DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico

Honeytokens

xx-xx

Seguridad de la Información

Integrante	LU	Correo electrónico
Melli, Tomás Felipe	371/22	tomas.melli1@gmail.com
Marco Romano Fina	1712/21	marcoromanofinaa@gmail.com
Milagros Lucía Peris	305/22	miliperis23@gmail.com
Victoria Espil	843/19	victoriaespil99@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2610 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
http://www.exactas.uba.ar

Índice

1 Introducción

Un *HoneyToken* se define como *un recurso del sistema de información falso cuyo valor radica en su uso no autorizado o ilícito* [1]. A diferencia de otros mecanismos de defensa cibernética que se centran en **negar el acceso a las amenazas**, el valor de un Honeytoken radica en su uso y cumple tres características fundamentales :

- **Identificador único**: debe tener un token de identificación único.
- **Uso indebido** : al ser un dato falso, su uso es indebido.
- **Detección inmediata** : debemos tener un sistema de detección de su uso y por tanto, al ser utilizado indebidamente, el registro es inmediato.

HoneyToken vs HoneyPot

Ya dijimos que los Honeytokens son señuelos digtales (dato, archivo o recurso falso) que su uso indebido debe ser detectado. Por otra parte, existe el concepto de Honeypot que es también un recurso del sistema falso, pero en este caso, simulan activos reales, como bases de datos,incrustados en sistemas reales, como aplicaciones o redes enteras. Su propósito es atraer y engañar a los atacantes haciendoles creer que están interactuando con un sistema genuino. Es intuitivo asegurar que estos sistemas son más complejos de desplegar y mantener que un dato falso que tiene como fin detectar su uso indebido. Y es así, construir una infraestructura que atraiga y registre las tácticas del atacante.

Importancia de la decepción

La **decepción** es un concepto fundamental en la implementación de honeypots y honeytokens. Estos sistemas son más eficaces cuando logran **convencer al atacante de que se trata de un sistema real**, logrando así que el atacante interactúe y revele sus tácticas y técnicas. Además de ocultar efectivamente su verdadera naturaleza, un honeypot debe **mantener el interés del atacante** el mayor tiempo posible, de modo que se puedan analizar y descubrir sus métodos de ataque a partir de dichas interacciones.

Limitaciones y evolución

Tradicionalmente, los honeypots son **pasivos** y, por tanto, **poco eficaces para detectar ataques nuevos o complejos**, como los realizados por actores patrocinados por estados. Si no incorporan técnicas de engaño, los atacantes pueden **identificarlos fácilmente** y evitarlos. Para aumentar su efectividad, pueden implementarse **honeypots activos**, que utilizan **respuestas engañosas e inteligentes** para hacer creer al atacante que ha comprometido un sistema real y vulnerable, otorgándole así una falsa sensación de control total.

Funciones principales

Los honeypots cumplen con tres funciones principales: **detección, prevención e investigación** [2].

Detección

Una de las mayores ventajas de los honeypots frente a otras herramientas de seguridad es su **baja tasa de falsos positivos**. Como los usuarios legítimos no interactúan con ellos, la posibilidad de detección falsa es prácticamente nula. Esta característica les permite **detectar ataques de tipo "zero-day"** con mayor precisión que otras soluciones tradicionales.

Prevención

La función de **prevención** se basa en la capacidad de los honeypots de **desviar y contener a los atacantes** lejos de los sistemas reales. Al actuar como **señuelos**, los honeypots engañan al adversario haciéndole creer que ha encontrado un objetivo legítimo, mientras que en realidad está interactuando con un entorno controlado. De esta forma, los honeypots contribuyen a **reducir el impacto y el alcance de los ataques**, al mismo tiempo que permiten reforzar la seguridad del sistema principal mediante la observación de las tácticas empleadas.

Investigación

Finalmente, la función de **investigación** está orientada a la **recopilación y análisis de información** sobre los atacantes. Los honeypots permiten estudiar en detalle las **técnicas, tácticas y procedimientos (TTPs)** utilizados, así como identificar nuevas vulnerabilidades o comportamientos emergentes. La información obtenida se utiliza para **mejorar las estrategias de defensa, ajustar políticas de seguridad y desarrollar contramedidas más eficaces**. En el ámbito académico y de ciberinteligencia, esta función es fundamental para comprender la evolución del panorama de amenazas y fortalecer la postura defensiva de las organizaciones.

Selección del tipo de honeypot

Elegir el tipo adecuado de honeypot es una decisión clave que debe tomarse tras evaluar diversos factores críticos:

1. **Estado de la red:** Es esencial analizar la topología, el tamaño y los activos críticos de la red. Un *honeypot de baja interacción* puede ser ideal para redes pequeñas o con recursos limitados, mientras que un *honeypot de alta interacción* es más apropiado en entornos complejos y con mayores recursos.
2. **Disponibilidad de recursos:** Los honeypots de alta interacción, al emular completamente sistemas reales, **requieren más hardware y personal** para su administración, a diferencia de los de baja interacción.
3. **Panorama de amenazas:** La elección del honeypot debe alinearse con los **vectores y tácticas de ataque predominantes** en la red. Diseñar honeypots que simulen las estrategias de los adversarios potenciales puede ofrecer **información valiosa** y mejorar significativamente la seguridad general del entorno.

Clasificación según el propósito

Los honeypots pueden clasificarse de acuerdo con su objetivo principal:

1. **Honeypots de Investigación:** Diseñados para **recopilar y analizar datos** sobre las técnicas, tácticas y motivaciones de los atacantes. Se utilizan principalmente por investigadores y analistas de amenazas para estudiar **nuevas vulnerabilidades y tendencias emergentes**.
2. **Honeypots de Producción:** Se integran en redes operativas reales con el propósito de **distraer y desviar a los atacantes** lejos de los sistemas críticos. Funcionan como **señuelos** que protegen los activos legítimos y mejoran la seguridad operativa general.
3. **Honeypots de Alta Interacción:** Emulan de forma realista sistemas y servicios completos, permitiendo **interacciones extensas** con los atacantes. Proporcionan información detallada sobre las técnicas y estrategias de ataque, aunque su **gestión es más compleja** y requiere mayores recursos.
4. **Honeypots de Baja Interacción:** Simulan servicios con **funcionalidad limitada**, lo que reduce el riesgo de exposición a vulnerabilidades. Aunque brindan menos datos que los de alta interacción, son **más fáciles de implementar y mantener**, siendo útiles en diversos escenarios.

Call home

Call home es el mecanismo gracias al cual cierto objeto (en este contexto, el honeytoken) envía la señal de alerta al sistema que hayamos montado para monitorearlo. Por ejemplo, el más sencillo es el acceso a un HTTP Honeytoken (url). El Cliente (atacante) hace un **GET** a la url, el servidor recibe el request y se realiza la notificación (puede ser al mail, por ejemplo como lo permite hacer Canarytokens).

References

[1] Consultado en: LEARNING CYBERATTACK PATTERNS WITH ACTIVE HONEYPOTS

[2] Consultado en: *Deception in Cybersecurity* A Comprehensive Survey on Cyber Deception Techniques to Improve Honeypot Performance

2 Primeros pasos

Cómo esperamos que funcione nuestro sistema?

La idea del sistema es que un usuario pueda, desde una **CLI (command line interface)** crear honeytokens y recibir una notificación cuando se haya detectado su uso desde el servidor. La esencia del trabajo es esa. En particular, el cliente podrá decidir qué tipo de honeytoken crear. Como consecuencia podrá, recibirlo como respuesta de la aplicación y utilizarlo dentro de su organización como un mecanismo de seguridad adicional. Lo interesante es que recibirá vía mail información sobre la persona que hace uso del honeytoken.

Herramientas

En esta sección queremos introducir las herramientas que vamos a utilizar y por qué decidimos utlizarlas:

- **Lenguaje de programación → Java**: queríamos trabajar con un lenguaje de OOP, implmentar algo de los patrones aprnedidos en `Ingeniería de Software I` y esta fue la oportunidad.
- **Spring Boot** : en particular este framework que facilita implementar un servidor que capture peticiones HTTP, entre otras herramientas importantes para implementar honeytokens (uuid, endpoint dinámico, logging,...).
- **Sendgrid** : un servicio para enviar emails desde aplicaciones, sin tener que montar nuestro propio servidor de correo.
- **Docker** : para empaquetar y ejecutar la aplicación en contenedores.

Diseño e implementación

3 Server-side

El servidor provee:

- Un mecanismo centralizado para la **creación de distintos tipos de tokens**.
- Endpoints para la **detección y activación de tokens**.
- Un caso particular de detección mediante **credenciales falsas** (fake login).

La arquitectura está basada en controladores REST, el uso del patrón **Strategy**, y servicios desacoplados para la generación de alertas.

3.1 Punto de Entrada de la Aplicación

3.1.1 HoneyTokensServerApplication

Esta clase representa el **punto inicial de ejecución** de la aplicación Spring Boot.

- Se encuentra anotada con `@SpringBootApplication`, lo que habilita:
  - Auto-configuración del framework.
  - Escaneo de componentes.
  - Configuración basada en anotaciones.
- Contiene el método `main`, encargado de iniciar el servidor embebido y el contexto de Spring.

Su responsabilidad es exclusivamente **bootstrapear la aplicación**.

3.2 Estrategias de Creación de Tokens

La creación de honeytokens se implementa mediante el patrón **Strategy**, permitiendo definir comportamientos específicos según el tipo de token sin modificar el controlador principal.

3.2.1 TokenCreationStrategy

La interfaz `TokenCreationStrategy` define el contrato común para todas las estrategias de creación:

- `create(TokenRequest request)`
- `getTokenType()`

Cada implementación indica qué tipo de token soporta y cómo debe ser creado.

3.2.2 AbstractLinkCreationStrategy

La clase abstracta `AbstractLinkCreationStrategy` encapsula la lógica común para los tokens basados en URL.

Entre sus responsabilidades se encuentran:

- Obtención o creación del usuario asociado.
- Generación de un identificador único (UUID).
- Construcción de la URL pública del token.
- Persistencia del token en la base de datos.

Las subclases únicamente definen el tipo concreto de token a instanciar.

3.2.3 Estrategias Concretas de Creación

- `UrlCreationStrategy`: crea tokens de tipo URL.
- `QrCreationStrategy`: crea tokens de tipo QR.
- `CredentialCreationStrategy`: crea tokens de tipo credenciales.
- `WordCreationStrategy`: crea tokens asociados a documentos Word.

Para el caso de **WebImageToken**, se utiliza una estrategia específica (`WebImageCreationStrategy`) que:

- Decodifica la imagen enviada desde la CLI.
- Almacena los bytes de la imagen junto al token.
- Genera una URL que devuelve el recurso visual al ser accedida.

3.3 Controladores

3.3.1 TokenCreationController

Este controlador expone un endpoint REST para la **generación de honeytokens**, es el que utiliza la CLI.

Endpoint

- **POST** `/tokens/generate`

Funcionamiento

1. Recibe un objeto `TokenRequest` que indica el tipo de token a generar.
2. Utiliza una colección de implementaciones de `TokenCreationStrategy`, inyectadas automáticamente por Spring.
3. Construye un `Map<String, TokenCreationStrategy>` donde:
   - La clave es el tipo de token en mayúsculas.
   - El valor es la estrategia correspondiente.
4. Selecciona dinámicamente la estrategia según el tipo solicitado.
5. Genera el token y devuelve un `TokenResponse` que contiene la **URL asociada al token**.

3.3.2 TokenDetectionController

Este controlador maneja la **activación de tokens** cuando se accede a una URL asociada a un honeytoken.

Endpoint

- **GET** `/home/{tokenId}/{filename}`

Funcionamiento

1. Se extrae el `tokenId` desde la URL.
2. Se consulta el `TokenRepository` para verificar la existencia del token.
3. En caso de token válido:
   - Se invoca al `AlertService` para procesar la activación.
   - Se registra información contextual del request HTTP.
4. Se selecciona una implementación de `TokenResponseStrategy` compatible con el tipo de token.
5. Se genera la respuesta HTTP correspondiente.

3.3.3 FakeLoginController

Este controlador implementa un escenario de **credenciales falsas**, simulando un login de autenticación legítimo.

Endpoints

- **GET** `/portal/login`
- **POST** `/api/v1/internal/auth`

Funcionamiento

1. El usuario accede a una página de login aparentemente válida.
2. Al enviar usuario y contraseña:
   - Se intenta interpretar la contraseña como un `UUID`.
   - Si coincide con el identificador de un `CredentialToken`, el token se considera activado.
   - Se notifica el evento al `AlertService`.
3. Independientemente del resultado, el sistema redirige al login con un mensaje de error.

Objetivo de Seguridad

- El atacante no puede distinguir si el fallo fue real o si activó un honeytoken.
- Permite detectar uso de credenciales robadas.

3.4 Servicios

El servidor de HoneyTokens utiliza servicios que desacoplan responsabilidades:

3.4.1 Gestión de Usuarios

UserService

El servicio `UserService` es responsable de la gestión de usuarios del sistema, identificados principalmente por su dirección de correo electrónico.

Su método principal, `getOrCreateUser`, implementa la siguiente lógica:

- Verifica si existe un usuario asociado al correo recibido.
- En caso afirmativo, recupera el usuario desde la base de datos.
- En caso contrario, crea y persiste un nuevo usuario.

Este enfoque permite registrar automáticamente a los usuarios que generan honeytokens.

3.4.2 Servicio de Envío de Alertas por Correo

EmailService

El servicio `EmailService` es el encargado de notificar al usuario cuando un honeytoken es activado. Para ello, utiliza la plataforma externa **SendGrid**.

Al detectarse una activación, el servicio:

- Construye un correo electrónico en formato HTML.
- Incluye información detallada del token activado.
- Agrega datos contextuales del acceso, como IP, User-Agent, URL accedida y fecha.
- Envía el correo al usuario asociado al token.

El contenido del mensaje está pensado para poder llevar un registro completo de sobre el acceso indebido.

3.5 Procesamiento de Alertas

3.5.1 AlertService

El servicio `AlertService` centraliza la lógica de procesamiento de activaciones de honeytokens.

Ante una activación, realiza las siguientes acciones:

1. Verifica si el token ya fue activado previamente.
2. Crea una nueva instancia de `Alert` a partir del `HttpServletRequest`.
3. Registra la activación en el token.
4. Persiste los cambios en la base de datos.
5. Dispara el envío del correo de alerta mediante `EmailService`.

3.6 Estrategias de Respuesta ante la Activación

Una vez que un token es activado, el servidor debe responder de forma coherente con el tipo de recurso simulado. Esta lógica se implementa mediante el patrón Strategy en las respuestas HTTP.

3.6.1 TokenResponseStrategy

La interfaz `TokenResponseStrategy` define dos métodos:

- `supports(Token token)`: indica si la estrategia aplica al token.
- `generateResponse(Token token)`: genera la respuesta HTTP.

3.6.2 Estrategias Concretas de Respuesta

- `UrlResponseStrategy`: devuelve una página HTML simulando un error 403 Forbidden.
- `QrResponseStrategy`: responde con una página HTML orientada a dispositivos móviles, simulando un QR inválido.
- `WebImageResponseStrategy`: devuelve la imagen almacenada en el token.
- `WordResponseStrategy`: devuelve una imagen estática utilizada por documentos Word como recurso externo.

Estas respuestas están diseñadas para resultar creíbles desde el punto de vista del atacante, mientras el sistema registra la activación del token.

4 Aplicación CLI para Creación de HoneyTokens – Client-side

Ya hablamos sobre lo que esperamos que haga nuestro sistema del lado del usuario, ahora vamos a abordar la arquitectura desarrollada de forma sintética.

4.1 Punto de Entrada de la CLI

4.1.1 HoneyTokensCliApplication

Esta clase constituye el **punto de entrada** de la aplicación CLI.

- Se encuentra anotada con `@SpringBootApplication`.
- Utiliza `WebApplicationType.NONE`, indicando que la aplicación no levanta un servidor web.
- Habilita el escaneo de comandos mediante `@CommandScan`.
- Define un bean de tipo `RestTemplate` para la comunicación con el servidor de HoneyTokens.

La responsabilidad de esta clase es inicializar el contexto de Spring y dejar disponibles los comandos interactivos para el usuario.

4.2 Arquitectura de Comandos de Creación

4.2.1 AbstractCreateCommand

La clase `AbstractCreateCommand` define una **clase abstracta base** para todos los comandos de creación de tokens.

- Centraliza la lógica de selección del servicio adecuado según el tipo de token.
- Recibe un `Map<String, TokenGeneratorService>` donde:
  - La clave corresponde al nombre del servicio registrado en Spring.
  - El valor es la implementación concreta de generación del token.

El método protegido `execute`:

- Busca el servicio correspondiente al tipo de token.
- Valida la existencia del servicio.
- Invoca el método `generate`, pasándole el `TokenCreationContext`.

Este diseño aplica el **patrón Strategy**, permitiendo desacoplar los comandos de la lógica concreta de generación de tokens.

Comandos Concretos de Creación

Cada tipo de honeytoken se implementa como un comando específico que extiende `AbstractCreateCommand`. Todos siguen una estructura común:

- Definición del comando mediante anotaciones de Spring Shell.
- Construcción de un `TokenCreationContext` usando el patrón **Builder**.
- Delegación de la ejecución al método `execute`.

4.2.2 CreateCredentialToken

Este comando permite generar honeytokens de tipo **credenciales falsas**.

- Comando: `generate credentials`
- Parámetros:
  - Dirección de correo electrónico.
  - Mensaje de alerta.
  - Nombre del archivo JSON a generar.

4.2.3 CreateUrlCommand

Este comando genera honeytokens de tipo **URL**.

- Comando: `generate url`
- Parámetros:
  - Correo electrónico.
  - Mensaje de alerta.

4.2.4 CreateQrCommand

Este comando permite generar honeytokens de tipo **QR**.

- Comando: `generate qr`
- Parámetros:
  - Correo electrónico.
  - Mensaje de alerta.
  - Nombre del archivo de imagen QR a generar.

El resultado es una imagen QR que apunta a una URL controlada por el servidor.

4.2.5 CreateWebImageCommand

Este comando genera honeytokens de tipo **WebImage**.

- Comando: `generate webImage`
- Parámetros:
  - Correo electrónico.
  - Mensaje de alerta.
  - Archivo de imagen a devolver al acceder al token.

4.2.6 CreateWordCommand

Este comando permite generar un honeytoken de tipo documento **Word**.

- Comando: `generate word`
- Parámetros:
  - Correo electrónico.
  - Mensaje de alerta.

4.3 Servicios de Generación de Tokens en la CLI

La aplicación CLI delega la creación concreta de cada tipo de honeytoken a un conjunto de servicios. Estos implementan la interfaz `TokenGeneratorService` y encapsulan la lógica necesaria para comunicarse con el servidor de HoneyTokens y, cuando corresponde, generar los HoneyTokens (archivos, imágenes o documentos).

4.3.1 Interfaz TokenGeneratorService

La interfaz `TokenGeneratorService` define un único método:

- `generate(TokenCreationContext context)`

Este método recibe un objeto `TokenCreationContext`, el cual contiene toda la información necesaria para la creación del token (correo, mensaje de alerta, nombres de archivos, imágenes, etc.).

Cada implementación concreta representa una estrategia distinta de generación de honeytokens, aplicando el patrón **Strategy**.

4.4 Servicios Concretos de Generación

4.4.1 UrlTokenService

El servicio `UrlTokenService` es responsable de la creación de honeytokens de tipo **URL**.

- Construye un `TokenRequest` indicando el tipo `URL`.
- Envía la solicitud al servidor mediante `TokenApiClient`.
- Recibe la URL del token generado.
- Muestra la URL resultante al usuario.

Este servicio actúa exclusivamente como un cliente del servidor, sin generarlos directamente.

4.4.2 QrTokenService

El servicio `QrTokenService` implementa la creación de honeytokens de tipo **QR**.

Su funcionamiento se divide en dos etapas:

1. Solicita al servidor la creación del token y obtiene la URL asociada.
2. Genera una imagen QR local que codifica dicha URL.

Para la generación del código QR se utiliza la librería ZXing, y la imagen final se guarda en formato PNG dentro de un directorio específico (lo llamamos `generated`).

4.4.3 CredentialsTokenService

El servicio `CredentialsTokenService` se encarga de generar honeytokens de tipo **credenciales falsas**.

El proceso consta de los siguientes pasos:

1. Solicita al servidor la creación del token y obtiene la URL asociada.
2. Extrae el identificador del token (UUID) desde la URL recibida.
3. Genera un archivo JSON que simula credenciales reales.
4. Inserta el identificador del token como contraseña administrativa.

4.4.4 WebImageTokenService

El servicio `WebImageTokenService` permite crear honeytokens de tipo **WebImage**.

- Lee los bytes de una imagen proporcionada por el usuario.
- Incluye dichos bytes como metadata en el `TokenRequest`.
- Envía la solicitud al servidor para que almacene la imagen.
- Recibe y muestra la URL del token generado.

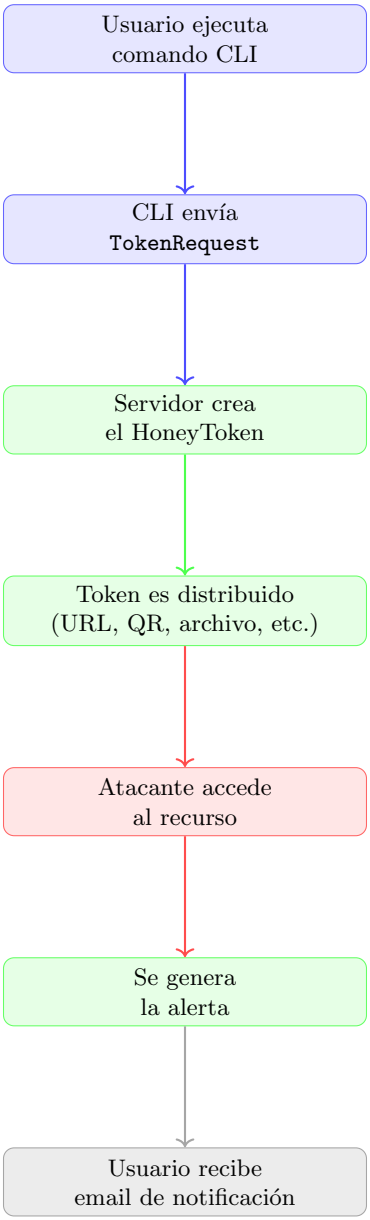
4.4.5 WordTokenService

El servicio `WordTokenService` implementa la creación de honeytokens asociados a documentos **Word**.

El mecanismo utilizado es el siguiente:

1. Se solicita al servidor la creación del token y se obtiene una URL que apunta a una imagen controlada.
2. Se genera un documento `.docx` a partir de un template existente.
3. Se modifica el archivo de relaciones internas del documento para que una imagen sea cargada desde la URL externa.

5 Flujo de funcionamiento



6 Manual de uso y Prueba

6.1 Ejecución del sistema con Docker

A continuación se describen los pasos necesarios para ejecutar el sistema utilizando contenedores Docker.

1. **Build y levantado de los contenedores**

   Desde el directorio raíz del proyecto se debe ejecutar el siguiente comando:

   ```
   sudo docker-compose up --build
   ```

   En caso de no contar con `docker-compose` instalado, puede instalarse mediante:

   ```
   sudo apt install docker-compose
   ```

2. **Acceso a la aplicación CLI**

   Una vez finalizado el proceso de build, el servidor queda en ejecución. Para interactuar con la aplicación CLI, se debe abrir una nueva terminal y realizar el *attach* al contenedor correspondiente:

   ```
   sudo docker attach honey-cli
   ```

   A partir de este punto, el usuario puede utilizar los comandos de la CLI para generar honeytokens.

6.2 Generación de HoneyTokens

Con el comando `generate` podremos crear distintos tipos de honeytokens como se ve a continuación:

```
HoneyTokens:> generate
Available commands
generate Credentials (comando para generar honeyToken tipo fake Credentials)
generate qr                  (comando para generar honeyToken tipo QR)
generate url                 (comando para generar honeyToken tipo url)
generate webImage            (comando para generar honeyToken tipo WebImage)
generate word                (comando para generar honeyToken tipo Word)
```

Con `Tab` podremos ver qué parámetros son necesarios para cada tipo. Es importante destacar que para el caso de los archivos generados, van a ir a parar a `/app/generated`.

6.2.1 URL

Como mencionamos, elegimos el tipo `url` y necesitamos definir **a qué mail enviar la alerta** y **qué mensaje queremos que aparezca** :

```
HoneyTokens:> generate url --mail "tomas.melli1@gmail.com" --message "Soy una url"
la url es http://localhost:8081/home/e2593f52-7bbd-44a4-bbef-3e94424f87ed/confidential
```

Al final, en la última sección vamos a ver cómo es el formato de la alerta cuando accedemos a esa URL.

6.2.2 QR

```
HoneyTokens:> generate qr --mail "tomas.melli1@gmail.com" --message "Soy un qr" --nameFile "QR"
QR generado en: /app/generated/QR.png
```

Para probar si funciona, podemos hacer un *drag-and-drop* del qr generado en un lector online y observar que la lectura :

Con lo cual, un acceso a ese endopint controlado por nosotros lanza la alerta.

6.2.3 WebImage

Cuando estamos trabajando en el contenedor, puede que la imagen no esté en el directorio de trabajo y va a aparecer el siguiente error :

```
HoneyTokens:> generate webImage --mail "tomas.melli1@gmail.com" --message "Soy una webimage" --image prueba.png
java.lang.RuntimeException:  No se pudo leer el archivo de imagen.
        at com.honeyTokens.honeyTokens_cli.tokensServices.WebImageTokenService.processImage(WebImageTokenService.java:57)
        at com.honeyTokens.honeyTokens_cli.tokensServices.WebImageTokenService.generate(WebImageTokenService.java:35)
        at com.honeyTokens.honeyTokens_cli.commands.createCommands.AbstractCreateCommand.execute(AbstractCreateCommand.java:28)
        at com.honeyTokens.honeyTokens_cli.commands.createCommands.CreateWebImageCommand.generateHoneyTokenWebImage(CreateWebImage
Command.java:38)
```

O sea, *no encuentra la imagen*. Para ello, la que querramos usar como `webImage HoneyToken` la vamos a copiar dentro del contenedor como sigue :

1. Nos paramos en el directorio donde esté la imagen y corremos :

```
sudo docker cp prueba.png honey-cli:/app/prueba.png
```

2. Miramos dentro del contenedor si efectivamente está :

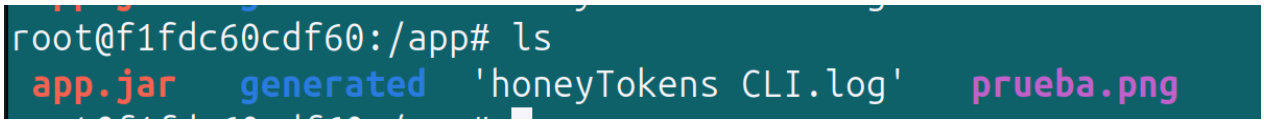
   - Buscamos el contenedor :

     ```
     sudo docker ps
     ```

   - Entramos al contenedor :

     ```
     sudo docker exec -it honey-cli /bin/bash
     ```

3. Constatamos que esté :



Ahora corremos :

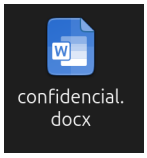
```
HoneyTokens:> generate webImage --mail "tomas.melli1@gmail.com" --message "Soy una webimage" --image prueba.png
la url es http://localhost:8081/home/5714a47e-2678-455a-8685-111228e6a5c2/image.png
```

Y la tenemos en `/app/generated` lista para utilizar.

6.2.4 Word

```
HoneyTokens:> generate word --mail "tomas.melli1@gmail.com" --message "Soy un word"
```

Como consecuencia en *app/generated* vamos a encontrar el archivo `confidential.docx` :



Es importante destacar que el honeytoken de tipo Word no se activa por la descompresión del archivo `.docx`, a pesar de que este formato sea internamente un contenedor ZIP. La activación ocurre únicamente cuando el documento es abierto por un procesador de texto compatible, como Microsoft Word. Ya que al abrir el archivo, el cliente intenta cargar un recurso externo (una imagen) cuya URL apunta al servidor nuestro.

6.3 Credentials

```
HoneyTokens:> generate Credentials --mail "tomas.melli1@gmail.com" --message "Soy una credencial falsa" --nameFile "secret"
```

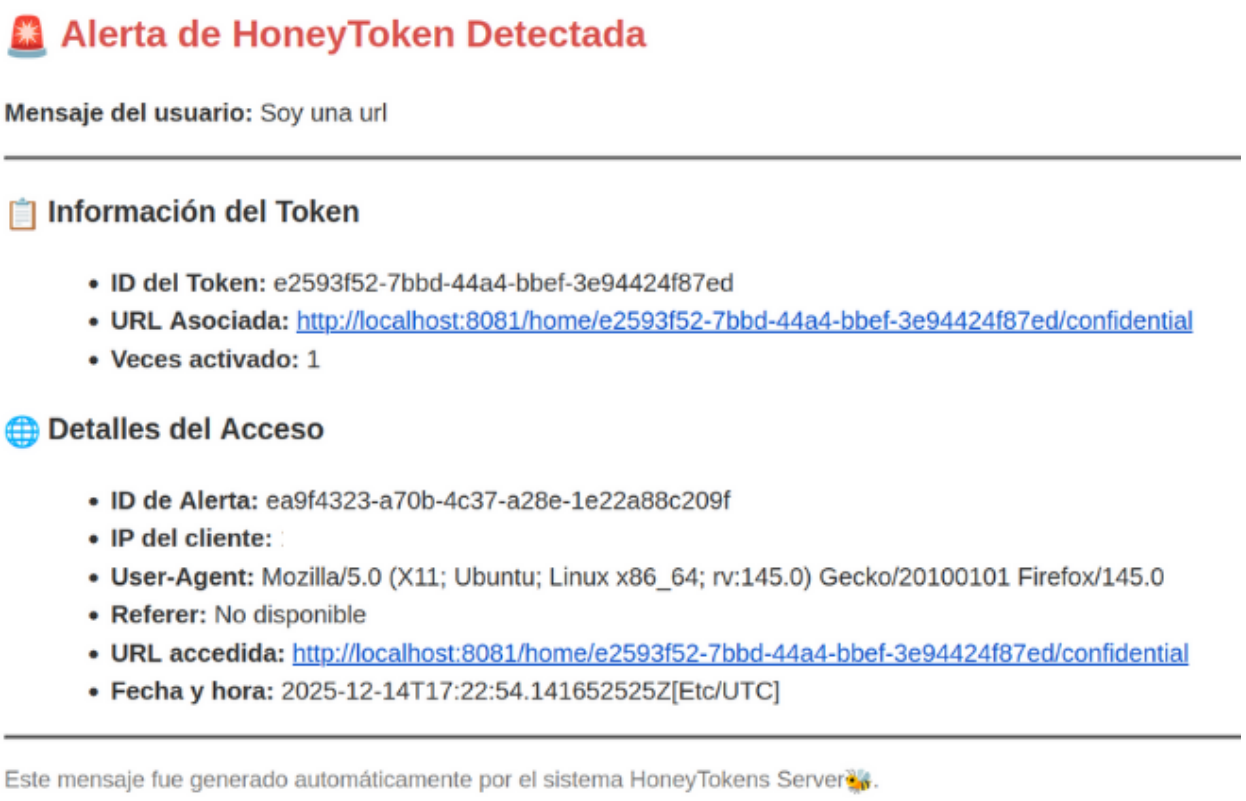
Como consecuencia en *app/generated* vamos a encontrar el archivo `secret.json` :

```
{
    "app_name": "Corporate ERP Production",
    "version": "v2.4.1-stable",
    "auth_config": {
        "portal_url": "http://localhost:8081/portal/login",
        "admin_username": "sysadmin",
        "admin_password": "23566cea-b1b8-4152-8edc-25aa4f917d5f",
        "timeout_ms": 5000
    }
}
```

Que si alguien anda chusmeando e ingresa al *fake-login* apuntado por `portal_url` con las credenciales falsas, levantará la alarma.

6.4 Detección de acceso indebido

Ahora bien, cuando se accede al endpoint que controlamos, al mail que hayamos definido para recibir la alerta obtendremos un mensaje como :



7 Conclusiones

En el presente trabajo se abordó el concepto de *honeytokens* como mecanismo de seguridad basado en la decepción, analizando tanto sus fundamentos teóricos como su aplicación práctica en un sistema funcional. A lo largo del desarrollo se evidenció que, a diferencia de las defensas tradicionales orientadas a la prevención, los honeytokens aportan valor principalmente en la **detección temprana de accesos indebidos** y en la **reducción de falsos positivos**.

Como resultado principal, se diseñó e implementó una solución completa que permite la creación y monitoreo de distintos tipos de honeytokens mediante una arquitectura cliente-servidor. El servidor, desarrollado con Spring Boot, centraliza la generación, detección y notificación de eventos, mientras que la aplicación CLI facilita su uso desde el punto de vista del usuario final. El uso de patrones de diseño como **Strategy** y **Builder** permitió lograr una solución extensible, desacoplada y fácil de mantener, alineada con buenas prácticas de ingeniería de software (agradecimientos a Ing I :)).

La implementación de múltiples tipos de tokens (URL, QR, credenciales, imágenes web y documentos Word) demuestra la versatilidad del enfoque y cómo los honeytokens pueden integrarse de forma natural en distintos escenarios reales, desde archivos aparentemente inofensivos hasta credenciales falsas. Asimismo, el mecanismo de *call home* y la notificación automática por correo electrónico permiten una respuesta inmediata ante un uso indebido, aportando información contextual valiosa para el análisis del incidente.

No obstante, es importante destacar que los honeytokens no reemplazan a otros controles de seguridad tradicionales, sino que deben ser utilizados como una **capa adicional** dentro de una estrategia de defensa en profundidad.

Como trabajo futuro, se podría extender el sistema incorporando soporte para nuevos tipos de tokens. En conjunto, el trabajo permitió no solo comprender el valor teórico de los honeytokens, sino también experimentar de forma práctica su implementación y sus beneficios dentro del contexto de la seguridad de la información.