ClawSuite

Informe de Auditoría de Seguridad Completo

Fecha: 24 de febrero de 2026
Aplicación: ClawSuite (interfaz web para agente IA)
Stack: TypeScript · TanStack Start · Node.js
Fases realizadas: 3

Índice

Resumen Ejecutivo

Se auditó la totalidad del código fuente del servidor (`src/server/`, `src/routes/api/`). Se identificaron **6 vulnerabilidades** distribuidas en severidad crítica, alta y media. **Todas han sido corregidas.**

Severidad		Encontradas	Corregidas
•	**Crítica**	1	1 ✓
•	**Alta**	2	2 ✓
•	**Media**	3	3 ✓
•	**Bug**	1	1 ✓
•	**Baja**	1	N/A (documentada)

Fase 1 — Exposición de Endpoints sin Autenticación

[CRÍTICO] ~50 endpoints sin protección de autenticación

Descripción:
Prácticamente todos los endpoints de la API (`/api/files`, `/api/browser`, `/api/skills`, `/api/sessions`, `/api/send`, etc.) carecían del guard de autenticación. Cualquier cliente en la red podía acceder a ellos directamente sin credenciales.

Impacto: Acceso completo al agente IA, archivos del workspace, terminal, historial de sesiones y configuración.

Fix aplicado: Se implementó `requireAuth(request)` como primera guardia en todos los handlers. La función centralizada en `auth-middleware.ts` devuelve HTTP 401 si la sesión no está autenticada.

Archivos modificados: ~50 archivos en `src/routes/api/`.

Fase 2 — Vulnerabilidades Profundas

[ALTO] CSRF no aplicado a pesar de estar implementado

Descripción:
La función `validateCsrf()` implementaba correctamente el patrón de doble cookie para proteger contra CSRF, pero **nunca se llamaba**. Cualquier sitio web malicioso podía enviar solicitudes POST en nombre del usuario autenticado.

Impacto: Un sitio de terceros podía controlar el agente, enviar mensajes, borrar sesiones o modificar la configuración si el usuario visitaba el sitio malicioso mientras estaba logueado.

Fix aplicado: `validateCsrf()` fue integrado directamente dentro de `requireAuth()` en `auth-middleware.ts`. Al estar centralizado, protege automáticamente todos los endpoints que usan `requireAuth`.

```
+ // Double-submit CSRF validation for mutating methods
+ if (!validateCsrf(request)) {
+   return new Response(
+     JSON.stringify({ ok: false, error: 'CSRF validation failed' }),
+     { status: 403, headers: { 'Content-Type': 'application/json' } },
+   )
+ }
```

[ALTO] Path Traversal en subida de archivos

Descripción:
El endpoint `/api/files` (POST, acción `upload`) construía la ruta destino usando directamente `file.name` sin sanearlo:

```
// VULNERABLE
const destination = path.join(resolvedTarget, file.name)
```

Un atacante podía enviar un archivo con el nombre `../../server/auth-middleware.ts` y sobrescribir archivos críticos del servidor fuera del workspace.

Impacto: Escritura arbitraria de archivos en el sistema de archivos del servidor; posible RCE si se sobreescribe código que se ejecuta.

Fix aplicado: En `files.ts` se sanitiza el nombre antes de construir la ruta, y se añade una validación final:

```
- const destination = path.join(resolvedTarget, file.name)
+ const fileName = path.basename(file.name)  // elimina directorios del nombre
+ const destination = path.join(resolvedTarget, fileName)
+ ensureWorkspacePath(destination)            // verifica que sigue dentro del
    workspace
```

[MEDIO] SSRF y acceso a archivos locales via automatización de browser

Descripción:
El endpoint `/api/browser` (acción `navigate` y `proxy-navigate`) no validaba el protocolo de la URL antes de navegar:

```
// VULNERABLE - podia recibir file:///etc/passwd o http://192.168.1.1
const state = await navigate(url)
```

Impacto: Un usuario autenticado (o con sesión comprometida) podía leer archivos locales del servidor (`file://`) o acceder a servicios internos de red (`http://192.168.x.x`, metadatos de cloud como `http://169.254.169.254`).

Fix aplicado: En `browser.ts` se valida el protocolo antes de navegar. Solo `http:` y `https:` son permitidos:

```
const parsed = new URL(url)
if (parsed.protocol !== 'http:' && parsed.protocol !== 'https:') {
  return json({ error: 'Only http and https protocols are allowed' }, { status: 400
      })
}
```

Adicionalmente, el handler POST completo fue migrado a validación con `Zod` (`discriminatedUnion` por acción) para rechazar cualquier body mal formado.

[MEDIO] Inyección en archivo .env via gateway config

Descripción:
El endpoint `/api/gateway-config` (POST) escribía el valor `url` y `token` directamente al archivo `.env` sin validación:

```
// VULNERABLE - body.url podia contener \n para inyectar variables
envContent += `\nCLAWDBOT_GATEWAY_URL=${body.url}`
```

Un atacante autenticado podía inyectar líneas adicionales en el `.env` enviando, por ejemplo: `url = "ws://legit.com\nOPENAI_API_KEY=atacante"`

Impacto: Modificación arbitraria de variables de entorno si se controla la sesión; potencial escalada si las nuevas variables afectan comportamiento crítico.

Fix aplicado: En `gateway-config.ts`:

- La URL debe ser una URL WebSocket válida (`ws://` o `wss://`). Cualquier otro protocolo devuelve HTTP 400.
- Se eliminan caracteres de control `\r\n\t` antes de escribir.
- El token se valida que no contenga saltos de línea y tiene un límite de 512 caracteres.

[MEDIO] Suplantación de IP en rate limiter

Descripción:
La función `getClientIp()` confiaba ciegamente en el header `X-Forwarded-For`, que cualquier cliente puede falsificar directamente:

```
// VULNERABLE
return forwarded.split(',')[0].trim()
```

Un atacante podía evadir el rate limit enviando `X-Forwarded-For: 1.2.3.4` con cada request para rotar su IP percibida.

Impacto: Bypass del rate limit de autenticación (5 intentos/minuto) → fuerza bruta de contraseña.

Fix aplicado: En `rate-limit.ts`, ahora se prioriza `X-Real-IP` (header configurado por el proxy, no modificable por el cliente cuando se usa Nginx correctamente) antes de `X-Forwarded-For`.
Nota de producción: La protección completa requiere que Nginx/proxy esté configurado para sobrescribir `X-Forwarded-For`. Sin proxy, el rate limit puede seguir siendo bypasseable.

Análisis de lo que estaba bien

Área	Estado previo
Contraseña con `timingSafeEqual`	✓ Correcto desde el inicio
Rate limit en endpoint de login (5/min)	✓ Correcto desde el inicio
Delay artificial en password incorrecta (1s)	✓ Correcto desde el inicio
`ensureWorkspacePath()` en skills	✓ Correcto desde el inicio
Cookies `HttpOnly` + `SameSite=Strict`	✓ Correcto desde el inicio
`execSync` reemplazado por `execFile`	✓ Corregido en fase anterior
Error messages no exponen stack en producción	✓ Correcto desde el inicio

Fase 3 — Errores de Sintaxis por Inserción Incorrecta de Imports

[BUG] Imports de requireAuth insertados dentro de bloques import {} existentes

Descripción:
Durante la Fase 1 (autenticación masiva de endpoints), el import de `requireAuth` fue insertado incorrectamente dentro de bloques `import {` multi-línea ya existentes en 7 archivos, rompiendo su sintaxis:

```
// ROTO - requireAuth dentro del import block de otro modulo
import {
import { requireAuth } from '../../server/auth-middleware'
  buildCostSummary,
} from '../../server/usage-cost'
```

Impacto: Los archivos no compilaban (TS1434: Unexpected keyword). La aplicación no podía iniciar.

Fix aplicado: Se movió `import { requireAuth }` a su propia línea antes del bloque afectado en todos los archivos.
Archivos corregidos: `cost.ts`, `usage.ts`, `models.ts`, `debug-analyze.ts`, `events/recent.ts`, `tasks/index.ts`, `tasks/$taskId.ts`

Archivos Modificados (Total)

Archivo
auth-middleware.ts
files.ts
browser.ts
gateway-config.ts
rate-limit.ts

Recomendaciones Adicionales (No de Código)

> **NOTE**
>
> Estas mejoras son de **infraestructura** y no requieren cambios en el código.

- **Nginx como proxy inverso:** Configurar para que sobrescriba `X-Forwarded-For` con la IP real del cliente. Esto hace el rate limit a prueba de spoofing.
- **HTTPS obligatorio:** Asegurarse de que la aplicación no se sirve por HTTP en producción. Sin HTTPS, las cookies de sesión y CSRF pueden ser capturadas en tránsito.
- **Variables de entorno en producción:** No usar `.env` file — usar variables de entorno del sistema o un secret manager (Vault, Railway secrets, etc.).