INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

AI1_Actividad Integradora 1

Curso: TC2038 **Grupo:** 601

Integrantes:

Daniel Emilio Fuentes Portaluppi - A01708302
Daniel Sebastián Cajas Morales - A01708637
Diego Ernesto Sandoval Vargas - A01709113

Profesor: Ramona Fuentes Valdéz

Octubre 1, 2023

Situación problema 1

Transmisiones de datos comprometidas

Cuando se transmite información de un dispositivo a otro, se transmite una serie sucesiva de bits, que llevan una cabecera, datos y cola. Existe mucha gente mal intencionada, que puede interceptar estas transmisiones, modificar estas partes del envío, y enviarlas al destinatario, incrustando sus propios scripts o pequeños programas que pueden tomar cierto control del dispositivo que recibe la información.

Suponiendo que conocemos secuencias de bits de código malintencionado:

¿Serías capaz de identificarlo dentro del flujo de bits de una transmisión?

¿Podremos identificar si el inicio de los datos se encuentra más adelante en el flujo de bits?

Si tuviéramos dos transmisiones de información y sospechamos que en ambas han sido intervenidas y que traen el mismo código malicioso, ¿Podríamos dar propuestas del código mal intencionado?

Parte 1

El programa debe analizar si el contenido de los archivos ***mcode1.txt***, ***mcode2.txt*** y ***mcode3.txt*** están contenidos en los archivos ***transmission1.txt*** y ***transmission2.txt*** y desplegar un true o false si es que las secuencias de *chars* están contenidas o no. En caso de ser true, muestra true, seguido de exactamente un espacio, seguido de la posición en el archivo de **transmisiónX.txt** donde inicia el código de **mcodeY.txt**

Solución

Para esta primera parte utilizamos el algoritmo de Naive. Es un método muy simple que utiliza el enfoque de fuerza bruta.
Funciona comparando el primer carácter del patrón con el texto que se puede buscar.

- Si encuentra una coincidencia, se avanzan los punteros en ambas cadenas.
- Si no se encuentra la coincidencia, el puntero del texto se incrementa y el puntero del patrón se restablece.
- Este proceso se repite hasta el final del texto.

Complejidad

Este algoritmo tiene una complejidad de *O(m*(n-m))*

Capturas del código

```
/**
 * @brief Function that searches for a string in a transmission.
 *
 * @param pat
 * @param txt
 *
 * @complexity O(m*(n-m))
 */
void search(char *pat, char *txt)
{
    int M = strlen(pat);
    int N = strlen(txt);

    if (M > N)
    {
        cout << "La longitud del patron es mayor que la de la cadena." <<
endl;
        return;
    }

    bool found = false;

    for (int i = 0; i <= N - M; i++)
    {
        int j;

        for (j = 0; j < M; j++)
        {
            if (txt[i + j] != pat[j])
                break;
        }
        if (j == M)
        {
            cout << "\n(true) " << i << endl;
            found = true;
        }
    }

    if (!found)
    {
        cout << "\n(false) Cadena no encontrada en la transmision" << endl;
    }
}
```

Parte 2

Suponiendo que el código malicioso tiene siempre código "espejeado" (palíndromos de *chars*), sería buena idea buscar este tipo de código en una transmisión. El programa después debe buscar si hay código "espejeado" dentro de los archivos de transmisión. (palíndromo a nivel *chars*, no meterse a nivel *bits*). El programa muestra en una sola línea dos enteros separados por un espacio correspondientes a la posición (*iniciando en 1*) en donde inicia y termina el código "espejeado" más largo (palíndromo) para cada archivo de transmisión. Puede asumirse que siempre se encontrará este tipo de código.

Solución

Para esta segunda parte se utilizaran 2 algoritmos sencillos, uno para generar el palíndromo o mcode espejado y otro para encontrar este código malicioso dentro de nuestra transmisión:

- Compara los caracteres del patrón con los del texto, uno por uno.
- Si todos los caracteres coinciden, se registra la posición de inicio y fin de la coincidencia.
- Luego, se continúa buscando más coincidencias en el texto.
- Al final, se obtiene una lista de todas las posiciones donde se encontraron coincidencias.

Complejidad

Este algoritmo tiene una complejidad de *O(m*(n-m))*

Capturas del código

```
/**
 * @brief Function that generates a palindrome from a string.
 *
 * @param maliciusString
 * @return the palindrome generated.
 *
 * @complexity O(n)
 */
string generatePalindrome(string maliciusString)
{
    string palindrome = "";
    int size = maliciusString.length();
    for (int i = size - 1; i >= 0; i--)
    {
        palindrome += maliciusString[i];
    }
    return palindrome;
}
```

```
/**
 * @brief Function that recieves a malicius string and search for a palindrome.
 * in the string.
 *
 * @param maliciusString
 * @return
 the initial position and final positions of the palindrome in the string.
 *
 * @complexity O(m*(n-m))
 */
vector<pair<int, int>> searchPalindrome(string maliciusString, string
transmission)
{
    string mCodePalindrome = generatePalindrome(maliciusString);
    return searchAll(mCodePalindrome, transmission);
}
```

```cpp
vector<pair<int, int>> searchAll(string pattern, string transmission)
{
    vector<pair<int, int>> positions;

    int lenTransmission = transmission.length();

    for (int i = 0; i < lenTransmission; i++)
    {
        int j = 0;
        int k = i;
        while (transmission[k] == pattern[j] && j < pattern.length())
        {
            j++;
            k++;
        }
        if (j == pattern.length())
        {
            positions.push_back(make_pair(i, k - 1));
        }
    }

    return positions;
}
```

Parte 3

Finalmente el programa analiza que tan similares son los archivos de transmisión, y debe mostrar la posición inicial y la posición final (iniciando en 1) del primer archivo en donde se encuentra el substring más largo común entre ambos archivos de transmisión.

Solución

Para esta parte utilizamos una solución basada en programación dinámica, se genera una matriz que almacena el tamaño del substring más largo, y si se encuentran más coincidencias este se va alargando, si no se regresa a cero. Se guarda el final del substring más largo y su tamaño para poder extraerlo fácilmente al finalizar. Luego se utiliza un algoritmo Naive para encontrar todas las coincidencias de este substring en cada transmisión.

Complejidad

Este algoritmo tiene una complejidad de *O(m*n)*

Capturas del código

```cpp
/**
 * @brief
 Function that recieves two strings and returns the longest common substring.
 *
 * @param s1
 * @param s2
 * @return the longest common substring.
 *
 * @complexity O(n*m)
 */
string longuestCommonSubstring(string s1, string s2)
{
    int n = s1.length();
    int m = s2.length();
    int dp[n + 1][m + 1];
    int mx = 0, end = -1;
    for (int i = 0; i <= n; i++)
    {
        for (int j = 0; j <= m; ++j)
        {
            if (i == 0 || j == 0)
            {
                dp[i][j] = 0;
            }
            else if (s1[i - 1] == s2[j - 1])
            {
                dp[i][j] = 1 + dp[i - 1][j - 1];
                if (mx < dp[i][j])
                {
                    mx = dp[i][j];
                    end = i - 1;
                }
            }
            else
            {
                dp[i][j] = 0;
            }
        }
    }

    return s1.substr(end - mx + 1, mx);
}
```

Función Main

```
int main(int argc, char *argv[])
{
    ifstream inputFileT1, inputFileT2, inputFileMC1, inputFileMC2, inputFileMC3;
    string line, transmision1, transmision2, mcode1, mcode2, mcode3;

    if (argc < 6)
    {
        cout << "Uso: " << argv[0] << " <transmision1.txt> <transmision2.txt> <mcode1.txt> <mcode2.txt> <mcode3.txt>" << endl;
        return 1;
    }

    // Opening files
    inputFileT1.open(argv[1]);
    inputFileT2.open(argv[2]);
    inputFileMC1.open(argv[3]);
    inputFileMC2.open(argv[4]);
    inputFileMC3.open(argv[5]);

    if (!inputFileT1 || !inputFileT2 || !inputFileMC1 || !inputFileMC2 || !inputFileMC3)
    {
        cout << "Error abriendo los archivos" << endl;
        return 1;
    }

    // Reading Transmissions
    while (getline(inputFileT1, line))
    {
        transmision1 += line;
    }

    while (getline(inputFileT2, line))
    {
        transmision2 += line;
    }

    // Reading Malicious Codes
    inputFileMC1 >> mcode1;
    inputFileMC2 >> mcode2;
    inputFileMC3 >> mcode3;

    cout << "Transmision 1: " << transmision1 << endl
         << endl;
    cout << "Transmision 2: " << transmision2 << endl
         << endl;
    cout << "mcode 1: " << mcode1 << endl
         << endl;
    cout << "mcode 2: " << mcode2 << endl
         << endl;
    cout << "mcode 3: " << mcode3 << endl
         << endl;

    cout << "La cadena maliciosa1 se encuentra en la transmision1: " << endl;
    search(mcode1, transmision1);

    cout << "\nLa cadena maliciosa2 se encuentra en la transmision1: " << endl;
    search(mcode2, transmision1);

    cout << "\nLa cadena maliciosa3 se encuentra en la transmision1: " << endl;
    search(mcode3, transmision1);

    cout << "\nLa cadena maliciosa1 se encuentra en la transmision2: " << endl;
    search(mcode1, transmision2);

    cout << "\nLa cadena maliciosa2 se encuentra en la transmision2: " << endl;
    search(mcode2, transmision2);

    cout << "\nLa cadena maliciosa3 se encuentra en la transmision2: " << endl;
    search(mcode3, transmision2);

    cout << "\nPosiciones de los Palindromos maliciosos: " << endl;
```

```
cout << "\nPalindromo de cadena maliciosa1 en la transmicion 1: " << endl;
    vector<pair<int, int>> positions = searchPalindrome(mcode1, transmision1);
    for (int i = 0; i < positions.size(); i++)
    {
        cout << "Posicion inicial: " << positions[i].first << " Posicion final: " << positions[i].second << endl;
    }

    cout << "\nPalindromo de cadena maliciosa2 en la transmicion 1: " << endl;
    positions = searchPalindrome(mcode2, transmision1);
    for (int i = 0; i < positions.size(); i++)
    {
        cout << "Posicion inicial: " << positions[i].first << " Posicion final: " << positions[i].second << endl;
    }

    cout << "\nPalindromo de cadena maliciosa3 en la transmicion 1: " << endl;
    positions = searchPalindrome(mcode3, transmision1);
    for (int i = 0; i < positions.size(); i++)
    {
        cout << "Posicion inicial: " << positions[i].first << " Posicion final: " << positions[i].second << endl;
    }

    cout << "\nPalindromo de cadena maliciosa1 en la transmicion 2: " << endl;
    positions = searchPalindrome(mcode1, transmision2);
    for (int i = 0; i < positions.size(); i++)
    {
        cout << "Posicion inicial: " << positions[i].first << " Posicion final: " << positions[i].second << endl;
    }

    cout << "\nPalindromo de cadena maliciosa2 en la transmicion 2: " << endl;
    positions = searchPalindrome(mcode2, transmision2);
    for (int i = 0; i < positions.size(); i++)
    {
        cout << "Posicion inicial: " << positions[i].first << " Posicion final: " << positions[i].second << endl;
    }

    cout << "\nPalindromo de cadena maliciosa3 en la transmicion 2: " << endl;
    positions = searchPalindrome(mcode3, transmision2);
    for (int i = 0; i < positions.size(); i++)
    {
        cout << "Posicion inicial: " << positions[i].first << " Posicion final: " << positions[i].second << endl;
    }

    cout << "\nSubcadena comun mas larga entre la transmision 1 y la transmision 2: " << endl;
    string longestCommonSubstring = longuestCommonSubstring(transmision1, transmision2);
    cout << longestCommonSubstring << endl;

    cout << "\nCoincidencias en transmision 1:" << endl;
    positions = searchAll(longestCommonSubstring, transmision1);
    for (int i = 0; i < positions.size(); i++)
    {
        cout << "Posicion inicial: " << positions[i].first << " Posicion final: " << positions[i].second << endl;
    }

    cout << "\nCoincidencias en transmision 2:" << endl;
    positions = searchAll(longestCommonSubstring, transmision2);
    for (int i = 0; i < positions.size(); i++)
    {
        cout << "Posicion inicial: " << positions[i].first << " Posicion final: " << positions[i].second << endl;
    }

    return 0;
}
```

Output


Transmision 1: 70616e206465206e6172716e6a6150616e206465206e6172616e6a61
Transmision 2: 70616e206465206e6172716e6a6170616e206465206e6172716e6a6170616e206465206e6172716e6a6170616e206465206e6172
mcode 1: 16e61
mcode 2: 737
mcode 3: 17271
La cadena maliciosa1 se encuentra en la transmision1:
(false) Cadena no encontrada en la transmision
La cadena maliciosa2 se encuentra en la transmision1:
(false) Cadena no encontrada en la transmision
La cadena maliciosa3 se encuentra en la transmision1:
(true) 17
La cadena maliciosa1 se encuentra en la transmision2:
(false) Cadena no encontrada en la transmision
La cadena maliciosa2 se encuentra en la transmision2:
(false) Cadena no encontrada en la transmision
La cadena maliciosa3 se encuentra en la transmision2:
(true) 17
(true) 45
(true) 73
(true) 101
(true) 129
(true) 157
Posiciones de los Palindromos maliciosos:
Palindromo de cadena maliciosa1 en la transmicion 1:
No se encontro palindromo malicioso en la transmicion
Palindromo de cadena maliciosa2 en la transmicion 1:
No se encontro palindromo malicioso en la transmicion
Palindromo de cadena maliciosa3 en la transmicion 1:
Posicion inicial: 17 Posicion final: 21
Palindromo de cadena maliciosa1 en la transmicion 2:
No se encontro palindromo malicioso en la transmicion
Palindromo de cadena maliciosa2 en la transmicion 2:
No se encontro palindromo malicioso en la transmicion
Palindromo de cadena maliciosa3 en la transmicion 2:
Posicion inicial: 17 Posicion final: 21
Posicion inicial: 45 Posicion final: 49
Posicion inicial: 73 Posicion final: 77
Posicion inicial: 101 Posicion final: 105
Posicion inicial: 129 Posicion final: 133
Posicion inicial: 157 Posicion final: 161
Subcadena comun mas larga entre la transmision 1 y la transmision 2:
70616e206465206e6172716e6a61
Coincidencias en transmision 1:
Posicion inicial: 0 Posicion final: 27
Coincidencias en transmision 2:
Posicion inicial: 0 Posicion final: 27
Posicion inicial: 28 Posicion final: 55
Posicion inicial: 56 Posicion final: 83
Posicion inicial: 84 Posicion final: 111
Posicion inicial: 112 Posicion final: 139
Posicion inicial: 140 Posicion final: 167

Conclusión

Los algoritmos de manejo de strings son muy importantes para la solución de una situación problema como la que se nos ha presentado. Vivimos en un mundo cada vez más conectado, donde la seguridad de la información es de suma importancia. La capacidad para poder identificar y analizar secuencias específicas de bits maliciosos es de suma importancia. Todos los algoritmos presentados en este reporte facilitan la detección de estas secuencias. Y a pesar de eso existen otras más alternativas que pueden solucionar el mismo problema.

Además, la aplicación de estos algoritmos se vuelve más poderosa cuando se comparan múltiples transmisiones comprometidas. Poder comprender este tipo de algoritmos se convierte en un componente crucial en la lucha para preservar la integridad y confidencialidad de la información en el mundo actual.