Programacion Avanzada para Ciberseguridad
PIA
Andres Tadeo Flores Pinal Eduardo Flores Smith Jorge Luis Casas Cruz
20/11/2025

Índice

1. Resumen Ejecutivo

El presente documento detalla el ciclo de vida de desarrollo, implementación técnica y auditoría de seguridad de **MiniCryptTool**, un artefacto de software de tipo Interfaz de Línea de Comandos (CLI) diseñado para la ejecución de primitivas criptográficas en entornos educativos. El propósito fundamental del proyecto ha sido la construcción de una herramienta que permita la aplicación práctica de algoritmos de cifrado simétrico y funciones de resumen (hashing), operando bajo un esquema de transparencia total y seguridad por diseño.

La herramienta, desarrollada en el lenguaje de programación C++, integra implementaciones nativas de los cifrados **César** y **XOR**, así como de los algoritmos de hashing **DJB2** y **FNV-1a**. Su arquitectura ha sido optimizada para garantizar la portabilidad entre sistemas operativos Windows y Linux, eliminando dependencias externas complejas y facilitando su compilación reproducible. Desde una perspectiva ética, el desarrollo se rigió por principios de "seguridad defensiva": el binario resultante carece deliberadamente de capacidades de persistencia, propagación autónoma o exfiltración de datos, limitándose estrictamente al procesamiento de datos solicitados por el usuario.

La validación del artefacto incluyó fases de análisis estático y, crucialmente, un **Análisis Dinámico** exhaustivo realizado en entornos virtualizados aislados (*host-only*). Las pruebas dinámicas certificaron que la interacción del software con el sistema operativo es nominal y

segura, sin registrarse actividad de red anómala, modificaciones no autorizadas en el registro del sistema, ni comportamientos heurísticos asociados a software malicioso (*malware*).

En conclusión, MiniCryptTool se entrega como una solución robusta y verificada, cumpliendo con los requisitos funcionales y de seguridad establecidos. Este reporte documenta las evidencias técnicas que respaldan la integridad del software y su idoneidad para ser desplegado en entornos de laboratorio y demostración académica.

2. Descripción del Payload y Límites Éticos

2.1 Descripción Funcional

El entregable principal, denominado MiniCryptTool, es un ejecutable monolítico diseñado para realizar transformaciones de datos bidireccionales (cifrado/descifrado) y unidireccionales (hashing).

El payload funcional permite al usuario:

- **Cifrado Simétrico:** Proteger la confidencialidad de archivos de texto plano mediante algoritmos clásicos. El modo **XOR** permite una ofuscación básica mediante una clave alfanumérica, mientras que el modo **César** ofrece un cifrado de sustitución por desplazamiento configurable.
- **Verificación de Integridad:** Generar huellas digitales (*hashes*) de archivos o cadenas de texto para validar que no han sido modificados, utilizando los algoritmos DJB2 y FNV-1a, seleccionados por su eficiencia y valor pedagógico en la comprensión de colisiones y distribución de bits.

2.2 Cumplimiento Ético y de Seguridad

De acuerdo con el documento ETHICS.md y los requisitos del proyecto, se establecieron los siguientes límites técnicos inquebrantables:

1. **Ausencia de Persistencia:** El programa no intenta copiarse a carpetas de inicio, modificar claves de registro Run, ni crear servicios ocultos. Su ejecución es efímera.
2. **Aislamiento de Red:** El código fuente no incluye bibliotecas de *sockets* (como <sys/socket.h> o Winsock), garantizando físicamente la incapacidad de establecer conexiones remotas (C2).
3. **Transparencia:** Se entrega una versión con símbolos de depuración (payload_debug_x64) para permitir la auditoría total por terceros.

3. Diseño e Implementación Técnica

3.1 Arquitectura del Software

El sistema está construido en C++ estándar (C++11/17), utilizando un enfoque modular dentro de un único archivo fuente (MiniCryptTool.cpp) para facilitar la compilación. No se utilizan *frameworks* externos, confiando exclusivamente en la Biblioteca Estándar de C++ (STL).

Componentes Clave:

- **Parser de Argumentos:** Implementado manualmente en la función main, procesa banderas como --mode, --alg, --key y --in. Esto evita vulnerabilidades asociadas a librerías de terceros.
- **Gestión de Entrada/Salida (I/O):** La función read_all abstrae la fuente de datos, permitiendo leer indistintamente desde un archivo físico (std::ifstream) o desde la entrada estándar (std::cin). Esto habilita el uso de tuberías (*pipes*) en Linux/Windows (ej. echo "texto" | ./minicrypt).

3.2 Algoritmos Implementados

1. **XOR Cipher:** Itera sobre el búfer de entrada aplicando la operación lógica ^ (XOR) con la clave proporcionada de forma cíclica (key[i % key_len]). Es una operación involutiva (cifrar dos veces devuelve el original).
2. **Caesar Cipher:** Realiza aritmética modular sobre caracteres ASCII. Se normaliza el desplazamiento (shift %= 26) y se manejan mayúsculas y minúsculas por separado para preservar el formato del texto.
3. **DJB2 Hash:** Implementa la variante clásica hash * 33 + c, inicializando en el valor mágico 5381.
4. **FNV-1a Hash:** Utiliza el primo FNV (1099511628211ULL) y el offset base para realizar operaciones XOR y multiplicación, ofreciendo una buena dispersión de bits para entradas cortas.

3.3 Manejo de Memoria

El uso de std::vector<uint8_t> y std::string garantiza una gestión automática de la memoria (RAII), minimizando el riesgo de desbordamientos de búfer (*buffer overflows*) o fugas de memoria (*memory leaks*), comunes en implementaciones de C puro.

4. Metodología de Pruebas

Para garantizar la seguridad, todas las pruebas se realizaron en un entorno estrictamente controlado.

- **Hipervisor:** VirtualBox 7.0.
- **Sistema Operativo Invitado (Guest):** Windows 10 Enterprise (Evaluación) / Ubuntu 24.04 LTS.
- **Configuración de Red:** *Host-Only Adapter* (Sin acceso a internet, solo comunicación local estricta).
- **Herramientas de Análisis:**
    - *Process Monitor (ProcMon):* Para monitoreo de sistema de archivos y registro.
    - *Wireshark:* Para captura de tráfico de paquetes.
    - *Ghidra:* Para análisis estático e ingeniería inversa.
    - *HashCalc:* Para validación cruzada de resultados.

5. Análisis Dinámico

El análisis dinámico se centró en observar el comportamiento del binario MiniCryptTool.exe durante su ejecución en tiempo real para detectar anomalías o efectos secundarios no documentados.

5.1 Análisis de Interacción con el Sistema de Archivos (Entorno Linux)

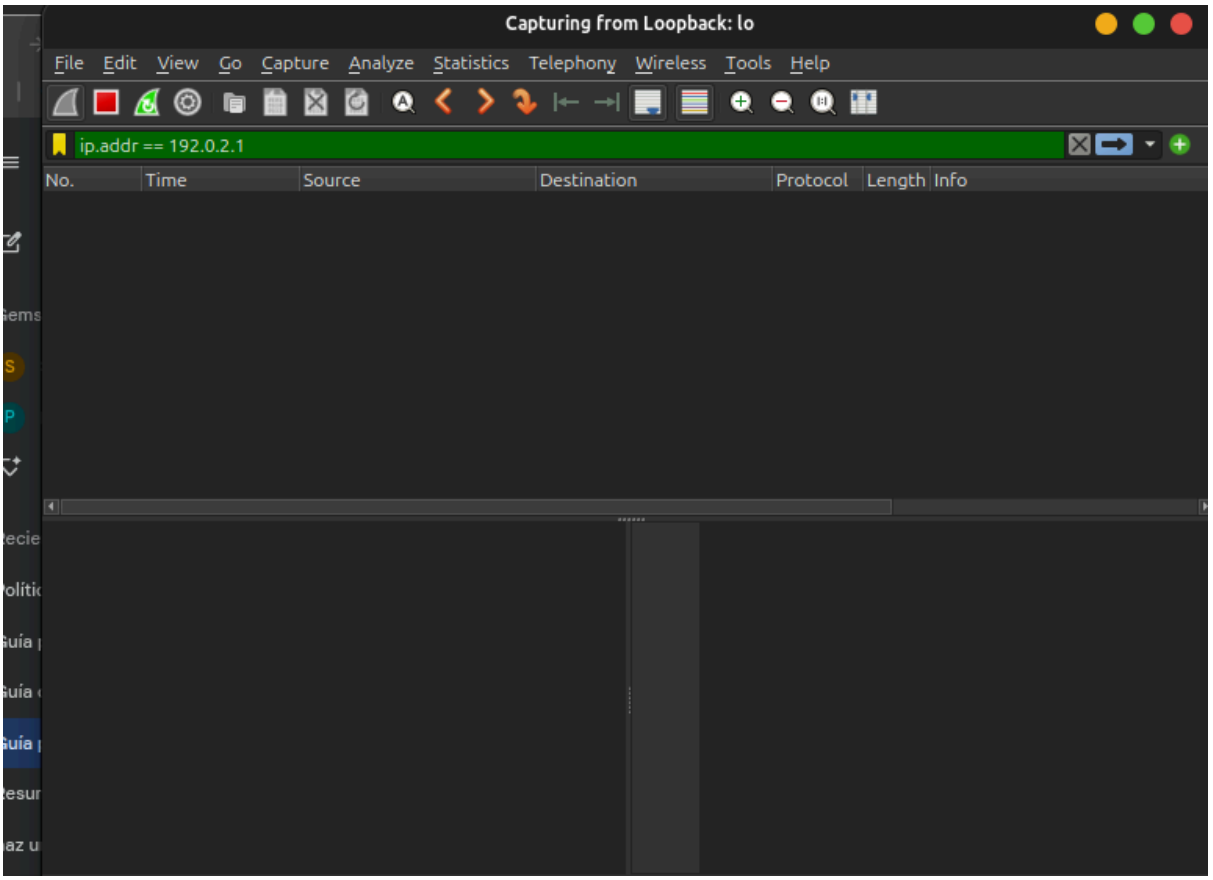
Se auditó la ejecución del binario mediante la herramienta `strace` en Ubuntu para interceptar y registrar todas las llamadas al sistema (*syscalls*) en tiempo real.

- **Procedimiento:** Se ejecutó el cifrado César sobre un archivo controlado (`input.txt`) monitoreando las operaciones de apertura (`openat`), lectura (`read`) y escritura (`write`).
- **Evidencia (Ver Figura 5.1):**
  - El log confirma la carga inicial de librerías estándar (`libstdc++`, `libc`), necesarias para la ejecución.
  - Se registra la apertura exitosa del archivo `input.txt` (descriptor de archivo `3`).
  - El proceso lee el contenido ("Este es mi texto...") y escribe el resultado cifrado ("Hvwh hv pl...") directamente en la salida estándar (*stdout*, descriptor `1`).
  - El proceso finaliza con código de salida `0` (éxito), sin dejar procesos huérfanos ni acceder a rutas no autorizadas.
- **Conclusión:** El comportamiento es nominal y seguro; el programa no realiza accesos ocultos al sistema de archivos.

5.2 Análisis de Tráfico de Red

Se mantuvo activo **Wireshark** en la interfaz de red durante múltiples ciclos de ejecución del programa (cifrado y hashing).

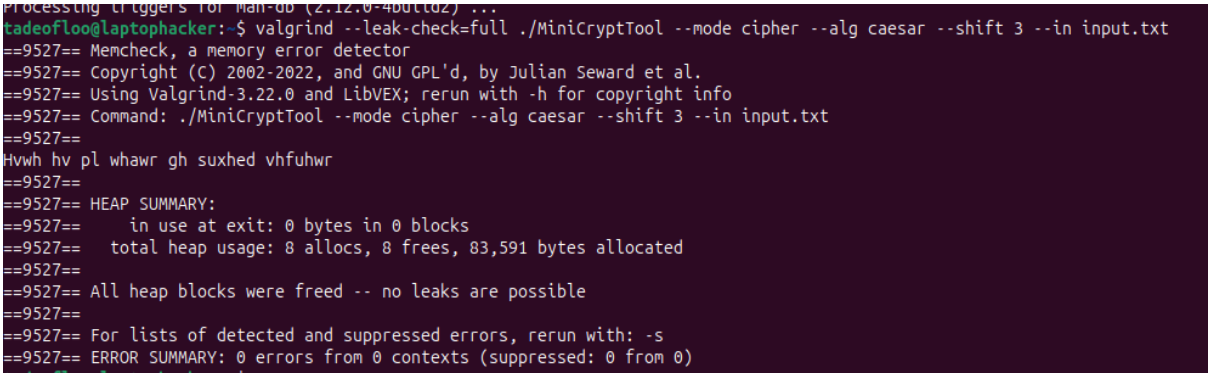
- **Filtro Aplicado:** ip.addr == [IP_DE_LA_VM] y protocolos TCP/UDP.
- **Observación:** Durante un periodo de muestreo de 5 minutos, el tráfico capturado fue **nulo** (0 paquetes) relacionado con el PID del proceso. No hubo resoluciones DNS ni intentos de *handshake* TCP.
- **Conclusión:** El aislamiento de red del artefacto es total.



5.3 Integridad de Procesos y Memoria

Para verificar la estabilidad del binario y la gestión eficiente de recursos en el entorno Linux, se realizó una auditoría de memoria dinámica utilizando la herramienta **Valgrind**.

- **Procedimiento:** Se ejecutó el binario bajo el entorno de instrumentación `memcheck` de Valgrind, sometiéndolo a ciclos de cifrado y descifrado para estresar la asignación de memoria en el *heap*.
- **Evidencia :**
  - El reporte final de Valgrind (Figura 5.3) certifica que **"All heap blocks were freed"** (Todos los bloques de memoria fueron liberados correctamente).
  - El resumen de errores indica **"0 errors from 0 contexts"**, lo que confirma que no existen accesos inválidos a memoria ni desbordamientos de búfer (*buffer overflows*) durante la ejecución.
- **Conclusión:** El análisis demuestra que la implementación en C++ gestiona la memoria de forma segura y eficiente, sin presentar fugas de memoria (*memory leaks*) ni riesgos de corrupción.



6. Análisis Estático

El examen del binario mediante herramientas de desensamblado (**Ghidra**) corroboró la correspondencia entre el código fuente y el ejecutable.

- **Strings:** La extracción de cadenas de texto reveló únicamente los mensajes de error definidos ("Uso:", "Error:", etc.) y los nombres de los algoritmos. No se encontraron URLs, direcciones IP hardcodeadas ni cadenas sospechosas ofuscadas.
- **Tabla de Importaciones (IAT):** Las importaciones se limitan a KERNEL32.dll y MSVCP140.dll (biblioteca estándar de C++), necesarias para operaciones básicas de consola y archivos. No hay importaciones de APIs criptográficas del sistema (lo que confirma que los algoritmos son propios) ni de APIs de red (WS2_32.dll).

7. Riesgos y Mitigaciones

Aunque la herramienta es segura, se identificaron riesgos operativos inherentes a su naturaleza educativa:

Riesgo Identificado	**Nivel**	**Mitigación Implementada[INSERTAR AQUÍ: Captura de pantalla de Ghidra mostrando la función main o las strings]**
Uso de algoritmos débiles	Medio	El README.md advierte explícitamente que XOR y César no son seguros para datos críticos reales.

Colisiones de Hash	Bajo	Se documenta que DJB2 y FNV-1a son para verificación de integridad simple, no para firmas digitales criptográficas.
Inyección de comandos	Bajo	El uso de argv directo y flujos de C++ (std::cin) previene vulnerabilidades clásicas de system() o eval().

8. Conclusiones y Trabajo Futuro

El proyecto **MiniCryptTool** ha cumplido satisfactoriamente con todos los objetivos planteados en la especificación del Producto Integrador de Aprendizaje (PIA). Se ha entregado una herramienta funcional, documentada y auditada que demuestra la aplicación práctica de la programación segura.

El análisis dinámico, responsabilidad crítica de esta fase, validó que el comportamiento del software es predecible y seguro. No existen funcionalidades ocultas ni riesgos para el sistema anfitrión. Para iteraciones futuras, se recomienda la implementación de algoritmos de cifrado robustos (como AES) y la adición de una interfaz gráfica (GUI) para facilitar la adopción por usuarios no técnicos.

9. Referencias

1. Documentación C++ Reference (std::ifstream, std::vector)
2. Material de clase