Kimai : Time recording could not be started [undefined]

Contexto. Compatibilidad entre Kimai y ModSecurity (OWASP CRS): Diagnóstico y solución de falsos positivos

En una instalación de Kimai (versión 2.36.1) empece a sufrir un problema que no me permitía, continuar, parar ningún proceso de mi tablero de control de tiempos.

Sin embargo, si podía editarlos, o crearlos desde cero.

Me pillo, en mal momento y no se me ocurrio, mirar las Webmaster Tools del navegador para ver algo, y me limite a revisar los logs de la app.

Tras un desafortunado encuentro con el desarollador en su github, en el que de poco, me acuso de mentiroso, me puse las pilas y dedicí dedicarle un tiempo.

Análisis profundo

Kimai - Issue undifined

Los síntomas observados fueron:

- El frontend devolvía el mensaje genérico "Unknown error".
- Los logs de Kimai (`var/log/prod.log`) no contenían ningún rastro del fallo.
- Las acciones fallaban sin generar excepciones visibles.

Dada la ausencia de logs, y por las características de la app, pense en un problema de javascript, ya fuera pro acción o por implicación.

Abrí las web master tools, y allí estaba.

Webmaster tools - error 404

Mi servidor, era el que interceptaba y anulaba las peticiones antes de que la aplicación pudiera gestionarlas.

¿Cómo podía haber sido tan descuidado?

Mi entorno de sistemas esta altamente protegido pro ModSecurity + OSWAP Rules. Y estaba teniendo un hermoso 404, que la app era incapaz de apoyar en el mensaje de error.

Al analizar los logs de mod_security, se identificaron errores silenciosos al interactuar con determinadas funciones de la aplicación, especialmente en rutas protegidas como:

- `/en/timesheet/{id}/edit`
- `/api/timesheets/{id}/restart`
- `/api/timesheets/{id}/stop`

Diagnóstico técnico

Cómo identificar los falsos positivos

Directadmin (nginx)

En mi caso es usar el gestor de Mod Security de Directadmin

Apache

```
# Buscar bloqueos relacionados con Kimai
tail -f /var/log/apache2/modsec_audit.log | grep -E "(timesheet|kimai)"

# Extraer IDs de reglas que están bloqueando
grep -o "id:[0-9]*" /var/log/apache2/modsec_audit.log | sort | uniq -c
```

Las reglas que generaron el bloqueo fueron:

Regla `911100` – HTTP Method not allowed

- ModSecurity detectó el uso del método `PATCH` o `PUT` en las rutas afectadas.
- Esta regla está diseñada para permitir únicamente métodos `GET`, `POST`, `HEAD`, y `OPTIONS`.
- Resultado: la solicitud se bloquea antes de llegar a la aplicación.

Ejemplo de traza:

```
[2025-01-30 10:15:23.123456] [client 192.168.1.100] ModSecurity: Access denied with code 403
(phase 2).
Matched "Operator `Within' with parameter `GET HEAD POST OPTIONS'"
against variable `REQUEST_METHOD' (Value: `PATCH')
[file "/etc/modsecurity/crs/rules/REQUEST-911-METHOD-ENFORCEMENT.conf"]
[line "27"] [id "911100"]
```

Regla `930120` – LFI attempt (false positive)

- En campos como `description`, que contienen cadenas largas o técnicas, se produjo un falso positivo de "Local File Inclusion".
- Esta regla analiza parámetros `ARGS` y puede detectar patrones engañosos si el contenido es complejo.

Ejemplo:

```
Matched "Operator `PmFromFile' with parameter `lfi-os-files.data'"
against variable `ARGS:timesheet_edit_form[description]'
```

Solución propuesta

Dado que las rutas afectadas están protegidas por autenticación y son de uso legítimo dentro del flujo de Kimai, se optó por desactivar selectivamente las reglas conflictivas **únicamente para esas rutas**, utilizando exclusiones personalizadas de ModSecurity.

Puedes ver mas información en Mod Security. Desactivación global de reglas por path

Reglas de exclusión recomendadas (ModSecurity v3 / OWASP)

Archivo: `REQUEST-900-EXCLUSION-RULES-BEFORE-CRS.conf`

```
SecRule REQUEST_URI "@rx ^/(en/timesheet/\d+/edit|api/timesheets/\d+/(restart|stop))$" \
    "id:11010,\
    phase:1,\
    pass,\
    nolog,\
```

```
    ctl:ruleRemoveById=911100,\
    ctl:ruleRemoveById=930120"
```

Opcional: si se desea una exclusión más general para toda la API de timesheets:

```
SecRule REQUEST_URI "@beginsWith /api/timesheets/" \
    "id:11011,\
    phase:1,\
    pass,\
    nolog,\
    ctl:ruleRemoveById=911100"
```

Verificación de la solución

Comprobar que las reglas se han cargado correctamente:

```
# Verificar la sintaxis de ModSecurity
nginx -t

# Recargar la configuración
systemctl reload nginx
```

Testear las funcionalidades afectadas:

1. Editar un timesheet existente
2. Reiniciar/detener un timer via API
3. Verificar que no aparecen más bloqueos en `modsec_audit.log`

Consideraciones de seguridad

- Estas exclusiones deben limitarse únicamente a rutas y métodos necesarios.
- En entornos críticos se puede condicionar la exclusión a cabeceras, IPs de confianza o sesiones autenticadas.
- No se recomienda desactivar reglas globalmente sin una evaluación de riesgo previa.

> ⚠ **NOTA IMPORTANTE: Sobre la desactivación global de reglas**
>
> Una práctica lamentablemente extendida en entornos de hosting profesional es la desactivación global de reglas ModSecurity ante el primer falso positivo. Esto representa un **grave error de seguridad** que debe evitarse a toda costa.
>
> **Por qué es una mala práctica:**
>
> - **Exposición innecesaria**: Desactivar una regla globalmente expone TODAS las aplicaciones del servidor, no solo la que presenta el problema
> - **Pérdida de protección**: Reglas como la 911100 (métodos HTTP) o 930120 (LFI) protegen contra ataques reales y documentados
> - **Efecto cascada**: Cada regla desactivada globalmente reduce la efectividad del WAF hasta convertirlo en decorativo
> - **Falsa sensación de seguridad**: Tener ModSecurity "activo" pero con reglas críticas deshabilitadas es peor que no tenerlo
>
> **La forma correcta:**
>
> - Identificar la ruta/aplicación específica afectada
> - Documentar el falso positivo y su justificación técnica
> - Aplicar exclusiones quirúrgicas solo donde sea necesario
> - Revisar periódicamente las exclusiones para validar su vigencia
>
> En entornos de hosting compartido, donde conviven múltiples aplicaciones y clientes, esta disciplina no es opcional: es una responsabilidad profesional fundamental.

Alternativas más granulares

Si se requiere mayor seguridad, se pueden implementar exclusiones condicionales:

```
# Solo para usuarios autenticados (verificando cookie de sesión)
SecRule REQUEST_URI "@rx ^/api/timesheets/\d+/(restart|stop)$" \
    "id:11012,\
    phase:1,\
    pass,\
    nolog,\
    chain"
    SecRule REQUEST_COOKIES:PHPSESSID "@rx ^[a-zA-Z0-9]{26,32}$" \
        "ctl:ruleRemoveById=911100"
```

```
# Solo desde IPs internas
SecRule REQUEST_URI "@rx ^/api/timesheets/" \
    "id:11013,\
    phase:1,\
    pass,\
    nolog,\
    chain"
    SecRule REMOTE_ADDR "@ipMatch 192.168.0.0/16,10.0.0.0/8" \
        "ctl:ruleRemoveById=911100"
```

Conclusión

Kimai, como muchas aplicaciones modernas que emplean métodos HTTP extendidos (PUT, PATCH, DELETE), puede encontrarse con bloqueos inesperados si se implementa un WAF sin ajustar sus reglas a los flujos funcionales reales.

Este caso demuestra la importancia de realizar una auditoría cruzada entre el comportamiento esperado de la aplicación y las políticas de seguridad impuestas por el entorno.

Documentar y aislar los falsos positivos ayuda no solo a mantener el entorno seguro, sino también a garantizar la funcionalidad legítima para el usuario final.

Referencias

- Kimai Documentation - API
- OWASP ModSecurity CRS
- ModSecurity Reference Manual
- CRS False Positives Tuning

Aviso

Revision #1
Created 2025-07-03 15:59:48 UTC by Abkrim

Updated 2025-07-03 16:40:17 UTC by Abkrim