Seguridad en APIs

Eduardo Fernández Casal
Máster Interuniversitario de Seguridad en las Tecnologías de la Información y de las comunicaciones (MISTIC)
Protocolos y aplicaciones de seguridad

Manel Mendoza Flores
Victor Garcia Font

Junio, 2019

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada 3.0 España de Creative Commons

FICHA DEL TRABAJO FINAL

Título del trabajo:	Seguridad en APIs
Nombre del autor:	*Eduardo Fernández Casal*
Nombre del consultor/a:	Manel Mendoza Flores
Nombre del PRA:	Victor Garcia Font
Fecha de entrega (mm/aaaa):	06/2019
Titulación:	Máster Interuniversitario de Seguridad en las Tecnologías de la Información y de las comunicaciones (MISTIC)
Área del Trabajo Final:	Protocolos y aplicaciones de seguridad
Idioma del trabajo:	Castellano
Palabras clave	*API, security testing, penetration testing*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

Desde mediados de la pasada década el desarrollo de APIs web se ha disparado y su presencia se ha multiplicado, convirtiéndose en imprescindibles. Lamentablemente, ese auge no ha ido acompañado de las mejores prácticas en términos de seguridad. En este documento pretendemos dar una visión global desde el prisma de la seguridad. Hemos recorrido la evolución histórica de las APIs web y analizado las causas de su auge y ubicuidad actuales. Hemos listado y detallado sus vulnerabilidades más frecuentes y los ataques más comunes de los que son víctima. Cuando ha sido posible hemos ofrecido ejemplos de casos reales. También hemos listado soluciones para mitigar o prevenir estas vulnerabilidades en forma de pruebas de seguridad.

Abstract (in English, 250 words or less):

Since the middle of the last decade the development of web APIs has skyrocketed and their presence has multiplied, becoming essential. Regrettably, this boom has not been accompanied by the best practices in terms of security. In this document we intend to give a global vision from a security perspective. We have traced the historical evolution of web APIs and analyzed the causes of their current boom and ubiquity. We have listed and detailed their most frequent vulnerabilities and the most common attacks of which they are victims. When possible, we have offered examples of real cases. We have also listed solutions to mitigate or prevent these vulnerabilities in the form of security testing.

Índice

1. Introducción **7**
1.1 Contexto y justificación del Trabajo 7
1.2 Objetivos del Trabajo 8
1.3 Enfoque y método seguido 9
1.4 Planificación del Trabajo 10
1.5 Riesgos preliminares 12
1.6 Breve sumario de productos obtenidos 13
1.7 Breve descripción de los otros capítulos de la memoria 13
2. APIs **15**
2.1 Definición 15
2.1.1 APIs web 16
2.2 Cronología 17
2.3 Situación actual y tendencias futuras 19
2.4 Tipos 21
2.4.1 Acceso 21
2.4.2 Tecnologías 21
2.4.2.1 RPC 22
2.4.2.1.1 XML-RPC 23
2.4.2.1.2 JSON-RPC 23
2.4.2.1.3 SOAP 24
2.4.2.2 REST 25
2.4.2.3 GraphQL 26
3. Riesgos y amenazas **28**
3.1 Ataques de denegación de servicio 29
3.1.1 Ataques en la capa de infraestructura 30
3.1.1.1 Ataques volumétricos 30
3.1.1.2 Ataques de protocolo 30
3.1.2 Ataques en la capa de aplicación 31
3.1.2.1 HTTP flood 31
3.1.2.2 Slowloris 31
3.1.2.3 Otros 31
3.2 Modificación de los datos de entrada 32
3.2.1 Payload 32
3.2.1.1 Inyecciones 32
3.2.1.2 Parameter pollution 34

3.2.1.3 SSRF 34
3.2.1.2 Ataques XXE 35
3.2.1.5 Deserialización 36
3.2.1.6 Recursos enumerados 36
3.2.2 Metadatos del protocolo 37
3.2.2.1 HTTP verb tampering 37
3.2.2.2 HTTP header injection 38
3.3 Ataques de intermediario 39
3.3.1 ARP y DNS Spoofing 39
3.3.2 Session hijacking 40
3.3.3 SSL/TLS hijacking 40
3.4 Suplantación 40

4. Estrategias de mitigación 43
4.1 Ataques de denegación de servicio 43
4.1.1 Red 43
4.1.2 Infraestructura 43
4.1.2 Aplicación 44
4.2 Modificación de los datos de entrada 44
4.2.1 Payload 44
4.2.1.1 Validación de los datos de entrada 44
4.2.1.2 Parseo de XML seguro 45
4.2.1.3 Deserialización segura 45
4.2.2 Metadatos del protocolo 45
4.3 Ataques de intermediario 46
4.3.1 ARP y DNS Spoofing 46
4.3.2 Session hijacking 46
4.3.3 SSL/TLS hijacking 47
4.4 Suplantación 47

5. Estrategias de prevención 50
5.1 Penetration testing 55
5.1.1 Tipos 56
5.1.2 Herramientas 57
5.2 Fuzz testing 59

6. Conclusiones 61

7. Glosario 62

8. Bibliografía 65

Lista de figuras

Ilustración 1: Crecimiento de las APIs desde 2005
Ilustración 2: Listado de tareas
Ilustración 3: Planificación temporal
Ilustración 4: Tecnologías empleadas en el desarrollo de APIs web
Ilustración 5: Ataques de denegación de servicio
Ilustración 6: Comportamiento ante parámetros repetidos
Ilustración 7: Ejemplo de cabeceras HTTP
Ilustración 8: Proceso OAuth
Ilustración 9: ¿Cuán importante es la seguridad de la APIs en tu organización?
Ilustración 10: ¿Cuánto esfuerzo dedicáis a día de hoy?
Ilustración 11: Low Orbit Ion Cannon
Ilustración 12: Fiddler
Ilustración 13: Wireshark
Ilustración 14: Llamada API en Postman
Ilustración 15: Prueba en Postman
Ilustración 16: Fases de penetration testing
Ilustración 17: OWASP ZAP
Ilustración 18: Pestaña de alertas en OWASP ZAP
Ilustración 19: Fuzzdb en OWASP ZAP

Tabla 1: Tecnologías empleadas en el desarrollo de APIs web
Tabla 2: Ejemplos de ataques

1. Introducción

1.1 Contexto y justificación del Trabajo

El concepto de API o *application programming interface* no es ninguna novedad en el mundo del desarrollo del software. Su presencia y explotación han sido indispensables para el desarrollo en todo tipo de plataformas. Como un ejemplo de estas tenemos la API de Windows, que pone a nuestra disposición funciones organizadas en bibliotecas para el desarrollo de aplicaciones ejecutables en las diferentes versiones de ese sistema operativo.

Con la llegada de los 4 grandes (Google, Amazon, Facebook, y Apple) también entraron en escena numerosas APIs web, a través de las cuales, principalmente los tres primeros, ofrecían sus servicios. Solo Google Maps llegó a ofrecer hasta 18 APIs diferentes. El gran atractivo que poseían muchas de ellas era su gratuidad, que provocó que fueran adoptadas de manera generalizada por la comunidad desarrolladora. Incluso hasta el punto de llegar a crearse una cierta dependencia. Como nota añadiré que Google Maps dejó de ser gratuita en 2011 y en 2018 experimentó una subida de precios de en torno al 1400%.

Polémicas aparte, durante este período se ponía de manifiesto de manera cada vez más clara un cambio en el enfoque del desarrollo del software tal y como se venía estructurando en las décadas anteriores. Se abandonan los enfoques monolíticos y se adaptan metodologías ágiles, desarrollos modulares y soluciones distribuidas. No perdamos de vista que vivimos en la cultura de la *start-up* y del *next big thing*. Estamos a la caza de la siguiente gran idea y es imprescindible optimizar recursos para llegar al mercado lo antes posible. ¿Por qué perder tiempo en desarrollar funcionalidad que no es crítica, cuando podría dedicar más tiempo a la funcionalidad *core* de mi aplicación? Aparece así un nicho de mercado. Esa funcionalidad no crítica, común en muchos desarrollos puede ser asumida por un tercero que se especializará en ella. Ofrecerá así un producto más robusto, más flexible y en definitiva mejor. Y lo hará a través de una API pública. Se democratiza y populariza así el rol del proveedor de APIs al existir todo un ecosistema de consumidores que lo pueden mantener.

Todo esto se ha convertido en la tormenta perfecta para una explosión de APIs, cuyo número se ha incrementado de manera dramática en los últimos años. Según datos del directorio de *ProgrammableWeb* se ha superado el número de 21000 APIs públicas en marzo de 2019. Una cifra que deja fuera todas las APIs privadas o de socio y que muestra un claro crecimiento si la comparamos con las 19000 alcanzadas en enero de 2018.

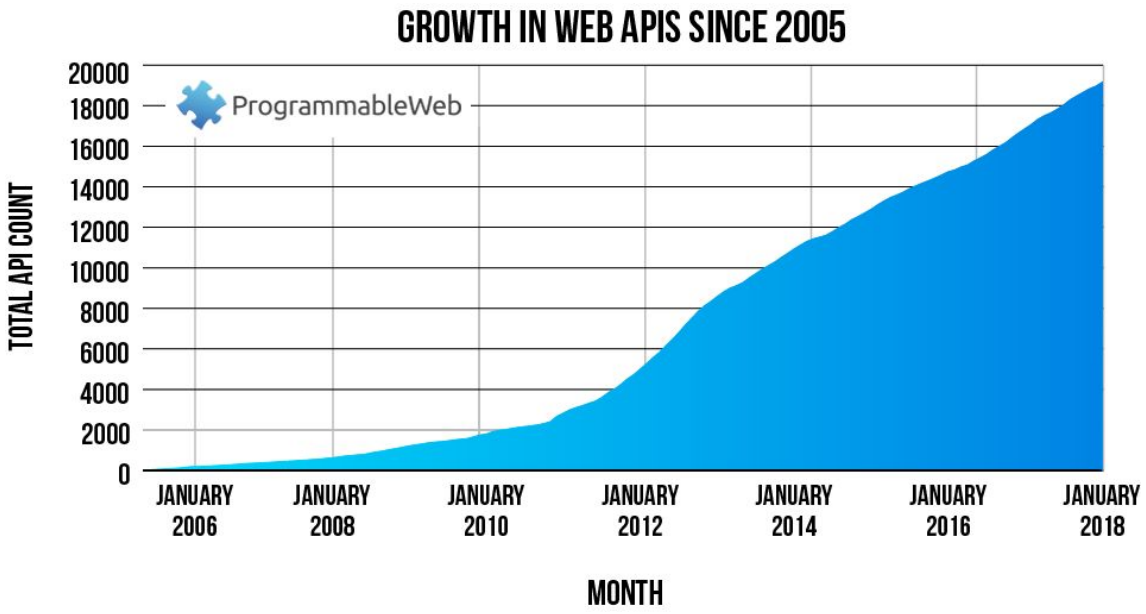

Ilustración 1: Crecimiento de las APIs desde 2005, extraído de [1]

Este trabajo se centra en el análisis de las APIs desde el punto de vista de la seguridad. En primer lugar definiremos de manera clara qué es una API y realizaremos una clasificación en función de la tecnología empleada en su desarrollo. Analizaremos los riesgos de seguridad más frecuentes que encontramos durante su desarrollo y ofreceremos medidas para mitigar y prevenir esas amenazas, definiendo así un compendio de buenas prácticas en el desarrollo seguro y testing de APIs.

1.2 Objetivos del Trabajo

El propósito de este trabajo es en el análisis de los mecanismos, controles y protecciones en arquitecturas basadas en APIs. Este objetivo general se apoya a su vez en numerosos objetivos más concretos:

- Entender el concepto de API Web.
- Conocer su situación y extensión actuales.
- Identificar las diferentes tipologías y tecnologías empleadas en su implementación.
- Acotar el concepto de amenazas de seguridad en el marco del desarrollo de APIs web.

- Identificar los riesgos y amenazas de seguridad más prevalentes en el desarrollo de APIs web.

- Ofrecer ejemplos reales de estos riesgos y amenazas y valorar su impacto.

- Definir estrategias de mitigación de estos riesgos y amenazas.

- Definir estrategias de prevención mediante técnicas de *testing* adaptadas al desarrollo de APIs web.

- Elaborar conclusiones finales y definir líneas de trabajo futuro en base a los hallazgos de este trabajo.

1.3 Enfoque y método seguido

Este trabajo final de máster es un trabajo de documentación e investigación. No se va a centrar en el desarrollo o adaptación de ningún producto. La estrategia seguida ha sido la división de la carga de trabajo en grandes bloques temáticos que se alinean con los capítulos que conformarán esta memoria. Estos a su vez se desglosan en tareas que se acometerán de manera secuencial. En su gran mayoría las tareas consistirán en un estudio exhaustivo de un tema en particular, ya sea la situación actual, estrategias o herramientas disponibles.

Bloque 1: Contextualización del trabajo. Definición de API y tipologías. Evolución histórica. Situación actual.

Bloque 2: Análisis de riesgos y vulnerabilidades de seguridad en las APIs. Categorización. Análisis del impacto. Casos reales.

Bloque 3: Mitigación de riesgos y vulnerabilidades de seguridad. Estrategias de prevención mediante *testing*.

Bloque 4: Conclusiones del trabajo actual. Líneas de trabajo futuro abiertas en este documento.

1.4 Planificación del Trabajo

A continuación se incluye un listado de las tareas a realizar:

Name	Begin date	End date
Trabajo final de master	2/20/19	6/21/19
Investigación previa	2/20/19	2/26/19
Plan de trabajo	2/27/19	3/4/19
Contexto y justificación	2/27/19	2/27/19
Objetivos	2/28/19	2/28/19
Enfoque y método seguido	3/1/19	3/1/19
Planificación	3/2/19	3/2/19
Breve sumario de producto...	3/3/19	3/3/19
Breve descripción de los ot...	3/4/19	3/4/19
Entrega 1	3/5/19	3/5/19
APIs	3/5/19	3/16/19
Definición y estado del arte	3/5/19	3/11/19
Tipologías	3/12/19	3/16/19
Riesgos y amenazas	3/17/19	4/11/19
Tipologías	3/17/19	4/1/19
Entrega 2	4/2/19	4/2/19
Ejemplos reales	4/2/19	4/11/19
Estrategias de mitigación	4/12/19	4/29/19
Tipologías	4/12/19	4/29/19
Entrega 3	4/30/19	4/30/19
Estrategias de prevención (te...	4/30/19	5/29/19
Security testing	4/30/19	5/9/19
Penetration testing	5/10/19	5/19/19
Fuzzy testing	5/20/19	5/29/19
Memoria final	5/30/19	6/3/19
Conclusiones	5/30/19	6/3/19
Entrega 4 - Memoria final	6/4/19	6/4/19
Presentación en vídeo	6/4/19	6/10/19
Elaboración del guión	6/4/19	6/7/19
Grabación del vídeo	6/8/19	6/10/19
Entrega 5 - Presentación e...	6/11/19	6/11/19
Defensa final	6/11/19	6/21/19
Preparación de la defensa	6/11/19	6/16/19
Defensa	6/17/19	6/21/19

Ilustración 2: Listado de tareas

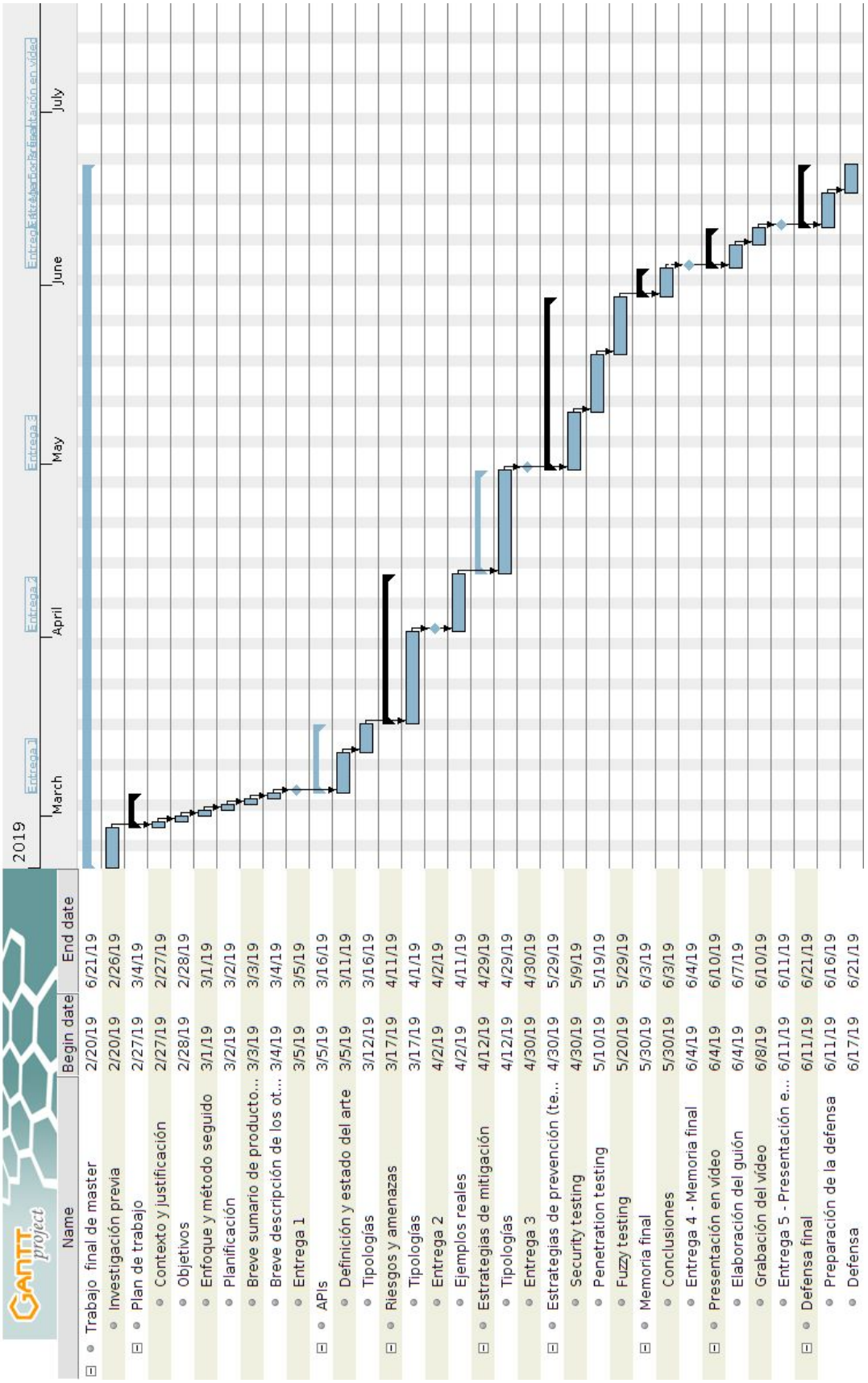

Ilustración 3: Planificación temporal

La planificación temporal establece 18 semanas de trabajo. Por cada semana de trabajo se espera una dedicación media de 14 horas, divididas entre 4 horas a lo largo de la semana laboral y 5 horas cada día no laborable. Esto hace un total de 252 horas. Esta estimación concuerda con los 9 créditos ETCS de la asignatura. Recordemos que en el sistema europeo de transferencia de créditos se espera una dedicación entre 25 y 30 horas por crédito. Esto prevé un esfuerzo en total de entre 225 y 270 horas.

Al tratarse de un proyecto de investigación no será necesaria otra inversión que la del tiempo dedicado. No se requiere una inversión en infraestructura o el pago de ningún tipo de servicio o licencia. Por esta razón, el coste total del proyecto dependerá en exclusiva del coste por hora que se acuerde. En caso de fijar 30 €/h estaríamos ante un monto de 7560 €.

1.5 Riesgos preliminares

A pesar de la labor de planificación inicial, es posible que durante la realización de este trabajo puedan darse situaciones que impacten negativamente a su desarrollo. Estos pueden producir retrasos en la planificación temporal mostrada en el punto anterior, hasta el punto de poner en el peligro la consecución del proyecto completo.

Riesgo 1: Proyecto demasiado ambicioso. El alcance del proyecto es demasiado amplio para la limitación temporal fijada por la asignatura (9 créditos ECTS). Para paliar este riesgo podemos priorizar las tareas que consideremos más importantes en detrimento de aquellas menos relevantes o anecdóticas. También es posible descartar aquellas que no aporten valor. Como medida más radical podemos optar por modificar el alcance del proyecto.

Riesgo 2: Ausencia de información. Por la naturaleza de documentación e investigación de este trabajo, el grueso de las tareas se basa en la recopilación de información. Por ello es posible que para alguno de los temas que se traten no sea imposible obtener datos que aporten valor a la memoria. Como acciones mitigadoras podemos ampliar las fuentes a manejar para que además de incluir sitios web y libros, consultemos también foros, manuales de herramientas o publicaciones científicas.

1.6 Breve sumario de productos obtenidos

Por la propia organización de la asignatura, este trabajo final de máster se articula en 5 entregas que se corresponden con 4 documentos incrementales de la memoria y una presentación en vídeo:

- **Entrega 1**: consiste en realizar un plan de trabajo. El plan de trabajo es un elemento clave del funcionamiento del TFM ya que describe el problema que se pretende resolver, el trabajo concreto que se llevará a cabo y la descomposición de este trabajo en tareas y metas temporales. De esta manera, se espera que un plan de trabajo contenga.

- **Entrega 2**: entrega parcial del trabajo hasta esa fecha. De acuerdo a la planificación temporal incluirá el capítulo 2 completo y el capítulo 3 de manera parcial.

- **Entrega 3**: entrega parcial del trabajo hasta esa fecha. De acuerdo a la planificación temporal incluirá los capítulos 2, 3 y 4 íntegramente.

- **Entrega 4**: memoria final del trabajo de fin de máster. Agrupa el trabajo presentado en las 3 entregas anteriores además del capítulo 5, conclusiones, glosario, bibliografía y anexos.

- **Entrega 5**: un vídeo de 15 minutos como máximo en el que se presenta una síntesis del trabajo realizado sobre una presentación de diapositivas.

1.7 Breve descripción de los otros capítulos de la memoria

Dada la naturaleza de documentación e investigación de este trabajo, esta memoria y la posterior presentación en vídeo serán los únicos entregables. Por esta razón, todas las tareas que se acometerán tendrán como fin último la elaboración de alguno de los capítulos de este documento. Hay por tanto una relación directa entre los objetivos definidos previamente y el resto de capítulos de esta memoria.

Capítulo 2: APIs. Se define el concepto de API Web. Se recorre su evolución histórica y analiza su situación actual. Se describen sus posibles categorizaciones en función de su uso y de las tecnologías empleadas en su implementación.

Capítulo 3: Riesgos y amenazas. Se acota el concepto de amenaza de seguridad en el marco del desarrollo de APIs web y se identifican los más frecuentes. Se listan casos reales y se valora su impacto.

Capítulo 4: Estrategias de mitigación. Se establecen métodos y buenas prácticas en el desarrollo de APIs web para evitar los riesgos y amenazas vistos anteriormente.

Capítulo 5: Estrategias de prevención (testing). Se definen estrategias de prevención mediante técnicas de testing adaptadas al desarrollo de APIs web como pueden ser *penetration testing* o *fuzzy testing*.

Capítulo 6: Conclusiones y trabajo futuro. Se elaboran las conclusiones finales y se definen líneas de trabajo futuro en base a los hallazgos de este trabajo.

2. APIs

2.1 Definición

API es el acrónimo de *application programming interface* o interfaz de programación de aplicaciones. En términos genéricos establece un contrato entre dos partes en el que una se compromete a proveer a la otra de un servicio determinado, siguiendo unas reglas de comunicación concretas. Esas dos partes pueden pertenecer a la misma aplicación o tratarse de dos aplicaciones diferentes. Dentro de esta definición tan amplia y un tanto vaga caben elementos muy variados, dependiendo de la naturaleza de los componentes involucrados, el servicio prestado y el método de comunicación.

Como ejemplos de APIs tenemos:

- *BIOS interrupt calls*. Las llamadas de interrupción del BIOS permiten acceder a las funcionalidades más básicas de una máquina de arquitectura x86. Son usadas por cargadores de arranque y programas de DOS. Los sistemas operativos también hacían uso de ellas para llevar a cabo operaciones de entrada/salida, aunque ahora usan rutinas propias. Se invocan usando la instrucción `INT`.

- Windows API. La interfaz de programación de aplicaciones de Windows es un conjunto de bibliotecas de enlace dinámico o DLL. Estas contienen rutinas que permiten a los programas de Windows acceder a funcionalidades del sistema operativo tales como el sistemas de archivos, dispositivos, procesos.

- Direct3D. Esta API forma parte de DirectX. DirectX engloba un conjunto de APIs orientadas hacia multimedia en plataformas Microsoft. Direct3D en concreto se emplea en el renderizado de gráficos tridimensionales con aceleración por hardware.

- OpenGL. Esta es una API multilenguaje y multiplataforma de generación de gráficos 2D y 3D. Tiene un cometido similar a Direct3D y son de hecho competidoras directas en el mercado.

- DOM. Acrónimo de *Document Object Model* o Modelo de Objetos del Documento. Esta API representa documentos HTML, XHTML o XML como árboles y permite acceder a los diferentes elementos de manera programática, para su consulta y modificación.

Como podemos comprobar, existen multitud de escenarios. En ciertos casos la opción es única. En otros casos, la variedad es tal que incluso ante un mismo problema podemos encontrar diferentes alternativas. No obstante, como norma general podemos decir que simplifican la visión general al añadir un nivel de

abstracción que oculta la complejidad subyacente. Por esta misma razón, el empleo de APIs redunda en un aumento de la velocidad del desarrollo. Los programadores pueden emplear funciones y herramientas listas para ser usadas y evitar implementar de cero.

2.1.1 APIs web

Aunque no existe consenso global sobre la definición exacta, podemos decir que las APIs web son un subconjunto de APIs en las que la comunicación se lleva a cabo a través de Internet, haciendo uso de protocolos web como puede ser HTTP. En este “contrato” del que hablábamos anteriormente, una API web define una serie de endpoints o URLs de acceso, la forma de las peticiones y la estructura de las respuestas.

En general, servicio web y API web se usan como sinónimos, al entender el servicio web como una API envuelta en HTTP. No obstante, en la actualidad tiende a asociarse el término de manera más concreta con las APIs REST, que veremos más adelante.

Un ejemplo de este tipo de APIs es la Graph API de Facebook. Permite a las aplicaciones leer y escribir en el grafo social de Facebook, donde se incluye personas, fotos, ventos y páginas ligados mediante relaciones de amistad y etiquetados. Facebook expone su “contrato” en su documentación oficial: https://developers.facebook.com/docs/graph-api. Ahí podemos consultar las llamadas necesarias para obtener una información en concreto y el tipo de respuesta que generan. Por ejemplo, para obtener el logo de Facebook tenemos que acceder a la siguiente URL:

https://graph.facebook.com/facebook/picture?redirect=false

La respuesta es un JSON con esta forma:


```
{
   "data": {
      "height": 50,
      "is_silhouette": false,
      "url":
"https://scontent.xx.fbcdn.net/v/t1.0-1/p50x50/31562081_1015732468336672
9_5005221974700457984_n.png?_nc_cat=1&_nc_ht=scontent.xx&oh=38a74c83e23b
dc78a44395213a3f354c&oe=5D0B3855",
      "width": 50
   }
}
```


Con esta información un desarrollador puede conectarse al endpoint https://graph.facebook.com, realizar llamadas correctas y extraer su resultado de una estructura acordada.

2.2 Cronología

Este listado de fechas no pretende ser un compendio exhaustivo, sino que pretende aportar una visión general de la evolución de las APIs web en su contexto histórico.

1981. Bruce Jay Nelson acuña el término *Remote Procedure Call*.

1989. Tim Berners-Lee inventa la *World Wide Web*.

Agosto de 1995. Publicado el protocolo RPC de manera oficial como RFC 1831.

1998. Dave Winer, Don Box, Bob Atkinson y Mohsen Al-Ghosein diseñan XML-RPC para Microsoft, a partir del cual evoluciona SOAP.

2000. Roy Thomas Fielding publica su tesis doctoral *Architectural Styles and the Design of Network-based Software Architectures* en la que sienta las bases de la arquitectura REST.

7 de febrero de 2000. Salesforce lanza su API en la conferencia IDG Demo 2000. Salesforce es una empresa S&P 500 conocida por su CRM Sales Cloud.

20 de noviembre de 2000. eBay lanza su API, junto con su programa de desarrollo de manera limitada.

16 de julio de 2002. Amazon lanza *Amazon.com Web Services* o AWS. Inicialmente solo permite buscar y mostrar productos de Amazon.com en formato XML.

Agosto de 2004. Flickr lanza su API REST. La posibilidad de embeber imágenes, aúpan a Flickr al primer puesto de las plataformas visuales del momento.

2005. Publicada la especificación de JSON-RPC 1.0.

Marzo de 2006. La computación en la nube inicia su andadura de la mano de Amazon cuando S3 comienza funcionar. Este servicio web permite el almacenado de información. Inicialmente consistía únicamente en una API REST, sin ningún tipo de interfaz de usuario.

Agosto de 2006. Meses después de ofrecer S3, Amazon continúa en la misma línea con Amazon EC2. En este caso ofrece la posibilidad de contratar servidores virtuales dentro de sus propios centros de datos. De la misma manera que hizo con S3, Amazon tardará más de 3 años en ofrece una interfaz de usuario.

Queda patente en esta línea de tiempo el espaldarazo que supuso para las APIs web los lanzamientos por parte de Amazon de S3 y EC2. Estamos hablando de la introducción de herramientas que permitían el despliegue de aplicaciones mediante el uso de APIs. Hasta entonces Amazon solo permitía la inclusión de contenido de su tienda en otros sitios web. Flickr hacía lo propio con imágenes en redes sociales. Con S3 y EC2 se abre la posibilidad de gestionar negocios a través de APIs web. Dejan de estar relegadas a un uso casual y anecdótico para pasar a comenzar a ganar relevancia en el mundo del desarrollo.

15 de agosto de 2006. Facebook publica su plataforma web y su API. Su API REST permite a desarrolladores comenzar a desarrollar aplicaciones y juegos casi de manera inmediata, convirtiéndose en una gran ventaja sobre su gran competidor MySpace.

29 de junio de 2006. Google revela la API de Google Maps. Con ella se inicia un estallido de mashups, que ya venían ocurriendo aun antes de la publicación de la propia API. En la actualidad existen alrededor de 2000

20 de septiembre de 2006. Twitter hace pública su API REST. Inicialmente con autenticación básica, este API nace como respuesta a numerosas APIs "piratas" y a la práctica de *screen scraping*.

29 de junio de 2007. Apple pone a la venta el iPhone de primera generación, arquetipo del teléfono móvil tal y como lo conocemos hoy. Comienza el auge de los teléfonos inteligentes.

10 de julio de 2008. Apple abre la *App Store*. Inicialmente solo estaban disponibles 500 aplicaciones.

22 de octubre de 2008. Google abre *Android Market*.

Mayo de 2009. Publicada la versión 2 del protocolo RPC como RFC 5531. Ese mismo año Sun modifica la licencia y pasa a ser BSD modificada, permitiendo su uso libre.

Noviembre de 2009. Después de aprovechar el festival SXSW para presentarse en sociedad, Foursquare presenta su API después de conseguir financiación.

Febrero de 2011. Instagram lanza su API oficial, forzados por el hecho de que meses antes Mislav Marohnić lanzara su propia versión pirata.

Con el advenimiento de los teléfonos inteligentes las APIs web ganaron otro escenario en el que crecer en el desarrollo de aplicaciones móviles.

2015. Facebook hace público GraphQL, después de 3 años de desarrollo.

2.3 Situación actual y tendencias futuras

De acuerdo a los datos recopilados por ProgrammableWeb en los seis primeros años de nuestra década el número de APIs web se ha incrementó en un 758%. La media anual de nuevas APIs superaba las 2000, con más de 170 nuevas APIs añadidas de manera mensual [1].

Si nos centramos en datos más actualizados, tal y como apuntamos en la introducción, en enero de 2018 se alcanzaron las 19000. En el momento de redacción de esta memoria, en marzo de 2019, ya se han alcanzado la cifra de 21000. Podemos decir así que se mantiene la tendencia de crecimiento, si bien no se ha producido una explosión exponencial, como se llegaba a aventurar de manera un tanto especulativa al comienzo de la década. No perdamos de vista en cualquier caso que estas cifras siempre hacen referencia a APIs web públicas. Las privadas y de socio permanecen ocultas, ya que no son dadas de alta en ningún listado oficial y por lo tanto es muy difícil hablar de cantidades globales.

Si nos centramos en las tecnologías empleadas, ProgrammableWeb [6] nos ofrece un desglose de aquellas registradas en su sitio web. Observamos cómo REST ha tomado la delantera y es la arquitectura de moda en el desarrollo. El hecho de que Facebook, Twitter y Google entre otros hagan uso de ella puede explicar su popularidad.

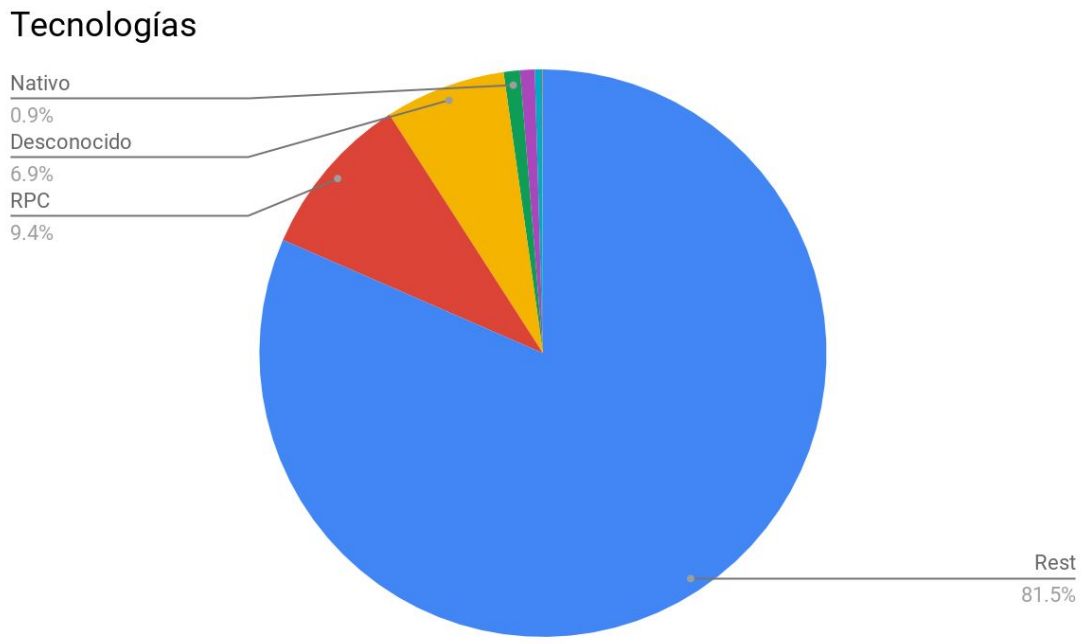

Ilustración 4: Tecnologías empleadas en el desarrollo de APIs web

Por otro lado, después del empuje para el desarrollo de las APIs que supusieron sucesivamente el *e-commerce*, las redes sociales, la computación en la nube y la ubicuidad de los dispositivos móviles, parece que el siguiente sector que producirá otro *boom* será el de la banca.

Tradicionalmente, la relación entre un banco y sus clientes ha sido siempre de 1 a 1. No obstante, con la introducción a nivel europeo de la segunda Directiva de Servicios de Pago o PSD2 este paradigma cambiará. El objetivo principal de la directiva es unificar el mercado europeo y ofrecer más seguridad a los consumidores. Además, los bancos dejarán de tener el monopolio del servicio de pagos. Aparece así la figura de los *Third Party Payment providers* o TPPs. Después de haber sido convenientemente autorizados, se les permitirá a estos el acceso a las cuentas de los clientes de un banco y la posibilidad de hacer pagos en su nombre. Para el consumidor no habrá perjuicio alguno porque estos TTPs estarán sujetos a las mismas reglas y supervisión que los bancos tradicionales. Será de hecho el gran beneficiado en las transacciones online. Ya no será necesario recurrir a intermediarios como las compañías de las tarjetas para finalmente realizar el cargo en cuenta. El consumidor podrá autorizar a un comercio para que realice cargos en su nombre. Los bancos estarán obligados a servir los datos de sus clientes de algún modo a estos TPPs para que puedan ofrecer sus servicios. El consenso apunta a que el modo elegido serán APIs web abiertas.

El gran reto para los bancos es superar la complejidad que suponen interconectar su infraestructura ya existente, antes de siquiera comenzar a plantearse la implementación de una API abierta. La empresas de Tecnología Financiera no adolecen de este obstáculo, por lo que serán capaces de desarrollar de manera mucho más rápida y eficiente.

El segundo problema que deberán afrontar en la falta de estándares. La PSD2 fija únicamente la obligación de poder compartir datos con terceros. En ningún caso se hace una mención explícita a APIs. No fuerza en ningún caso el uso de ninguna tecnología en concreto. En un principio esto puede dar una libertad muy anhelada a los bancos a la hora de diseñar e implementar sus propias APIs. A larga contribuye a la fragmentación y limita la cooperatividad, lo que puede acabar malogrando en definitiva el fin el último de la PSD2.

Por último, no perdamos de vista que el principal objetivo de la directiva es ofrecer más seguridad a los consumidores. La seguridad de las APIs desarrolladas será primordial. Históricamente los bancos han estado siempre al cargo de su propia seguridad. A partir de ahora será necesaria una colaboración más cercana entre proveedor y consumidor de la API para garantizar la protección del cliente. [7] [62]

2.4 Tipos

Existen numerosas maneras de categorizar APIs web. De hecho, el apelativo web ya es una especificación dentro del término más genérico que es API, como ya hemos visto anteriormente. En este punto solo vamos a hacer referencia a dos clasificaciones, ya que son las únicas que manejaremos a lo largo del documento.

2.4.1 Acceso

En función de las restricciones de acceso a la API tenemos tres tipos:

- **APIs públicas**. No están sujetas a ningún tipo de restricción, son de acceso libre. Esto no quiere decir que sean gratuitas y mucho menos, que no sean seguras. En GitHub[8] o RapidAPI[9] entre otros podemos encontrar un listado con las más relevantes.

- ***Partner APIs* o APIs de socio**. No están disponibles de manera libre. Es necesario algún tipo de privilegio para hacer uso de ellas.

- **APIs privadas o internas**. No son públicas y su consumo es gratuito. Generalmente su uso se limita a la organización en la que se se han creado, entre diferentes equipos con el fin de optimizar el uso de los recursos. Deberían ser objeto de las mismas exigencias a nivel de seguridad y documentación que el resto de APIs. Lamentablemente, esto no es siempre así. De hecho, muchas de ellas se aprovechan del hecho de no ser públicas para implementar políticas de seguridad muy pobres, haciendo valer el definitiva el principio de seguridad por oscuridad. En muchos casos el único requisito es acceder desde la propia red interna o usar una clave genérica que todo el mundo comparte.

2.4.2 Tecnologías

El desarrollo de APIs web puede llegar a ser extremadamente complejo y en su consecución es posible recurrir a protocolos, arquitecturas, paradigmas de programación, etc. En este epígrafe pretendemos recopilar aquellas más reseñables, ya sea por su difusión, por su relevancia histórica o por su potencial. Dado que trataremos tecnologías muy variadas, es importante recalcar que esta categorización sería excluyente. Por ejemplo, RPC es un protocolo, mientras que REST es una arquitectura, por lo que una API web podría hacer uso de ambos a la vez.

	XML-RPC	JSON-RPC	SOAP	REST	GraphQL
Tipo	Protocolo	Protocolo	Protocolo	Arquitectura	Lenguaje de consultas
Formato de datos	XML	JSON	XML	XML JSON HTML	JSON
Protocolo de comunicación	HTTP	HTTP	TCP HTTP STMP	HTTP	HTTP
Tipado de datos	Débil	Débil	Fuerte	Débil	Fuerte
Definición de datos	Servidor	Servidor	Servidor	Servidor	Cliente
Endpoints	Único	Único	Único	Múltiples	Único
Caché	No	No	No	Sí	No
Versionado	Sí	Sí	Sí	Sí	No

Tabla 1: Tecnologías empleadas en el desarrollo de APIs web

2.4.2.1 RPC

RPC (*Remote Procedure Call*) o llamada a procedimiento remoto permite la comunicación entre aplicaciones usando un formato estándar comprendido tanto por emisor como por receptor. RPC emplea el modelo cliente- servidor. El cliente es el programa que solicita el servicio, mientras que el programa que lo provee es el servidor. El servidor expone una serie de operaciones que son invocadas por el cliente adjuntando los parámetros como parte de la URL o en el cuerpo de la petición. La comunicación se resume en un intercambio de información de esta forma. El cliente es el que está al cargo de realizar las llamadas correspondientes con los parámetros correctos para lograr un fin determinado.

Las implementaciones de RPC más empleadas en el desarrollo de APIs web han sido XML-RPC, JSON-RPC y SOAP.

2.4.2.1.1 XML-RPC

XML-RPC es una implementación de RPC que usa XML como formato de datos y HTTP como protocolo de comunicación. Permite representar 7 tipos de datos primitivos (binarios, booleanos, fechas, decimales, enteros, cadenas de texto y nulos) y 2 tipos estructurados (structs y arrays). Este protocolo se encuentra limitado en caso de necesitar manejar tipos de datos complejos.

Un ejemplo de un par petición-respuesta es el siguiente:

Petición

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<methodCall>
   <methodName>subtract</methodName>
   <params>
      <param>
         <value><int>42</int></value>
      </param>

      <param>
         <value><int>23</int></value>
      </param>
   </params>
</methodCall>
```

Respuesta

```
<methodResponse>
   <params>
      <param>
         <value><int>19</int></value>
      </param>
   </params>
</methodResponse>
```

XML-RPC es una implementación muy ligera de RPC. Es muy sencillo de desarrollar y depurar. Su sencillez es también su mayor defecto y pueden convertirlo en una mala opción para desarrollos complejos.

2.4.2.1.2 JSON-RPC

JSON-RPC es otra implementación de RPC muy similar a XML-RPC, pero usa JSON como formato de datos y no fija un protocolo de comunicación. JSON puede representar 4 tipos de datos primitivos (cadenas de texto, números, booleanos y nulos) y 2 tipos estructurados (objetos y arrays). Presenta el mismo problema que existe con XML-RPC. Además, introduce cierta

ambigüedad. Puede ser difícil diferenciar entre enteros y decimales, por ejemplo.

Un ejemplo de un par petición-respuesta es el siguiente:

Petición

```
{"jsonrpc": "2.0", "method": "subtract", "params": [42, 23], "id": 1}
```

Respuesta

```
{"jsonrpc": "2.0", "result": 19, "id": 1}
```

Tal y como sucede con XML-RPC, su uso no está muy extendido.

2.4.2.1.3 SOAP

SOAP es el acrónimo de Simple Object Access Protocol es un protocolo que define el intercambio de información estructurada. Puede ser considerado como un sucesor de XML-RPC y como este, utiliza XML exclusivamente como formato de los datos intercambiados. Para evitar las limitaciones que introducía XML-RPC, SOAP introduce el uso de WSDL o *Web Services Description Language*. Se trata de un formato para describir el funcionamiento del servicio, sus métodos, los parámetros y los valores devueltos para cada uno de ellos. En caso de que alguno de estos manejase algún tipo de datos especial, la información iría embebida dentro del archivo WSDL en forma de XML Schema.

SOAP define una estructura fija para cada mensaje intercambiado:

```
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

      <soap:Header>
            …
      </soap:Header>

      <soap:Body>
            ...
            <soap:Fault>
                  ...
            </soap:Fault>
      </soap:Body>

</soap:Envelope>
```

- <Envelope> Es el elemento raíz de todo mensaje SOAP.
    - <Header> Incluye información sobre la aplicación y es opcional.
    - <Body> Contiene el cuerpo del mensaje, la información relevante para el consumidor. Obviamente es obligatorio.
        - <Fault> Incluye información sobre los posibles errores encontrados durante el procesamiento del mensaje.

Esta estructura fija junto con el tipado de datos fuerte que emplea resulta en un exceso de verbosidad de los mensajes en ambos sentidos. Si el volumen de peticiones y respuestas es muy alto, la carga en la red puede acabar impactando negativamente el rendimiento de la aplicación, especialmente en el caso de aplicaciones móviles. Otro de los grandes inconvenientes es la ausencia de mecanismos de caché.

Como aspectos positivos de SOAP podemos destacar flexibilidad en la comunicación, pudiendo trabajar con HTTP, pero también SMTP y TCP. Así como la seguridad, con la extensión WS-Security que permite garantizar la integridad, confidencialidad y no repudio de los mensajes.

Desde el punto de vista del desarrollo, SOAP requiere un mayor acoplamiento con el servidor, lo que complica el desarrollo de nuevas funcionalidades o la modificación de las ya existentes. Su desarrollo es menos ágil, lo que ha contribuido a su pérdida de popularidad en favor de REST.

2.4.2.2 REST

REST o *Representational State Transfer* es un estilo de arquitectura sin estado en el que los recursos son identificados mediante una URI. El enfoque es totalmente diferente a la hora de obtener el dato. En REST accedemos al recurso vía URI. En RPC en general y SOAP en particular realizamos una operación. Es el recurso, el dato al fin y al cabo, el que vertebra las APIs REST y no la operación. Las operaciones son un conjunto estándar limitado, los verbos HTTP. Los más comunes son:

- GET: obtener un recurso en particular mediante id o un listado de ellos.
- POST: crear un nuevo recurso.
- PUT: actualizar un recurso ya existente.

- PATCH: actualizar parcialmente un recurso.
- DELETE: eliminar un recurso ya existente.

Otra de las diferencias más llamativas con SOAP es que el formato de los datos ya no es fijo. La estructura es variable y puede soportar XML, HTML, JSON e incluso texto plano. JSON es de hecho el formato más empleado. La ausencia de una estructura fija hace que los mensajes sean por norma general mucho menos pesados, lo cual es toda una ventaja en aplicaciones móviles. El rendimiento también mejora con la caché de HTTP.

Los únicos protocolos de comunicación soportados son HTTP y HTTPS. Además, no hay tipado fuerte de datos, lo que puede resultar en problemas de comunicación entre emisor y receptor.

En general su interfaz uniforme y el desacoplamiento entre cliente y servidor ha permitido su alzamiento como el estándar de facto por su portabilidad, escalabilidad y rapidez de desarrollo.

2.4.2.3 GraphQL

GraphQL es un lenguaje de consulta y modificación de datos. Conceptualmente trabaja en la misma línea que REST, manejando recursos que representan la información que deseamos obtener. Intenta solventar un problema del que adolecen tanto las APIs SOAP como las REST, la definición del dato se hace en el servidor. Por esta razón es frecuente una petición devuelva más información de la necesaria (*overfetching*) o no la suficiente, de manera que sea obligatorio realizar sucesivas llamadas (*underfetching*). Cuando se realiza una petición en GraphQL, el cliente especifica exactamente la información que desea. Incluso es posible obtener datos de diferentes bases de datos en la misma llamada. Esta flexibilidad repercute negativamente en la posibilidad de cachear los datos y no lo soporta de manera nativa.

Una consulta muy sencilla puede ser una que pida los nombre de todos los clientes:

```
{
  allClients {
    clients {
      surname
      name
    }
  }
}
```

La respuesta contendrá el nombre nada más, sin más información extra:

```
{
  "data": {
    "allClients": {
      "clients": [
        {
          "surname": "A New Hope",
          "name": "George Lucas"
        },
        {
          "surname": "The Empire Strikes Back",
          "name": "Irvin Kershner"
        }
    }
}
```

Es una tecnología muy prometedora, pero muy reciente y aún no ha sido adoptada de manera global.

3. Riesgos y amenazas

Con el auténtico auge que están viviendo las APIs web en la actualidad se ha producido una explosión en su desarrollo, pero quizá se ha perdido de vista sus peculiaridades que las distinguen de un desarrollo web al uso. Tampoco hay que olvidar que una API pública puede exponer a potenciales atacantes pistas sobre las estructuras subyacentes en el mejor de los casos, y acceso directo a ellas en el peor.

A continuación realizaremos una categorización de los riesgos y amenazas que sufren las APIs web en función de la conducta elegida por el atacante o o de su propósito. En la mayoría de los casos, estas vulnerabilidades están íntimamente ligadas a las tecnologías elegidas para su implementación. En otros casos se deben a código desarrollado *ad-hoc*. No se trata de una división estricta, ya que como veremos existen múltiples ataques que podrían asignarse a varias categorías.

Las intenciones de los atacantes pueden ser la denegación del servicio, el acceso a información privilegiada o su modificación. El vector empleado para lograrlo puede ser los datos de entrada, la identidad del usuario o intermediarios. En este documento nos centraremos principalmente en la segunda categorización, por la heterogeneidad que presentan las primeras categorizaciones. No obstante, empezaremos por los ataques de denegación de servicio por su impacto.

	Datos de entrada	Intermediario	Suplantación
Denegación de servicio	Ataques XEE Sloris		
Obtención de información	Inyección	*Session hijacking*	Captura de *API key*
Modificación de información	Inyección *Parameter pollution*	*Spoofing*	Captura de *API key*

Tabla 2: Ejemplos de ataques

3.1 Ataques de denegación de servicio

Su objetivo es inutilizar la API web de tal manera que sus usuarios legítimos no pueden hacer uso de ella. Evidentemente, este tipo de ataques no son únicos de las APIs web, sino que pueden afectar a cualquier servicio online. Los ataques más sencillos trabajan con un solo dispositivo atacante. En ocasiones, sin embargo, el ataque se realiza desde varias máquinas simultáneamente. Esto los convierte en ataques distribuidos de denegación de servicio, mucho más difíciles de mitigar.

Con frecuencia el usuario malicioso hace uso de *booter services*, que ponen a su disposición numerosas máquinas atacantes que permiten alcanzar un número crítico de peticiones de manera mucho más rápida. WebStresser.org, uno de los *booter services* más populares fue desmantelado en abril de 2018 en una operación conjunta entre EE.UU., Reino Unido y Países Bajos. Había llegado a tener más de 150000 usuarios registrados y permitía hacer uso de sus servicios por menos de 15 dólares. Se les considera responsables de más de 4 millones de ataques en un espacio de 3 años. Lo más relevante es que permitía a usuarios poco avezados lanzar ataques de denegación de servicio a discreción. [14]

Uno de los ataques más recientes de este tipo fue el que sufrió la Comisión Federal de Comunicaciones de EE.UU. Esta agencia es la encargada de la regulación de las telecomunicaciones interestatales e internacionales de todo el país. El incidente tuvo lugar en mayo de 2017, mientras la FCC se encontraba en una polémica encendida por la decisión de su presidente, Ajit Pai, de apoyar la derogación de las medidas introducidas durante el mandato de Obama para salvaguardar la neutralidad de la red. La agencia mantiene que fue víctima de un ataque distribuido de denegación de servicio cuyo objetivo fue la API de su sitio web. Como consecuencia, usuarios legítimos fueron incapaces de realizar comentarios en la página web de la FCC. El ataque suscitó sus dudas desde el primer momento, debido a la reticencia de la agencia a compartir documentación alguna sobre este. Meses después, una investigación del propio inspector de la FCC desmentía la primera versión y concluía que la caída del servicio se había debido a un aumento desmesurado del tráfico. Ese aumento habría sido causado por la emisión de un reportaje crítico con la política de la FCC sobre la neutralidad de la red durante el programa del cómico John Olivier. [15][16]

Los ataques de denegación de servicio se pueden dividir en función de la capa que explotan. Según los datos de Akamai, la mayor parte se centran en infraestructura. [26]

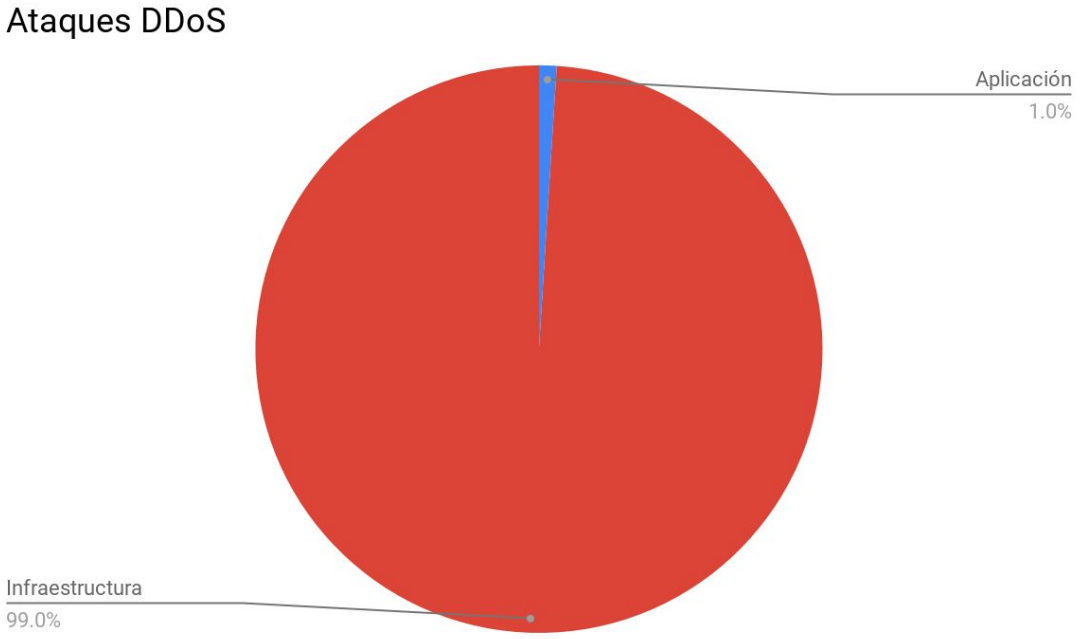

Ilustración 5: Ataques de denegación de servicio

3.1.1 Ataques en la capa de infraestructura

3.1.1.1 Ataques volumétricos

Buscan saturar el ancho de banda de la máquina atacada y son medidos en bits por segundo (Bps). Entre los más comunes se encuentran los ataques UDP flood y ping flood que saturan la máquina con paquetes UDP e ICMP respectivamente.

3.1.1.2 Ataques de protocolo

Los ataques de protocolo se miden en paquetes por segundo (Pps) y van más allá, ya que su objetivo es consumir los recursos de la máquina víctima y otros elementos intermedios como pueden ser switches. Ejemplos de este tipo de ataques son los ataques pitufo, los pings de las muerte, ataques con paquetes fragmentados o SYN floods.

3.1.2 Ataques en la capa de aplicación

Hacen uso de lo que en apariencia son peticiones correctas. Se trata de ataques en general sofisticados que explotan vulnerabilidades en alguno de los sistemas que soportan la API, sea el sistema operativo, el servidor web o el servidor de base de datos. Su objetivo último es derribar el servicio que sustenta a la API web y se miden en peticiones por segundo (Rps). De acuerdo al informe de Impervia del año 2017 [27] estos ataques están creciendo en frecuencia. En concreto un 43% con respecto al último trimestre del año anterior. Además, de ellos un 63,3% fue víctima de este tipo de ataque repetidas veces.

Existen numerosos ataques de este tipo, debido a que potencialmente disponen de una gran superficie de ataque.

3.1.2.1 HTTP flood

Un usuario malicioso puede pervertir el uso normal de una conexión legítima a una API web sin necesidad de modificar su contenido. Solo por el mero hecho de llevar a cabo una petición de manera repetida y continuada puede inundar a los servidores encargados de tal manera que se vean sobrecargados y dejen de funcionar de manera correcta. Este tipo de ataques satura el sistema con peticiones HTTP válidas de manera masiva. Son especialmente efectivos cuando cada petición requiere una respuesta costosa en recursos para el sistema, como una búsqueda exhaustiva en base de datos.

3.1.2.2 Slowloris

Este ataque permite derribar un servidor mediante el envío de peticiones parciales. El envío continuado de cabeceras HTTP obliga al servidor a mantener todas las conexiones abiertas, llegando al punto en el que peticiones legítimas son rechazadas.

Entre los servidores afectados se encuentra Apache en sus versiones 1.x y 2.x y dhttp. IIS6.0, IIS7.0, lighttpd y nginx no son vulnerables. [32]

3.1.2.3 Otros

Existen muchos otros tipos de ataques que explotan alguna de las tecnologías empleadas en el desarrollo web. Por ejemplo, es posible explotar vulnerabilidades en una petición SOAP y producir un buffer overflow en la máquina [28]. También es posible aprovecharse la implementación de las

tablas hash en PHP [29]. En general este tipo de ataques requiere el envío de información maliciosa en los datos de entrada de las peticiones y los veremos más a fondo a continuación.

3.2 Modificación de los datos de entrada

Este tipo de ataques se aprovechan del modo de uso normal de una API web. El cliente realiza una petición al servidor en la forma de una conexión a una endpoint concreto junto con los datos que desea transmitir. A estos datos se les conoce como carga útil o *payload*. Los metadatos estarían de excluidos de esta definición. Un atacante puede explotar todos ellos con intenciones maliciosas:

- Payload
    - Query string
    - Cuerpo de la petición
- Metadatos del protocolo
    - Cabeceras HTTP

La táctica de estos ataques es manipular el comportamiento esperado de alguno de estos elementos, para aprovecharse de la infraestructura que soporta la API web. Uno de los ataques más comunes dentro de esta categoría son las inyecciones de SQL.

3.2.1 Payload

Este tipo de ataques se ocultan tras la apariencia de peticiones aparentemente correctas e inofensivas. No obstante, sus parámetros de entrada son modificados con la intención de explotar tanto la aplicación como la infraestructura.

3.2.1.1 Inyecciones

Las inyecciones de código se deben a la introducción de valores no sanitizados como parámetros de una petición a la API. A continuación mostramos apenas unos ejemplos.

- Inyección de SQL

  La intención del atacante es ejecutar código SQL malicioso. Utiliza los datos transmitidos a la API como vector para atacar al servidor de base de datos que la sirve. Mediante un ataque de este tipo el usuario es capaz de alterar la base de datos, consultarla a su gusto e incluso saltarse los mecanismos de identificación y autorización.

  Imaginemos una carga útil inofensiva para consultar los datos de un pedido:

  ```
  {"order": {"id": "0B-AqCD3tjqjuT1ZLY2NoSn"}}
  ```

  Sin la protección adecuada, una carga útil modificada de tal manera, permitiría retornar la información de todos los pedidos.

  ```
  {"order": {"id": "0B-AqCD3tjqjuT1ZLY2NoSn OR '1'='1';"}}
  ```

- Inyección de XML

  Este tipo de ataques es crítico en APIs SOAP ya que XML es el formato de todos los datos intercambiados entre cliente y servidor. Si el atacante es capaz de inyectar los metacaracteres <, > y /, podrá inyectar etiquetas y alterar la lógica de la API. La inyección tiene que hacerse de tal forma que preserve la sintaxis XML, de otro modo el ataque no funcionará.

  Mediante un ataque de inyección SOAP un atacante puede llegar a hacer cambios en la base de datos. [18]

- Otros

  Es posible que mediante la inyección de código el usuario atacante sea capaz de ejecutar código LDAP, XPATH e incluso comandos del sistema en la máquina atacada.

Una vulnerabilidad en la API del servicio postal de Estado Unidos expuso los datos de 60 millones de usuarios. Un fallo en la sanitización de los datos de entrada no controlaba la inclusión de caracteres especiales en los términos de búsqueda, permitiendo así a cualquier usuario el acceso sin control a la información almacenada en la base de datos. [46]

3.2.1.2 *Parameter pollution*

El objetivo es contaminar los parámetros HTTP de una petición y modificar su funcionamiento normal de la aplicación. El protocolo HTTP permite el envío del mismo parámetro varias veces. Dependiendo de la tecnología subyacente el comportamiento varía. Muchas trabajan con el primer valor, otras con el último y algunas llevan a cabo algún tipo de combinación.

Technology/HTTP back-end	Overall Parsing Result	Example
ASP.NET/IIS	All occurrences of the specific parameter	par1=val1,val2
ASP/IIS	All occurrences of the specific parameter	par1=val1,val2
PHP/Apache	Last occurrence	par1=val2
PHP/Zeus	Last occurrence	par1=val2
JSP,Servlet/Apache Tomcat	First occurrence	par1=val1
JSP,Servlet/Oracle Application Server 10g	First occurrence	par1=val1
JSP,Servlet/Jetty	First occurrence	par1=val1
IBM Lotus Domino	Last occurrence	par1=val2
IBM HTTP Server	First occurrence	par1=val1
mod_perl,libapreq2/Apache	First occurrence	par1=val1
Perl CGI/Apache	First occurrence	par1=val1
mod_perl,lib???/Apache	Becomes an array	ARRAY(0x8b9059c)
mod_wsgi (Python)/Apache	First occurrence	par1=val1
Python/Zope	Becomes an array	['val1', 'val2']
IceWarp	Last occurrence	par1=val2
AXIS 2400	All occurrences of the specific parameter	par1=val1,val2
Linksys Wireless-G PTZ Internet Camera	Last occurrence	par1=val2
Ricoh Aficio 1022 Printer	First occurrence	par1=val1
webcamXP PRO	First occurrence	par1=val1
DBMan	All occurrences of the specific parameter	par1=val1~~val2

Ilustración 6: Comportamiento ante parámetros repetidos [19]

Imaginemos que un parámetro aportado por un usuario malicioso es empleado para generar una petición. El atacante puede alterar la URL, añadir parámetros o sobreescribir parámetros *hardcoded*. Algunos frameworks como Symphony incluyen el verbo REST en la petición como parámetro si no se trata de peticiones GET o POST. Con este tipo ataques el usuario incluso ese valor podría ser modificado. [20]

Por ejemplo, imaginemos que la API recoge el ID de un pedido para redirigir al al usuario al detalle del mismo. Si no existe sanitización alguna el usuario podría enviar el valor `id=whatever&_method=PUT&paid=true` para forzar que la operación sea una actualización del pedido y lo marque como pagado.

3.2.1.3 SSRF

Los ataques Server-Side Request Forgery (SSRF) permiten lanzar peticiones en nombre de la propia API, generalmente hacia elementos protegidos tras un

cortafuegos o una DMZ. Tiene lugar cuando el servidor debe llevar a cabo una petición cuya URL es formada usando información aportada por el usuario.

Por ejemplo, la API espera recibir la ruta a una imagen para fijarla como el avatar del usuario. Un usuario malicioso puede enviar una URL de un recurso accesible únicamente desde la red interna. La API llevará a cabo la petición, que tendrá éxito puesto que ambas máquinas se encuentra en la misma red, y el atacante será capaz de visualizar la información.

3.2.1.2 Ataques XXE

Los ataques XML External Entity o ataques XXE explotan el uso de los archivos Document Type Definition como definición de un archivo XML. Los DTDs permiten la definición de alias que pueden ser usados de manera maliciosa. El ataque de mil millones de risas es un ataque XXE que también podría ser considerado un ataque de denegación de servicio.

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
<!ENTITY lol "lol">
<!ENTITY lol2 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
<!ENTITY lol3
"&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
<!ENTITY lol4
"&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
<!ENTITY lol5
"&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
<!ENTITY lol6
"&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
<!ENTITY lol7
"&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
<!ENTITY lol8
"&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
<!ENTITY lol9
"&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

Al parsear este documento, la expansión de todas las entidades contendría 10^9 risas, que alcanzarían los 3 GB de espacio en memoria.

Con un alias de un archivo local podríamos acceder a información privilegiada.

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
<!ELEMENT foo ANY>
  <!ENTITY passwords SYSTEM
  "file:///etc/passwd">

]>
<lolz>&passwords;</lolz>
```

Con un alias a una ruta interna estaríamos ante un ataque SSRF.

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
<!ELEMENT foo ANY>
  <!ENTITY passwords SYSTEM
  "http://192.168.1.187/passwords.txt">

]>
<lolz>&passwords;</lolz>
```

3.2.1.5 Deserialización

La serialización es el proceso mediante el cual un objeto complejo es convertido en un formato susceptible de ser almacenado o enviado, para luego ser reconstruido. Como ya hemos mencionado en el capítulo anterior, los formatos más empleados por las APIs web son XML y JSON. El proceso de deserialización puede conllevar riesgos cuando los datos proceden de usuarios que han incluido contenido malicioso en los datos serializados. El fin último puede ser ganar acceso a recursos restringidos, la ejecución de código arbitrario o incluso la denegación del servicio.

Una de las mayores fugas de datos conocidas podría deberse a un ataque de este tipo. La fuga de datos de 2017 de la empresa Equifax que afectó a más de 140 millones de estadounidenses se debió a una vulnerabilidad del plugin Rest de Apache Struts. Este llevaba a cabo la deserialización sin ningún tipo de filtro y permitiría la ejecución de código. [22]

También las versiones de Jenkins previas a la 1.650 eran vulnerables a un ataque de este tipo. Usuarios sin privilegios eran capaces de ejecutar código, mediante el envío archivos XML convenientemente modificados a ciertos endpoints de su API. [21]

3.2.1.6 Recursos enumerados

El uso de IDs numéricos y autoincrementales es una práctica heredada del desarrollo web que parece inofensiva, pero simplifica enormemente la tarea de los atacantes, que ya no tienen la necesidad de averiguar los IDs de las entidades.

La fuga de datos de la filial india de McDonald's a través del endpoint http://services.mcdelivery.co.in/ProcessUser.svc/GetUserProfile de su API fue debido a la ausencia de cualquier tipo de mecanismos de autenticación. Expuso los datos personales de más de 2 millones de usuarios. El uso de IDs consecutivos simplificó aún más la tarea a los atacantes. [23]

3.2.2 Metadatos del protocolo

En este tipo de ataques el vector no es la carga útil del de la petición, sino la información adicional que es necesario incluir para que el protocolo de comunicación pueda funcionar de manera correcta. En el caso de las APIs web, el protocolo más común es HTTP. Por lo tanto, será en el que nos centraremos. En HTTP los metadatos se organizan en pares de clave-valor llamados cabeceras. En las cabeceras de una petición se incluye información como el tipo y versión del navegador empleado para realizar la petición, el sistema operativo del cliente o los formatos aceptados por el navegador. En las cabeceras de respuesta aparece información como la fecha y hora, el tipo de servidor, la codificación de la respuesta y su tamaño.[33]

```
▾ Request Headers
  ⚠ Provisional headers are shown
  Origin: https://www.reddit.com
  Range: bytes=957-1216
  Referer: https://www.reddit.com/
  User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.103 Safari/537.36
```

Ilustración 7: Ejemplo de cabeceras HTTP

El *modus operandi* consistirá en la inclusión de información maliciosa en las cabeceras de las peticiones con el fin de pervertir el funcionamiento del servidor. En el apartado 3.1.2.2 ya habíamos descrito el comportamiento del ataque Slowloris, que puede considerarse además de un ataque de denegación de servicio como un ataque por manipulación de metadatos del protocolo. En este caso no se introduce información maliciosa, pero el envío continuado de cabeceras para intentar mantener activa la conexión es en cualquier caso un empleo malicioso de estas.

3.2.2.1 HTTP verb tampering

En algunos casos, el control de acceso en servidores se hace teniendo en cuenta tanto el verbo HTTP como el recurso en sí. Un enfoque simplista puede restringir convenientemente solamente ciertos verbos y no otros. Sería posible así acceder de manera libre a recursos restringidos empleando verbos HTTP alternativos como HEAD o incluso cadenas de texto arbitrarias. [34]

Con frecuencia estas prácticas no se emplean de manera aislada, sino en conjunción con otro tipo de ataques como los de inyección de código. Por ejemplo, es posible modificar el verbo de una de petición para que sea de tipo POST. Es útil si sabemos que solo son sanitizados los parámetros de la *query string* y así inyectar el código malicioso en el cuerpo de la petición.

3.2.2.2 HTTP header injection

El ejemplo anterior mostraba cómo es posible modificar las cabeceras HTTP que enviamos al servidor. No obstante, es posible ir más allá y forzar al servidor a incluir código malicioso en las cabeceras HTTP de sus respuestas. Esto es posible cuando el servidor incluye datos introducidos por el usuario en ellas sin llevar a cabo ningún tipo de sanitización previa. Los caracteres problemáticos en este caso son los retornos de carro (`%0d` or `\r`) y las marcas de fin de línea (`%0a` or `\n`).

Supongamos que el servidor recoge el nombre indicado por el usuario para construir la cookie. Un usuario legítimo recibiría la siguiente respuesta.

```
HTTP/1.1 200 OK
...
Set-Cookie: author=Eduardo
```

Un usuario malicioso puede hacer uso de los caracteres especiales para introducir como nombre la cadena:

```
"Eduardor%0d%0aContent-Length:%200%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aC
ontent-Type:%20text/html%0d%0aContent-Length:%20100%0d%0a…
```

Generándose así dos respuestas:

```
HTTP/1.1 200 OK
...
Set-Cookie: author=Eduardo F. Casal
Content-Length: 0

HTTP/1.1 200 OK
Content-Type:"text/html"
Content-Length: 1000
```

La inyección de información en las cabeceras permite al atacante llevar a cabo ataques XSS, ya que le permite construir peticiones que inyecten código JavaScript en el cuerpo de la respuesta. Permite asimismo el envenenamiento de caché en servidores proxy. La inyección de una cabecera como en el ejemplo puede partir la respuesta del servidor en dos. Si es posible asociar esa segunda respuesta con una dirección URL concreta en el proxy, terceros usuarios que soliciten esa URL en futuras ocasiones recibirían la respuesta manipulada.

Con la modificación de la cabecera Set-Cookie sería posible modificar la sesión. También es posible revertir la política de seguridad fijada por la cabecera Content-Security-Policy e incluso llevar a cabo redirecciones maliciosas mediante la cabecera location.

3.3 Ataques de intermediario

Este tipo de ataques también es conocido como *man-in-the-middle*. El atacante es capaz de interceptar mensajes intercambiados entre emisor y receptor. El objetivo buscado por el atacante puede ser:

- Acceso a información privilegiada
- Modificación de peticiones legítimas
- Repetición de peticiones

Evidentemente, son susceptibles aquellos sistemas que intercambian datos no encriptados o firmados. También cuando no es posible establecer una sesión segura.

Los ataques MITM tienen lugar cuando el atacante es capaz de interceptar una conexión legítima y desdoblarla en dos. La primera conexión conecta al cliente con el atacante, mientras que la segunda comunica al atacante con el servidor. Esto convierte al atacante en una especie de servidor proxy que intercepta toda la información intercambiada entre las dos partes, pudiendo leerla y modificarla a voluntad, sin que ninguno de los involucrados sea consciente de ello.

3.3.1 ARP y DNS Spoofing

El protocolo ARP o protocolo de resolución de direcciones es el protocolo de la capa red encargado de relacionar direcciones IP con direcciones MAC. Cuando un dispositivo necesita comunicarse con una dirección IP concreta consulta la caché ARP para obtener la dirección IP correspondiente. En caso de no encontrarla, realizará una petición ARP a la que en teoría solo debería responder el dispositivo con esa dirección IP. Sin embargo, un usuario malintencionado podría responder a peticiones ARP que no le correspondiesen y suplantar identidades.

El protocolo DNS funciona de manera similar, pero traduce nombres de dominio a direcciones IP. En ambos casos, la víctima puede acabar conéctandose a un dispositivo malicioso bajo la falsa creencia de que se trata del auténtico servidor.

3.3.2 Session hijacking

Cuando hablamos de session hijacking nos referimos a la explotación maliciosa de una sesión establecida entre cliente y servidor. Entendemos como sesión una conexión con estado entre dos dispositivos. La conexión es iniciada, mantenida y finalizada mediante procesos claros y definidos. HTTP es un protocolo sin estado, esto es, cada petición es independiente de la siguiente. En sesiones HTTP ese estado suele ser implementado mediante una cookie de sesión que permite al servidor saber que el cliente ha sido autenticado. Esto supone que esa cookie podría ser capturada y empleada para suplantar la identidad de un cliente ante el servidor.

En julio de 2014 Stevie Graham descubrió una vulnerabilidad de la app de Instagram debido al uso de HTTP en algunos endpoints de la API. Esas llamadas incluían en sus cabeceras las cookies de sesión, lo que las hacía vulnerable a este tipo de ataques. [36]

3.3.3 SSL/TLS hijacking

El uso de SSL/TLS no nos blinda ante este tipo de ataques. Es posible llevar a cabo un ataque MITM sobre una conexión HTTPS de la misma manera que haríamos sobre una conexión HTTP. Solo sería necesario establecer dos sesiones SSL independientes para cada conexión desdoblada. El navegador podría informar al usuario de la invalidez del certificado digital, pero aún así el usuario podría ignorar el aviso por desconocimiento. En ciertos casos, el certificado digital empleado por el atacante puede estar comprometido o estar firmado por una Autoridad Certificadora (CA) válida y el nombre del dominio (CN) coincide con el del sitio web original. En esos casos, tampoco ni siquiera el navegador sería capaz de detectar problema alguno.[35]

3.4 Suplantación

El atacante se aprovecha de mecanismos de autenticación o autorización débiles para hacerse pasar por otros usuarios con el fin último de acceder a información privilegiada o llevar a cabo operaciones restringidas. Los ataques de session hijacking que vimos en el apartado en 3.3.2 pueden considerarse también como ataques de suplantación.

Los métodos de autenticación más comunes en APIs web son autenticación HTTP básica, API keys y OAuth. La autenticación HTTP básica es el enfoque más sencillo ya que no requiere el uso de cookies, IDs de sesión o procesos

complejos de establecimiento de la comunicación. Se emplea una cabecera HTTP en la que se comparte el nombre de usuario y la contraseña codificados en Base64, pero en ningún caso encriptados. Esto lo convierte en una opción muy poco recomendable en entornos reales.

En el caso de las API keys, se le asigna una cadena de texto plano generalmente a cada cliente que se conecta a la API. La manera de generar varía según el caso. En ocasiones se genera de manera aleatoria, mientras que en otras se obtiene a partir de los datos del cliente (por ejemplo, direcciones IP y MAC).

En 2018 una aplicación móvil creada ex profeso para la conferencia RSA de ese año incluía claves de seguridad en el propio código. Esto permitió a atacantes acceder a información personal de todos los asistentes de la conferencia. [45]

La gran diferencia que separa a OAuth de las dos alternativas anteriores es que OAuth es un sistema de autenticación, pero también de autorización. El usuario necesita un token para loguearse en el sistema. El usuario realiza una petición al servidor de autenticación. Si el servidor permite el acceso, devuelve un token al usuario, que este envía al sistema al que quiere loguearse. El token puede ser validado antes de servir datos al usuario.

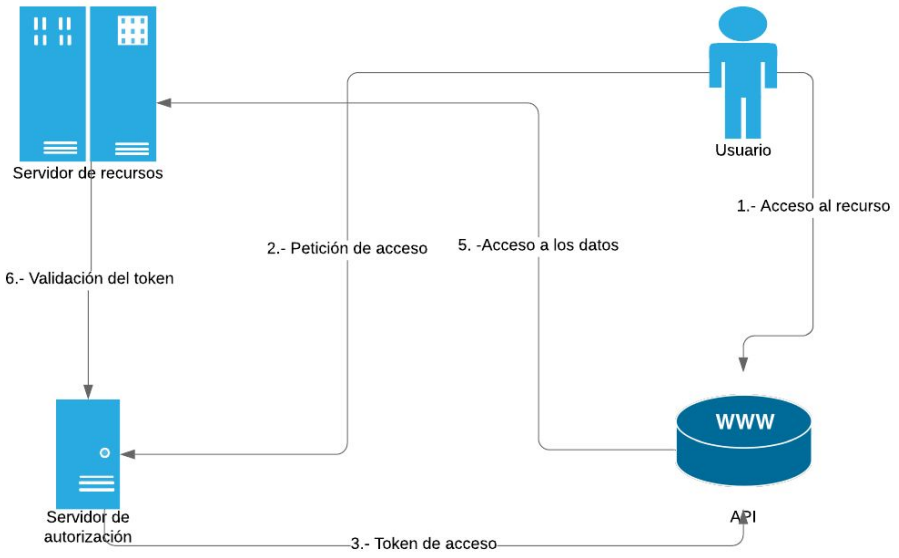

Ilustración 8: Proceso OAuth

Es posible limitar el número de sistemas para los cuales el usuario tiene acceso y limitar la validez del token generado. Su facilidad de configuración y su potencia lo han convertido en el estándar de facto. De hecho, tanto Twitter como Facebook han pasado por lo que de manera informal se ha conocido como OAuthpocalypse. En agosto de 2000 Twitter dejó de permitir autenticación básica en apps de terceros [40]. Una las razones que adujeron fue eliminar la posibilidad de exponer las contraseñas de los usuarios. En septiembre de 2011 Facebook obligó a todas las apps a migrar a OAuth 2.0 [41]. Meses antes Symantec había informado de que hasta 100000 aplicaciones podrían haber filtrado tokens de acceso a terceros.

4. Estrategias de mitigación

4.1 Ataques de denegación de servicio

Independientemente del vector empleado para llevar a cabo el ataque, existen una serie de medidas genéricas que pueden ser tomadas para evitar y llegado el caso, mitigar los efectos del ataque. El Instituto Nacional de Ciberseguridad de España [17] hace las siguientes recomendaciones:

4.1.1 Red

El primer paso para protegernos de los ataques de denegación más básicos es proveernos de suficiente ancho de banda con el fin de poder soportar picos puntuales de tráfico. Con la llegada de los *booter services* y los ataques amplificados es inviable poder contrarrestar ataques de denegación de servicio con esta técnica.

Las redes de entregas de contenidos o CDNs son una alternativa cuando es imposible disponer de más ancho de banda y sea necesario servir un gran volumen de peticiones de zonas alejadas. Akamai y Cloudflare son ejemplos de CDNs ampliamente utilizados. Suelen permitir además incluir reglas con las que identificar tráfico legítimo o aplicar cuotas y *throttling*.

Un CDN permite reducir la latencia ya que el usuario recibirá respuesta del servidor más cercano geográficamente. El gran número de servidores aumenta la fiabilidad y permite gestionar con más flexibilidad posibles picos de tráfico. [63]

4.1.2 Infraestructura

Es necesario mantener los elementos que conforman las infraestructura actualizados y convenientemente configurados. No solamente los servidores, sino también proxies, firewalls, routers y switches. Un firewall bien configurado puede bloquear ataques sencillos por sí mismo. Por otro lado, con cada servicio activado y puerto abierto aumenta la superficie de ataque, por lo que es imperativo la desactivación de todos aquellos que no sean empleados para evitar su explotación maliciosa.

Una infraestructura con redundancia en diferentes centros de datos y bien balanceada hará más complicada la tarea del atacante de bloquear el servicio

por completo. Será más difícil comprometer todos los servidores a la vez y los restantes podrían a soportar la carga de trabajo que en condiciones normales asumirían sus homólogos.

4.1.2 Aplicación

Como en el caso anterior, la instalación de actualizaciones seguridad es una tarea prioritaria para evitar problemas de seguridad. Lamentablemente, debido a la propia naturaleza de las APIs web, disponer de una versión estática como estrategia de mitigación no es viable.

Los ataques distribuidos de denegación de servicio en la capa de aplicación son más difíciles de detectar porque cada elemento de la botnet realizan peticiones en apariencia legítimas de manera individual. Por ello es recomendable el uso de *captchas*.

4.2 Modificación de los datos de entrada

En general, los Web Application Firewalls o WAFs son extremadamente útiles ya que permiten evitar este tipo de ataques, incluso cuando el código presenta vulnerabilidades sin subsanar, mediante la implementación de diferentes políticas de seguridad. Como su nombre indica este tipo de firewalls trabajan únicamente en la capa de aplicación. Actúan como un proxy inverso protegiendo a la aplicación, la API web en este caso, de usuarios maliciosos. Un WAF bien configurado es capaz de proteger de ataques de inyección, XSS y CSRF. El objetivo principal de un WAF es el control de contenido, no tanto de volumen. Las políticas de seguridad se aplican sobre el tráfico HTTP que pasa a través ellos.

Es imperativo monitorizar el impacto que este procesamiento puede tener, ya que puede generar una latencia que en definitiva sería contraproducente al propio propósito del WAF. Además de combinarlo con otras medidas como puede ser el empleo de un CDN.

4.2.1 Payload

4.2.1.1 Validación de los datos de entrada

En general la estrategia para evitar este tipo de ataques es considerar todo dato de entrada como potencialmente malicioso. Así pues, el primer paso será validación y sanitización de los datos de entrada, incluyendo longitud, rango

formato y tipo. Con frecuencia el lenguaje de programación dispone de librerías para ello, lo que no supondrá un trabajo extra de desarrollo. Podemos optar por el uso de schemas para una validación estricta de todos los datos de entrada. Tanto JSON como XML permiten esta posibilidad. La validación de cadenas de texto mediante expresiones regulares permite evitar los ataques SSRF.

También es posible validar el tamaño total de la petición. En ese caso, la respuesta incluirá el código de estado 413 Payload Too Large. Si el trabajo con grandes volúmenes de información es ineludible, es recomendable derivarlo hacia colas y trabajos en segundo plano. Esto es especialmente recomendable con servidores bloqueantes como Apache. Este tipo de servidores atienden cada petición de manera síncrona de manera que una acumulación de tareas puede impactar negativamente en el rendimiento del servicio global.

Todo contenido que no se adecúe a estas reglas deberá ser registrado y rechazado. Idealmente, la mayoría de estos sucesos serán fortuitos. No obstante, en caso de ser fruto de un ataque el registro de información puede resultar extremadamente útil *a posteriori* para prevenir futuros ataques. Es

4.2.1.2 Parseo de XML seguro

La manera más segura de evitar los ataques XXE es deshabilitar el empleo de DTDs. En caso de no ser posible evitar su uso por completo, deberemos limitar la inclusión de entidades y declaraciones de tipos externos.

4.2.1.3 Deserialización segura

Las vulnerabilidades asociadas a la deserialización tienen soluciones que varían en función del lenguaje de programación empleado. Como normal general se recomienda abandonar el uso de formatos de serialización nativos en favor de formatos estándar como JSON o XML. Una medida adicional de seguridad limitar la deserialización de los datos a aquellos que hayan sido firmados.[31]

4.2.2 Metadatos del protocolo

Para evitar los ataques de manipulación de los verbos HTTP podemos recurrir al uso de listas blancas a la hora de restringir el acceso a los recursos. Una fuente de vulnerabilidades viene de asumir que los verbos no incluidos en la configuración no están permitidos, cuando en ocasiones verbos como HEAD pueden pasar desapercibidos y ser usados como vectores en ataques. Es necesario tener en cuenta el verbo y el recurso al que se quiere acceder para

emitir un juicio sobre la validez de la petición. Todas la peticiones que no cumplan las restricciones fijadas deberán ser rechazas con el código 405 Method not allowed.

Como precaución adicional podemos comprobar el origen de los datos de entrada: *query string* o cuerpo de la petición. Podemos optar por hacer uso de ellos indistintamente, pero siempre convenientemente sanitizados.

En cuanto a los ataques de inyección de cabeceras HTTP solo cabe resaltar la necesidad de tratar todos los datos de entrada como potencialmente maliciosos como bien mencionábamos en el apartado anterior. En este caso es necesario prestar especial atención a los retornos de carro (`%0d` or `\r`) y las marcas de fin de línea (`%0a` or `\n`), conocidos comúnmente como caracteres CRLF o CR+LF, sobre todo cuando fijemos el valor de una cookie.

4.3 Ataques de intermediario

4.3.1 ARP y DNS Spoofing

Para evitar los ataques los ataques de falseamiento de ARP podemos recurrir a redes privadas virtuales o VPNs. Incluso si la red está comprometida, el atacante no será capaz de desencriptar la información interceptada. En redes pequeñas es posible recurrir a tablas estáticas para implementar la caché ARP.

En redes grandes esta solución es prácticamente inviable. En estos casos es posible recurrir al uso de DHCP *snooping* y rechazar los paquetes ARP fraudulentos. También es posible emplear RARP o ARP inverso para desechar direcciones MAC que tengan más de una dirección IP asociada. Es importante no perder de vista de que existen usos legítimos de este comportamiento.

El protocolo DNS es inseguro por sí mismo ya que se ideó en un punto en la seguridad no era una consideración crítica. Por esta razón nace DNSSEC, un conjunto de extensiones seguridad para DNS que permiten la autenticación de datos devueltos por el DNS.

4.3.2 Session hijacking

Todas las peticiones deben ser servidas vía HTTPS. Como vimos en el apartado 3.3.2 cualquier petición sobre HTTP puede servir como vector de ataques. Una petición HTTP puede ser usada como punto inicio de un ataque MITM y esto incluye hojas de estilos, imágenes, scripts y contenido de terceros.

Las cookies de sesión deben ser configuradas como Secure y HttpOnly. Cuando una cookie es enviada configurada como Secure el navegador nunca la enviará de vuelta en conexiones HTTP. Su contenido estará protegido por la encriptación proporcionada por HTTPS. La etiqueta HttpOnly asegura que el contenido de la cookie no puede ser accedido desde el cliente JavaScript, solo desde el servidor. [37]

4.3.3 SSL/TLS hijacking

SSL y TLS son usados como sinónimos en muchos casos, aunque esto no es del todo correcto. SSL precede a TLS y vio la luz en su versión 2.0 en 1995. TLS nació en 1999 como sucesor de la versión 3.0 de SSL. Ambas versiones de SSL están plagadas de vulnerabilidades y no se recomienda su uso. La versión más reciente de TLS es la 1.3 y elimina algoritmos con vulnerabilidades como DES y MD5. [50]

Así pues, el uso de TLS es imperativo en todas las comunicaciones, no solo aquellas en las que consideremos que se transmita información restringida. TLS garantiza la confidencialidad e integridad de toda la información intercambiada.

En cualquier caso, si la configuración del servidor no es correcta la API podrá ser víctima de ataques. Es imperativo el uso de claves fuertes, certificados que soporten todos los dominios y autoridades de certificación reconocidas en la generación de los certificados del servidor. [38]

Desde el cliente es necesario validar de manera correcta el certificado. Esto implica asegurarse de que el certificado ha sido firmado por una autoridad de certificación fiable, sino también que el valor del Common Name (CN) presente coincide con el del host con el que quiere conectarse.

Si estamos tratando con una API que maneje datos extremadamente sensibles, puede ser interesante valorar el uso de la práctica conocida como *certificate pinning*. En este caso, el cliente puede estar configurado para aceptar solo un certificado en particular (*hard certificate pinning*) o para aceptar solo aquellos firmados por un conjunto de autoridades de certificación concretas.

4.4 Suplantación

El primer paso a la hora de mitigar los ataques de suplantación es aplicar el método de autenticación más apropiado a cada situación. La autenticación HTTP básica es extremadamente insegura pero puede ser aceptable en redes

internas. En cualquier caso es imprescindible el empleo de TLS. Recordemos que los credenciales van codificados, pero no encriptados.

El uso claves de API es apropiado cuando las operaciones son limitadas. Por ejemplo, APIs públicas en las que la única operación permitida es la lectura. En cualquier caso, es recomendable el uso de algún tipo de método de autenticación aunque la API sea de libre acceso. Así evitaremos un uso abusivo de la API y sus recursos. Las peticiones demasiado numerosas pueden ser devueltas con el código 429 Too Many Requests. En última instancia la clave puede ser revocada indefinidamente.

La forma en la que la clave API es enviada también es importante. La clave API puede aparecer en la cabecera, el cuerpo o incluso en el query string. En general la información sensible no debe aparecer nunca en la URL de las peticiones. Toda contraseña, token o clave en la URL pueden ser registrados el historiales o logs.

Las claves API solo son un método de autenticación y no implementan autorización sin un desarrollo adicional. En caso de APIs más complejas que manejen datos de más de un usuario se requieren soluciones más complejas para gestionar el acceso de manera más segura y potente. OAuth maneja tokens de acceso cuyo ámbito y alcance se pueden limitar. De esta manera es posible restringir los datos a los que tiene acceso un usuario y las operaciones que puede llevar a cabo sobre ellos. Además, es posible revocar los tokens de acceso y fijar su caducidad. Para renovarlos se hace uso de un token especial llamado token de refresco o refresh token.

No obstante, OAuth 2.0 también presenta vulnerabilidades que han sido ampliamente documentadas [64]. Una de esas vulnerabilidades permite explotar la `redirect_uri`, la URI a la que el servidor de autorización redireccionará su respuesta. Esto ya es una vulnerabilidad en sí misma si no se emplean precauciones adicionales. Para evitar que los atacantes usen cualquier URI, muchos servidores OAuth fuerzan que esa URI tenga una estructura concreta. Generalmente obligan a que esa URI pertenezca a un dominio concreto. Aún así, si el atacante consigue encontrar una vulnerabilidad en dicha página, por ejemplo XSS, puede obtener el código de autorización.

Otro tipo de ataques se aprovecha de la ausencia del parámetro state. El parámetro state es opcional por lo que es posible para un atacante iniciar el proceso de Authorization Code Grant, realizar únicamente la primera petición al servidor de autorización, obtener el código de autorización, para luego forzar a la víctima, mediante ingeniería social por ejemplo, a completar el proceso usado dicho código de autorización.

En ocasiones OAuth se combina con el uso de tokens web JSON o JWT. Este define un estándar un abierto para la compartición de información de todo tipo entre dos partes en formato JSON. El uso de este formato permite el ahorro de consultas a la base datos para consultar la validez del token, porque el propio token contiene la información necesaria para determinar la identidad y el

alcance del acceso garantizado codificada en base64. Este tipo de tokens no pueden ser revocados, por lo que hay combinarlos con el uso de tokens de refresco y fechas de caducidad explícitas. En la cabecera de cada token se incluye el algoritmo usado para su firma como pueden ser SHA256 o RSA. Como medida de seguridad es recomendable rechazar todo token sin firmar. Además en la carga útil del token es recomendable incluir una serie de claves [44]:

- `iss`: emisor del token, ha de coincidir con el dueño de la clave cifrado empleada en el token.
- `aud`: audiencia del token. Indica si el token está destinado al usuario que está haciendo uso de él.
- `exp`: fecha de expiración del token.
- `nbf`: fecha de inicio de validez del token.

Estos tokens incluyen en su cabecera una clave `alg` que indica el algoritmo empleado para su firma. Primero se codifican en base 64 la cabecera y la payload para generar el token sin firmar, para posteriormente cifrarlo con este algoritmo. Para verificar un token debemos emplear el mismo algoritmo que indica la cabecera y es ahí donde surgen los problemas, ya que son los atacantes los que controlan el algoritmo empleado. Es posible emplear el valor `none`, de manera que no se emplee algoritmo alguno. Puede tener sentido su uso en situaciones en las que la integridad del token ya ha sido comprobada, pero algunas librerías tratan estos tokens como tokens válidos con una firma verificada. En la práctica esto permite a cualquier usuario crear tokens “firmados” con el contenido que deseen. Es importante no perder de vista que incluso los métodos más seguros requieren un uso y configuración adecuados para evitar vulnerabilidades.

5. Estrategias de prevención

Una vez que hemos visto todos los riesgos y vulnerabilidades que pueden afectar a las APIs web es imperativo tomar un enfoque proactivo e intentar asegurar en medida de lo posible el funcionamiento correcto de ellas. Un funcionamiento correcto no solamente se limita a la obtención de los resultados esperados, sino que también garantiza que el proceso se realiza de manera segura, de tal forma que la integridad de los datos y de la infraestructura subyacente no se vea afectada.

Desgraciadamente, con frecuencia las pruebas no reciben toda la atención que merece, y más cuando este se centra hacia la seguridad. En una encuesta realizada por Smartbear, empresa puntera en la automatización de pruebas, durante un seminario web sobre seguridad se realizaron dos sencillas preguntas. Las dos preguntas eran ¿Cuán importante es la seguridad de la APIs en tu organización? ¿Cuánto esfuerzo dedicáis a las pruebas de seguridad a día hoy? [47]

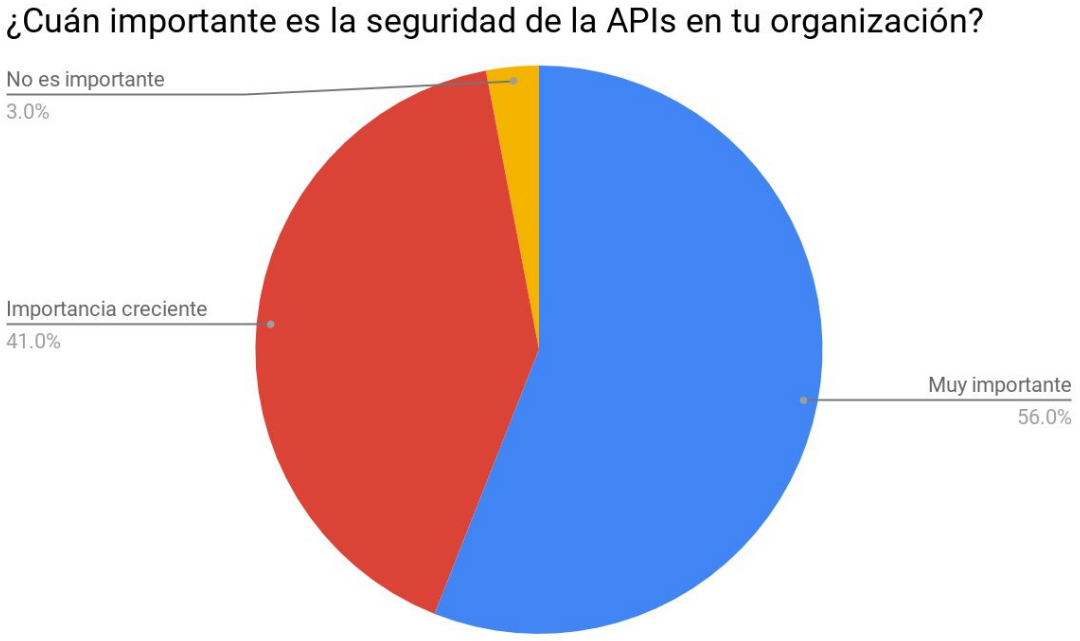

Ilustración 9: ¿Cuán importante es la seguridad de la APIs en tu organización?

Ilustración 10: ¿Cuánto esfuerzo dedicáis a día de hoy?

A pesar de que no disponemos de datos sobre el tamaño de la muestra, queda patente la desconexión entre la percepción y el esfuerzo real dedicado.

La superficie vulnerable es amplia y el tipo de ataques es muy variado, por lo que el uso de herramientas que faciliten la tarea se hace casi indispensable. En todo caso no es recomendable olvidar que la API se apoya y que los esfuerzos no deben limitarse a la capa de aplicación. Como ya hemos visto, muchos ataques de denegación de servicio explotan capas inferiores.

Low Orbit Ion Cannon es una herramienta de código abierto que permite realizar ataques de denegación de servicio. Su uso legítimo se enmarca dentro de las pruebas de stress. Sin embargo, es fácil ver cómo podría ser usada con fines maliciosos. LOIC permite lanzar ataques TCP, UDP o HTTP *flood* de manera muy sencilla a través de una interfaz gráfica.

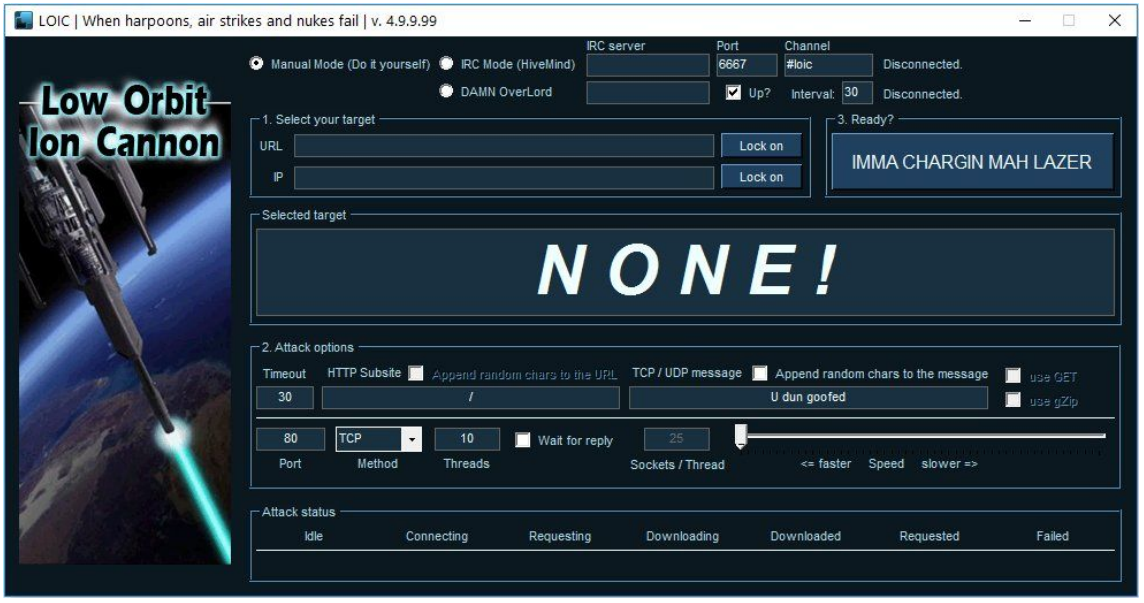

Ilustración 11: Low Orbit Ion Cannon

Fue empleado por Anonymous en sus ataques a la iglesia de la Cienciología y a Asociación de Industria Discográfica de Estados Unidos (RIAA).

El uso de sniffers también resulta muy conveniente a la hora de analizar el tráfico intercambiado con la API. Fiddler y Wireshark son dos de los sniffers más populares. Fiddler actúa como un servidor proxy y permite capturar tráfico HTTP y HTTPS para su posterior análisis. Para hacerlo se comporta de manera similar a un atacante de tipo *man-in-the-middle* con certificados autofirmados. Evidentemente, para que Fiddler puede desencriptar los paquetes HTTPS es necesario que el certificado sea aceptado por el navegador. Fiddler no es una autoridad de certificación de confianza y por seguridad el certificado no será aceptado inicialmente.[48]

Wireshark, en cambio, es una analizado de paquetes, por lo que no se limita al tráfico HTTP(s) y puede ayudar a detectar errores en capas inferiores a la de aplicación. También en caso de trabajar con APIs SOAP que usen SMTP en vez de HTTP. En caso de trabajar con HTTPS es posible desencriptar los paquetes si se emplea una infraestructura de clave pública y se conoce la clave privada. [51] [52]

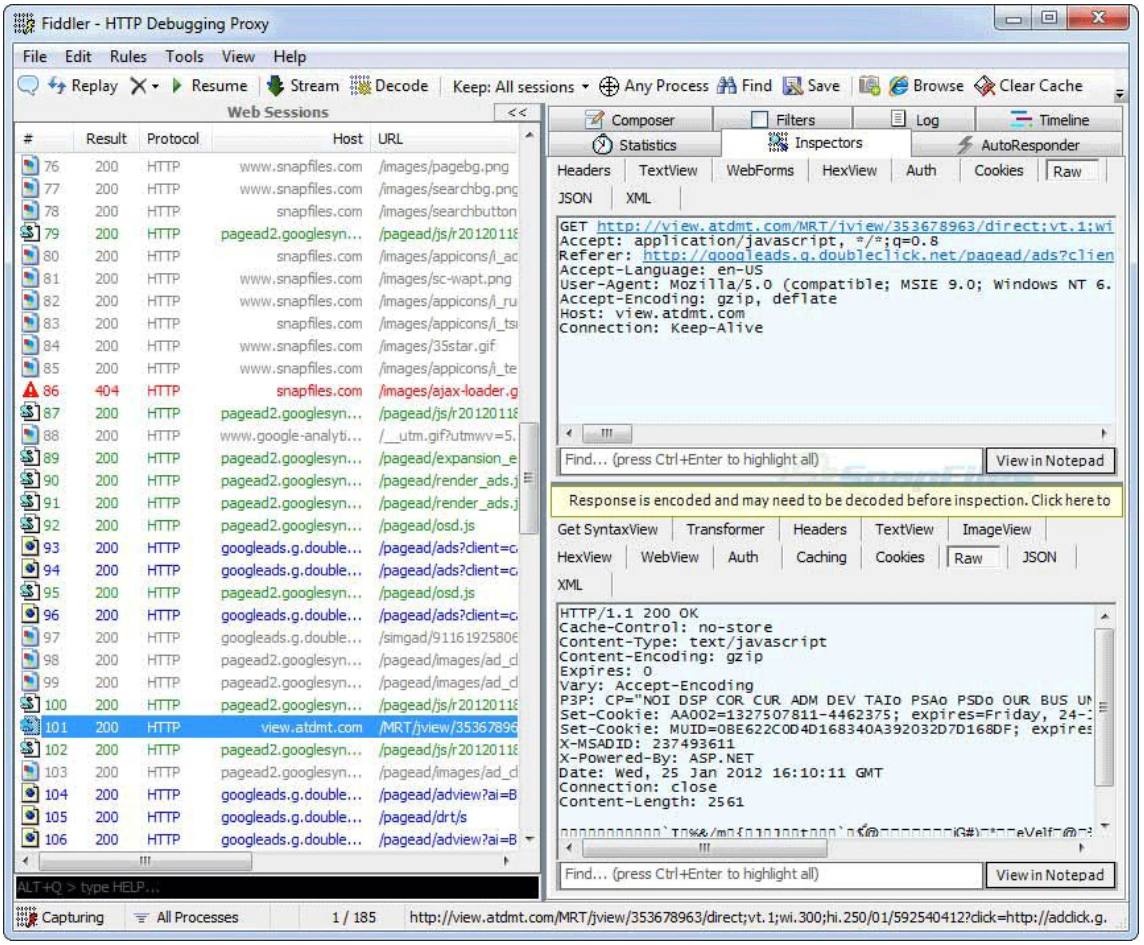

Ilustración 12: Fiddler

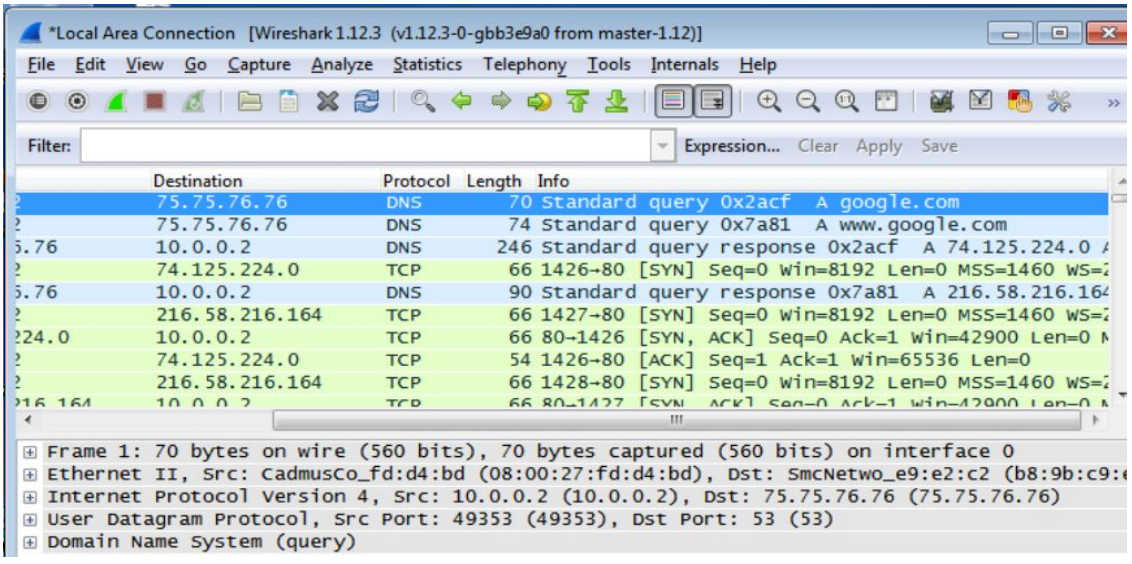

Ilustración 13: Wireshark

A la hora de automatizar pruebas de APIs una de las herramientas más empleadas es Postman. Postman es un cliente API que nos permite ejecutar llamadas a APIs web y procesar sus respuestas de manera muy sencilla.

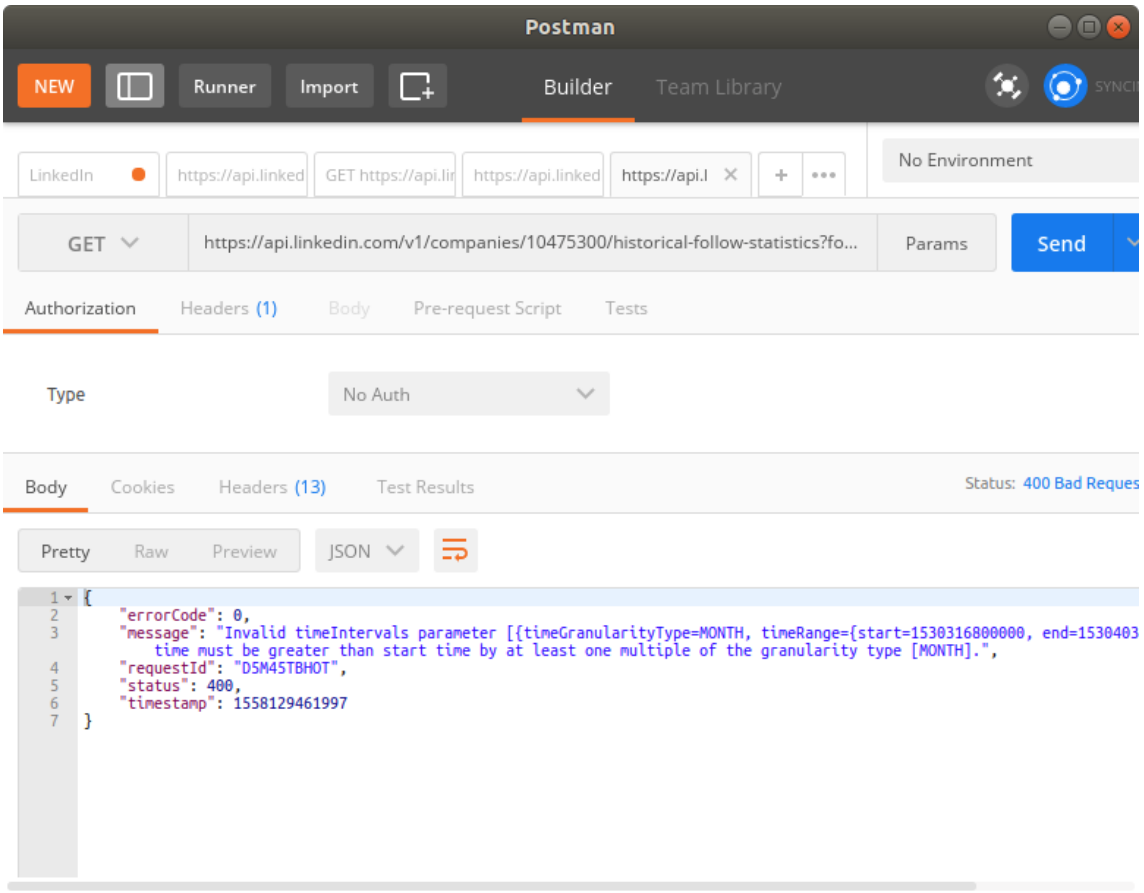

Ilustración 14: Llamada API en Postman

Con una interfaz sencilla es posible fijar la URL y el verbo HTTP, añadir cabeceras, escoger el método de autenticación, añadir parámetros y definir el contenido del cuerpo de la petición. Una vez enviada, la respuesta es visualizada en la parte inferior de la interfaz de usuario. La verdadera potencia de la herramienta está en la posibilidad de crear pruebas. Estas pruebas son scripts de JavaScript que realizan llamadas a la API y comparan sus estados y resultados con los valores previstos y esperados. El siguiente ejemplo comprueba que el resultado de la llamada API devuelve un JSON con una clave de nombre “amount” cuyo valor ha de ser 145:

```
pm.test("Your test name", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.amount).to.eql(145);
});
```

Estas pruebas se pueden organizar en colecciones y ejecutar desde línea de comandos. De esta manera es muy sencillo integrarlas dentro de entornos de integración continua ya habilitados.

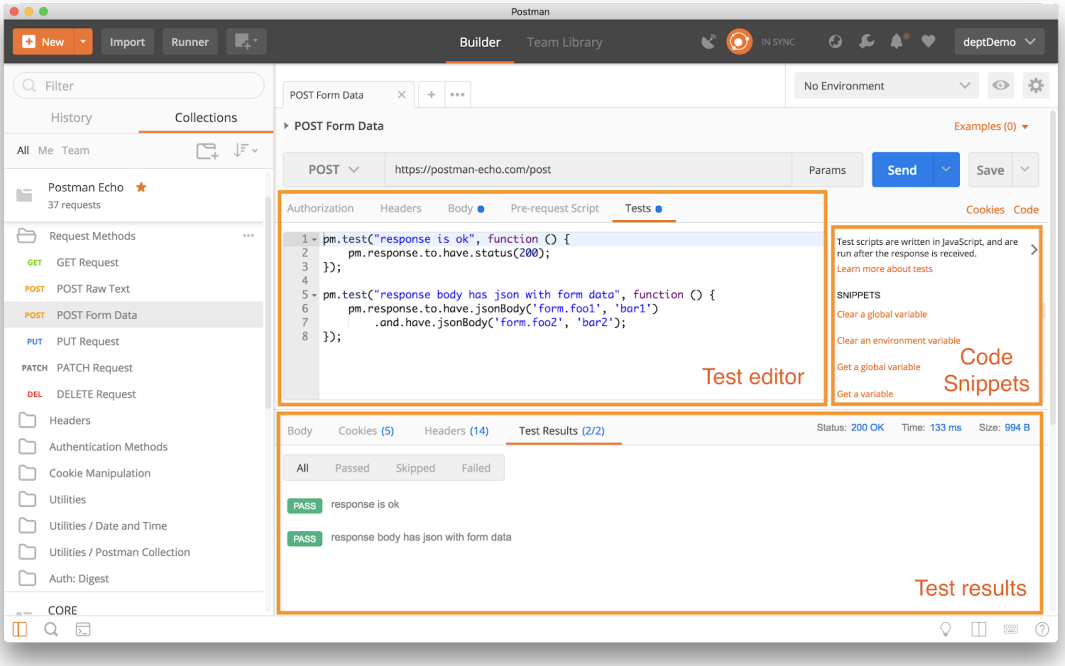

Ilustración 15: Prueba en Postman

5.1 Penetration testing

Las pruebas de seguridad pueden resultar abrumadoras por la extensión de la superficie de ataque. Penetration testing también conocido como *pen testing* o *hacking* ético pretende formalizar el proceso de búsqueda de vulnerabilidades mediante la simulación de ataques. El proceso puede realizarse de manera manual o automatizada y puede aplicarse a aplicaciones web pero también redes y sistemas completos. Su objetivo principal es la identificación de puntos débiles para su posterior subsanación.

El trabajo se puede organizar en múltiples fases. En la literatura el número varía [54][55], a continuación listamos las más importantes:

- Recopilación de información. La primera fase consiste en obtener información del sistema que vamos a atacar. Para ellos se recurre a las técnicas de *footprinting* y *fingerprinting*. Un ejemplo de estas prácticas es el escaneo de puertos con nmap.

- Identificación de vulnerabilidades. En esta fase se listan los puntos de acceso, vectores de ataque y amenazas internas y externas. Con frecuencia esta tarea es automatizada y el probador se limita a valorar de la explotabilidad de las vulnerabilidades encontradas.

- Explotación. El objetivo es aprovecharse de la vulnerabilidad con el objetivo de llevar a cabo la simulación del ataque hasta un límite establecido con anterioridad. El proceso debe quedar documentado de manera exhaustiva. Después de esta fase el sistema debe de ser devuelto a su situación inicial.

- Análisis. Finalizada la explotación de vulnerabilidades, el objetivo es recopilar toda la información generada para ofrecer una evaluación global de seguridad y establecer un plan de acción para subsanar las vulnerabilidades encontradas.

Ilustración 16: Fases de *penetration testing*

5.1.1 Tipos

Existen numerosos tipos de estrategias a seguir dentro de este tipo de pruebas de seguridad:

- Pruebas de caja blanca: en este caso el probador sí que dispone de información adicional sobre el sistema, como puede ser datos sobre la infraestructura, tecnologías empleadas o incluso el propio código.

- Pruebas externas: el ataque se realiza desde el exterior del sistema, simulando el *modus operandi* de un usuario no autorizado.

- Pruebas internas: el ataque es realizado por parte de un usuario autorizado con permisos estándar. La idea es simular el efecto de un ataque de una quinta columna.

- Pruebas ciegas: el probador no dispone de ninguna información extra sobre la infraestructura, como si se tratara de un atacante real. También se conocen como pruebas de caja negra. Implican un mayor esfuerzo y son por lo general muy costosas ya que el tiempo empleado en la recopilación de información se dispara.

- Pruebas ciegas dobles: da un paso más en el realismo de las pruebas ciegas y en este caso el grueso de la organización no es informado de que la prueba se está llevando a cabo. Este tipo de pruebas es el más cercano a un ataque real.

5.1.2 Herramientas

Herramientas como Wireshark, Fiddler y Postman que ya hemos mencionado y cuyo funcionamiento ya hemos descrito también puede ser empleadas. No obstante, resulta interesante el empleo de herramientas que permitan automatizar el proceso de identificación de vulnerabilidades. Existen numerosas alternativas entre que las destacan Metasploit [58], Nessus [59] y Burpsuite [60]. Para detectar vulnerabilidades a nivel de aplicación OWASP ZAP (Zed Attack Proxy) [56] es especialmente útil.

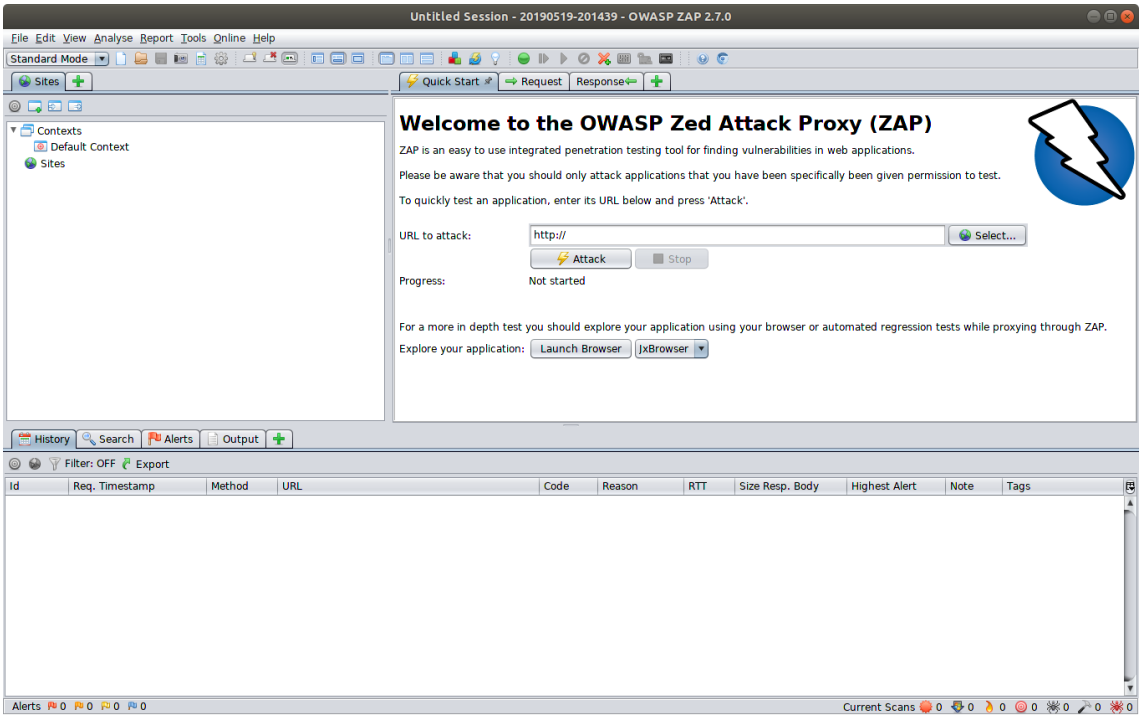

Ilustración 17: OWASP ZAP

Es una herramienta gratis y libre que funciona como un proxy entre el cliente y la aplicación web. Permite el análisis de vulnerabilidades a medida que el probador navega o de manera autónoma, en lo que denominan modo araña. Soporta dos modos de exploración. La exploración pasiva solo analiza las respuesta y no modifica los datos de la API. La exploración activa realiza ataques reales por lo que solo es recomendable hacerlo en entornos controlados. A medida que OWASP ZAP detecte vulnerabilidades, informará de ellas en la pestaña de alertas.

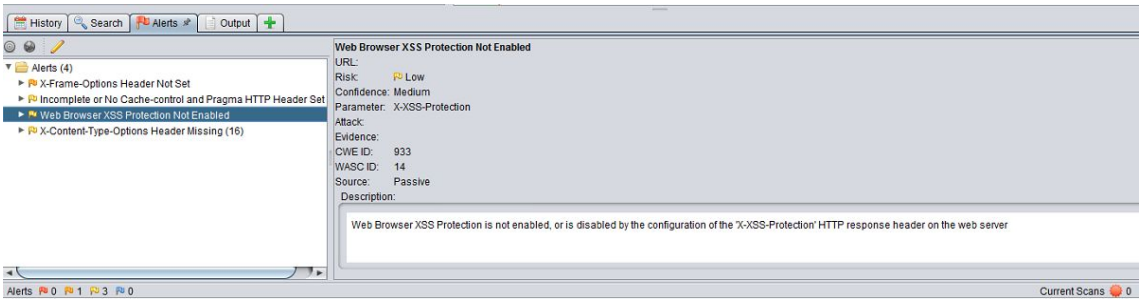

Ilustración 18: Pestaña de alertas en OWASP ZAP

5.2 Fuzz testing

Otra técnica de pruebas es la conocida como *fuzz testing* o *fuzzing*. El objetivo sigue siendo buscar vulnerabilidades de manera automatizada, pero mediante el uso de datos de entrada aleatorios o inválidos. Es una técnica poco costosa, si bien un tanto lenta, que permite detectar vulnerabilidades como inyecciones de código y XSS de manera sencilla, también ataques de denegación de servicio. En cualquier caso, no es recomendable abandonar otras técnicas, ya que las pruebas de *fuzzing* por sí mismas no pueden detectar vulnerabilidades más complejas.

El componente encargado de generar esos datos se denomina *fuzzer*. Dependiendo del método seguido para generar los datos aleatorios podemos categorizar los *fuzzers* en dos grupos:

- De mutación. Usa muestras de datos como punto de partida que modifica sucesivamente.

- De generación. Crea nuevos datos a partir de una especificación proporcionada por el probador o especificada en un protocolo.

OWASP ZAP dispone de una extensión llamada fuzzdb [57] que permite incorporar esta funcionalidad.

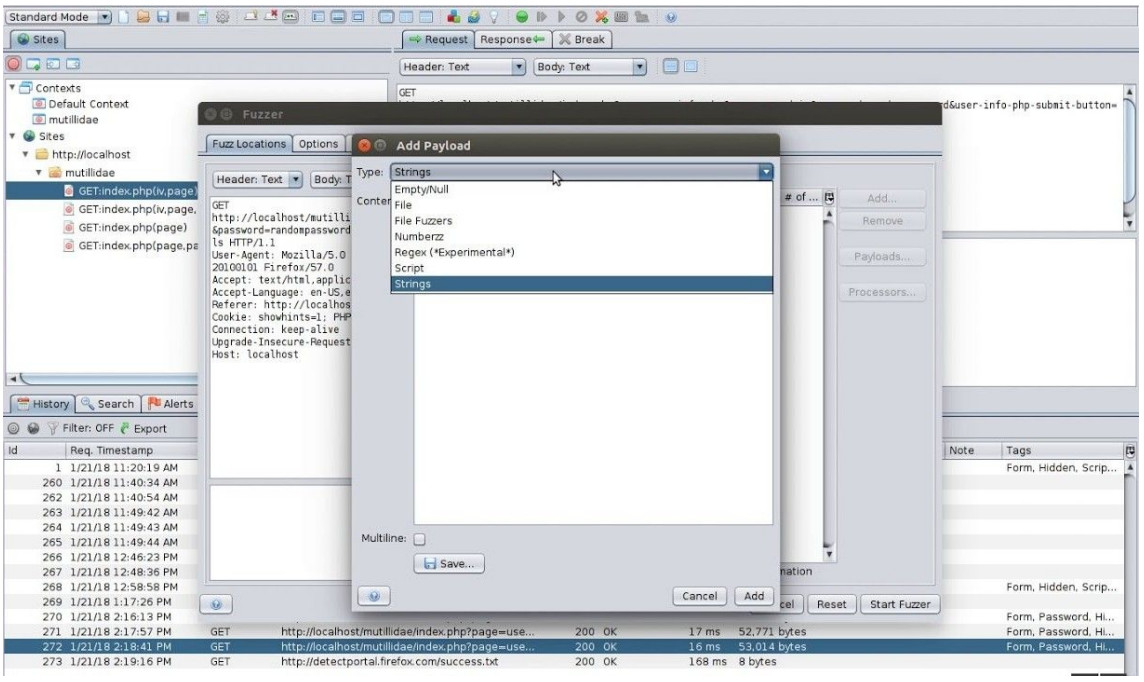

Ilustración 19: Fuzzdb en OWASP ZAP

6. Conclusiones

Hemos recorrido la evolución histórica de las APIs web y analizado las causas de su auge y ubicuidad actuales. Hemos listado y detallado sus vulnerabilidades más frecuentes y los ataques más comunes de los que son víctima. Cuando ha sido posible hemos ofrecido ejemplos de casos reales. También hemos listado soluciones para mitigar o prevenir estas vulnerabilidades en forma de pruebas de seguridad.

El objetivo de este trabajo era obtener una visión global de las APIs web desde el punto de vista de la seguridad. Lo podemos considerar como cumplido, ya que hemos respetado la estructura de la memoria fijada en el plan de trabajo la haber recopilado y presentado información sobre todos los temas que pretendíamos tratar inicialmente.

En general, la planificación se ha cumplido. La división del trabajo en entregas parciales ha permitido controlar de manera más eficaz el tiempo. No obstante, ha sido necesario bascular contenido de la segunda entrega a la tercera por razones ajenas a este proyecto. Gracias al desglose de tareas tan detallado ha sido posible aplicar un correctivo para recuperar el tiempo perdido de manera sencilla.

Debido al objetivo generalista del trabajo en sí, se abren numerosas líneas de trabajo a futuro. Por limitaciones de tiempo se ha tratado de manera muy somera las tecnologías empleadas en el desarrollo de APIs web. De hecho solo se han tratado tres SOAP, REST y GraphQL como representantes del pasado, presente y futuro del desarrollo. Existen muchas otras e incluso las que hemos tratado podrían estudiarse de manera más profunda. Hemos mencionado muy levemente la autenticación OAuth y los JWT. Sería posible dedicar trabajos completos al análisis de su funcionamiento y sus posibles vulnerabilidades. También existe la posibilidad de ahondar más en las diferentes técnicas de pruebas de seguridad, de las que hemos incluido sólo dos como las más reseñables.

7. Glosario

API: *application programming interface* o interfaz de programación de aplicaciones.

ARP: *Address Resolution Protocol* o protocolo de resolución de direcciones. Protocolo de la capa de red que relaciona direcciones MAC con direcciones IP.

Booter service: servicio para realizar ataques DDoS bajo demanda.

Bot: abreviatura de robot. Aplicación o script que realiza tareas automatizadas bajo demanda. Bots maliciosos permiten a atacantes tomar el control de manera remota de los equipos infectados.

Botnet: Red de bots conectados entre sí que pueden ser controlados colectivamente de manera remota.

CDN: *content delivery network* o red de distribución de contenidos. Red servidores distribuidos geográficamente con el fin de garantizar una alta disponibilidad.

DDoS: distributed denial of service o ataque distribuido de denegación de servicio. Es un tipo de ataque de denegación de servicio en el que el servicio víctima es atacado desde varios puntos.

DES: *Data Encryption Standard* algoritmo de encriptación de clave simétrica. La longitud es de solo 56 bits, lo que lo convierte en poco seguro actualmente.

DHCP: *Dynamic Host Configuration Protocol* o protocolo de configuración dinámica de host es protocolo de red en el que un servidor asigna dinámicamente una dirección IP a un cliente.

DHCP snooping: técnicas dedicadas a mejorar la seguridad de una infraestructura DHCP. Incluye el filtrado de peticiones ARP y el control de direcciones IP.

DNS: *Domain Name System* o sistema de nombres de dominio sistema jerárquico encargado de traducir nombres de dominio en direcciones IP.

DoS: *denial of service* o ataque de denegación de servicio. Busca inutilizar un servicio, de manera que no sea accesible por ningún usuario.

Dirección MAC: identificador de 48 bits unívoco de un dispositivo de red.

Footprinting: técnica de recopilación de información de sistemas informáticos. Suele preceder a un ataque.

Fuzzing: técnica automatizada de pruebas que consiste es emplear valores incorrectos, inesperados o aleatorios como entrada.

GraphQL: lenguaje de código abierto para la consulta y manipulación de datos en APIs.

Hard certificate pinning: técnica que consiste en forzar un cliente a aceptar un certificado en particular cuyos detalles se ha fijado previamente.

ICMP: *Internet Control Message Protocol* o protocolo de control de mensajes de Internet se emplea para enviar mensajes de error al emisor de un paquete IP.

JSON: *JavaScript Object Notation* o notación de objeto de JavaScript es un formato de texto sencillo para el intercambio de datos.

JWT: *JSON Web Token* es un estándar abierto basado en JSON para la creación de tokens de acceso

Mashup: un nuevo sitio web o servicio construido usando como componentes fragmentos de otros sitios web o servicios. Son extremadamente comunes aquellos que hacen uso de la API de Google Maps.
MD5:

Payload: Datos transmitidos en una comunicación, aparte de cabeceras y metadatos. También conocido como carga útil.

PSD2: *Payment Services Directive 2*. Directiva europea que tiene como objetivo mejorar la competencia, la innovación y la protección al consumidor en el mercado financiero.

OAuth: *Open Authorization* es un estándar abierto que permite la autenticación y autorización sin necesidad de intercambiar contraseñas.

Penetration testing: ciberataque autorizado y simulado con el fin de evaluar la seguridad de un sistema.

Query string: parte de la URL que asigna valores a parámetros.

REST: *Representational State Transfer* es un estilo de arquitectura sin estado en el que los recursos son identificados mediante una URI.

RPC: *Remote Procedure Call* o llamada a procedimiento remoto permite la comunicación entre aplicaciones usando un formato estándar comprendido tanto por emisor como por receptor

Sniffer: hardware o software que permite capturar y registrar el tráfico sobre una red.

SOAP: Simple Object Access Protocol es un protocolo que define el intercambio de información estructurada

SSRF: *Server Side Request Forgery* son ataques que permiten a un atacante forzar a un dispositivo a realizar peticiones desde dentro del sistema hacia el exterior.

SYN flood: tipo de ataque DoS consiste en el envío de peticiones SYN del protocolo TCP.

Throttling: modificación intencionada de la velocidad de un servicio con el fin de regular el tráfico.

TCP: *Transmission Control Protocol* o Protocolo de control de transmisión garantiza el envío de datos sin errores y en orden.

TPP: Third Party Payment providers o proveedores externos de pago son introducidos por la PSD2 y actuarán como intermediarios entre banco y cliente.

UDP: *User Datagram Protocol* o protocolo de datagramas de usuario es un protocolo de la capa de transporte que, a diferencia de TCP, no garantiza el envío correcto o en orden.

VPN: *Virtual Private Network* o red privada virtual permite el envío de información sobre redes públicas o no controladas como Internet con la seguridad de una red privada.

XML: eXtensible Markup Language o lenguaje de marcado extensible permite la codificación de información de manera legible para humanos y dispositivos.

8. Bibliografía

[1] ProgrammableWeb. API Directory Eclipses 17,000 as API Economy Continues Surge. 2019-02-22
https://www.programmableweb.com/news/programmableweb-api-directory-eclipses-17000-api-economy-continues-surge/research/2017/03/13

[2] Api Evangelist. History of APIs. 2019-03-05
http://apievangelist.com/2012/12/20/history-of-apis/

[3] Hackernoon. What are Web APIs. 2019-03-05
https://hackernoon.com/what-are-web-apis-c74053fa4072

[4] OpenLegacy Blog. The History of APIs and How They Impact Your Future. 2019-03-05
https://www.openlegacy.com/blog/the-history-of-apis-and-how-they-impact-your-future

[5] Pusher. Announcing The State of APIs 2018 report. 2019-03-05
https://blog.pusher.com/state-of-apis-2018/

[6] ProgrammableWeb. Which API Types and Architectural Styles are Most Used?. 2019-03-07
https://www.programmableweb.com/news/which-api-types-and-architectural-styles-are-most-used/research/2017/11/26

[7] Digimonica. PSD2 and the API challenge for Open Banking. 2019-03-07
https://diginomica.com/psd2-and-the-api-challenge-for-open-banking/

[8] RapidApi. 2019-03-07
https://rapidapi.com/

[9] GitHub. toddmotto/public-apis. 2019-03-07
https://github.com/toddmotto/public-apis

[10] IBM. SOAP. 2019-03-09
https://www.ibm.com/support/knowledgecenter/en/SSGMCP_5.2.0/com.ibm.cics.ts.webservices.doc/concepts/soap/dfhws_overview.html

[11] JSON-RPC. JSON-RPC 2.0 Specification. 2019-03-09
https://www.jsonrpc.org/specification

[12] XML.RPC. XML-RPC Specification. 2019-03-09
http://xmlrpc.scripting.com/spec.html

[13] W3C. SOAP specification. 2019-03-09
https://www.w3.org/TR/soap/

[14] KrebsOnSecurity. DDoS-for-Hire Service Webstresser Dismantled. 2019-04-12
https://krebsonsecurity.com/2018/04/ddos-for-hire-service-webstresser-dismantled/

[15] Wired. A Straightforward Timeline of the FCC's Twisty DDoS Debacle. 2019-04-12
https://www.wired.com/story/fcc-net-neutrality-investigation/

[16] FFC. Alleged Multiple Distributed Denial-Of-Service (DDoS) Attacks involving the FCC's Electronic Comment Filing System (ECFS). 2019-04-12
https://www.fcc.gov/sites/default/files/fcc-oig-roi-ecfs-ddos-08072018.pdf

[17] INCIBE-CERT. Medidas de protección frente ataques de denegación de servicio (DoS). 2019-04-12
https://www.incibe-cert.es/blog/medidas-proteccion-frente-ataques-denegacion-servicio-dos

[18] SecureLayer7. Penetration Testing with SOAP Service and Mitigation. 2019-04-12
http://blog.securelayer7.net/owasp-top-10-penetration-testing-soap-application-mitigation/

[19] OWASP. TTP Parameter Pollution OWASP EU09 Poland presentation. 2019-04-13
https://www.owasp.org/images/b/ba/AppsecEU09_CarettoniDiPaola_v0.8.pdf

[20] The Symfony CookBook. How to Use HTTP Methods beyond GET and POST in Routes. 2019-04-13
https://symfony.com/doc/2.6/cookbook/routing/method_parameters.html

[21] National Vulnerability Database. CVE-2016-0792. 2019-04-13
https://nvd.nist.gov/vuln/detail/CVE-2016-0792

[22] INCIBE-CERT. Ejecución remota de código en el plugin REST de Apache Struts 2. 2019-04-13
https://www.incibe-cert.es/alerta-temprana/avisos-seguridad/ejecucion-remota-codigo-el-plugin-rest-apache-struts-2

[23] Hacker Noon. McDonalds India is leaking 2.2 million users data. 2019-04-13
https://hackernoon.com/mcdonalds-india-is-leaking-2-2-million-users-data-d5758b2eb3f8

[24] Akamai. 2019 State of the Internet / Security: DDoS and Application Attacks. 2019-04-15
https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/state-of-the-internet-security-ddos-and-application-attacks-2019.pdf

[25] Akamai. 2019 State of the Internet / Security: Retail Attacks and API Traffic. 2019-04-15
https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/state-of-the-internet-security-retail-attacks-and-api-traffic-report-2019.pdf

[26] Akamai. [state of the internet] / security Q4 2017 Report
https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/q4-2017-state-of-the-internet-security-report.pdf

[27] Imperva. Q1 2017 Global DDoS Threat Landscape Report. 2019-04-15
https://www.imperva.com/blog/q1-2017-global-ddos-threat-landscape-report/

[28] WS-Attacks. SOAP Parameter DOS. 2019-04-15
https://www.ws-attacks.org/SOAP_Parameter_DOS

[29] PHP Denial of Service Attack Revisited. 2019-04-15
http://lukasmartinelli.ch/web/2014/11/17/php-dos-attack-revisited.html

[30] GitHub - shieldfy. API-Security-Checklist: Checklist of the most important security countermeasures when designing, testing, and releasing your API. 2019-04-16
https://github.com/shieldfy/API-Security-Checklist

[31] OWASP/CheatSheetSeries · CheatSheetSeries / Deserialization Cheat Sheet. 2019-04-16
https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Deserialization_Cheat_Sheet.md

[32] ha.ckers. Slowloris HTTP DoS. 2019-04-16
https://web.archive.org/web/20150426090206/http://ha.ckers.org/slowloris

[33] W3. HTTP/1.1: Header Field Definitions. 2019-04-16
https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html

[34] Infosec. Http Verb Tempering: Bypassing Web Authentication and Authorization. 2019-04-16
https://resources.infosecinstitute.com/http-verb-tempering-bypassing-web-authentication-and-authorization/

[35] OWASP. Man-in-the-middle attack. 2019-04-20
https://www.owasp.org/index.php/Man-in-the-middle_attack

[36] SecurityWeek.Com. Instagram App Exposes Users to Man-in-the-Middle Attacks: Researcher. 2019-04-20
https://www.securityweek.com/instagram-app-exposes-users-man-middle-attacks-researcher

[37] HTTP | MDN. HTTP cookies. 2019-04-20
https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies#Secure_and_HttpOnly_cookies

[38] OWASP/CheatSheetSeries · GitHub. Cheat Sheet Series / Transport Layer Protection Cheat Sheet. 2019-04-20
https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.md#server-certificate

[39] Symantec. Certificate Pinning. 2019-04-20
https://www.symantec.com/content/dam/symantec/docs/white-papers/certificate-pinning-en.pdf

[40] Twitter. Doug Bowman. 2019-04-22
https://twitter.com/stop/status/22537707799

[41] Facebook for Developers. Developer Roadmap Update: Moving to OAuth 2.0 + HTTPS. 2019-04-22
https://developers.facebook.com/blog/post/497

[42] Symantec Connect. Facebook Applications Accidentally Leaking Access to Third Parties - Updated|. 2019-04-22
https://www.symantec.com/connect/blogs/facebook-applications-accidentally-leaking-access-third-parties

[43] Hypertext Transfer Protocol (HTTP/1.1): Authentication. 2019-04-22
https://tools.ietf.org/html/rfc7235

[44] JWT. Introduction to JSON Web Tokens. 2019-04-22
https://jwt.io/introduction/

[45] Ars Technica. When you go to a security conference, and its mobile app leaks your data. 2019-04-22
https://arstechnica.com/information-technology/2018/04/insecure-rsa-conference-app-leaked-attendee-data/

[46] Data Breach Today. US Postal Service Plugs API Flaw - One Year Later. 2019-04-22
https://www.databreachtoday.com/us-postal-service-plugs-api-flaw-one-year-later-a-11722

[47] Test & Protect Your API: Practical Tips for API Security. 2019-04-28
https://smartbear.com/learn/api-testing/practical-tips-for-api-security/

[48] Telerik. Fiddler - Free Web Debugging Proxy. 2019-05-03
https://www.telerik.com/fiddler

[49] Wireshark · Go Deep. 2019-05-03
https://www.wireshark.org/

[50] Network World. What is Transport Layer Security (TLS)? 2019-05-03
https://www.networkworld.com/article/2303073/lan-wan-what-is-transport-layer-security-protocol.html

[51] Accedian. How to Decrypt an HTTPS Exchange with Wireshark? 2019-05-03
https://accedian.com/enterprises/blog/how-to-decrypt-an-https-exchange-with-wireshark/

[52] Devops World – Medium. HTTP/HTTPS Analysis Using Wireshark. 2019-05-03
https://medium.com/devops-world/http-https-analysis-using-wireshark-cbe07c23520

[53] SearchSecurity. What is pen test (penetration testing)? 2019-05-04
https://searchsecurity.techtarget.com/definition/penetration-testing

[54] A Complete Guide to the Phases of Penetration Testing. 2019-05-04
http://blog.cipher.com/the-6-primary-phases-of-penetration-testing

[55] CyberX. 7 Penetration Testing Phases to Achieve Amazing Results - CyberX. 2019-05-04
https://cyberx.tech/penetration-testing-phases/

[56] OWASP Zed Attack Proxy Project. 2019-05-04
https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

[57] fuzzdb-project/fuzzdb. 2019-05-04
https://github.com/fuzzdb-project/fuzzdb

[58] Nessus Professional™ Vulnerability Scanner. 2019-05-04
https://www.tenable.com/products/nessus/nessus-professional

[59] Metasploit. Getting Started with Metasploit for Penetration Testing. 2019-05-04
https://www.metasploit.com/

[60] PortSwigger. Burp Suite Scanner. 2019-05-04
https://portswigger.net/burp

[61] OWASP Testing Guide Appendix C: Fuzz Vectors. 2019-05-04
https://www.owasp.org/index.php/OWASP_Testing_Guide_Appendix_C:_Fuzz_Vectors

[62] Accenture. Innovate with security. 2019-03-07
https://www.accenture.com/_acnmedia/PDF-85/Accenture-Innovate-With-Security.pdf

[63] Akamai Developer - How a CDN Can Make Your APIs More Powerful. 2019-05-23
https://developer.akamai.com/blog/2018/11/13/how-cdn-can-make-your-apis-more-powerful

[64] A Comprehensive Formal Security Analysis of OAuth 2.0. 2019-06-18
https://arxiv.org/pdf/1601.01229v3.pdf

[65] JSON Web Algorithms (JWA). 2019-06-18
https://tools.ietf.org/html/rfc7518