Instituto Tecnológico de Estudios Superiores

Monterrey Campus Querétaro

[TC2038 – Análisis y diseño de algoritmos avanzados]

AI1_Actividad Integradora 1

Profesores:

Ramona Fuentes Valdez

Presenta:

Ian Joab Padrón Corona	A01708940
Diego Vega Camacho	A01704492
Arturo Cristián Díaz López	A01709526

Introducción

En un mundo cada vez más digitalizado, la seguridad de las transmisiones de datos se ha convertido en una preocupación fundamental. Este proceso, que involucra la inserción de scripts o programas maliciosos en la información transmitida, plantea interrogantes cruciales: ¿es posible detectar la presencia de código malintencionado en el flujo de bits de una transmisión? ¿Es factible determinar si el inicio de los datos ha sido desplazado en el flujo de bits?. Además, surge la pregunta de si es posible identificar una parte de código en dos transmisiones diferentes que han sido intervenidas y su grado de similitud.

En este contexto, la capacidad de identificar y comprender la presencia de código malicioso en las transmisiones de datos es un desafío crucial que la tecnología debe abordar. Para el desarrollo de esta Actividad Integradora, exploramos las posibilidades de detectar código malintencionado en transmisiones de datos, evaluar la ubicación de dicho código en el flujo de información y proponer métodos para identificar patrones comunes de código malicioso en diferentes transmisiones comprometidas. Esto con el fin de fortalecer la seguridad digital y proteger la información sensible en un entorno cada vez más interconectado.

Fase 1: Lectura de los archivos

Para abordar la detección de código malicioso en las transmisiones, es crucial tener la capacidad de acceder y analizar el contenido de archivos que puedan contener datos comprometidos. Es por ello que para el desarrollo de esta actividad, implementamos una función permite la lectura del contenido de un archivo y su retorno como una cadena de caracteres, lo que es fundamental para examinar posibles amenazas en los datos transmitidos.

```
// ================================================================================
// Función readFromFile, lee el contenido de un archivo y lo devuelve como una cadena de
// caracteres
//
// @params filename: Archivo que se desea leer
//
// @return: Regresa el contenido del archivo como una cadena de caracteres
// @complexity O(n)
// ================================================================================
// |
// |
// |
// |
// |
// v
```

```cpp
string readFromFile(const string& filename) {
    ifstream file(filename);
    if (!file.is_open()) {
        cerr << "\nNo se encontro el archivo." << endl;
        return "";
    }

    std::string buffer;
    std::string line;

    while (std::getline(file, line))
    {
        buffer += line;
    }

    file.close();
    return buffer;
}
```

Imagen 1. Función para leer los archivos de texto

Fase 2: Verificar el código malicioso

Posteriormente, es esencial contar con una herramienta que pueda analizar el contenido de las transmisiones en busca de código malicioso. Para ello, implementamos una función que busca una subcadena específica que representa el código malicioso dentro del texto de la transmisión y muestra su posición dentro del archivo de transmisión.

```cpp
// ================================================================================
// Función searchMaliciousCode, busca si una subcadena de código malicioso está contenido
// en un archivo de transmisión y muestra la posición donde se encuentra
//
// @params transmission: Texto de transmisión que se desea analizar
// @params maliciousCode: Código malicioso que se desea buscar
// @params transmissionName: Nombre del archivo de transmisión que se desea analizar
//
// @return: Regresa true o false dependiendo si el código malicioso está contenido en el
//        archivo de transmisión
// @complexity O(n)
// ================================================================================
```

```cpp
void searchMaliciousCode(const string& transmission, const string& maliciousCode, const string& transmissionName) {
    size_t pos = transmission.find(maliciousCode);
    size_t startPos = 0;

    while (pos != string::npos) {
        size_t endPos = pos + maliciousCode.length() - 1;
        cout << "(true) Posicion inicial: " << pos << "  Posicion final: " << endPos << endl;
        startPos = pos + maliciousCode.length();
        pos = transmission.find(maliciousCode, startPos);
    }

    if (startPos == 0) {
        cout << "Codigo Malicioso: '" << maliciousCode << "' no encontrado en (" << transmissionName << ")" << endl;
    }
}
```

Imagen 2. Función para determinar si una cadena maliciosa se encuentra en el texto

Fase 3: Similitud de los archivos de transmisión

Una vez identificado el código malicioso, implementamos la función ‘‘findLongestCommonSubstring‘‘ que aborda la detección de código o secuencias en dos transmisiones diferentes. Al comparar las transmisiones y buscar la subcadena común más larga, la función tiene el potencial de identificar patrones similares que podrían representar código malicioso compartido entre ambas transmisiones.

```
// ================================================================================
// Función findLongestCommonSubstring, busca el substring más largo común en dos cadenas
// y muestra su posición
//
// @params transmission1: Primer cadena de caracteres que se usará para buscar el substring más largo
// @params transmission2: Segunda cadena de caracteres que se usará para buscar el substring más largo
//
// @return: Imprime la información del substring más largo común entre las dos cadenas,
//          además, indica la posición donde se encuentra en la cadena
//
// @complexity O(n^2)
// ================================================================================
```

```cpp
string findLongestCommonSubstring(const string& transmission1, const string& transmission2) {
    int len1 = transmission1.length();
    int len2 = transmission2.length();

    vector<vector<int>> dp(len1 + 1, vector<int>(len2 + 1, 0));
    int maxLength = 0;
    int endIndex = 0;

    for (int i = 1; i <= len1; i++) {
        for (int j = 1; j <= len2; j++) {
            if (transmission1[i - 1] == transmission2[j - 1]) {
                dp[i][j] = dp[i - 1][j - 1] + 1;
                if (dp[i][j] > maxLength) {
                    maxLength = dp[i][j];
                    endIndex = i - 1;
                }
            }
        }
    }

    int startIndex = endIndex - maxLength + 1;
    if (maxLength > 0) {
        return transmission1.substr(startIndex, maxLength);
    } else {
        return "Archivo tranmision no valido";
    }
}
```

Imagen 3. Función para determinar el substring más largo similar entre 2 cadenas de texto

Además de esto, implementamos la función “searchSubstring” la cual busca una subcadena en una cadena de transmisión y muestra la posición en la que se encuentra la misma subcadena. Adicionalmente a esto, podremos ver la posición inicial y final de la cadena

```
// ============================================================================================
// Función searchSubstringPositions, busca una subcadena en una cadena de transmisión y muestra la
//  posición donde se encuentra
//
// @params transmission: Texto de transmisión que se desea analizar
// @params substring: Subcadena que se desea buscar
//
// @return: Retorna los índices donde se encuentra la subcadena en la cadena de transmisión.
//
// @complexity O(n)
// ============================================================================================
```

```
void searchSubstringPositions(const string &transmission, const string &substring) {
    size_t startPos = 0;
    size_t pos = transmission.find(substring, startPos);

    while (pos != string::npos) {
        size_t endPos = pos + substring.length();
        cout << "Posicion inicial: " << pos + 1 << "  Posicion final: " << endPos << endl;
        startPos = pos + 1;
        pos = transmission.find(substring, startPos);
    }
}
```

Imagen 4. Función para determinar las posiciones del substring más largo en las transmisiones de texto

Fase 4: Construcción de la función principal

Finalmente, basta con llamar a las diversas funciones previamente implementadas a través de una función ‘main()’. Esta función principal actúa como una herramienta integral para la detección y evaluación de amenazas en las transmisiones de datos, dando solución a la actividad integradora.

```
int main() {
    string mcode1Content = readFromFile("mcode01.txt");
    string mcode2Content = readFromFile("mcode02.txt");
    string mcode3Content = readFromFile("mcode03.txt");
    string transmission1Content = readFromFile("transmission01.txt");
    string transmission2Content = readFromFile("transmission02.txt");

    cout << " " << endl;
    cout << "Archivo de transmission 1" << endl;
    cout << transmission1Content << endl;
    cout << " " << endl;

    cout << "Archivo de transmission 2" << endl;
    cout << transmission2Content << endl;
    cout << " " << endl;

    cout << "Archivo de mcode1" << endl;
    cout << mcode1Content << endl;
    cout << " " << endl;

    cout << "Archivo de mcode2" << endl;
    cout << mcode2Content << endl;
    cout << " " << endl;

    cout << "Archivo de mcode3" << endl;
    cout << mcode3Content << endl;
    cout << " " << endl;

    cout << "       T R A N S M I S S I O N  1              " << endl;
    cout << " " << endl;

    cout << "mcode1: " << endl;
    searchMaliciousCode(transmission1Content, mcode1Content, "transmission01.txt");
    cout << " " << endl;

    cout << "mcode2: " << endl;
    searchMaliciousCode(transmission1Content, mcode2Content, "transmission01.txt");
    cout << " " << endl;

    cout << "mcode3: " << endl;
    searchMaliciousCode(transmission1Content, mcode3Content, "transmission01.txt");
    cout << " " << endl;

    cout << "       T R A N S M I S S I O N  2              " << endl;
    cout << " " << endl;

    cout << "mcode1: " << endl;
    searchMaliciousCode(transmission2Content, mcode1Content, "transmission02.txt");
    cout << " " << endl;

    cout << "mcode2: " << endl;
    searchMaliciousCode(transmission2Content, mcode2Content, "transmission02.txt");
    cout << " " << endl;

    cout << "mcode3: " << endl;
    searchMaliciousCode(transmission2Content, mcode3Content, "transmission02.txt");
    cout << " " << endl;


    cout << "       S U B S T R I N G  C O M P A R T I D O  M A S  L A R G O       " << endl;
    cout << " " << endl;
    string longestCommonSubstring = findLongestCommonSubstring(transmission1Content, transmission2Content);
    cout << longestCommonSubstring << endl;
    cout << " " << endl;

    cout << "Posiciones en la transmission1: " << endl;
    searchSubstring(transmission1Content, longestCommonSubstring);
    cout << "\n" << endl;

    cout << "Posiciones en la transmission2: " << endl;
    searchSubstring(transmission2Content, longestCommonSubstring);
    cout << "\n" << endl;
    return 0;
}
}
```

Conclusiones

En este escenario, destacamos la importancia de contar con herramientas y funciones específicas para abordar la detección de código malicioso y amenazas en las comunicaciones digitales. Su implementación en la función principal y las distintas fases implementadas coordina eficazmente estos pasos, brindando una visión completa de posibles detecciones y similitudes en las transmisiones..

En última instancia, esta actividad integradora resalta la necesidad constante de desarrollar y aplicar herramientas innovadoras para abordar las amenazas cibernéticas en el entorno digital actual. La detección y respuesta eficiente a posibles amenazas son cruciales para garantizar la seguridad de las transmisiones de datos, y las funciones mencionadas desempeñan un papel fundamental en esta misión.

Finalmente, cabe mencionar que de aprovechar esta herramienta que desarrollamos y sus enfoques, ahora estamos mejor preparados para proteger la información que manejamos en futuros proyectos, en un mundo cada vez más interconectado y vulnerable a las amenazas cibernéticas.