“Informe”

Dirigido a la Docente:

KATHERINE OSPINO BERMEJO

En la asignatura de algoritmia y programación 2

Presentado por

David Buelvas

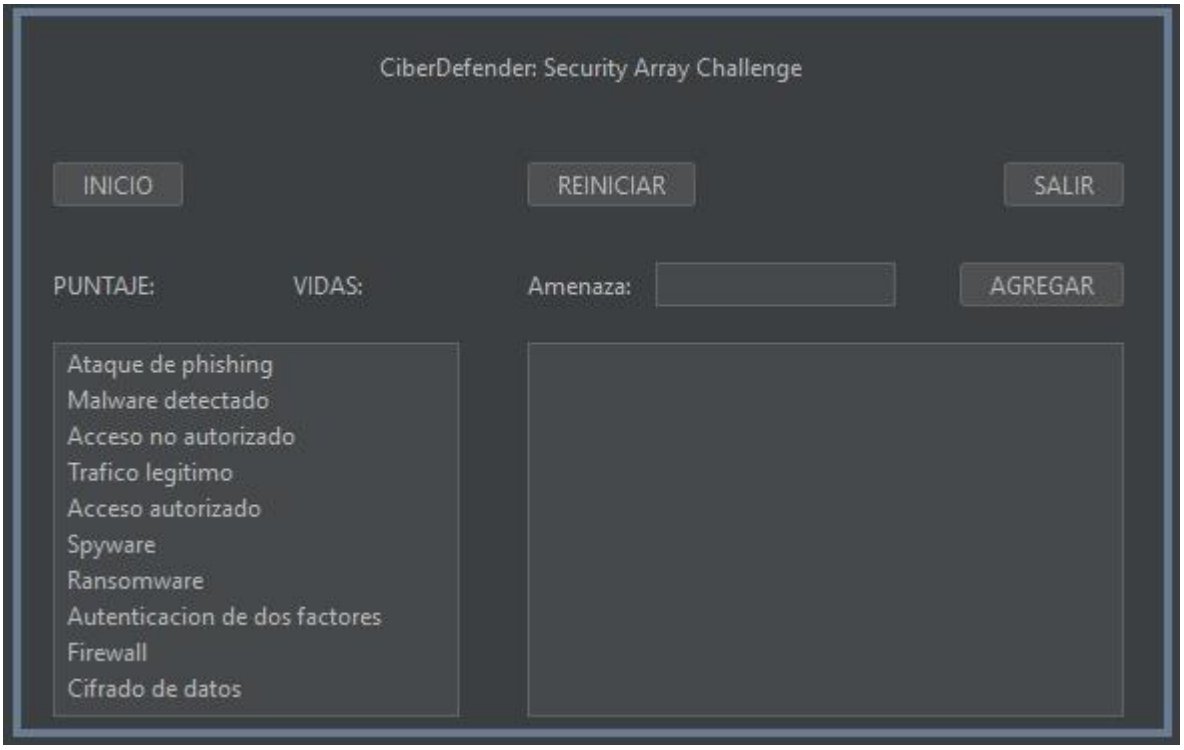
Andrés Gonzales

VIERNES 14 DE NOVIEMBRE DEL 2025

Diseño modular del proyecto

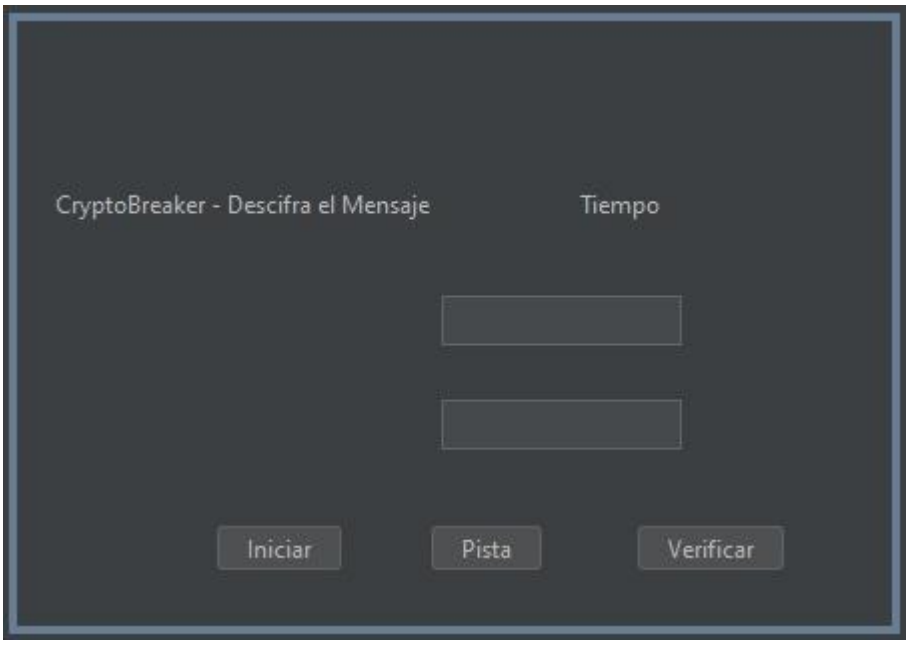
El proyecto se desarrolló bajo un enfoque modular, donde cada parte del sistema representa una herramienta distinta orientada a la ciberseguridad.
Esto permitió dividir el trabajo en bloques independientes, más fáciles de probar y mantener.
Los módulos creados fueron los siguientes:

Módulo 1 – CiberDefender (Arreglos y validación de amenazas):
Permite registrar eventos ingresados por el usuario, clasificarlos como amenaza válida o no válida y llevar puntaje y vidas.

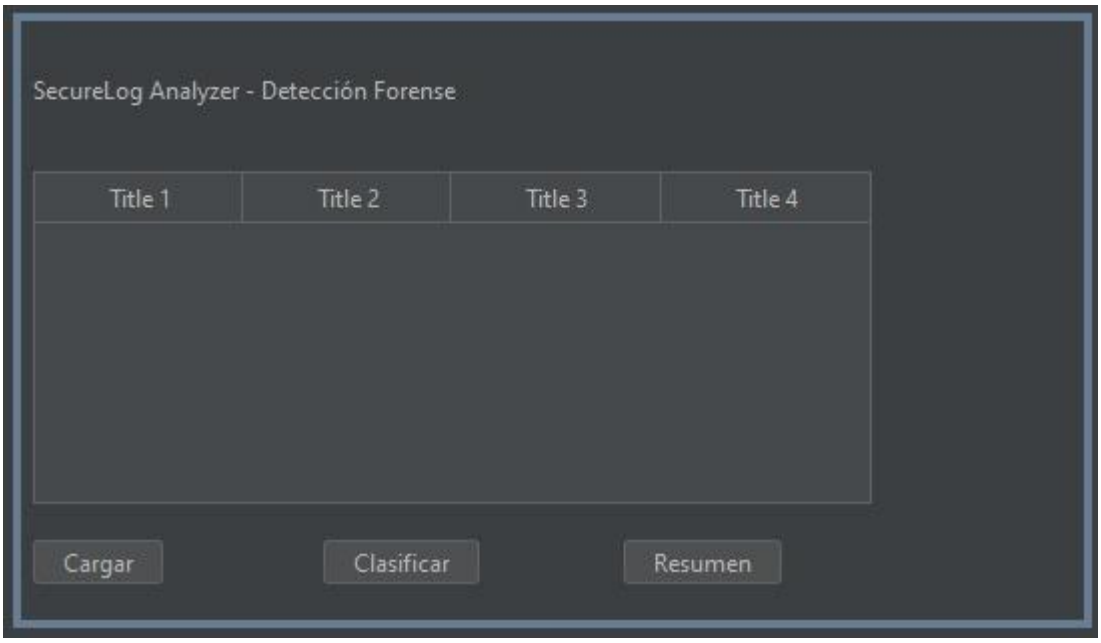


Módulo 2 – CryptoBreaker (Cifrado y temporizador):
El módulo *FirewallMatrix* simula un sistema de defensa digital en el que el jugador debe identificar y bloquear puertos comprometidos dentro de una matriz 5×5. El objetivo es analizar la cuadrícula, detectar los ataques y bloquear la mayor cantidad posible con la menorcantidad de errores.

Módulo 3 – CryptoBreaker (Cifrado y temporizador):
Presenta un mensaje cifrado con un desplazamiento tipo César y el jugador debe descifrarlo antes de que el temporizador llegue a cero.



Módulo 4 – SecureLog Analyzer (Lectura de archivos y clasificación forense): Lee un archivo de registros de red (logs.txt), clasifica cada evento como Normal, Sospechoso o Crítico y muestra un resumen.



PROCEDIMIENTOS Y FUNCIONES EMPLEADOS

MÓDULO 1

-Función esPhishing(String texto)

Esta es la función principal del módulo.
Recibe la cadena ingresada por el jugador y retorna **true** si encuentra palabras asociadas a phishing.

private boolean esPhishing(String texto)

-Procedimiento actualizarPuntaje(boolean acierto)

Actualiza el puntaje según el resultado de la clasificación.

private void actualizarPuntaje(boolean acierto)

-Procedimiento guardarResultado(String evento, String tipo)

Graba en un archivo **resultados_modulo1.txt** el evento analizado y su clasificación.

private void guardarResultado(String evento, String tipo)

Procedimientos generados por Swing

El código contiene procedimientos para manejar los botones:

- btnInicioMouseClicked
- btnAgregarMouseClicked
- btnReiniciarMouseClicked
- btnSalirMouseClicked

Cada uno actúa como controlador (controlador de eventos) y organiza la interacción con el usuario.

MÓDULO 2

-Procedimiento generarAtaqueAleatorio()

Genera un ataque dentro de la matriz 5×5.

private void generarAtaqueAleatorio()

-Procedimiento actualizarTabla()

Actualiza visualmente el contenido del JTable.

private void actualizarTabla()

-Función implícita: la verificación dentro de verificarBloqueo()

private void verificarBloqueo(int fila, int columna, JTable tabla)

Este método funciona como una función lógica, aunque está implementado como procedimiento.

-Procedimiento verificarResultado()

Evalúa si el jugador logró bloquear ≥ 80%.

private void verificarResultado()

MÓDULO 3

-Función desencriptar(char[] m)

Es la función principal del módulo.

private String desencriptar(char[] m)

-Procedimiento mostrarPista()

private void mostrarPista()

Muestra una pista al usuario mediante un JOptionPane.

-Procedimiento crearTimer()

private void crearTimer()

Es un procedimiento que configura la mecánica del tiempo.

-Procedimiento actualizarTiempo()

Actualiza el label del tiempo en pantalla:

private void actualizarTiempo()

No retorna nada, simplemente refresca la UI.

MÓDULO 4

-Función: clasificarEvento(String evento)

private String clasificarEvento(String evento)

Es la función central del módulo.

-Procedimiento: cargarLogsDesdeArchivo()

private void cargarLogsDesdeArchivo()

Este procedimiento lee el archivo logs.txt línea por línea usando:

- BufferedReader

- FileReader

-Procedimiento: clasificarTodos()

private void clasificarTodos()

-Procedimiento: mostrarResumen()

private void mostrarResumen()

Es un procedimiento informativo que resume el análisis completo.

MANEJO DE ARREGLOS

MÓDULO 1

Este módulo usa dos arreglos principales:

Arreglo de eventos predefinidos

private final String[] eventos = {...}

Se usa para llenar la lista listaEventosDeSeguridad.

Arreglo dinámico de amenazas detectadas

private String[] v = new String[10];

MÓDULO 2

Matriz bidimensional 5×5

int[][] matrizPuertos = new int[5][5];

Recorridos clásicos de doble bucle

Ambos métodos generarAtaqueAleatorio y actualizarTabla usan:

```
for (int i = 0; i < 5; i++) {
   for (int j = 0; j < 5; j++) {
      ...
   }
}
```

Conversión matriz → JTable

El JTable no almacena la matriz directamente, así que se copian sus valores:

jTable1.setValueAt(matrizPuertos[i][j], i, j);

Esto facilita la visualización del ataque.

MÓDULO 3

Arreglo de caracteres cifrado

char[] mensajeCifrado = {'K', 'R', 'R', 'H'};

Recorrido tipo foreach

for (char c : m)

MÓDULO 4

Arreglos de palabras clave

Se usan dos arreglos para la detección semántica:

String[] criticas = { "ataque", "malware", "exploit", ... }

String[] sospecha = { "sospech", "alert", "fallido", ... }

Listas dinámicas (ArrayList)

private ArrayList<String> eventosList;

private ArrayList<String> categoriasList;

Recorrido mediante ciclos

El módulo emplea ciclos clásicos:

for (int i = 0; i < eventosList.size(); i++)

MANEJO DE ARCHIVOS

MÓDULO 1

FileWriter fw = new FileWriter("resultados_modulo1.txt", true);

Este sirve como bitácora de análisis de amenazas del jugador.

MÓDULO 2

El módulo FirewallMatrix no requiere lectura ni escritura de archivos según los requisitos planteados para el laboratorio. Por esta razón, toda la información se maneja únicamente en memoria mediante una matriz bidimensional

MÓDULO 3

El módulo CryptoBreaker no requiere lectura ni escritura de archivos, debido a que su funcionalidad se centra exclusivamente en el manejo de un arreglo de caracteres, temporizador y validación directa de la respuesta del usuario

MÓDULO 4

Lectura del archivo logs.txt

En el procedimiento cargarLogsDesdeArchivo() se utiliza:

FileReader → abre el archivo

BufferedReader → lectura eficiente línea a línea

Formato del archivo esperado:

Intento fallido de login

Ataque de malware detectado

Usuario ingresó al sistema

Ransomware encontrado

Integración con la interfaz

Cada línea leída:

- se guarda en eventosList
- se agrega a la tabla para que el usuario la vea
- se deja su clasificación vacía hasta presionar "Clasificar"

Manejo de errores

Si el archivo no existe o no se puede abrir, el programa muestra:

JOptionPane.showMessageDialog(this, "Error leyendo logs.txt: ...")

Evita que el programa se cierre inesperadamente.

Dificultades y soluciones implementadas.

En el módulo 2 tuvimos dos dificultades. La primera era sincronizar la matriz interna con la tabla visual (JTable). Inicialmente los valores de la matriz no coincidían con lo que se mostraba en la tabla, y para solucionarlo, se creó el procedimiento actualizarTabla() que copia cada elemento de la matriz en la interfaz. Además, cada clic actualiza tanto la tabla como los contadores de aciertos y errores. La segunda dificultad fue capturar correctamente el clic en una celda específica. Un error común en Swing es que el clic no registra la fila y la columna esperada. Como solución, se agregó un MouseListener al JTable que obtiene la celda seleccionada con:

int fila = jTable1.getSelectedRow();

int columna = jTable1.getSelectedColumn();

Esto permitió verificar si el puerto estaba comprometido y mostrar el resultado en tiempo real.

Respecto al módulo 3, también tuvimos dos problemas. Al principio el temporizador se detenía o no actualizaba correctamente el label. Como solución utilizamos la clase javax.swing.Timer, que ejecuta acciones cada segundo sin bloquear la GUI. El método actualizarTiempo() refresca la interfaz adecuadamente. La segunda dificultad fue implementar correctamente el descifrado César. El desplazamiento debía ser uniforme para todas las letras y evitar errores de conversión. Como solución se implementó una función desencriptar() que recorre el arreglo de caracteres y resta 3 posiciones ASCII. Esto garantiza que el descifrado sea consistente.

Y por último nuestra ultima dificultad fue en el módulo 4, y era mantener sincronizada la tabla con las listas internas. Lo que pasaba era que cuando se clasificaban eventos, la tabla no reflejaba los cambios. Como solución se actualizó el DefaultTableModel directamente con:

modelo.setValueAt(categoria, i, 1);

Esto asegura que la GUI siempre coincida con los datos internos.