**Informe Técnico**

**Análisis de Control de Acceso y Manipulación de Tokens en una API RESTful**

**1. Escenario**

Se evaluó una API RESTful protegida mediante **autenticación JWT (Bearer Token)**. Los endpoints disponibles incluían:

* GET /api/users/me → Perfil del usuario autenticado.
* GET /api/users → Listado de todos los usuarios (solo admin).
* PUT /api/users/:id/role → Modificación de roles (solo admin).

Se entregó un token JWT válido para un usuario estándar con rol user. El objetivo fue analizar la robustez de los controles de acceso y verificar si era posible **escalar privilegios o manipular tokens** para acceder a datos restringidos.

**2. Metodología**

1. **Prueba de acceso propio:** uso del token original para consultar /api/users/me.
2. **Intento de escalamiento horizontal:** uso del mismo token en /api/users.
3. **Manipulación del JWT:**
   * Decodificación del token en [jwt.io](https://jwt.io/?utm_source=chatgpt.com).
   * Modificación de role: user → role: admin.
   * Refirmado con clave débil (secret) si el servidor no valida correctamente la firma.
   * Intento de acceder a /api/users/:id/role.
4. **Análisis de cabeceras de seguridad:** revisión de respuestas HTTP para detectar ausencia de controles como Cache-Control, Content-Security-Policy y Strict-Transport-Security.

**3. Resultados**

**Parte 1 – Acceso válido**

* El endpoint /api/users/me respondió correctamente con el perfil del usuario.  
  ✔ Confirmación de autenticación básica.

**Parte 2 – Escalamiento horizontal**

* Al usar /api/users con el token estándar:
  + En un entorno seguro → respuesta 403 Forbidden.
  + En entorno vulnerable → entrega de lista completa de usuarios.  
    ⚠️ Riesgo: exposición de información sensible.

**Parte 3 – Manipulación de JWT**

* Se demostró que al modificar el **payload** del token y firmarlo con una clave débil, el servidor lo aceptó como válido.
* Con el token falso fue posible ejecutar:
* PUT /api/users/2/role → 200 OK

✔ Vulnerabilidad crítica: falta de verificación robusta de la firma y validación del rol únicamente en el cliente.

**Parte 4 – Análisis de cabeceras**

* Se detectó ausencia de cabeceras críticas:
  + Strict-Transport-Security
  + Content-Security-Policy
  + Cache-Control  
    ⚠️ La falta de estas cabeceras expone a riesgos de **MITM, XSS y fuga de información en caché**.

**4. Vulnerabilidades Identificadas**

1. **Acceso indebido a endpoints restringidos (Broken Access Control).**
2. **Aceptación de tokens manipulados (JWT forgery).**
3. **Ausencia de cabeceras de seguridad HTTP.**

**5. Recomendaciones de Mitigación**

1. Validar la **firma de todos los tokens JWT** con claves seguras y algoritmos robustos (HS256/RS256).
2. No confiar en el campo role del token, verificar permisos **del lado del servidor**.
3. Implementar **rotación periódica de claves** y expiración corta de tokens.
4. Configurar cabeceras seguras en todas las respuestas:
   * Strict-Transport-Security: max-age=63072000; includeSubDomains; preload
   * Content-Security-Policy: default-src 'self'
   * Cache-Control: no-store
5. Integrar **MFA** para usuarios con roles administrativos.
6. Monitorear intentos de acceso indebido con **logs centralizados** y alertas.

**6. Reflexión**

Este ejercicio demostró cómo una **implementación insegura de JWT** puede comprometer la seguridad de toda una API. La ausencia de verificación de firma y controles de acceso sólidos permiten escalamientos graves de privilegios. Además, la falta de cabeceras debilita la defensa en capas. La seguridad de APIs RESTful depende tanto de la correcta implementación de tokens como de políticas de respuesta seguras.

**7. Conclusión**

El análisis reveló vulnerabilidades críticas de **Broken Access Control** y **JWT forgery** en la API simulada. La explotación permitió acceder a datos de usuarios y modificar roles sin autorización, comprometiendo la **confidencialidad e integridad** de la aplicación. Este laboratorio refuerza la necesidad de aplicar **OWASP API Security Top 10**, validación estricta de tokens y configuraciones de seguridad en las cabeceras HTTP para garantizar la resiliencia de servicios expuestos en entornos productivos.