Diego Esteban Quintero Rey
Modelos estocásticos y simulación en computación y comunicaciones
ME01
Proyecto Final: Implementación de un enfoque basado en teoría de juegos para contrarrestar amenazas persistentes avanzadas

DISEÑO

FLUJO LÓGICO Y ARQUITECTURA DE LA APLICACIÓN

En el diagrama los cuadrantes en color verde/aguamarina y color morado/lila son funciones. Los cuadrantes en color blanco y amarillo/naranja son variables, o es decir, entradas y salidas de las funciones.

Las funciones en color morado/lila son las que se utilizan en el experimento del artículo [1]. Las variables en color amarillo/naranja son las mínimas necesarias para iniciar un experimento como el del artículo [1]. Esto significa que esa sección del diagrama corresponde a la parte de la aplicación que no depende de la topología del sistema.

Por su parte toda la región que cubre las funcione en color verde/aguamarina conforman la parte de la aplicación que es dependiente de la topología del sistema. El usuario puede decidir qué tipo de experimento ejecutar definiendo los parámetros correspondientes tal como se explica en el manual de usuario y técnico.

Las variables marcadas en color rojo corresponden a aquellas que podrían definirse como parámetros de la aplicación en una futura versión. Estos son: número de expertos, número de categorías, máximo orden de las derivadas, umbral de riesgo.

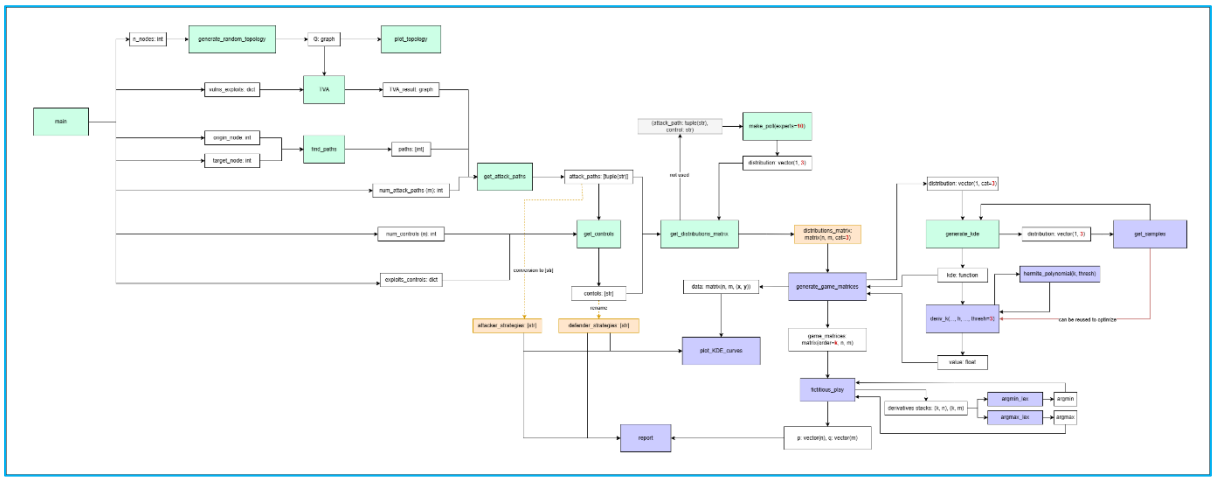


Ilustración 1 - Flujo y arquitectura de la aplicación

A continuación se muestra el mismo diagrama en tres secciones separadas para una mejor visualización de los detalles.

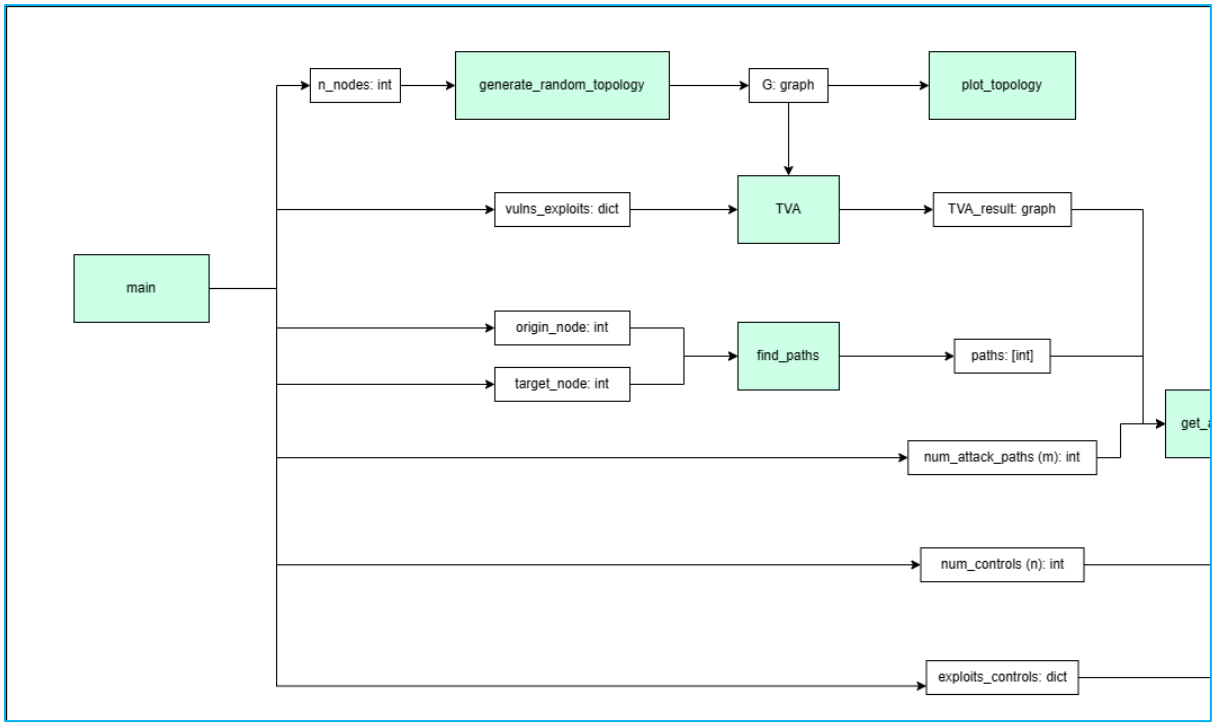


Ilustración 2 - Flujo y arquitectura de la aplicación - Parte 1

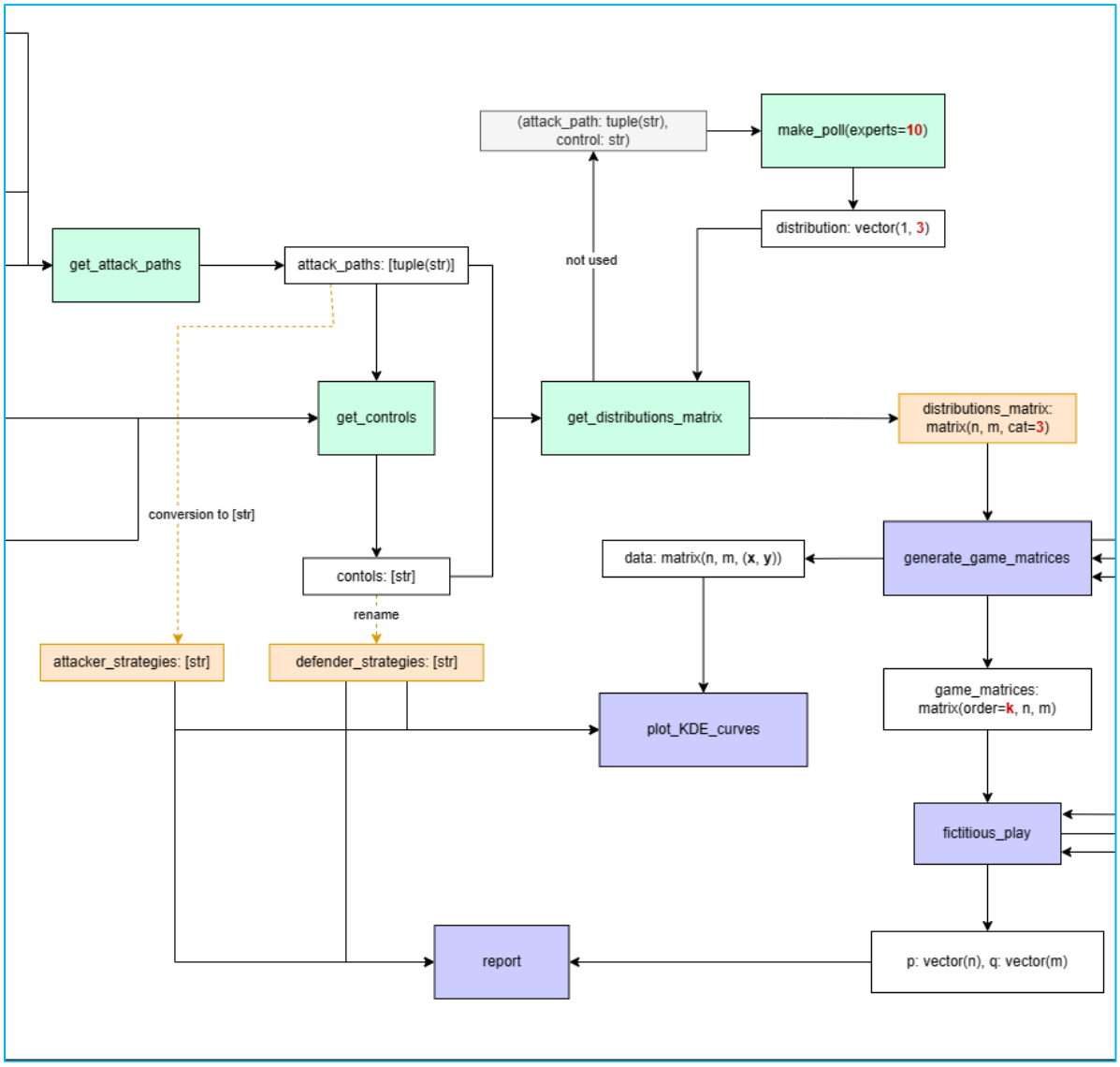


Ilustración 3 - Flujo y arquitectura de la aplicación - Parte 2

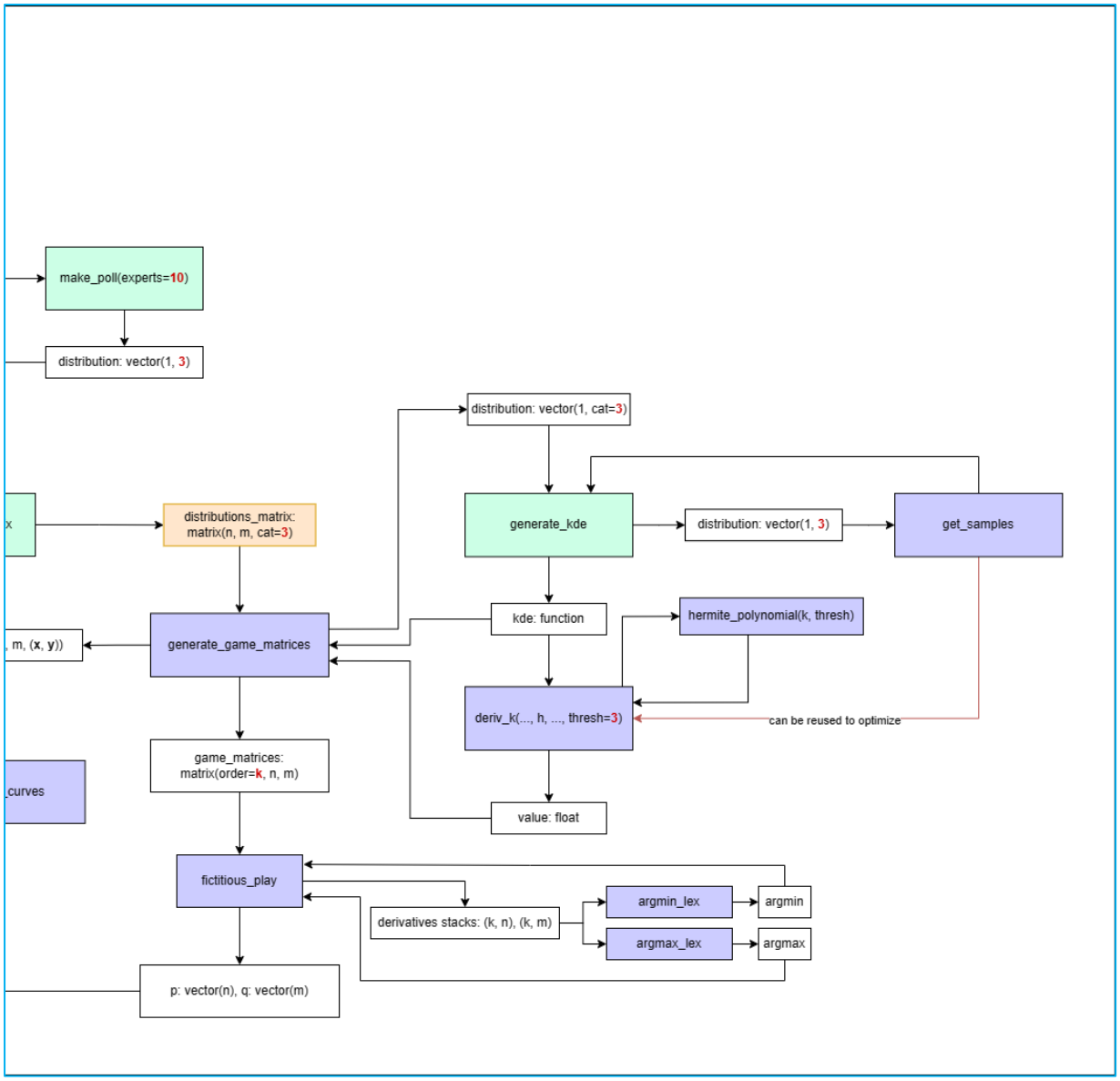
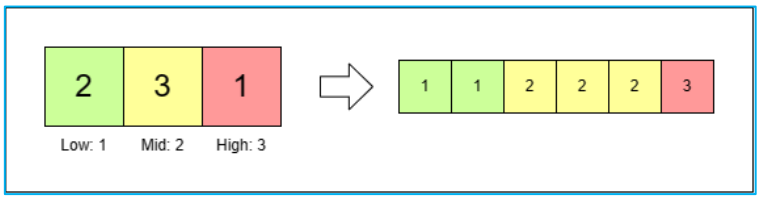


Ilustración 4 - Flujo y arquitectura de la aplicación - Parte 3

DISEÑO DE LAS FUNCIONES

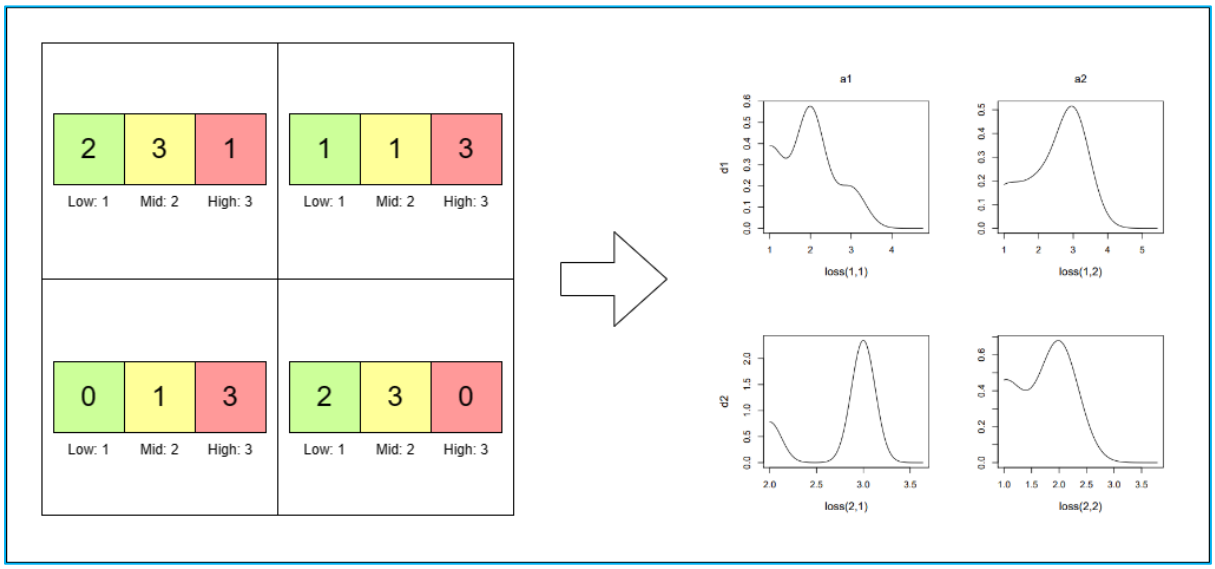
Get Samples

Esta función genera un arreglo de muestras a partir de un arreglo de frecuencias. El arreglo de frecuencia se obtiene de la función make_poll en esta implementación, con diez expertos. Se simula aleatoriamente que cada experto hace una votación del nivel de pérdida de la combinación (defensa, ataque) y así se genera el arreglo de frecuencias que posteriormente alimenta la función que genera la KDE y la función get_samples para obtener el vector de muestras L_{ij}.



Matriz de distribuciones

La siguiente imagen muestra el ejemplo de una matriz de distribuciones. Cada celda es una distribución generada tras la votación de los expertos. Tras aplicar la KDE a cada entrada se obtiene la función que puede ser graficada como se ve a la derecha.



Juego Ficticio "Paralelo" y Ordenamiento Lexicográfico

En esta imagen se ilustra la lógica detrás del ordenamiento lexicográfico y la implementación en "paralelo" del juego ficticio. En realidad, no está implementada en paralelo, pero podría hacerse.

Las matrices de la derecha son las que reciben las funciones argmin_lex y argmin_max respectivamente. Estas matrices son de tamaño (k, n) y (k, m) respectivamente. Donde k es el orden máximo de derivadas, y n, m el tamaño de filas y columnas respectivamente. Cada columna en esta matriz contiene todas las derivadas evaluadas en el umbral de riesgo a, de la KDE de la fila o columna, correspondiente.

El objetivo es comparar estas columnas, vectores de derivadas, lexicográficamente para obtener la fila mínima o la columna máxima. Al algoritmo de juego ficticio solo le interesa los índices, y por eso se llamaron argmin_lex y argmax_lex.

Una vez el algoritmo de juego ficticio tiene los índices hace las actualizaciones correspondientes de los acumuladores de filas y columnas. Pero aplica esto en todas las k matrices. Esta es precisamente la parte que podría paralelizarse.

De esta manera es como si se estuviera jugando el juego en todas las matrices simultáneamente, pero con un mismo conjunto de estrategias para todas las matrices. El objetivo de las matrices de derivadas de orden superior es desempatar cuando hay un conflicto (dos valores iguales) en la matriz de orden 0. Idealmente las matrices se deben crear a demanda, según se necesiten, y jugar el juego con las mismas reglas de la matriz de orden 0 hasta llegar a la iteración en la que se presentó el conflicto. Pero en este diseño, se construyeron $k = 20$ matrices desde el inicio y se tienen en memoria durante todo el algoritmo.

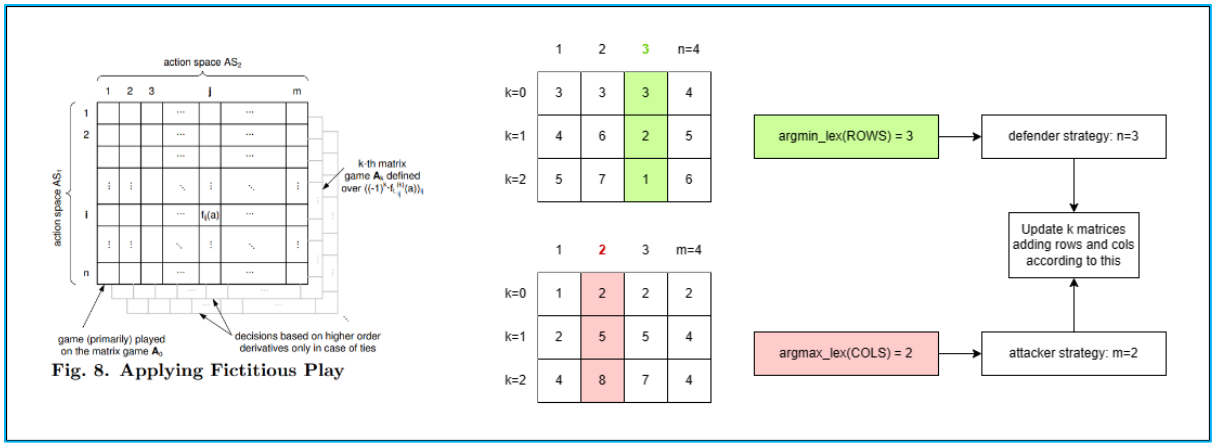
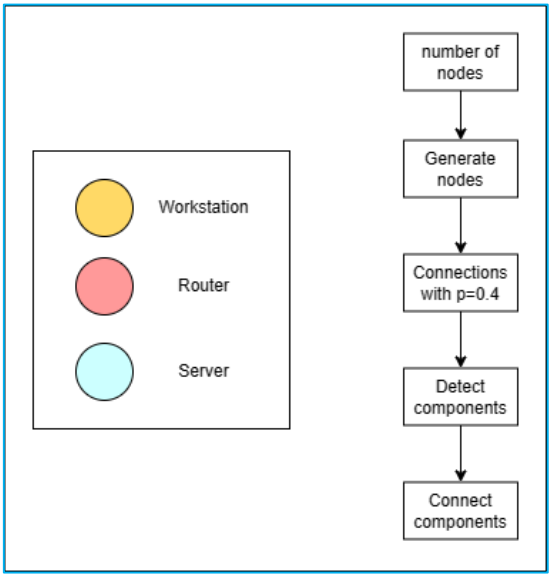


Fig. 8. Applying Fictitious Play

Topología Aleatoria

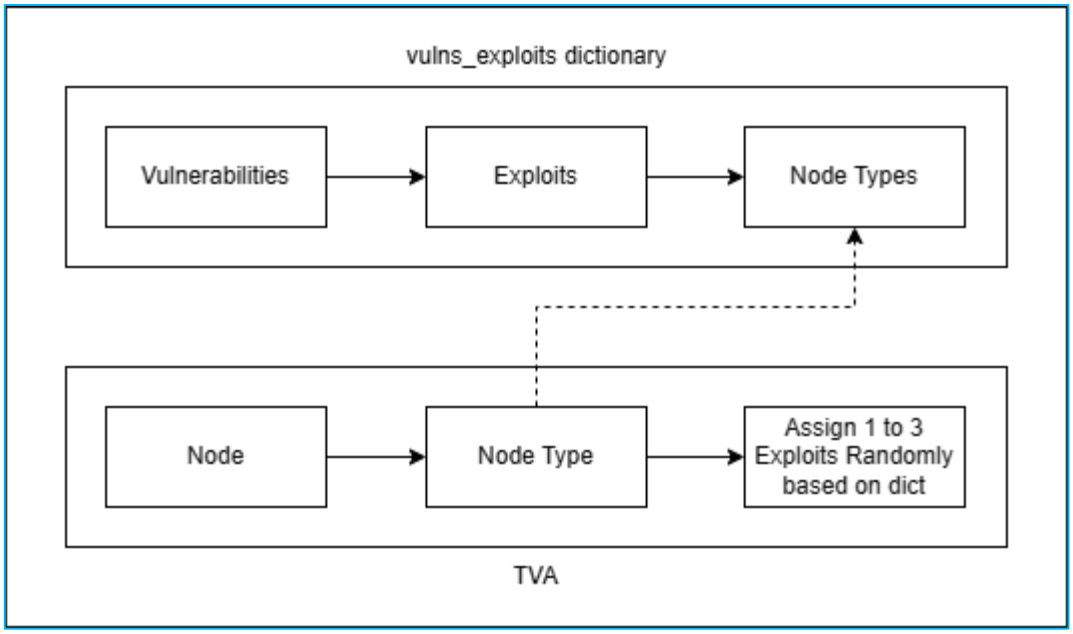
La topología del sistema es un grafo que tiene tres tipos de nodos en este diseño: estación de trabajo, enrutador y servidor. Primero se genera de nodos que define el usuario y se les asigna un tipo aleatoriamente. Luego se establecen conexiones entre los nodos con probabilidad de 0.4. Al final se detectan los componentes conectados y si hay más de uno, se conectan entre sí para tener una red de un único componente conectado.



Análisis de vulnerabilidades topológico

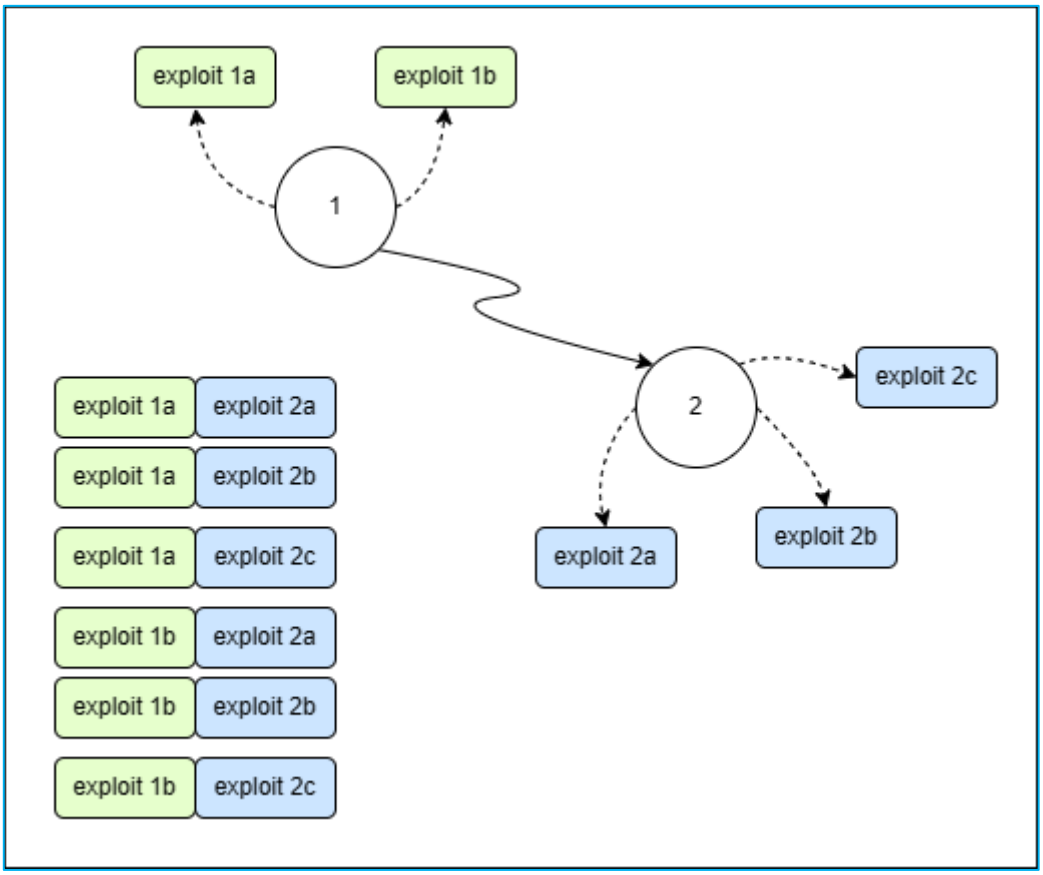
El TVA en esta implementación consiste en iterar sobre todos los nodos de la red, consultar su tipo, buscar todos los exploits que tienen ese tipo de nodo asociado en el diccionario de vulnerabilidades y exploits, y asignar de 1 a 3 exploits al nodo aleatoriamente. Este rango de 1 a 3 fue definido arbitrariamente y podría ser un parámetro de la aplicación.

Un sistema más avanzado usaría información recopilada por los expertos o un conjunto de datos disponible que indique exploits son más comunes para cierto tipo de nodo en una infraestructura particular.



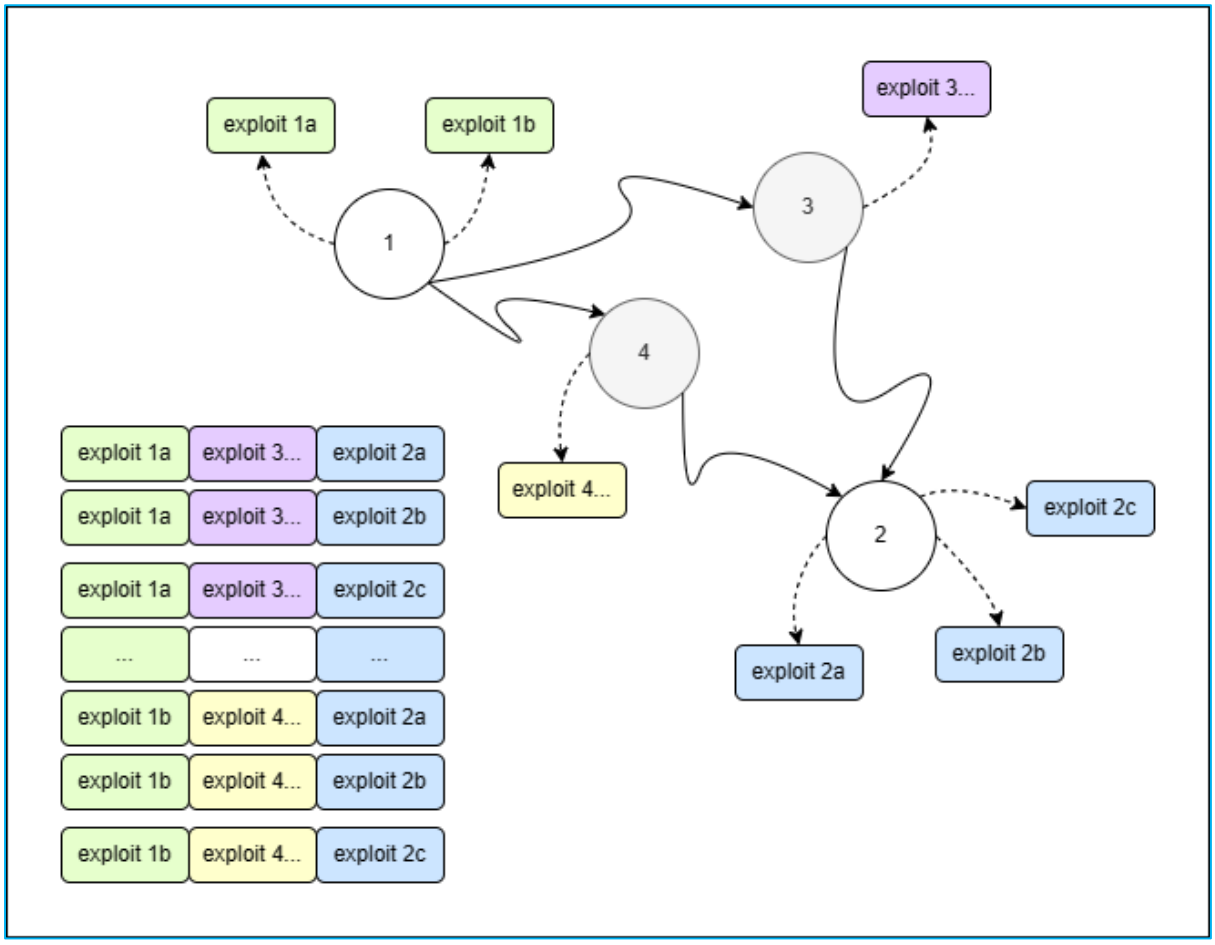
Caminos de ataque

La siguiente imagen muestra el ejemplo de dos nodos vecinos en un grafo. Cada uno con sus exploits asignados tras el TVA. Los caminos de ataque entre estos dos nodos se forman con todas las combinaciones de exploits que forman ambos nodos.



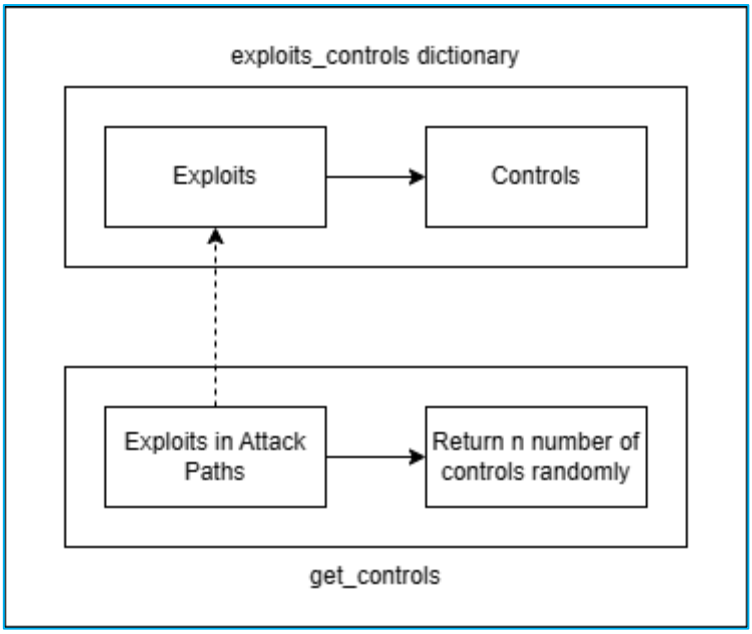
Si hay nodos intermedios, el número de caminos entre ambos nodos, aumenta naturalmente. Pero como además estos nodos también tienen exploits asignados, el número de caminos de ataque crece aun más de manera exponencial. Por esta razón, en este diseño, se pone un límite m de caminos de

ataque que define el usuario al iniciar un experimento. De otra manera se tendrían matrices de juego muy grandes y la complejidad para encontrar la solución podría llegar a ser intratable.



Definición de controles (estrategias del defensor)

En este diseño, se buscan todos los exploits diferentes que se encuentran en los caminos de ataque definidos, y se consulta en el diccionario de exploits y controles, aquellos controles que aplican. Luego, se seleccionan n controles, donde n es el número de controles que define el usuario al iniciar el experimento. Nuevamente, se podría no poner este limitante y entregar todos los controles disponibles, pero el tamaño de la matriz crecería a tal punto de que la complejidad para encontrar la solución llegue a ser intratable.



REFERENCIAS

[1] Rass S, König S, Schauer S (2017) Defending Against Advanced Persistent Threats Using Game-Theory. PLOS ONE 12(1): e0168675. https://doi.org/10.1371/journal.pone.0168675