Buenos días, mi nombre es Javier Fernández y voy a presentar el trabajo de fin de grado para tecnologías para la sociedad de la información sobre el desarrollo de estrategias de defensa para ataques de Denial of Wallet en arquitecturas Serverless.

Primero veremos lo que son los servicios serverless y definiremos los ataques de Denial of Wallet. Más adelante veremos la metodología y arquitectura diseñados para el trabajo y procederemos con los ataques investigados y las defensas diseñadas.

Serverless es un modelo de desarrollo cloud con tres características principales:

* La orientación a eventos
* El pago por uso
* La abstracción de la infraestructura física.

El desarrollador solo se tiene que centrar en el desarrollo del código y la definición del entorno de ejecución del código. Con estas dos definiciones se genera una función contenerizada que se despliega bajo demanda cuando se produce uno de los eventos definidos para ejecución de la función (llamad api, registro en BDD, etc.)

Esto permite al desarrollador centrarse en la implementación de su aplicación, elimina las tareas de gestión de infraestructura física, y reduce el coste al evitar los costes de infraestructura ociosa.

Este tipo de servicios son sensibles a un nuevo tipo de ciberataque llamado Denial of Wallet. Se trata de un ataque similar al DoS, pero el objetivo principal no es limitar el acceso a la función, sino que generar un coste económico en la victima.

Se puede principiar siguiendo distintas tipologías. Según el objeto del ataque:

* Interno: genera ejecuciones de la función.
* Externo: infiltra la máquina que aloja los servicios, ralentizando su ejecución y aumentando, por tanto, el coste.

Se puede clasificar también según el número de solicitudes enviadas y la duración del ataque, en el que tendríamos tres tipos:

* Flood: ataque corto, con un número muy elevado de solicitudes. Debido a la alta escalabilidad de las funciones serverless, son susceptibles a costes muy elevados.
* Leech: ataque muy prolongado con una tasa de solicitudes enviadas muy baja. Debido a su larga duración, puede generar costes significativos.
* Linear: número de solicitudes va aumentando progresivamente, evitando así la detección por firewalls adaptativos.

Estos ataques pueden realizarse directamente contra la API, contra el endpoint interno o podrían también pueden ser ataques distribuidos.

El objetivo de este trabajo ha sido desarrollo de estrategias de defensa para este tipo de ataques, para lo cual se han realizado tres principales tareas. Primero el desarrollo de la metodología, la definición de una arquitectura y el propio desarrollo de los experimentos y las defensas.

Se ha seguido es una metodología iterativa con las siguientes etapas:

1. establece del tipo de ataque que se quiere investigar
2. se implementa el ataque
3. se ejecuta
4. se extraen los datos para analizar el ataque.
5. se diseña e implementa una defensa para los ataques
6. se ejecuta esta defensa
7. se extraen y se analizan los

Para la arquitectura existían una serie de requisitos

* Entorno serverless.
* Recopilación de datos.
* Coste mínimo.

Por esto decidimos utilizar OpenFaaS: una tecnología de código abierto que consiste en una serie de PODs, desplegables en un nodo de Kubernetes, que lo habilitan para el alojamiento de funciones Serverless.

La arquitectura final incluye tres dispositivos: uno víctima, otro atacante, ambos conectados por cable a un switch, a su vez conectado al router, y un ultimo dispositivo que generase un tráfico base, directamente conectado al router.

En el nodo víctima se implementó una función llamada reducirImagen que consistía en una función que reducía el tamaño de una imagen de cualquier dimensión a un tamaño de 200 x 300 px. En el diagrama se pueden observar los servicios que se desplegaron dentro de la víctima para cumplir con todos los objetivos.

En el recuadro amarillo están algunos de los servicios incluidos en openfaas:

* Gateway: gestiona el ciclo de vida de la función, balanceo de carga.
* Función.
* Prometheus: recopila ciertas métricas.
* Alert Manager: gestiona las alertas configuradas sobre estas métricas.
* Kube-state-metircs: recopila más métricas.
* Prometheus (federación): reúne las métricas de las distintas fuentes.
* Grafana: recoge esas métricas para su manipulación mediante GUI.

El nodo atacante alojaba un programa escrito en Java que realizaba los ataques utilizando esta jerarquía de clases. La interfaz ICliente, IAtaque, e IExperimento permitían simplificar las modificaciones del código, y su reutilización.

Ataque Flood:

* Imagen JPG de 226 KB
* Iteraciones: 10
* Duración: 15 min
* Máximo número de solicitudes
* 10 Threads

El tamaño y formato de la imagen utilizada se decidió tras realizar un experimento preliminar donde determinamos que con un tamaño de 226KB y en formato JPG conseguíamos una mayor saturación de la función.

Gráficas:

* Iteraciones: pico inicial debido a la ociosidad de la función (no hay cola)
* Tiempo de procesamiento: aumenta a medida que se satura la función.

Ambas gráficas disminuyen en el momento de la duplicaicón.

CPU y memoria no destacable. Estabilizados, y sin grandes diferencias entre réplicas.

Para la defensa, se quiso aprovechar la infraestructura existente al máximo, y por tanto, para la detección del ataque, utilizamos el alertmanager de OpenFaaS. Se configuró una alerta en prometheus con la siguiente métrica:

rate(gateway\_function\_invocation\_total[10s]) > 1

En el alert manager se configuró un webhook que enviase una solicitud http a la url localhost:5000/block. Ahí se desplego un servidor Flask que recibía la solicitud e invocaba un script de creación de reglas de firewall que bloqueaba el tráfico malicioso.

Gráficas Defensa:

* Destacar pico inicial en invocaciones, tiempo, y CPU.
* Mencionar duplicación y eliminación de la réplica, y reserva de memoria.

Aquí tenemos la reducción concedida de todas las iteraciones y, por probabilidad, los precios y revisions del coste del ataque. En todas ellas, si consiguieron valores superiores al 90 % y de media tenemos un valor de un 97,5 % de porción reducida.

Ataque Leech (objetivo del trabajo):

* Imagen JPG de 226 KB
* Iteraciones: 10
* Duración: 1h
* 1 solicitud cada 30 - 60 s
* 1 Thread

Gráficas:

* Tiempo de procesamiento, tasa de invocación, y consumo de CPU con mayor variabilidad.
* Función no saturada.
* Consumo de memoria irrelevante.
* Nueva gráfica: 60 de tráfico base + 13

Comparación de tráficos y nueva métrica:

* Calculo de tráfico
* Variabilidad en un rango muy pequeño -> Todo o ningún tráfico
* Nueva regla de bloqueo

Gráficas defensa:

* Tiempo de procesamiento y memoria no afectado -> baja saturación
* CPU e invocaciones disminuyen con la defensa
* Solicitudes 10 minutos disminuye drásticamente

Resultados Leech:

* Iteración 5: anomalía, números muy pequeños, saturación de la CPU durante esa iteración.
* Reducción de 30%

Conclusión