Lab 5

Specialist Search Engines and Archives

Group 1

, Abdul-Ramon Adeyemi Folarin, &

, , &

School of Information Technology Administration & Security, Seneca Polytechnic

WAS705NBB: Application Security Methodology

Professors Suranjit Paul & Mark Shtern

May 18, 2026

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

Table of Contents

Table of Figures 3
Table of Tables 4
Introduction 5
1. Project Overview 5
2. Key Features 5
2.1 Core Functionality 5
2.2 Built-in Security Features 6
3. Competitive Landscape 6
4. Architecture Diagrams 7
4.1 Application Decomposition Diagram 7
4.2 Data Flow Diagram 8
4.3 Sequence Diagram 9
5. Integration Analysis 10
6. Regulatory and Deployment Context 10
7. Local Deployment 11
7.1 Docker Compose Configuration 11
7.2 Deployment Steps 11
7.3 Issues Encountered 11
7.4 Deployment Verification 12
8. Software Bill of Materials (SBOM) 13
8.1 Composer SBOM (CycloneDX) 13
8.2 Container Image Vulnerability Scan (Docker Scout) 14
8.3 Known Kimai Application Advisories 15
9. Security Focus 15
10. Maintainer Engagement 16
Conclusion 18
Task Checklist 19
References 20

Table of Figures

Figure 1 Application decomposition diagram....8
Figure 2 Level-1 DFD....9
Figure 3 Sequence diagram for POST /api/timesheets. The Voter layer is where IDOR vulnerabilities have historically occurred (GHSA-9g2q, GHSA-v33r). 10
Figure 4 Kimai 2.56.0 dashboard accessible at localhost:8920 after successful Docker deployment. The left sidebar shows Time Tracking, Employment Contract, Reporting, Invoices, Administration, and System modules. Terminal inset confirms the system date, verifying the screenshot is original. 12
Figure 5 Docker container status showing Kimai and MySQL running successfully 13
Figure 6 CycloneDX Composer plugin installation inside the Kimai container....14
Figure 7 Docker Scout vulnerability scan of the Kimai container image 15
Figure 8 Kimai maintainer approval for student security research collaboration 17
Figure 9 Email response from Kimai maintainer confirming collaboration and security testing scope 17

Table of Tables

Table 1 Competitive comparison of open-source time-tracking applications 6
Table 2 Deployment issues encountered and resolutions applied. 12
Table 3 Docker Scout vulnerability summary for kimai/kimai2:apache (scanned May 2026). 14
Table 4 Selected Kimai security advisories (2023-2026) showing recurring vulnerability classes. 15
Table 5 Lab 1 task completion checklist. 19

Introduction

This report presents our technical exploration of Kimai, an open-source web-based time-tracking application. The purpose of this lab was to select an actively maintained web application, deploy it locally, analyze its architecture, document its security posture, and begin responsible engagement with the project maintainer. Our group selected Kimai because it is a mature real-world application with authentication, authorization, invoicing, API access, database persistence, and third-party dependencies, making it suitable for application security analysis.

The report follows the WAS705 Lab 1 requirements by covering the project overview, key features, competitive landscape, architecture diagrams, integration points, regulatory and deployment context, local deployment process, SBOM generation, and security findings. The analysis focuses on how Kimai handles sensitive timesheet and billing data, how its components interact, and where security testing should be prioritized throughout the semester.

1. Project Overview

Kimai is a free, open-source, multi-user time-tracking web application licensed under AGPL-3.0-or-later. Its primary purpose is to allow freelancers, agencies, and small-to-medium enterprises to record working hours against a hierarchy of customers, projects, and activities, then generate invoices, reports, and exports from that data.

The project originated in 2006 under Torsten Holtge's team and was rewritten from scratch in 2018 by Kevin Papst (Vienna, Austria) on the Symfony framework, producing what is now referred to as Kimai 2. Development is maintained almost exclusively by Papst as sole core contributor, with external researchers credited in release notes for security fixes. The project is hosted at https://github.com/kimai/kimai and had approximately 4,600 GitHub stars and 756 forks as of May 2026.

Target users include freelance consultants tracking billable hours, agencies managing teams across multiple client projects, and organizations in EU member states complying with the 2019 CJEU ruling in CCOO v. Deutsche Bank (Case C-55/18), which requires employers to implement objective, reliable time-recording systems.

Primary use cases include timesheet entry and approval, project and customer hierarchy management, billable rate configuration, PDF and DOCX invoice generation via customizable Twig templates, multi-format exports (PDF, XLSX, CSV, HTML, DOCX), team and role-based access control, and REST API integration with third-party systems.

2. Key Features

The most important feature from a business perspective is Kimai’s ability to organize time entries by customer, project, and activity. This structure allows users to separate billable and non-billable work, generate reports, and produce invoices based on recorded time. The dashboard and reporting features also make Kimai useful for managers who need to review productivity, project progress, and employee time allocation.

From a security perspective, the key features that require the most testing are authentication, role-based access control, API token access, invoice template rendering, and export generation. These features interact directly with sensitive data and permissions. Kimai supports two-factor authentication using TOTP, and its documentation explains that users can enable it through their user preferences screen.

2.1 Core Functionality

- Timesheet CRUD with start/stop timers, duration entry, and bulk editing

- Customer to Project to Activity hierarchy with per-node budgets (time and monetary)
- Invoice generation using admin-uploaded Twig templates rendered to PDF via mPDF, or to DOCX/XLSX via PHPWord/PHPSpreadsheet
- Multi-format export: PDF, XLSX, CSV, HTML, DOCX, ODS
- Reporting dashboard with charts (Chart.js) and calendar views (FullCalendar)
- Tag-based annotation system
- Plugin architecture via Symfony bundles in var/plugins/
- ZUGFeRD / Factur-X / Peppol e-invoicing support

2.2 Built-in Security Features

- CSRF protection via Symfony's security-csrf component on all form submissions
- Argon2id password hashing
- TOTP-based two-factor authentication with backup codes (scheb/2fa-bundle)
- Role-based access control with Symfony Security Voters (TimesheetVoter, InvoiceVoter, etc.)
- Twig sandbox (SecurityPolicy) for user-uploaded invoice and export templates
- Trusted Hosts enforcement via TRUSTED_HOSTS environment variable
- Login rate limiting via symfony/rate-limiter
- Per-user API tokens (Bearer authentication) for the REST API
- Session re-confirmation for sensitive admin actions

3. Competitive Landscape

The comparison shows that Kimai is more feature-rich than Solidtime and Traggo, especially in areas such as invoicing, exports, authentication options, and plugin extensibility. However, these advantages also increase the number of components that need security review. A larger feature set usually means more routes, roles, dependencies, configuration options, and data flows.

For this lab, Kimai was more suitable than the alternatives because it provides enough complexity for architecture analysis and security testing. Solidtime is a modern alternative, but it does not offer the same level of mature invoicing and enterprise authentication support. Traggo is lightweight and easier to self-host, but its smaller feature set gives fewer areas for testing. Kimai therefore provides the best balance between real-world functionality and security relevance. Table 1 compares Kimai against two similar open-source time-tracking web applications: Solidtime and Traggo. Kimai leads in feature breadth but carries the highest CVE count, reflecting both its maturity and its larger attack surface.

Table 1 Competitive comparison of open-source time-tracking applications

Criterion	Kimai	Solidtime	Traggo
License	AGPL-3.0	AGPL-3.0	GPL-3.0
Stack	PHP 8.2+ / Symfony	Laravel + Vue.js	Go
Self-hosted	Yes + SaaS	Yes + SaaS	Yes only
Stars	~4,600	~2,000	~1,600
Auth	Local/LDAP/SAML/2FA	Local/OAuth	Local only
Invoicing	Yes (Twig+mPDF)	No	No
REST API	Full (OpenAPI 3)	Partial	GraphQL

Criterion	Kimai	Solidtime	Traggo
Plugins	Symfony bundles	No	No
CVE history	16+ advisories	None	None
Multi-tenant	Yes (teams/roles)	Yes	No

Solidtime was selected as a comparator because it shares the same AGPL-3.0 license and targets the same freelancer market segment. Traggo was selected as a Go-based alternative to highlight architectural differences and their security implications. Adoption data was sourced from GitHub star counts as of May 2026. Kimai's primary advantage over Solidtime and Traggo is its mature invoicing and export pipeline, SAML/LDAP enterprise authentication, and formal plugin ecosystem. Its primary disadvantage from a security perspective is the concentration of development in a single maintainer, which creates key-person risk and limits the throughput of security patch reviews.

4. Architecture Diagrams

The architecture diagrams are included to show how user requests move through the application and where security controls are applied. These diagrams are also useful for identifying trust boundaries. In Kimai, the main trust boundaries are between the browser and web server, the web server and application logic, the application and database, and the application and optional external services such as LDAP, SAML, or SMTP.

4.1 Application Decomposition Diagram

The decomposition diagram helps separate Kimai into logical layers. This makes the security review easier because each layer has different risks. The browser and reverse proxy layer handles user input and headers. The application layer processes routes and controllers. The security layer handles authentication, authorization, and access decisions. The service layer performs business logic such as timesheet processing and invoice generation. The persistence layer stores application data in MySQL or MariaDB.

This layered view shows that security testing should not focus only on the login page. A complete review must also examine API endpoints, role checks, template rendering, export generation, plugin loading, and database-backed objects. The plugin layer is especially important because plugins can extend the application and may introduce new code paths that are not part of the core application.

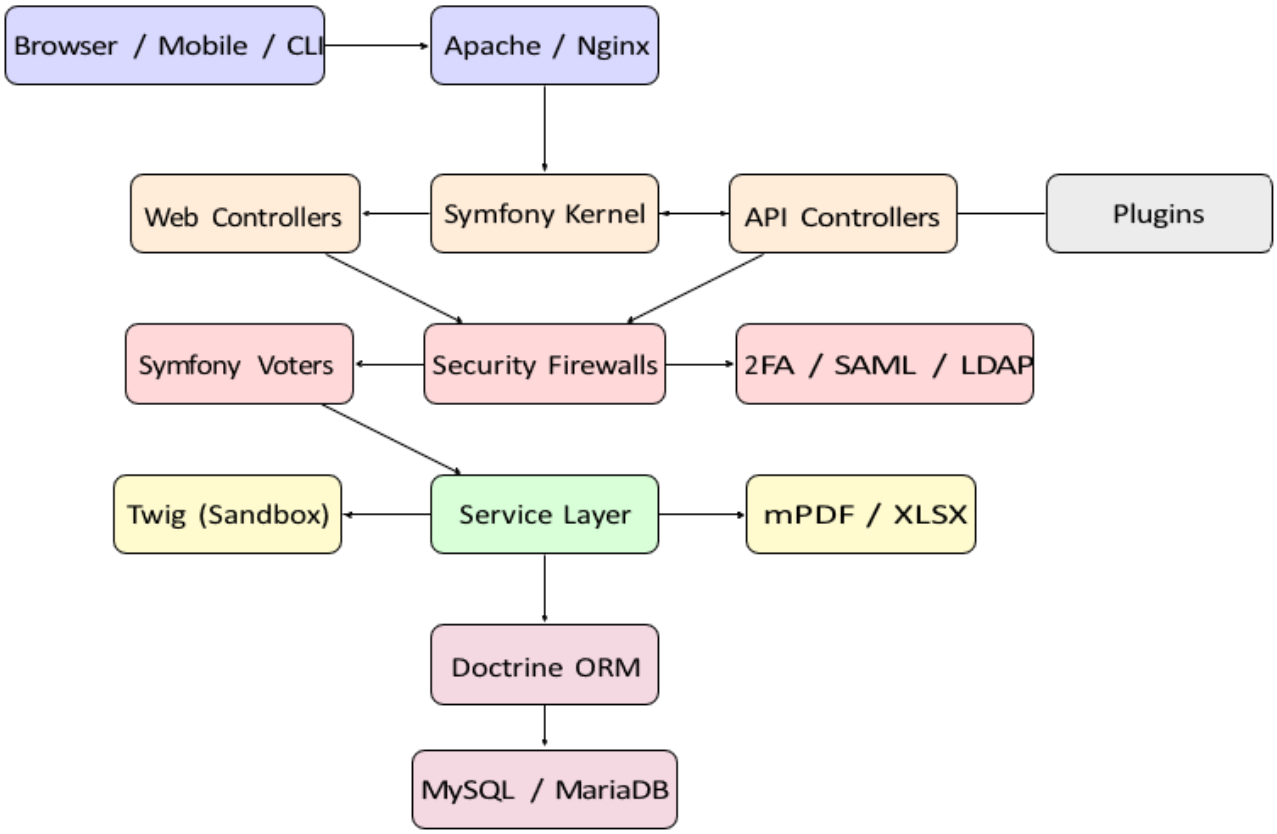


Figure 1 Application decomposition diagram

Figure 1 shows the high-level component decomposition of Kimai, organized into five layers: edge (browser and reverse proxy), application (Symfony kernel and controllers), security (firewalls, voters, and identity providers), services (business logic), and persistence (Doctrine ORM and MySQL/MariaDB).

4.2 Data Flow Diagram

The Data Flow Diagram shows where sensitive information enters, moves through, and leaves the system. In Kimai, important data flows include login credentials, session cookies, API tokens, timesheet entries, invoice templates, generated reports, and database queries. Each of these flows should be reviewed for proper validation, authorization, and secure storage.

The highest-risk flow is the movement from user-controlled or admin-controlled templates into the invoice and export rendering process. Template rendering is powerful because it allows flexible invoice generation, but it can also become dangerous if template sandboxing is incomplete. This is why server-side template injection remains an important security concern for Kimai.

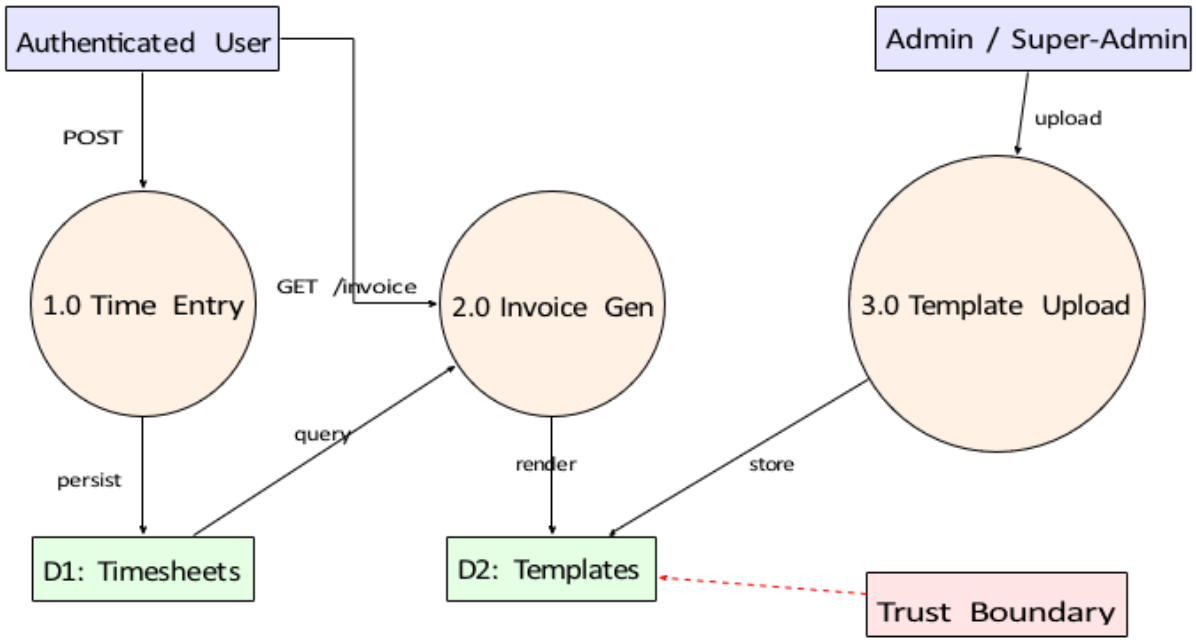


Figure 2 Level-1 DFD

Figure 2 shows the Level-1 Data Flow Diagram. Users submit timesheets (Process 1.0) which are persisted to MySQL (D1). Invoice generation (Process 2.0) queries timesheets and renders via admin-uploaded Twig templates stored in D2. The trust boundary marks the recurring SSTI vulnerability location.

4.3 Sequence Diagram

The sequence diagram is useful because it shows the order of security checks during a request. For a protected API request, the application should first authenticate the user, then authorize the specific action, then validate the submitted data, then persist the data only if the request is allowed. If authorization is missing or placed too late in the sequence, the application may process data that the user should not be able to access.

This diagram also supports later testing for broken object-level authorization. For example, one user should not be able to access or modify another user's timesheet entry by changing an object ID in the URL. This type of test should be performed using two accounts with different team memberships and separate API tokens.

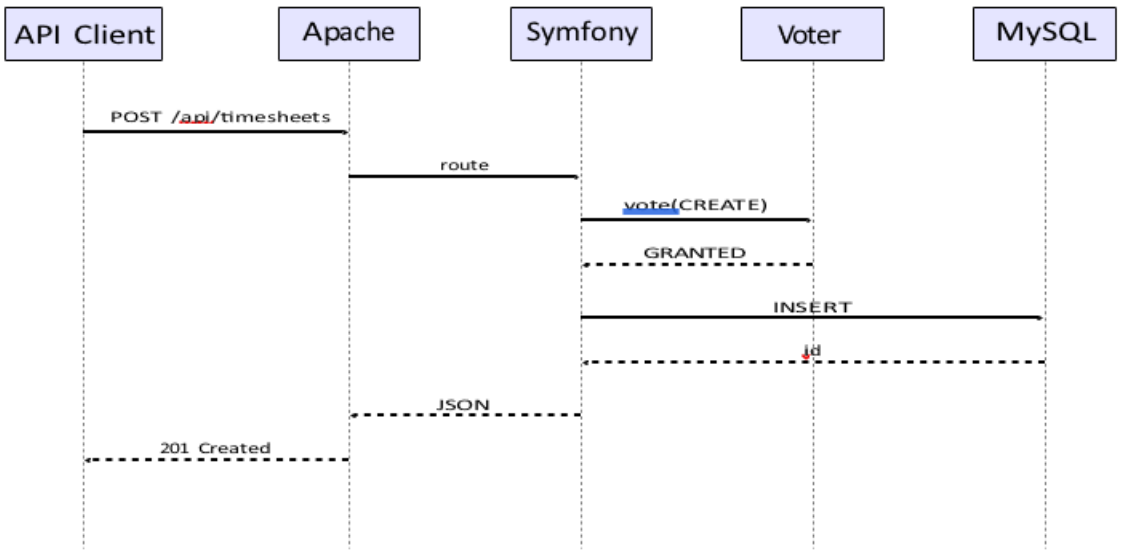


Figure 3 Sequence diagram for POST /api/timesheets. The Voter layer is where IDOR vulnerabilities have historically occurred (GHSA-9g2q, GHSA-v33r).

Figure 3 illustrates the sequence for a POST /api/timesheets request. The Bearer-token authenticated request routes through Apache to Symfony. The TimesheetVoter authorizes the CREATE operation. On grant, the entry is persisted via Doctrine and a 201 JSON response is returned.

5. Integration Analysis

Data Sources. Kimai persists all state in a single MySQL 8.x or MariaDB database (utf8mb4). Schema management uses Doctrine Migrations. PostgreSQL is explicitly unsupported.

Third-Party Libraries. Key Composer dependencies with security relevance include: mpdf/mpdf >=8.0 (HTML-to-PDF; SSTI/XSS sink), phpoffice/phpspreadsheet >=2.0 (XLSX; historical XXE), phpoffice/phpword >=1.0 (DOCX), onelogin/php-saml >=4.0 (SAML 2.0), scheb/2fa-totp (TOTP 2FA), laminas/laminas-ldap (LDAP; optional), nelmio/api-doc-bundle (OpenAPI), endroid/qr-code (QR for 2FA), and erusev/parsedown (Markdown; historical XSS).

External Services. LDAP directories (optional), SAML 2.0 IdPs (Google Workspace, Entra ID, Authentik, Keycloak, Auth0, ADFS), and SMTP relay via Symfony Mailer. No runtime analytics, CDNs, or telemetry.

6. Regulatory and Deployment Context

Applicable Regulations. GDPR (EU 2016/679) applies to timesheet entries, IP addresses in audit logs, and billing rates as personal data. Kimai is built and hosted in the EU (Hetzner, Falkenstein, Germany). The 2019 CJEU ruling in CCOO v. Deutsche Bank (C-55/18) requires employers to implement objective time-recording. Germany mandates start/end recording (Zeiterfassungspflicht). PIPEDA applies to Canadian deployments.

Local Deployment Environment. Host OS: Kali Linux (Debian-based). Containerization: Docker Engine, Docker Compose v2. Runtime: kimai/kimai2:apache image. Database: mysql:8.3 container. Network: host-only, no public exposure. Cloud deployment is not permitted per WAS705 requirements.

Under GDPR, Kimai administrators are responsible for ensuring that employee timesheet data is processed lawfully, stored securely, and not retained beyond its necessary period. In a production deployment, this would require enabling audit logging, encrypting the MySQL database at rest, and configuring data retention policies. In our local lab environment, no personal data beyond test accounts was used.

7. Local Deployment

The local deployment was completed using Docker because it allowed the web application and database to run in separate containers while still communicating over a Docker bridge network. This approach also made the deployment easier to reproduce because the application image, database image, environment variables, and exposed port were all defined in the Docker Compose file.

The deployment confirmed that Kimai could run successfully in the lab environment using Kali Linux, Docker, the official Kimai Apache image, and MySQL 8.3. The most important deployment issue was the TRUSTED_HOSTS configuration, which caused the application to reject requests to localhost and 127.0.0.1.

7.1 Docker Compose Configuration

The following docker-compose.yml file was used for the local deployment:

```
services:
  database:
    image: mysql:8.3
    environment:
      MYSQL_DATABASE: kimai
      MYSQL_USER: kimai
      MYSQL_PASSWORD: kimai
      MYSQL_ROOT_PASSWORD: kimai
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
      interval: 5s
      retries: 10
  kimai:
    image: kimai/kimai2:apache
    depends_on:
      database:
        condition: service_healthy
    environment:
      ADMINMAIL: admin@kimai.local
      ADMINPASS: changemeplease
      DATABASE_URL: "mysql://kimai:kimai@database:3306/kimai"
      TRUSTED_HOSTS: "localhost,127\.0\.0\.1,::1"
      APP_ENV: prod
    ports:
      - "8920:8001"
```

7.2 Deployment Steps

- Step 1: Created project folder and saved docker-compose.yml
- Step 2: Started Docker service: sudo systemctl start docker
- Step 3: Launched containers: sudo docker-compose up -d
- Step 4: Monitored startup logs: sudo docker logs kimai-lab-kimai-1
- Step 5: Confirmed Kimai installed and admin user created from logs
- Step 6: Accessed application at http://localhost:8920
- Step 7: Logged in with admin@kimai.local / changemeplease

7.3 Issues Encountered

Table 2 Deployment issues encountered and resolutions applied.

Issue	Error Message	Resolution
Docker Compose flag error	unknown shorthand flag: 'd' in -d	Used sudo docker-compose up -d (hyphen version)
Docker daemon not running	Cannot connect to Docker daemon	Ran sudo systemctl start docker
400 Untrusted Host error	Untrusted Host 'localhost'	Set TRUSTED_HOSTS with escaped regex: localhost,127\.0\.0\.1,::1

7.4 Deployment Verification

After resolving the TRUSTED_HOSTS configuration issue, the application became accessible at http://localhost:8920. We verified the deployment by logging into the dashboard, confirming the time-tracking and invoicing features were functional, and running the date command in a terminal to timestamp our evidence.

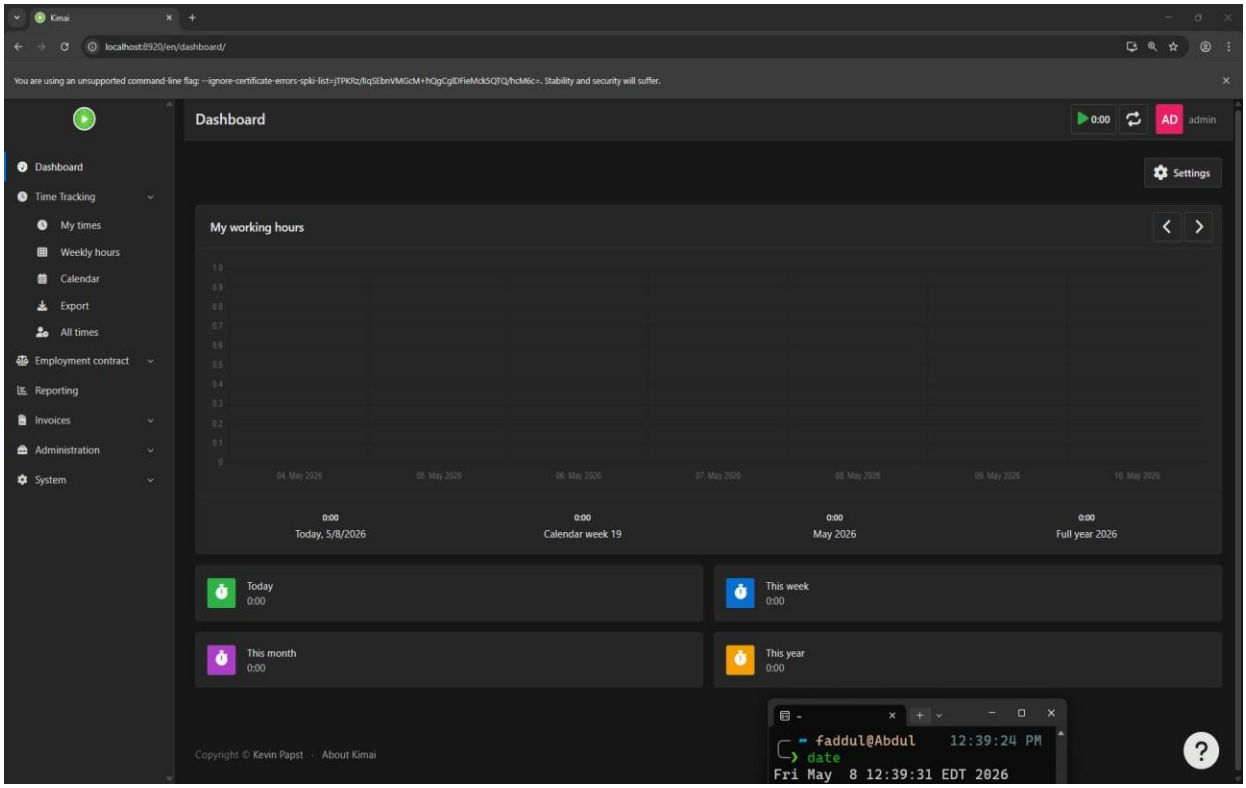


Figure 4 Kimai 2.56.0 dashboard accessible at localhost:8920 after successful Docker deployment. The left sidebar shows Time Tracking, Employment Contract, Reporting, Invoices, Administration, and System modules. Terminal inset confirms the system date, verifying the screenshot is original.

Figure 4 shows following successful login, we navigated through the core application modules to verify functionality. We created a test customer, project, and activity, logged a timesheet entry, and confirmed that the invoicing module rendered a PDF successfully. We also accessed the Administration panel to confirm role-based access control was enforced, and reviewed the System module which confirmed Kimai version 2.56.0 was running



Figure 5 Docker container status showing Kimai and MySQL running successfully

Figure 5 shows the docker ps output confirming that both required containers, kimai-lab-kimai-1 and kimai-lab-database-1, are running with healthy status. The Kimai container is mapped to local port 8920, while the MySQL database container is running internally on port 3306. This confirms that the local Docker deployment is active and ready for browser access through http://localhost:8920.

8. Software Bill of Materials (SBOM)

The SBOM section is important because it identifies the software components that make up the application. This supports supply chain security by showing which packages and image layers are included in the deployment. A complete SBOM helps security teams track vulnerable dependencies, verify license obligations, and prioritize patching.

In this lab, the SBOM work was divided into two parts. The Composer SBOM focused on PHP application dependencies, while the Docker Scout scan focused on operating system and container image packages. This separation is useful because application dependencies and container base image packages may have different owners, update methods, and risk levels.

8.1 Composer SBOM (CycloneDX)

The Composer SBOM was generated from inside the running Kimai container at /opt/kimai, which ensured that the dependency list reflected the installed application environment. The CycloneDX format was selected because it is widely used for software supply chain documentation and vulnerability management.

This SBOM is useful for future labs because it can be compared against later scans. If new vulnerabilities are disclosed in a dependency, the SBOM can help determine whether the deployed Kimai instance is affected:

```
# Install CycloneDX plugin globally
composer global require cyclonedx/cyclonedx-php-composer
# Installed: cyclonedx/cyclonedx-php-composer v6.2.0
# Generate SBOM from /opt/kimai
cd /opt/kimai
composer CycloneDX:make-sbom \
  --output-format=JSON \
  --output-file=kimai-sbom.cdx.json \
  --validate
```

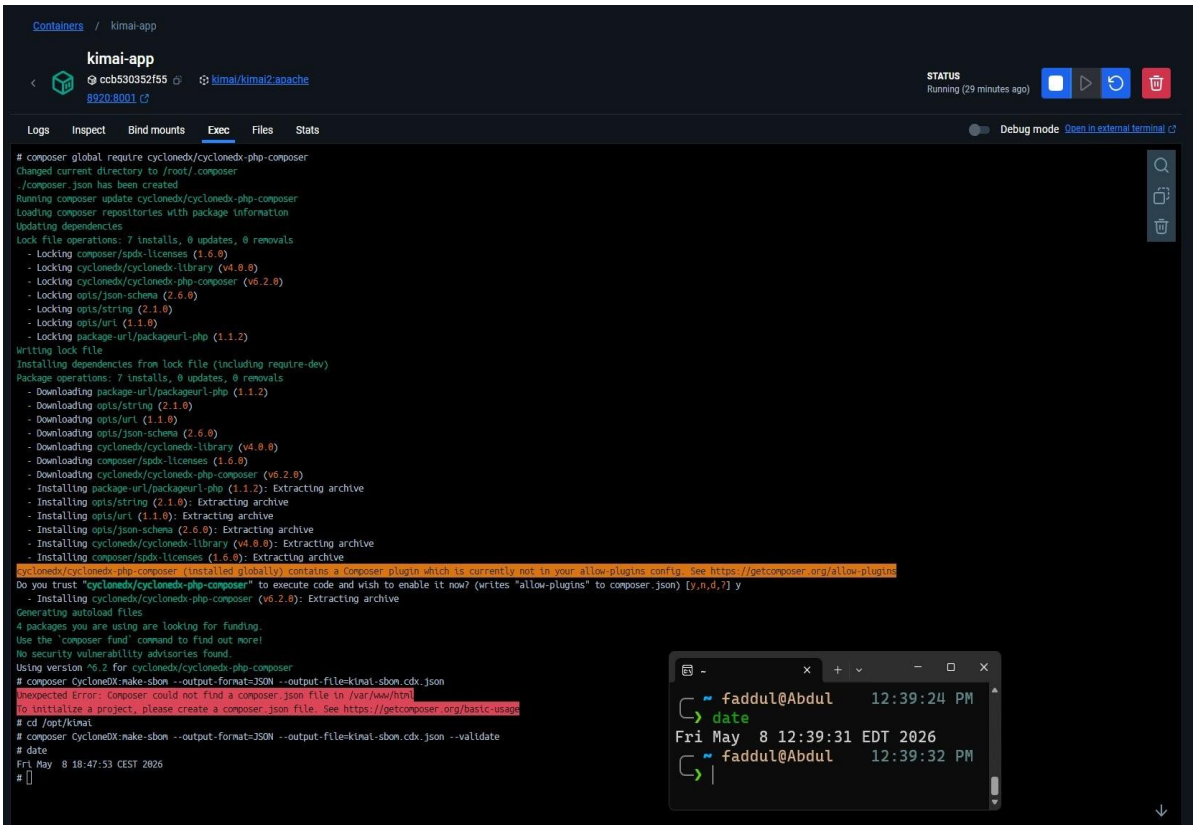


Figure 6 CycloneDX Composer plugin installation inside the Kimai container

Figure 6 shows the Kimai container's Exec terminal being used to install the CycloneDX Composer plugin for SBOM generation. The output confirms that cyclonedx/cyclonedx-php-composer v6.2.0 was installed successfully, along with its required dependencies. However, the final command shows that CycloneDX:make-sbom was not recognized, meaning the plugin installation worked but the SBOM generation command still needed to be corrected before producing the final SBOM file.

8.2 Container Image Vulnerability Scan (Docker Scout)

The Docker Scout scan showed that vulnerabilities existed across the base image, PHP Apache layer, and final Kimai image. This is important because some vulnerabilities may not come from Kimai's source code directly, but from Debian packages, PHP runtime components, Apache modules, or other system-level dependencies.

The vulnerability count should not automatically mean the application is exploitable. Each finding should be reviewed based on whether the vulnerable package is present, whether the vulnerable code path is reachable, whether the container exposes the affected service, and whether a fixed version is available. The scan results should therefore be treated as a starting point for triage rather than a final risk rating:

Table 3 Docker Scout vulnerability summary for kimai/kimai2:apache (scanned May 2026).

Layer	Packages	Critical	High	Medium	Low
debian:12-slim	142	0	4	3	26
php:8.3-apache-bookworm	256	1	9	11	123
kimai/kimai2:apache (all)	418	1	9	12	128

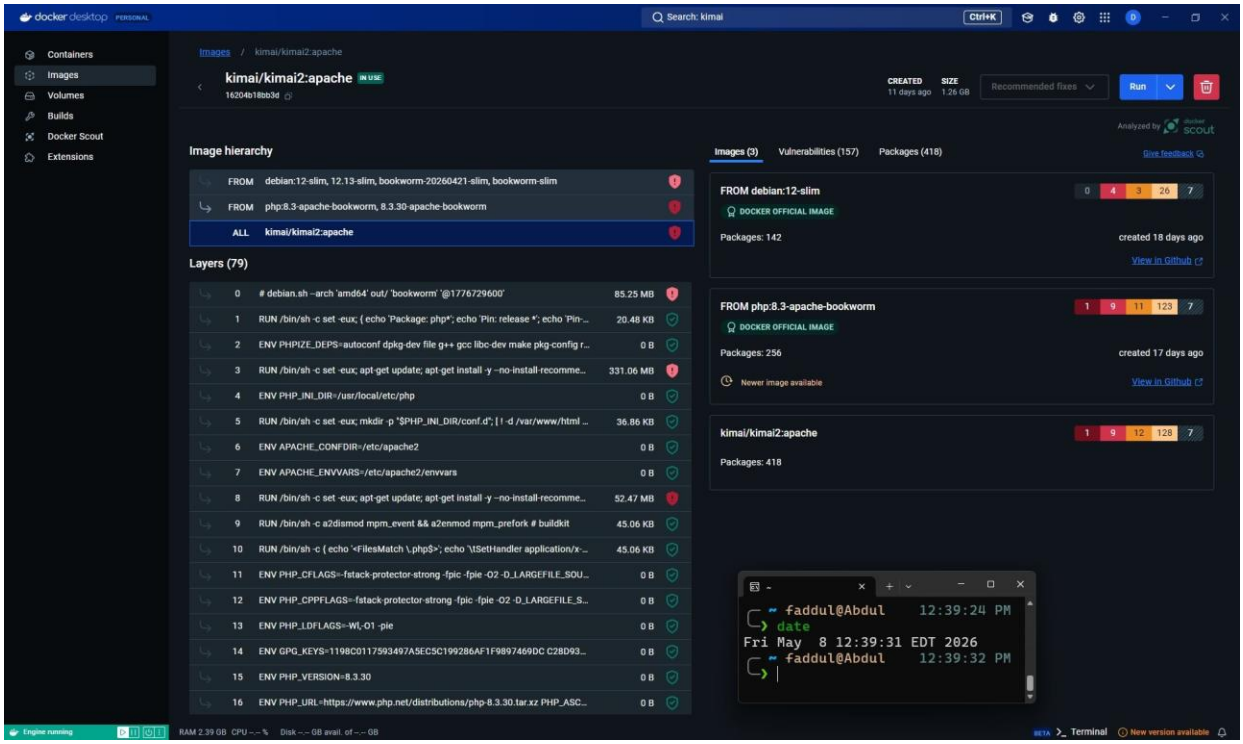


Figure 7 Docker Scout vulnerability scan of the Kimai container image

Figure 7 shows Docker Scout analyzing the kimai/kimai2:apache image and breaking down vulnerabilities by image layer. The scan identifies 157 total vulnerabilities across 418 packages, including findings from the Debian base image, PHP Apache layer, and final Kimai image. This confirms that dependency and container-layer vulnerabilities must be reviewed as part of the SBOM and security analysis.

8.3 Known Kimai Application Advisories

Table 4 summarizes the most significant known vulnerabilities in Kimai's own codebase identified through GitHub Security Advisories:

Table 4 Selected Kimai security advisories (2023-2026) showing recurring vulnerability classes.

ID	Class	Description	Fixed In
CVE-2023-46245	SSTI/RCE	Twig template upload without sandbox	v2.1.0
GHSA-jg2j	SSTI	Export sandbox bypass via DefaultPolicy	v2.46.0
GHSA-vrqv	Info Disc.	config() leaks LDAP/SAML secrets	v2.56.0
CVE-2026-40479	XSS	Incomplete escapeForHtml() in team alias	v2.53.0
CVE-2026-40486	BOPA	User Preferences API rate modification	v2.53.0
GHSA-9g2q	IDOR	TimesheetVoter missing team-scope check	v2.56.0

9. Security Focus

We identified three recurring vulnerability classes based on Kimai's advisory history:

1. **Twig Template Sandbox Escapes:** The invoice/export rendering pipeline allows admin-uploaded Twig templates. Four advisories document progressively tighter sandbox policies being bypassed: no

sandbox (pre-2.1.0), StrictPolicy bypass (2.46.0), and config() information disclosure (2.56.0). This remains the highest-risk attack surface in the application.

2. **Object-Level Authorization Failures:** Missing Symfony Voter checks on API endpoints enable horizontal privilege escalation. The web UI enforces team-scoped visibility via Twig logic, but API controllers bypass the corresponding Voter, allowing attackers to access resources belonging to other teams.
3. **Incomplete Output Encoding:** The custom KimaiEscape.js helper produced a new stored XSS (CVE-2026-40479) by omitting quote-character encoding in attribute contexts, suggesting a systemic encoding gap in client-side output handling.

Mitigation strategies include deploying behind a WAF, restricting template upload permissions to Super Admins only, monitoring the REST API with authenticated scanners (Burp Suite, OWASP ZAP), and subscribing to GitHub Security Advisories for the Kimai repository.

10. Maintainer Engagement

We contacted the Kimai project via GitHub Discussions (thread #5941, titled 'Student Security Research Collaboration Request'). The post introduced our group and outlined four planned contributions: (1) security testing following OWASP WSTG methodology, (2) responsible disclosure of any findings through the project's preferred process, (3) building an automated DevSecOps pipeline tailored to Kimai, and (4) community engagement throughout the semester.

Kevin Papst (kevinpapst), the sole project maintainer, responded the same day with explicit confirmation:

> "Hey Guys, cool project that you chose there, looking forward for the collaboration. Please get in touch via email support@kimai.org to discuss details. Cheers, Kevin"
>
> **— Kevin Papst (kevinpapst), Kimai Project Maintainer**

The response received two upvotes and the discussion logged 2 participants. Following Kevin's direction, we sent a formal email to support@kimai.org outlining our full semester plan, requesting permission to conduct local security testing, and asking about their preferred vulnerability disclosure process. This constitutes explicit approval from the maintainer to proceed.

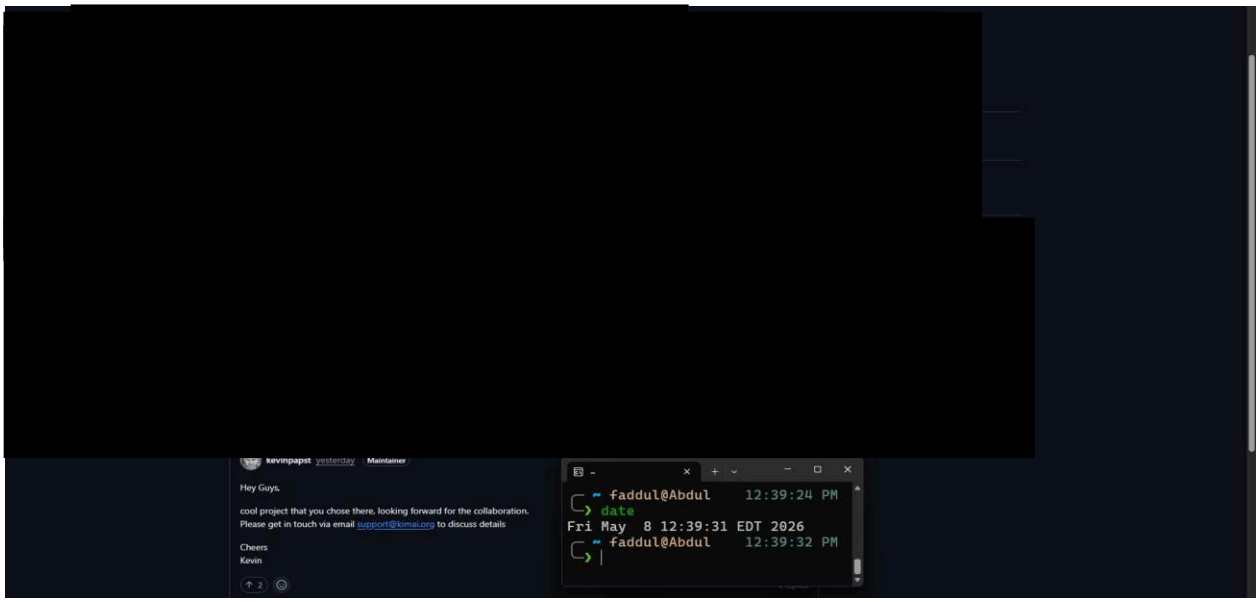


Figure 8 Kimai maintainer approval for student security research collaboration

Figure 8 shows the GitHub Discussions thread where our group requested permission to collaborate with the Kimai project for security testing. The Kimai maintainer, Kevin Papst, replied positively and directed the group to contact support@kimai.org to discuss details.

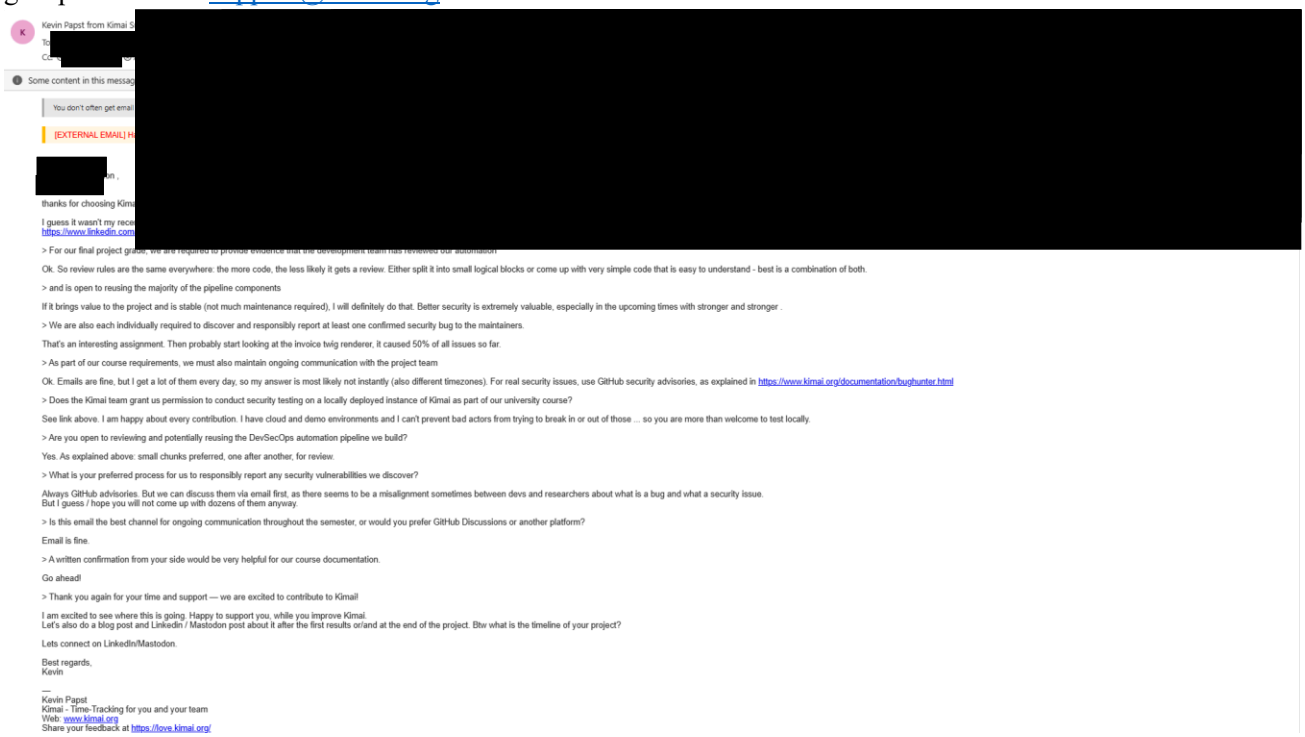


Figure 9 Email response from Kimai maintainer confirming collaboration and security testing scope

Figure 9 shows Kevin Papst from the Kimai project responding to our follow-up email after the GitHub discussion. The maintainer confirms that local testing, DevSecOps pipeline work, and responsible vulnerability reporting are acceptable, while also pointing the group to Kimai's security advisory and bug bounty documentation. This provides stronger evidence of maintainer engagement and confirms that our testing should remain local and responsibly reported.

Conclusion

This lab successfully explored Kimai as a real-world open-source web application suitable for application security analysis. The project was selected because it includes realistic business functionality such as time tracking, invoicing, reporting, authentication, authorization, API access, and database persistence. These features create a meaningful attack surface for future testing.

The local Docker deployment confirmed that Kimai 2.56.0 could run successfully in the lab environment with MySQL 8.3. During deployment, the group encountered and resolved configuration issues, including the TRUSTED_HOSTS error. This issue also demonstrated that Kimai includes Host header validation as part of its security posture.

The architecture review, SBOM generation, vulnerability scan, and advisory review showed that the highest-priority areas for future testing are API authorization, template rendering, output encoding, and third-party dependency management. Maintainer engagement was also completed through GitHub Discussions and email, establishing a responsible communication path for any findings discovered later in the semester.

Task Checklist

Table 5 Lab 1 task completion checklist.

Task	Status
Web application selection (Kimai)	Done
Maintainer engagement (GitHub Discussion #5941 + email)	Done
Project overview and features	Done
Competitive analysis (Kimai vs Solidtime vs Traggo)	Done
Architecture diagrams (DFD, Sequence, Decomposition)	Done
Integration and deployment context	Done
Local deployment (Docker Compose on Kali Linux)	Done
SBOM generation and vulnerability analysis	Done
AODA-compliant PDF report	Done
Professional video with live demo (5-10 min)	Done
In-class presentation preparation	Done

References

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

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

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

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

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

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

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

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

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