DATASET PARA ANÁLISIS Y DETECCIÓN DE AMENAZAS DENIAL OF WALLET (DOW) EN ARQUITECTURAS SERVERLESS

José Manuel Ortega-Candel
Higinio Mora
Francisco José Mora-Gimeno
Antonio Maciá-Lillo
Víctor Adsuar Abaldea

Marzo 2024

DATASET PARA ANÁLISIS Y DETECCIÓN DE AMENAZAS DENIAL OF WALLET (DOW) EN ARQUITECTURAS SERVERLESS

José Manuel Ortega-Candel[1]
Higinio Mora[1]
Francisco José Mora-Gimeno[1]
Antonio Maciá-Lillo[1]
Víctor Adsuar Abaldea[2]

1 Grupo de Arquitecturas Inteligentes Aplicadas
Departamento de Tecnología Informática y Computación
Universidad de Alicante

2 Cloud Levante SL. Centro Creación de Empresas Universidad de Alicante
Parque Científico de Alicante

Descripción y Documentación Técnica

Enero 2024

Contenidos:

1. Introducción 2

2. Paradigma Serverless 2

3. Descripción de los datos 3

4. Metodología 5

5. Representación de los Datos 8

6. Conclusiones Finales 12

Referencias 13

Dataset para análisis y detección de amenazas Denial of Wallet (DOW) en arquitecturas Serverless Computing

Resumen:

Los ataques de Denegación de Cartera (Denial of Wallet - DoW) se refieren a un tipo de ciberataque que tiene como objetivo explotar y agotar los recursos financieros de una organización desencadenando costes o cargos excesivos dentro de su entorno de computación en la nube. Estos ataques son particularmente relevantes en el contexto de las arquitecturas sin servidor (serverless) debido a características como el modelo de pago por uso, el autoescalado, el control limitado y la amplificación de costes.

La computación sin servidor, a menudo denominada función como servicio (Function as a Service - FaaS), es un modelo de computación en nube que permite a los desarrolladores crear y ejecutar aplicaciones sin necesidad de gestionar la infraestructura de servidor tradicional. Las arquitecturas sin servidor han ganado popularidad en la computación en nube debido a su flexibilidad y capacidad para escalar automáticamente en función de la demanda. Estas arquitecturas se basan en la ejecución de funciones sin necesidad de gestionar la infraestructura subyacente. Sin embargo, la falta de conjuntos de datos realistas y representativos que simulen invocaciones de funciones en entornos sin servidor ha supuesto un reto para la investigación y el desarrollo de soluciones en este campo.

El objetivo de este Dataset es crear un conjunto de datos para simular invocaciones de funciones en arquitecturas sin servidor. Además, proponemos una metodología para la generación del conjunto de datos, que implica la generación de datos sintéticos a partir de tráfico generado en plataformas cloud y la identificación de las principales características de las invocaciones de funciones.

Al generar este conjunto de datos, esperamos facilitar la detección de ataques Denial of Wallet utilizando técnicas de aprendizaje automático y redes neuronales. De este modo, este conjunto de datos podría proporcionar a otros investigadores y desarrolladores un conjunto de datos para probar y evaluar algoritmos de aprendizaje automático o utilizar otras técnicas basadas en la detección de ataques y anomalías en entornos sin servidor.

Palabras clave:

Functions as a Service, Serverless, Denial of Wallet, Botnet, Internet of Things, Mobile Cloud Computing

1. Introducción

Este conjunto de datos o Dataset constituye una herramienta para estudiar amenazas de seguridad informática en sistemas Cloud Computing bajo el modelo de servicio sin servidor o "Serverless". Este dataset se ha generado a partir de otros conjuntos de datos y herramientas públicas con el objetivo de disponer de un conjunto de datos de partida para investigar la amenaza de denegación de cartera DoW.

Las ventajas de disponer de este conjunto de datos son las siguientes:

- Este conjunto de datos podría ser útil en la detección de ataques DoW en muchas industrias y aplicaciones, incluyendo IoT (Internet de las cosas), comercio electrónico, finanzas y Fintech.
- El uso de este conjunto de datos puede mejorar significativamente la seguridad en arquitecturas sin servidor de varias maneras, como la detección de anomalías, la detección de amenazas y la mitigación.
- Este conjunto de datos puede ser relevante para el trabajo relacionado con la detección de posibles ataques DoW en arquitecturas sin servidor, ya que registra varios parámetros relacionados con las llamadas a funciones, así como otras métricas de rendimiento como el uso de memoria y CPU.
- El análisis de los datos ayudaría a identificar los parámetros necesarios para identificar posibles ataques DoW, principalmente debido a la variabilidad de los eventos y al número de invocaciones a funciones en momentos concretos de la ventana. Entre estos parámetros podemos destacar el número de funciones activas y los tiempos de retardo en la petición y la respuesta.
- El conjunto de datos puede utilizarse para entrenar y probar modelos de aprendizaje automático, como algoritmos de detección de anomalías o modelos predictivos. Estos modelos pueden reconocer patrones y comportamientos asociados a ataques DoW, proporcionando una detección temprana de amenazas. Los trabajos previos sobre detección de anomalías en ciberseguridad, como los estudios sobre aprendizaje automático y profundo para ataques de denegación de servicio, y otros tipos de ataques, pueden servir como referencia para el desarrollo de modelos [1, 2, 3].
- Los datos podrían utilizarse como referencia para predecir ataques DoW. El conjunto de datos puede utilizarse para probar y validar la precisión de los modelos predictivos.

Para maximizar su difusión, la publicación formal de este conjunto de datos se encuentra también en repositorios públicos[1] y en una publicación especializada [4].

2. Paradigma Serverless

En los últimos tiempos, está surgiendo una nueva tendencia de computación en nube entre los proveedores de servidores. Consiste en proporcionar a los usuarios la autogestión de la infraestructura para computar las aplicaciones en la nube como un conjunto de funciones [5, 6]. Así, esta tendencia se denomina Function-as-a-Service (FaaS) y representa un nuevo modelo de servicio ofrecido por el paradigma de la computación en nube. También se denomina Computación sin Servidor, ya que permite a los usuarios centrarse en las funciones y procesos sin preocuparse de la infraestructura de servidores subyacente. En su lugar, este trabajo lo realiza el proveedor del servidor, que ofrece flexibilidad y autoescalado sin fisuras para las necesidades de ejecución de funciones [7].

[1] Generation of a dataset for DoW attack detection in serverless architectures. Mendeley Data. https://data.mendeley.com/datasets/g8g9vdxyvn/1 (accedido 31/03/2024)

Esto supone un paso adelante en la abstracción de la programación en la nube y representa una nueva forma de construir "aplicaciones nativas de la nube", ya que se crean exclusivamente para su ejecución en entornos Cloud [6]. Este paradigma tiene mucha aplicación en el desarrollo de aplicaciones flexibles para numerosos ámbitos, como las ciudades inteligentes [7, 8], la industria [9], o las nuevas aplicaciones empresariales [10].

La computación sin servidor proporciona muchas ventajas sobre la infraestructura tradicional centrada en el servidor (IaaS - Infrastructure as a Service) o basada en la nube (PaaS - Platform as a Service), ya que proporciona mecanismos para optimizar la infraestructura informática, mejorar la gestión del hardware y facilitar a los desarrolladores la creación de nuevas aplicaciones.

En el área de IoT y edge computing, existen grandes esfuerzos en la investigación y desarrollo de un modelo FaaS viable [11]. Este paradigma, junto con la externalización del procesamiento a la nube [12], supone un paso importante para considerar la computación como una “utility” que permita acceder a recursos de computación ilimitados para particulares y empresas.

Sin embargo, aún quedan importantes retos por abordar [13]. Entre los problemas más relevantes está la gestión de la externalización del procesamiento [14, 15], la computación de altas prestaciones en la nube [16], el desarrollo de frameworks y la gestión del volumen de datos [17, 18], y la seguridad informática de este paradigma [19, 20].

3. Descripción de los datos

Un conjunto de datos que prediga los ataques de Denegación de Cartera/Denial of Wallet (DoW) tiene varios casos de uso y aplicaciones potenciales, como la detección temprana de amenazas, la gestión de costes en plataformas en la nube y la optimización de la asignación de recursos. Los investigadores y profesionales de la seguridad podrían utilizar este conjunto de datos para desarrollar modelos predictivos que identifiquen los ataques DoW emergentes. Esto puede conducir a la detección temprana de amenazas y a una respuesta más proactiva para mitigar los posibles riesgos financieros y operativos.

El conjunto de datos contiene invocaciones a 50 funciones diferentes durante un periodo de 24 horas y cada función proporciona el evento que la activó. Para dar más relevancia a los tipos de eventos, el conjunto de datos contiene una columna llamada **functionTrigger** que contiene los siguientes eventos posibles:

{"http", "storage", "sql", "stream", "notification", "email","nosql"}

En la Tabla 1 se indican los datos relacionados con las transacciones generadas en el conjunto de datos, y la tabla 2 muestra el nombre de cada columna junto con el tipo de datos:

Total transactions	**187087**
Attack transactions (bot=1)	**131072**
Legitimate transactions(bot=0)	**56015**
Percentage of attack transactions	**70.06 %**
Percentage of legitimate transactions	**29.94 %**

Tabla 1. Transacciones

Column	Data Type
Id	int64
IP	object
bot	bool
FunctionId	int64
functionTrigger	object
timestamp	object
SubmitTime	int64
RTT	int64
InvocationDelay	float64
ResponseDelay	float64
FunctionDuration	float64
ActiveFunctionsAtRequest	int64
ActiveFunctionsAtResponse	int64
maxcpu	float64
avgcpu	float64
p95maxcpu	float64
vmcategory	object
vmcorecountbucket	int64
vmmemorybucket	float64

Tabla 2. Tipos de datos del dataset.

Se han analizado otros trabajos e investigaciones sobre modelos serverless para identificar las métricas más relevantes para añadir al conjunto de datos [21-24]. Entre las principales métricas analizadas podemos destacar las siguientes:

- *Rendimiento de las comunicaciones*. Una aplicación serverless suele estar compuesta por varias funciones que interactúan entre sí y con otros servicios en la nube. Existen diferentes modelos de interacción entre funciones para iniciar la comunicación entre ellas, como el uso de un orquestador de funciones o el uso del SDK del proveedor de la nube.
- *Latencia de arranque para ejecutar la función*. Dado que las funciones sin servidor se inician bajo demanda, la latencia de procesamiento de cada solicitud contiene la latencia de puesta en marcha. Además, como la unidad de ejecución suele ser pequeña y de corta duración (milisegundos), la informática sin servidor puede ser más sensible a los picos de sobrecarga.
- *Tiempo de ejecución*. En las arquitecturas sin servidor utilizamos el tiempo de ejecución para saber qué funciones requieren un mayor coste computacional, y se utilizan métricas de rendimiento a nivel de hardware para obtener las instrucciones ejecutadas.
- *Uso de la CPU*. Mide la relación entre el tiempo que la aplicación pasa en la CPU, tanto en el espacio del usuario como en el del núcleo. Esta métrica ayuda a detectar aplicaciones que utilizan más recursos de los inicialmente asignados.

- *Uso de memoria*. La detección de picos de memoria es importante para determinar si la configuración es correcta, así como la facturación de las aplicaciones. También permite a los proveedores limitar el número de contenedores asignados para desplegar la aplicación.
- *Operaciones de entrada/salida*. El rendimiento medio de las operaciones de E/S del sistema de archivos y de la red disminuye con el número de llamadas a funciones.

4. Metodología

Dado que nuestro objetivo es crear un nuevo conjunto de datos, comenzamos analizando qué otras fuentes de datos tenemos disponibles dentro del ecosistema serverless que nos permita tener una base de la que partir. El conjunto de datos de prueba se ha generado a partir de las siguientes herramientas y conjuntos de datos:

- Dataset generado por el **simulador DoW** [25] cuya herramienta se puede encontrar en el repositorio de GitHub[2]. Esta herramienta permite generar datos sintéticos de tráfico normal o malicioso como parte de una botnet y podría ser utilizada en el entrenamiento de algoritmos para la detección de DoW. Esta herramienta simula el tráfico generado en las peticiones de una aplicación serverless e internamente genera miles de peticiones por segundo, por lo que podría utilizarse para simular ataques DDoS/DoW [26, 27]. Tiene la capacidad de calcular el coste de las invocaciones a funciones simulando el tráfico que éstas pueden generar. También genera datos de registro de uso para cada llamada con una etiqueta que denota si el tráfico forma parte de un bot o es legítimo. Estos datos podrían utilizarse en futuras investigaciones para diferenciar el tráfico legítimo del generado por botnets. El objetivo de esta herramienta es la generación de conjuntos de datos que contengan invocaciones de funciones en diferentes plataformas en la nube como AWS Lambda, IBM functions, Google Functions y Azure Functions y analizar si estas invocaciones son de origen botnet para la investigación de detección DoW.
- **Ejecución concurrente monte carlo serverless functions** a través de aws, google, ibm and alibaba es un conjunto de datos que contiene invocaciones de funciones en diferentes proveedores de nube [28]. El objetivo de este conjunto de datos es simular la ejecución de funciones sin servidor en proveedores de nube.
- **Conjunto de datos de Microsoft Azure**. El conjunto de datos de Microsoft Azure es una colección de 52.000 funciones que fueron invocadas 8.800 millones de veces durante un periodo de 2 semanas. Los tres objetos principales del conjunto de datos son roles, aplicaciones y usuarios, que se identifican mediante identificadores “hash” anónimos. Estos datos se dividen en tres partes principales: una serie temporal de llamadas, tiempos de ejecución y uso de memoria[3]. El objetivo de este conjunto de datos es identificar las principales métricas que podemos utilizar en entornos sin servidor para medir las invocaciones de funciones, los tiempos de ejecución y el uso de memoria.
    - *Invocaciones de funciones*: La serie temporal que contiene el número de invocaciones de una función en cada minuto de los 14 días de pruebas.
    - *Tiempos de ejecución*: El conjunto de datos contiene el tiempo de ejecución medio y un conjunto fijo de percentiles para cada función.
    - *Uso de memoria*: El conjunto de datos incluye el uso medio de memoria para cada aplicación y también se desglosa en un conjunto fijo de percentiles. A diferencia de los otros datos que se registran por función, el uso de memoria se registra para toda la aplicación porque Azure agrega la asignación de recursos para las funciones que pertenecen a la misma aplicación.

[2] Denial of Wallet Test Simulator https://github.com/psykodan/DoWTS (accedido 31/03/2024)
[3] Azure Public Dataset https://github.com/Azure/AzurePublicDataset (accedido 31/03/2024) https://github.com/Azure/AzurePublicDataset/blob/master/AzurePublicDatasetV2.md

La metodología que hemos utilizado para generar datos sintéticos a partir de invocaciones de funciones serverless, podríamos destacar los siguientes aspectos:

- Definir el formato de los datos: Determinar la estructura y los campos de los datos de la invocación a la función serverless. Por ejemplo, se podrían incluir campos como nombre de la función, argumentos pasados, respuesta de la función, tiempo de ejecución, etc.
- Configurar eventos: En una arquitectura sin servidor, las funciones se activan en respuesta a eventos específicos. Puede configurar diferentes eventos para desencadenar la ejecución de la función. Algunos ejemplos de eventos son los cambios en una cola de mensajes, las actualizaciones de una base de datos o las solicitudes HTTP.
- Generar eventos e invocaciones de funciones: Para esta tarea podemos utilizar herramientas como el simulador de pruebas DoW [25]. Esta herramienta nos permite configurar un generador de eventos que genera continuamente tráfico sintético basado en patrones para diferentes plataformas en la nube como AWS, Google Cloud, IBM y Azure.
- Almacenar los datos generados: Para facilitar el análisis, los resultados se almacenan en un archivo csv.
- Validar y analizar los datos generados: Por último, realizamos una validación de los datos generados para asegurarnos de que los requisitos son coherentes.
- Limpiar y preparar el conjunto de datos: Una vez generados los datos de invocación de funciones sin servidor, sería importante limpiar y preparar el conjunto de datos. Esta fase puede incluir la eliminación de datos duplicados, la gestión de valores nulos o la realización de transformaciones específicas en función de las necesidades del análisis posterior.

Para generar este conjunto de datos de llamadas a funciones sin servidor se han seguido los siguientes pasos.

1. Generar un conjunto de datos inicial con la herramienta Dow test simulator [25]. Este conjunto de datos inicial contiene información relacionada con las direcciones IP y los identificadores de función, junto con el indicador de si esa invocación forma parte de una botnet. También generamos una columna llamada functionTrigger que contiene el evento que desencadenó la invocación de la función. La Tabla 3 muestra las columnas generadas en el primer paso:

Id	IP	bot	FunctionId	functionTrigger	timestamp

Tabla 3. Columnas del conjunto de datos generadas en el primer paso

2. Utilizar el conjunto de datos con ejecuciones concurrentes en diferentes plataformas en la nube [28]: En esta fase el objetivo es añadir información relacionada con el tiempo de envío (submitTime), que es la duración en milisegundos (ms) que tarda una petición de red en ir desde un punto de un origen a un destino y volver al punto de partida (RTT), tiempo de retardo de la petición (InvocationDelay), tiempo de retardo de la respuesta (ResponseDelay), duración de la función (FunctionDuration), funciones activas en la petición (ActiveFunctionsAtRequest) y funciones activas en la respuesta (ActiveFunctionsAtResponse). En este paso tendríamos un conjunto de datos compuesto con las columnas del paso anterior más las columnas de la Tabla 4:

Submit Time	RTT	Invocation Delay	Response Delay	Function Duration	ActiveFunctions AtRequest	ActiveFunctions AtResponse

Tabla 4. Columnas del conjunto de datos generadas en el segundo paso

3. Utilizando el conjunto de datos que contiene información relacionada con el tipo de máquina virtual, uso de CPU y memoria en la plataforma en la nube Azure: En este caso el objetivo es añadir información relacionada con el uso máximo de CPU (maxcpu), uso medio de CPU (avgcpu), uso máximo de CPU percentil 95% (p95maxcpu), categoría de máquina virtual (vmcategory), núcleos de máquina virtual (vmcorecountbucket) y memoria de máquina virtual (vmmemorybucket). En este paso tendríamos un conjunto de datos compuesto con las columnas de los pasos anteriores más las columnas de la Tabla 5:

maxcpu	avgcpu	p95maxcpu	vmcategory	vmcorecountbucket	vmmemorybucket

Table 5. Columnas del conjunto de datos generadas en el último paso

El siguiente diagrama resume el proceso de generación de conjuntos de datos en 3 pasos:

Figura 1. Pasos para la generación del conjunto de datos

Estos son los significados de las variables introducidas en nuestro conjunto de datos:

- *IP*: Representa la dirección IP desde la que se realiza la petición.
- *bot*: Representa una variable booleana si la petición procede de una botnet.
- *FunctionId*: Representa el identificador de la función serverless.
- *functionTrigger*: Representa el evento o acción que desencadena la ejecución de la función sin servidor.
- *timestamp*: Representa el momento en el que se produce la invocación de la función sin servidor.
- *submitTime*: Representa hace referencia al momento en el que se inicia o envía la petición o evento a la plataforma serverless. Marca el momento en que la función sin servidor se activa para ejecutarse en respuesta a un evento, como una solicitud HTTP, un mensaje en una cola o un cambio en una base de datos.
- *RTT (Tiempo de ida y vuelta)*: Representa una medida del tiempo que tarda en enviarse una solicitud de un cliente a un servidor y en recibirse la respuesta del servidor al cliente.

- *Retraso de invocación*: Representa el tiempo que transcurre desde que un evento desencadena la ejecución de una función sin servidor hasta que la función comienza a ejecutarse y a procesar el evento.
- *ResponseDelay*: Representa la cantidad de tiempo que tarda la función sin servidor en producir una respuesta o completar su ejecución después de ser invocada.
- *FunctionDuration (Duración de la función)*: Representa la cantidad de tiempo que tarda una función sin servidor en ejecutarse y completar su tarea.
- *ResponseDelay*: Representa la cantidad de tiempo que tarda la función sin servidor en producir una respuesta o completar su ejecución después de ser invocada.
- *ActiveFunctionsAtRequest*: Representa el número de funciones que están activas en la petición durante una invocación sin servidor.
- *ActiveFunctionsAtRequest (Funciones activas en la petición)*: Representa el número de funciones que están activas en la respuesta durante una invocación sin servidor.
- *vmcategory*: Representa la categoría de la máquina virtual.
- *vmcorecountbucket*: Representa el número de núcleos de la máquina virtual.
- *vmcategory*: Representa la categoría de la máquina virtual.

5. Representación de los datos:

Podríamos empezar representando el histograma de las variables más representativas. La siguiente imagen representa una distribución de identificadores de función (50 funciones) en el eje x junto con los eventos generados por cada identificador de función en el eje y.

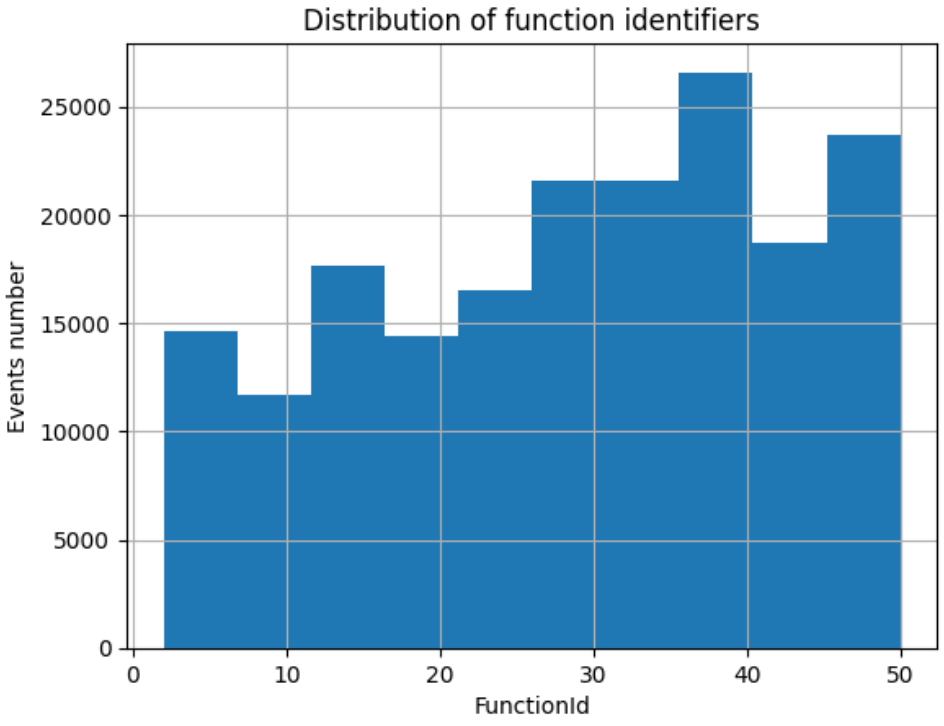

Figura 2. Distribución de los identificadores de función

La siguiente imagen representa una distribución de los tiempos de envío en el eje x junto con los eventos generados en el eje y.

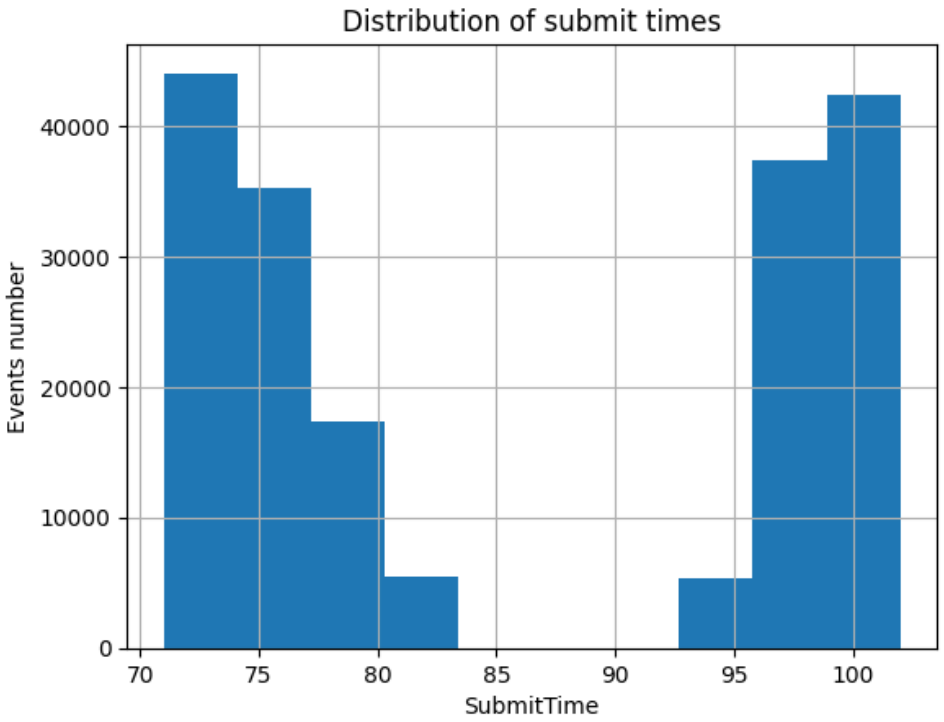

Figura 3. Distribución de los tiempos de envío

La imagen inferior representa una distribución de los tiempos de latencia de las peticiones (milisegundos) en el eje x junto con los eventos generados en el eje y.

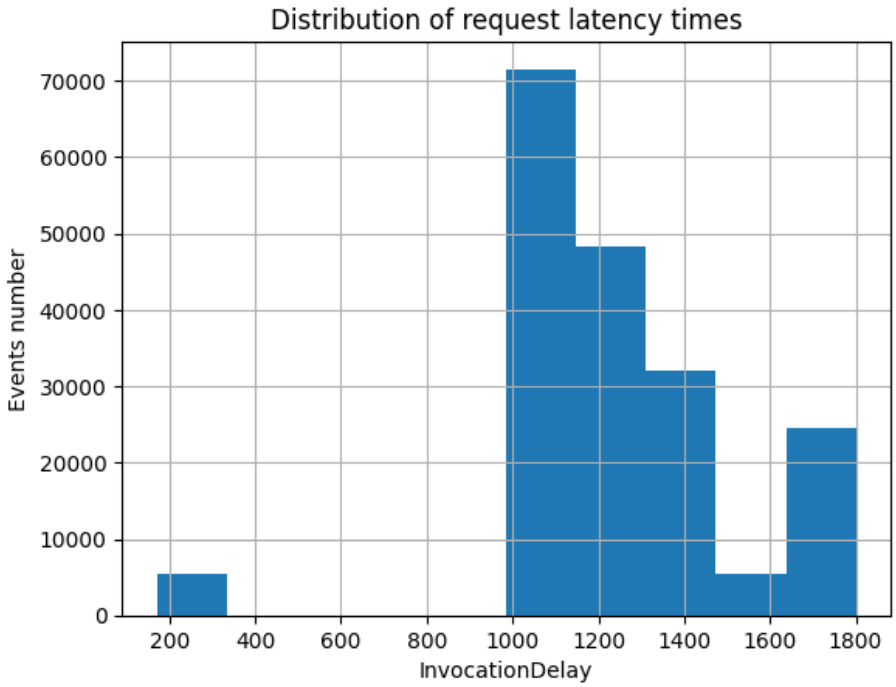

Figura 4. Distribución de los tiempos de latencia de las solicitudes

En la imagen inferior podemos ver la relación entre la variable que representa el tiempo de envío de la función (SubmitTime) y el número de funciones activas (ActiveFunctions). Podemos ver que para las transacciones de ataque ambas variables se incrementan, y podríamos separar los tipos de transacciones utilizando estas variables.

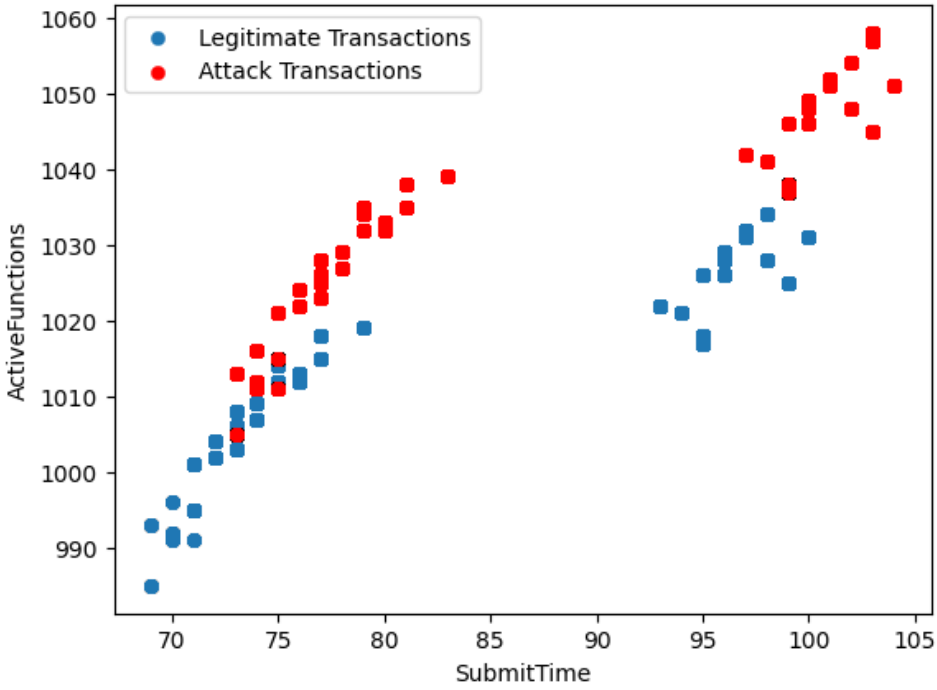

Figura 5. SubmitTime vs ActiveFunctions

En la tabla inferior vemos las estadísticas de transacciones de ataque y legítimas teniendo en cuenta la columna ActiveFunctions. Como podemos ver en la tabla, el conjunto de datos consta de 56015 transacciones legítimas (bot=0) y 131072 transacciones de ataque (bot=1).

	Transacciones legítimas	**Transacciones de ataque**
count	56015.000000	131072.000000
mean	1011.179916	1042.442177
std	10.046044	15.000330
min	975.000000	1015.000000
25%	1005.000000	1032.000000
50%	1011.000000	1043.000000
75%	1019.000000	1056.000000
max	1028.000000	1068.000000

Tabla 6. Estadísticas de ataques y transacciones legítimas

La imagen inferior muestra la relación entre el número de funciones activas por intervalo de tiempo cuando las peticiones son legítimas (bot=0) y cuando proceden de una red de bots (bot=1).

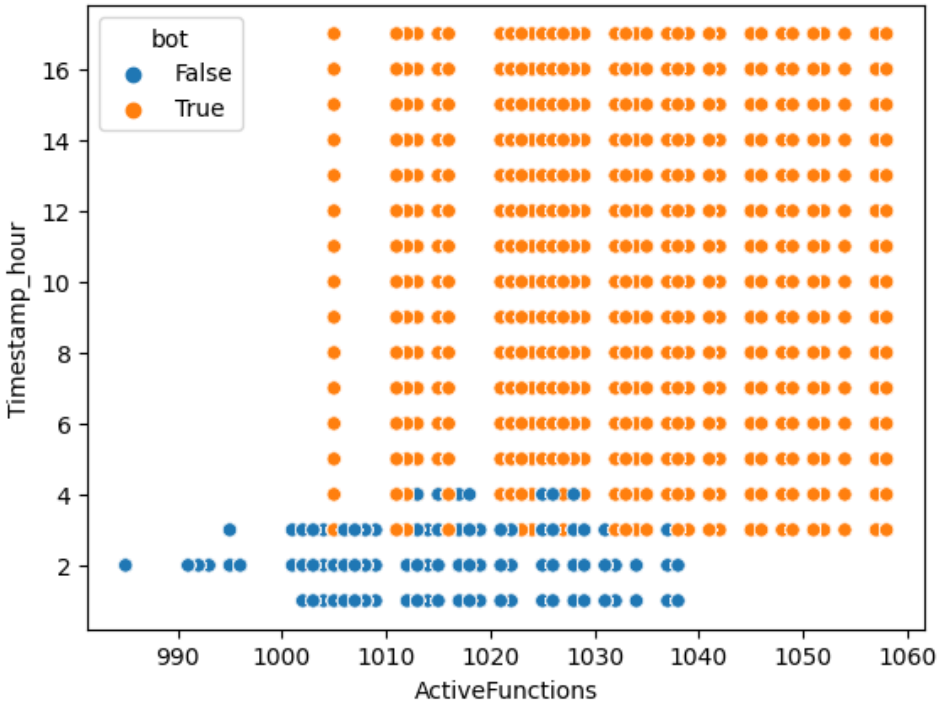

Figura 6. ActiveFunctions vs Timestamp_hour

También podríamos utilizar la variable InvocationDelay junto con la variable ActiveFunctions para diferenciar las transacciones legítimas de las que provienen de una botnet. La siguiente imagen representa la variable ActiveFunctions frente a la variable InvocationDelay utilizando la variable bot como referencia.

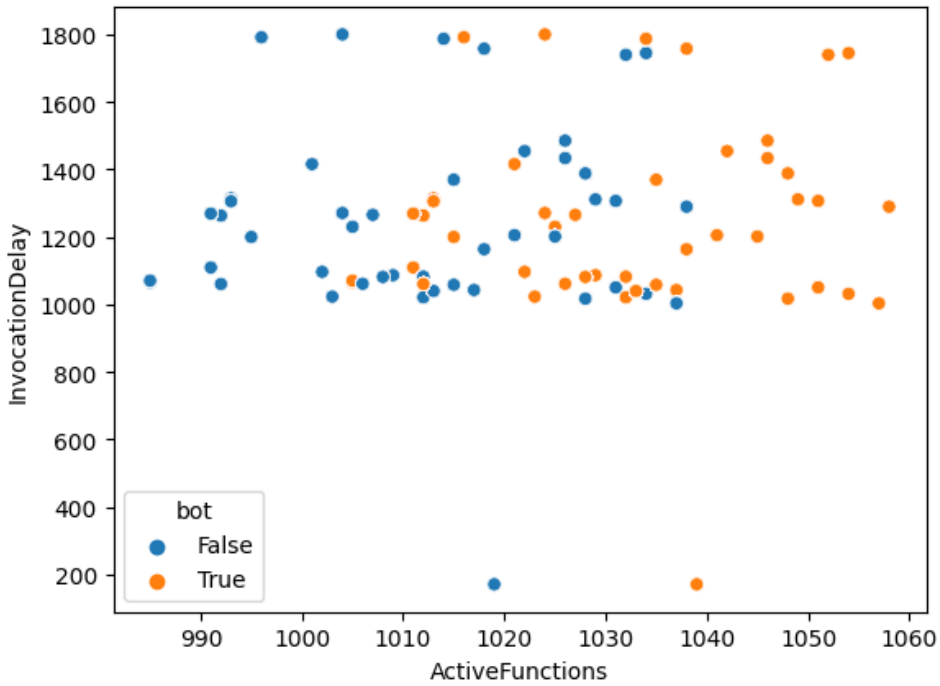

Figura 7 InvocationDelay vs ActiveFunctions

Por último, pudimos obtener estadísticas relacionadas con el número de peticiones legítimas y peticiones procedentes de una botnet, agrupadas por tipo de evento. En las imágenes inferiores podemos ver que las notificaciones son los tipos de evento más repetidos.

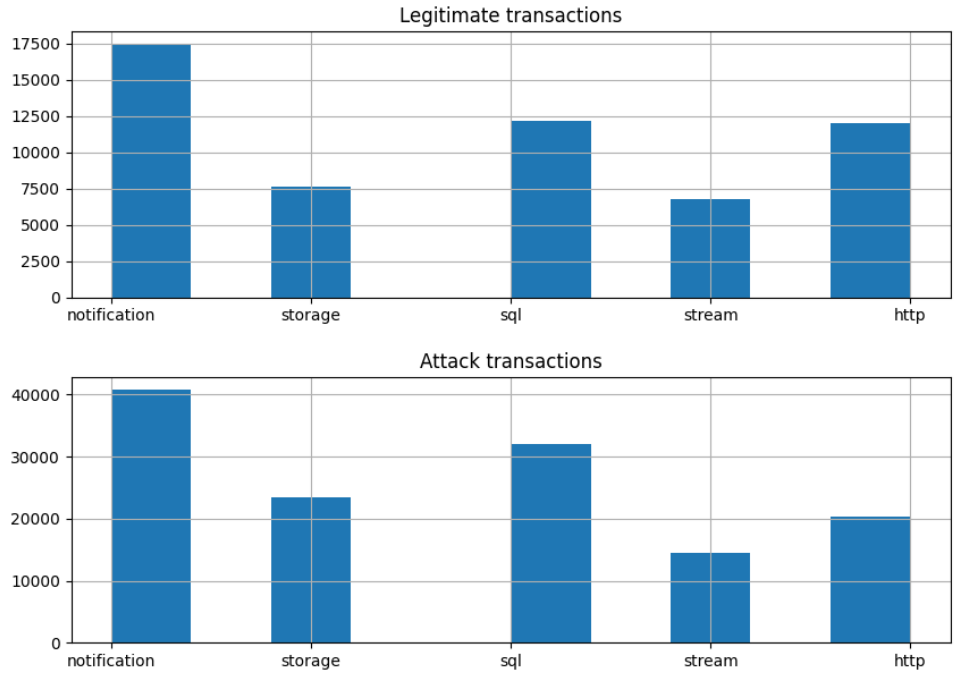

Figura 8. Legitimate vs Attack transactions

6. Conclusiones finales

En una arquitectura sin servidor, las funciones se ejecutan de forma aislada y bajo demanda en un entorno de ejecución gestionado por el proveedor de servicios en la nube. Para generar un conjunto de datos que pueda utilizarse para la detección de ataques DoW, primero simulamos llamadas a funciones en una arquitectura sin servidor y combinamos los resultados con otros conjuntos de datos que contienen información relacionada con el número de llamadas a funciones, los tiempos de envío y respuesta, así como el uso de memoria y CPU.

Este conjunto de datos puede proporcionar información valiosa y beneficios en varios aspectos de la investigación de la arquitectura sin servidor, como la optimización del rendimiento, la asignación de recursos y la gestión de costes. Por ejemplo, los investigadores pueden crear modelos de aprendizaje automático para predecir futuros costes y recursos informáticos basándose en datos históricos.

El proceso de generación del conjunto de datos ha proporcionado valiosas perspectivas y hallazgos clave que son fundamentales para reforzar nuestra comprensión de los posibles ataques de denegación de monedero (DoW) en arquitecturas sin servidor. Mediante la recopilación y el análisis de datos históricos sobre funciones activas y uso de recursos, hemos identificado indicadores importantes que pueden ayudar a predecir y mitigar estas amenazas financieras.

La investigación futura puede centrarse en mejorar el análisis del comportamiento de la carga de trabajo sin servidor. Esto podría incluir el estudio de patrones de comportamiento de usuarios y aplicaciones para detectar anomalías que puedan conducir a ataques DoW. La aplicación de técnicas de aprendizaje automático y aprendizaje profundo al análisis de comportamiento puede hacerlo más sofisticado, mejorando la identificación de este tipo de amenazas.

Agradecimientos:

Este trabajo se ha realizado en colaboración con la empresa Cloud Levante SL. como parte de las actividades de la estancia AEST del investigador Dr. Higinio Mora del grupo de investigación de Arquitecturas Inteligentes Aplicadas de la Universidad de Alicante.

Referencias

[1] Kumari, K., Mrunalini, M. (2022) Detecting Denial of Service attacks using machine learning algorithms. J Big Data 9, 56. https://doi.org/10.1186/s40537-022-00616-0

[2] Meenakshi, K. Kumar and S. Behal, (2021) "Distributed Denial of Service Attack Detection using Deep Learning Approaches," 2021 8th International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi (India) pp. 491-495.

[3] FJ. Mora-Gimeno, H. Mora, B. Volckaert, A. Atrey (2021) Intrusion detection system based on integrated system calls graph and neural networks. IEEE Access 9, 9822-9833. https://doi.org/10.1109/ACCESS.2021.3049249

[4] JM Ortega Candel, FJ Mora Gimeno, H Mora (2024) Generation of a dataset for DoW attack detection in serverless architectures. Data in Brief, 109921 https://doi.org/10.1016/j.dib.2023.109921

[5] G. A. S. Cassel, V. F. Rodrigues, R. da Rosa Righi, M. R. Bez, A. C. Nepomuceno y C. A. da Costa, (2022) Serverless computing for Internet of Things: A systematic literature review. Future Generation Computer Systems, vol. 128, pp. 299-316. https://doi.org/10.1016/j.future.2021.10.020

[6] H Mora, FJ Mora-Gimeno, A Jimeno-Morenilla, A Macia-Lillo, A Elouali. (2022) Serverless Computing at the Edge for AIoT Applications. 2022 International Conference on Artificial Intelligence of Things (ICAIoT), 1-6. https://doi.org/10.1109/ICAIoT57170.2022.10121879

[7] J. Schleier-Smith, V. Sreekanti, A. Khandelwal, J. Carreira, N. J. Yadwadkar, R. A. Popa, J. E. Gonzalez, I. Stoica y D. A. Patterson, (2021) What serverless computing is and should become: The next phase of cloud computing, Communications of the ACM, vol. 64, p. 76–84. https://doi.org/10.1145/3406011

[6] S. R. Goniwada (2022), Introduction to Cloud Native Architecture, Cloud Native Architecture and Design: A Handbook for Modern Day Architecture and Design with Enterprise-Grade Examples, Berkeley, CA: Apress. ISBN-13: 978-1484272251

[7] R Pérez-delHoyo, H Mora, JM Nolasco-Vidal, R Abad-Ortiz, Rafael A. Mollá-Sirvent (2021) Addressing new challenges in smart urban planning using Information and Communication Technologies. Systems Research and Behavioral Science 38 (3), 342-354. https://doi.org/10.1002/sres.2787

[8] H Mora, J Peral, A Ferrandez, D Gil, J Szymanski (2019) Distributed architectures for intensive urban computing: a case study on smart lighting for sustainable cities, IEEE Access 7, 58449-58465. https://doi.org/10.1109/ACCESS.2019.2914613

[9] Cinque, M. Real-Time FaaS: serverless computing for Industry 4.0. SOCA 17, 73–75 (2023). https://doi.org/10.1007/s11761-023-00360-0

[10] Lynn T., Rosati P., Lejeune A., Emeakaroha V. (2017) A Preliminary Review of Enterprise Serverless Cloud Computing (Function-as-a-Service) Platforms. IEEE International Conference on Cloud Computing Technology and Science (CloudCom). https://doi.org/10.1109/CloudCom.2017.15

[11] A. Hall y U. Ramachandran, (2019) An execution model for serverless functions at the edge, Proceedings of the International Conference on Internet of Things Design and Implementation,2019. https://doi.org/10.1145/3302505.3310084

[12] V Sánchez Ribes, H Mora, A Sobecki, FJ Mora Gimeno (2020) Mobile Cloud computing architecture for massively parallelizable geometric computation. Computers in Industry 123, 103336. https://doi.org/10.1016/j.compind.2020.103336

[13] Xie R., Tang Q., Qiao S., Zhu H, Yu FR., Huang T. (2021) When Serverless Computing Meets Edge Computing: Architecture, Challenges, and Open Issues. IEEE Wireless Communications. Vol. 28 (5). https://doi.org/10.1109/MWC.001.2000466

[14] H Mora, FA Pujol, T Ramírez, A Jimeno-Morenilla, J Szymanski. (2023) Network-assisted processing of advanced IoT applications: challenges and proof-of-concept application. Cluster Computing, 1-17. https://doi.org/10.1007/s10586-023-04050-6

[15] H Mora, D Gil, JF Colom Lopez, MT Signes Pont (2015) Flexible framework for real-time embedded systems based on mobile cloud computing paradigm. Mobile information systems. Volume 2015, Article ID 652462. https://doi.org/10.1155/2015/652462

[16] V Sánchez-Ribes, A Maciá-Lillo, H Mora, A Jimeno-Morenilla (2023) Efficient GPU Cloud architectures for outsourcing high-performance processing to the Cloud. The International Journal of Advanced Manufacturing Technology, 1-10. https://doi.org/10.1007/s00170-023-11252-0

[17] A Elouali, H Mora Mora, FJ Mora-Gimeno. (2023) Data transmission reduction formalization for cloud offloading-based IoT systems. Journal of Cloud Computing 12 (1), 1-12 https://doi.org/10.1186/s13677-023-00424-8

[18] Kritikos L., Skrzypek P. (2018) A Review of Serverless Frameworks. IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion). https://doi.org/10.1109/UCC-Companion.2018.00051

[19] JM Ortega-Candel, A Elouali, FJ Mora-Gimeno, H Mora. (2022) Cloud vs Serverless Computing: A Security Point of View. International Conference on Ubiquitous Computing and Ambient Intelligence. Lecture Notes in Networks and Systems, vol 594. Springer, Cham. https://doi.org/10.1007/978-3-031-21333-5_109

[20] JM Ortega-Candel, A Elouali, FJ Mora-Gimeno, H Mora. (2022). Serverless Security Analysis for IoT Applications. Proceedings of the International Conference on Ubiquitous Computing & Ambient Intelligence. Lecture Notes in Networks and Systems, vol 594. Springer, Cham. https://doi.org/10.1007/978-3-031-21333-5_39

[21] Pascal Maissen, Pascal Felber, Peter Kropf, and Valerio Schiavoni. (2020). FaaSdom: a benchmark suite for serverless computing. In Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems (DEBS '20). Association for Computing Machinery, New York, NY, USA, 73–84. https://doi.org/10.1145/3401025.3401738

[22] Marcin Copik, Grzegorz Kwasniewski, Maciej Besta, Michal Podstawski, and Torsten Hoefler. (2021). SeBS: a serverless benchmark suite for function-as-a-service computing. In Proceedings of the 22nd International Middleware Conference (Middleware '21). Association for Computing Machinery, New York, NY, USA, 64–78. https://doi.org/10.1145/3464298.3476133

[23] Tianyi Yu, Qingyuan Liu, Dong Du, Yubin Xia, Binyu Zang, Ziqian Lu, Pingchao Yang, Chenggang Qin, and Haibo Chen. (2020). Characterizing serverless platforms with serverlessbench. In Proceedings of the 11th ACM Symposium on Cloud Computing (SoCC '20). Association for Computing Machinery, New York, NY, USA, 30–44. https://doi.org/10.1145/3419111.3421280

[24] R. Hancock, S. Udayashankar, A. J. Mashtizadeh and S. Al-Kiswany, (2022) "OrcBench: A Representative Serverless Benchmark," IEEE 15th International Conference on Cloud Computing (CLOUD), Barcelona, Spain, 2022, pp. 103-108, https://doi.org/10.1109/CLOUD55607.2022.00028

[25] Kelly, D., Glavin, F.G. & Barrett, E. (2023) DoWTS – Denial-of-Wallet Test Simulator: Synthetic data generation for preemptive defence. J Intell Inf Syst 60, 325–348. https://doi.org/10.1007/s10844-022-00735-3

[26] G. Sujatha, Y. Kanchhal and G. George, "An Advanced Approach for Detection of Distributed Denial of Service (DDoS) Attacks Using Machine Learning Techniques," 2022 3rd International Conference on Smart Electronics and Communication (ICOSEC), Trichy, India, 2022, pp. 821-827, https://doi.org/10.1109/ICOSEC54921.2022.9951944

[27] D. Mileski and H. Mihajloska, "Distributed Denial of Wallet Attack on Serverless Pay-as-you-go Model," 2022 30th Telecommunications Forum (TELFOR), Belgrade, Serbia, 2022, pp. 1-4, https://doi.org/10.1109/TELFOR56187.2022.9983732

[28] Ristov S., Pedratscher S., Fahringer T. (2022) Concurrent execution of Monte Carlo serverless functions across AWS, Google, IBM, and Alibaba. https://doi.org/10.21227/tz10-0f93