Universidad del Valle de Guatemala
Security Data Science
Adrian Rodríguez - 21691
Daniel Gómez - 21429

**Laboratorio 4 - *Familias de Malware***

---

**Nota:** se omitieron gráficas para mantener conciso el reporte, todas las imágenes se encuentran en el repositorio y en los *Jupyter Notebooks*.

## Procesamiento y Extracción de Características de Malware

El flujo de trabajo se divide en las siguientes etapas clave:

### 1. Desempaquetado

Se implementaron funciones para detectar (is_upx_packed) y desempaquetar (unpack) archivos comprimidos con UPX. Esta etapa es crucial dado que muchos especímenes de malware emplean ofuscación para dificultar el análisis estático. El desempaquetado se realiza de manera recursiva en el directorio de malware , asegurando que todos los archivos potencialmente ofuscados sean procesados. El objetivo principal es revelar el contenido ejecutable original, permitiendo un análisis más preciso.

### 2. Extracción de Metadatos y Características PE

Se desarrollaron funciones para extraer metadatos y características de archivos ejecutables Portable Executable (PE).

- extract_file_metadata: Calcula el hash MD5 y el tamaño del archivo.
- extract_file_header: Extrae información del encabezado PE, incluyendo la arquitectura (Machine), número de secciones, marca de tiempo y características.
- extract_optional_header: Recupera detalles del encabezado opcional, como el subsistema, características DLL, tamaños de código y datos, punto de entrada y la arquitectura (32/64 bits).
- extract_section_details: Analiza las secciones del archivo, extrayendo nombres, entropías y tamaños de las secciones. Además, identifica la presencia de secciones con alta entropía y calcula la entropía promedio de las secciones.
- extract_imports: Extrae las bibliotecas de enlace dinámico (DLLs) importadas y las funciones asociadas.
- extract_security_features: Identifica la presencia de características de seguridad como certificados, información de depuración, TLS y configuración de carga.
- extract_features: Consolida todas las características extraídas en un DataFrame de pandas, indexado por el hash MD5 del archivo. Este proceso transforma los datos binarios en un formato estructurado y analizable.

### 3. Análisis Exploratorio (EDA)

Se realizó un análisis exploratorio para comprender las propiedades de los datos extraídos.

Se generaron estadísticas descriptivas (forma, valores faltantes) y visualizaciones (histogramas de tamaño y entropía, matrices de correlación, distribución de DLLs comunes).

**4. Preprocesamiento de Datos**

- Se crearon variables binarias para las secciones y DLLs presentes en los archivos (extract_section_features, extract_dll_features).
- Se derivaron métricas adicionales, como la entropía y el tamaño relativo de las secciones.
- Se realizó codificación de variables categóricas.
- Se imputaron los valores faltantes con ceros.
- Se escalaron las características numéricas para normalizar su rango.
- Se aplicó la selección de características (select_features) para eliminar características de baja varianza, optimizando el conjunto de datos final.

El conjunto de datos preprocesado se guardó en formato CSV para poder generar los clústeres posteriormente.

## Implementación

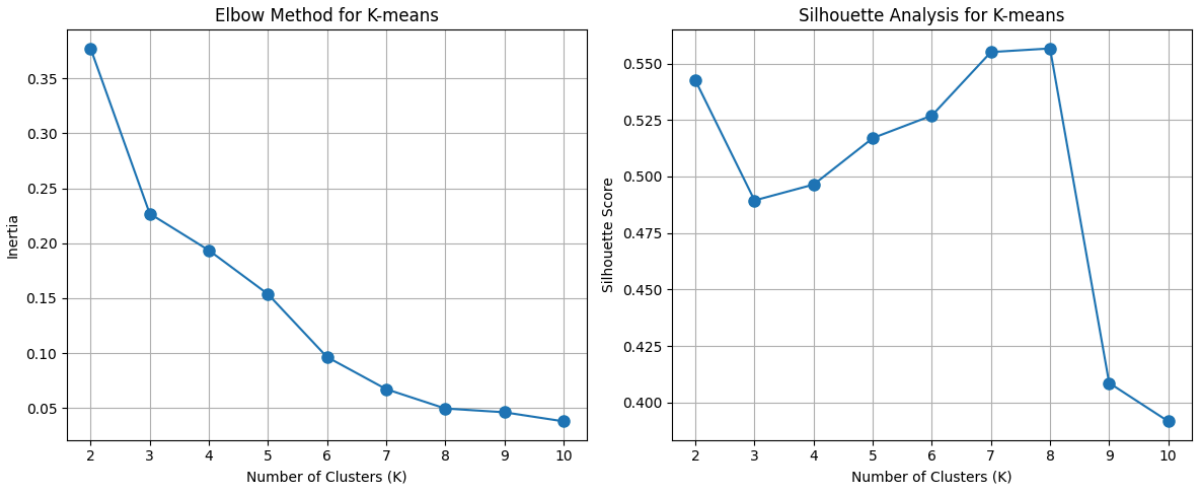
Se cargó el conjunto de datos preprocesado desde "data/processed_data.csv". Se eliminó la columna 'id' ya que no es una característica para el clustering. Se generó una columna 'feature_text' concatenando todas las características para la generación de embeddings. Se creó una columna llamada kmeans_family y kmedoids_family, para poder almacenar los resultados de los modelos.

**1. Generación de Embeddings con Gemini**

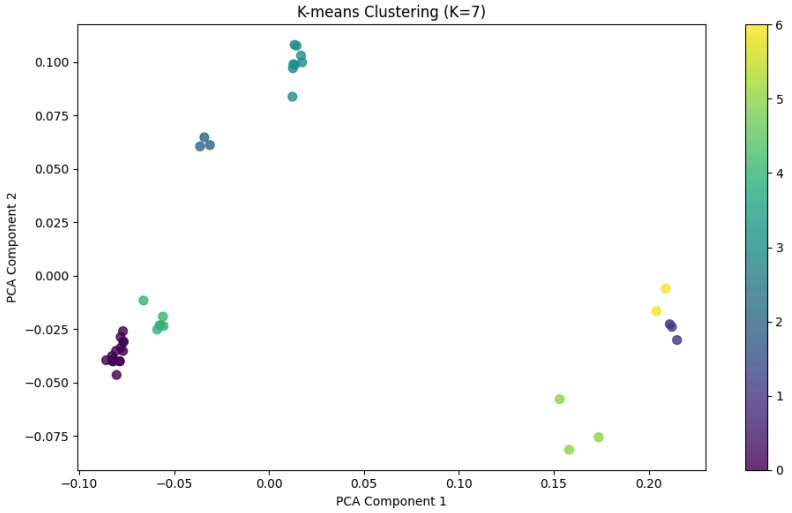
Se utilizó la API de Gemini para generar embeddings de las características concatenadas en 'feature_text'. Se almacenaron los embeddings en una nueva columna llamada 'embeddings'. Se aplicó PCA para reducir la dimensionalidad de los embeddings a 2 componentes para su visualización.

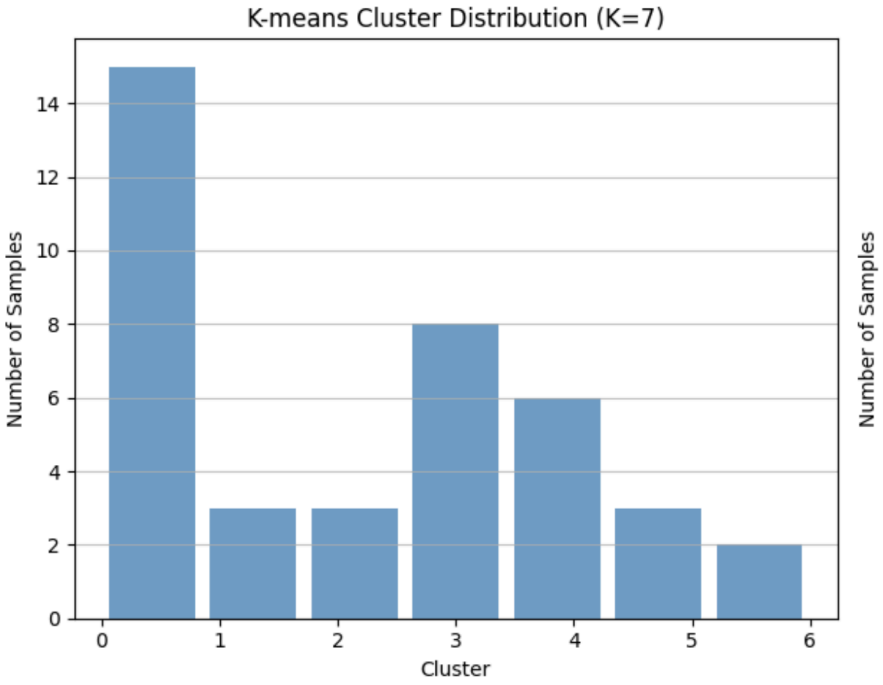
**2. Clustering con K-means**

Se aplicó el método del codo y el coeficiente de Silhouette para determinar el número óptimo de clústeres.



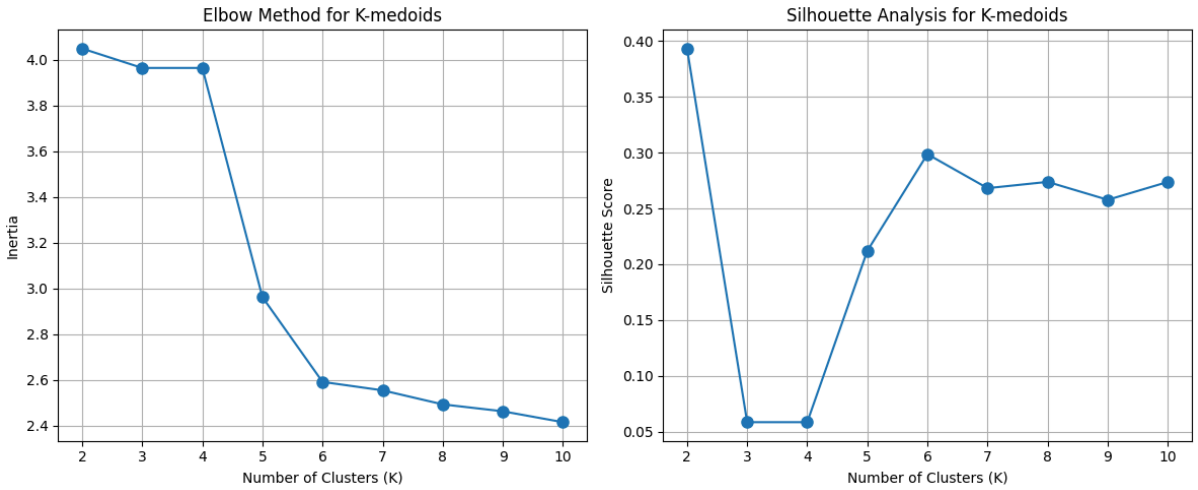

Basado en las gráficas, se seleccionó **K=7** como el número óptimo de clústeres. Se aplicó K-means con este número óptimo y se asignaron etiquetas de clúster a cada ejemplar.



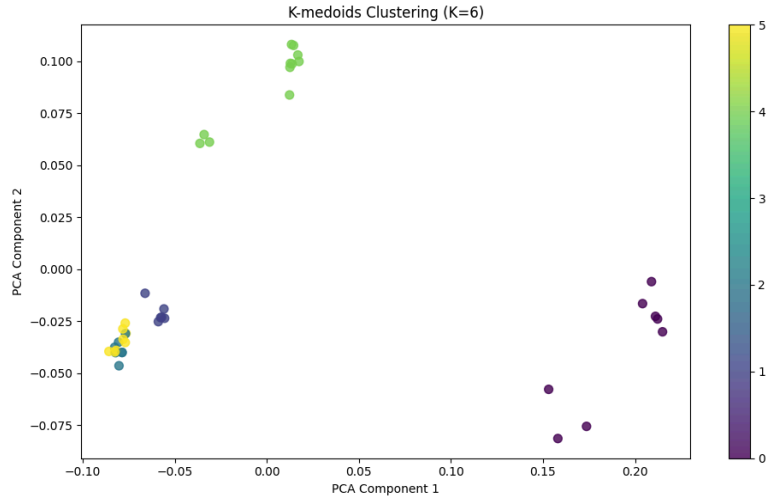


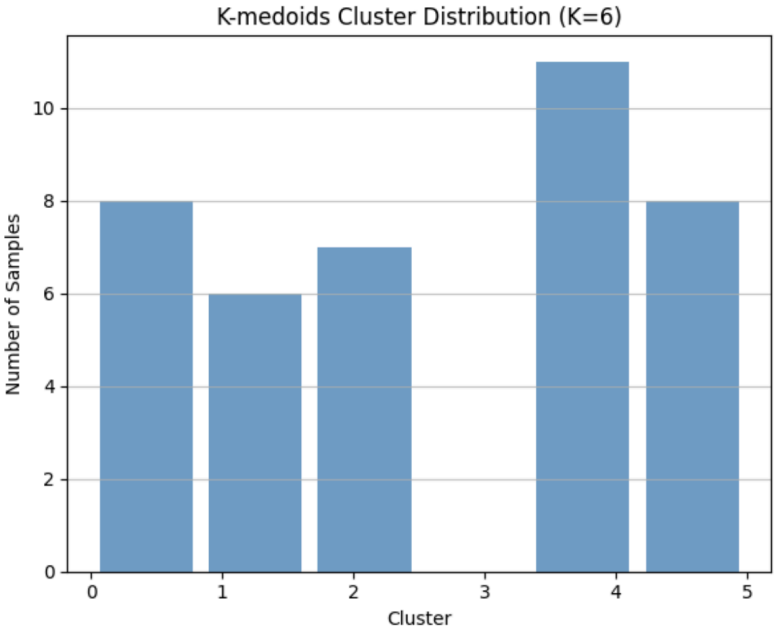

### 3. Clustering con K-medoids

Se aplicó el método del codo y el coeficiente de Silhouette para determinar el número óptimo de clústeres.




Basado en las gráficas, se seleccionó **K=6** como el número óptimo de clústeres. Se aplicó K-medoids con este número óptimo y se asignaron etiquetas de clúster a cada ejemplar.



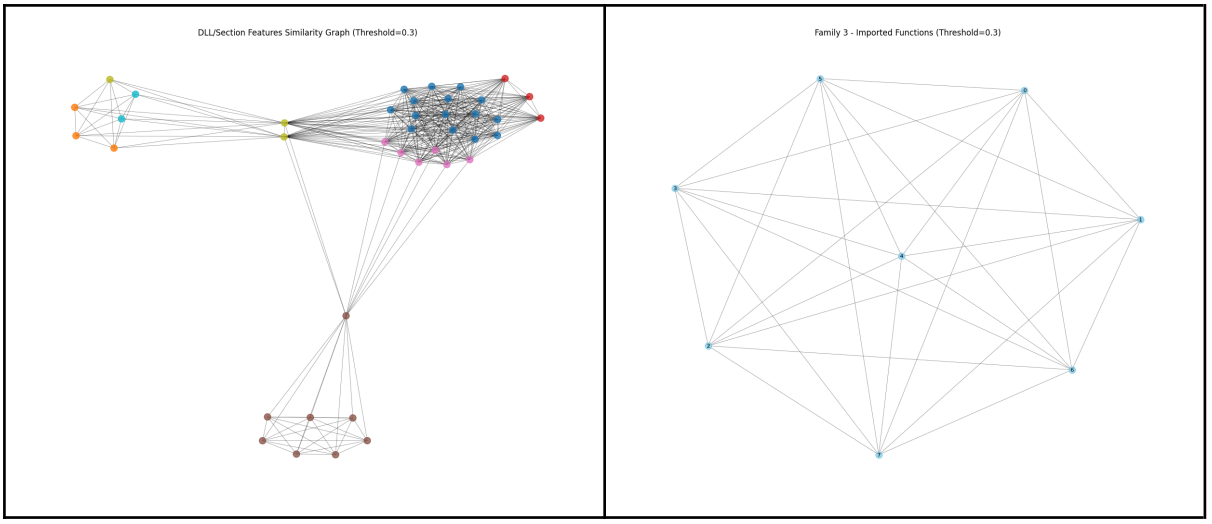




## 4. Análisis de Similitud con Índice de Jaccard

Se calcularon matrices de similitud de Jaccard para características de DLL/secciones y funciones importadas. Se crearon grafos de similitud para diferentes umbrales (0.3, 0.5, 0.7).

### Algunos Ejemplos:

| Threshold 0.3 (DLLs) | Threshold 0.3 (Funciones) |
|---|---|




## Preguntas

*¿Para qué número de clústeres se obtiene el coeficiente de Silhouette más alto?*

Para K-means, el coeficiente de Silhouette más alto se obtuvo con K=7.
Para K-medoids, el coeficiente de Silhouette más alto se obtuvo con K=6.

*¿En qué medida coincide el coeficiente de Silhouette con el método del codo?*

En ambos algoritmos, el coeficiente de Silhouette y el método del codo mostraron tendencias similares, indicando el número óptimo de clústeres.

*¿Cuántas familias cree que existen entre los ejemplares de malware proporcionados?*

Basado en los resultados de K-means y K-medoids, se sugiere que existen entre 6 y 7 familias de malware en el conjunto de datos.

*¿En qué medida coincide el análisis de similitud con las familias encontradas utilizando los algoritmos de partición?*

El análisis de similitud con el índice de Jaccard confirmó la coherencia de las familias identificadas por los algoritmos de partición, especialmente para umbrales de similitud más altos.