OFICINA DE POSGRADOS

Tema:

PROPUESTA DE BUENAS PRÁCTICAS DE SEGURIDAD PARA CREACIÓN, TRANSPORTE Y ALMACENAMIENTO DE JSON WEB TOKENS

Línea de Investigación:

Seguridad de la Información

Autor: Fausto Danilo Cevallos Muñoz

Director: Paul Fernando Bernal Barzallo, Mg.

Ambato – Ecuador

Marzo 2022

PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

SEDE AMBATO

HOJA DE APROBACIÓN

Tema:

PROPUESTA DE BUENAS PRÁCTICAS DE SEGURIDAD PARA CREACIÓN, TRANSPORTE Y ALMACENAMIENTO DE JSON WEB TOKENS

Línea de Investigación:

SEGURIDAD DE LA INFORMACIÓN

Autor:

Fausto Danilo Cevallos Muñoz

Paul Fernando Bernal Barzallo, Mg. f._____
CALIFICADOR

Galo Mauricio López Sevilla, Mg. f._____
CALIFICADOR

Darío Javier Robayo Jácome, Mg. f._____
CALIFICADOR

Juan Carlos Acosta Teneda, PhD. f._____
COORDINADOR DE LA OFICINA DE POSGRADOS

Pontificia Universidad Católica del Ecuador
OFICINA DE POSGRADOS

Hugo Rogelio Altamirano Villaroel, Dr. f._____
SECRETARIO GENERAL PUCESA

Pontificia Universidad Católica del Ecuador
SECRETARÍA GENERAL
PROCURADURÍA

Ambato – Ecuador

Marzo 2022

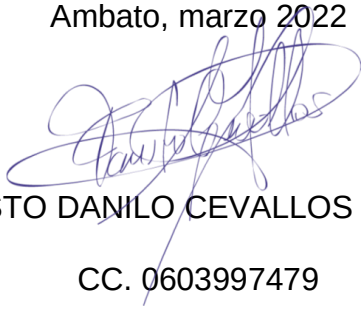
DECLARACIÓN DE AUTENCIDAD Y RESPONSABILIDAD

Yo: **FAUSTO DANILO CEVALLOS MUÑOZ**, con CC. **060399747-9** autor del trabajo de grado titulado: **"PROPUESTA DE BUENAS PRÁCTICAS DE SEGURIDAD PARA CREACIÓN, TRANSPORTE Y ALMACENAMIENTO DE JSON WEB TOKENS"**, previo a la obtención del título profesional de MAGÍSTER EN CIBERSEGURIDAD, en la OFICINA DE POSGRADOS.

1.- Declaro tener pleno conocimiento de la obligación que tiene la Pontificia Universidad Católica del Ecuador, de conformidad con el artículo 144 de la Ley Orgánica de Educación Superior, de entregar a la SENESCYT en formato digital una copia del referido trabajo de graduación para que sea integrado al Sistema Nacional de Información de la Educación Superior del Ecuador para su difusión pública respetando los derechos de autor.

2.- Autorizo a la Pontificia Universidad Católica del Ecuador a difundir a través del sitio web de la Biblioteca de la PUCE Ambato, el referido trabajo de graduación, respetando las políticas de propiedad intelectual de Universidad.

Ambato, marzo 2022



FAUSTO DANILO CEVALLOS MUÑOZ

CC. 0603997479

RESUMEN

El desarrollo de software es uno de los pilares fundamentales en el manejo de información en la actualidad, es por ello que constantemente, se busca incrementar su seguridad, uno de los mayores inconvenientes es manejar procesos de autenticación y autorización, en la implementación de interfaces API es común el uso de JSON Web Tokens, los cuales llegarían a ser considerados como datos "seguros", en el presente estudio, se busca identificar las mejores prácticas de seguridad en su creación, transporte y almacenamiento, así como; determinar posibles vulnerabilidades. Para la investigación, se empleó como metodología de desarrollo la revisión sistemática de literatura en fuentes como SCOPUS, SCIENCE DIRECT, DIALNET y La Biblioteca Digital de la Pontificia Universidad Católica del Ecuador - Sede Ambato, se estableció 3 criterios de inclusión, 2 criterios de exclusión y 2 criterios de calidad, de 1138 documentos analizados, se obtuvieron 11 estudios primarios, como resultado de la investigación, se logró obtener una propuesta de buenas prácticas, se realizaron pruebas de concepto para verificar su aplicabilidad, las mismas que consistieron en validar la fortaleza de los tokens ante ataques de fuerza bruta y el impacto que tiene una validación débil de permisos y privilegios.

PALABRAS CLAVES: Tokens, JWT, JWS, JWE, Buenas Prácticas.

ABSTRACT

Software development is one of the fundamental pillars in the management of information today, which is why it constantly seeksto increase its security, one of the biggest problems is to manage authentication and authorization processes, in the implementation of API interfaces is common the use of JSON Web Tokens, which can be considered as "secure" data, this study seeks to identify the best security practices in its creation, transport and storage, as well as; determine potential vulnerabilities. The methodology used for the research was the systematic review of literature in sources such as SCOPUS, SCIENCE DIRECT, DIALNET and the Digital Library of the Pontificia Universidad Católica del Ecuador -Sede Ambato, 3 inclusion criteria, 2 exclusion criteria and 2 quality criteria were established, from 1138 documents analyzed, 11 primary studies were obtained, as a result of the research a proposal of good practices was obtained, proofs of concept were carried out to verify its applicability, which consisted in validating the strength of the tokens against brute force attacks and the impact of a weak validation of permissions and privileges.

Keywords: Tokens, JWT, JWS, JWE, GoodPractices

ÍNDICE GENERAL

ÍNDICE DE IMÁGENES

INTRODUCCIÓN

El internet ha marcado la evolución de la comunicación en el mundo moderno, las aplicaciones y sistemas informáticos integran la mayoría de los canales de intercambio de información, los medios tradicionales como el teléfono convencional y los diarios impresos tienen menor uso en comparación con aplicaciones como Twitter, Telegram o las redes sociales.

La mayoría de las aplicaciones emplean fuertemente el protocolo *Hypertext Transfer Protocol* (HTTP) o su versión segura (HTTPS), se aprecia en la barra de direcciones en los navegadores al ingresar a un sitio web, según Clay (2021) en el Ecuador el 52.6% del tráfico web generado entre diciembre de 2019 y diciembre de 2020, se dió por el uso de laptops y computadoras.

Por su uso masivo, las aplicaciones web tienen la necesidad constante de evolucionar, ya sea para optimizar el consumo de recursos en los servidores o tener menor tiempo de respuesta en cada petición del usuario, por ello, se diseñan con base en el consumo de servicios web, para RedHat (2021) existen dos enfoques de intercambio de datos en línea que son: *Simple Object Access Protocol* (SOAP) y *Representational State Transfer* (REST), REST es la tendencia actual en el desarrollo de aplicaciones web por su flexibilidad.

En este contexto los servicios compartidos no siempre son de acceso público, por la criticidad de la información que contienen, el mecanismo de protección habitual a nivel de usuario es el uso de credenciales; pero las aplicaciones, se comunican a través del envío de sesiones, *tokens* o cookies de autenticación y autorización, por razones de seguridad según Baars et al. (2020). Recomienda no almacenarán credenciales en texto plano en el código fuente de los aplicativos.

El presente estudio tomará en consideración al *JSON Web Token* (JWT), es un estándar abierto (RFC519) publicado en 2015, el *token* es una cadena de caracteres que permite proteger servicios web basados en la arquitectura REST, el servidor valida la petición del recurso mediante una cabecera de autorización en el protocolo HTTP, caso contrario la petición será denegada.

Pese a ser un estándar robusto y ampliamente utilizado en las aplicaciones modernas, según Mitre (2019). existen múltiples vulnerabilidades, se clasifican en distintos ámbitos de implementación, ya sean por parte de librerías externas al software desarrollado o por parte de un desarrollador, al ser vulnerabilidades en procesos de autenticación/autorización tienen un impacto crítico sobre la seguridad de la información.

OWASP (2021). en el Top 10 de riesgos de seguridad de 2021 identifica, en primer lugar, la vulnerabilidad "*Broken Access Control*", en segundo lugar, "*Cryptographic Failures*", en cuarto lugar "*Insecure Design*" y en quinto lugar "*Security Misconfiguration*", el estudio deja en evidencia que hoy por hoy, los fallos a nivel de autenticación y autorización son muy comunes.

Según Abril (2021). en su publicación digital del 29 de julio de 2021 enuncia: "Ecuador está entre los países con más ciberataques en América Latina", en uno de sus acápites menciona que existe una "Falta de personal en las empresas para cuidar la seguridad" y acusa el problema a que "…se ha producido una gran cantidad de programadores pero muy pocos expertos en seguridad...".

La idea acerca de la falta de profesionales en el área de seguridad de la información, no solo en ciberseguridad, es acertada; pero existe otro punto de vista que ayuda a disminuir estos problemas y es la capacitación continua del personal de tecnología, en especial a los desarrolladores de software, la tecnología avanza a un ritmo muy acelerado y no siempre permite a las empresas acoplarse a esos cambios.

La capacitación para asegurar un servicio implica un amplio conocimiento de conceptos como: administración de servidores, redes, bases de datos y desarrollo de software; el control de acceso es un factor clave en la seguridad de las aplicaciones basadas en el consumo de servicios web, por lo que surge la interrogante: ¿Cómo se lograría incrementar la seguridad en la creación, transporte y almacenamiento de JWT?

El presente trabajo investigación tiene como idea a defender que: El contar con buenas prácticas de seguridad en el uso de JWT permitirá incrementar el nivel de

seguridad de las aplicaciones que empleen este estándar, generalmente enfocado en aplicaciones web de tipo cliente servidor, microservicios, *Single Page Application* (SPA), *Multiple Page Application* (MPA) y aplicaciones móviles que hacen consumo de servicios externos.

Al desarrollar la idea a defender, se tiene como objetivo general proponer buenas prácticas de seguridad para la creación, transporte y almacenamiento de JSON Web Tokens para lo cual, se proponen los siguientes objetivos específicos:

1. Analizar el estado del arte del proceso de construcción de un JSON Web Token.
2. Diagnosticar las vulnerabilidades que afectan a cada una de las etapas del aprovisionamiento del *token*.
3. Recopilar los factores de seguridad que permiten mitigar las vulnerabilidades identificadas.

Al ser una guía de recomendaciones, se realiza un estudio preliminar de aquellas propuestas existentes para incluir, debatir o refutar con los resultados de la investigación y condensar en un solo documento que sea de fácil comprensión para las empresas o desarrolladores.

No se pretende simplemente realizar un compendio de información, se busca ampliar el concepto genérico de "*token* de seguridad" que usualmente, se ve como un dato seguro, no se toma en cuenta el proceso de transmisión y almacenamiento en el navegador.

La presente investigación permitirá que los desarrolladores de software y profesionales de la ciberseguridad, cuenten con un conjunto de lineamientos claros sobre autenticación y autorización con JWT, sus posibles vulnerabilidades, ataques y recomendaciones de seguridad, con enfoque independiente de la arquitectura de la aplicación y el lenguaje de programación; facilita su implementación, proceso de análisis de vulnerabilidades y logra así, incrementar la percepción de seguridad en los usuarios de las aplicaciones.

Por la naturaleza del proyecto, se considera una investigación de tipo bibliográfica, se realizará una revisión sistemática de literatura con un enfoque inductivo, para

identificar factores comunes en el uso de JWT, una vez finalizado el análisis, se realiza la propuesta de buenas prácticas y pruebas de concepto.

CAPÍTULO I. ESTADO DEL ARTE Y LA PRÁCTICA

En el contexto de validar la autenticación y autorización de servicios expuestos en la web, existen varias recomendaciones, metodologías y estándares a seguir, pese a ello, se ha notado que existen vulnerabilidades, por lo que es necesario comprender como funcionan estos procesos.

1.1. Trabajos Relacionados

En el estudio realizado por Villa (2019). existe una propuesta de buenas prácticas para el aseguramiento de los servicios web basados en API (*Application Programming Interface*), considera su desarrollo en base a las recomendaciones de seguridad REST que emite la OWASP y expone una “convergencia hacia el uso de JWT” con el uso de firmas criptográficas o un Código de Autenticación de Mensaje (MAC).

En el documento de investigación, se emiten ideas de interés y deja en claro que existen pruebas de seguridad, pero condiciona la conceptualización del uso de JWT como un dato seguro, no se hace mención a las pruebas realizadas en específico sobre el *token*, salvo omitir su uso en la cabecera de autorización de la petición para evidenciar la protección de acceso a las rutas en específico.

OWASP (n.d.). en su guía *REST Security* enuncia medidas para incrementar la seguridad del *token*, de forma inicial, se verifica la protección de integridad mediante una firma o MAC, no permitir el uso de *token* con una especificación del algoritmo “*none*” o ninguno, mantener fechas de caducidad y si llega a caducar será incluido en la aplicación en una lista de rechazados, para invalidar cualquier solicitud que contenga *tokens* obsoletos.

Auth0 es una plataforma que permite a los desarrolladores emplear procesos de control de acceso de forma rápida y fácil, posee una guía de uso de *tokens* Auth0 (n.d.-a). la que recomienda que la firma del *token* sea con una llave segura y secreta, mantener fechas de expiración, no se emplearán datos sensibles en el *payload* y serán enviados mediante uso de *HyperText Transfer Protocol Secure* (HTTPS).

Un aspecto importante que menciona la guía y será considerado de forma rigurosa, es evitar el intercambio de información innecesaria, es decir, excesivas solicitudes entre el cliente y el servidor, puesto que incrementaría las posibilidades de interceptar la comunicación y el atacante obtendría los datos de la cabecera de autenticación.

En cuanto al almacenamiento de JWT, la guía *Token Storage* Auth0 (n.d.-b). plantea varios puntos de vista, el primero es una aplicación web tradicional tipo monolito, en la cual, el *token* será almacenado en una cookie en el servidor, pero no tenedrá un tamaño mayor a 4 *Kilobytes* (KB), en una aplicación móvil o nativa, se recomienda el uso de espacios seguros en el sistema operativo, en Android KeyStore y en iOS KeyChains.

Con estas recomendaciones, se presume la existencia de aplicaciones vulnerables, generalmente las credenciales en las aplicaciones móviles son almacenadas en texto plano, bases de datos ligeras como SQLite o en espacios de memoria compartidos, por otro lado, las webs tradicionales hacen uso de cookies, pero estas estarán protegidas de acceso desde el cliente en los navegadores, para prevenir ataques con vector *Cross-Site Scripting* (XSS).

Debido al auge de las aplicaciones desarrolladas con el lenguaje de programación JavaScript, las *Single Page Application,* también, tienen consideraciones de almacenamiento de datos sensibles, Auth0 recomienda emplear su *Software Development Kit* (SDK), pero no siempre una empresa desea emplear software de terceros, eso marca una dependencia directa de procesos sensibles en el flujo de información.

Las aplicaciones web funcionan en navegadores, ya sean de teléfonos móviles o computadores, estos tienen varios espacios de almacenamiento, como cookies, en memoria en tiempo de ejecución, almacenamiento por sesión (pestaña) y almacenamiento local para permitir persistencia entre sesiones.

Para Auth0 el mejor medio de protección de los *tokens* en un navegador es el almacenamiento en memoria al emplear *Web Workers*, pero esto no permite la persistencia de la sesión del usuario entre pestañas o al refrescar el navegador y

mantiene la dependencia del SKD, se ejemplifica este procedimiento con Whatsapp Web, que a la fecha de la presente investigación no permite emplear dos pestañas simultáneas en el mismo navegador.

Finalmente, el almacenamiento local del navegador es una alternativa muy útil a los desarrolladores, persiste sesiones y mantiene el estado al refrescar la página, pero este medio necesita ser asegurado, puesto que si un atacante logra inyectar código malicioso sobre la SPA obtendría accesos al *token* almacenado.

En el trabajo de Salas (2020). expone una propuesta de aplicación de seguridad en API con JWT y un certificado auto firmado de clave pública, esta implementación utiliza un gestor de API “KONGA”, en este caso, se aplican las recomendaciones, sin embargo, Salas menciona, también, en sus conclusiones que existen detractores del uso de JWT por posibles “fallas de seguridad”, que se debe a la falta de aplicación del estándar y problemas de configuración inadecuada.

Los desarrolladores de software o programadores hoy en día están inmersos en una gran cantidad de cambios, que se dan a gran velocidad no solo por las tecnologías nacientes o los lanzamientos de nuevas versiones, sino que depende los atacantes que cada vez crecen en número y deja una discusión abierta, las vulnerabilidades son inherentes a la tecnología o por omisión del personal que las implementa.

1.2. Tipo de Aplicaciones que usan tokens

Existe una gran variedad de aplicaciones, arquitecturas y filosofías sobre la construcción de software, desde los monolitos hasta el consumo de servicios, en su mayoría cualquiera es capaz de implementar control de acceso basado en *token* de sesión, para Wike (2021). en la publicación “Tipos de aplicaciones para la Plataforma de identidad de Microsoft” hace referencia a los siguientes tipos de aplicación basados en OAuth 2.0:

1.2.1. Aplicaciones de una Página (SPA)

Este tipo de aplicaciones tienen enfoque en la división de responsabilidades en la arquitectura de software, hoy ya es tradicional hablar de *frontend* y *backend* como

conceptos predefinidos, una SPA está construida para funcionar desde el ordenador o dispositivo móvil del cliente desde un navegador, se caracteriza por el consumo de servicios desde un servidor externo mediante peticiones HTTP.

Las SPA son creadas con *frameworks* de desarrollo de software como Angular, React y Vue, basados en el lenguaje de programación JavaScript, para el consumo de servicios emplean peticiones (*request*) mediante *Asynchronous JavaScript* And XML (ajax), los *tokens* son enviados como parte del cuerpo de la petición o en la cabecera, lo más común es el uso de la cabecera *Authorization*.

Las peticiones hacen uso de los verbos del protocolo HTTP como son *GET, POST, PUT, DELETE y OPTIONS* a modo de convención, se emplean para consultar, agregar, editar y eliminar recursos desde el servicio o la API, no generan *token* pero si permiten el envío de credenciales para obtenerlo desde el servidor.

Su nivel de control de acceso funciona en base a mostrar y ocultar funcionalidad al usuario según los roles y permisos asignados al mismo, sin embargo, son distribuidas de manera parcial o completa mediante la compilación de sus fuentes en archivos con extensión ".js", un actor malicioso tiene la facilidad de analizar el código fuente desde el navegador y encontrar las rutas de las API existentes, si el *token* no valida esto de manera adecuada genera una brecha de seguridad.

1.2.2. Aplicaciones Web

Son aplicaciones tradicionales diseñadas para ejecutarse en el servidor, sus lenguajes de desarrollo más comunes son .NET, PHP, Java, Ruby, NodeJS; estas aplicaciones si son responsables en procesos de revisión de autenticación y autorización de recursos, puesto que; sus direcciones url son las API que consumen las aplicaciones clientes como las SPA.

Este proceso puede delegarse a un tercero como el caso de Auth0 o un API *gateway* en donde el programador y la empresa que brinda el servicio es responsable del nivel de seguridad, en el caso de Microsoft emplean el concepto de OpenID Connect.

Las aplicaciones web que funcionan en el servidor, son aquellas que reciben de manera directa las peticiones maliciosas de los atacantes, es importante mantener en mente los principios de sanitización, validación y escapado de los datos de salida, en el caso del manejo de errores, ser muy metódico para no exponer información sensible de manera pública en las caberas de respuesta de error por lo general, se encuentran con *status code* 3xx, 4xx, 5xx.

Una aplicación web tiene dos comportamientos marcados, el primero funcionar como una aplicación completa, provisiona al usuario tanto de *frontend* como *backend*, pero puede que su diseño conceptual sea limitado solo a brindar rutas de acceso a recursos (*backend*), este diseño tiene la denominación de API.

1.2.3. Aplicaciones móviles nativas e híbridas

Las aplicaciones nativas están diseñadas para ejecutarse en un solo sistema operativo móvil, son desarrolladas en los lenguajes de programación especificados por los fabricantes y emplean sus SDK, como el caso del lenguaje Objective-C para iOS, Java para Android, y .Net para Windows Phone.

Las aplicaciones híbridas tienen el marco conceptual de un desarrollo único y su compilación permite que sean instalables en cualquier sistema operativo móvil, en este caso, se emplean aplicaciones SPA que en su proceso de compilación tienen la opción de configurar el sistema operativo de despliegue, que es posible con herramientas como Apache Cordova.

Más allá de su arquitectura, una aplicación móvil tiene la posibilidad de funcionar como un monolito, es decir, no necesita recursos externos o como un cliente que consume servicios, este último paradigma necesita una comunicación segura mediante el uso de *tokens* de sesión (luego del ingreso de credenciales) o delegar el proceso a una autenticación por redes sociales.

1.2.4. Demonios o aplicaciones de servidor

Existen aplicaciones que trabajan sin la necesidad de la intervención humana, tienen tareas puntuales y específicas, un ejemplo es la ejecución de scripts de

automatización de respaldos, copias de seguridad de archivos y tareas programadas por los administradores del sistema.

Estos scripts pueden integrarse con *tokens* de autenticación o claves de servicios externos, por ejemplo, el uso de *websockets* en una aplicación en tiempo real, que permite enviar notificaciones a los usuarios, como ejemplo el proveedor de servicios en tiempo real Pusher no expone las credenciales del usuario, sino que genera un id de aplicación y una clave secreta de acceso.

1.3. Almacenamiento en el Cliente

Los tipos de aplicaciones que hacen uso de JWT para procesos de seguridad son sumamente amplios, para el presente estudio, se contemplará solo las aplicaciones web, esto en el marco del cambio en el orden de las vulnerabilidades según el Top Ten de riesgos de seguridad en aplicaciones web de OWASP del año 2017 al 2021.

Las aplicaciones para navegadores (de escritorio o móviles), las cuales, permiten interpretar el código del servidor, generan la interfaz que el usuario utiliza en su día a día, estos navegadores tienen distintos espacios para almacenar la información, que se envía y recepta en cada petición (*request*), según la guía de desarrollo de Mozilla realizada por Nachec & Enesimus (n.d.). existen el uso de *cookies*, almacenamiento web y *service workers*.

Las cookies son un método tradicional y común para almacenar información en el almacenamiento web, se tiene los espacios local, de sesión o IndexedDB que permite guardar documentos, no todas las guías de desarrollo o los programas de estudio enfocados en el desarrollo web hacen énfasis en estos conceptos.

1.3.1. Cookies

Las *cookies* se emplean en las peticiones web para intercambiar información entre el servidor y el cliente, permiten almacenar datos y hacerlos persistentes por un periodo de tiempo, se visualiza en los navegadores en la sección de cookies o con las herramientas de desarrollo.

Este medio de almacenamiento es el más usado hoy en día, es empleado en las políticas de *cookies*, por ejemplo, Google (2020). menciona emplea cookies para funcionalidades como las preferencias de usuario, seguridad, para autenticar usuarios "…cuando interactúan con los servicios", analíticas y publicidad para que los anuncios sean personalizados según las preferencias configuradas.

Para tener una mejor gestión de cookies en temas de seguridad, se recomienda emplear las banderas de *Secure* y *HTTPOnly* al crearlas, con esto prevenir que sean modificadas por parte de un atacante, por las vulnerabilidades identificadas como: secuestro de sesión, *XSS* y *Cross-Site Request Forgery* CSRF.

1.3.2. Almacenamiento Local y de Sesión (Local and Session Storage)

Son espacios de almacenamiento en el navegador, independientes entre dominios y son accesibles mediante código JavaScript, su diferencia radica en el ámbito de uso y persistencia de datos, se limita a la pestaña abierta en cada sesión, si ésta o el navegador se cierran, se pierde la información.

El espacio local es más utilizado según Mozilla en su blog de desarrollo, dado que permite una persistencia similar a las cookies, el navegador cerrarse o si el usuario cambia de pestaña el estado perdura, no pierde datos, estos espacios almacenan solo información en texto plano, datos sencillos.

1.3.3. IndexedDB y Service workers

Son espacios de almacenamiento moderno, usado por aplicaciones que necesitan persistir en el cliente datos complejos o en el momento de que el usuario pierde conexión, son gestionados con el lenguaje de programación JavaScript.

En el caso del uso de API IndexedDB las aplicaciones mantienen una estructura de datos según su necesidad, es tan versátil almacena archivos de video, texto e imágenes, es de uso asíncrono y su compatibilidad es limitada, por lo que, no se emplearía en todos los navegadores.

Para Gaunt (2020). un *service worker* es un proceso que el navegador ejecuta en segundo plano, es independiente del usuario y trabaja sin necesidad de conexión

a internet, lo que permite que un cliente usar la aplicación fuera de línea y sincronizar sus datos al restaurarse la conectividad.

El *servicie worker* es una funcionalidad que el desarrollador tiene que registrar y activar, funciona en los principales navegadores, brinda prestaciones para notificaciones *push* y acceder a *IndexedDB*.

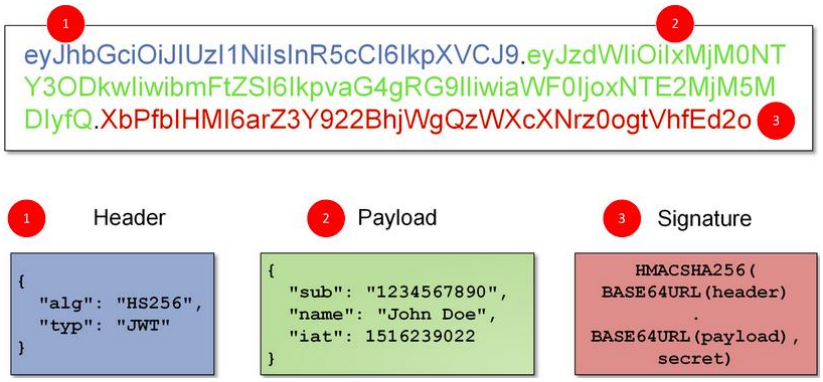
1.4. Tipo de Tokens JWT

El *token* JWT tiene la siguiente estructura: *Head*+"."+*Payload*+"."+*Signature*, en el *Head* está el tipo de *token* y el algoritmo para el cálculo del *Signature*, en *Payload* está almacena la información a ser enviada de forma segura, finalmente, el *Signature* contempla la suma de verificación para la mantener integridad de la información contenida, según Peyrott (2018). Son de tipo firmados o encriptados.

1.4.1. JSON Web Firmado (JWS)

Es el tipo de *token* más utilizado, mantienen la estructura básica de 3 partes, se visualiza en la Imagen 1, el *header*, payload y *signature*, separados por un punto ".", la última sección permite verificar la integridad de los datos en la cabecera y la carga, lo realiza mediante un hash que es calculado según el algoritmo establecido en el *header*, los algoritmos de firma comunes son: HMAC con SHA (HS), RSASSA PKCS1 con SHA (RS), ECDSA con P-256 y SHA-256 (ES).

Imagen 1 Estructura JWT Firmado



Fuente: Jones et al. (2015)

En la estructura de la Imagen 1, se aprecia que en la parte superior está la información codificada y firmada, las partes 1 y 2 son generadas mediante una codificación base64 (reversible) y la firma es la suma de verificación.

Cada algoritmo especificado tiene una manera de firmar los datos e implementar claves con formato de cadena de caracteres, se sugiere de una longitud considerable (más de 16 caracteres) con caracteres especiales y alfanuméricos, otros hacen uso de la infraestructura de clave pública para la verificación de integridad.

1.4.2. JSON Web Encryption (JWE)

Los JWS son *token* que permiten verificar la integridad de los datos, pero son reversibles, lo que deja abierta la posibilidad de que un atacante lo intercepte y ver la información que hay en su interior, una alternativa son los *tokens* encriptados o JWE por sus siglas.

El *token* encriptado hace uso del mismo principio de la infraestructura de clave pública, por lo que emplea un par de llaves (pública y privada), la llave pública será enviada al cliente junto a la especificación del algoritmo de encriptación, mientras que la llave privada estará únicamente en el servidor que descifrará la información, lo que permite disminuir un ataque por interceptación del *token* y que el atacante agregue datos dinámicamente.

Imagen 2 Estructura JWT Encriptado

Fuente: Jones et al. (2015)

En la imagen 2 están los 5 elementos de un JWE, el *header* similar al JWS pero, no se emplea el valor de "*typ*" sino "*enc*", *Encrypted Key* es considerada la llave para generar el cifrado, el valor *Vector* generalmente es un valor aleatorio, según Jones et al., (2012). de 96 bit, el *payload* son los datos encriptados mediante la clave, el vector y los datos no cifrados, finalmente, *Tag* repreenta un valor generado del cifrado y permite validar el contenido.

En los JWE, se utilizan los siguientes algoritmos de encriptación: RSA, AES, Curvas Elípticas, PKCA o de manera directa mediante *Content Encryption Key* (CEK), con el uso de una cadena de caracteres en texto plano, este tipo de *token* de igual manera codifica sus secciones en base64.

Finalmente existen dos especificaciones que permiten conocer más a fondo sobre la construcción de los *tokens Json Web Keys* (JWK) y *JSON Web Algorithms*, el primero describe las cabeceras de los *tokens* y el segundo los algoritmos de cifrado o suma de verificación empleados en los JWT, JWS y JWE.

El tipo de aplicación, el espacio de almacenamiento y el tipo de *token,* que se emplean en el desarrollo de software, marcan y determinan el nivel de seguridad en autenticación y autorización, la comunicación ideal será realizada por el protocolo HTTPS, es importante analizar, que se transmitirá o no al cliente.

Por lo general en el diseño de aplicaciones, no se asumen estas consideraciones y adoptan convenciones en base a la literatura existente (formal e informal), se aprecia en el acápite de trabajos relacionados, que no todos abordan el tema de manera integral, ni mencionan el medio o datos a transportar, pese a existir un estándar todos realizan una implementación personalizada o basada en soluciones de terceros.

CAPÍTULO II. DISEÑO METODOLÓGICO

2.1. Metodología de la Investigación

La investigación está enfocada en un área muy puntual del desarrollo de software, es por ello, que se emplea un tipo de investigación bibliográfica, con la finalidad de analizar la información existente sobre los JWT en fuentes como libros, artículos científicos, libros blancos, blogs corporativos, entre otros.

Para unificar la información, que se ha recopilado, se empleará el método inductivo, el objetivo es reducir las ambigüedades en el momento de que un desarrollador intente aplicar un estándar, los componentes subjetivos y la falta de experiencia influyen directamente en el éxito o fracaso de la implementación de JWT en procesos de autenticación y autorización.

Se implementará una técnica de análisis de casos, en la bibliografía se encuentran dos tipos de escenarios, el primero que son recomendaciones generales sobre los *tokens*, mientras que en otras fuentes, se tienen experiencias directas con casos de implementación, se emplea un *checklist* para realizar la verificación de las fuentes bibliográficas, finalmente los estudios primarios serán contrastados con el cumplimiento de los criterios de inclusión y su aporte con base a la respuesta de las preguntas de investigación.

2.2. Metodología de Desarrollo

Para la metodología de desarrollo, se empleará una Revisión Sistemática de Literatura (RSL) que según Tramullas (2020). “...resultan clave para identificar tendencias y nuevas áreas de investigación...”, que se llega a sintetizar la información de forma crítica y permite consolidar un “todo” acerca del tema investigado.

La presente RSL permitirá conocer las propuestas, que se han generado en base a las recomendaciones de seguridad en el uso de *tokens* JWT dentro de las aplicaciones web, de forma adicional dejará en evidencia las vulnerabilidades que afectan a los mismos en los procesos de creación, almacenamiento y transporte desde el servidor de autenticación y autorización hasta el navegador del usuario.

Para la revisión sistemática de literatura, se proponen las siguientes preguntas de investigación:

PI1.- ¿Cuáles son las buenas prácticas de seguridad, que se recomiendan en el uso de tokens JWT?

PI2.- ¿Cuáles son las recomendaciones en el transporte y almacenamiento de tokens JWT?

PI3.- ¿Cuáles son las principales vulnerabilidades de los *tokens* JWT?

Para resolver las preguntas de investigación, se define varias palabras clave, que son singulares y permiten identificar estudios de forma rápida, por ejemplo: web, *token* y las siglas correspondientes a los tipos de *token* JWT, JWE, JWS; se creó la siguiente cadena de búsqueda que emplea conectores lógicos:

token y web y (JWT o JWE o JWS)

La RSL, se realiza dentro de las bases de datos digitales SCOPUS, SCIENCE DIRECT, DIALNET y Biblioteca Digital existentes en la Pontificia Universidad Católica del Ecuador - Sede Ambato (PUCESA), en el caso de que las publicaciones no sean suficientes, se incluirán fuentes no formales como blogs reconocidos y publicaciones realizadas por expertos en el área.

Son tomados en cuenta los estudios que presentan los siguientes criterios de inclusión (CI):

CI1.- Son redactados en el idioma inglés o español.

CI2.- Están publicados en el rango de años de 2018 a 2021.

CI3.- Hacen referencia a la implementación de *tokens* en aplicaciones web.

No serán tomados en cuenta los estudios según los siguientes criterios (CE):

CE1.- Contienen palabras clave, pero no realizan un aporte significativo sobre la temática de estudio.

CE2.- Repitan las mismas políticas implementación de *tokens* en aplicaciones web.

Para garantizar que los estudios analizados tienen un enfoque en la seguridad del uso de JWT, se aplican los siguientes criterios de calidad (CC):

CC1.- Tienen relación directa con prácticas de seguridad sobre el uso de *tokens* de tipo JWT, JWS, JWE.

CC2.- Mencionan las vulnerabilidades que afectan a los *tokens* estudiados.

Con los criterios de inclusión, exclusión y calidad, se realiza la búsqueda de estudios primarios mediante el siguiente procedimiento:

1. Adaptar la cadena de búsqueda en los motores y aplicar filtros.
2. Se aplican criterios de inclusión y exclusión para depurar los estudios seleccionados.
3. Se aplican los criterios de calidad para obtener los estudios que serán objeto de análisis.
4. Se realiza la lectura de los artículos de los resultados de búsqueda para obtener posibles estudios primarios.

CAPÍTULO III. ANÁLISIS DE LOS RESULTADOS DE LA INVESTIGACIÓN

Al realizar la RSL, se identificó que no existe una gran cantidad de estudios primarios a ser considerados, la Tabla 1 muestra el resultado del proceso de selección, de un total de 1138 documentos analizados, se seleccionaron 7, los cuales cumplen con los criterios establecidos, es importante recalcar que cada base de datos tiene un proceso de búsqueda particular, debido a ello los resultados sumamente amplios fueron descartados, se continua con búsquedas particulares, estos motores fueron DSpace de la PUCESA y Dialnet, cuyos filtros de búsqueda son menos amigables en comparación con ScienceDirect y Scopus.

Tabla 1.- Ejecución RSL

Procedimiento	DSpace PUCESA	Dialnet	Science Direct	Scopus	Total
Adaptar la cadena de búsqueda en los motores de búsqueda y aplicar filtros.	159	886	67	26	1138
Criterios de inclusión y exclusión.	1	192	3	19	215
Criterios de Calidad.	1	0	2	5	8
Lectura de los Artículos.	0	0	2	5	7

Fuente: Elaboración propia

Los documentos son recopilados y organizados en el gestor bibliográfico Mendeley, se procede a realizar una lectura analítica sobre el aporte que brindan a las preguntas de investigación, luego de este análisis, se identifica que no es suficiente la información recopilada, por este motivo, se incluyen aquellos trabajos relacionados en el Estado del Arte y La Práctica de la presente investigación.

Estos documentos adicionales son emitidos por empresas reconocidas en el ámbito del desarrollo de software y la seguridad de la información, se consideran entidades como OWASP y Auth0 con sus respectivas guías o *white papers* publicados, son 4 publicaciones que serán incluidas.

3.1. Resultados de la RSL

Los estudios primarios son codificados como EP y numerados desde 1 hasta 11, se realiza un resumen en la Tabla 2 sobre los EP que aportan a las preguntas de investigación (PI), que se plantearon en la fase de planificación.

Tabla 2.- Resultado RSL

ID	TÍTULO	P1	P2	P3
EP1	Cobot attack: a security assessment exemplified by a specific collaborative robot.	X		X
EP2	CSRF protection in JavaScript frameworks and the security of JavaScript applications.		X	
EP3	Securing IoT Platforms.	X	X	X
EP4	Framework To Secure The Oauth 2.0 And Json Web Token For Rest Api.	X	X	X
EP5	An authentication based scheme for applications using JSON web token.	X		X
EP6	Exploring Lattice-based Post-Quantum Signature for JWT Authentication: Review and Case Study.	X		X
EP7	An Extensive Formal Security Analysis of the OpenID Financial-grade API.		X	
EP8	JWT Handbook	X		X
EP9	Token Best Practices	X	X	
EP10	Token Storage		X	
EP11	REST Security Cheat Sheet	X	X	X

Fuente: Elaboración propia

Los estudios coinciden en ciertas recomendaciones como, se muestra en la Tabla 2, un estudio sustentaría las tres preguntas de investigación, sin embargo, tienen diferencias y consideraciones puntuales por ello, se recopilan los aportes de la siguiente manera:

PI1.- ¿Cuáles son las buenas prácticas de seguridad, que se recomiendan en el uso de *tokens* JWT?

EP1 Se invalidará los *tokens* al momento en el que el usuario cierra sesión.

EP3 Al crear un *token,* se le asignará un tiempo de vida.

EP4 Sí se crea un *token* y se almacena en una cookie, se activará la bandera *HTTP Only* y *Secure Cookie* en el servidor web.

EP5 Para prevenir ataques de fuerza bruta, se genera un *token* por cada petición que realiza el usuario, deja el valor del intervalo entre peticiones como el tiempo máximo para el atacante. Para evitar predecir el proceso de generación del *token,* se agrega un valor único y randómico en el *payload*, así la firma será distinta, finalmente la aplicación validará los permisos y privilegios del *token* en cada petición.

EP6 Las firmas de los JWT emplearán algoritmos *post-quantum* como DILITHIUM, pese a que incremente el tamaño de la firma tiene una mejor seguridad con una pérdida mínima de rendimiento.

EP8 Los *tokens* serán validados según la necesidad de la aplicación o políticas empresariales.

EP9 No crear tokens con información sensible en el *payload*, específicamente en *tokens* que son firmados, dado que la codificación es reversible.

EP11 Al emplear claves de texto o infraestructura de clave pública (PKI) en un JWT, utilizar claves fuertes y de buena longitud, para prevenir ataques de fuerza bruta o que sean predichos fácilmente, al almacenarlos en cookies, se considerarán las banderas *HTTP Only* + *Secure, Same Site* + *Cookie prefix*.

PI2.- ¿Cuáles son las recomendaciones en el transporte y almacenamiento de tokens JWT?

EP2 Recomienda almacenar *tokens* en cookies de sesión para prevenir ataques *Cross-site request forgery* (CSRF).

EP3 Los *tokens* serán transmitidos por un canal seguro HTTPS, aquellos válidos y no válidos o expirados, se almacenarán en un repositorio en el servidor que los generó para su validación.

EP4 Para prevenir ataques CSRF emplear la política de orígenes cruzados en el servidor web *"Cross Origin Resource Sharing*" (CORS) y para evitar un ataque de hombre en el medio (MitM), se tendrá un *token* cifrado JWE.

EP7 Para una comunicación efectiva, se empleará multual TLS (mTLS) o autenticación mutua, en un modelo web tradicional HTTPS, el servidor envía su certificado al navegador del usuario, en cambio en mTLS, ambas parten intercambian certificados.

EP9 El sistema de autenticación, validación y autorización será diseñado en base a la menor cantidad de peticiones posibles, para disminuir la superficie de ataque o que el *token* sea interceptado.

EP10 Se recomienda almacenar los *tokens* en la memoria del navegador, con el uso *web workers* para su transmisión y almacenamiento.

EP11 Se sugiere guardar el *token* en el almacenamiento de sesión del browser, para prevenir peticiones de lugares no autorizados puede emplearse *Content Security Police* (CSP).

PI3.- ¿Cuáles son las principales vulnerabilidades de los tokens JWT?

EP1 Las aplicaciones que implementen JWT tienen la posibilidad de ser vulnerables al secuestro de sesiones.

EP4 Las aplicaciones con JWT serían vulnerables ante ataques de *Cross Site Scripting* (XSS), dado que al lograr inyectar código JavaScript en el navegador, se tendría acceso al almacenamiento. Si un *token,* se almacena en una cookie tendría un vector de ataque mediante CSRF y finalmente interceptarse con un ataque MitM.

EP5 Existe la posibilidad de conflictos en los roles y permisos de usuario si en la aplicación, se realiza una edición de privilegios, debido a que el *token* fue emitido con antelación. El *token* es susceptible a técnicas de fuerza bruta y llegar a ser predecible.

EP6 Otra técnica es *token impersonation*, al no validarse de forma adecuada, un atacante cambiaría los datos dentro del *token* y ver información de otro usuario. Los algoritmos de cifrado propuestos hasta el momento como RS256 y ES256 serían vulnerados mediante ataques de computación cuántica.

EP8 Un *token* que no contenga firma posee una vulnerabilidad, si el ente validador no toma en cuenta este valor antes de decodificar la cadena, un JWT mal diseñado impactaría en el rendimiento de la aplicación, el consumo de ancho de banda y cantidad de peticiones a base de datos, no se aceptarán *tokens* con una especificación de algoritmo “*none*”.

EP11 En el caso de que el usuario no cierre la sesión y deje un *token* sin revocar, finalmente un JWT estaría sujeto a criptoanálisis.

Discusión

Al realizar la RSL, se aprecia que muchos estudios hacen referencia al uso de JWT de forma tradicional o empírica, es decir, emplean un plugin o software de terceros para su implementación, si las librerías tienen una madurez considerable y están constantemente actualizadas es muy probable que el *token* sea un dato seguro, pero se evidencia que en los EP tendrían vulnerabilidades importantes.

La filosofía de “instalar y usar” permite que los desarrolladores de software lleguen a caer en un exceso de confianza, el *token* provisto por las librerías de terceros no tiene políticas de restricción por sí mismo EP8 menciona, que se valida según las reglas establecidas por la empresa y transfiere así la responsabilidad al equipo de desarrollo de software.

En procesos de autenticación y autorización, se agregan restricciones y controles a nivel de aplicativo, es común el uso de clases de tipo “*middleware*” o “*guard*”, que permita mitigar acciones no autorizadas, antes de que impacten en los recursos expuestos por la API o el servicio, se empleará el principio de menor privilegio y desconfiar en los datos que sean enviados desde el cliente.

Se han considerado recomendaciones que son genéricas para los proyectos, las políticas de CSP y CORS no son nativas del *token*, pero ayudan de gran manera a rechazar orígenes no autorizados, en las comunidades y foros, se recomienda emplear estas políticas de manera abierta, lo cual, sería mejor no tenerlas.

En cuanto al uso del almacenamiento en el cliente, se aprecia que existen diversos criterios en los estudios, se menciona el uso de *cookies*, *web storage* e incluso la memoria del navegador, pero es interesante que EP10 recomienda el uso de *web workers* (procesos del navegador en segundo plano), aislados de la aplicación principal para precautelar la confidencialidad del *token*.

Para EP7 el uso del protocolo HTTPS puede no ser suficiente en el intercambio del *token* entre el servidor y el cliente, en el caso de aplicaciones bancarias, por ejemplo, mencionan que sería recomendable usar mTLS, proceso que permiten una validación doble de la identidad de las partes que intervienen en la

comunicación, el servidor valida que el cliente es quien dice ser y viceversa, con la interrogante de si HTTPS es suficiente para prevenir ataques MitM.

El uso de web workers, mTLS y cifrado *post-quantum* (EP6) permite hablar de niveles de seguridad en el uso de JWT, en donde para aplicaciones críticas o con un grado de confidencialidad alto, se emplearía prácticas sumamente rigurosas, pero a cambio, se tendrá en mente que el costo de la aplicación y el tiempo de entrega serán extremadamente altos, más allá del conocimiento técnico en el equipo de desarrollo.

Finalmente, se exhorta a la comunidad de desarrollo de software, a dejar de ver los procesos de seguridad de las aplicaciones como empíricos, en donde el uso de credenciales o un *token* es suficiente para estar “seguros”, se motiva a que sean considerados dentro de los requerimientos con la debida rigurosidad.

3.2. Propuesta

Con base en los resultados obtenidos de la RSL, se evidencia la necesidad de unificarlos y emitir una propuesta de buenas prácticas de seguridad en el uso de JSON Web *tokens*, incluye sugerencias de los procesos mínimos de restricciones, que se emplearán en una aplicación.

3.2.1. Consideraciones Generales

Se sugiere realizar una clasificación de los datos que serán almacenados en la API, con la finalidad de identificar el tipo de *token* (firmado o encriptado) y las recomendaciones por aplicar.

Al emplear una librería o plugin de terceros, verificar que cumpla con las restricciones de seguridad que requiere el diseño de la aplicación a crear.

Una aplicación tiene la opción de emplear más de un *token* JWT.

En el servidor de emisión del *token,* se validará autenticación, roles y permisos sobre el uso del *token* en cada petición que se realice.

3.2.2. Creación del Token JWT

Para la creación de un *token,* se propone las siguientes recomendaciones:

- Se especifica de forma clara las reglas de validación que serán aplicadas sobre el *token*, ya sea para autenticación (contiene información del usuario) o autorización (contiene la información de roles y permisos del usuario).
- Se almacena la menor cantidad de información en el *token*, la información no es de carácter sensible.
- Si es un *token* encriptado, se recomienda utilizar el algoritmo RS256 o superior, en el caso de ser firmado, colocar una clave de longitud considerable, no predecible y con las debidas protecciones.
- En el caso de *token* cifrados y encriptados, se recomienda que las llaves sean cambiadas en periodos de tiempo definidos, por ejemplo: cada mes cambiar la clave de cifrado, esto dependerá de la criticidad de la información a resguardar.
- Se crea una lista de *tokens* autorizados junto a los usuarios asignados para prevenir, que se realice una impersonación del *token*.
- Se mantiene una lista actualizada de *tokens* revocados, esto permitirá descartar de forma inmediata *tokens* obsoletos, expirados y disminuye las posibilidades de que un atacante pueda reutilizarlos.
- Se recomienda asignar un tiempo de vida al *toke*n, luego del cual automáticamente sea revocado, dado que el *token* por sí mismo no puede invalidarse, otra manera es cerrar sesión.
- En el servidor de emisión del *token*, emplear cookies para los JWT, se recomienda, que se activen las banderas de: *HTTP Only + Secure, Same Site + Cookie*, el tiempo de vida corresponderá al mismo del *token*.
- En el servidor de emisión, se sugiere crear las políticas CSP, CORS y especificar los verbos HTTP (*GET, POST, OPTIONS*) que serán utilizados, para prevenir ataques de tipo CSRF.
- La aplicación o sistema de verificación de información del *token* no confiará en la información contenida en el JWT, al ser manipulable por un atacante (*tokens* firmados sin clave), se verifica con las listas de *tokens* creados y autorizados, antes de procesar la información.

- En el servidor de emisión, no receptar *tokens* que no tengan firma o que en el *header* no tenga especificado un tipo de algoritmo, serán desechados de inmediato.
- Se recomienda restringir el uso de solo un algoritmo de cifrado o firmado en los *tokens*, si se emplea encriptación (RS256), no se permitirán algoritmos de firmado (HS256).

En el caso de que la información sea extremadamente sensible, se recomienda:

- Emplear algoritmos de cifrado *post-quantum*.
- El uso del *token* en cada petición que realice el usuario, a más de emplear la clave de cifrado, puede agregarse un valor randómico en el cuerpo del JWT.
- No enviar el *token* completo, una variación es enviar el *header* con el *payload* en notación base64 y resguardar en el servidor la suma de verificación para contrastar la integridad de los datos.

3.2.3. Transporte del Token JWT

El transporte del *token,* no se limita a la transmisión mediante el canal de comunicación físico, sino que debe incluirse en como la aplicación cliente prepara los datos a ser trasmitidos por la web, si el cliente no realiza el procedimiento adecuado estaría expuesto ante una fuga de datos.

- El *token* será enviado en el protocolo HTTPS, de ser posible, se configuran las banderas de *Strict Transport Security*, por ejemplo: el tiempo de expiración puede igualarse con el tiempo de vida del *token*.
- Limitar las peticiones a los verbos HTTP especificados por el servidor de emisión del *token*.
- Rechazar las peticiones que no contengan un *header* de autorización, que no contengan el *token* en los datos de la petición, esto depende de la configuración de la aplicación.
- En el caso de información bancaria, militar o de alta confidencialidad, puede emplearse la opción de mTLS para que el cliente y el servidor logren verificar la identidad de su contraparte.

- En el caso de emplear un tercer servidor de emisión del *token* (federado), se asegura la comunicación entre los actores que interviene como: el servidor API de recursos, la aplicación cliente y el servidor federado con HTTPS o mTLS de ser el caso.
- Al emplear varios *tokens,* se recomienda especificar de forma clara que *token* es de autenticación y agregarlo en la cabecera o en el cuerpo de la petición.

3.2.4. Almacenamiento del Token JWT

El almacenamiento del *token* en el navegador responderá a las necesidades de la aplicación cliente, es de suma importancia analizar su impacto, lo ideal sería manejar estos datos en memoria, pero tiene las desventajas de que si el usuario cierra la pestaña perderá esa información y tendrá que autenticarse nuevamente, un comportamiento similar presenta el *session storage* pero es visible por un atacante con las herramientas de inspección del navegador o mediante la consola con programación JavaScript.

Whatsapp y Telegram a la fecha del presente estudio, no permiten que un usuario acceda a su aplicación web desde más de una pestaña, al hacerlo inmediatamente, se invalida la pestaña anterior, la principal diferencia es que las aplicaciones hacen que la transición sea muy sencilla con el uso de *workers*, que sería recomendado frente al uso en memoria *o session storage*.

En el caso de emplear *session storage, local storage* o indexedDB prevenir fuertemente los ataques XSS, puesto que por sí mismos no brindan esta protección, el uso de *local storage* es muy habitual en el desarrollo de software actual. Las *cookies* que son configuradas de forma segura ante ataques CSRF, ayudan a mitigar los problemas que presenta el *web storage*, una ventaja es el tiempo de vida, que se asigna a una cookie.

En la aplicación cliente no deben reflejarse pistas de los objetos de autenticación o su conformación, si la aplicación necesita enviar un dato adicional en el *token*, la llave de cifrado será protegida de manera rigurosa, al emplear herramientas de

ofuscación y cifrado del código fuente en el navegador, para prevenir que sea fácilmente accesible por un atacante.

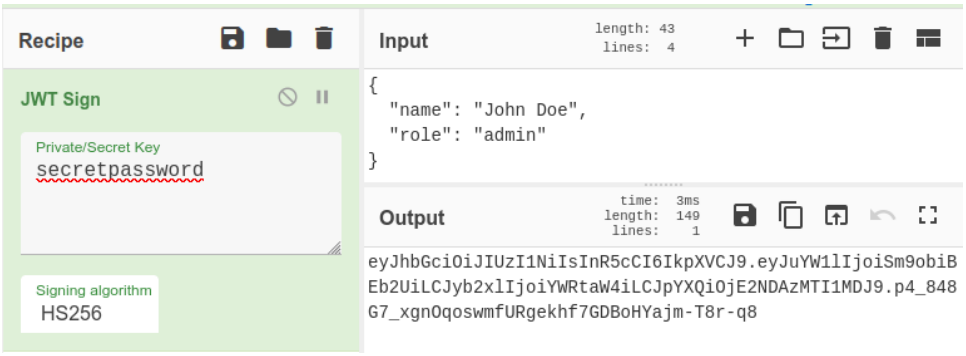
3.3. Pruebas de Concepto

Una vez identificadas las mejores prácticas, se realizan pruebas de concepto para verificar el impacto que tendría su omisión, en el primer caso, se busca verificar la resistencia del *token* ante ataques de fuerza bruta y en el segundo el impacto de las validaciones débiles de autorización, al final, se sugiere una línea base de recomendaciones a implementar, en el caso de no ser viable la implementación de todas las buenas prácticas.

3.3.1. Fuerza Bruta

En la propuesta de recomendaciones de creación de un JSON *Web Token*, se sugiere crear un *token* de tipo JWE para tener un mayor nivel de seguridad, dado que los *tokens* firmados, con una clave débil, serían rotos por técnicas de fuerza bruta, para la prueba, se empleará John the Ripper en un computador con un procesador Intel i7-8550u con frecuencia base de 1.80Ghz, 16 Gb de memoria *RAM* y una tarjeta de video AMD Radeon de 4Gb.

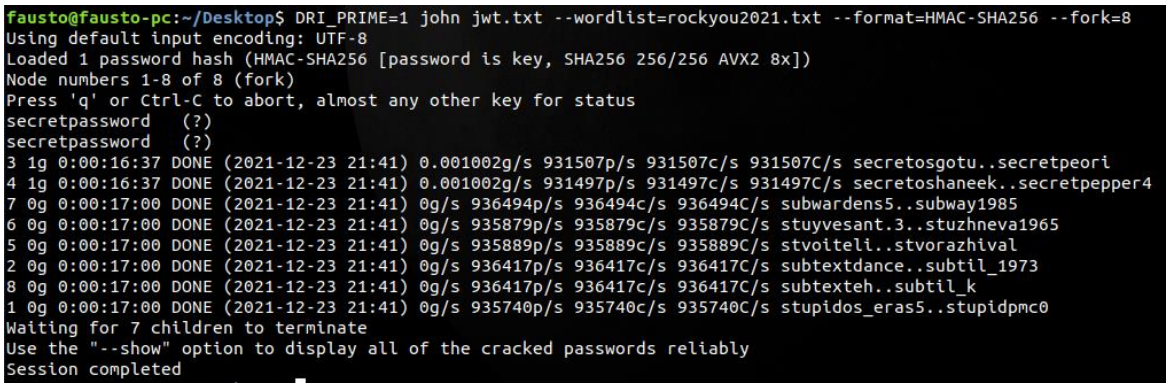
Imagen 3 Creación JWS inseguro 1



Fuente: Elaboración propia

Con la herramienta en línea cyberchef https://gchq.github.io, se crea un *token* genérico como en la imagen 3, con una clave de cifrado débil que no cumple los estándares mínimos como mayúsculas, minúsculas, caracteres especiales, números y cantidad de caracteres, se coloca el *token* en un archivo, finalmente, se ejecuta la herramienta John the Ripper desde la GPU.

Imagen 4 Resultado Cracking 1



Fuente: Elaboración propia

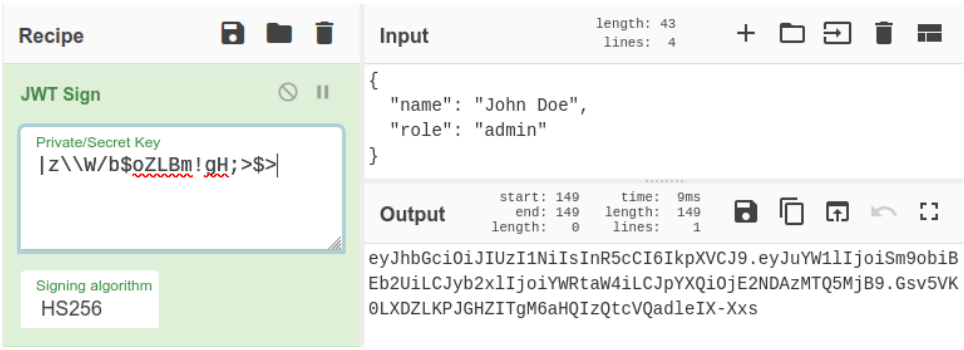
En la imagen 4, se evidencia que la utilidad encontró la clave luego de 17 minutos de análisis, se ejecutó el comando:

DRI_PRIME=1 john jwt.txt --wordlist=rockyou2021.txt --format=HMAC-SHA256 --fork=8

El valor “DRI_PRIME” permite que la ejecución del ataque, se realice mediante el uso de la GPU del computador, el segundo parámetro envía el archivo que contiene el *token* analizado, se utiliza como diccionario el archivo rockyou del año 2021 con cerca de 8.400 millones de claves filtradas y un peso de aproximadamente 100Gb descomprimido, “*format*” especifica el algoritmo de cifrado, finalmente fork permite el uso de paralelismo en el procesamiento.

Por la constitución de la clave, se interpreta de que es sencillo realizar un ataque de este tipo, sin embargo, con claves que son aparentemente “complejas” sucedería lo mismo, para el segundo caso de estudio, se utilizará la misma herramienta e información más el cambio de contraseña.

Imagen 5 Creación JWS inseguro 2



Fuente: Elaboración propia

En la imagen 5, se ve que el *token* creado tiene una clave de cifrado mucho más compleja, contiene caracteres especiales, letras mayúsculas, letras minúsculas y una longitud de 20 caracteres, lo cual, evidencia que cumple el criterio de "robustez", con estos nuevos datos, se procede a realizar el cracking nuevamente con John the Ripper.

Imagen 6 Resultado cracking 2

```
fausto@fausto-pc:~/Desktop$ DRI_PRIME=1 john jwt.txt --wordlist=rockyou2021.txt --format=HMAC-SHA256 --fork=8
Using default input encoding: UTF-8
Loaded 1 password hash (HMAC-SHA256 [password is key, SHA256 256/256 AVX2 8x])
Node numbers 1-8 of 8 (fork)
Press 'q' or Ctrl-C to abort, almost any other key for status
|z\\W/b$oZLBm!gH;>$> (?)
2 1g 0:00:18:54 DONE (2021-12-23 22:26) 0.000881g/s 931670p/s 931670c/s 931670C/s |zPo]MYw)..|zhoma5ub|
3 0g 0:00:18:54 DONE (2021-12-23 22:26) 0g/s 931811p/s 931811c/s 931811C/s ~~~tanusha~~~..~~~~~~~~~~~~~~~~~!
5 0g 0:00:18:54 DONE (2021-12-23 22:26) 0g/s 931803p/s 931803c/s 931803C/s ~~~tee..~~~~~~~~~~~~~~~~~~!
6 0g 0:00:18:54 DONE (2021-12-23 22:26) 0g/s 931795p/s 931795c/s 931795C/s ~~~tek....~~~~~~~~~~~~~~~~~~~~
4 0g 0:00:18:54 DONE (2021-12-23 22:26) 0g/s 931795p/s 931795c/s 931795C/s ~~~teddi~~~..~~~~~~~~~~~~~~~~~~
7 0g 0:00:18:54 DONE (2021-12-23 22:26) 0g/s 931795p/s 931795c/s 931795C/s ~~~tektonik~~~..~~~~~~~~~~~~~~~~~~~~
1 0g 0:00:18:54 DONE (2021-12-23 22:26) 0g/s 931795p/s 931795c/s 931795C/s ~~~tanechka..~~~~~~~~~~~~~~~~~~
Waiting for 7 children to terminate
8 0g 0:00:18:54 DONE (2021-12-23 22:26) 0g/s 931754p/s 931754c/s 931754C/s ~~~ten-ten..~~~~~~~~~~~~~~~~!
Session completed
```

Fuente: Elaboración propia

En el resultado de la segunda prueba, se aprecia en la imagen 6, al ejecutar el ataque la contraseña fue expuesta en 19 minutos aproximadamente, pese a ser más compleja que la primera, por lo que la vulnerabilidad del algoritmo de firmado dependerá directamente de la capacidad computacional del atacante, su posibilidad de obtener claves filtradas o crear diccionarios específicos, existen diversas formas de obtener esta información, sería tan sencillo como buscar un repositorio en GitHub como en la Imagen 7.

Imagen 7 Resultado búsqueda JWT_SECRET en Github

Fuente: Elaboración propia

Es por ello, que se recomienda emplear un *token* JWE encriptado, ya sea con algoritmos de clave pública o curvas elípticas, lo que lleva el desafío de cracking a

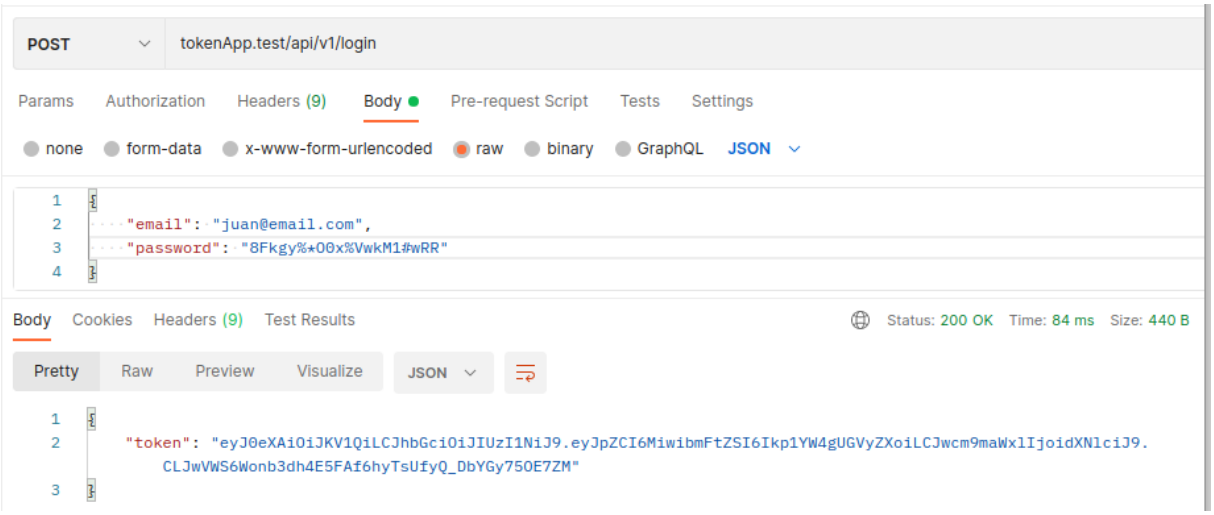
un nivel más elevado para el atacante, al realizar la suma de varias medidas de seguridad, estas debilidades serán mitigadas, una agrupación viable sería mantener la llave de cifrado o la clave privada en lugares que sean restringidos como carpetas del servidor protegidas por permisos u ocultas, no estarán versionados en repositorios públicos, su tiempo de vida será menor al posible tiempo de cracking y deben cambiarse de forma periódica.

3.3.2. Validación Débil

Para la prueba de concepto de validaciones de autorización, se realiza una aplicación “tokenApp” en el lenguaje de programación PHP con el *framework* de desarrollo Laravel, se exponen dos servicios, el primero un login mediante correo electrónico y contraseña cuya respuesta será un *token* JWT, el segundo permite ver el listado de usuarios existentes para el perfil administrador, la aplicación estará disponible en el repositorio de Github https://github.com/Fasuto/tokenApp, para simular la interacción entre el cliente y el servidor, se emplea Postman.

Como en el resultado de la prueba de fuerza bruta, si un atacante logra obtener la clave de cifrado de un JWT firmado, podrá fácilmente cambiar los datos contenidos y enviarlos al servidor para que sean procesados, si no cuenta con políticas de autorización maduras, el atacante haría uso de técnicas como el escalado de privilegios o impersonar el *token* haciéndose pasar por otro usuario.

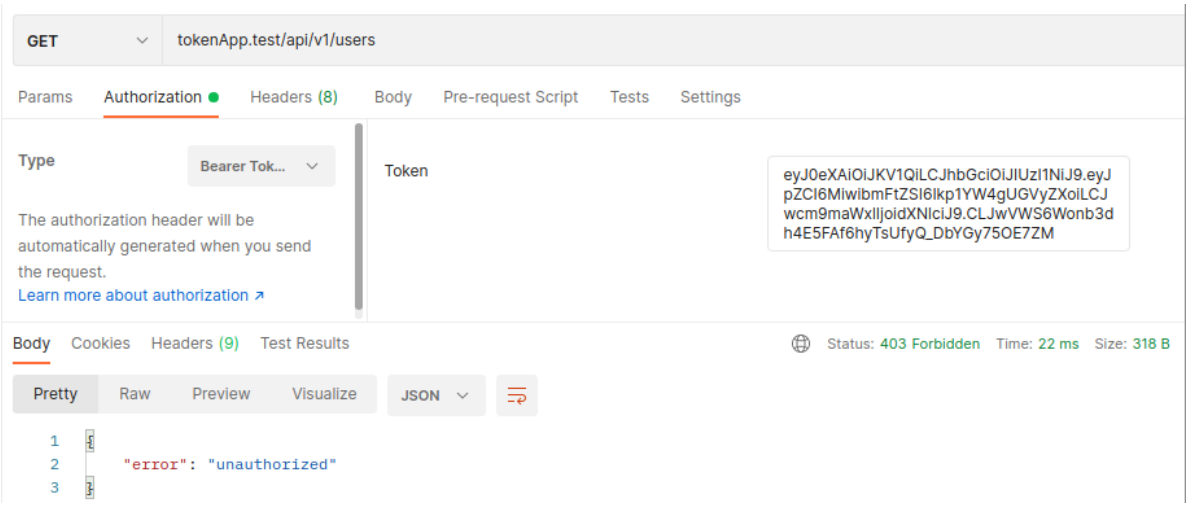
Imagen 8 Login correcto usuario Juan



Fuente: Elaboración propia

En la imagen 8, se identifica una petición normal de inicio de sesión en la aplicación, el usuario Juan sin perfil de administrador, tiene una clave fuerte para el manejo de credenciales, la aplicación retorna un *token* firmado vulnerable, Juan realiza una petición legítima para ver el listado de usuario, pero al no tener permisos su solicitud es rechazada, como se observa en la Imagen 9.

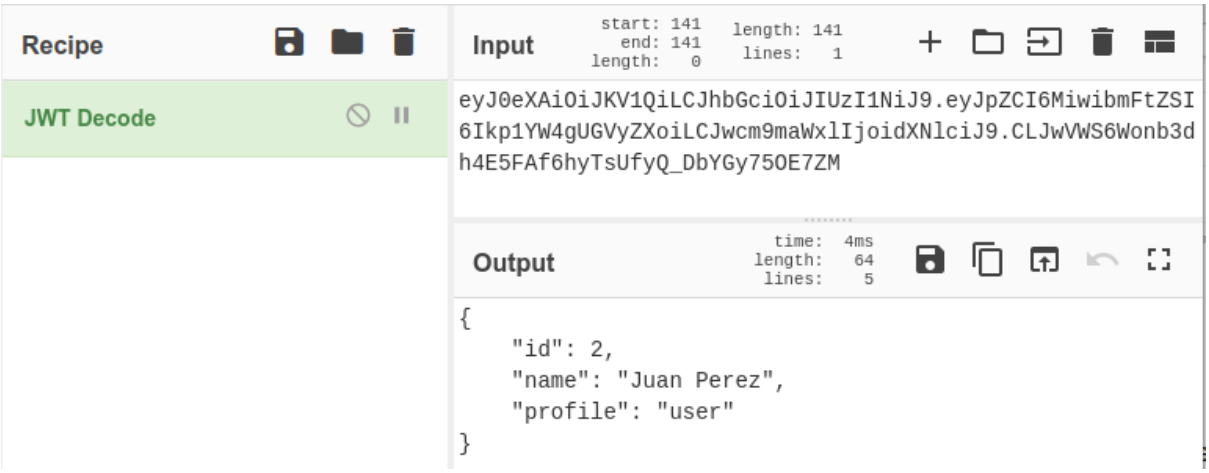
Imagen 9 Solicitud de lista de usuario rechazada



Fuente: Elaboración propia

Las imágenes 8 y 9 muestran el comportamiento ideal de la aplicación, pero un atacante ha logrado interceptar *token* y realizar el proceso de cracking, con ello obtiene como resultado que la contraseña de firmado es “secret” y edita el *token*, la imagen 10 muestra el *token* original con los permisos asignados en el proceso de autenticación y la imagen 11 el *token* modificado por el atacante.

Imagen 10 Información original del token



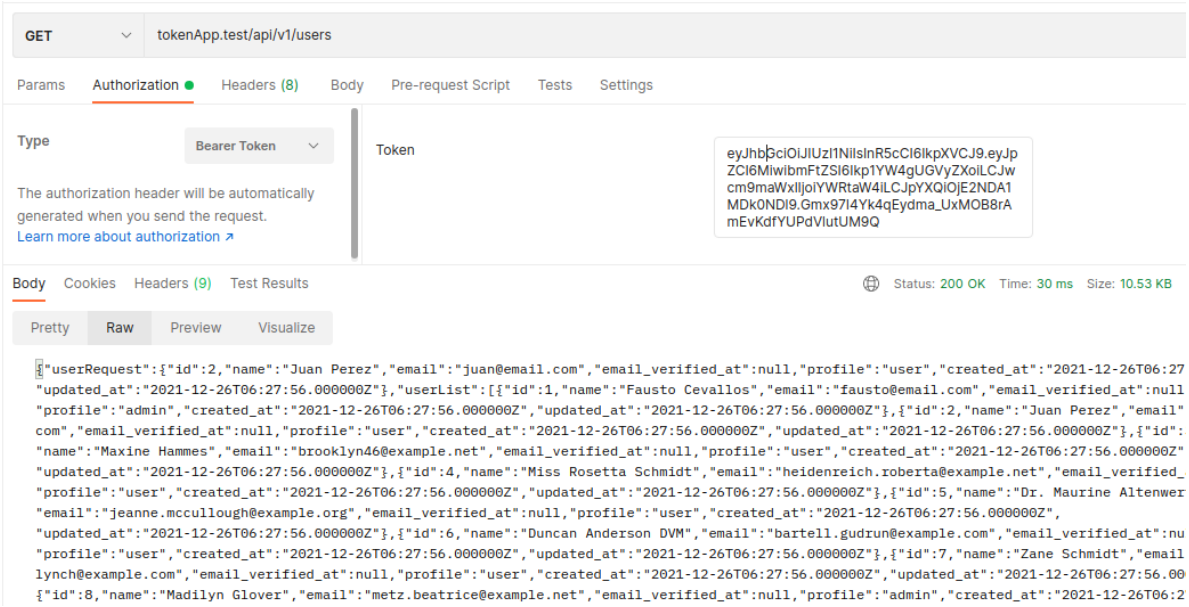
Fuente: Elaboración propia

Imagen 11 Información token adulterado

Fuente: Elaboración propia

En la Imagen 11, se observa que el atacante modificó el perfil del usuario para realizar un ataque de escalado de privilegios con el valor “admin” este *token,* se inyecta nuevamente en la aplicación y logra obtener el listado de usuarios restringidos por perfil, esto se evidencia en la Imagen 12.

Imagen 12 Ataque de privilegios exitoso

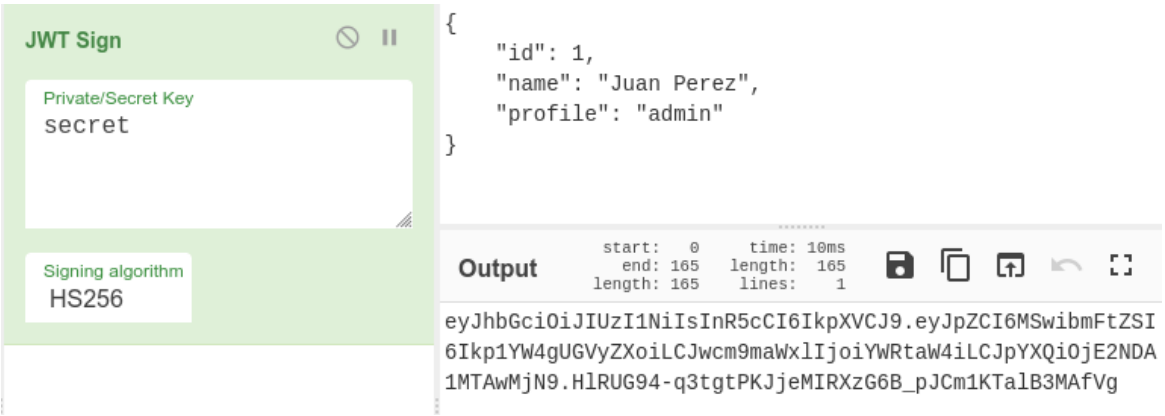


Fuente: Elaboración propia

Con el resultado de la Imagen 12 el atacante verifica que los datos retornados son “*userRequest*” usuario que hace la petición y “*userList*” el listado de usuarios, al no conocer las credenciales de acceso de los demás usuarios procede a realizar una

impersonación del *token*, para lo cual, tiene que cambiar el valor del id nuevamente y obtiene el *token* de la Imagen 13.

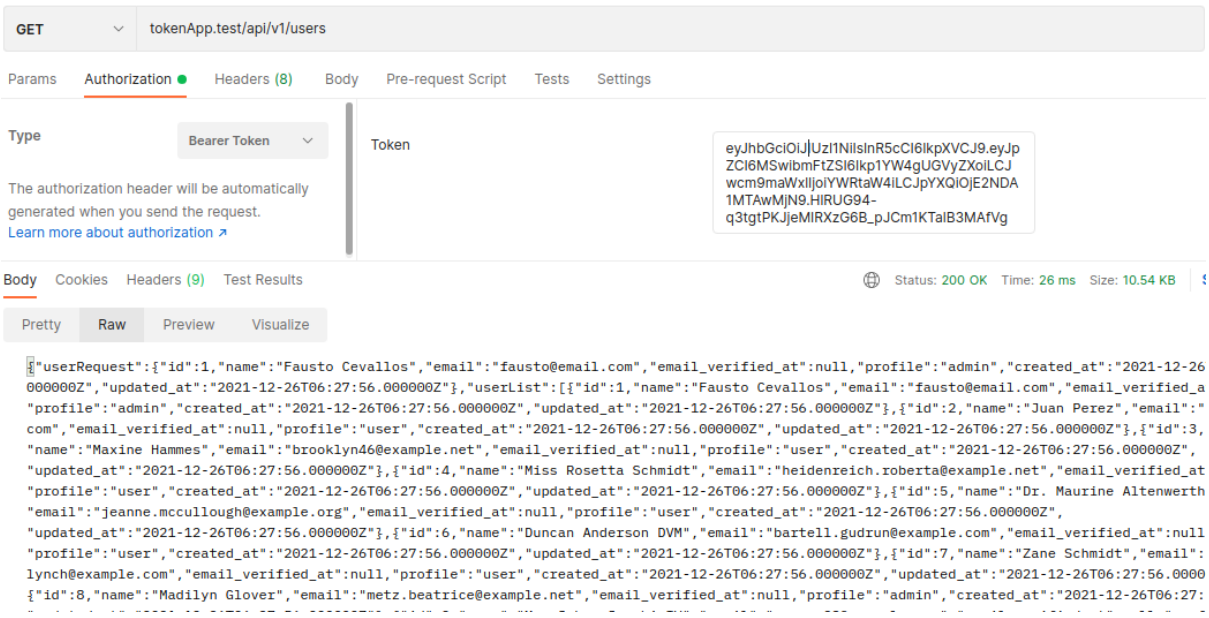
Imagen 13 Ataque token impersonation



Fuente: Elaboración propia

El atacante cambia el id del *token*, el usuario Juan tiene el id “2” pero, se envía al servidor con el valor “1”, identificador que no le corresponde, esto provoca que la aplicación vulnerable responda con los datos de otro usuario, permitiéndole así al atacante tomar cualquier cuenta de usuario sin necesidad de saber las credenciales de inicio de sesión y lograr una suplantación de identidad.

Imagen 14 Ataque token impersonation exitoso



Fuente: Elaboración propia

En la imagen 14, se identifica que los datos de *userRequest* varían de la imagen 12, por lo que el atacante tuvo éxito en su petición, ahora es el usuario Fausto

Cevallos con quién logró autenticarse sin necesidad de conocer las claves, con estos datos, se procede a verificar el código vulnerable.

Imagen 15 Middleware de autenticación

```
18  public function handle(Request $request, Closure $next)
19  {
20      $header = $request->header( key: 'authorization');
21      if(empty($header)){
22          return response()->json(['error'=>'authorization token is required'], status: 401);
23      }
24
25      $token = explode( separator: " ", $header)[1];
26      if(!PocJwt::verifyToken($token)){
27          return response()->json(['error'=>'authorization token is invalid'], status: 403);
28      }
29
30      $request->request->add(['user' => PocJwt::decodeToken($token)]);
31      return $next($request);
32  }
```

Fuente: Elaboración propia

La ruta de listado de usuarios del API publicado tiene una verificación de autorización en la Imagen 15, cada petición "*request*" tiene un proceso de verificación, el cual, consiste en: validar que exista una cabecera de autorización (líneas 20,21,22,23), validar la integridad de la firma del *token* (líneas 25, 26, 27,28) y agregar los datos internos del *token* a la petición (línea 30), finalmente la petición es enviada al controlador para su procesamiento (línea 31).

Imagen 16 Función vulnerable en el controlador

```
16  public final function index(Request $request): JsonResponse
17  {
18      if($request->user->profile !== 'admin'){
19          return response()->json(['error'=>'unauthorized'], status: 403);
20      }
21      $userRequest = User::findOrFail($request->user->id);
22      $users = User::all();
23      return response()->json([
24          'userRequest' => $userRequest,
25          'userList'=>$users
26      ]);
27  }
```

Fuente: Elaboración propia

En la Imagen 16, se identifica la lógica de código vulnerable, el *token* al contener toda la información de autenticación del usuario, permite que un desarrollador valide directamente los permisos sin contrastarlos con los originales al considerar el *token* como “seguro”, las líneas 18, 19 y 20 permiten al atacante ejecutar la escalada de privilegios, mientras que la línea 21 le permite impersonar usuarios, sin contar con una validación de los datos de entrada en las consultas.

Para resolver las vulnerabilidades, se aplicarán diferentes técnicas, para la prueba de concepto diseñada, se realizará un proceso de almacenamiento del *token* en la base de datos luego del primer login del usuario, en caso de existir, se tomarán los permisos asignados originalmente y se descarta los del *token*.

Imagen 17 Middleware de autenticación corregido

```
19    public function handle(Request $request, Closure $next)
20    {
21        $header = $request->header( key: 'authorization');
22        if(empty($header)){
23            return response()->json(['error'=>'authorization token is required'],  status: 401);
24        }
25
26        $token = explode( separator: " ", $header)[1];
27        if(!PocJwt::verifyToken($token)){
28            return response()->json(['error'=>'authorization token is invalid'],  status: 403);
29        }
30
31        $user = User::where('token',$token)->first();
32        if(empty($user)){
33            return response()->json(['error'=>'authorization failed'],  status: 403);
34        }
35
36        $request->request->add(['user' => $user]);
37        return $next($request);
38    }
```

Fuente: Elaboración propia

En la Imagen 17, se toma en cuenta el control agregado para prevenir las vulnerabilidades, en las líneas 31,32,33 y 34, se realiza la validación de la existencia del *token* y su respectiva relación con el usuario, en caso de no existir, se envía un mensaje “authorization failed”, en la línea 31,se toma la información del usuario de la base de datos, se desconfía de la información del *token* por si fuere editada y deja en evidencia que el id (dato sensible) no es necesario que sea incluido en el *token*.

En la imagen 18 se aprecia que fue eliminada la línea de búsqueda de usuario dado que ya fue validado con la base de datos.

Imagen 18 Función corregida en el controlador

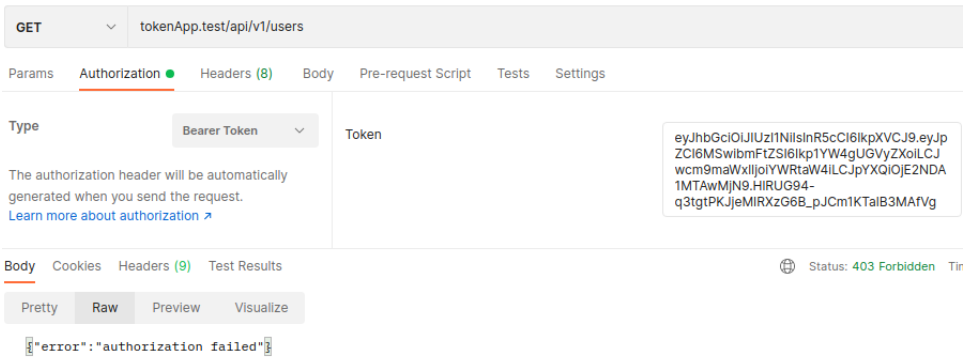
```php
public final function index(Request $request): JsonResponse
{
    if($request->user->profile !== 'admin'){
        return response()->json(['error'=>'unauthorized'], status: 403);
    }

    $users = User::all();
    return response()->json([
        'userRequest' => $request->user,
        'userList'=>$users
    ]);
}
```

Fuente: Elaboración propia

Finalmente, para verificar, que se han corregido las vulnerabilidades, se vuelve a inyectar en la petición del *token* malicioso de la Imagen 14 que contenía el cambio del id y rol del usuario Juan, a lo que el servidor emite como resultado una respuesta de autenticación fallida como, se aprecia en la Imagen 19.

Imagen 19 Ataque fallido



Fuente: Elaboración propia

En la revisión de estudios primarios, se menciona que el *token,* no es seguro por sí mismo, hay que contar reglas de autenticación y autorización maduras para prevenir intrusiones, se evidencia en la prueba de concepto pese a que el *token* y su clave de firma fueron filtrados, el atacante no escalará privilegios ni impersonar usuarios, comprometerá la información a la que el *token* esté autorizado.

3.3.3. Línea base de recomendaciones

Luego de ejecutar las pruebas de concepto e identificar algunas vulnerabilidades en el uso de JWT, se emiten las siguientes prácticas, como una línea base:

1. Especificar de forma clara las reglas de validación que serán aplicadas sobre el *token* y prevenir la inclusión de información sensible de manera innecesaria.
2. Se recomienda emplear un *token* de tipo encriptado JWE con un algoritmo mínimo RS256 con claves asimétricas.
3. Mantener una lista de *tokens* autorizados a realizar peticiones.
4. Asignar tiempos de vida cortos (menores al tiempo estimado de cracking del algoritmo).
5. No receptar *tokens* que no tengan firma o que en el *header* no tenga especificado un tipo de algoritmo.
6. Restringir el uso de solo un algoritmo de cifrado o firmado en los *tokens*.
7. Los datos de la aplicación serán enviados mediante protocolos de comunicación seguros como HTTPS o mTLS.
8. En el servidor de emisión crear las políticas de CSP, CORS y especificar los verbos HTTP (GET, POST, OPTIONS) que serán utilizados
9. Priorizar el almacenamiento de los *tokens* en memoria en el cliente.

En el caso de utilizar librerías de terceros considerar, que se encuentren en constante mantenimiento por su fabricante y actualizadas en los aplicativos que las implementen.

3.4. Viabilidad e Implementación

Existen dos escenarios para analizar la viabilidad e implementación de JWT, el primero es considerarlo en el diseño de una nueva aplicación como fase inicial, en la planeación de la arquitectura como un requerimiento no funcional; el segundo caso en una aplicación existente, donde existe la posibilidad de tener un coste elevado en tiempo de desarrollo, sin embargo, con las recomendaciones a continuación el proceso será mucho más sencillo y llevadero.

3.4.1. Aplicaciones Nuevas

Si una empresa o desarrollador busca asegurar sus aplicaciones con JWT, se recomienda tener en cuenta las siguientes reglas sobre el *token*:

1. El *token* es de tipo encriptado y sus llaves asíncronas serán de longitudes considerables, superiores a los 1024/2048 bits.
2. Establecer una política de aislamiento de claves en los ambientes de desarrollo, pruebas y producción, no serán versionadas.
3. Crear una política de cambios de claves.
4. Identificar el uso del *token*, si es empleado netamente para autenticación o para autorización, en el caso de autenticación como, se aprecia en las pruebas de concepto, no contendrán datos sensibles.
5. Se recomienda crear un único punto de entrada para la validación de autenticación y autorización, un *helper,* clase intermedia o una función, por la cual, deben enviarse todos los endpoints de la aplicación.
6. Manejar un tiempo de vida del *token*, el mismo que tendrá una duración corta y administrable por bases de datos o variables de configuración (.env).
7. El proceso de autenticación será el responsable de la creación del *token* una vez que el usuario haya ingresado sus credenciales.
8. En todos los *endpoints* de ingreso de datos del usuario, se realizará un proceso de sanitización, validación y escapado de caracteres.

Este tipo de diseño permite que la aplicación cuente con un nivel de seguridad aceptable y ahorre tiempo en el desarrollo de software, esto dado que el desarrollador, que se integre al proyecto o genere una nueva funcionalidad, no realizará procesos de autenticación y autorización, estos ya están predefinidos por lo que simplemente, se hace uso de ellos, el desarrollador estaría dedicado únicamente, a la funcionalidad requerida.

3.4.2. Aplicaciones Existentes

En este escenario en particular la limitante más fuerte es la percepción del impacto del cambio, por lo general los gerentes de proyecto o dueños del producto, se ven

sorprendidos por la magnitud que implica, no obstante, puede que sea más sencillo de lo que parece, para ello, se recomienda lo siguiente:

1. Crear una clase, método o función en la aplicación que realice el proceso de validación de autenticación y validación con JWT, según las recomendaciones 1,2,3,4,5,6 para una aplicación nueva.
2. Identificar el proceso de inicio de sesión de la aplicación para que haga uso de un mecanismo dual, es decir, en el caso de que una aplicación use sesiones y desee cambiar a JWT, por un periodo de tiempo aceptará los dos mecanismos de autenticación y autorización.
3. Una vez generado el mecanismo de autenticación dual, identificar los procesos de menor impacto de cambio para aislarlos, se realiza la implementación de la validación por JWT y eliminar el mecanismo antiguo, este paso será iterativo hasta completar la migración con procesos cada vez más complejos.

El tomar en consideración el punto 3 permite generar un dinamismo de confianza en el equipo de desarrollo y en el cliente, al realizar iteraciones pequeñas y progresivas permitirá evidenciar que el impacto es menor al dimensionado en una fase inicial.

Al ser una arquitectura con base en el consumo de servicios, lo ideal es que este proceso lo realice un microservicio, el cual, facilite emplear principios de desarrollo de software como el de responsabilidad única, al poseer una lógica propia y única el microservicio es susceptible a cambiar a una velocidad mayor, en contraste con editar archivo por archivo en cada punto de la lógica de negocio.

En los dos escenarios descritos, se recomienda, que se realicen pruebas de concepto para validar la madurez de su implementación, esto de forma indistinta si el *token* es generado de forma autónoma o se emplea una librería de terceros, en el caso actual de estudio, se realizó con el lenguaje de programación PHP, sin embargo, tiene la factibilidad de ser implementado en cualquier lenguaje de programación que acepte JWT.

CONCLUSIONES

- El proceso de revisión del estado del arte, sobre la construcción de un *JSON Web Token*, se identificó la importancia de que el equipo de desarrollo de software tenga conocimiento de que el *token* no es cien por ciento seguro y es reversible, es decir, lo que a simple vista parecen valores ilegibles, un atacante o usuario con conocimientos en decodificación, llegaría a obtener los datos internos.

- Se verificó que el *token* es susceptible a vulnerabilidades en cada una de sus etapas de aprovisionamiento, según la RSL, se ha encontrado que en su construcción es de vital importancia, que se contemple una clave de cifrado en lugar de solo aplicar la firma, en el transporte podría no llegar a ser suficiente el uso de HTTPS y en su almacenamiento *el web storage* no es la mejor de las prácticas.

- En la presente investigación, se han recopilado un conjunto de buenas prácticas que permiten mitigar las vulnerabilidades identificadas en los JWT y tienen un amplio campo de acción, en la creación, se menciona la integración de criptografía post cuántica para protección de los datos, en el transporte de información extremadamente sensible, se contempla el uso de mTLS y en el almacenamiento, se considera el uso de *web workers* o la memoria del navegador.

- Con la ejecución de las pruebas de concepto, se logra conceptualizar el problema de la utilización del *token* como un dato “seguro”, se realizaron pruebas de cracking de la contraseña del algoritmo de firmado, escalada de privilegios e impersonación de usuarios en un ambiente no seguro, al aplicar las buenas prácticas, se logró corregir las vulnerabilidades de la aplicación e incrementar su percepción de seguridad.

RECOMENDACIONES

- Se recomienda realizar una investigación sobre el impacto que tendría la implementación de algoritmos *post-quantum* y protocolos de comunicación segura como mTLS, en las aplicaciones actuales que usan JWT.

- Se recomienda a las empresas y personas que realizan desarrollo de software, mantener ejercicios de *cracking* de contraseñas y de *tokens*, con algoritmos tradicionales y *post-quantum*, con la finalidad de obtener métricas para realizar un protocolo de periodicidad para el cambio de contraseñas y llaves asimétricas.

- En las aplicaciones que usen JWT en los procesos de autenticación y autorización, se recomienda incluir en su metodología de desarrollo de software, procesos bien establecidos para la validación de permisos, con la finalidad de prevenir vulnerabilidades de escalada de privilegios e impersonación del *token*.

BIBLIOGRAFÍA

Abril, L. (2021). Ecuador está entre los países con más ciberataques en América Latina - El Comercio. https://www.elcomercio.com/tendencias/tecnologia/ecuador-ciberataques-america-latina-hacker.html

Ahmed, S., & Mahmood, Q. (2019). An authentication based scheme for applications using JSON web token. *Proceedings - 22nd International Multitopic Conference, INMIC 2019*, 1–6. https://doi.org/10.1109/INMIC48123.2019.9022766

Alkhulaifi, A., & El-Alfy, E. S. M. (2020). Exploring Lattice-based Post-Quantum Signature for JWT Authentication: Review and Case Study. *IEEE Vehicular Technology Conference*, *2020-May*, 2–6. https://doi.org/10.1109/VTC2020-Spring48590.2020.9129505

Auth0. (n.d.-a). Token Best Practices. Retrieved November 26, 2021, from https://auth0.com/docs/best-practices/token-best-practices

Auth0. (n.d.-b). Token Storage. Retrieved November 26, 2021, from https://auth0.com/docs/security/data-security/token-storage

Baars, N., hblankenship, kingthorin, & OWASP Foundation. (2020). Password Plaintext Storage | OWASP. OWASP. https://owasp.org/www-community/vulnerabilities/Password_Plaintext_Storage

Datta, S. K., & Bonnet, C. (n.d.). Securing Iot. *2019 IEEE International Conference on Consumer Electronics (ICCE)*, 1–2.

Fett, D., Hosseyni, P., & Kusters, R. (2019). An extensive formal security analysis of the openid financial-grade API. *Proceedings - IEEE Symposium on Security and Privacy*, *2019-May*, 453–471. https://doi.org/10.1109/SP.2019.00067

Gaunt, M. (2020). Introducción a los service workers | Web Fundamentals. Web Fundamentals.

https://developers.google.com/web/fundamentals/primers/service-workers?hl=es

Google. (2020). Cómo utiliza Google las cookies. https://policies.google.com/technologies/cookies?hl=es#types-of-cookies

Hollerer, S., Fischer, C., Brenner, B., Papa, M., Schlund, S., Kastner, W., Fabini, J., & Zseby, T. (2020). Cobot attack: A security assessment exemplified by a specific collaborative robot. *Procedia Manufacturing*, *54*, 191–196. https://doi.org/10.1016/j.promfg.2021.07.029

Jones, M., Rescorla, E., & Hildebrand, J. (2012). JSON Web Encryption (JWE). https://tools.ietf.org/id/draft-ietf-jose-json-web-encryption-11.html#rfc.appendix.A.1.6

Mitre. (2019). CVE - Search Results. https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=jwt

Nachec, & Enesimus. (n.d.). Almacenamiento del lado cliente - Aprende sobre desarrollo web | MDN. Retrieved December 3, 2021, from https://developer.mozilla.org/es/docs/Learn/JavaScript/Client-side_web_APIs/Client-side_storage

OWASP. (2021). OWASP Top Ten Web Application Security Risks | OWASP. Owasp. https://owasp.org/www-project-top-ten/

Peguero, K., & Cheng, X. (2021). CSRF protection in JavaScript frameworks and the security of JavaScript applications. *High-Confidence Computing*, *1*(2), 100035. https://doi.org/10.1016/j.hcc.2021.100035

Peyrott, S. (2018). The JWT Handbook. *Auth0 Inc.* https://auth0.com/resources/ebooks/jwt-handbook

REST Security - OWASP Cheat Sheet Series. (n.d.). Retrieved November 26, 2021, from https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html

Rushdy, E., Khedr, W., & Salah, N. (2021). Framework to secure the OAuth 2.0 and JSON web token for rest API. *Journal of Theoretical and Applied Information Technology*, *99*(9), 2144–2161.

Salas, E. (2020). Aplicando seguridad a una API REST con JSON Web Tokens. http://openaccess.uoc.edu/webapps/o2/bitstream/10609/126696/1/emsalasgTFM2020.pdf

Villa, M. F. (2019). PROPUESTA DE UN MÉTODO DE MEJORES PRÁCTICAS PARA OPTIMIZAR EL NIVEL DE SEGURIDAD EN EL DESARROLLO DE SERVICIOS WEB RESTFul. http://dspace.espoch.edu.ec/handle/123456789/13028

Wike, R. (2021). Tipos de aplicaciones para la Plataforma de identidad de Microsoft | Microsoft Docs. https://docs.microsoft.com/es-es/azure/active-directory/develop/v2-app-types