



**Universidad Politécnica de Madrid**  
Escuela Técnica Superior  
de Ingeniería de Sistemas Informáticos

# **Desarrollo de Estrategias de Defensa para Ataques Denial of Wallet en Arquitecturas Serverless**

Trabajo de Fin de Grado en Tecnologías para la Sociedad de la Información

Elaborado por Javier Fernández Castiella  
Tutorado por Jorge Blasco Alís

## Abstracto

Este trabajo tiene como objetivo el desarrollo de estrategias de defensa frente a ataques Denial of Wallet (DoW) en entornos Serverless, una tecnología cada vez más adoptada en la computación en la nube. Estos ataques, que generan costes económicos al activar funciones de manera maliciosa, representan una amenaza emergente para los usuarios. A través de un enfoque iterativo y el uso de entornos simulados con tecnologías de código abierto, se analizan distintos tipos de ataque y se proponen defensas específicas. La efectividad de cada defensa se evalúa comparando los daños antes y después de su implementación. Además, se abordan las implicaciones legales, éticas y sociales del trabajo. El objetivo final es contribuir a la mejora de la ciberseguridad en arquitecturas Serverless mediante soluciones prácticas y aplicables.

## Abstract

This project aims to develop defence strategies against Denial of Wallet (DoW) attacks in Serverless environments, a rapidly growing area within cloud computing. These attacks seek to inflict financial damage on victims by maliciously triggering serverless functions, without compromising the service itself. Using an iterative approach and simulated environments built with open-source technologies, various attack types are implemented and analysed. Specific defence mechanisms are then developed, and their effectiveness is evaluated by comparing the impact before and after deployment. Legal, ethical, and social aspects are also considered. Ultimately, this work aims to enhance cybersecurity in Serverless architectures through practical and applicable defensive strategies.

## Resumen

Este trabajo tiene como objetivo principal el desarrollo de estrategias de defensa frente a ataques Denial of Wallet (DoW) en entornos Serverless, una tecnología en expansión dentro del ámbito de la computación en la nube. Serverless permite ejecutar funciones sin necesidad de gestionar servidores, cobrando únicamente por el tiempo de ejecución, lo que aporta ventajas como escalabilidad, portabilidad y reducción de costes. Sin embargo, esta eficiencia también presenta nuevas amenazas, entre ellas el ataque DoW, que busca generar costes económicos a la víctima mediante la activación maliciosa y repetitiva de funciones serverless, sin necesidad de comprometer el servicio.

Actualmente, estos ataques son poco comunes, lo que ofrece una oportunidad única para investigar y diseñar defensas eficaces antes de que se conviertan en una amenaza extendida. A través de una metodología iterativa, se implementarán diferentes estrategias de ataque (masivos, prolongados, de tasa adaptativa, etc.) sobre un entorno serverless simulado con tecnologías de código abierto. Se analizarán los costes generados y se desarrollarán defensas específicas para cada tipo de ataque. Posteriormente, se evaluará la efectividad de dichas defensas mediante una comparativa del daño antes y después de su aplicación.

Finalmente, se considerarán los aspectos éticos, sociales, legales y medioambientales del desarrollo e implementación de estas defensas. Ante el crecimiento exponencial del uso de la nube y el aumento de ataques en estos entornos, este trabajo busca contribuir a la ciberseguridad preventiva en arquitecturas serverless.

# Índice

1. Introducción .....	9
1.1 Motivación .....	10
1.2 Objetivo y alcance .....	11
1.3 Estructura del documento.....	12
2. Marco Teórico.....	13
2.1 El paradigma Cloud .....	13
2.2 Serverless y sus tecnologías.....	15
2.2.1 Flujo General.....	16
2.2.2 Seguridad en Serverless.....	18
2.2.2.1 Modelo de Responsabilidad Compartida.....	18
2.2.2.2 Tipología de amenazas .....	19
2.2.2.3 Prácticas Recomendadas .....	19
2.3 Estado del arte .....	20
2.3.1 Definición de Denial of Wallet.....	20
2.3.2 Tipología de ataques DoW.....	21
2.3.3 Detección de ataques .....	22
2.3.4 Simulación.....	23
3. Metodología .....	25
4. Arquitectura .....	27
4.1 Víctima .....	27
4.2 Atacante .....	31
4.3 Trafico Base .....	33
4.4 Red.....	33
5. Desarrollo .....	35

5.1 Tiempo de procesamiento.....	35
5.2 Tamaño y formato óptimo.....	36
5.3 Ataque Masivo DoS.....	44
5.3.1 Implementación.....	45
5.3.2 Ejecución.....	46
5.3.3 Análisis del Ataque .....	46
5.3.4 Desarrollo de la Defensa .....	50
5.3.5 Análisis de la Defensa.....	52
5.3.6 Conclusiones.....	57
5.4 Ataque Leech .....	57
5.4.1 Implementación.....	57
5.4.2 Ejecución.....	57
5.4.3 Análisis del Ataque .....	58
5.4.4 Desarrollo de la Defensa .....	63
5.4.5 Análisis de la defensa .....	64
5.4.6 Conclusiones.....	69
6. Limitaciones .....	71
7. Aspectos Legales, Sociales, Éticos, y Ambientales .....	73
7.1 Agenda 2030 y Objetivos de desarrollo sostenible.....	73
7.2 Impacto Social .....	75
7.3 Otras Consideraciones Éticas .....	77
7.4 Aspectos Legales .....	78
8. Conclusiones .....	81
9. Bibliografía.....	82

## Índice de Figuras

<b>Figura 1.</b> Arquitectura de los contenedores .....	pág. 14
<b>Figura 2.</b> Arquitectura de las máquinas virtuales .....	pág. 14
<b>Figura 3.</b> Tipos de servicios cloud según el dominio de las capas .....	pág. 14
<b>Figura 4.</b> Flujo de las solicitudes en OpenFaaS .....	pág. 16
<b>Figura 5.</b> Flujo de la metodología .....	pág. 26
<b>Figura 6.</b> Formato de las solicitudes HTTP .....	pág. 28
<b>Figura 7.</b> Configuración de despliegue de la función .....	pág. 29
<b>Figura 8.</b> Panel de estatus de la función .....	pág. 29
<b>Figura 9.</b> Panel de monitorización del nodo Kubernetes .....	pág. 30
<b>Figura 10.</b> Arquitectura de los servicios del dispositivo víctima .....	pág. 31
<b>Figura 11.</b> Diagrama de clases del atacante .....	pág. 32
<b>Figura 12.</b> Arquitectura de la red .....	pág. 34
<b>Figura 13.</b> Tiempo de procesamiento en función del tamaño de la imagen en formato PNG .....	pág. 35
<b>Figura 14.</b> Tiempo de procesamiento en función del tamaño de la imagen en formato JPG .....	pág. 36
<b>Figura 15.</b> Rélicas de la función durante el procesamiento de imágenes de distintos tamaños en formato JPG .....	pág. 39
<b>Figura 16.</b> Rélicas de la función durante el procesamiento de imágenes de distintos tamaños en formato PNG .....	pág. 39
<b>Figura 17.</b> Consumo de CPU por réplica durante el procesamiento de imágenes en formato JPG .....	pág. 40
<b>Figura 18.</b> Consumo de CPU por réplica durante el procesamiento de imágenes en formato PNG .....	pág. 41
<b>Figura 19.</b> Consumo de memoria por réplica durante el procesamiento de imágenes en formato JPG .....	pág. 42
<b>Figura 20.</b> Consumo de memoria por réplica durante el procesamiento de imágenes en formato PNG .....	pág. 42
<b>Figura 21.</b> Tiempo medio de respuesta durante el procesamiento de imágenes en formato JPG .....	pág. 43
<b>Figura 22.</b> Tiempo medio de respuesta durante el procesamiento de imágenes en formato PNG .....	pág. 43
<b>Figura 23.</b> Rélicas de la función durante la ejecución de un ataque masivo .....	pág. 46
<b>Figura 24.</b> Número de invocaciones recibidas en los últimos 10s .....	pág. 47
<b>Figura 25.</b> Tiempo medio de procesamiento de las solicitudes .....	pág. 47
<b>Figura 26.</b> Consumo de CPU por réplica de la función durante el ataque masivo .....	pág. 48
<b>Figura 27.</b> Consumo de CPU por réplica de la función durante el ataque masivo (continuación) .....	pág. 49
<b>Figura 28.</b> Configuración de la alerta y el webhook .....	pág. 50

<b>Figura 29.</b> Arquitectura de los servicios de la víctima con defensa .....	pág. 51
<b>Figura 30.</b> Rélicas de la función durante el ataque masivo con defensa ...	pág. 52
<b>Figura 31.</b> Tasa de invocación de la función durante el ataque masivo con defensa .....	pág. 52
<b>Figura 32.</b> Tiempo medio de procesamiento durante el ataque masivo con defensa .....	pág. 53
<b>Figura 33.</b> Consumo de CPU durante el ataque masivo con defensa .....	pág. 54
<b>Figura 34.</b> Consumo de memoria durante el ataque masivo con defensa....	pág. 54
<b>Figura 35.</b> Rélicas de la función durante la ejecución del ataque <i>leech</i> .....	pág. 58
<b>Figura 36.</b> Tasa de invocación durante la ejecución del ataque <i>leech</i> .....	pág. 59
<b>Figura 37.</b> Invocaciones recibidas en los últimos 10 minutos durante la ejecución del ataque <i>leech</i> .....	pág. 59
<b>Figura 38.</b> Tiempo de procesamiento durante la ejecución del ataque <i>leech</i> .....	pág. 60
<b>Figura 39.</b> Consumo de CPU durante la ejecución del ataque <i>leech</i> .....	pág. 60
<b>Figura 40.</b> Consumo de memoria durante la ejecución del ataque <i>leech</i> ....	pág. 61
<b>Figura 41.</b> Comparación de tráfico base y malicioso, y umbral escogido.....	pág. 60
<b>Figura 42.</b> Rélicas durante la ejecución del ataque <i>leech</i> con defensa.....	pág. 62
<b>Figura 43.</b> Tiempo de procesamiento durante la ejecución del ataque <i>leech</i> con defensa .....	pág. 64
<b>Figura 44.</b> Número de invocaciones recibidas en los últimos 10 minutos durante la ejecución del ataque <i>leech</i> con defensa .....	pág. 65
<b>Figura 45.</b> Tiempo medio de procesamiento durante la ejecución del ataque <i>leech</i> con defensa .....	pág. 65
<b>Figura 46.</b> Consumo de CPU durante la ejecución del ataque <i>leech</i> con defensa .....	pág. 66
<b>Figura 47.</b> Consumo de memoria durante la ejecución del ataque <i>leech</i> con defensa .....	pág. 67

## 1. Introducción

La computación en la nube es una de las tecnologías más importantes de nuestros tiempos, sobre la cual se basan un sinfín de servicios, plataformas, y compañías. Sin embargo, los modelos tradicionales de alquiler de recursos físicos o virtuales en centros de computación suponen un sobrecoste para el contratante en aquellos momentos en los que no se necesitan dichos recursos. Las tecnologías Serverless surgieron hacia finales de los 2000 prometiendo poner fin a esto.

El modelo es simple, para el programador al menos: definir una función abstrayéndose completamente del entorno físico en el que será ejecutada, y pagar única y exclusivamente por el uso de los recursos cuando se ejecute la función. En esto consiste Serverless. En abandonar el servidor tradicional en favor de una función contenerizada, que puede ser desplegada bajo demanda, por solicitud directa o distintos eventos, e igualmente detenerla tras su fin.

Las ventajas son evidentes e incluyen mayor flexibilidad y escalabilidad, mayor facilidad de implementación, disminución de costes, aumento de la rentabilidad de la infraestructura física, portabilidad, etc. Todo esto ha contribuido a una rápida adopción de la tecnología, que, aunque no haya reemplazado a otros modelos cloud, sí está en aumento constante con entre el 45% a 75% de los clientes de distintas plataformas Cloud usando al menos una función Serverless [1].

No obstante, una nueva tecnología supone también nuevas posibles amenazas. En nuestro caso una de estas amenazas ya se ha tipificado como “Denial of Wallet” o Denegación de Cartera. Este nuevo tipo de ciberataque pretende generar costes de procesamiento a la víctima mediante la activación de las funciones a través de solicitudes directas. Se asemeja al ataque por denegación de servicio, pero a una escala distinta. En este caso no se pretende bloquear el servicio si no que basta con generar costes, ya sea de manera puntual o más prolongada en el tiempo.

Se trata de un ataque del que, de momento, existen pocos ejemplos lo cual proporciona una oportunidad única de investigar y definir estrategias de defensa para prevenirlos antes de que ocurran. En abril de 2024 se hizo público un caso en

el que un usuario de AWS vio su factura aumentar más de 1000\$ debido a que uno de sus buckets de AWS recibió millones de solicitudes ajenas [2]. En este caso las solicitudes fueron todas en un día, pero otras estrategias permiten el envío de solicitudes en tiempos mucho más prolongados dificultando su detección.

El objetivo de este trabajo es el desarrollo de estrategias de defensas para ataques DoW en entornos serverless. Aprovechando el momento actual, donde no supone una gran amenaza, pretendo investigar el efecto que pueden causar estos ataques, desarrollar estrategias de defensa, y medir su efectividad y aplicabilidad en entornos reales, definiendo una metodología rigurosa y basándome en la literatura científica actual entorno a este fenómeno.

## 1.1 Motivación

Según un informe de Statista sobre el uso de cloud pública, se estima que la facturación de la industria cloud en 2024 alcance la cifra de 773.000M \$, con un crecimiento anual estimado de un 18,49% en el periodo 2024-2029 [3]. Y es que, debido a la cantidad de ventajas que aporta este tipo de servicios, no es de extrañar. Desde la flexibilidad de la infraestructura, la escalabilidad, hasta la disminución de costes, resulta una opción muy atractiva para una gran cantidad de negocios. Con soluciones personalizadas, distintos tipos de servicios (SaaS, PaaS, IaaS...) y una innovación y mejora constante de la industria, tanto grandes como pequeñas empresas se pueden beneficiar de esta, ya no tan nueva, tecnología.

Esta adopción masiva supone también una serie de retos tanto para el cliente como para el proveedor del servicio. En concreto, en un informe elaborado por Flexera donde se incluían un total de 753 empresas de diversos tamaños, el segundo reto más importante para las entidades es la ciberseguridad [4]. Otro de los retos más importantes es la falta de conocimiento sobre la tecnología, ya que, al subcontratarlo, se delega parcialmente en el proveedor.

Estas amenazas se ven reflejadas en las cifras. Solo en el año 2023 las intrusiones en entornos cloud aumentaron un 75% [5]. Las cifras aumentan tanto en ataques cloud conscientes, donde los actores son conscientes de haberse infiltrado en

entornos cloud y utilizan sus características en el ataque, como en ataques agnósticos, donde los actores no son conscientes del entorno cloud. Respectivamente un 110% y un 60%.

Con un panorama político cada vez más tenso, donde tanto organizaciones individuales como otras secundadas por gobiernos hostiles, y donde el cibercrimen supone ya un coste de 9,5B \$ anuales [6], es de vital importancia adelantarse a estos actores nefarios y desarrollar estrategias de defensa preventivas para evitar los daños.

## 1.2 Objetivo y alcance

El objetivo de este proyecto es el desarrollo de estrategias de defensa adaptadas a las distintas estrategias de ataque en ciberataques *Denial of Wallet* analizando su efectividad y aplicabilidad en entornos reales, en concreto AWS.

Para ello se pretende desarrollar una arquitectura, usando soluciones de código abierto, que simule de manera verosímil un entorno cloud Serverless. En este entorno se simularán ciberataques siguiendo distintas estrategias, se analizarán los ataques determinando los costes incurridos por la víctima, el consumo de recursos virtuales y físicos, y demás aspectos relevantes.

Tras este análisis, se desarrollarán estrategias de defensa adaptadas a cada ataque que permitan reducir el daño causado, analizando su efectividad mediante una comparativa de los daños causados por el mismo ataque antes y después de la implementación de la defensa.

Se analizarán distintas estrategias de ataque que incluyen ataques masivos, similares a DoS, ataques de larga duración, ataques de tasa adaptiva, etc. Para cada ataque la simulación se adaptará a las limitaciones impuestas por los recursos disponibles, utilizando métodos de extrapolación cuando se alcance el límite de los recursos.

Por último y debido a la importancia del aspecto legal, ético, y social de cualquier labor, se determinarán los impactos sociales y ambientales y se analizarán las implicaciones legales del propio proyecto.

### 1.3 Estructura del documento

Habiendo introducido el proyecto y establecido claramente la motivación y el objetivo del desarrollo de estrategias de defensa para ataques *DoW*, se define en esta sección la estructura del resto del documento.

- Marco Teórico: se realizará un análisis de las tecnologías implicadas en este proyecto y la literatura científica reciente relevante.
- Metodología: se definirá el método de investigación a seguir que permitirá obtener resultados claros y realizar comparativas justas sobre las distintas estrategias obtenidas.
- Arquitectura: se explicará la arquitectura diseñada explicando razonadamente las decisiones de diseño tomadas y sus implicaciones.
- Iteraciones: esta sección contiene la investigación realizada de manera iterativa. Incluirá los experimentos realizados, sus consecuencias, las estrategias obtenidas, y el análisis de los resultados para cada tipo de ataque.
- Impacto social, ambiental, y legal: se realizará un análisis de los distintos aspectos sociales del proyecto.
- Conclusiones: se determinará aquí si los objetivos del proyecto han sido alcanzados y se resumirán las conclusiones obtenidas durante la investigación.

## 2. Marco Teórico

En esta sección se realizará un análisis de las tecnologías implicadas en este proyecto y de la literatura científica relevante.

### 2.1 El paradigma Cloud

La computación en la nube, en inglés *Cloud Computing*, se define según la ISO como “paradigm for enabling network access to a scalable and elastic pool of shareable physical or virtual resources with self-service provisioning and administration on-demand [7]”. En definitiva, se caracteriza por el reemplazo de los recursos locales por una serie de recursos, físicos o virtuales, a los que se accede de manera remota, para cualquier tipo de tarea.

Se basa en múltiples tecnologías cuyas funcionalidades se combinan para ofrecer el servicio. Una de las más importantes es la virtualización. Se trata de un proceso mediante el cual se utiliza software para abstraer el hardware físico de un dispositivo y así permitir la ejecución de distintas máquinas sobre un único hardware [8].

Existen muchos tipos de virtualización, pero examinaremos los más relevantes para este proyecto, que son [9]:

- Hypervisor: un tipo de virtualización en el que un motor de virtualización, hypervisor, permite alojar distintos sistemas operativos sobre la misma máquina. El hypervisor actúa de interfaz entre el SO nativo y el virtual, permitiendo así la abstracción del hardware.
- Contenerización: en este tipo de virtualización, las aplicaciones que se desean alojar se empaquetan en una unidad denominada contenedor, junto con las librerías y todos los archivos necesarios para su ejecución. No requiere simular el sistema operativo virtual, sino que el motor de contenedores realiza la labor de interfaz entre el SO nativo y las aplicaciones.

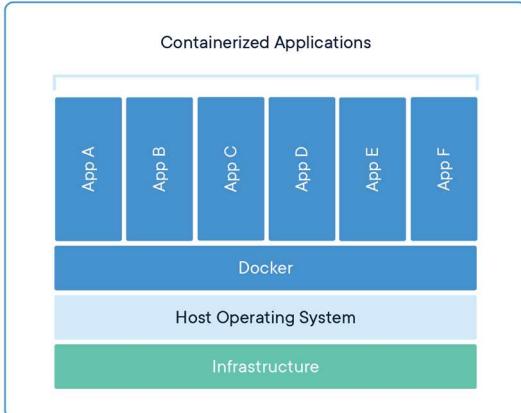


Figura 1: Arquitectura de los contenedores

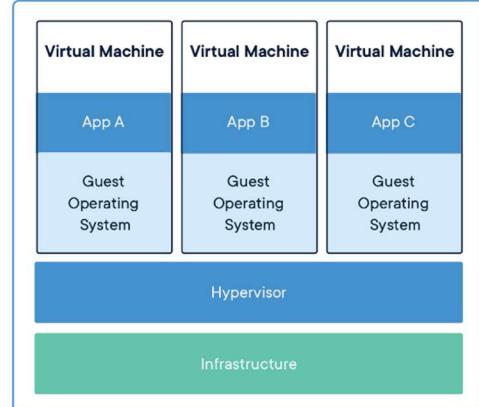


Figura 2: Arquitectura de las máquinas virtuales

Combinando las tecnologías de virtualización con redes y sistemas de gestión de recursos, se puede ofrecer al usuario una infinitud de servicios online.

Adaptándose a las necesidades de cada usuario, los recursos disponibles, y el esfuerzo que desee realizar en la gestión de infraestructura, se han implementado distintos modelos de servicios cloud. Se reparten en una escala en la que varía qué partes del sistema son gestionadas por el usuario y cuáles por el proveedor. Los principales modelos de servicios son [10]:

- *Infrastructure as a Service (IaaS)*: en este modelo, el proveedor gestiona la infraestructura física, ofreciendo al usuario una infraestructura virtual que deberá definir y gestionar.
- *Platform as a Service (PaaS)*: el proveedor gestiona la infraestructura física y virtual, ofreciendo al usuario la posibilidad de desplegar sus aplicaciones y datos.
- *Software as a Service (SaaS)*: el proveedor gestiona todas las capas de la aplicación y el usuario interactúa con esta a través de internet.

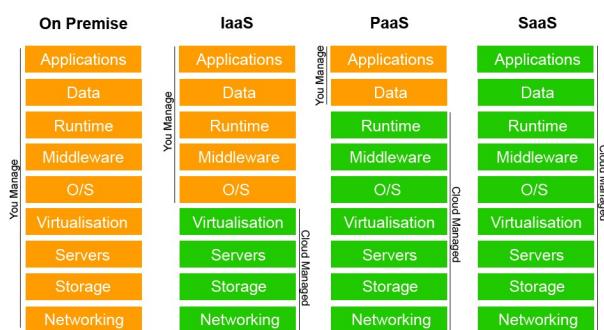


Figura 3: Tipos de servicios cloud según el dominio de las capas

## 2.2 Serverless y sus tecnologías

El primer ejemplo de tecnologías similares a Serverless se encuentra en Google App Engine (GAE). En el blog del lanzamiento de su versión de prueba, Paul McDonald escribía “*Google App Engine packages these building blocks and takes care of the infrastructure stack, leaving you more time to focus on writing code and improving your application [11].*” Se puede aquí observar una de las características principales del paradigma Serverless: la abstracción total de la infraestructura física. Por aquel entonces no se utilizaba todavía el término serverless y GAE no ofrecía todas las características de lo que hoy concebimos como Serverless, pudiendo considerarse un precursor.

No fue hasta 2014, con el lanzamiento de AWS Lambda, que se vislumbró por primera vez un producto comercial verdaderamente Serverless. En la entrada al blog de noticias AWS del lanzamiento de AWS Lambda podemos ver las características principales de los servicios Serverless [12]:

- Orientado a eventos: “*Lambda will automatically run code in response to modifications to objects uploaded to (...) buckets, messages arriving in (...) streams, or table updates (...)*”
- Abstracción de la infraestructura física: “*Lambda is a zero-administration compute platform.*”
- Pago por tiempo de computación: “*You pay for compute time in units of 100 milliseconds and you pay for each request.*”

A medida que se afianzó esta tecnología, surgió el término FaaS o *Function as a Service* que puede entenderse como un subconjunto de Serverless. Donde Serverless se puede definir como un modelo de computación en la nube en el que se abstracta la infraestructura física, delegando su gestión en el proveedor, y se paga por tiempo de computación, FaaS consiste específicamente en funciones serverless generalmente activadas por eventos externos o internos al entorno

cloud. Así, FaaS es un tipo de servicio Serverless, pero no todos los servicios Serverless son FaaS.

Los principales proveedores de servicios cloud, Amazon Web Services, Microsoft Azure, y Google Cloud, ofrecen todos sus respectivos productos Serverless: AWS Lambda, Azure Serverless Compute y Google Cloud Run. Sin embargo, para explicar el funcionamiento de este framework nos basaremos en OpenFaaS. Se trata de una solución de código abierto que ofrece servicios FaaS sin coste y usando tecnologías estandarizadas y libres. Aunque cada plataforma tenga matices y use tecnologías específicas, y quizás ligeramente distintas, el comportamiento y las tecnologías de todas son similares a grandes rasgos.

### 2.2.1 Flujo General

Para indagar en el funcionamiento de OpenFaaS, se explorará primero el flujo de trabajo general definido para una solicitud a una función.

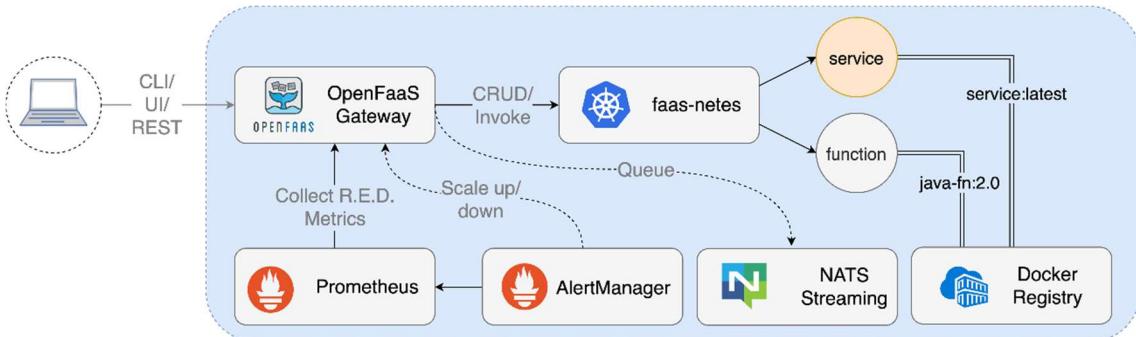


Figura 4: Flujo de las solicitudes en OpenFaaS

La función puede ser invocada por consola, interfaz gráfica, o solicitud HTTP. La dirección de destino de esta solicitud seguirá el siguiente formato [13]:

`http://<dominio>:<puerto>/function/<nombre>`

El dominio y puerto dependen de las preferencias del usuario. Para el dominio puede usarse un dominio DNS o una dirección IP. El puerto se define cuando se activa el redireccionamiento de puerto al entorno virtual ya que todos los elementos de la ilustración 1 contenidos en el recuadro azul se despliegan sobre un clúster de kubernetes, como veremos más adelante.

Cuando el host recibe la solicitud, lo hace mediante el Gateway API. Este Gateway es el encargado de enviar las invocaciones al operador o a la cola, exponer las métricas recolectadas por prometheus, ofrecer una interfaz gráfica, coordinar la gestión de funciones con el operador, y de manera opcional realiza tareas de balance de cargas [14].

El siguiente elemento en el flujo es el operador faas-netes. En el caso de la versión pro, este elemento se encarga de la gestión del ciclo de vida de las funciones, escalamiento, balance de cargas, etc. En definitiva, capacita el clúster para ofrecer FaaS. Sin embargo, en la Community Edition (CE) esta labor la realiza directamente el API Gateway [15, 16].

Los dos siguientes elementos son el servicio y la función. La función se encapsula en un POD de kubernetes que contiene el watchdog y la función. El watchdog permite abstraer la parte http del proceso y la función realiza el procesamiento en sí [17]. Su funcionamiento será desglosado con mayor detalle más adelante. Según la documentación oficial de OpenFaaS “*A Kubernetes Service object is also created and is used to access the function's HTTP endpoint on port 80, within the cluster* [13].”

Hasta ahora se ha examinado un flujo síncrono. Sin embargo, si en un flujo asíncrono, la solicitud se envía a la cola NATS en vez del operador o trabajador [18]. En este caso el Gateway serializa la solicitud y la inserta en la cola Nats. Otro elemento denominado queue worker se encarga de deserializarla y enviarla al trabajador.

Como se ha mencionado anteriormente, una de las responsabilidades del Gateway es la exposición de métricas. Estas métricas y las generadas por los distintos componentes son recolectadas por la instancia de Prometheus que se despliega de forma nativa con OpenFaaS CE, y expuestas en formato Prometheus [19].

El último elemento de la ilustración es el AlertManager. Este componente es un elemento de Prometheus que recibe alertas de la instancia Prometheus cuando se cumple la regla definida en el archivo `prometheus-cfg.yml`. En ese momento, el

AlertManager notifica al gateway, que se encarga de escalar el número de réplicas de la función para adaptarse a la demanda [20].

### *2.2.2 Seguridad en Serverless*

La arquitectura Serverless, subdivide un flujo de trabajo en tareas independientes, activadas mediante eventos y ejecutadas en entornos aislados. Este modelo difiere de otros modelos más tradicionales, como un desarrollo monolítico o incluso de microservicios, al introducir una mayor granularidad y aislamiento de los entornos [21]. A consecuencia, surgen nuevos retos de seguridad como por ejemplo el cruce de datos entre distintos entornos, securizados de manera independiente, o la gestión de la comunicación entre funciones que tradicionalmente realizarían los desarrolladores. Por ello, es necesario una reevaluación de las amenazas y paradigmas de seguridad que adapte las prácticas a estas nuevas necesidades.

#### *2.2.2.1 Modelo de Responsabilidad Compartida*

Como se ha explicado en secciones anteriores, los servicios cloud reparten la gestión de las distintas capas de una aplicación entre el proveedor de servicios y el usuario. Aplicando este modelo a la seguridad, surge el modelo de responsabilidad compartida, que establece que partes de la seguridad corresponden a la competencia del desarrollador o del proveedor.

En concreto, cada parte será responsable de la seguridad de las capas que gestione:

- Proveedor de servicios Cloud: son responsables de la infraestructura física y virtual, la red, el sistema operativo, la contenerización y los datos. De esta manera, serán responsables, entre otros aspectos, de la seguridad en las comunicaciones, los sistemas operativos e imágenes base usadas, las vulnerabilidades de sus APIs y portal, mantener el aislamiento entre entornos de ejecución...
- Desarrollador de la aplicación: será responsable de los aspectos relacionados con la especificación el entorno de ejecución, la

implementación aplicación, la gestión de permisos y el acceso de los usuarios...

Solo se podrá obtener una aplicación segura si ambos agentes aplican buenas prácticas en sus respectivas áreas de responsabilidad.

#### 2.2.2.2 Tipología de amenazas

Las amenazas a un servicio serverless pueden clasificarse según la etapa del ciclo de vida en las que son pertinentes o según qué parte del contrato sea responsable. Surgen entonces tres categorías:

- Amenazas al desarrollador en la configuración del servicio: abarcan aquellas tareas realizadas en la fase de desarrollo de la aplicación, incluyen permisos, acceso a eventos, y privilegios de control demasiado amplios; vulnerabilidades en servicios cloud asociados, repositorios, imágenes, y dependencias vulnerables; ataques a las herramientas de integración y despliegue continuo.
- Amenazas al desarrollador en el despliegue: son aquellas amenazas aplicables a la etapa de despliegue y operación de la aplicación cuya responsabilidad recae sobre el desarrollador. Entre ellas se encuentran la inyección de datos, fugas de contexto global, autenticación rota o insegura, agotamiento de recursos, e inseguridad en el logeo, monitorización y gestión de secretos.
- Amenazas al proveedor en el despliegue: amenazas en *runtime*, responsabilidad del proveedor. Abarcan las vulnerabilidades del entorno de ejecución, fugas entre unidades invocables, corrección de los servicios serverless, y vulnerabilidades de la API, portal y consola.

#### 2.2.2.3 Prácticas Recomendadas

Debido a que por el momento no existen técnicas de defensa *a posteriori* es de vital importancia que el desarrollo y configuración de las funciones Serverless se realice con el máximo cuidado para minimizar los riesgos.

AWS, en su whitepaper “AWS Well-Architected Framework” enumera una serie de buenas prácticas que incluyen:

- El uso de sistemas de gestión de identidad y acceso.
- El uso de controles para detectar vulnerabilidades en el código, las librerías, y lo entornos seleccionados.
- Protección de datos: cifrado de datos sensibles en tránsito, realizar validación de solicitudes, No exponer directamente las URL de los servicios Serverless
- Protección de la infraestructura: establecer límites de red a nivel de aplicación, uso de proxis, y uso de credenciales temporales cuando una función Serverless requiera de un servicio externo.
- Monitorización y logueo: por razones tanto legales como de detección de ataques.

## 2.3 Estado del arte

En esta sección se pretender revisar la literatura científica actual sobre ataques DoW, para establecer una definición concreta y obtener información sobre su tipología, motivaciones, y efectos.

Para garantizar la calidad de la información, se ha optado por la revisión de artículos recientes, posteriores a 2021, que hayan sido publicados por fuentes fiables: IEEE, ACM, Springer Nature, ScienceDirect, y Oxford University Press.

### 2.3.1 Definición de Denial of Wallet

Comenzaremos primeramente por la definición del ataque *Denial of Wallet* analizando las distintas definiciones propuestas por los investigadores.

En su artículo “Denial of wallet—Defining a looming threat to serverless computing”, Kelly et al., 2021, definen un ataque *DoW* como “*(...) the intentional mass, and continual, invocation of serverless functions, resulting in financial*

*exhaustion of the victim in the form of inflated usage bills [22].*" Esta definición es citada por la mayoría de los artículos examinados.

Solo Sehn et al., 2022, en su artículo "*Gringotts: Fast and Accurate Internal Denial-of-Wallet Detection for Serverless Computing*" [23] proponen una definición distinta. En esta, los ataques DoW se pueden entender como un subtipo de ataques DoS, ejecutados sobre plataformas Serverless. Las diferencias que destacan son tres:

1. Los ataques DoS buscan exceder los límites de la infraestructura, mientras que DoW se centra en explotar el modelo *Pay-as-you-go* de Serverless.
2. DoS resulta en la inaccesibilidad del servicio, mientras que DoW mantiene el servicio disponible para el resto de los usuarios.
3. Las consecuencias para DoS incluyen daños a la reputación, interrupción del servicio y pérdida de clientes, mientras que DoW solamente genera un sobrecoste para la víctima.

Para el resto del trabajo, se tomará la primera definición para DoW al condensar de manera concisa y clara las características del ataque, expuestas también por el resto de los estudios.

### *2.3.2 Tipología de ataques DoW*

En su artículo, Shen et al. [23], proponen la existencia de dos tipos de ataques:

- Externos, donde el ataque se realiza invocando directa o indirectamente la API de los servicios.
- Internos, donde se busca desplegar software malicioso en las máquinas físicas y virtuales usadas por la víctima, ralentizando la ejecución de los servicios de la víctima.

Por otra parte, Kelly et al. [22], se centran en estrategias de ataque que entran en la categoría de ataques externos. Los distintos métodos que proponen incluyen:

1. Método tradicional: envío de solicitudes con altas cargas de trabajo mediante métodos como REDoS o Bilion-Laugh, donde se explotan el

funcionamiento de las expresiones regulares y los parseadores xml respectivamente para generar cargas de trabajo elevadas.

2. Ataques de corta duración: consisten en el envío masivo de solicitudes.
3. Ataques de larga duración (*leech* o sanguijuela): consisten en el envío de solicitudes espaciadas en el tiempo de tal forma que pasen desapercibidas por el firewall.
4. Denial of Wallet Distribuido: utilizan una red de bots voluntarios o involuntarios para realizar el ataque, dificultando su detección.
5. Decepción de sistemas de mitigación adaptativos: cuando las reglas de firewall son modificadas según el tráfico medido, el atacante puede aumentar la intensidad del ataque progresivamente para establecer bases falsas.
6. Explotación Serverless: consiste en la ejecución de los servicios Serverless mediante otros triggers, como el almacenamiento.
7. Usuarios Falsos: creación de usuarios falsos e invocación de los endpoints mediante herramientas de automatización de navegación.

En su artículo sobre Ataques DoW Distribuidos, Mileski y Mihajloska indagan en el funcionamiento de varios tipos de ataques [24]. El primero “*Blast DDoS*” consiste en el envío masivo de solicitudes en un momento puntual, que provoca además de DoW el fallo del servicio (DoS), y el segundo “*Continuous Unconspicuous DDoS*” que consiste en el envío de menor volumen de solicitudes en un mayor tiempo. Además, profundizan también en el ataque “*Background Chained DDoW*” que utiliza automatizadores de navegación para provocar el envío de solicitudes cuando la URL de los servicios no es accesible al atacante.

#### 2.3.3 *Detección de ataques*

De los tipos de ataques mencionados hasta ahora, el de sanguijuela y el de decepción de sistemas de mitigación adaptativos presentan grandes dificultades para su detección. El cambio en el tráfico resulta imperceptible para la víctima y elude los mecanismos de defensa tradicionales.

En su estudio “DoWNet—classification of Denial-of-Wallet attacks on serverless application traffic”, Kelly et al. (2024) [25] utilizaron una representación gráfica del tráfico mediante cuadrículas y una red neuronal convolucional que clasificaba las imágenes para determinar si existía o no un ataque, y determinar el tipo de ataque.

El equipo de investigadores consiguió una precisión y exactitud del 97%. A pesar de las limitaciones de tratarse de ataques simulados, no basados en datos históricos al no existir ejemplos de ataques, su sistema permite la detección de estos ataques, y su simplicidad permite ser desplegado en entornos Serverless para acompañar los servicios que monitoriza.

Por otro lado, Shen et al. (2022) [23] diseñaron un sistema de detección de ataques DoW externos llamado Gringotts. Se basa en dos elementos: un agente, que recopila métricas de ejecución por solicitud, y un programa de diagnóstico. Este último realiza la detección de los ataques basándose en la premisa que, una misma carga de trabajo requiere un mismo tiempo de ejecución: por tanto, si una función está siendo atacada de manera externa, para esa carga de trabajo, el tiempo de ejecución será mayor. Con este sistema, consiguieron una precisión de más del 90%.

#### *2.3.4 Simulación*

Para la simulación de ataques, existen diversos métodos. En su primer estudio entorno a los ataques DoW [22], Kelly et al., utilizan el framework OpenFaaS que ya se ha explicado en las secciones anteriores. La principal ventaja de su uso es que, al tratar de simular un ataque DoW, el uso de una plataforma Serverless comercial implicaría un coste desorbitado, si no existe algún tipo de colaboración con el ofertante del servicio. Al tratarse de infraestructura física propia, los investigadores tienen a su disposición un control sobre la misma mucho mayor del que tendría un cliente real, que les ofrece además mayor capacidad de diagnóstico y recopilación de datos.

Además, el equipo desarrolló un emulador de coste que utiliza los datos ofertados por OpenFaaS y la información de facturación de las principales plataformas

Serverless, para determinar el precio incurrido por la víctima. Se trata de un simple programa de Python, que utiliza solicitudes http para obtener los datos necesarios y calcular los precios.

En su segundo estudio, DoWTS – Denial-of-Wallet Test Simulator [26], Kelly et al. (2022) desarrollaron una herramienta de simulación de ataques DoW que también permite generar *datasets* para el estudio de estos. Este simulador utiliza datos introducidos por el usuario como el número de usuarios reales del servicio, numero de bots, el intervalo de tiempo, etc., para simular el tráfico real del servicio y el generado por el atacante. Almacena en logs datos relevantes como las IPs simuladas de los usuarios y bots, los momentos en los que se reciben las solicitudes...

Este simulador permite obtener una aproximación de cómo podría ser un ataque DoW, teniendo en cuenta los patrones de uso de una web de e-commerce, que fue usada como ejemplo, y distintos patrones de ataque incluyendo un flujo constante, exponencial, y aleatorio. Los datasets generados pueden resultar de gran utilidad para estudios preliminares, aunque resulta evidente la necesidad de entornos que permitan la emulación de estos ataques.

### 3. Metodología

Como se ha mencionado en secciones anteriores, el objetivo del proyecto es el desarrollo de defensas para las distintas estrategias de ataques DoW en un entorno Serverless, y la evaluación de su efectividad. Para esto, se debe definir una metodología rigurosa que permita definir los métodos de desarrollo y criterios de evaluación.

Debido a la existencia de múltiples estrategias de ataque, para las que se deben adoptar distintas estrategias de defensa, se pretende seguir un modelo de desarrollo iterativo. Cada iteración será independiente, y el producto de cada una de ellas incluirá una estrategia de defensa implementada, junto con los datos y gráficos obtenidos de la evaluación.

Las distintas etapas de cada iteración serán:

1. Implementación del ataque: primeramente, se implementará la estrategia de ataque para poder realizarlo contra un entorno Serverless.
2. Ejecución del ataque: se ejecutará el ataque desarrollado, recopilando los datos relevantes para determinar los daños.
3. Análisis de datos: se procesarán los datos, se analizarán y se determinarán los daños causados, usando como principal métrica de daño el coste incurrido por la víctima.
4. Desarrollo de la defensa: se investigará y desarrollará una estrategia de defensa adaptada al ataque.
5. Ejecución del ataque con defensa: se ejecutará el mismo ataque ejecutado en la segunda fase de la iteración, ahora con la defensa implementada y desplegada, recopilando también los mismos datos.
6. Análisis de datos y conclusiones: se procesarán los datos obtenidos, comparándolos con los datos del ataque inicial, obteniendo así una comparativa. Se determinará la efectividad de la defensa mediante una comparación del coste a la víctima.

7. Limitaciones: se analizarán las limitaciones de las técnicas utilizadas, los sistemas desplegados, las simplificaciones aplicadas, y sus efectos sobre la validez de estas técnicas en entornos reales.

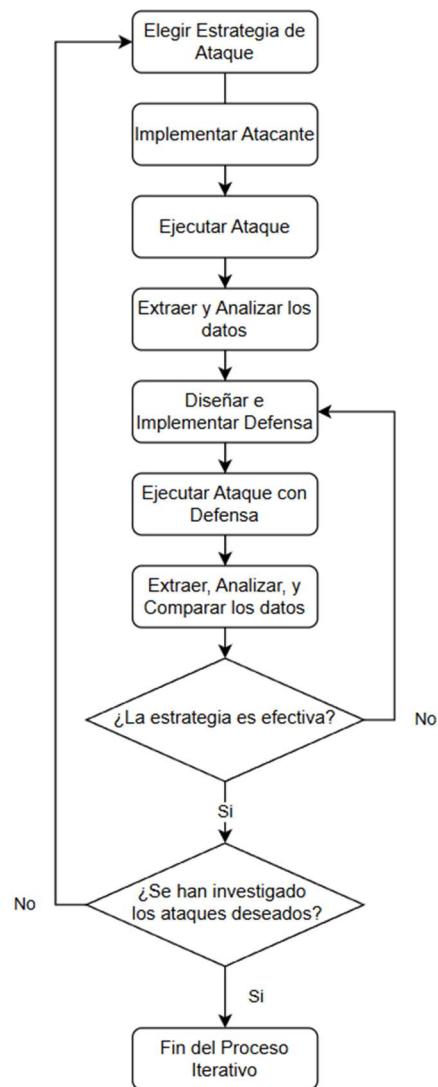


Figura 5: Flujo de la metodología

## 4. Arquitectura

Para el desarrollo de los ataques y defensas, es necesario desplegar una o varias funciones en un dispositivo que aloje un entorno Serverless, que actúe de víctima. Se deberá configurar un dispositivo atacante, en el que implementar las distintas estrategias, y una red que los conecte.

### 4.1 Víctima

Para configurar el entorno Serverless se ha elegido OpenFaaS, cuyo funcionamiento ya se ha explicado de manera detallada en el apartado 2. El uso de una solución de código abierto, desplegada en un dispositivo propio, otorga una serie de ventajas:

- Reducción de coste: al buscar una emulación fiel de ataques DoW, cuyo objetivo es agotamiento de los recursos financieros de la víctima, los ataques conllevarían un coste desorbitado.
- Configuración personalizada: en un dispositivo propio es posible modificar y extender el funcionamiento del entorno Serverless, al tener acceso a capas que normalmente son gestionadas por el proveedor cloud. Esto es también necesario para el desarrollo de estrategias de defensa, ya que es posible que sea necesario gestionar el tráfico a nivel de red.
- Compatibilidad: el uso de una tecnología de código abierto, diseñada para su uso en plataformas estandarizadas como Kubernetes, permite extender fácilmente el funcionamiento del sistema, añadiendo, cuando sea necesario, distintos módulos.

OpenFaaS se ha desplegado en un nodo Kubernetes, en concreto la instancia de Kubernetes incluida con la instalación de Docker Desktop, debido a su facilidad de instalación y la interfaz gráfica que permite monitorizar fácilmente el estado de los pods, para posteriormente realizar un diagnóstico más detallado. Este nodo se instaló en un ordenador personal, conectado por ethernet a la red local, para reducir la latencia en las comunicaciones entre víctima y atacante.

Para la instalación de OpenFaaS se han seguido las instrucciones proporcionadas en la web oficial [27]. Tras instalar la herramienta de instalación Arkade, se usó la misma para desplegar OpenFaaS en el nodo Kubernetes.

Una vez desplegado, se procedió a desarrollar la función contra la que se realizarán los ataques. La función usa la plantilla para Java de OpenFaaS, que define una clase Handler.java, con un constructor y un método handle. Al ser una función bastante simple, no fue necesario implementar ninguna clase más.

La función ReducirlImagen es una simple función de redimensionamiento de imágenes. La función acepta como input una solicitud http cuyo cuerpo tiene en formato JSON un elemento {"image": "byte[]"} donde el valor de byte[] es un array de los bytes que conforman la imagen original. Como propiedades de la cabecera de la solicitud se han incluido el formato del archivo original, el tamaño en bytes, las dimensiones en pixeles, y el tipo de contenido “application/json”.

En cuanto a la respuesta, contiene simplemente el código de respuesta, y en el cuerpo la imagen procesada seguida del tiempo de procesamiento. Estos documentos http se pueden representar de la siguiente forma:

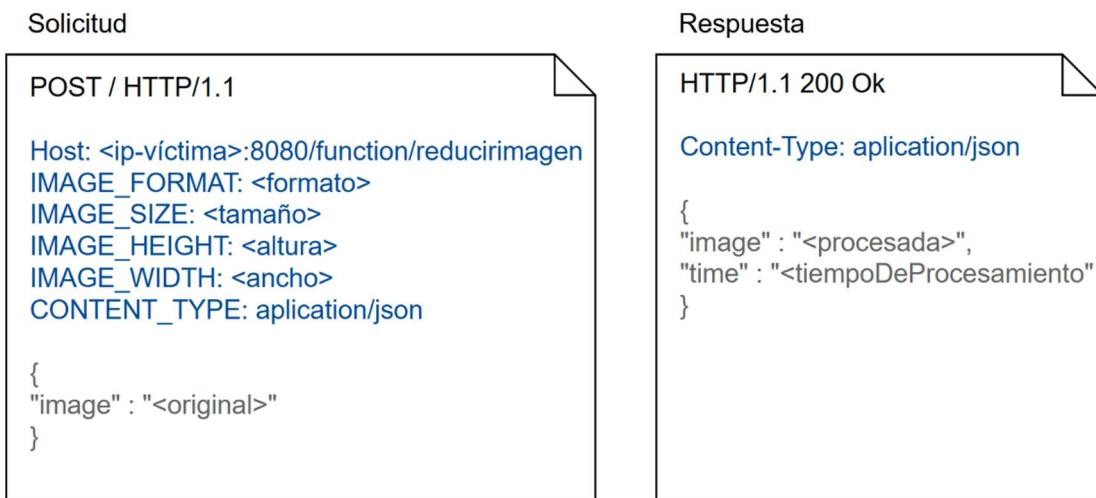


Figura 6: Formato de las solicitudes HTTP

En la función se ha utilizado la biblioteca jackson de fasterxml para la serialización y deserialización del objeto JSON del cuerpo de la solicitud. Su dependencia se

definió en el archivo build.gradle al ser este el que OpenFaaS utiliza para definir las bibliotecas que se deben empaquetar en el entorno de ejecución de la función.

Para desplegar la función, es necesario primero construir la imagen y subirla al repositorio de DockerHub usando el comando openfaas up –f /reducirImagen.yml. El despliegue se hizo mediante la interfaz gráfica, añadiendo como parámetro de gateway la dirección IP de la víctima seguida del puerto.

Figura 7: Configuración de despliegue de la función

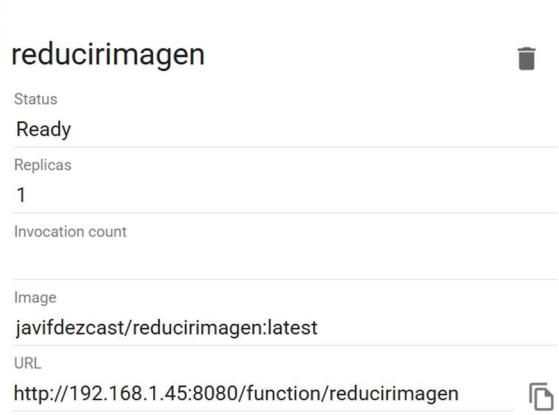


Figura 8: Panel de estatus de la función

Tras el correcto despliegue de la función y debido a las limitaciones en recolección de datos impuestas por la versión Community Edition de OpenFaaS, que solo permiten la recopilación de métricas muy limitadas, se desplegó una nueva instancia de prometheus para recolectar más métricas. Se utilizó el helm chart oficial de Prometheus Community [28], para desplegar una instancia de prometheus simple que incluyera además un servidor kube-state-metrics, para recolectar métricas relevantes del nodo kubernetes como el consumo de CPU, memoria, etc. Fue necesario modificar el helm chart para establecer el valor de la tasa de recolección de las métricas, y definir la federación de prometheus que permitiera obtener los datos de la instancia nativa de OpenFaaS.

Para la consulta de los datos y monitorización se desplegó una instancia de Grafana. Para ello se obtuvo, mediante helm, el repositorio grafana/grafana y se instaló my-grafana en el namespace monitoring. Se definió como fuente de los datos el servidor prometheus, usando la URL de su endpoint. Una vez configurado se definieron tres variables para su uso en consultas: el nombre de la función, el

namespace de los pods, y el nombre del contenedor. Estas variables permiten visualizar únicamente las métricas de los pods de la función reducir imagen, excluyendo los pods de openfaas, prometheus, y otros pods del clúster. Por último, se creó un dashboard con cuatro paneles con las siguientes métricas:

- Número de réplicas de la función.
- Tiempo medio de procesamiento en los últimos 5s.
- Consumo de memoria por réplica.
- Consumo de CPU por réplica.

El dashboard resultante fue el siguiente:

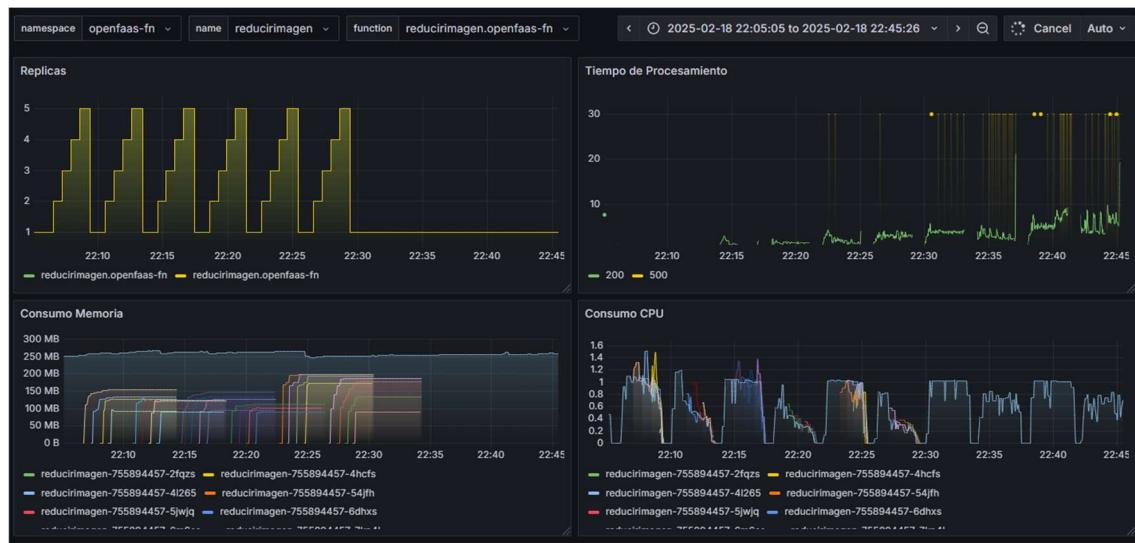


Figura 9: Panel de monitorización del nodo kubernetes

Así los elementos relevantes de la arquitectura de la víctima se pueden representar en el siguiente diagrama.

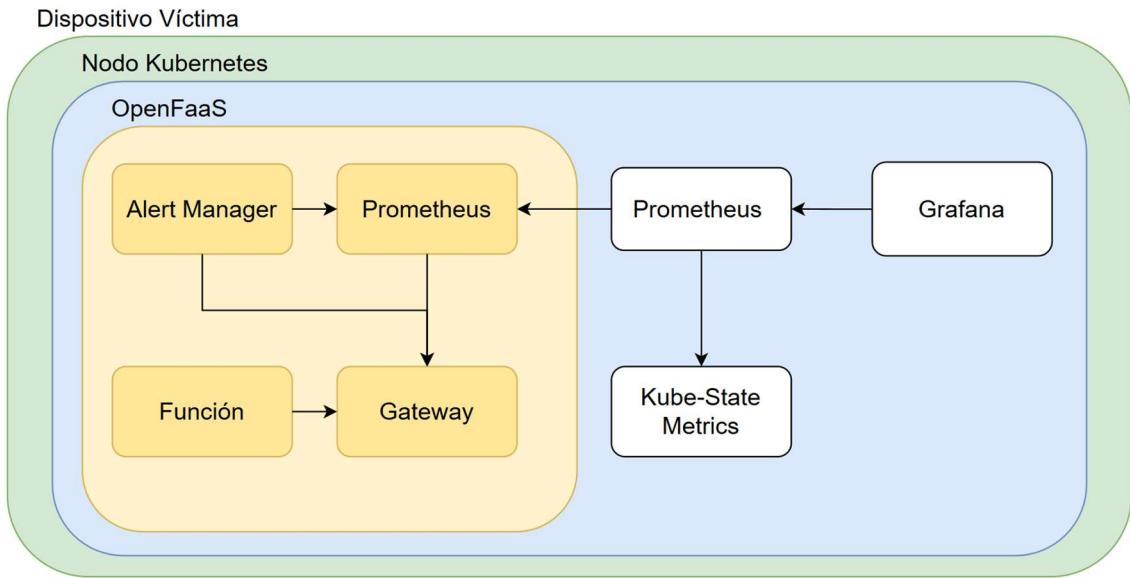


Figura 10: Arquitectura de los servicios del dispositivo víctima

Para exponer el servicio gateway, y grafana, se usó redireccionamiento de puertos al ser la opción más simple y no influir de manera notable en el tiempo de recepción de las solicitudes.

## 4.2 Atacante

La arquitectura del atacante resulta mucho más simple al tratarse únicamente de un programa que envía solicitudes a la víctima. Sin embargo, para cada estrategia de ataque debe implementarse un atacante distinto. Para que los resultados tengan una mayor fiabilidad, los ataques se realizarán un numero de iteraciones. De esta manera, se implementarán varias clases Experimento que usarán la clase Atacante correspondiente para ejecutar los ataques de manera iterativa. La jerarquía de clases e interfaces es la siguiente.

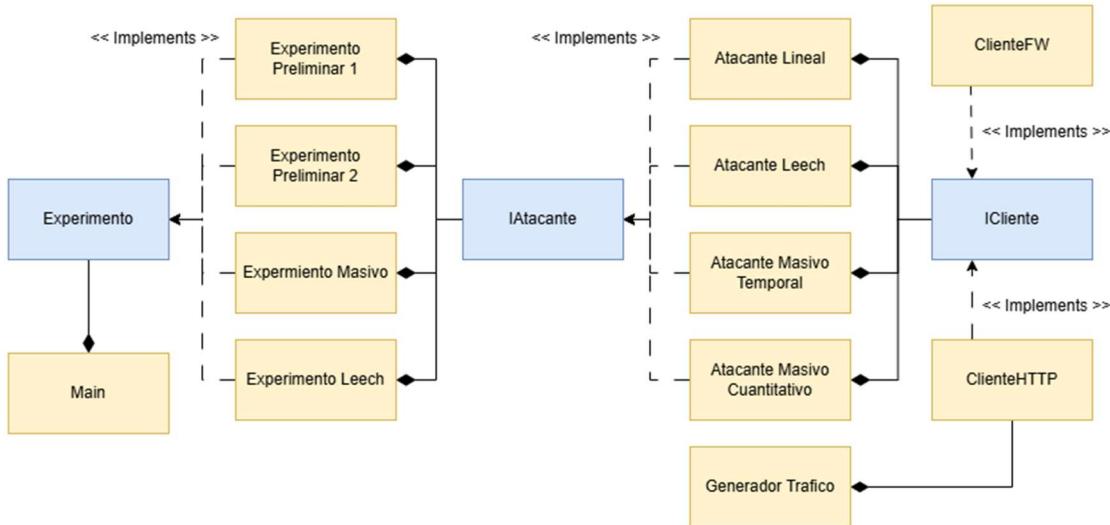


Figura 11: Diagrama de clases del atacante

Por razones de conocimiento, velocidad, y librerías disponibles, se ha elegido Java como lenguaje para la implementación de los atacantes. Usando polimorfismo y clases abstractas, se pretende que cada atacante se implemente en una clase distinta pero que todas implementen una misma interfaz, facilitando así la programación de los métodos principales que invocarán las subclases.

Debido a que partes del comportamiento de las clases es compartido, como por ejemplo el envío de solicitudes, establecer la conexión, recibir la respuesta, estos métodos serán implementados en una clase a la que tendrán acceso por composición. De esta forma se evita la repetición del código y se facilita el aislamiento de errores y el testeо.

La interfaz `ICliente` contiene un único método `String[] solicitudRespuesta()` que en su implementación enviará la solicitud http, procesando la respuesta, y devolviendo un array que contiene datos como el tiempo de procesamiento, el momento de envío y recepción del par solicitud-respuesta, y el código de respuesta.

La interfaz `IAtacante` tiene un único método `List<String[]> atacar()` que utiliza el cliente para realizar envíos. Según el tipo de atacante cambia el número de solicitudes enviadas, la duración del ataque, la velocidad de envío, etc.

Por último, la interfaz IExperimento tiene un único método ejecutar() que realiza el ataque, variando su implementación según el tipo de ataque que se realice, y el número de iteraciones.

Para almacenar los valores de variables constantes que se usan en múltiples métodos y clases se creó una clase Constantes. Incluye valores como el URL al que se envían las solicitudes o los nombres de los archivos usados para los ataques.

### 4.3 Trafico Base

Para simular un entorno más realista, se decidió añadir un tercer dispositivo que generase un tráfico base que emulase tráfico no malicioso. Esto permitirá garantizar que las estrategias de defensa solo afectan al tráfico malicioso, y demostrar que la función sigue en pie y accesible tras el despliegue de dichas defensas, además de generar condiciones más verosímiles.

Para su implementación se reutilizó código ya desarrollado como el Cliente HTTP y su interfaz. El funcionamiento del generador de tráfico básico es muy simple: envía una solicitud http con la imagen en el mismo formato y tamaño que el que se haya utilizado para el experimento, y esperar 10s para mandar la siguiente solicitud. Este bucle se ejecuta durante la duración del experimento, terminando por finalización del tiempo establecido o por acción propia.

### 4.4 Red

Finalmente, se analizará la configuración de red usada para conectar ambos dispositivos. Ambos están conectados a una red LAN doméstica mediante ethernet para reducir la latencia en las comunicaciones. La topología de la red es la siguiente: los dispositivos se conectan a un switch en configuración de estrella, y este a su vez se conecta con el router. Gracias a los enlaces full dúplex, no se generan conflictos entre envío y recepción de paquetes, sobre todo en el enlace entre el switch y el router.

Al ser una red doméstica, donde existe tráfico ajeno al sistema, se ha configurado una VLAN de forma que tanto los dispositivos de la arquitectura propuesta como los externos permanezcan aislados mutuamente.

Como se especifica en la documentación de OpenFaaS, el Gateway utiliza por defecto el puerto 8080. Por ello, ha sido necesario, tanto en el atacante como en la víctima, crear reglas de tráfico en el firewall que permitan la comunicación bidireccional en estos puertos. Se ha aplicado el principio de mínima permisividad, para que se permita únicamente el tráfico necesario, limitándolo además a tráfico interno a la LAN. Así evitamos que agentes malintencionados aprovechen esta apertura para infiltrarse en la red LAN o en los dispositivos.

El switch es un modelo OvisLink Live-FSH8SB. Se trata de un switch de uso doméstico con 8 puertos ethernet de doble velocidad 10/100 Mbps, auto MDI/MDI-X, una tabla de direcciones MAC de 1000 entradas y un buffer integrado de 768Kbit.

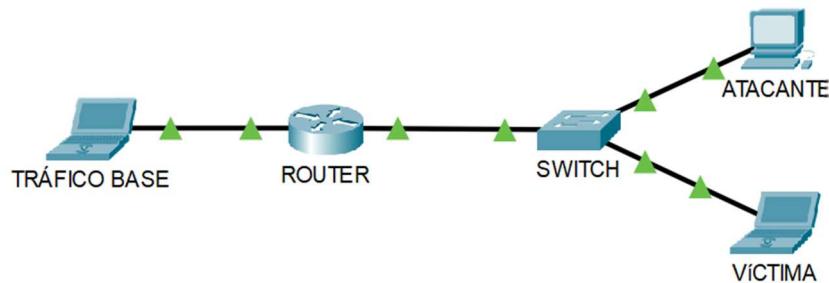


Figura 12: Arquitectura de la red

## 5. Desarrollo

En esta sección se incluirán los experimentos preliminares y las iteraciones de desarrollo de defensas.

### 5.1 Tiempo de procesamiento.

Para comprobar el correcto funcionamiento del sistema desplegado, se realizó un experimento simple. Se envió una misma imagen, escalada a tamaños que corresponden con incrementos de 10% del tamaño original (10%-100%) para que fuera procesada por la función. La imagen original tiene un tamaño de 2000 x 3000 pixeles. Para cada tamaño de la imagen se probó con formato JPG y PNG. En cada iteración se realizaron 31 procesamientos para poder obtener medias significativas. Se anotaron en un archivo CSV los momentos de envío de la solicitud y de respuesta para determinar el tiempo de procesamiento desde el cliente, y se procesaron los datos usando MatLab obteniendo los siguientes gráficos:

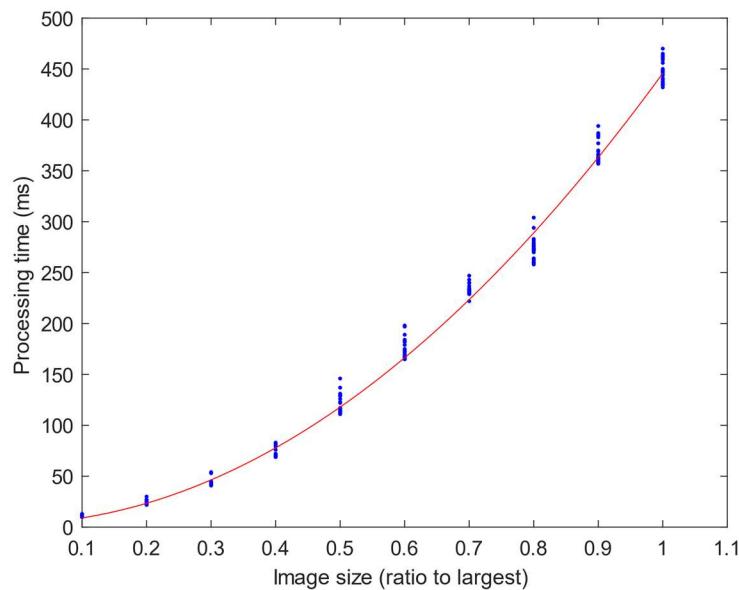


Figura 13: Tiempo de procesamiento en función del tamaño de la imagen en formato PNG

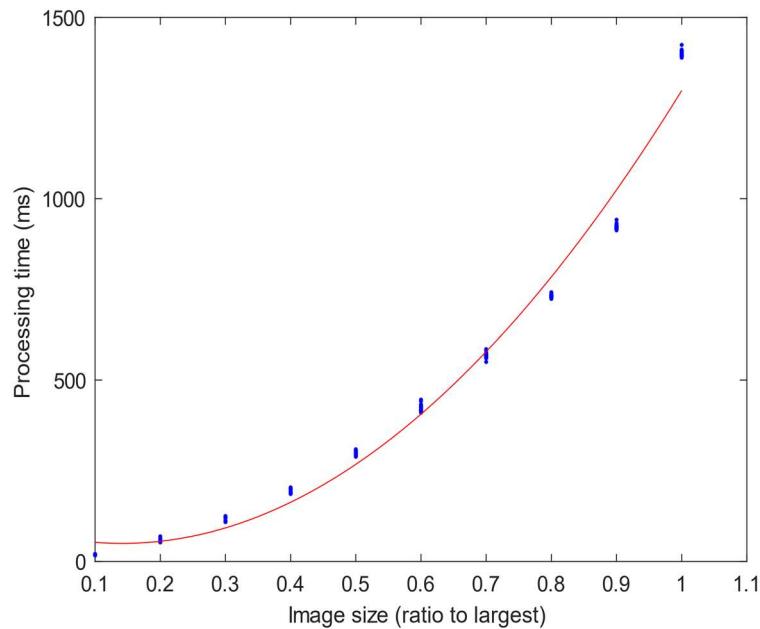


Figura 13: Tiempo de procesamiento en función del tamaño de la imagen en formato JPG

Para ambos formatos la relación entre el tamaño de la imagen y el tiempo de procesamiento parece seguir una relación cuadrática, notándose un tiempo de procesamiento significativamente mayor para el formato JPG.

Las ecuaciones cuadráticas encontradas para el tiempo de procesamiento de las imágenes png y jpg respectivamente, con un coeficiente de confianza del 95% son:

$$f(x) = 426,4x^2 + 16,31x + 3,149 \quad f(x) = 1692x^2 - 477,6x + 83,39$$

Con este experimento se puede comprobar que la función reducir imagen funciona de forma esperada, y que la configuración de la víctima, el atacante, y la red, son correctas.

## 5.2 Tamaño y formato óptimo

Antes de comenzar con los experimentos reales, es necesario determinar qué tamaños son mejores para sobrecargar la función. Se busca encontrar un equilibrio entre una carga de trabajo alta y una tasa de envío de solicitudes alta.

Para encontrar que tamaño de imagen y formato alcanzan dicho equilibrio se realizó el siguiente experimento. De nuevo con la misma imagen escalada en intervalos de 10% y en formatos JPG y PNG, se realizaron ataques “masivos”, en los que se envían tantas solicitudes como sea posible.

Los ataques duraron 3 minutos por tamaño y formato, incluyendo además un minuto de *cooldown* entre ataques para que el gateway eliminase las réplicas existentes y volviera a un estado base. Para cada combinación tamaño-formato se creó un ThreadPool con un numero variable de hilos, que ejecutaban cada uno un ataque masivo.

Se define una iteración como un ciclo en el que se realiza un ataque de 3+1 minutos por cada tamaño y formato. Se realizaron un total de 7 iteraciones para que los resultados no se vean afectados por condiciones externas como el estado de red, u otras variables.

En el dispositivo víctima se recolectaron mediante Grafana las métricas de tiempo medio de procesamiento, número de réplicas de la función, consumo de memoria y consumo de CPU. En el atacante se registró en un archivo de texto el momento en el que empieza el ataque para cada pareja formato-tamaño. Con esto, se pudo agregar los datos en función del tamaño y formato de la imagen, obteniendo datos medios con mayor significancia.

En ejecuciones preliminares del experimento se observó que a partir del 50% del tamaño de la imagen original, no se conseguía que la función se replicara. Al ser esto uno de los principales objetivos, en el caso de un ataque masivo, se optó por utilizar solo los tamaños que si conseguían replicas: 10 – 60 % del tamaño original.

Debido a que el tiempo de procesamiento los tamaños es distinto, el número de hilos tuvo que ser ajustado para que no hubiera problemas con la conexión http debido a posibles timeouts. En la siguiente Tabla se puede observar el número de hilos para cada tamaño:

Tamaño con respecto a la original (%)	10	20	30	40	50	60
Número de hilos	30	30	30	30	30	15

*Tabla 1: hilos utilizados en la generación de tráfico según el tamaño de la imagen*

Tras realizar, el experimento, se extrajeron los datos de Grafana en formato CSV, se procesaron, y agregaron para generar gráficos en MatLab. Por fallos de prometheus en la recolección de métricas, hay momentos cuyos datos no se grabaron. Para estos fallos, se tomó como valor la media del valor anterior y el siguiente.

En los gráficos resultantes, para aquellos casos en los que la función generaba varias replicas, se utilizó el mismo color de línea, pero distinto patrón para poder reconocer la serie como parte de este par formato-tamaño, y mejorar la legibilidad.

Para las réplicas de la función, el comportamiento es muy similar en ambos casos. Los tamaños 0, 1, y 2 consiguen replicar la función siempre siguiendo el mismo patrón. En concreto, los tamaños 0 y 1 se replican en el mismo momento por lo que su línea está solapada en la gráfica y no se aprecia con claridad.

Debido a que la métrica que utiliza OpenFaaS CE para duplicar la función es el número de solicitudes por segundo, y que el tiempo de procesamiento para mayores tamaños es mayor, no es posible alcanzar el threshold en el procesamiento de la imagen JPD para los tamaños mayores de 2. En el caso del tamaño 3 en formato png, si se logra duplicación de la función, pero no en todas las iteraciones del experimento. Por esto se ven números decimales para una variable que en principio debería ser entera.

Replicas de la Funcion para Distintos Tamaños de Imagen

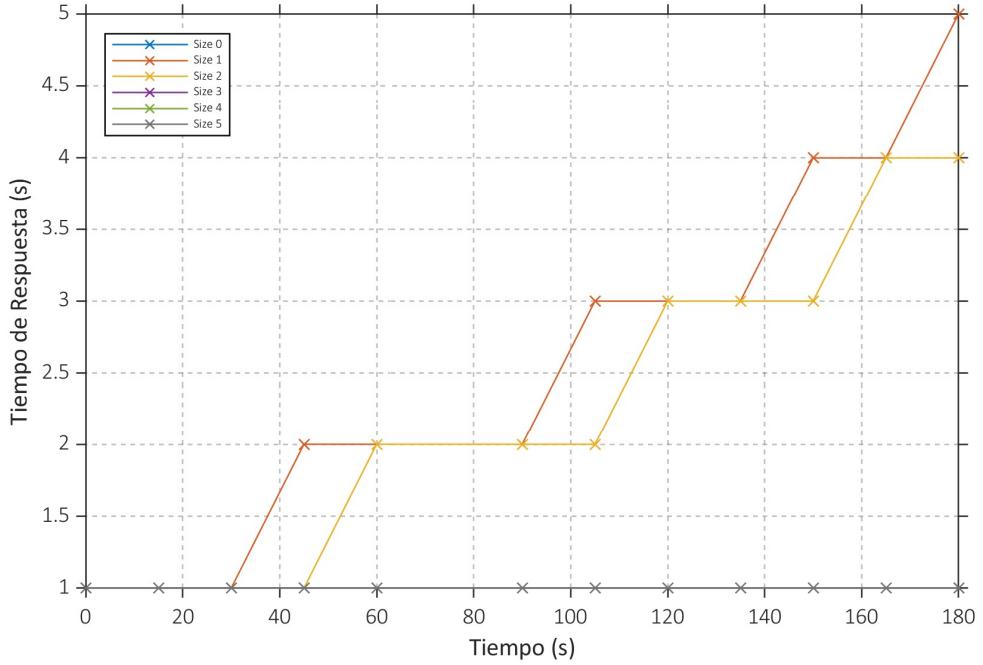


Figura 15: Replicas de la función durante el procesamiento de imágenes de distintos tamaños en formato JPG.

Replicas de la Funcion para Distintos Tamaños de Imagen

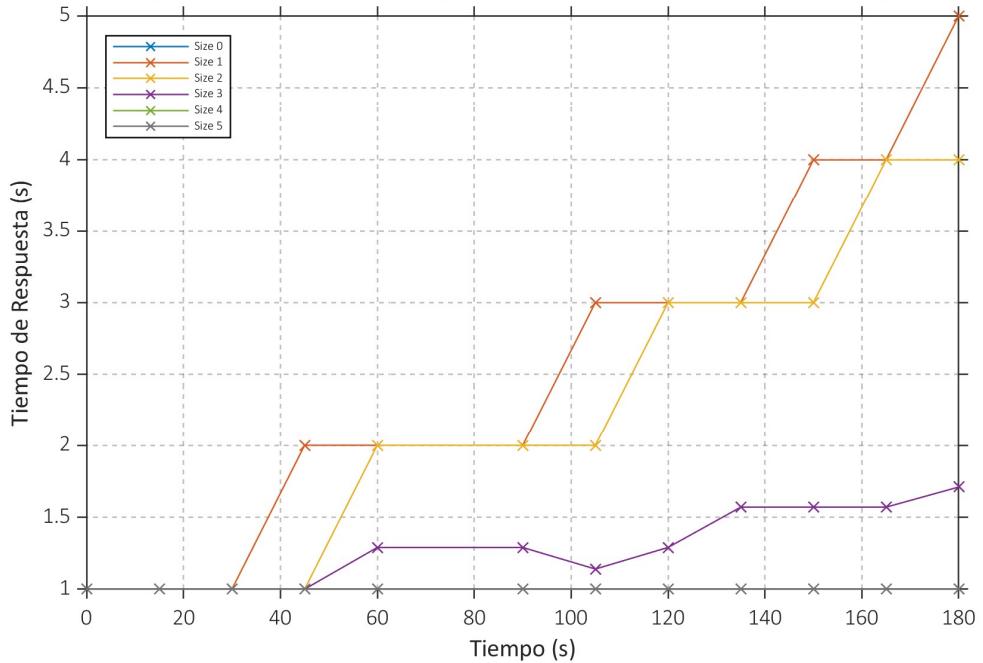


Figura 16: Replicas de la función durante el procesamiento de imágenes de distintos tamaños en formato PNG.

Los gráficos de consumo de CPU por réplica muestran una clara diferencia en la intensidad de la operación de redimensionamiento entre formatos. Mientras que en el formato JPG las réplicas oscilan entorno a un consumo del 1%, las réplicas para el formato png no suelen superar el 0,8%. En el gráfico del formato .png se observa un comportamiento distinto para los tamaños que provocan la duplicación de la función: a medida que aumenta el tiempo, y el número de replicas, disminuye el porcentaje del consumo de CPU. Aunque no se ha indagado en el origen de este comportamiento podría explicarse por qué una misma carga de trabajo se divide entre las distintas réplicas.

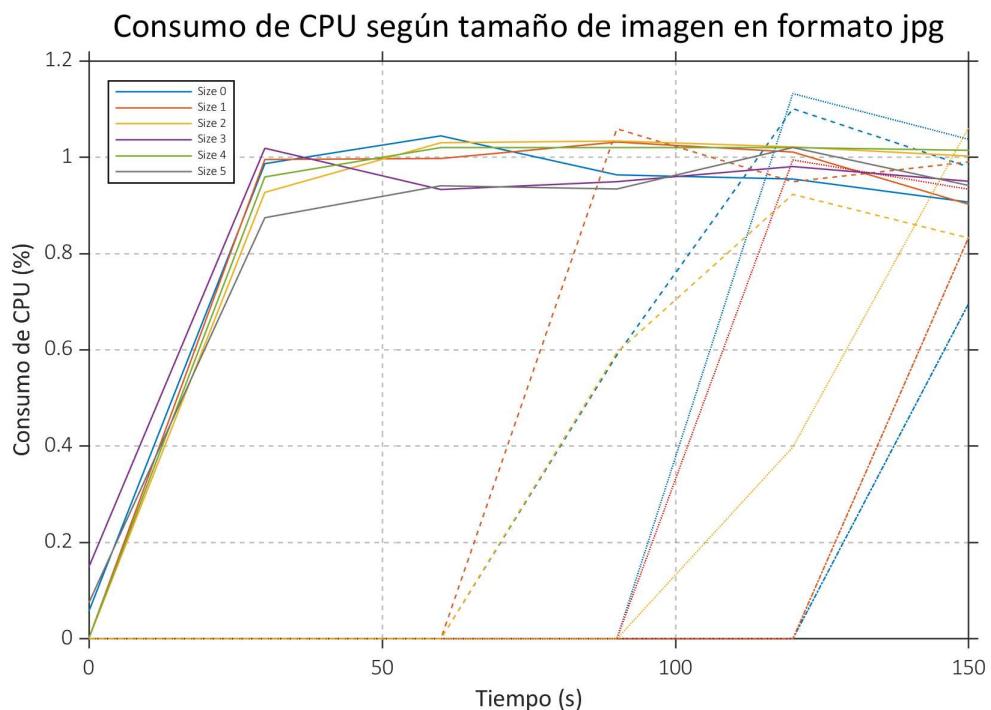


Figura 17: Consumo de CPU por réplica durante el procesamiento de imágenes en formato JPG.

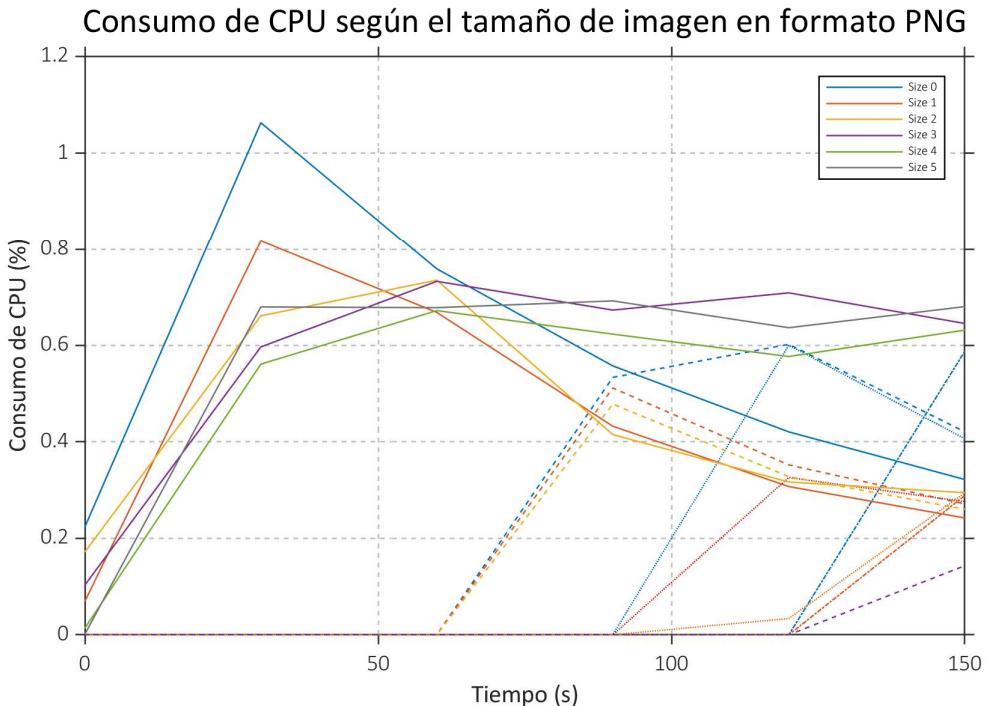


Figura 18: Consumo de CPU por réplica durante el procesamiento de imágenes en formato PNG.

En el caso del consumo de memoria por réplica, se optó por el uso de dos ejes. El derecho para la réplica principal de la función, cuyo consumo de memoria se mantuvo estable en torno a los 240MB, y el izquierdo para las réplicas secundarias, representadas con distintos tipos de líneas discontinuas, ya que su rango de valores era mucho mayor.

A diferencia del consumo de CPU, no se observa un comportamiento distinto entre formatos, ni entre tamaños de imagen. Todas las réplicas principales consumen una cantidad similar de memoria, y las secundarias aumentan todas de manera similar. Cabe señalar que para el tamaño 3 en formato png si se duplicó la función a diferencia de en formato jpg. En el caso del consumo de memoria, se observa que la segunda réplica comienza a consumir memoria a partir del segundo 60 del experimento, a diferencia del segundo 120 en el consumo de CPU, posiblemente debido a fallos en la recolección de métricas por la sobrecarga que experimenta el sistema, el origen de estas métricas, y la propia consulta que las genera.

### Consumo de Memoria para Distintos Tamaños de Imágenes

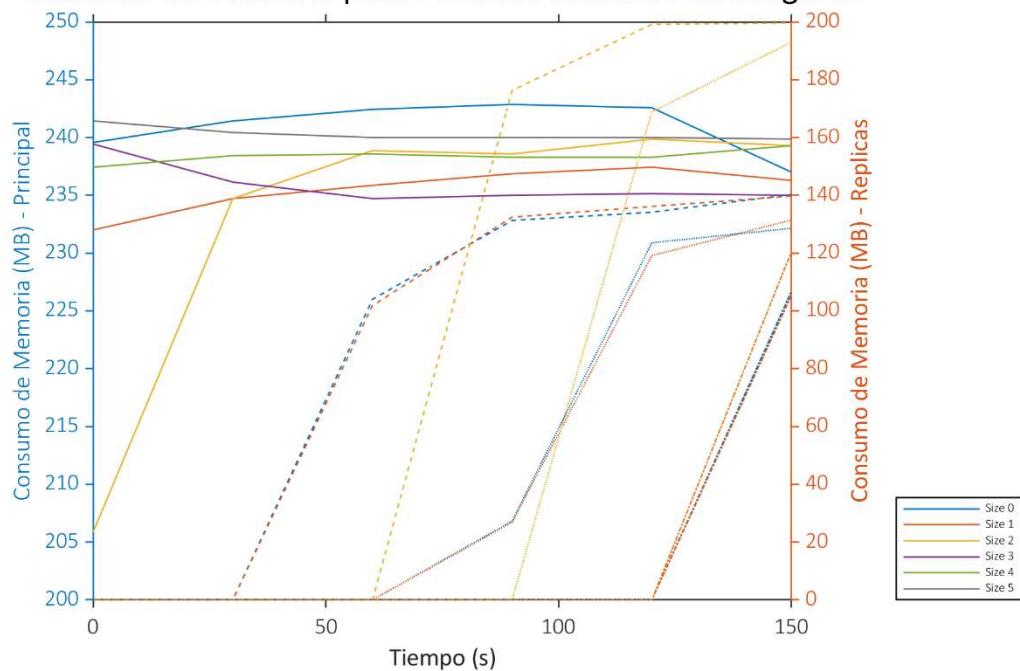


Figura 19: Consumo de memoria por réplica durante el procesamiento de imágenes en formato JPG.

### Consumo de Memoria para Distintos Tamaños de Imágenes

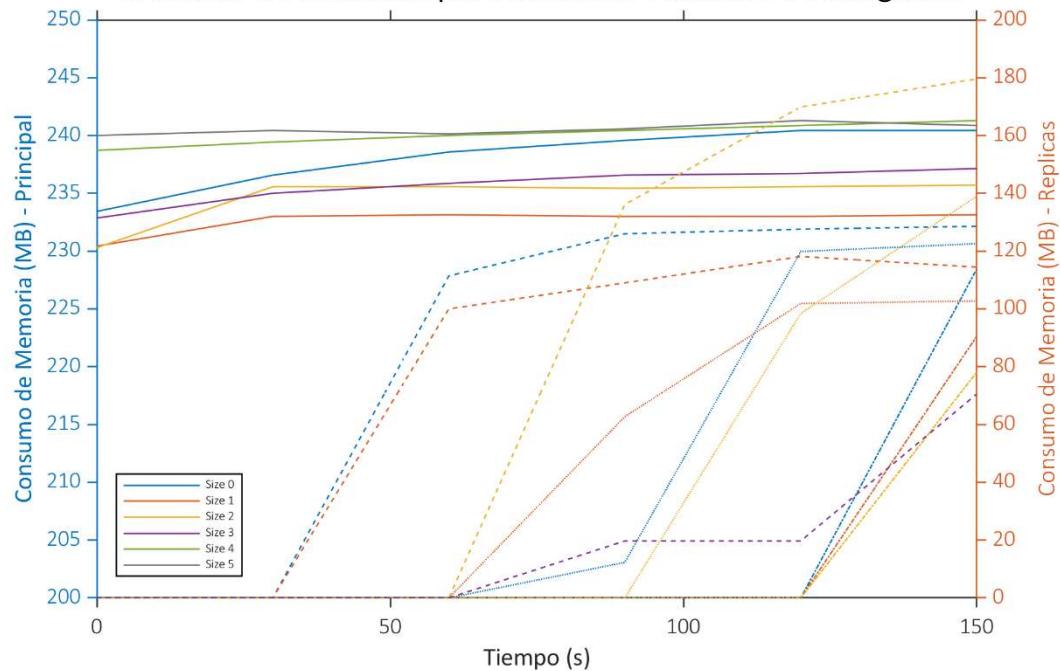


Figura 20: Consumo de memoria por réplica durante el procesamiento de imágenes en formato PNG.

El tiempo de procesamiento de las solicitudes, para aquellas que produjeron una respuesta 200, sigue el comportamiento esperado: aumenta de manera más o menos proporcional al tamaño de la imagen. Un aumento del número de repeticiones del experimento habría producido un resultado mejor distribuido siguiendo el teorema del límite central.

Tiempo de respuesta según tamaño de imagen en formato JPG

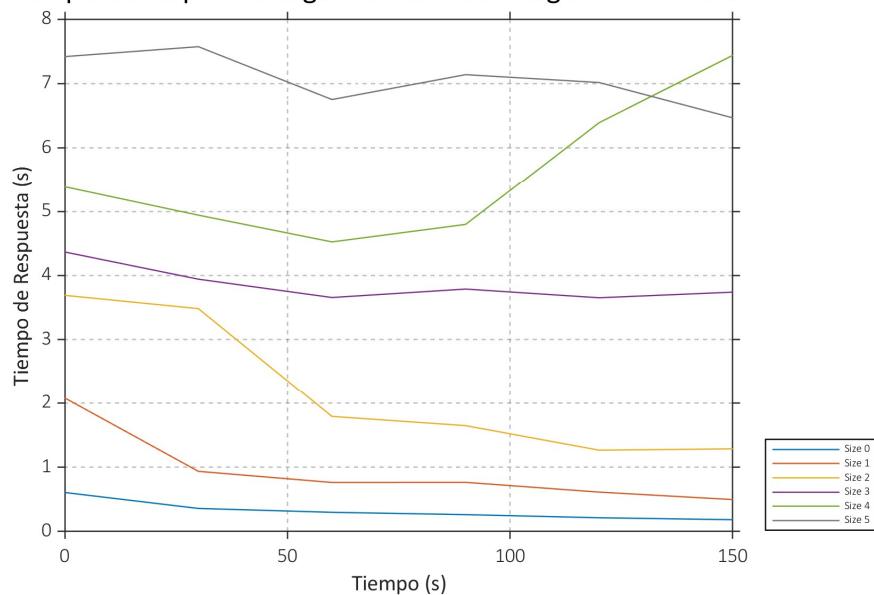


Figura 21: Tiempo medio de respuesta durante el procesamiento de imágenes en formato JPG.

Tiempo de respuesta según tamaño de imagen en formato PNG

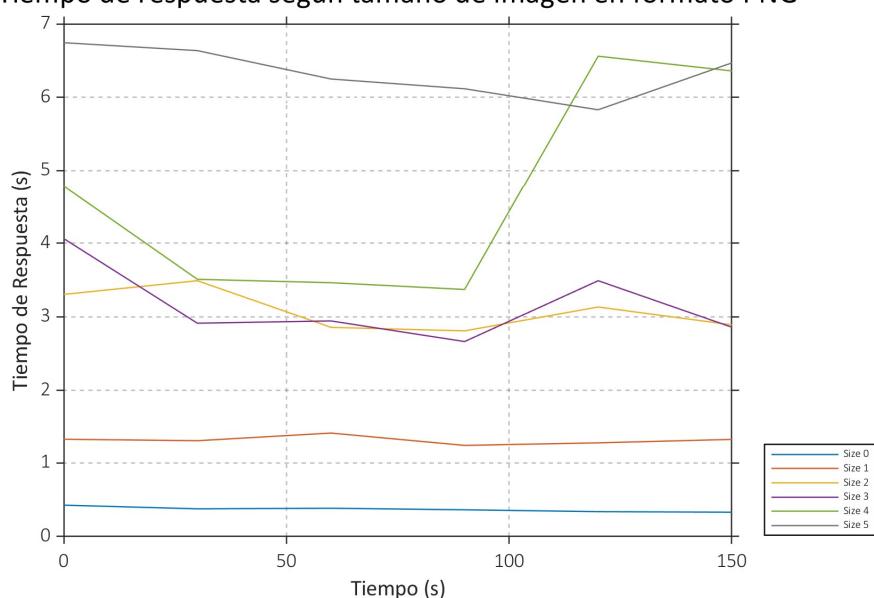


Figura 22: Tiempo medio de respuesta durante el procesamiento de imágenes en formato PNG.

Ciertas ejecuciones produjeron errores en las respuestas, en concreto respuestas 500 (internal server error) y 502 (bad gateway). Estos errores se pueden ver en la siguiente tabla:

	500				502
	3		4		4
	jpg	png	jpg	png	png
0		17,143			
30		12,857			
60		17,143			
90		17,143			
12			17,143		4,286
150		17,143	25,714	17,143	

Tabla 2: Tiempo medio de respuesta para respuestas con errores por formato, tamaño, y código de error

Finalmente, debido a que el tamaño 2, en formato jpg, consigue replicar la función, generar un alto consumo de CPU y memoria, se eligió este tamaño y formato para usar en los siguientes experimentos.

### 5.3 Ataque Masivo DoS

Como ya se mencionó anteriormente, los ataques DoW comparten ciertas características con DoS, aunque, en este segundo caso, el objetivo del ataque es hacer que el rendimiento de un servicio disminuya o quede totalmente inaccesible. Como define OWASP “The Denial of Service (DoS) attack is focused on making a resource (site, application, server) unavailable for the purpose it was designed. (...)

If a service receives a very large number of requests, it may cease to be available to legitimate users.”

Para facilitar el desarrollo del atacante y demostrar la viabilidad de la infraestructura implementada, se optó por una primera iteración donde se simulara un ataque DoS y su correspondiente defensa.

### 5.3.1 Implementación

Para el experimento DoS se reutilizaron las clases implementadas para los experimentos preliminares. El experimento consistió en 10 ejecuciones de un ataque masivo de una duración de 15 minutos. En cada ejecución se creó un *ThreadPool* de 10 atacantes que envían un número ilimitado de solicitudes durante la duración del experimento. Para facilitar la extracción y análisis de datos, se aseguró que los ataques comenzaran siempre cuando el contador de segundos alcanzaba 0. Se introdujo una pausa de un minuto entre iteraciones como tiempo de *cooldown* que permitiera al sistema volver a las condiciones iniciales.

A diferencia de otros experimentos donde la existencia de tráfico base era irrelevante, en este caso sí se optó por la generación de tráfico usando el dispositivo mencionado en la sección 4.3. Este dispositivo permite demostrar en la ejecución

de la defensa que solo se bloquea el tráfico malicioso, y que el servicio sigue disponible y accesible a otros clientes.

### *5.3.2 Ejecución*

La duración total de las 10 iteraciones de ataque sin defensa fue de 160 minutos. Se ejecutó en un horario de poco tráfico en la red para limitar el efecto de la saturación de la red en los resultados.

Para este, y futuros experimentos, se introdujo un nuevo panel de recolección de datos en el dashboard de Grafana. La consulta asociada es

```
sum by (function_name) (increase(gateway_function_invocation_total[10s]))
```

Esta consulta muestra para cada segundo el número de veces que fue invocada la función en los últimos 10 segundos y nos permite observar en tiempo real la saturación que experimenta la función.

### *5.3.3 Análisis del Ataque*

Como se observa en la gráfica, y de manera similar a el experimento anterior, la función se réplica hasta alcanzar un máximo de 5 réplicas. Esto se debe a que, por defecto, las funciones de OpenFaaS tienen un máximo de 5 réplicas. La función alcanza este máximo entorno a los 3 minutos de ejecución del experimento. Como se observará en el resto de los gráficos el número de replicas tiene un gran impacto

en el resto de las métricas debido a la distribución del trabajo entre estas y el consumo de recursos.

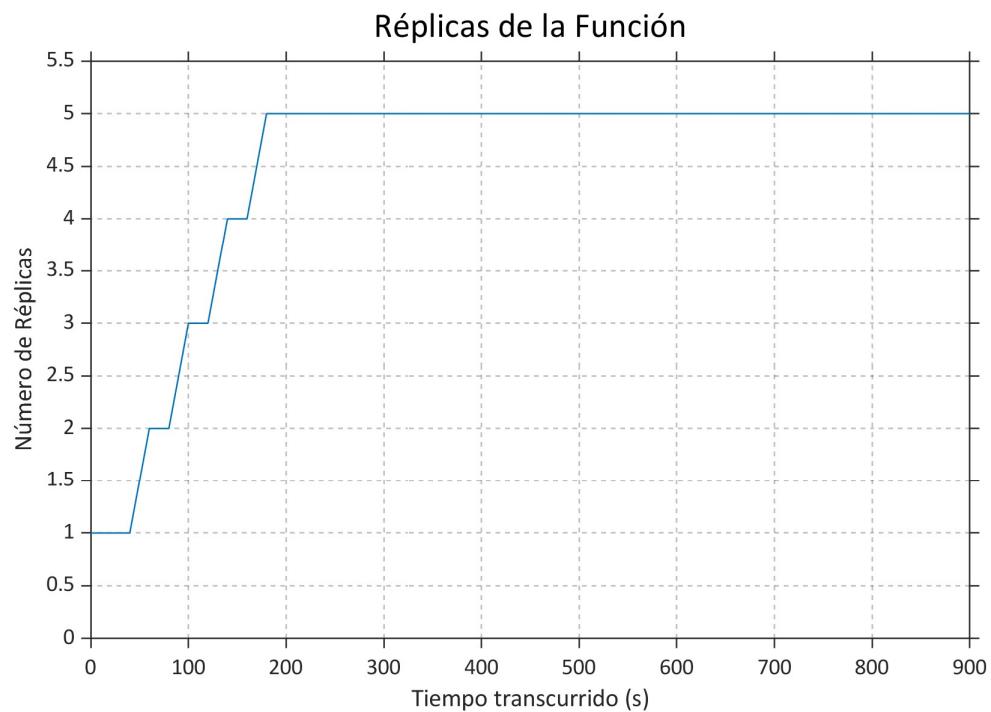


Figura 23: R\'eplicas de la funci\'on durante la ejecuci\'on de un ataque masivo

En el caso de las invocaciones, se observa un enorme pico inicial en el n\'umero de invocaciones. Debido a que la funci\'on se encuentra inicialmente ociosa y puede comenzar a atender solicitudes. Posteriormente desciende significativamente, y comienza a ascender de manera escalonada, asociado a un aumento de las r\'eplicas de la funci\'on que permite atender m\'as solicitudes. El valor se estabiliza una vez se alcanza el m\'aximo n\'umero de funciones en torno a 500 solicitudes.

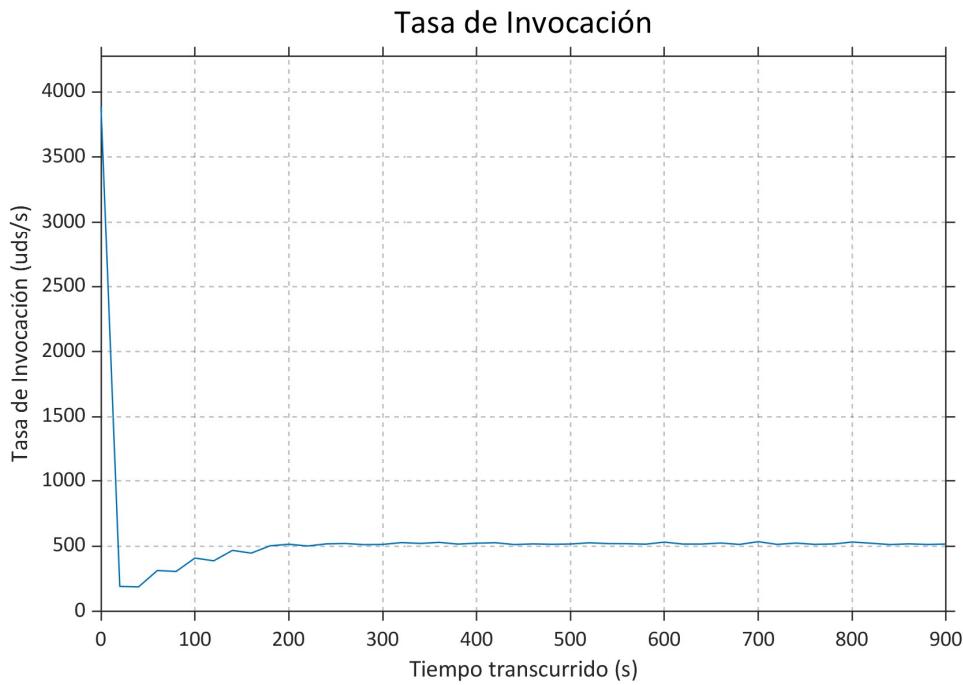


Figura 24: Número de invocaciones recibidas en los últimos 10s.

El gráfico del tiempo de procesamiento también muestra un comportamiento más variado al inicio del experimento. El tiempo de procesamiento comienza siendo mínimo, y asciende rápidamente a medida que más solicitudes tienen que esperar el fin de las ejecuciones anteriores. Desciende también de manera paulatina con cada duplicación de la función hasta estabilizarse en torno a 0,4 s de procesamiento.

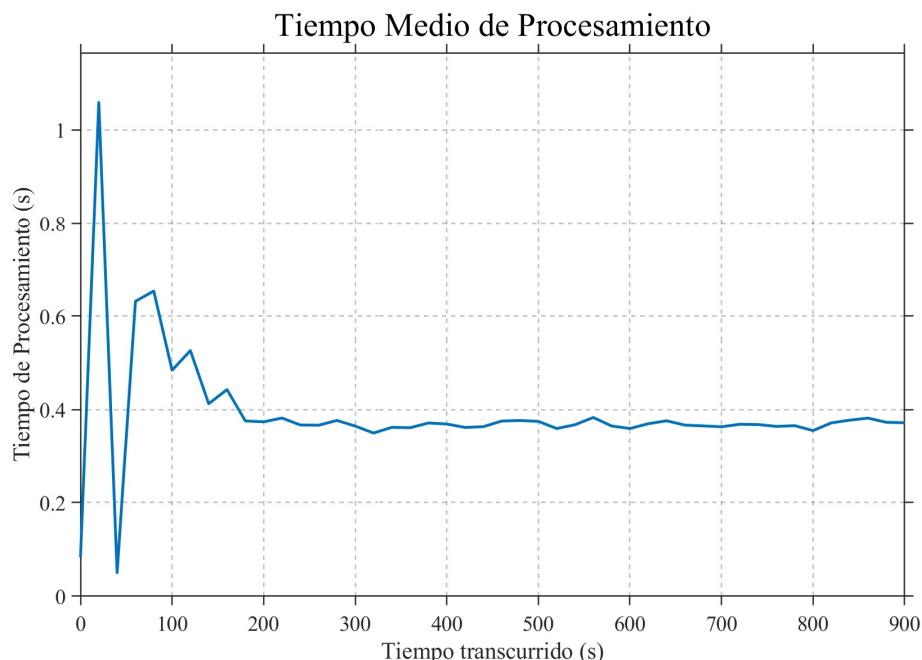


Figura 25: Tiempo medio de procesamiento de las solicitudes

El consumo de CPU muestra un comportamiento muy parecido al del experimento anterior. Cada réplica genera una muestra de esta métrica que inicialmente experimenta un aumento repentino. A medida que aumentan las réplicas y el trabajo se distribuye desciende el consumo hasta que todas ellas se mantienen en un rango de 0,65%-0,85%. Debido a que el rango de valores para el consumo de CPU es mucho menor que el del resto de métricas, se observa una mayor variación. Además, el consumo de CPU se puede ver afectado por otros procesos del sistema operativo, dando lugar a picos y valles de actividad.

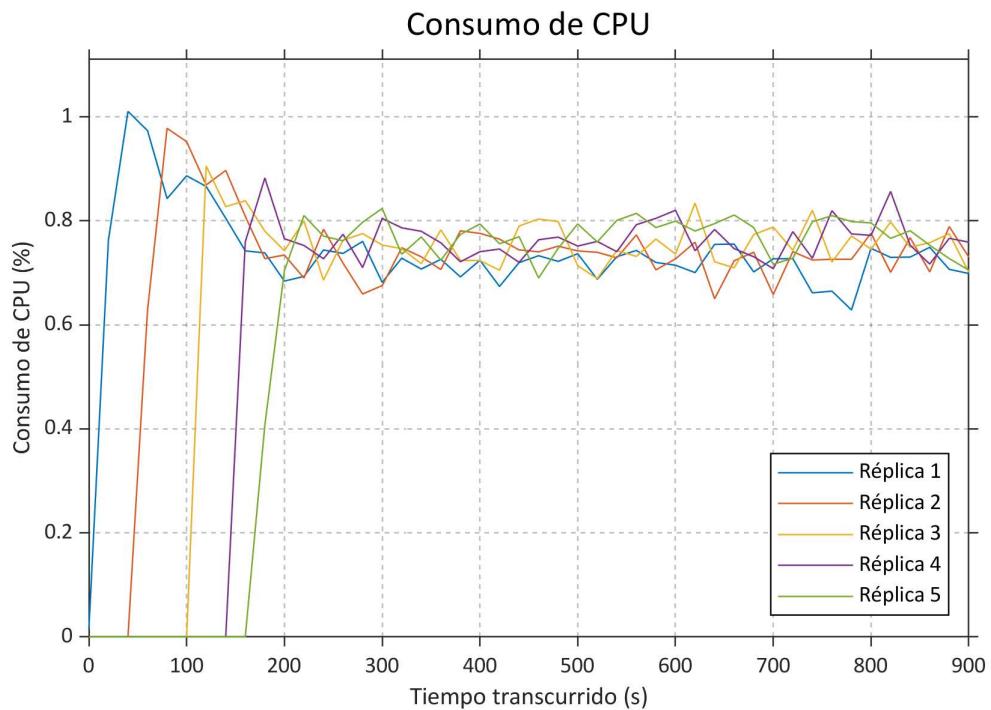


Figura 26: Consumo de CPU por réplica de la función durante el ataque masivo

A diferencia del consumo de CPU, la réplica inicial no aumenta su consumo de memoria al tener una cantidad de memoria ya alocada. A medida que se duplica la

función las nuevas replicas si aumentan su consumo de memoria. Todas se estabilizan entorno a un consumo de 200-230MB.

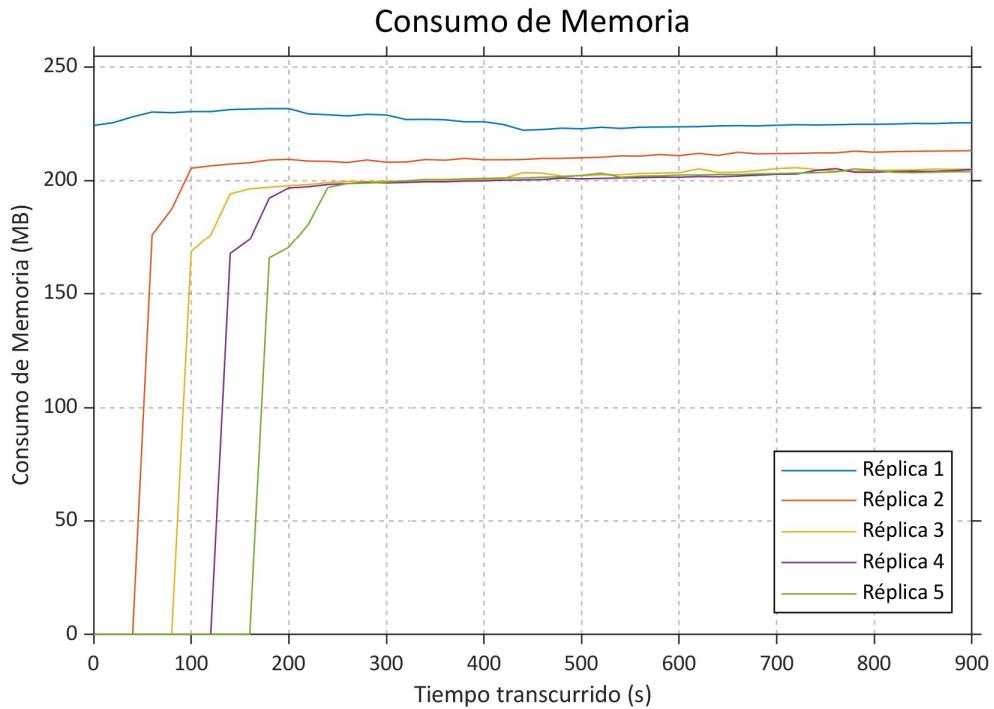


Figura 27: Consumo de CPU por réplica de la función durante el ataque masivo

### 5.3.4 Desarrollo de la Defensa

Para proteger la función del ataque DoS es necesario ser capaz de detectar el ataque y bloquear el tráfico que lo está causando.

Un ataque DoS debido a su objetivo de hacer el servicio inaccesible mediante la saturación de los recursos, es fácil de detectar. Una bajada del rendimiento repentina, un aumento desproporcionado del número de solicitudes recibidas o del consumo de recursos, e incluso la propia inaccesibilidad del servicio, son buenas señales de que está siendo atacado.

En este caso se usó la tasa de invocaciones como indicador del ataque. Utilizando los despliegues incluidos en OpenFaaS de Prometheus y AlerManager, se configuró una alarma utilizando la siguiente consulta:

```
sum by (function_name) (rate(gateway_function_invocation_total[10s])) > 1
```

Una vez prometheus detectó que se alcanza el umbral y se mantiene durante 10s, enviará una notificación al AlerManager. La configuración de la alerta se realizó en

la instancia de Prometheus de OpenFaaS, al estar ya configurada para enviar las alertas al AlertManager.

La función del AlertManager es agregar la información recibida en la alerta de Prometheus y reenviarla a un endpoint definido por un usuario. Existen endpoints predefinidos para el usuario además de un endpoint de tipo webhook http configurable para aplicaciones definidas por el usuario. En este caso se utilizó el webhook con la siguiente configuración:

```
- alert: OpenFaaS_DoS_Attack_Detected
  expr: sum(rate(gateway_function_invocation_total[10s])) BY (function_name) > 1
  for: 10s
  labels:
    action: firewall-block
    service: gateway
    severity: critical
  annotations:
    description: "Blocking IP due to high invocation rate"
    summary: "IP is making too many requests!"
  - name: 'firewall-blocker'
    webhook_configs:
      - url: http://localhost:5000/block
        send_resolved: false
      http_config:
        basic_auth:
          username: admin
```

Figura 28: Configuración de la alerta y el webhook

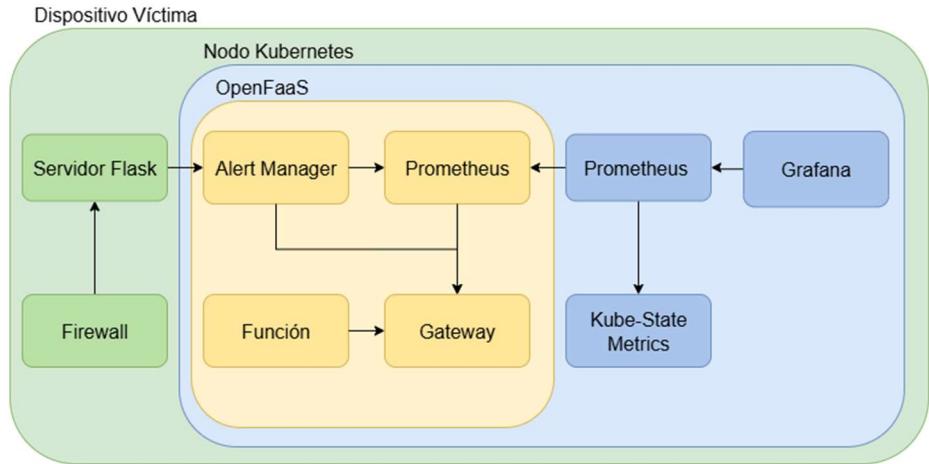
Como se observa en la configuración, se enviará una solicitud http POST a la dirección 192.168.1.44 (dirección IP del dispositivo víctima) en el puerto 5000 donde será recibida y procesada.

Se desplegó un servidor Flask desarrollado en python con un funcionamiento simple. En el servidor se definieron 2 funciones: una llamada “block” para bloquear la ip de la que procede el ataque, y otra llamada “unblock” para desbloquearla. El bloqueo se realizó mediante un script de powershell que crea una nueva regla en el firewall con el siguiente comando:

```
New-NetFirewallRule -DisplayName $ruleNameInbound -Direction Inbound -Action Block -RemoteAddress $ip -LocalPort 8080 -Protocol TCP
New-NetFirewallRule -DisplayName $ruleNameOutbound -Direction Outbound -Action Block -RemoteAddress $ip -LocalPort 8080 -Protocol TCP
```

Estas simples reglas bloquean el tráfico entrante y saliente al puerto 8080 de la ip recibida por parámetro. Al tratarse de una configuración simple con un solo atacante, la dirección IP proporcionada es estática. En un entorno real sería

necesario identificar en tiempo real de que dirección proviene el tráfico malicioso para su bloqueo. El siguiente diagrama muestra la arquitectura de la defensa implementada.

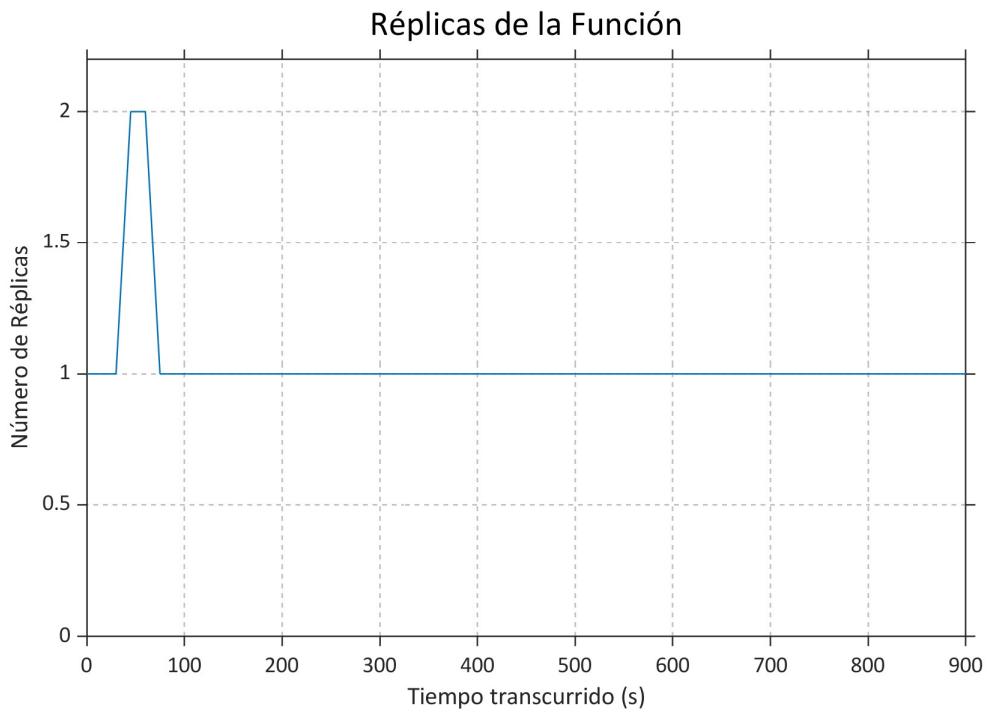


*Figura 29: Arquitectura de los servicios de la víctima con defensa*

El desbloqueo de la IP utiliza un script que borra las reglas cuyos nombres coincidan con los establecidos. La función de desbloqueo de IP fue necesaria para la ejecución del experimento, ya que es necesario abrir el tráfico para realizar las sucesivas iteraciones.

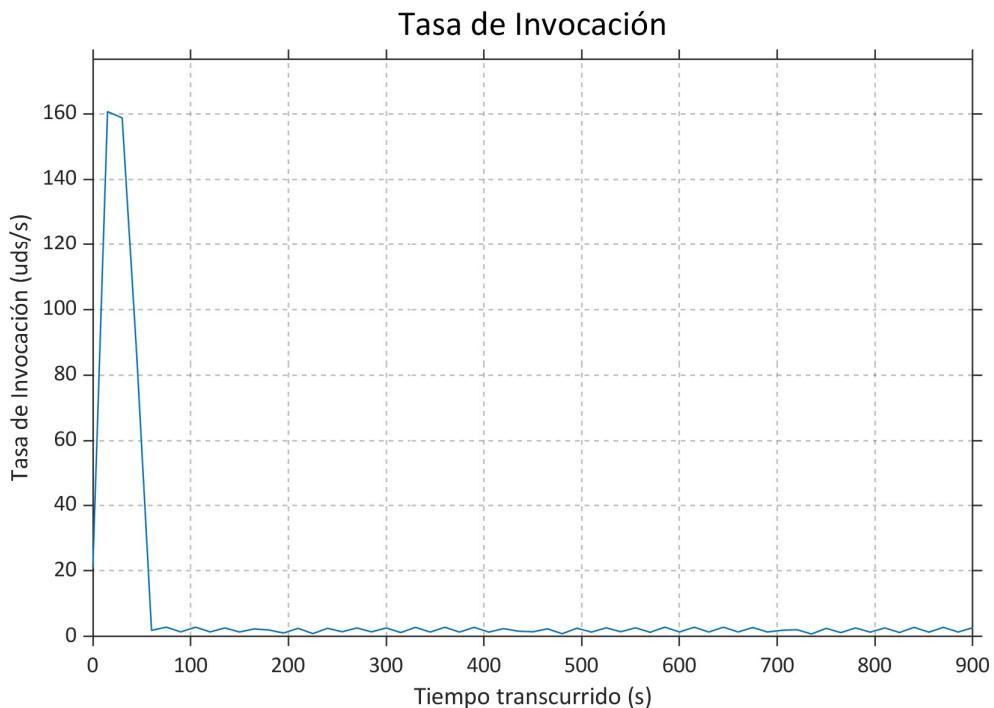
### 5.3.5 Análisis de la Defensa

Tras la ejecución del experimento con la defensa implementada y desplegada, se observan cambios muy notables en los datos recopilados. Observando el gráfico de réplicas de la función, se puede apreciar que en este caso solo se alcanza un máximo de 2 réplicas al principio de la ejecución, antes de que se lancará la alerta programada.



*Figura 30: Replicas de la función durante el ataque masivo con defensa*

En el caso de las invocaciones, existe un pico inicial elevado que alcanza un máximo de 160 invocaciones, pero rápidamente disminuye. A partir del segundo 60 el número de invocaciones oscila entre 1 y 3, representando el tráfico base generado. Esto nos demuestra que la función sigue disponible y accesible y que solo el tráfico maligno ha sido bloqueado



*Figura 31: Tasa de invocación de la función durante el ataque masivo con defensa*

El tiempo medio de procesamiento, de manera similar a los últimos dos gráficos, se observa un pico inicial previo al bloqueo del tráfico. Se alcanza un tiempo de El tiempo de procesamiento máximo de en torno a 1,3 que tras el bloqueo del ataque disminuye rápidamente a aproximadamente 0,15s.

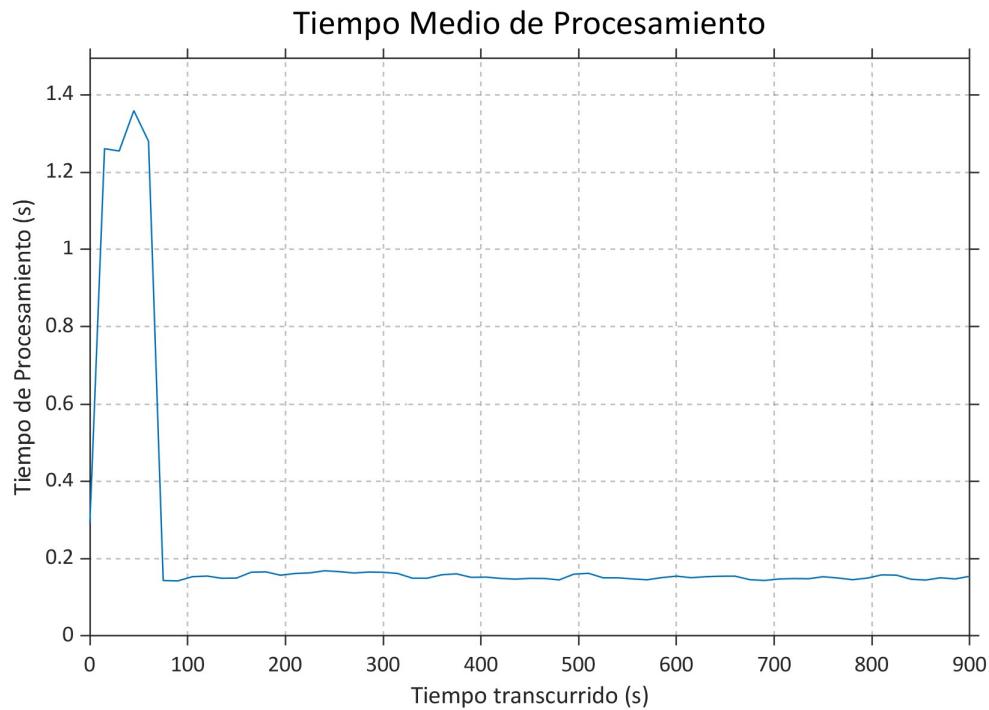


Figura 32: Tiempo medio de procesamiento durante el ataque masivo con defensa

El consumo de CPU sigue un comportamiento similar: un pico inicial seguido de una bajada repentina que demuestra la efectividad de la defensa. De manera similar a el ataque, el pico de consumo de CPU es de 1%, pero este no se mantiene a lo largo del tiempo. La breve vida de la segunda replica se observa en este gráfico que por razones de latencia en la recolección de los datos solo tiene una muestra para el consumo de CPU de esta réplica.

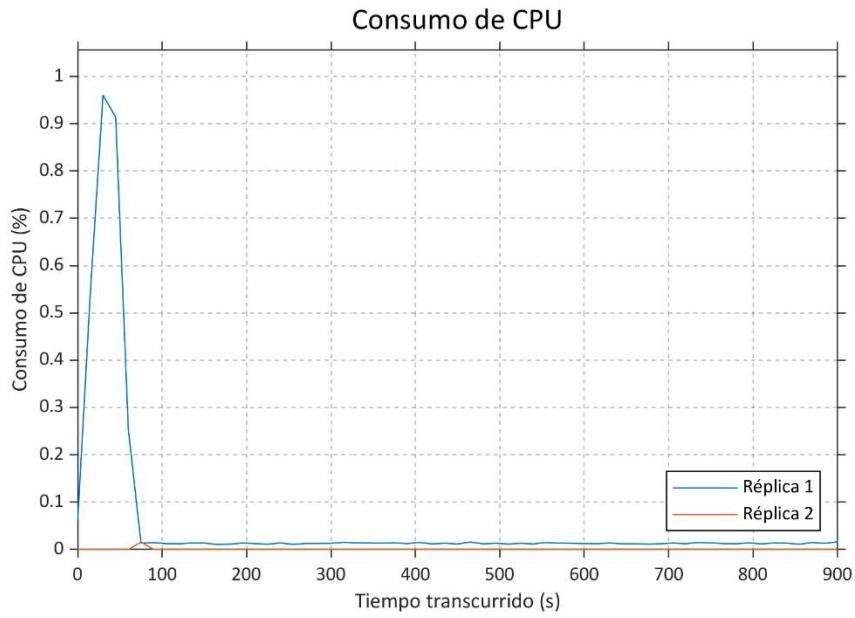


Figura 33: Consumo de CPU durante el ataque masivo con defensa

El consumo de memoria, que ya se había observado ser más estable que el de CPU, mantiene dicho comportamiento. La primera réplica mantiene un consumo estable de unos 240MB de memoria mientras que la segunda recibe inicialmente un bloque de unos 75MB que luego son liberados. A pesar de que la segunda réplica de la función ya esté eliminada en el segundo 75 del experimento, la reserva de memoria tiene una mayor duración por lo que en la gráfica se puede observar que esta memoria queda reservada hasta el segundo 375 del experimento.

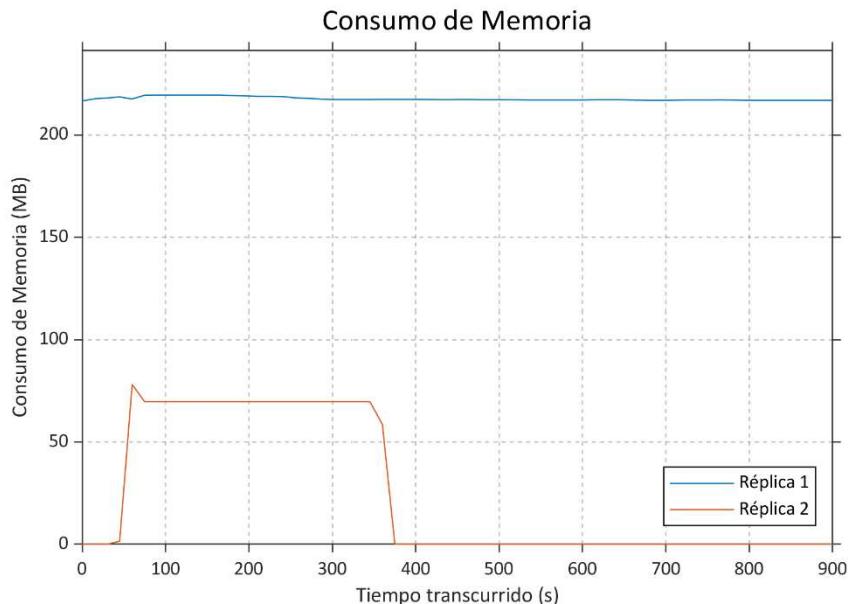


Figura 34: Consumo de memoria durante el ataque masivo con defensa

El último dato que debemos examinar es el del precio incurrido por la función durante el ataque y la defensa. En la siguiente tabla podemos ver el precio incurrido por el procesamiento en cada una de las iteraciones en las principales plataformas de servicios Serverless.

AWS	Google	Azure	IBM	AWS	Google	Azure	IBM
0.1366	0.1398	0.1313	0.1347	0.0021	0.0026	0.0083	0.0021
0.1383	0.1412	0.1329	0.1367	0.0020	0.0027	0.0086	0.0019
0.1384	0.1413	0.1330	0.1368	0.0021	0.0027	0.0086	0.0020
0.1355	0.1386	0.1302	0.1337	0.0023	0.0028	0.0089	0.0022
0.1368	0.1398	0.1315	0.1351	0.0021	0.0027	0.0086	0.0020
0.1448	0.1477	0.1392	0.1432	0.0020	0.0026	0.0084	0.0020
0.1368	0.1399	0.1315	0.1351	0.0021	0.0027	0.0086	0.0020
0.1379	0.1410	0.1325	0.1361	0.0016	0.0028	0.0089	0.0015
0.1304	0.1335	0.1253	0.1285	0.0020	0.0027	0.0085	0.0019
0.1362	0.1393	0.1309	0.1344	0.0020	0.0028	0.0089	0.0020

Tabla 3: precios incurridos por iteración y operador

Aunque el precio generado por el ataque no sea muy elevado, de entorno 13 céntimos, la reducción en el precio es muy significativa. En la siguiente tabla se observa en qué porcentaje se redujo el coste en cada ejecución. Se aprecia que la reducción en precio es muy elevada, variando entre un mínimo de 93,18% a un máximo de 98,87%.

	AWS	Google	Azure	IBM	Media
Iteración 1	98.45%	98.13%	93.65%	98.47%	97.18%
Iteración 2	98.56%	98.10%	93.56%	98.59%	97.20%
Iteración 3	98.49%	98.09%	93.52%	98.52%	97.16%
Iteración 4	98.33%	97.99%	93.18%	98.36%	96.97%
Iteración 5	98.49%	98.06%	93.43%	98.53%	97.13%
Iteración 6	98.59%	98.21%	93.93%	98.62%	97.34%
Iteración 7	98.46%	98.08%	93.49%	98.48%	97.13%
Iteración 8	98.83%	98.02%	93.27%	98.87%	97.25%
Iteración 9	98.46%	98.01%	93.24%	98.49%	97.05%
Iteración 10	98.51%	98.00%	93.21%	98.55%	97.07%
Media	98.52%	98.07%	93.45%	98.55%	97.15%

Tabla 4: Reducción conseguida por iteración y operador

### **5.3.6 Conclusiones**

El experimento propuesto muestra la validez de la arquitectura diseñada para la simulación de un entorno cloud Serverless y un ataque a este tipo de servicios. Utilizando las herramientas propuestas se ha conseguido saturar una función y protegerla, alcanzando unos resultados significativos. La reducción de coste incurrido alcanzada supera las expectativas previas a la ejecución del experimento y demuestra la eficacia de las técnicas escogidas para la mitigación del impacto.

## **5.4 Ataque Leech**

Como ya se ha expuesto en secciones anteriores, un ataque DoW no busca la saturación del servicio mediante una alta tasa de solicitudes, sino que genera un tráfico mucho menor en un periodo de tiempo mucho más prolongado. Con esto se pretende evitar las defensas tradicionales empleadas para ataques DoS, generando aun así un coste significativo en la víctima, debido a la larga duración del ataque.

En esta sección se expondrá el proceso de implementación y ejecución del ataque Leech y su defensa, analizando las distintas decisiones de implementación que se llevaron a cabo, y las conclusiones obtenidas del proceso.

### **5.4.1 Implementación**

Para la implementación del ataque Leech se realizaron modificaciones mínimas gracias a la encapsulación y reutilización de código. Se creó una nueva clase ExperimentoLeech que, usando el ClienteHTTP, enviaba solicitudes HTTP mucho más espaciadas. Debido a la baja tasa de invocación de la función no fue necesario utilizar paralelismo en este experimento, lo cual redujo significativamente el tiempo de desarrollo. Al igual que en el experimento anterior, se aseguró que las distintas ejecuciones de los experimentos comenzasen cuando los segundos alcanzasen 0.

### **5.4.2 Ejecución**

De manera similar a otros experimentos, se decidió realizar un total de 10 ejecuciones del mismo experimento para poder obtener medias que representasen

valores mas significativos estadísticamente. La duración de cada iteración del experimento fue de 1h.

Para simular un ataque Leech se redujo la tasa de envío de solicitudes. Se configuró el atacante para que enviara una solicitud HTTP y suspendiera el hilo un tiempo aleatorio entre 30 y 60s, con lo que se esperaba que el atacante enviase en torno a 80 solicitudes durante el experimento.

Debido a la prolongada duración del experimento, por errores desconocidos, seguramente relacionados con la política de timeout del dispositivo víctima, durante la primera ejecución se observó que a partir de la 6<sup>a</sup> iteración no se recibieron más solicitudes, por lo que fue necesario lanzar una segunda vez el experimento. Los datos fueron agregados de las dos ejecuciones del experimento.

Por último, debido a que la baja tasa de invocación del atacante, se creó un nuevo panel de recopilación de datos que midiera el número de solicitudes recibido en los últimos 10m usando la siguiente consulta:

```
sum by (function_name) (increase(gateway_function_invocation_total[10m]))
```

Este nuevo panel facilitó enormemente la monitorización del experimento, haciendo más evidentes las fluctuaciones en el tráfico.

#### *5.4.3 Análisis del Ataque*

La primera métrica que analizaremos, de manera similar al experimento anterior, será el número de réplicas. Como se puede observar en el gráfico, el número de réplicas se mantiene constante en 1, a lo largo de todo el ataque, lo cual es de esperar debido a la baja tasa de invocaciones.

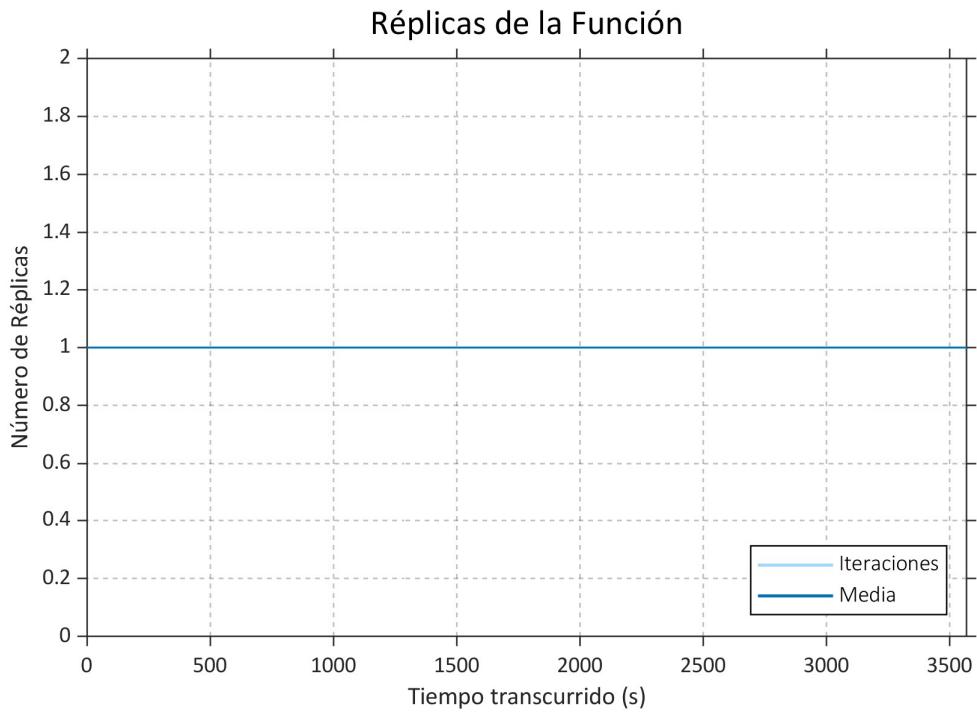


Figura 35: Réplicas de la función durante la ejecución del ataque leech

En el caso de las invocaciones se analizarán tanto el gráfico de la tasa de invocación como el de invocaciones totales. En el primer caso, se puede apreciar un comportamiento ciertamente errático. El valor de solicitudes recibidas en los últimos 10s oscila entre 2 y 2,7 solicitudes. Este comportamiento es de esperar ya que el generador de tráfico base envía 1 solicitud cada 10 segundos, mientras que el atacante envía una solicitud cada 30-60s.

Suponiendo un periodo de 90s podemos esperar 9 solicitudes de base y 2 de tráfico maligno. Podemos calcular el número de invocaciones esperadas en 10 s de la siguiente forma:

$$I = \frac{9 + 2}{90} \times 10 = 1.2\bar{2}$$

Como se aprecia en la gráfica, el valor real de la tasa de invocación oscila en torno a este valor, y si calculamos la media de los valores obtenidos obtenemos un resultado de 1,2054.

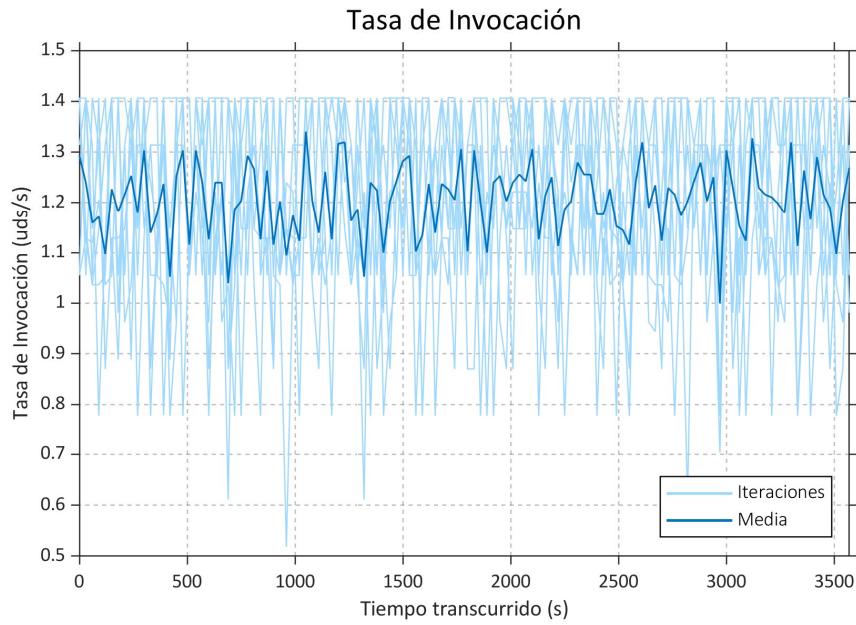


Figura 36: Tasa de invocación durante la ejecución del ataque leech

El segundo gráfico, de número de solicitudes totales recibidas en los últimos 10m. Se observa que el valor medio se mantiene en torno a las 72 solicitudes. Con los siguientes cálculos podemos obtener el valor esperado para esta métrica.

$$S = S_B + S_M = \frac{600 \text{ s}}{10 \text{ s/sol}} + \frac{600 \text{ s}}{45 \text{ s/sol}} = 73, \bar{3}$$

El valor observado es de 72.4225 solicitudes, cercano al calculado, ya que además debemos tener en cuenta el factor probabilístico del tráfico malicioso.

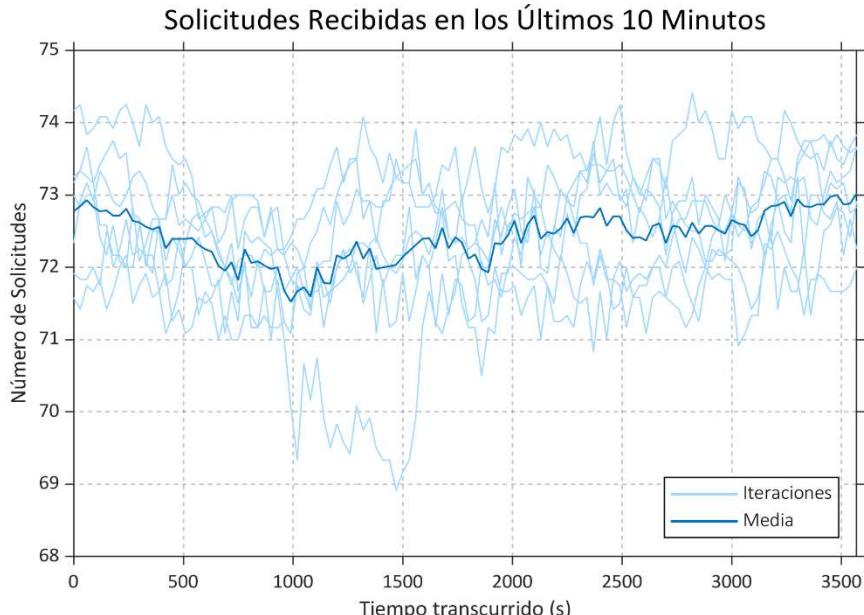


Figura 37: Invocaciones recibidas en los últimos 10 minutos durante la ejecución del ataque leech

En el caso del tiempo medio de procesamiento de las solicitudes observamos que el valor se mantiene bajo, generalmente por debajo de 0,1s. Debido a la baja saturación de la función no se registran variaciones a lo largo del experimento, más allá de las que puedan ser provocadas por el uso de los recursos del sistema por otros programas y procesos.

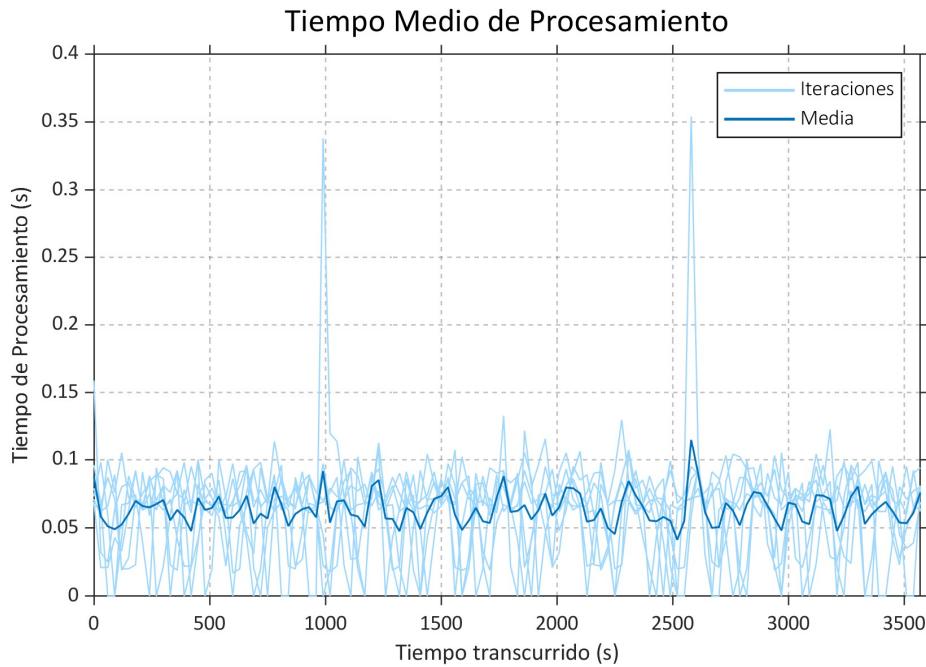


Figura 38: Tiempo de procesamiento durante la ejecución del ataque leech

De manera similar, el consumo de CPU no presenta fluctuaciones significativas. Este permanece en torno al 0,015% durante la duración del experimento.

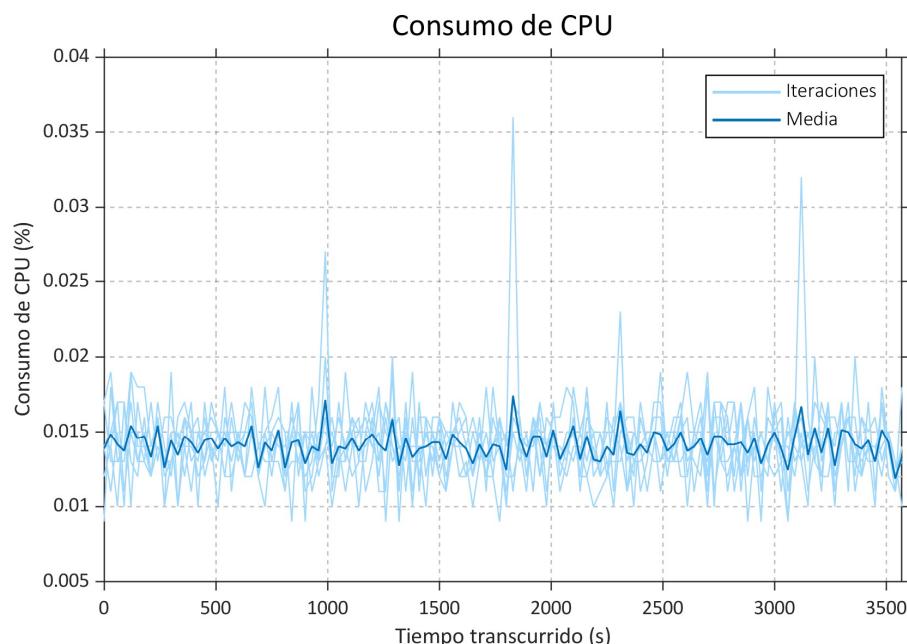
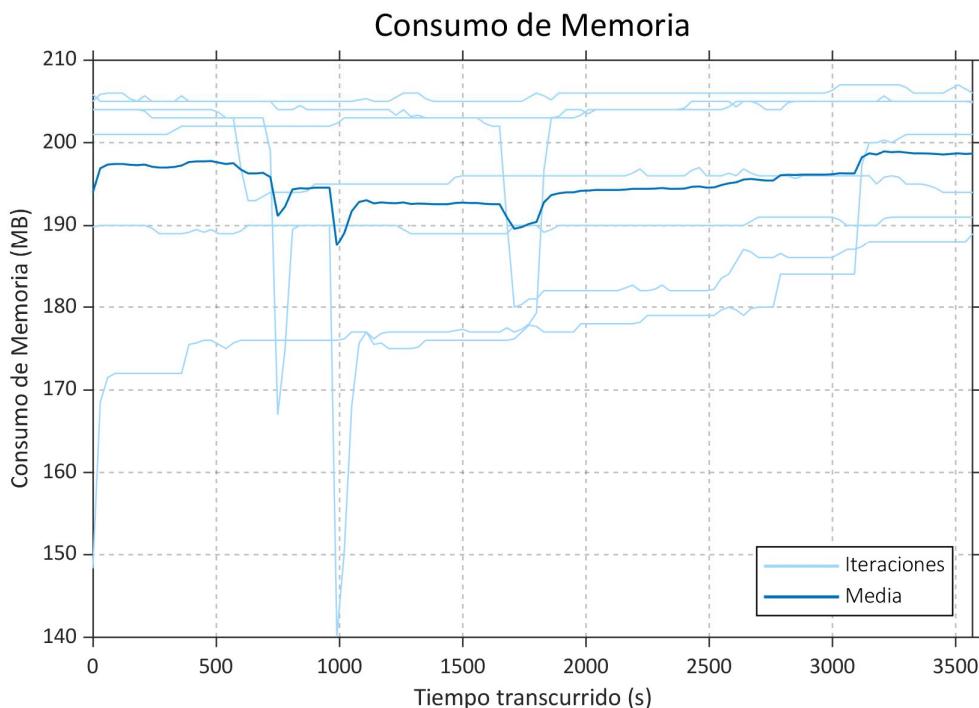


Figura 39: Consumo de CPU durante la ejecución del ataque leech

Por último, se examinarán los datos obtenidos de consumo de memoria. En este caso sí podemos observar mayor variación entre las iteraciones. Los valores se mantienen todos en un rango de 140 – 205 MB de memoria. Ocasionalmente se observan descensos repentinos, que remontan rápidamente alcanzando los valores observados anteriormente.



*Figura 40: Consumo de memoria durante la ejecución del ataque leech*

Es importante notar que las reservas de memoria duran un periodo de tiempo, por lo que el estado del sistema al final de una iteración del sistema puede afectar los valores del inicio de la siguiente. Por esto podemos observar que una de las iteraciones, en concreto la primera, que empieza de un estado “frío”, muestra un aumento al inicio de la ejecución. Sin embargo, más allá de esta, el resto de las variaciones no coinciden en el tiempo ni la magnitud como para poder extraer ninguna conclusión.

#### 5.4.4 Desarrollo de la Defensa

Debido a la baja tasa de invocaciones, y la poca e inestable variación de esta, la métrica y regla de bloqueo empleada con anterioridad para la defensa no es compatible con este ataque.

En vez de utilizar la tasa de invocación, se planteó usar el número total de invocaciones ya que este no muestra una oscilación tan elevada como la tasa de invocación, y además tiene valores esperados estables y calculables.

Como vimos en el anterior apartado, el número de invocaciones recibidas en el ataque en 10m se estima entorno a 73,3 de los cuales 60 son de tráfico benigno. Para demostrar este comportamiento, se recopilaron los datos de invocaciones totales cuando, empezando desde un estado frío, la víctima recibía únicamente el tráfico base, y se comparó con otra ejecución en frío de tráfico base y malicioso.

Como se observa en la siguiente figura, el tráfico base no supera nunca el valor de 60 mientras que con el tráfico malicioso el valor oscila entorno a las 73 solicitudes.

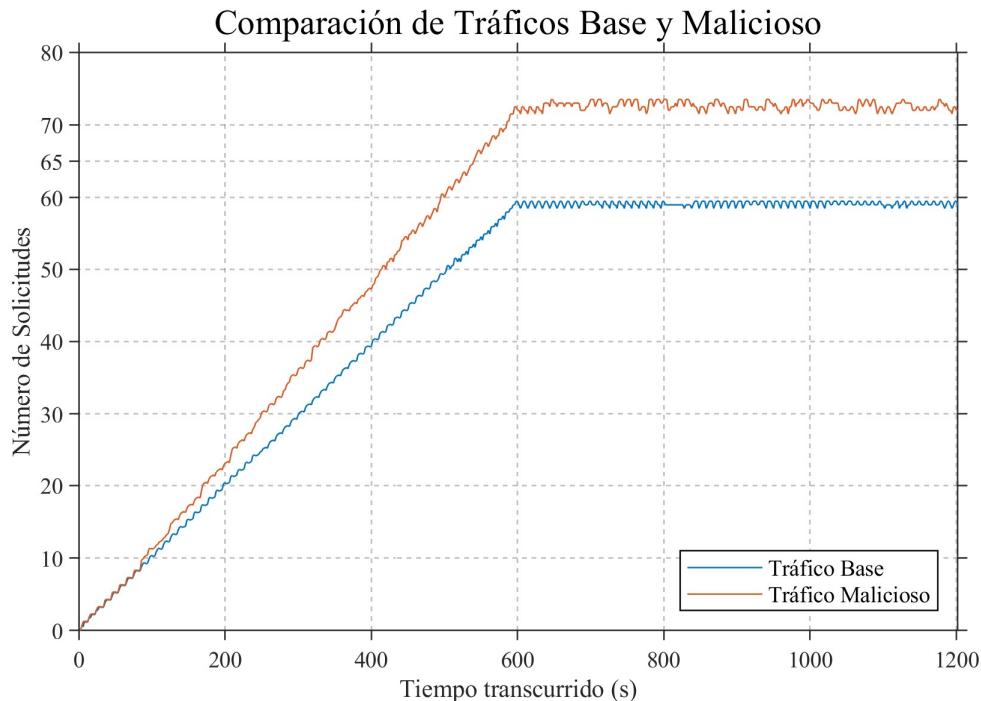


Figura 40: Comparación de tráfico base y malicioso, y umbral escogido.

Así podemos definir un umbral por encima del cual determinamos que el tráfico es malicioso. En un contexto real, este proceso se asemejaría a evaluar el uso habitual de un servicio para poder detectar cuando un servicio está recibiendo niveles de tráfico anómalos.

En este caso se definió un umbral de 65 solicitudes que permitiera reducir al máximo el tráfico malicioso permitido. La arquitectura de la defensa se mantuvo igual modificando solo la regla de alerta utilizada para el bloqueo de tráfico. La regla fue la siguiente:

```
sum by(function_name)(increase(gateway_function_invocation_total[10m])) > 65
```

Tras esta modificación, se levantó el servidor encargado del bloqueo del tráfico, y se lanzó de nuevo el experimento con la defensa.

#### 5.4.5 Análisis de la defensa

De la misma forma que en la ejecución sin defensa, la saturación de la función es mínima, por lo que no se duplica la función, sino que se mantiene en 1 réplica durante todo el experimento.

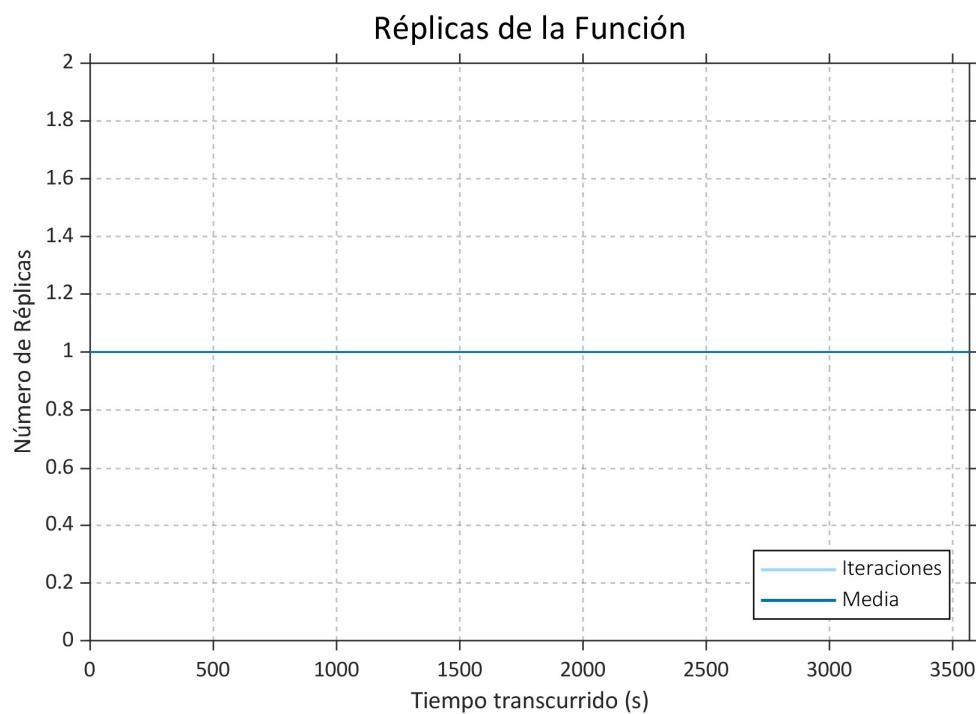


Figura 41: Réplicas durante la ejecución del ataque leech con defensa

Sin embargo, sí se aprecia una diferencia en las invocaciones de la función. En el caso de la tasa de invocación se observa un descenso del 1,2 al 1 alrededor del segundo 250. Al activarse la defensa, y pasar a recibir solamente el tráfico base, la tasa baja a 1 solicitud cada 10s, que es precisamente la configuración del generador.

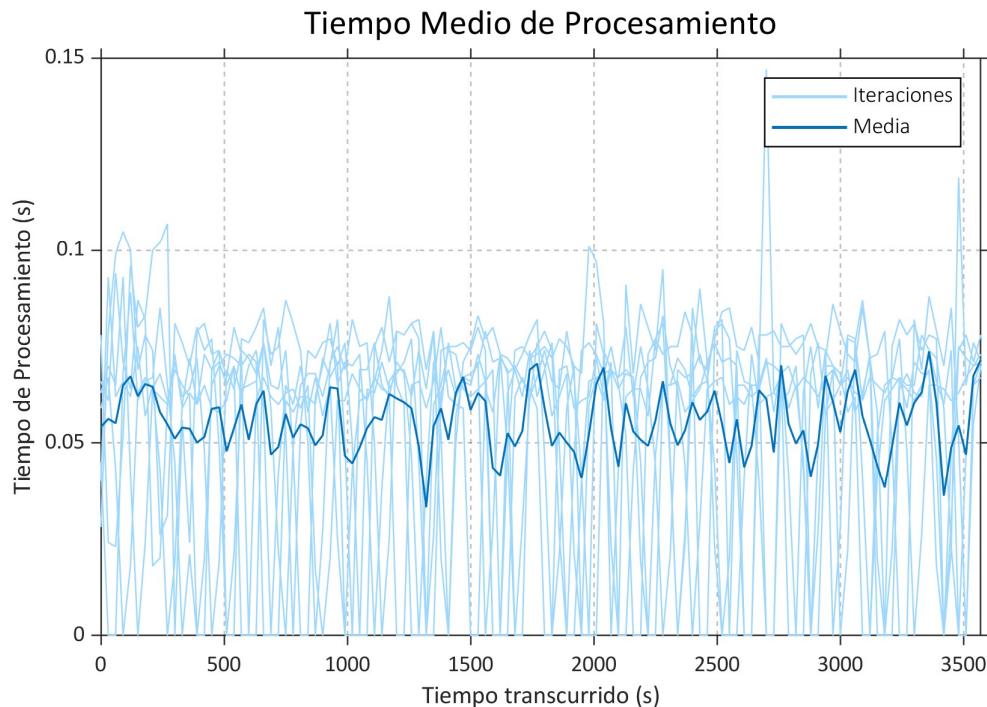


Figura 42: Tiempo de procesamiento durante la ejecución del ataque leech con defensa

En el caso del número de solicitudes recibidas en los últimos 10m se observa también un aumento hasta que esta métrica supera el umbral designado de 65. En este momento se activa la defensa. A pesar de que el tráfico se bloquea inmediatamente, se observa el descenso 10m después debido a la definición de la métrica.

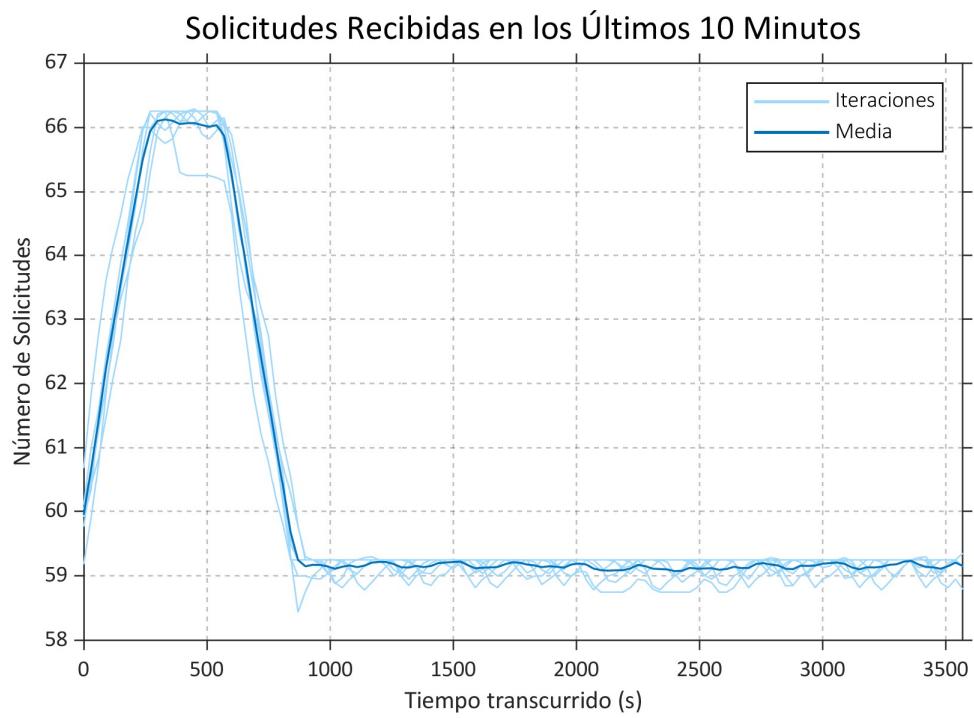


Figura 44: Número de invocaciones recibidas en los últimos 10 minutos durante la ejecución del ataque leech con defensa

El tiempo de procesamiento es muy similar al del ataque sin defensa, de nuevo debido a que el ataque no satura los recursos del sistema. Se mantiene estable alrededor de los 0.05

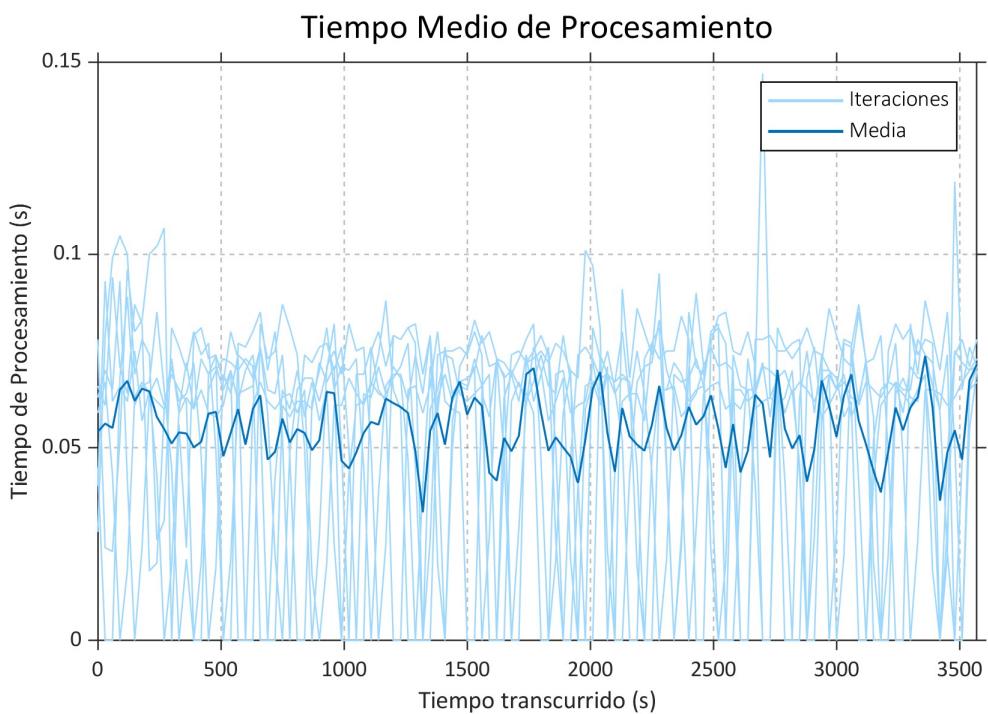


Figura 45: Tiempo medio de procesamiento durante la ejecución del ataque leech con defensa

Sí se puede apreciar un efecto de la defensa en el consumo de CPU. Mientras que antes de la activación de la defensa el consumo es superior a 0,014%, tras su activación desciende hasta aproximadamente 0,013%, con cierta variabilidad, pero nunca superando 0,014%.

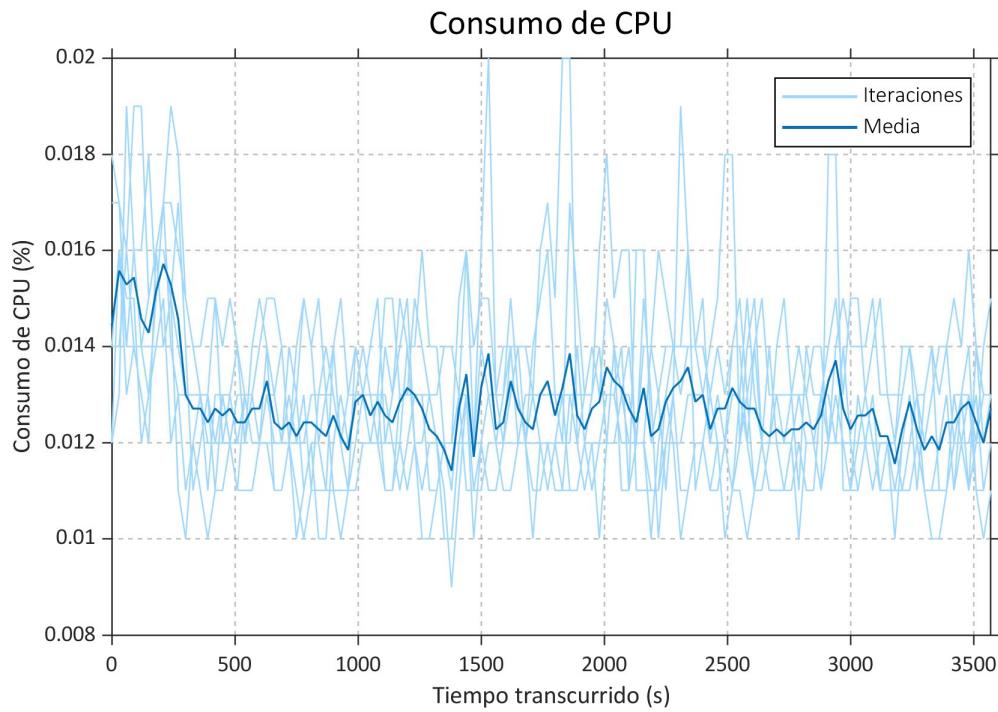


Figura 46: Consumo de CPU durante la ejecución del ataque leech con defensa

Por último, observando el gráfico de memoria, el valor medio de consumo podría, a priori, indicar una bajada de consumo debido a la defensa. No obstante, este descenso es provocado por una única iteración que muestra este comportamiento, seguramente provocado por variaciones esporádicas. El resto de las iteraciones muestran variaciones, pero ninguna de ellas correlacionadas con la activación de la defensa.

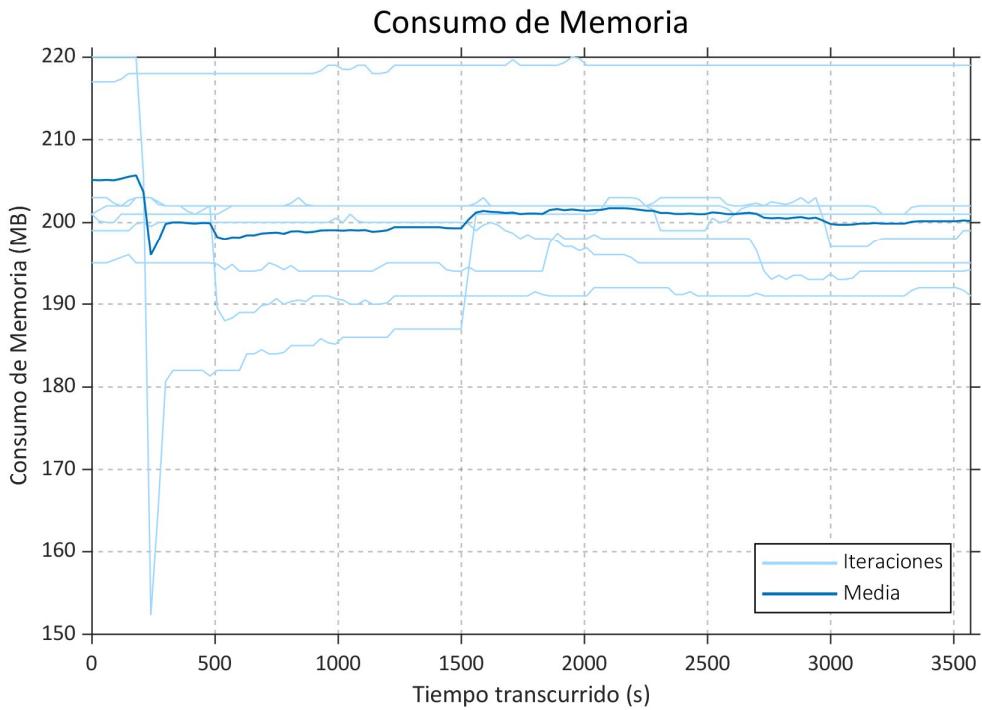


Figura 47: Consumo de memoria durante la ejecución del ataque leech con defensa

Por último, se analizará el cambio en el coste del experimento. En la siguiente figura se puede observar el coste de cada iteración. En el lado izquierdo se observa el coste, en dólares, para cada una de las plataformas del experimento sin defensa, y a la derecha el coste con defensa.

AWS	Google	Azure	IMB	AWS	Google	Azure	IMB
0.0022	0.0011	0.0007	0.0005	0.0006	0.0003	0.0002	0.0001
0.0029	0.0014	0.0009	0.0006	0.0014	0.0007	0.0004	0.0003
0.0036	0.0017	0.0012	0.0007	0.0027	0.0013	0.0009	0.0005
0.0042	0.0021	0.0014	0.0009	0.0033	0.0016	0.001	0.0006
0.002	0.001	0.0006	0.0004	0.0038	0.0018	0.0012	0.0007
0.0026	0.0013	0.0009	0.0005	0.0008	0.0004	0.0003	0.0002
0.0033	0.0016	0.0011	0.0007	0.0013	0.0007	0.0005	0.0003

Tabla 5: precios incurridos por iteración y operador en el ataque leech con y sin defensa

En general se observa un descenso en el coste, aunque a diferencia del ataque masivo, en este caso no es tan pronunciado ni uniforme. En la siguiente tabla se puede apreciar la variación del coste por iteración, proveedor de servicios, y sus medias.

	AWS	Google	Azure	IMB	Media
IT1	72.73%	72.73%	71.43%	80.00%	74.22%
IT2	51.72%	50.00%	55.56%	50.00%	51.82%
IT3	25.00%	23.53%	25.00%	28.57%	25.53%
IT4	21.43%	23.81%	28.57%	33.33%	26.79%
IT5	-90.00%	-80.00%	-100.00%	-75.00%	-86.25%
IT6	69.23%	69.23%	66.67%	60.00%	66.28%
IT7	60.61%	56.25%	54.55%	57.14%	57.14%
Media	30.10%	30.79%	28.82%	33.44%	30.79%

Tabla 5: reducción en los precios por iteración y operador en el ataque leech con y sin defensa

Se observa que todas las iteraciones alcanzan un descenso de entre 25% a 75% del coste original, salvo la iteración 5. En este caso el precio aumentó un 86,25%. Se trata de un comportamiento muy inesperado que debe ser analizado.

Para calcular el precio total de una iteración debemos conocer el número de solicitudes procesadas. Con una consulta de PromQL y procesando los datos recopilados para la elaboración de gráficas obtenemos los siguientes datos

	Solicitudes	CPU
Sin Defensa	862	0,013033%
Con Defensa	722	0,014042%

Tabla 6: datos relevantes para el cálculo de precio

Se observa que, aunque el número de solicitudes sea menor, el consumo de CPU, que también se utiliza para calcular el precio es mayor en la iteración con defensa. Esto podría explicar el comportamiento de esta iteración, sobre todo ya que, al trabajar con números muy pequeños, estas mínimas diferencias pueden impactar significativamente al precio.

#### 5.4.6 Conclusiones

Tras la implementación y ejecución del ataque y su consecuente defensa, en este caso aproximándose a un tipo de ataque totalmente nuevo, se ha conseguido desarrollar una estrategia de mitigación de ataques DoW para servicios Serverless.

Se han alcanzado reducciones significativas y consistentes, con una media de reducción de precio de 30%. Se ha demostrado la efectividad de la arquitectura propuesta para simular un entorno Serverless, proporcionar un entorno aislado para la simulación de situaciones de carga, y desarrollar estrategias de defensa apropiadas.

## 6. Limitaciones

En la realización de este trabajo se ha buscado en todo momento el máximo acercamiento posible a un entorno real, para garantizar la aplicabilidad de los resultados obtenidos fuera de un entorno de investigación. Sin embargo, por evidentes limitaciones de tiempo, presupuesto, y acceso a recursos, existen una serie de limitaciones tanto en la metodología como en la arquitectura que deben ser mencionadas.

En primer lugar, se debe notar el bajo número de dispositivos físicos usados para la simulación de un entorno Cloud. En entornos reales, existen miles de dispositivos dentro de un solo centro de datos, a su vez perteneciente a una enorme red de centros de los operadores. Por otro lado, los ataques a estos servicios no se realizan desde un único dispositivo; grupos organizados con extensas redes de dispositivos infectados se coordinan para perpetrar ataques de mayor escala.

La arquitectura diseñada ha servido para el desarrollo de las estrategias de detección y detención de tráfico, y para medir su efectividad. Sin embargo, sería necesario profundizar mediante un despliegue de estas estrategias en entornos mayores que permitan medir su efectividad cuando las condiciones son más verosímiles.

Por otro lado, debemos destacar la elección de software utilizado para el desarrollo. Se optó por utilizar software open-source, para evitar incurrir en costes económicos, especialmente cuando se busca simular condiciones de tráfico elevado. A pesar de lo útil que ha resultado la elección de esta tecnología, implica también unas limitaciones en cuanto a los datos a los que se tiene acceso, la extensibilidad del producto, etc.

En un entorno real, dentro de la red de infraestructura cloud de un proveedor, el proveedor tiene acceso a más datos y mayor control sobre toda la infraestructura para, por ejemplo, controlar el tráfico, medir anomalías, discriminar el origen de una solicitud posiblemente fraudulenta, segregar el tráfico por usuarios, etc.

Además, hay que mencionar la infraestructura física utilizada. Los dispositivos usados para la arquitectura no son servidores, ni la red en la que se ha desplegado es de altas prestaciones. En este sentido, una peor infraestructura física afecta de manera directa a métricas vitales como el tiempo de transmisión de una solicitud, el tiempo de procesamiento, la cantidad de memoria disponible, etc. También se debe notar que la red utilizada es una red doméstica, por lo que la saturación de la red puede haberse visto afectada por el uso compartido de la misma.

Todas estas limitaciones apuntan a la necesidad de replicar los resultados en entornos mayores, para verificar los datos y conclusiones obtenidas en este trabajo de investigación.

## 7. Aspectos Legales, Sociales, Éticos, y Ambientales

El objetivo de esta sección es realizar un análisis exhaustivo del impacto social, ético, y ambiental del trabajo desarrollado, enumerando quien se vería afectado por este trabajo, especificando los impactos positivos que podría tener, y determinando si existe algún impacto negativo, siempre dentro del marco de la Agenda 2030 y los objetivos de desarrollo sostenible (ODS).

Para ello se definirán los objetivos de desarrollo sostenible dentro de la Agenda 2030, se determinará a cuáles de ellos contribuye el trabajo realizado, de qué manera aportan al cumplimiento del objetivo, y se evaluará el alcance de este impacto.

Por otro lado, es necesario realizar un análisis del marco legal vigente y como el trabajo desempeñado encaja dentro de la legislación actual. Se tendrán en cuenta la legislación a nivel tanto europeo como nacional, haciendo hincapié en las recientes modificaciones a la legislación en materia de servicios de la información y mercados digitales.

### 7.1 Agenda 2030 y Objetivos de desarrollo sostenible.

En el año 2015, la Asamblea General de la Organización de Naciones Unidas adoptó la denominada “Agenda 2030 para el Desarrollo Sostenible”, un “plan de acción [...]” que también tiene la intención de fortalecer la paz universal y el acceso a la justicia [29].” Se trata de un plan para orientar las acciones de los estados miembros de la organización con el objetivo de satisfacer las necesidades de la población y responder a los retos únicos que se plantean actualmente, incluyendo el cambio climático, la pobreza, la desigualdad, etc.

El plan está compuesto por 17 Objetivos de Desarrollo Sostenible. Incluyen el fin de la pobreza (ODS 1), la igualdad de género (ODS 5), la reducción de las desigualdades (ODS 10), etc. Para facilitar su cumplimiento, cada objetivo tiene una serie de metas que permiten determinar el progreso hacia el mismo y cuyo cumplimiento determinaría la satisfacción del objetivo.

El trabajo desarrollado contribuye a varios objetivos, de lo que el número 9, “Industria, Innovación e Infraestructuras” es el más importante. Este objetivo pretende “construir infraestructuras resilientes, promover la industrialización sostenible y fomentar la innovación [30]”. Este trabajo de investigación se ha desarrollado en un área pionera, donde actualmente se sigue realizando investigación. Indaga sobre una tecnología desarrollada recientemente, con un inmenso potencial, y que ya está aportando un valor real a muchas empresas y consumidores. En los próximos años “*SMEs (...) are expected to witness significant growth (...) due to the increasing awareness of the benefits [31]*”. Esta tecnología puede ser de gran utilidad para pequeñas empresas, o empresas en países con menor desarrollo tecnológico al ofrecer servicios muy flexibles, de fácil acceso, y sin necesidad de infraestructura física propia, y, por tanto, su seguridad es de vital importancia para el desarrollo industrial y tecnológico.

Un ataque DoW representa un robo indirecto de recursos, ya sea a una corporación como una PyME, además de una posible amenaza a la privacidad de los usuarios de un servicio dependiente de una plataforma Serverless. La securización de estos servicios sería un gran avance en cuanto al objetivo de desarrollo 16, “Paz, justicia, e instituciones sólidas”. Reduciría las amenazas a las plataformas, sus clientes, y los usuarios finales cuyos derechos son vulnerados por este tipo de ataques.

En cuanto al impacto ambiental de los ataques, puede parecer mínimo, pero de la misma forma que el daño acumulado puede generar graves impactos económicos, también supone un gasto significativo de energía y recursos físicos. Estos ataques conllevan la necesidad de despliegue de más infraestructura física al ocupar recursos en tráfico malicioso, con el inmenso impacto que conlleva la fabricación de dispositivos electrónicos, tanto en CO2 emitido como residuos electrónicos generados. Además, el consumo extra de energía puede suponer, dependiendo del origen de la energía usada en los centros de datos, una gran cantidad de CO2 emitido, e incluso un desperdicio de agua, utilizada en la refrigeración de los centros de datos que alojan las plataformas Serverless. Por ello la mitigación de

estos ataques aportaría beneficios a los objetivos 13 “Acción por el Clima”, 14 “Vida Submarina” y 15 “Vida de Ecosistemas Terrestres”.

Es también importante para este área de desarrollo el objetivo 17 “Alianzas para Lograr los Objetivos”. La colaboración entre empresas e instituciones educativas y de investigación, así como la colaboración internacional entre distintas universidades, y un entorno de competencia honesta y legal entre las distintas plataformas, es de vital importancia para el avance y desarrollo de las nuevas tecnologías Cloud. En este caso que, como mencionamos al principio del documento, brinda la oportunidad única de mitigar el impacto y proteger el servicio antes si quiera de que estos ataques se produzcan, es aún más importante la colaboración a distintos niveles para evitar los daños.

Por último, existe una necesidad de personal altamente cualificado para el despliegue de soluciones como la propuesta, lo cual significa una generación de empleo de calidad. La adopción de las tecnologías Serverless puede ser en sí misma un motor de crecimiento económico al ser accesibles a todo tipo de clientes por su bajo coste. Todo esto contribuye de manera directa al objetivo 8 “Trabajo Decente y Crecimiento Económico” e, indirectamente, al 10 “Reducción de las desigualdades”, en tanto que representa una oportunidad real para que países menos desarrollados adopten tecnologías que mejoren su productividad.

## 7.2 Impacto Social

Aunque la Agenda 2030 y los Objetivos de Desarrollo Sostenible son herramientas importantes para guiar el desarrollo de la economía a gran escala, existen muchos aspectos a considerar que escapan este marco de referencia.

Las tecnologías cloud se han convertido en herramienta ubicuas sobre las que se construye toda la infraestructura digital de la que depende una enorme parte de nuestra economía y sociedad; la digitalización de las agencias del estado, la asistencia sanitaria, la comunicación entre personas, el ocio, e incluso tareas mucho más simples como la gestión de un pequeño bar, o el inventario de un

negocio familiar, dependen en mayor o menor medida de los servicios digitales. Por ello garantizar un servicio de calidad y seguro es ahora más importante que nunca.

Tomemos por ejemplo el caso de una incipiente empresa de software de edición de video. A diferencia de un gran conglomerado como Adobe o Microsoft, sus recursos, ya sean fiscales o capital humano, son mucho más limitados. Para esta empresa la gestión propia de servidores o máquinas virtuales significa una asignación de recursos proporcionalmente mayor, y quizás opten por el uso de servicios serverless para parte de su procesamiento. En este caso, un ataque tipo DoW puede representar una amenaza real para su continuidad. Una compañía mayor podría, incluso, organizar de manera clandestina, un ataque para eliminar su competencia. Esto no solo afectaría a los empleados y clientes directos de la compañía, sino que crea un desequilibrio en el mercado que afecta a todos los usuarios. Otorga a las empresas establecidas una ventaja competitiva en detrimento del consumidor.

Aunque no existen instancias confirmadas de este tipo de comportamiento por parte de compañías privadas, no es difícil imaginar cómo podría llegar a suceder dado el incentivo. Por otro lado, se debe tener en cuenta que este tipo de acciones las pueden orquestar grupos de cibercriminales con sus propios motivos o incluso estados que buscan dañar rivales geoestratégicos atacando su economía e infraestructura privada.

También existe un riesgo real de ataque en sectores públicos. Nos encontramos en un momento de digitalización de los servicios estatales como la seguridad social, la sanidad, padrones municipales, etc. Esto aporta muchísimos beneficios como un reducido coste de gestión, facilidad de acceso para la ciudadanía, aceleración de la atención ofrecida, etc. Sin embargo, también existe un riesgo asociado a esta transformación. El alojamiento de datos en plataformas cloud, especialmente de datos sensibles, como el historial de salud o información fiscal de la ciudadanía, requiere de una seguridad especial para garantizar la privacidad.

En el caso de ataques DoW a infraestructura publicada alojada en Serverless, supondría una reducción de la calidad del servicio ofertado además de un aumento

de gasto para las administraciones, que afecta directamente a todos los ciudadanos y corporaciones que contribuyen fiscalmente a su financiación. Un ataque a una entidad pública tendría también consecuencias políticas, debido a la responsabilidad de los gobiernos de velar por el uso eficiente de los fondos recaudados, recalmando la necesidad de entornos Serverless seguros.

### 7.3 Otras Consideraciones Éticas

Como se explicó en la sección 2.2, las compañías que ofrecen servicios Serverless actúan bajo un modelo de responsabilidad compartida. Esto implica una responsabilidad por parte del usuario de desarrollar sus funciones aplicando las prácticas de seguridad necesarias para mitigar su propio riesgo. Este deber surge en primer lugar por el bien del propio cliente del servicio, para protegerse a sí mismo y sus usuarios, garantizando la continuidad de su actividad económica, y en segundo lugar por la responsabilidad para con el resto de los usuarios del servicio. Una brecha de seguridad en un servicio puede tener consecuencias para otros usuarios cuyo servicio pudiera estar siendo ejecutado en las mismas máquinas. En este sentido, el usuario tiene una responsabilidad ética para consigo mismo, pero también para con los demás usuarios.

Por otro lado, las grandes compañías de servicios Cloud, y en concreto Serverless, tienen una enorme responsabilidad ética. Debido a que el usuario solo gestiona la capa de aplicación, y tiene un acceso muy limitado a las demás capas del servicio, es el ofertante quien debe gestionar de manera correcta el resto de las partes de esta infraestructura. Deben garantizar la seguridad a nivel físico (acceso a los servidores, redes, y demás aparatos físicos) y a nivel digital (accesos lógicos a la plataforma, gestión del tráfico de la red, software utilizado para alojar los servicios...)

De nuevo, esta responsabilidad ética tiene varias perspectivas relevantes. Desde un punto de vista individual, la compañía, con el objetivo de obtener beneficios para sí misma y garantizar el bienestar de sus empleados, debe ofrecer un servicio seguro para no incurrir en daños reputacionales que pudieran disminuir el número

de clientes y reducir sus beneficios, valor en bolsa, y proyección de futuro. Esto afecta a todos los niveles de la compañía: empleados, directivos, clientes y accionistas.

No obstante, la compañía no opera en un vacío, sino que existe en un contexto social en el que interactúa con otras empresas, sus propios proveedores, la ciudadanía general, e incluso gobiernos y organizaciones supranacionales o no gubernamentales. En el caso de una gran compañía, un ataque a su infraestructura, especialmente si surge debido a una mala praxis, puede afectar a cientos de servicios comerciales y miles de usuarios. La interconexión de internet hace que una vulnerabilidad pueda ser dañina para un número muy elevado de personas, si no es posible contenerla.

Una vulnerabilidad de una plataforma serverless podría causar gastos multimillonarios. Si ocurriera un ataque DoW a una plataforma entera, por difícil que pueda parecer, significarían pérdidas millonarias para el conjunto de afectados.

Debido a esta responsabilidad propia hacia uno mismo, pero también hacia el resto de los miembros de la sociedad, es extremadamente importante el uso de estrategias de prevención, mitigación, y recuperación de ciberataques.

#### 7.4 Aspectos Legales

Además de la responsabilidad ética hacia la sociedad general, es de vital importancia tener en cuenta la responsabilidad legal de las plataformas Serverless. Existen recientes cambios en la legislación a nivel nacional y europeo que establecen las obligaciones y derechos de los implicados y deben ser examinados.

En noviembre de 2022, la Unión Europea aprobó la directiva NIS2 (*Network and Information Systems 2*) [32]. Esta directiva tiene como objetivo reforzar y mejorar las medidas de seguridad establecidas en directivas anteriores, así como establecer un marco común a nivel europeo en materia de ciberseguridad.

A nivel gubernamental, la directiva obliga a los estados miembros de la UE a establecer estrategias de ciberseguridad nacionales junto con autoridades competentes que garanticen su cumplimiento. Además, los estados deben crear autoridades de gestión de crisis de ciberseguridad, puntos de contactos únicos, equipos de respuestas para incidentes de ciberseguridad, y un grupo de cooperación a nivel europeo [33].

En cuanto a entidades privadas, la directiva amplia el alcance de aplicabilidad, designando dos tipos de entidades, esenciales o importantes, y aumentando el número de sectores incluidos. Según su sector, actividad, y tamaño, las entidades deben autoclasificarse, y aplicar las prácticas establecidas. Las empresas deberán implementar medidas de gestión del riesgo de ciberseguridad, informar de vulneraciones de ciberseguridad utilizando los mecanismos establecidos, y cumplir con la legislación a nivel nacional. Por último, las compañías deberán garantizar la seguridad de su cadena de suministro, designar personal específico para la gestión de su seguridad, e informar a las autoridades nacionales competentes de sus medidas de seguridad.

Finalmente, esta directiva establece una serie de medidas penales por incumplimiento. La directiva se establece a nivel europeo, y establece unos requisitos mínimos, pero debe ser implementada a nivel nacional pudiendo aumentar el rigor.

Asimismo, se debe mencionar también la Ley de Ciberseguridad (Cybersecurity Act) [34]. Esta ley tiene como objetivo fortalecer el rol de la Agencia Europea de Seguridad de Red e Información (ENISA o agencia europea de ciberseguridad). Designa la agencia como institución permanente asignándole más responsabilidades y recursos. Entre sus funciones se incluye aumentar la cooperación operacional a nivel europeo, coordinar respuestas en caso de ciberataques a gran escala.

Además de reforzar la agencia, la ley establece una directriz para una certificación de ciberseguridad europea, con el objetivo de aumentar la seguridad de los

productos y servicios informáticos, armonizar y estandarizar el mercado, y proveer información clara sobre seguridad a los consumidores. ENISA se designa como responsable del desarrollo de este marco de certificación, informando al público de los certificados, y propiciando los cambios necesarios en el entorno de los servicios digitales necesarios para la creación de este marco de certificación.

Por último, se debe examinar también el Reglamento de Resiliencia Operativa Digital (DORA) [35]. El objetivo de esta ley es “conseguir un alto nivel de resiliencia operativa digital para las entidades financieras reguladas [36]”. Para ello harmoniza las regulaciones de gestión de riesgo de las tecnologías de información y comunicación, fortalece el escrutinio de los proveedores de servicios a terceros, aplicándolas las mismas exigencias que a las entidades financieras mencionadas.

De estas tres regulaciones en materia de ciberseguridad, la Ley de Cyberseguridad no requiere de transposición nacional, sino que es de aplicación directa. En el caso de DORA, si es necesario la aprobación de una ley que desarrolle el marco penal del reglamento. En nuestro país esto se está llevando a cabo mediante el anteproyecto de Ley de Digitalización y Modernización del Sector Financiero aprobado el 17/12/2024 [37].

Por último, el NIS2 está en proceso de transposición a la legislación española mediante el anteproyecto de la Ley de Coordinación y Gobernanza de la Ciberseguridad [38].

El trabajo desarrollado resulta relevante a la aplicación de todas estas leyes ya que las estrategias de defensa, mitigación y recuperación de ciberataques son indispensables en la gestión de la ciberseguridad de cualquier empresa, tanto operadores de servicios cloud como sus clientes.

## 8. Conclusiones

Desde la investigación inicial, indagando en el estado actual del arte, las últimas investigaciones, y los más nuevos productos del mercado, pasando por el desarrollo de la arquitectura y metodología, y finalizando con la ejecución y análisis de los ataques y defensas, este trabajo ha resultado una inmersión en el proceso de investigación y desarrollo de nuevas técnicas.

Habiendo establecido claramente la gran importancia del desarrollo preventivo de estrategias de defensa y mitigación de ataques DoW, que esta nueva tecnología Serverless requiere, y el potencial daño que puede suponen el fracaso en esta tarea, espero haber podido comunicar efectivamente la urgencia de esta investigación.

Utilizando las mejores técnicas a mi alcance, y en coordinación con mi tutor de trabajo, creo haber demostrado la efectividad de la arquitectura propuesta como entorno para la simulación de ataques en entornos cloud. A pesar de las limitaciones de la arquitectura, metodología, y tecnologías disponibles, la investigación y desarrollo realizado sirven de prueba conceptual de la efectividad de las estrategias propuestas para la mitigación del daño de los ataques.

## 9. Bibliografía

- [1] Datadog, “The State of Serverless”, ago. 2023.
- [2] M. Pocwierz, “How an empty S3 bucket can make your AWS bill explode”, *Medium*, 29-abr-2024..
- [3] Statisa, “Public Cloud - Worldwide”, 2024.
- [4] Flexera, “Flexera 2024 State of the Cloud Report”, 2024.
- [5] Crowdstrike, “Global Threat Report”, 2024.
- [6] Esentire, “2023 Official Cybercrime Report”, 2023.
- [7] J. Surbiryala y C. Rong, “Cloud Computing: History and Overview”, 2025.
- [8] Amazon, “What is Virtualization?”, 2024.
- [9] Docker, “What is a Container?”, 2024.
- [10] ISO, “ISO/IEC 22123-1:2023Information technology — Cloud computingPart 1: Vocabulary”, ISO, Ginebra, Suiza, feb. 2023.
- [11] P. McDonald, “Introducing Google App Engine + our new blog”, *Google App Engine Blog*, abr-2008..
- [12] J. Barr, “AWS Lambda – Run Code in the Cloud”, *AWS News Blog*, nov-2024..
- [13] OpenFaaS, “Invocations”, 2024.
- [14] A. Ellis, “Gateway”, dic. 2024.
- [15] A. Ellis, “How and why you should upgrade to the Function Custom Resource Definition (CRD)”, *OpenFaaS Blog*, 08/2023..
- [16] A. Ellis, “Introducing the OpenFaaS Operator for Serverless on Kubernetes”, *OpenFaaS Blog*, 07/2018..
- [17] A. Ellis y H. Verstraete, “Watchdog”, mar. 2023.
- [18] OpenFaaS, “Async”, 2024.
- [19] OpenFaaS, “Monitoring Functions”, 2024.
- [20] OpenFaaS, “Auto-scaling your functions”, 2024.

- [21] Zero Trust Working Group, Cloud Security Alliance, “How to Design a Secure Serverless Architecture”, oct. 2023.
- [22] D. Kelly, F. G. Glavin, y E. Barrett, “Denial of wallet—Defining a looming threat to serverless computing”, *Journal of Information Security and Applications*, vol. 60, núm. 2021, ago. 2021.
- [23] J. Shen, H. Zhang, Y. Geng, J. Li, J. Wang, y M. Xu, “Gringotts: Fast and Accurate Internal Denial-of-Wallet Detection for Serverless Computing”, en *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 2627–2641.
- [24] D. Mileski y H. Mihajloska, “Distributed Denial of Wallet Attack on Serverless Pay-as-you-go Model”, en *2022 30th Telecommunications Forum (TELFOR)*, 2022, pp. 1–4.
- [25] K. Daniel y G. Frank, “DoWNet—classification of Denial-of-Wallet attacks on serverless application traffic”, *Journal of Cybersecurity*, vol. 10, núm. 1, nov. 2022.
- [26] D. Kelly, F. G. Glavin, y E. Barrett, “DoWTS – Denial-of-Wallet Test Simulator: Synthetic data generation for preemptive defence”, *Journal of Intelligent Information Systems*, vol. 60, pp. 325–348, abr. 2023.
- [27] OpenFaaS, “Deploy OpenFaaS Community Edition (CE) to Kubernetes”, 2024.
- [28] Prometheus Community, “Prometheus”, feb. 2025.
- [29] ONU, “La Asamblea General adopta la Agenda 2030 para el Desarrollo Sostenible”, *Centro de Noticias de la ONU*, sep-2015..
- [30] ONU, “Objetivo 9: Construir infraestructuras resilientes, promover la industrialización sostenible y fomentar la innovación”, *Objetivos de Desarrollo Sostenible*, 2015. [En línea]. Disponible en: <https://www.un.org/sustainabledevelopment/es/infrastructure/>. [Consultado: 05-2025].
- [31] A. Dhapte, “Serverless Computing Market Research Report”, jul. 2025.
- [32] Instituto Nacional de Ciberseguridad (INCIBE), “FAQ NIS2”, INCIBE, 2022. [En línea]. Disponible en: <https://www.incibe.es/incibe-cert/sectores-estrategicos/FAQNIS2>. [Consultado: 05-2022].
- [33] P. y. C. de la UE, *DIRECTIVE (EU) 2022/2555 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL*. 2022.
- [34] European Commission, *EU Cybersecurity Act*. 2025.

- [35] European Parliament and Council, *Digital Operational Resilience Act (DORA)*. 2022.
- [36] European Parliament and Council, *Reglamento (UE) 2022/2554 del Parlamento Europeo y del Consejo de 14 de diciembre de 2022 sobre la resiliencia operativa digital del sector financiero y por el que se modifican los Reglamentos (CE) nº 1060/2009, (UE) nº 648/2012, (UE) nº 600/2014, (UE) nº 909/2014 y (UE) 2016/1011*. 2022.
- [37] G. de E. Ministerio de Economía Comercio Y Empresa, *Anteproyecto de Ley de digitalización y modernización del sector financiero*. 2025.
- [38] G. de E. Ministerio del Interior, *Anteproyecto de Ley de Coordinación y Gobernanza de la Cibeseguridad*. 2025.