Kimai <= 2.57.0 Default APP_SECRET Authentication Bypass Vulnerability (CVE-2026-52824)

Author: Synthex **Site:** denizhalil.com CVSS 9.8 HIGH **Date:** July 2026

INTRODUCTION

Kimai, an open-source time tracking and project management platform, is widely used by organizations and freelancers worldwide to manage sensitive operational, financial, and client data. However, a critical security flaw identified as **CVE-2026-52824**, affecting version 2.57.0 and earlier, highlights the severe consequences of insecure default configurations in production environments. When cryptographic secrets remain unchanged from vendor defaults, the fundamental trust model of the application collapses, exposing administrative accounts to unauthorized access. In this article, we will examine the architectural root causes behind this authentication bypass, analyze the technical mechanics of how cryptographic tokens are forged, and detail the necessary mitigation steps required to secure affected systems.

LEARNING OBJECTIVES

After completing this article, you will understand:

- **Risks of Static Secret Keys:** The security hazards posed by hardcoded or predictable APP_SECRET values in web application frameworks.
- **Cookie Signing and HMAC Mechanics:** How Symfony-based applications handle cryptographic signature verification for "Remember Me" cookies and authentication tokens.
- **Secure Container Configuration:** Best practices for managing environment variables and hardening Docker deployments.

WHAT IS KIMAI <= 2.57.0 - DEFAULT APP_SECRET AUTHENTICATION BYPASS CVE-2026-52824

CVE-2026-52824 is a critical authentication bypass vulnerability impacting the Kimai open-source time-tracking system up to version 2.57.0, carrying a maximum severity CVSS v3 score of **9.8 (Critical)**. The flaw stems from a fundamental failure in default configuration management, where the application relies on a hardcoded, publicly known cryptographic secret key. When instance administrators deploy Kimai without explicitly overriding this default value during installation, the entire security boundary enforced by the underlying framework is rendered ineffective. At the heart of the issue is the APP_SECRET environment variable, which serves as the foundational key for generating cryptographic signatures and HMAC hashes across the platform. In standard Kimai architecture, this key is

responsible for securing user authentication state, generating valid "Remember Me" session cookies, signing password reset links, and validating anti-CSRF tokens. Because default installation templates and Docker entrypoint configurations shipped with a static fallback value, every uncustomized deployment effectively shares the exact same cryptographic key with potential attackers.

As a direct consequence, unauthenticated remote attackers can abuse this shared secret to forge valid authentication tokens locally without ever interacting with the database or providing legitimate credentials. By targeting predictable sequential user identifiers—such as the default administrator account ID (1)—an attacker can craft a signed session cookie and present it to the server. The application verifies the signature against its default local key, accepts the request as authentic, and grants full administrative access to the system.

- **Severity & Rating:** Rated as CVSS 9.8 (Critical) due to unauthenticated remote account takeover and session hijacking capabilities without user interaction.
- **Vulnerable Component:** Standard `.env` templates, `.env.dist` configuration files, and official Docker container startup scripts in Kimai versions ≤ 2.57.0.
- **Root Cause:** Insecure Default Initialization (**CWE-1188**) resulting from hardcoded fallback values assigned to the `APP_SECRET` cryptographic key.
- **Exploitability:** Highly accessible because sequential user IDs allow deterministic session cookie generation for targeted administrative accounts.

```
# Vulnerable Default Configuration (.env / docker-compose.yml)
# Kimai <= 2.57.0

# VULNERABLE: Hardcoded static default APP_SECRET in deployment templates
APP_SECRET=change_this_to_something_unique
APP_ENV=prod
TRUSTED_HOSTS=localhost,127.0.0.1
```

TECHNICAL DETAIL: HOW THE CVE-2026-52824 VULNERABILITY WORKS

The technical vulnerability lies within Kimai's integration with the underlying **Symfony framework** and its cryptographic authentication services. In a standard Symfony ecosystem, the APP_SECRET environment variable acts as the root symmetric key for all Hash-based Message Authentication Code (HMAC) calculations. When a user authenticates with the "Remember Me" option enabled, Kimai serializes the user's account details and generates a cryptographically signed cookie, storing it in the user's browser for long-term session persistence. The core breakdown occurs because the HMAC algorithm—typically HMAC-SHA256 in modern Symfony applications—depends entirely on the secrecy of the key to guarantee message integrity and authenticity. Because vulnerable Kimai deployments operate with the publicly documented default key (change_this_to_something_unique), the cryptographic protection is completely neutralized. An external actor can inspect the expected

structure of the KIMAI_REMEMBER cookie and generate a perfectly valid HMAC signature locally without needing access to the server or database.

During request processing, the server receives the forged cookie, extracts the user ID payload, and re-computes the HMAC signature using its own local APP_SECRET. Because both the attacker and the server share the exact same default key, the calculated hash matches the submitted hash perfectly. The framework accepts the forged cookie as genuine, bypasses all password verification checks, and instantiates an authenticated session for the targeted account—most notably the default administrator account (ID = 1).

- **HMAC Cryptographic Binding:** Symfony binds user account properties and expiration timestamps into a single payload, which is signed using `hash_hmac()` with `APP_SECRET` as the key.
- **Predictable Token Construction:** Because Kimai assigns sequential integer IDs to users, attackers can deterministically target specific accounts by incrementing the identifier in the cookie payload.
- **Lack of Secret Entropy Validation:** The application startup routines fail to evaluate whether `APP_SECRET` meets minimum entropy requirements or matches known vendor default strings.
- **Authentication Bypass Execution:** Successful signature verification causes the security provider to automatically retrieve the target user object from the database and authenticate the HTTP context.

```
// Vulnerable Conceptual Logic: Symfony Remember-Me Cookie Verification
// Kimai <= 2.57.0

class LegacyTokenVerifier
{
    private string $appSecret = "change_this_to_something_unique"; // VULNERABLE:
Hardcoded default secret

    public function verifyRememberMeCookie(string $cookieData, string
$providedSignature): bool
    {
        // Server calculates the expected HMAC signature using the local secret
        $expectedSignature = hash_hmac('sha256', $cookieData, $this->appSecret);

        // If $appSecret is known, an attacker can generate a matching
$providedSignature locally
        return hash_equals($expectedSignature, $providedSignature);
    }
}
```

CONCLUSION

CVE-2026-52824 serves as a textbook example of **CWE-1188 (Initialization of a Resource with an Insecure Default)** and highlights the catastrophic impact of hardcoded cryptographic defaults in

modern web applications. When an application's root secret key is static or publicly known, every high-level security mechanism built on top of it—including session integrity, token generation, and authentication boundaries—fails simultaneously. This vulnerability underscores that application security is only as strong as its underlying configuration management.

To fully mitigate this risk, organization administrators must immediately upgrade affected Kimai deployments to version **2.58.0** or higher, which enforces strict startup checks against known default secrets. Furthermore, existing instances must explicitly generate a strong, unique 32-byte secret key using cryptographic utilities like `openssl rand -hex 32` and update their `APP_SECRET` environment variables. Updating this secret automatically invalidates all previously forged or active session tokens, forcing all users to re-authenticate securely.

Ultimately, preventing default authentication bypasses requires adopting a "secure by default" philosophy across the entire software development lifecycle. Development teams and DevOps engineers must implement automated pipeline checks, container health verifications, and environment audits to detect uninitialized or hardcoded secrets before containers reach production. By combining timely software updates with robust configuration management practices, organizations can effectively insulate their environments against default secret exploitation.