Práctica 2. Sistema de Razonamiento Basado en Casos (CBE) para valoración de vulnerabilidades

---

1. Descripción y objetivos

Se pretende desarrollar un prototipo de herramienta de valoración de informes de vulnerabilidad en un formato CVE (Common Vulnerabilities and Exposures) simplificado (más detalles)

- Se usará un sistema CBR para asignar valoraciones numéricas de peligrosidad [de 0 a 10] y para identificar la forma de ataque (*attack vector*) [`NETWORK`, `ADJACENT NETOWORK` o `LOCAL`] a nuevos registros "CVE" en base a las de muestras de registros "CVE" anteriores
- Se usarán os componentes de la librería CBRkit y el "miniframework" CBR presentado en en el ejemplo de tasación de vehículos

Objetivos

1. Conocer y utilizar una librería CBR para implementar un sistema sencillo
2. Implementar las fases del ciclo CBR y aplicarla a un problema con datos "reales"
3. Utilizar métricas de similaridad sobre tipos de datos complejos (cadenas, listas, jerarquías de conceptos)
4. Realizar una evaluación informal de las alternativas de implementación disponibles

---

2. Herramientas y recursos a utilizar

- Librería CBRkit , para la carga de casos y las funciones de similaridad de la fase RECUPERAR
- "Miniframework" CBR, como base para el desarrollo del sistema de valoración de vulnerabilidades
- Colección de registros CVE simplificados (base de casos y casos a resolver) y fragmento de la jerarquía CWE

---

3. Desarrollo de trabajo

3.1 Conceptos básicos

CVE (*Common Vulnerabilities and Exposures*) [wikipedia, web CVE]:

Sistema de referencia para identificar y clasificar vulnerabilidades de seguridad en software y hardware. Cada registro CVE contiene, entre otros:

1. **Identificador CVE**: Código único asignado a la vulnerabilidad, en el formato `"CVE-Año-Número"`
2. **Descripción**: Resumen breve de la vulnerabilidad: qué es, cómo afecta al sistema, qué tipo de problema representa
3. **Clasificación en CWE**: Tipo de vulnerabilidad a la que pertene la vulnerabilidad registrada, tomada de la **jerarquía CWE** (*Common Weakness Enumeration*)
4. **Productos afectados:** Listado de productos afectados por la vulnerabilidad (incluye identificación de fabricante, producto y versión) empleando el entradas del **diccionario** CPE (Common Platform Enumeration)
5. **Referencias**: Enlaces a fuentes externas con más información sobre la vulnerabilidad (informes de seguridad, blogs, documentación oficial,...)
6. **Valoración**: Detalles sobre la gravedad de la vulnerabilidad, suele estar basada en el sistema de **puntuación CVSS** (*Common Vulnerability Scoring System*).

CWE (*Common Weakness Enumeration*) [wikipedia, web CWE]

Listado de tipos de debilidades y vulnerabilidades, organizado en jerarquías que agrupan tipos de vulnerabilidades en categorías más amplias. Cada registro de CWE contiene:

1. **Identificador CWE**: Código único en formato "CWE-Número"
2. **Nombre** y **Descripción** del tipo de vulnerabilidad
3. **Relaciones** con otros CWE (`childOf`, `relatedTo`, etc)
4. **Otros datos:** consecuencias, recomendaciones de mitigación, referencias externas, etc

CVSS (*Common Vulnerability Scoring System*) [wikipedia, web CVSS]

Es un marco utilizado para evaluar la gravedad de las vulnerabilidades de seguridad. Define un formato para codificar estas valoraciones y una metodología para calcularlas y asignarlas.CVSS define tres grupos de métricas **Métricas Base**, **Métricas Temporal** y **Métricas Ambiental**.

Las **métricas base** representan las características intrínsecas de la vulnerabilidad y son las que se utilizan para calcular la **puntuación principal** (`baseScore`), incluye los siguientes aspectos:

1. **Vector de Ataque (AV)**: Cómo un atacante puede acceder al sistema (valores: `network`, `adjacent network`, `local`)
2. **Complejidad del Ataque (AC)**: Cómo de difícil es llevar a cabo el ataque (valores: `low`, `high`)
3. **Privilegios Requeridos (PR)**: Nivel de acceso que necesita un atacante (valores: `none`, `low`, `high`)
4. **Interacción del Usuario (UI)**: Se requiere o no interacción del usuario `none`, `required`
5. **Impactos** en **Confidencialidad (C)**, **Integridad (I)** y **Disponibilidad (A)** (valores: `none`, `low`, `high`)

3.2 Formatos de los casos (registros CVE simplificados)

En el sistema a desarrollar se utilizarán unos registros CVE resumidos con la siguiente estrucutra:

- `id`: identificador del registro CVE en formato `"CVE-Año-Número"` *[tipo: cadena única]*
- `assigner`: identificador de la entidad que descubrió/publicó la vulnerabilidad *[tipo: cadena única]*
- `title`: descripción breve de la vulnerabilidad *[tipo: texto libre]*
- `description`: descripción completa *[tipo: texto libre]*
- `affected_products`: listado con los nombres de los productos afectados por la vulnerabilidad (incluye el nombre del fabricante (`vendor`), con formato `<vendor>::<product>`) *[tipo: lista de cadenas]*
- `cwe`: identificador del tipo de vulnerabilidad a la que pertenece la vulnerabilidad registrada (tomado de la jerarquía CWE) *[tipo: categoría de taxonomía]*
- `keywords`: palabras clave extraidas del campo `description` utilizando la librería de procesamiento del lenguaje SpaCy
- `metric`: versión resumida de un registro *CVSS* versión 3.x. Son los **atributos a predecir** con el sistema CBR. Incluye:
  - `score`: valor del `baseScore` de CVSS
  - `attackVector`: valor del `attackVector` de CVSS

Ejemplo: (entrada en CVE.org y NVD)

```
  {
    "id": "CVE-2022-38434",
    "assigner": "adobe",
    "title": "Adobe Photoshop SVG File Parsing Use-After-Free Remote Code Execution
Vulnerability",
    "description": "Adobe Photoshop versions 22.5.8 (and earlier) and 23.4.2 (and earlier) are
affected ...",
    "affected_products": ["Adobe::Photoshop"],
    "cwe": "CWE-416",
    "keywords": ["a Use", "Adobe Photoshop SVG File", "Free Remote Code Execution",
"Exploitation", ... ],
    "metric": {
      "score": 7.8,
      "attackVector": "LOCAL"
    }
  }
```

3.3 Datos a utilizar

Los datos a utilizar en la práctica han sido descargados y procesados desde el volcado de registros CVE disponible en el repositorio GitHub "cvelistv5" y de los volcados de CWE de CWE downloads

- Colección orginal . 12759 registros CVE simplificados (sin particionar)
- Base de casos. 12659 registros CVE simplificados extraidos del anterior para confirmar la base de casos
- Casos a resolver. 100 registros CVE simplificados para funcionar como *casos a resolver*
- Jerarquia CWE (view 1000). YAML con la jerarquía CWE conforme a la *vista* "Research Concepts"

3.4 Ciclo CBR a implementar

Para implementar el sistema de valoración de vulnerabilidades se proponen las siguientes estrategías para cda fase del ciclo CBR

Fase RECUPERAR

En esta fase se utilizarán los módulos de CBRkit `cbrkit.retrieval` y `cbrkit.sim` para implementar la recuperación de casos similares.

CALCULO de la SIMILARIDAD (punto clave)

En lo que respecta a los atributos de los **registros CVE simplificados**:

- Atributo `assigner`: es una cadena "escalar" (no texto libre, ni pertenece a una jeraquía), puede utilizarse una comparación simple (`cbrkit.sim.generic.equality`) o alguno de los métodos de comparación aproximada entre cadenas (`levenshtein`, `jaro` o similar de `cbrkit.sim.strings`)
- Atributo `affected_products`: es una lista de cadenas "escalares", puede utilizarse una métrica para colecciones de `cbrkit.sim.collections` como `isolated_mapping`
  - Es necesario aportar una **métrica de similaridad** para los elementos **individuales**
  - En este caso sería recomendable uno de los métodos de comparación aproximada de cadenas (`levenshtein`, `jaro` o similar) para capturar las similaridades entre nombres de versiones de un mismo software o de herramientas relacionadas
- Atributo `cwe`: es una cadeda "escalar" tomada de una jerarquía de conceptos (fichero `jerarquia_cwe_1000.yam`) y debe utilizarse uno de las similaridades jerárquicas de `cbrkit.sim.strings.taxonomy`
- Atributos `title` y `description`: ambos son cadenas de texto en formato libre, puede utilizarse alguno de los métodos de proximidad semántica entre textos de `cbrkit.sim.strings`

  Nota: puede omitirse `title` y usar sólo `description`, ya que suele estar incluido en `description`
  - `spacy`. Utiliza la función de similaridad de la librería spaCy [detalles en Word vectors and semantic similarity]
    - Es necesario indicar el nombre del modelo a utilizar (modelos para inglés)
    - Para ejecución en CPUs hay disponibles: `en_core_web_sm`, `en_core_web_md` y `en_core_web_lg`
    - Es necesario descargarlos manualmente antes de utilizarlos (basta con instalar sólo el modelo a emplear)

      ```
      pip install spacy
      python -m spacy download en_core_web_sm
      ó
      python -m spacy download en_core_web_md
      ó
      python -m spacy download en_core_web_lg
      ```

    - **Importante**: el cálculo de similaridad usando este método es **extremadamente lento**, se recomienda no emplearlo durante la depuración, dejándolo sólo para las pruebas finales o empleando en su lugar la similaridad entre los atributos `keyword`
  - `sentence_transformers`. Usa la librería SentenceTransformers para crear vectores densos que resumen la semántica del texto y luego compararlos usando la métrica del coseno
    - Es necesario indicar el nombre del modelo a utilizar (modelos disponibles)

        - **Importante:** requiere GPU (propia o via Google Colab) para poder utilizarse **[NO RECOMENDADO]**
    - `openai`. Usa el API de *embeddings* de OpenAI
        - Es necesario indicar el nombre del modelo a utilizar (modelos disponibles)
        - **Importante:** requiere tener un *token* de desarrollador de OpenAI **[NO RECOMENDADO]**

- Atributo `keywords`: lista con palabras clave extraídas automáticamente de los atributos `title` y `description`
    - Se propone como **alternativa más rápida** al uso de las similaridades `spacy` o `sentence_transformers` sobre el atributo `description`
    - La similaridad entre listas `keywords` puede defirnse usando la métricas para colecciones `jaccard` o similares de `cbrkit.sim.collections`

Fase REUTILIZAR

Esta fase se debe implementar desde cero.

Se trata de predecir los valores de los atributos `metric.score` (problema de regresión) y `metric.attackVector` (problema de clasificación).

- Para predecir `score` se propone calcular la media ponderada de los valores `metric.score` de los *casos similares* recuperados
- Para predecir `attackVector` se propone usar el valor más repetido (moda) de los valores `metric.attackVector` de los *casos similares* recuperados

Fase REVISAR

Esta fase se debe implementar desde cero.

- Se simulará la revisión del *caso resuelto* utilizando los valores reales de los atributos a predecir (`score` y `attackVector`)

Se aplicarán las siguientes **reglas** para decidir si un *caso resuelto* es éxito o fracaso y para corregirlo:

- Se considera un caso de **éxito** si se predice correctamente **al menos uno** de los valores `score` o `attackVector`
    - se considera que la predicción de `score` es correcta si la diferencia con el valor real es inferior a un *umbral* parametrizable
    - se considera que la predicción de `attackVector` es correcta si coincide con el valor real
- En **todos los casos** (`score` y `attackVector`) se corrigen los valores que no se consideren correctos según las condiciones anteriores

Fase RETENER

Esta fase se debe implementar desde cero.

Se aplicarán las siguientes reglas para decidir si retener o no un *caso revisado*.

- Los casos en lo que se haya corregido alguno de los atributos a predecir se incorporarán a la base de casos

- Los casos de éxito que hayan predicho correctamente el atributo `attackVector` se incorporarán **siempre** a la base de casos

5. Tareas a realizar

Las tareas concretas a realiza en la práctica serán las siguientes:

1. Diseñar la métrica (o métricas) de similaridad de casos a utilizar atendiendo a las indicaciones del apartado anterior y a las caracterísiticas de los atributos presentes en los registros "CVE simplificados"
    - Prestar especial atención al tratamiento del atributo `description` (o de `keywords` en caso de no utilizar `description`)
2. Implementación Python del sistema CBR
    - Crear una clase herede de la superclase `core.CBR` del "miniframework" disponible en GitHub (Ejemplo tasación de vehículos)
    - Replicar la estructura de la clase `tasador.TasadorCBR` adaptándola al problema de la valoración de registros CVE (inicialización del *caso a resolver*, método *prettyprint*, etc)
    - Implementar las funciones responsables de las fases RECUPERAR, REUTILIZAR, REVISAR y RETENER según las indicaciones del apartado anterior
3. Ejecución del sistema y pruebas
    - Realizar algunas pruebas informales, por ejemplo con un par de ponderaciones diferentes de la similaridad combinada del caso o eliminando algún atributo

---

6. Normas de entrega

Trabajo **individual** o en **parejas**

- **Importante:** ambos miembros del grupo deberán subir su entrega a la tarea de Moovi
- Una vez entregado en Moovi se habilitará un cuestionario de Moovi de 6-7 preguntas de respuesta múltiple (con 4 opciones posibles y penalización por fallos [3 fallos = 1 correcto])

Entregable:

- Ficheros Python que conforman el sistema desarrollado (incluido el `core.py` del "miniframework" CBR)
- Memoria **breve** con la siguiente estructura:
    - Descripción del problema
    - Descripción de la implementación
        - Métricas de similaridad empleadas: tipos de métricas, pesos utilizados, etc
        - Detalles de implementación de las fases del ciclo CBR
    - Ejemplo de funcionamiento y descripción de las pruebas realizadas
    - Conclusiones y problemas encontrados (posibles mejoras, idoneidad de la herramienta, etc)
    - Bibliografía consultada

  Nota: Incluir nombre, DNI y e-mail de todos los alumnos del grupo en la portada

Fecha de entrega: Hasta domingo, **8/12/2024**, 23:50