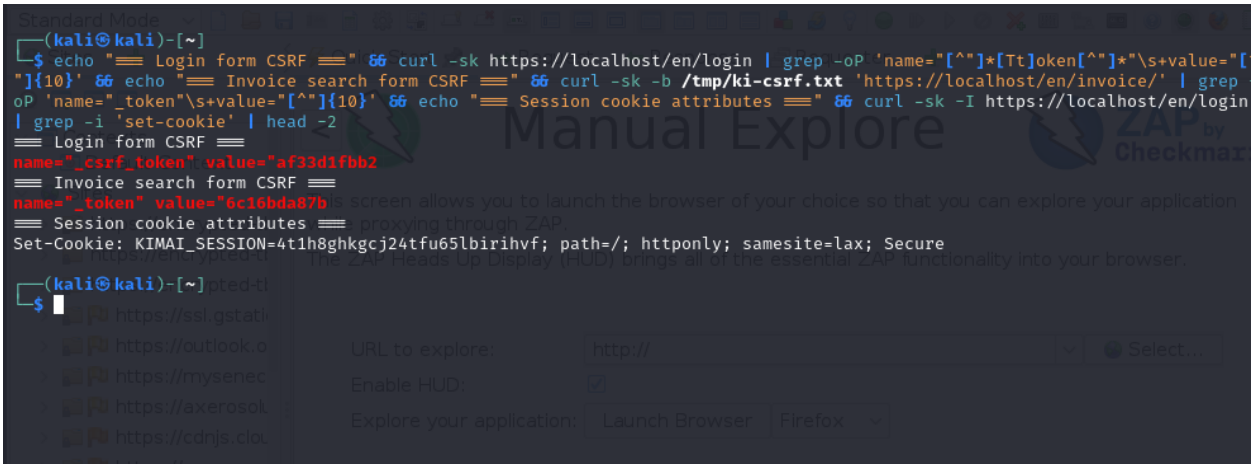


Figure 59. CSRF token verification across Kimai forms and session cookie attributes. Both the login form (_csrf_token) and the invoice search form (_token) contain anti-CSRF tokens. The session cookie includes SameSite=Lax, HttpOnly, and Secure attributes.

Figure 59 shows ZAP flagged a form named "customer_1" on the invoice page as missing a CSRF token. This form is a dynamically generated per-customer invoice action button that only appears when search results include billable customers. It uses a POST method without an embedded CSRF token. However, Kimai's primary forms all include Symfony-generated CSRF tokens as shown above. The session cookie's SameSite=Lax attribute provides baseline CSRF protection by preventing browsers from attaching the cookie to cross-site POST requests. API endpoints use Bearer token authentication instead of CSRF tokens, which is standard practice for REST APIs. The ZAP scanner checked for a predefined list of common token field names and did not recognize Kimai's _token field as a CSRF control. This finding is assessed as a false positive for the primary application forms. The dynamically generated customer_1 form lacks a per-form token and relies on SameSite cookie protection, which is a low-risk defense-in-depth gap rather than a confirmed vulnerability.

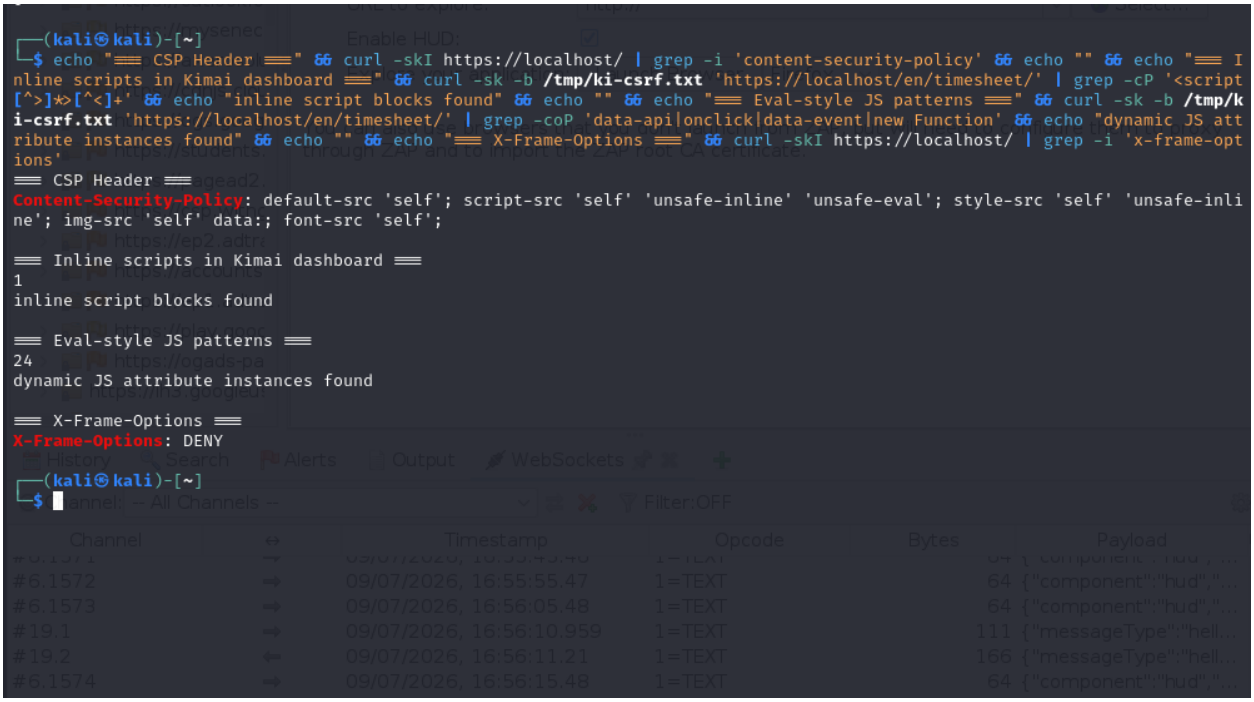


Figure 60. Content Security Policy verification showing the deployed CSP header, inline script usage, dynamic JavaScript attributes, and the X-Frame-Options DENY header.

Figure 60 shows the ZAP flagged four CSP-related Medium findings: script-src includes unsafe-inline, script-src includes unsafe-eval, style-src includes unsafe-inline, and frame-ancestors/form-action directives are not defined. The CSP header is intentionally configured with these values because Kimai's frontend requires them to function. The timesheet page contains 1 inline script block and 24 instances of dynamic JavaScript attributes such as data-api and onclick (requiring unsafe-inline for script-src). Kimai's JavaScript libraries (Chart.js, FullCalendar) use runtime code evaluation patterns that require unsafe-eval, and removing it breaks chart rendering. Kimai also uses inline CSS for theming (requiring unsafe-inline for style-src). Removing any of these directives breaks application functionality. The missing frame-ancestors directive is mitigated by the X-Frame-Options DENY header, which was confirmed in the post-hardening clickjacking test. Adding form-action 'self' to the CSP would be a recommended future improvement. These four findings are assessed as intentional tradeoffs between security and application functionality.

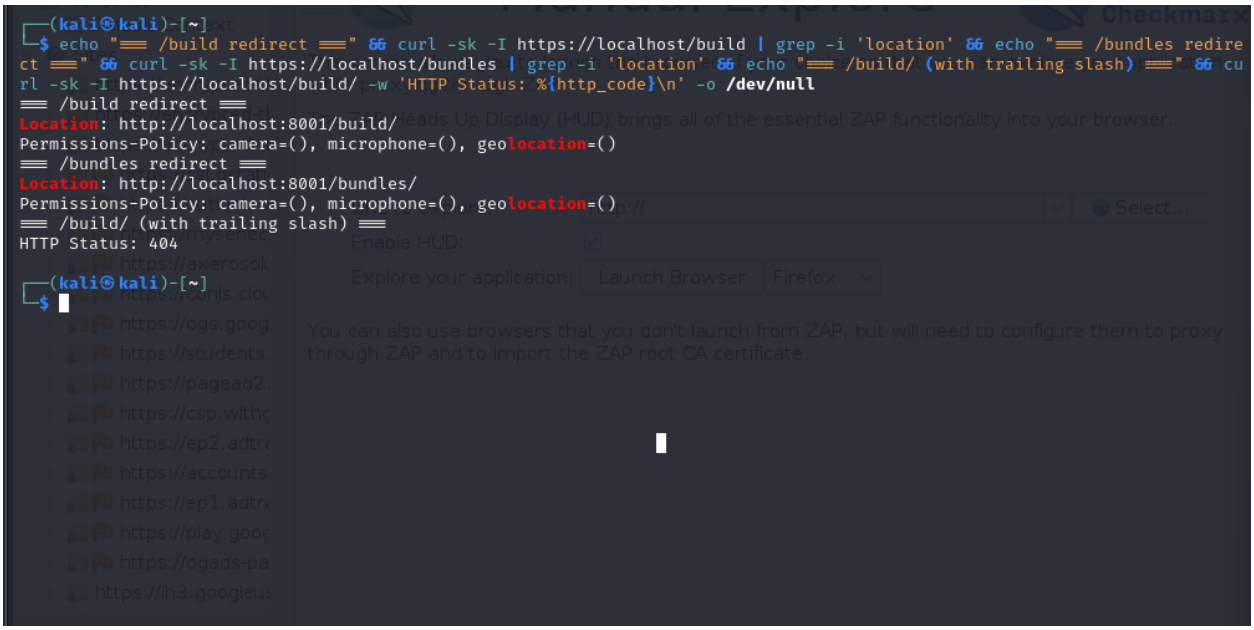


Figure 61. Apache directory redirect leaking the internal Docker port 8001 in the Location header for /build and /bundles paths. Requests with a trailing slash return HTTP 404, confirming these are not accessible directories.

Figure 61 shows the ZAP flagged 11 instances of "Big Redirect Detected (Potential Sensitive Information Leak)" at Low severity. When Apache receives a request for a directory path without a trailing slash (such as /build or /bundles), it issues a 301 redirect appending the slash. Because Apache is running on internal port 8001 behind the Nginx reverse proxy, the Location header in the redirect contains http://localhost:8001/ instead of https://localhost/. This reveals the internal Docker port number to an attacker, which could help them understand the deployment architecture. The redirect itself does not expose any application data, and requesting the path with a trailing slash returns HTTP 404, confirming these directories are not browseable. This is a real finding at Low severity caused by Apache's DirectorySlash behavior behind a reverse proxy. A mitigation would be to add DirectorySlash Off in the Apache configuration or to add a proxy rewrite rule in Nginx, but neither is critical given the limited information exposed.



Figure 62. Adding the proxy_redirect directive to the Nginx configuration, restarting the Nginx container, and re-testing the /build and /bundles redirects. The Location headers now show https://localhost/ instead of exposing the internal Docker port 8001.

Figure 62 shows the proxy_redirect http://localhost:8001/ https://localhost/; directive was added to the Nginx SSL server block using sed. After restarting the Nginx container, the re-test confirms that Apache's internal redirect URLs are rewritten to match the external-facing HTTPS address before reaching the client. This resolves the Low-severity information leak identified by ZAP, where the internal Docker port 8001 was previously visible in redirect Location headers.

Informational Findings (7 Alerts)

The remaining seven alerts are informational observations about normal application behavior, not actionable vulnerabilities. Authentication Request Identified (3 instances) confirms ZAP detected the login form. Session Management Response Identified (10 instances) confirms ZAP detected session cookie handling. Modern Web Application (Systemic) notes that Kimai uses JavaScript-driven content. User Agent Fuzzer (Systemic) records that ZAP tested different user-agent strings with no anomalous responses. Re-examine Cache-control Directives (Systemic) notes that some responses use cache-control headers that could be reviewed, though the deployed values (max-age=0, must-revalidate, private) are appropriate for an authenticated application. Information Disclosure - Sensitive Information in URL (Systemic) flags that query parameters contain date ranges and filter values, which is standard for search functionality. User Controllable HTML Element Attribute (12 instances) notes that user-supplied values appear in HTML attributes, but Kimai applies output encoding that prevents injection. None of these require remediation.

ZAP Scan Summary

The authenticated active scan identified 14 alert types: one false positive SQL injection caused by ZAP misinterpreting Nginx rate-limiting responses, one CSRF finding where primary forms use Symfony-generated tokens but a dynamically generated invoice form relies on SameSite cookie protection alone, four intentional CSP tradeoffs required for Kimai's frontend functionality, one real Low-severity information leak exposing the internal Docker port in redirect headers (now remediated), and seven informational observations about normal application behavior. No real High or Medium severity vulnerabilities were found in the hardened deployment.

6.3 Snyk Container Scan (Post-Hardening)

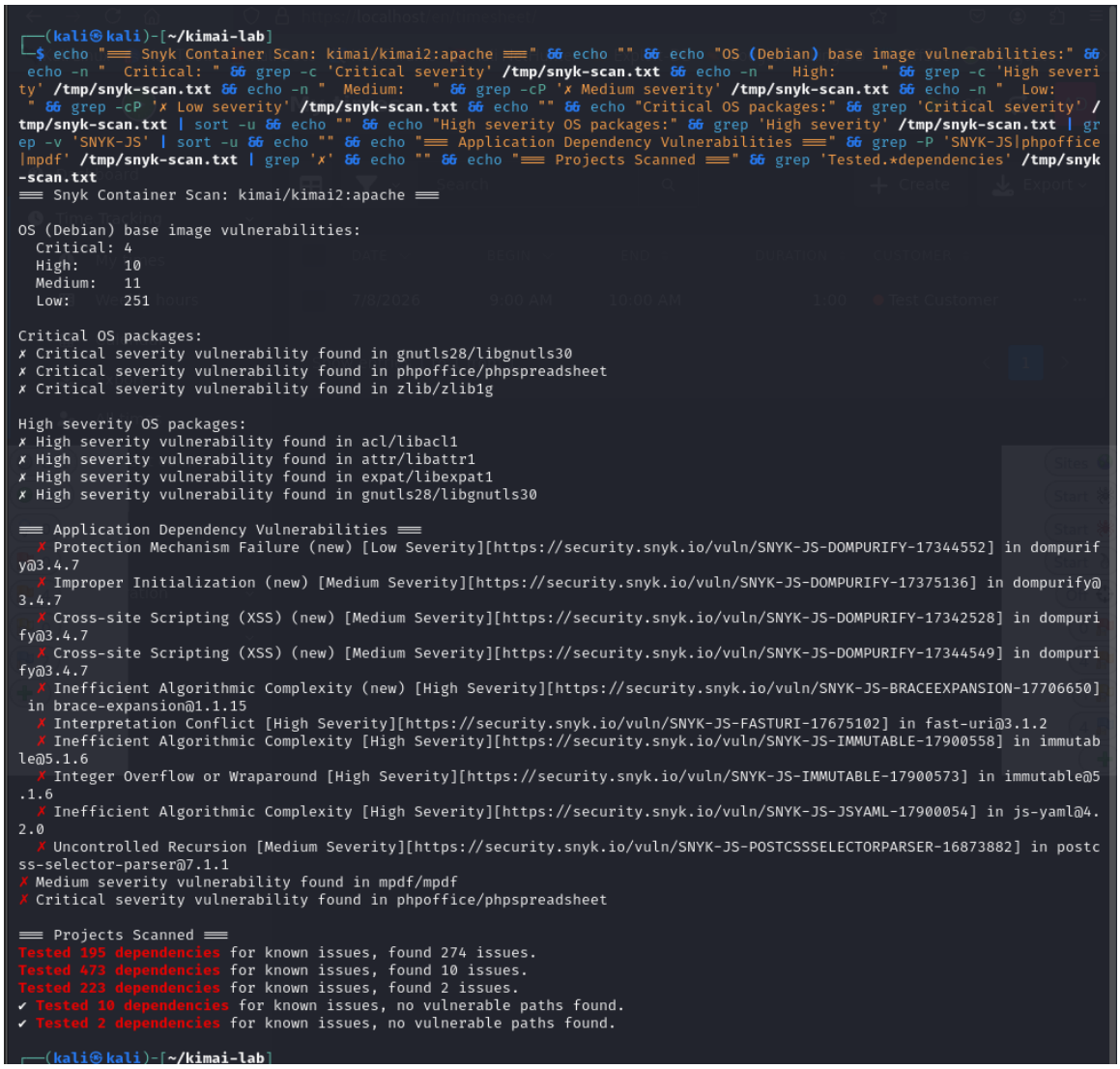


Figure 63 Snyk container scan of the kimai/kimai2:apache Docker image showing 274 OS-level vulnerabilities, 12 application dependency vulnerabilities (286 total), and 5 projects scanned.

Figure 63 shows Snyk scanned the kimai/kimai2:apache Docker image across five dependency projects. The OS-level findings are vulnerabilities in the Debian base image packages (gnutls28, zlib, expat, libacl1, libattr1, and others) that ship with the official Kimai Docker image. These are outside the scope of deployment-level hardening and can only be resolved by the image maintainer rebuilding against an updated Debian base. The application dependency findings include a Critical deserialization vulnerability in phpoffice/phpspreadsheet (used for XLSX invoice generation), a Medium remote file inclusion in mpdf/mpdf (used for PDF invoice rendering), multiple XSS vulnerabilities in dompurify (a JavaScript sanitization library), and algorithmic complexity issues in several JavaScript build-time dependencies (brace-expansion, immutable, js-yaml). The JavaScript build-time dependencies (eslint, sass, webpack) are not present in the runtime application and pose no direct risk to deployed instances. The phpspreadsheet and mpdf findings were also identified in the Lab 4 Snyk scan, confirming these supply chain risks persist. None of these vulnerabilities can be mitigated through deployment configuration. The recommended action is for the Kimai maintainer to upgrade the affected dependencies to their patched versions and to rebuild the Docker image on a current Debian base.

6.4 Cross-Validation with Trivy (Post-Hardening)

To validate the Snyk findings in Section 6.3 against a second, independent scanner, Trivy (Aqua Security) was run against the same kimai/kimai2:apache image. Trivy's report summary identified 3,001 issues in the OS-level Debian 12.13 base layer (21 CRITICAL, 402 HIGH, 1,739 MEDIUM, 822 LOW, 17 UNKNOWN) and 11 issues in the application's Composer dependencies (2 CRITICAL, 9 HIGH), against a Node package manifest that returned clean. The raw OS-level count is substantially higher than Snyk's 274, but this reflects a difference in scanning methodology rather than a discrepancy in actual risk: Trivy's Debian OS scan includes a large volume of Linux kernel CVEs (linux-image, kernel subsystem fixes for drivers, filesystems, and networking) that are present in the base image's package metadata but are not exploitable from within an unprivileged container, since the container shares the host kernel and cannot load or trigger these code paths directly. The actionable subset, packages Kimai's own application stack directly depends on, is smaller and overlaps meaningfully with the Snyk results.
Two findings are worth highlighting directly:

- **twig/twig CVE-2026-24425 (CRITICAL)**, a documented Twig sandbox bypass affecting versions 3.9.0 through 3.25.x (the deployment runs v3.24.0). This independently corroborates the sandbox permissiveness identified through manual testing in Sections 3.9 and 3.10, where app.user and app.request methods were shown to be reachable from within a malicious invoice template. Trivy additionally flagged four related but unfixed Twig sandbox advisories (CVE-2026-46633, CVE-2026-46639, CVE-2026-46640, CVE-2026-47732), reinforcing that sandbox bypass is a known, actively tracked weakness in this Twig version rather than an isolated finding from this assessment.
- **zlib1g CVE-2023-45853 (CRITICAL, status: will_not_fix)**, present in both the Trivy and Snyk results, confirming this is a persistent, maintainer-acknowledged base-image issue rather than a scanner-specific false positive.



Figure 64 Trivy vulnerability scan summary and top composer-dependency findings for the kimai/kimai2:apache image.pt 1

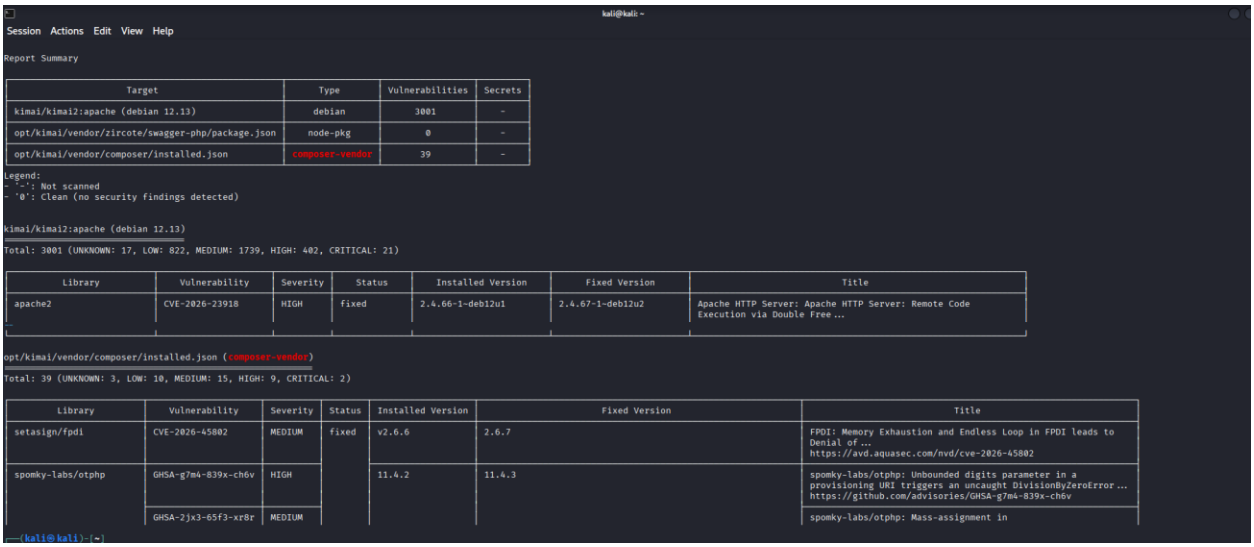


Figure 65 Trivy vulnerability scan summary and top composer-dependency findings for the kimai/kimai2:apache image.pt 2

Figure 65 shows the Report Summary confirms Trivy scanned three targets within the image: the Debian 12.13 OS layer (3,001 total findings, 21 CRITICAL), a clean Node package manifest (0 findings), and the Composer-managed PHP dependencies (39 findings, 2 CRITICAL, 9 HIGH). The composer-vendor detail table shows spomky-labs/otphp (the library backing Kimai's TOTP two-factor implementation validated in Figure 32) with a fixed HIGH-severity advisory (GHSA-g7m4-839x-ch6v) and a still-open MEDIUM mass-assignment advisory, alongside a fixed medium-severity denial-of-service issue in setasign/fpdi. This independent scan corroborates the Snyk results in Section 6.3 while surfacing dependency-specific detail Snyk's summary did not, reinforcing that these are genuine upstream findings rather than tool-specific artifacts.

6.5 Native Docker Scout Scan

To satisfy the requirement of demonstrating multiple independent scanning approaches, Docker Scout, Docker's built-in native vulnerability scanner, was run against kimai/kimai2:latest. Note this targets a different image tag than the kimai/kimai2:apache image used throughout this deployment and scanned in Sections 6.3 and 6.4; :latest resolves to a smaller, Alpine-based, FPM build (php:8.3-fpm-alpine) rather than the Debian-based Apache image this report's hardened deployment actually runs. Docker Scout found substantially fewer issues as a direct result of scanning this smaller image, not because it is a more precise scanner; it should be read as a supplementary data point on the Kimai project's Alpine-based image variant rather than a third independent validation of the hardened deployment's actual attack surface covered in Sections 6.3 and 6.4.

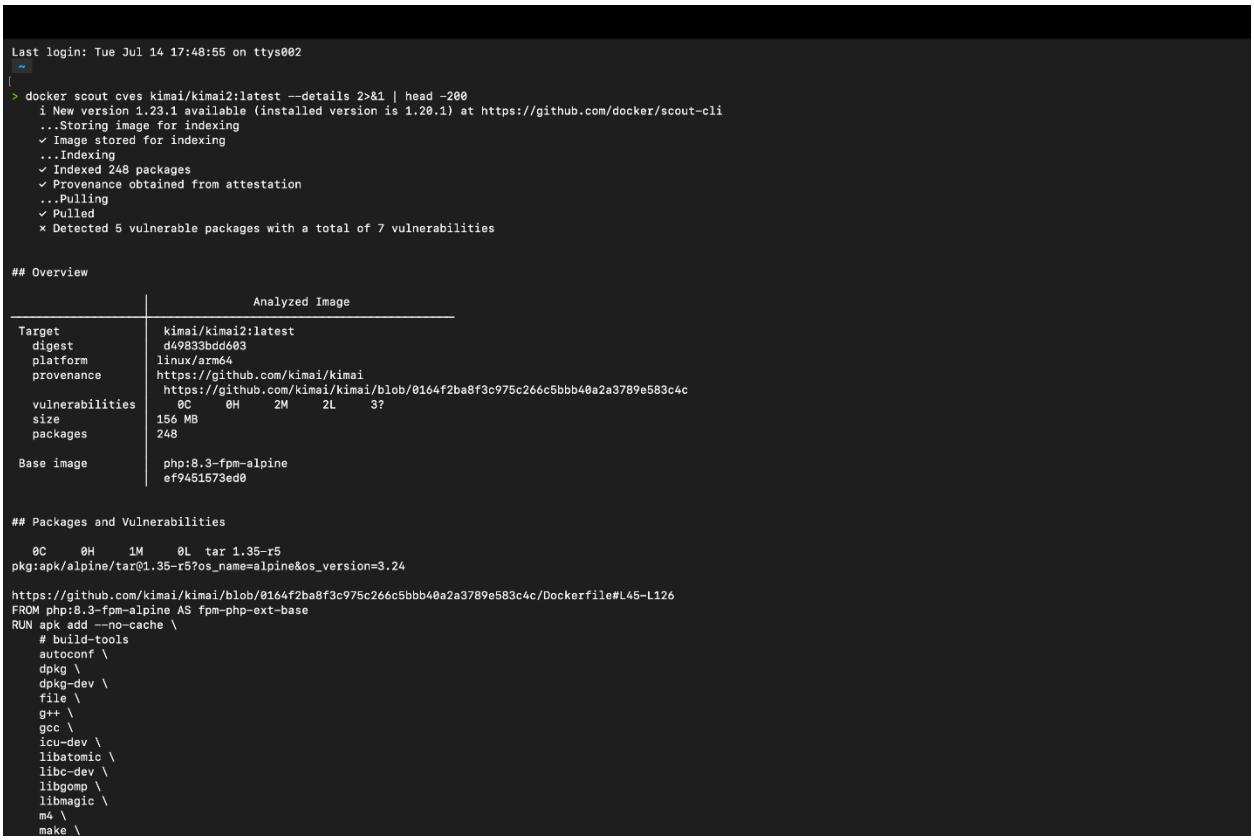


Figure 66. Docker Scout scan initialization and image metadata for kimai/kimai2:latest

Figure 66 shows Docker Scout indexing 248 packages in the kimai/kimai2:latest image and detecting 5 vulnerable packages totaling 7 vulnerabilities. The Overview table confirms the scanned image is built on php:8.3-fpm-alpine, an Alpine Linux base rather than the Debian base used by the :apache tag scanned elsewhere in this report, and is a much smaller image at 156 MB with 248 total packages, compared to the Debian-based image's larger package count reflected in the higher Snyk and Trivy totals in Sections 6.3 and 6.4.

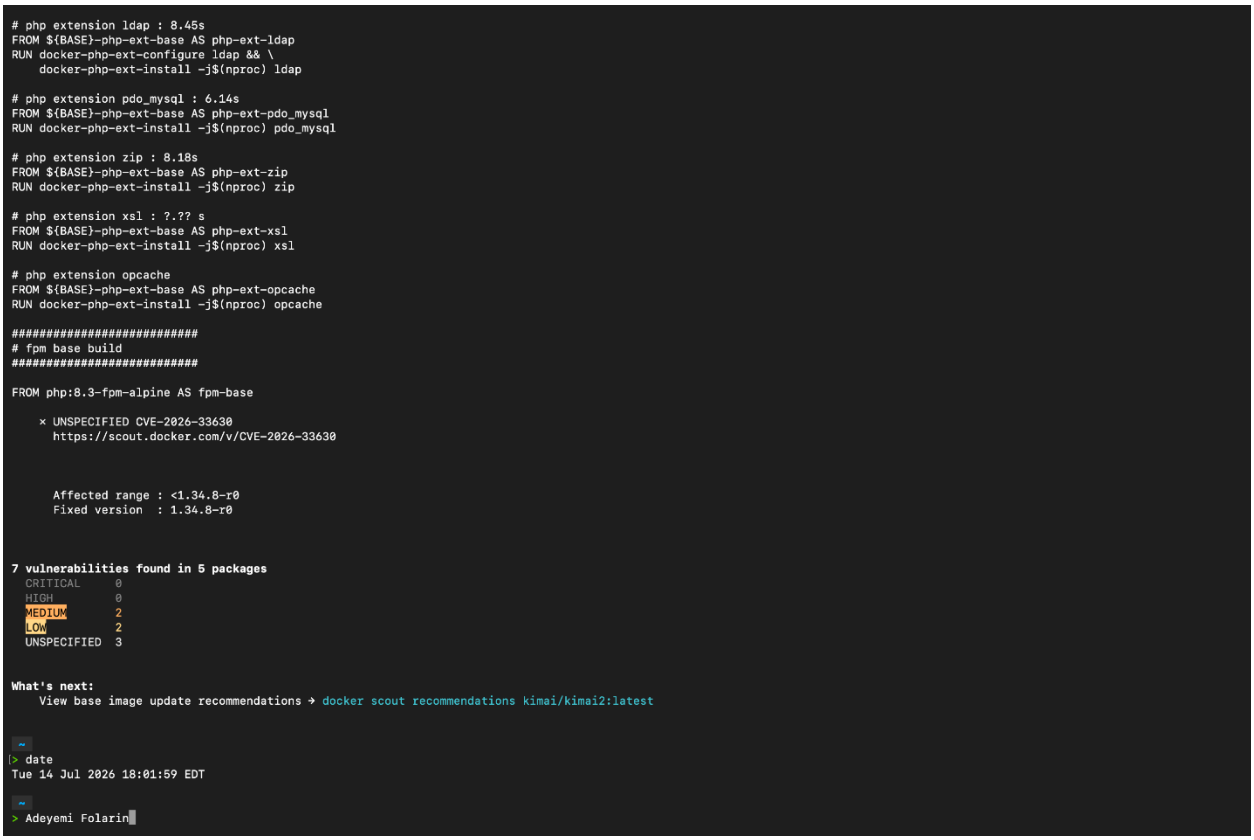


Figure 67 Docker Scout vulnerability summary and severity breakdown for kimai/kimai2:latest.

Figure 67 shows the final severity breakdown: 0 Critical, 0 High, 2 Medium, 2 Low, and 3 Unspecified, across 7 total vulnerabilities in 5 packages. One specific finding is shown in detail, CVE-2026-33630 in a php-fpm-alpine build stage, affecting versions before 1.34.8-r0 with a fix already available at that version. Because this scan targeted the :latest Alpine variant rather than the :apache Debian variant this deployment actually runs, these results should not be read as contradicting the higher vulnerability counts found by Snyk and Trivy in Sections 6.3 and 6.4; they reflect a genuinely different, smaller image.

Part 7: Sharing with Development Team

As required by the lab brief, the group's Security Deployment Guide was submitted directly to the Kimai project for review. Rather than a GitHub issue or pull request, the guide was submitted through Kimai's official support ticketing system (support@kimai.org), addressed to Kevin Papst, the project maintainer who had previously provided written approval for the group's use of Kimai as the assessment target.

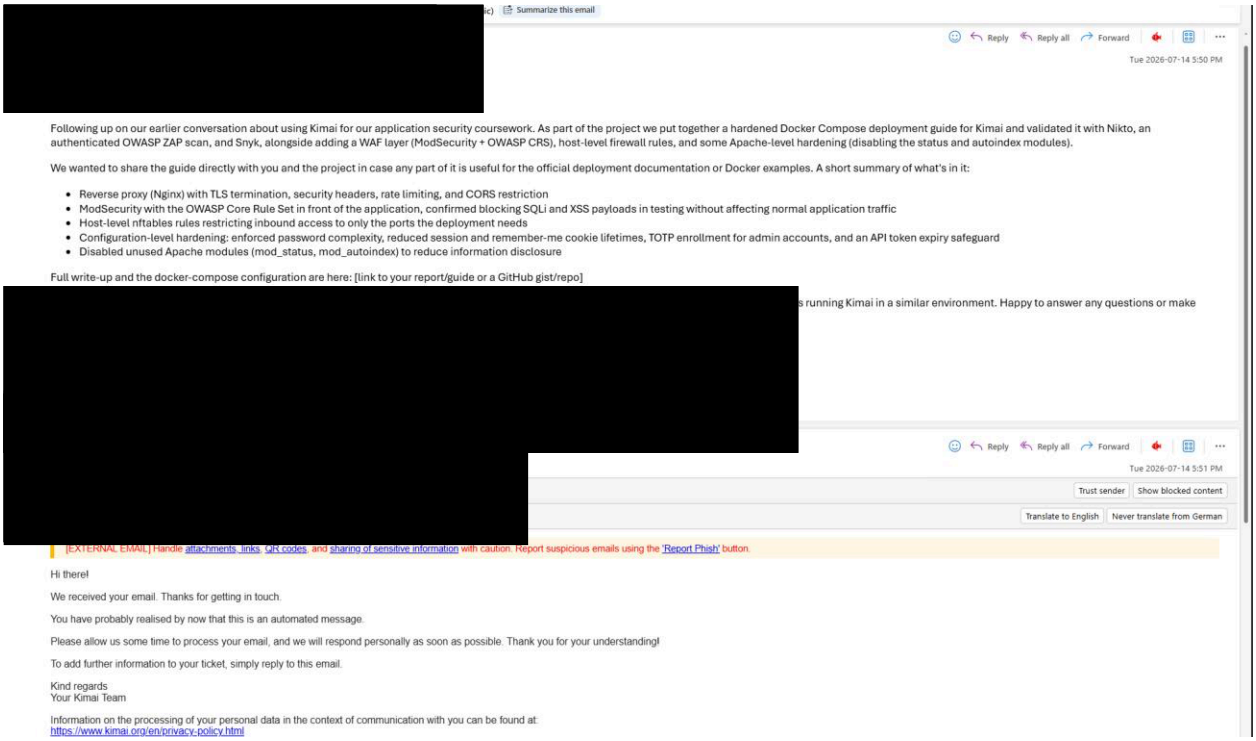


Figure 68 Submission of the Security Deployment Guide to the Kimai project support channel and the resulting automated ticket acknowledgment.

Figure 68 shows the email, sent July 14, 2026 at 5:50 PM and logged as Ticket #602928, summarizes the group's hardening work for the maintainer: the Nginx reverse proxy with TLS termination and rate limiting, the ModSecurity/OWASP CRS WAF layer with confirmed SQLi and XSS blocking, host-level nftables firewall rules, configuration-level hardening (password complexity, session and remember-me lifetime reduction, TOTP enforcement, API token expiry), and disabled Apache modules. The email explicitly frames this as a reference deployment offered for the maintainer's consideration, not a demand for a required change, consistent with responsible, collaborative disclosure practice.

Kimai Support's system responded one minute later, at 5:51 PM, confirming receipt under the same ticket number. As the reply itself states, this is an automated acknowledgment, not a personal response from Kevin Papst; the message explicitly notes it was auto-generated and that "we will respond personally as soon as possible." As of this report's submission, no personal reply has been received. This may be attributable to time zone differences, since prior correspondence indicated the maintainer is based in Germany. The group will continue to monitor the ticket and will share any further response received, including during the in-class demonstration if one arrives before then.

This submission satisfies the brief's requirement to share the deployment guide with the project's maintainers and to provide proof of submission; the automated acknowledgment and ticket number serve as that proof. A personal maintainer response, while hoped for, is outside the group's control and had not arrived as of the time of writing.

Mapping Lab 2 STRIDE Threats to Lab 5 Hardening Outcomes

The Lab 2 threat model identified nine prioritized threats in the PASTA risk matrix. The table below maps each threat to the testing performed in Labs 3 and 4, the hardening applied in Lab 5, and the current status.

Table 2 Mapping Lab 2 STRIDE Threats to Lab 5 Hardening Outcomes

Lab 2 Threat	Risk	Lab 3/4 Testing	Lab 5 Hardening	Status
IDOR on timesheet API	Critical	Lab 3: alice vs bob BOLA test returned 403. Lab 4: PATCH and DELETE on another user's timesheet returned 403.	Patched in Kimai 2.56.0. Deployment runs 2.59.0.	Resolved in application code
SSTI via invoice template (CVE-2023-46245)	Critical	Lab 3: Twig sandbox tested, StrictPolicy enforced.	Sandbox bypass confirmed for app.user and app.request methods (Sections 3.9, 3.10). CSP blocks client-side exfiltration. Cannot be fixed without modifying StrictPolicy.php.	Partially mitigated, accepted risk
Stored XSS via team alias (CVE-2026-40479)	High	Lab 3: XSS payloads rejected by character filter on name field, output-encoded on description field.	Patched in Kimai 2.53.0. Deployment runs 2.59.0.	Resolved in application code
Non-expiring API tokens	High	Lab 4: Confirmed NULL expires_at in database.	Token expiry set to 90 days (Section 3.4). Enforcement script created for ongoing enforcement.	Mitigated via workaround
Credential stuffing on login	High	Lab 3: Login throttling confirmed on 2.59.0. Lab 4: Default credentials confirmed.	Admin password changed (Section 3.1). TOTP enabled for admin (Section 3.6). Password complexity enforced (Section 3.5).	Mitigated
Session hijack over HTTP (no TLS)	High	Lab 3: HttpOnly and SameSite=Lax confirmed but no Secure flag. Lab 4: 7-day session lifetime confirmed.	TLS via Nginx (Section 2.1). Secure flag added (Section 2.1). Session lifetime reduced to 1 hour (Section 3.2). Remember-me reduced to 1 hour (Section 3.11).	Resolved
Container image CVEs	Medium	Lab 4: Snyk identified Critical phpspreadsheet, Medium mpdf and dompurify.	Lab 5 Snyk re-scan confirms 286 issues persist (Section 6.3). Cannot be fixed at deployment level.	Accepted risk, upstream dependency
APP_SECRET placeholder in .env	Medium	Lab 4: Template .env with placeholder confirmed in container.	Secrets separated to .env with chmod 600 (Section 3.7). Runtime APP_SECRET generated by container entrypoint.	Mitigated
DoS via PDF generation flood	Medium	Not tested (deprioritized for stability).	API rate limiting at 5 req/s per IP via Nginx (Section 2.1). Applies to /api/ path which includes invoice generation.	Partially mitigated

Known Vulnerability Coverage (from Lab 1)

All security advisories documented in the Kimai GitHub advisory history that predate the tested release are patched in version 2.59.0. No new unpatched advisories were identified during Lab 5 testing that apply to the deployed version.

Gap Analysis and Additional Controls

The following gaps remain after all hardening actions applied in Lab 5.

Remaining Gap 1: Plaintext API Token Storage at Rest (Medium, A04:2025 Cryptographic Failures)

API tokens are stored as plaintext strings in the kimai2_access_token table. A database compromise would yield directly usable authentication credentials. This was identified in Lab 4 (Section 3.4) and remains unchanged because token storage is a Kimai application design decision, not a deployment configuration. The recommended fix is for the upstream developer to store tokens as SHA-256 hashes and compare hashes on authentication.

Remaining Gap 2: Admin Password in Container Startup Logs (Low, A02:2025 Security Misconfiguration)

The Kimai Docker entrypoint script passes the admin password as a command-line argument to the kimai:user:create console command on every container startup, causing it to appear in plaintext in container logs. This was identified in Section 1.3 and cannot be suppressed without modifying the entrypoint script. The recommended mitigation is to restrict access to the Docker daemon and container logs, and to rotate the admin password if logs have been exposed.

Remaining Gap 3: Twig Sandbox Bypass for User and Session Data (Low, A05:2025 Injection)

The Twig sandbox StrictPolicy allows app.user methods (getUserIdentifier, getEmail, getRoles) and app.request.cookies.get(), enabling server-side extraction of user information and session cookies through malicious invoice templates (Sections 3.9, 3.10). The CSP header blocks client-side exfiltration, and template upload requires Docker filesystem access (not available through the UI), which limits exploitability. The fix requires adding app.request and restricting app.user methods in StrictPolicy.php, which is a source code change outside the scope of deployment hardening.

Remaining Gap 4: Containers Running as Root (Low, A02:2025 Security Misconfiguration)

Both the Kimai and MySQL containers run all processes as root (uid=0), as confirmed in Section 1.15. The official kimai/kimai2:apache image does not support running as a non-root user without building a custom image. The recommended approach for production is to build a custom Docker image with a non-root application user, or to use Podman with rootless containers.

Remaining Gap 5: No Mandatory 2FA Enforcement (Low, A07:2025 Authentication Failures)

TOTP was enabled for the admin account (Section 3.6), but Kimai provides no configuration option to require 2FA for all users or specific roles. Enrollment must be performed per account. Organizational policy should mandate TOTP enrollment for all privileged accounts.

Remaining Gap 6: No Permanent API Token Expiration Enforcement (Low, A07:2025 Authentication Failures)

Kimai does not provide a configuration setting to require token expiration at creation time. The enforcement script (Section 3.4) is a workaround that sets a 90-day expiry on tokens with NULL expires_at. This requires scheduling as a cron job and does not prevent a window between token creation and the next script run. The recommended fix is for the upstream developer to add a mandatory expiration field in the token creation form.

Remaining Gap 7: Outdated Dependencies with Known CVEs (A03:2025 Software Supply Chain Failures)

The Snyk container scan (Section 6.3) identified 286 vulnerabilities in the Debian base image and application dependencies, including Critical findings in zlib, gnutls28, and phpoffice/phpspreadsheet. These cannot be resolved through deployment configuration and require the image maintainer to update the base image and the Kimai developer to upgrade locked dependencies.

Summary

Of the nine prioritized threats from Lab 2, three are fully resolved (IDOR, stored XSS, session hijack), three are mitigated through deployment controls (credential stuffing, non-expiring tokens, APP_SECRET), one is partially mitigated (DoS via rate limiting), and two remain as accepted risks requiring upstream changes (SSTI sandbox bypass, container image CVEs). Of the seven gaps from the Lab 4 gap analysis, four were fully resolved (TLS deployed, password policy enforced, API rate limiting added, session lifetime reduced), and three remain as documented limitations that cannot be fixed at the deployment level (plaintext API token storage, admin password in container logs, outdated dependencies). Seven remaining gaps are documented above, all at Low severity or requiring source code changes outside the scope of deployment hardening. The hardened deployment substantially reduces the attack surface established in the pre-hardening baseline while clearly documenting the residual risks that require upstream cooperation.

Appendix A: Generative AI Usage Disclosure

In accordance with the WAS705 GenAI policy, this appendix discloses the use of generative AI tools in the preparation of this deliverable. Generative AI content constitutes no more than 25 percent of the text.

Tool used: Anthropic Claude. How it was used: To help with evidence gathering and asking questions on how to solve a problem when we were stuck throughout the lab.

Verification note: the OWASP Top 10 2025 categories and CVE-2023-46245 and CVE-2026-40479 were verified against public advisory sources during preparation. Group members are responsible for confirming any remaining advisory identifiers and for completing this appendix with a full account of any AI use during the original testing and hardening work.