Reporte técnico: Proyecto final de Sistemas Operativos y Laboratorio

1. Información del Proyecto

- **Título del Proyecto:** Buffer Overflow
- **Curso/Materia:** Sistemas Operativos
- **Integrantes:** Assandry Enrique Barón Rodríguez (assandry.baron@udea.edu.co), Huber Steven Arroyave Rojas (huber.arroyave@udea.edu.co)
- **Fecha de Entrega:** 13/07/2025

2. Introducción

El desbordamiento de buffer pasa cuando un programa escribe más datos de los que puede almacenar en un área de la memoria (buffer), ocasionando que se sobreescriban otras partes de la memoria adyacente. Para explicar mejor esto, es como si se estuviera metiendo más ropa de la que cabe en una maleta y al final, algunas cosas van a terminar fuera de lugar. Eso es lo que sucede con el desbordamiento de buffer. Esto, ocasiona que se generen errores o incluso ataques maliciosos, los cuales pueden ocasionar algún fallo en el sistema o escalar los privilegios, otorgando al atacante control no autorizado sobre el sistema comprometido.
A pesar de que hoy en día existen diferentes medidas de protección incorporadas en los sistemas operativos modernos, todavía existen varios sistemas que siguen siendo vulnerables, debido al código heredado que no fue diseñado con las estrictas prácticas de seguridad actuales, o a implementaciones mal gestionadas de la memoria en aplicaciones nuevas. La naturaleza de lenguajes de bajo nivel como C y C++, que permiten un manejo manual y directo de memoria, si bien ofrecen un gran control y eficiencia, también introduce un mayor riesgo de errores si no se implementan validaciones de entrada y límites de buffer adecuados.

Por ende comprender esta problemática es fundamental para desarrollar software más robusto y para defender los sistemas informáticos en un entorno digital cada vez más complejo y amenazante.

2.1. Objetivo del Proyecto

Describa claramente el objetivo principal de tu proyecto. ¿Qué problema busca resolver o qué funcionalidad implementa? ¿Qué se espera demostrar con este proyecto?

- **Objetivo principal:** Realizar un ataque de desbordamiento de buffer en un entorno controlado, para poder entender su funcionamiento y cómo prevenirlo
- **Objetivos específicos:**
    - Investigar todo lo relacionado con los fundamentos teoricos y tecnicos del buffer overflow
    - Simular el ataque en un entorno controlado utilizando algunas herramientas como kali Linux e Immunity Debugger.
    - Identificar las direcciones de memoria y los offsets que puedan ser relevantes para el ataque
    - Crear un código malicioso para poder inyectar y ejecutar el ataque
    - Analizar y explorar diferentes soluciones para poder mitigar el riesgo de los desbordamientos de buffer, y evaluar su efectividad frente a los ataques simulados.

2.2. Motivación y Justificación

Explique por qué eligió este proyecto o tema. ¿Cuál es su relevancia en el contexto de los sistemas operativos? ¿Por qué es interesante o importante?

El tema del buffer overflow, fue elegido ya que es uno de los temas que posee una gran relevancia tanto histórica como actual en la seguridad informática. A pesar de los grandes avances tecnológicos y el desarrollo de varios mecanismos para la protección de los sistemas operativos, sin embargo, dicha vulnerabilidad sigue estando presente en el software heredado y en las nuevas aplicaciones que no implementan de una buena manera las prácticas de programación segura.

Además de eso, este proyecto, permite aplicar tanto los conceptos del curso de sistemas operativos, como la gestión de la memoria, el control de los procesos y la protección del sistema. También un poco del curso electivo ofrecido por la universidad de seguridad de la información, el cual se está viendo simultáneamente.

La ejecución del ataque en un entorno controlado, permite comprender como un error que parece simple, puede ocasionar varias consecuencias críticas. Esto generó una mejor conciencia a la hora de realizar un código, buscando que sea más seguro y a su vez, comprender los mecanismos de defensa, que implementa los sistemas actuales.

2.3. Alcance del Proyecto

Defina los límites del proyecto. ¿Qué funcionalidades o aspectos específicos aborda? ¿Qué queda fuera del alcance de esta implementación?

Este proyecto, se centra en la simulación de un buffer overflow tipo stack overflow en una máquina vulnerable, utilizando un entorno virtualizado y para esto, es necesario lo siguiente:

- La creación de un entorno controlado, siendo necesario, Kali linux, windows 7 y Brainpan en virtualbox)

- El uso de herramientas como GDB, Immunity Debugger, mona.py y scripts de Metasploit.
- Identificación de offsets, badchars y dirección de salto.
- Construcción y ejecución de un payload personalizado para explotar la vulnerabilidad.
- Análisis de técnicas de mitigación (ASLR, DEP, stack canaries).

Por otro lado, fuera del alcance, quedan las explotación de vulnerabilidades en los sistemas reales, también el uso de técnicas avanzadas de evasión de mitigaciones modernas y el desarrollo de exploits persistentes.

3. Marco Teórico / Conceptos Fundamentales

Explique brevemente los conceptos relevantes relacionados con el proyecto (Tecnologías empleadas, para que se emplean,, etc)

El buffer overflow ha sido responsable de diferentes ataques informáticos, los cuales han sido muy perjudiciales para el tema de la informática. Un ejemplo de esto fue el gusano Morris en el año de 1998, el cual aprovechó el desbordamiento de buffer para poder propagarse y causar interrupciones masivas en la red. Este problema, lo podemos encontrar en diferentes lenguajes como C y C++, debido a que en dichos lenguajes el manejo de la memoria es manual y no existen los controles automáticos de límite. La figura 1 muestra un ejemplo de un buffer overflow en un arreglo (array)

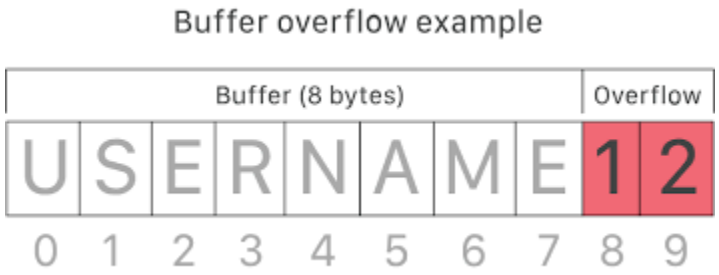

Figura 1: Ejemplo de un Buffer overflow

Para esta vulnerabilidad, existen varios tipos de ataques como el stack overflow. Esto sucede cuando se escribe más información en la pila de la que fue reservada, ocasionando que se sobreescriba la dirección de retorno almacenada por funciones. Permitiendo que el programa pueda ser engañado, para que se redirija hacia un código malicioso, como un shellcode inyectado en el mismo buffer.
El heap-based overflow, se produce cuando un programa escribe más datos en una región del heap, es decir, en la memoria dinámica, la cual es asignada por diferentes funciones como malloc(), lo cual puede corromper las estructuras de control, modificar los punteros y permitir la ejecución de código arbitrario.
También encontramos el string format vulnerability, el cual, ocurre cuando una función de formateo como el printf(), utiliza de una manera directa una entrada del usuario como cadena de formato. Ocasionando, que se afecte la integridad del programa, ya que esto, permite que se pueda leer o escribir en las direcciones de memorias arbitrarias.:

Para que estos ataques funcionen, es necesario entender cómo está organizada la memoria del sistema, principalmente en las arquitecturas como x86. Donde los registros EIP (Instruction Pointer) , el cual se encarga de controlar la ejecución de las instrucciones y ESP (Stack Pointer) que se encarga del acceso a la pila. Por lo cual, un atacante que sepa cómo funcionan dichos registros puede redirigir el flujo del programa mediante técnicas como Sleds de NOP, que son instrucciones que no hacen nada, usadas como alfombra para poder aterrizar de una forma segura en el shellcode. Offsets calculados para ubicar exactamente dónde escribir y la sobreescritura de direcciones de retorno que es cuando se escribe más allá del final de un buffer.

A pesar de que se han realizado grandes avances en la protección para poder evitar estos ataque, como la aleatorización del espacio de direcciones (ASLR), que es una técnica de seguridad en la cual, se aleatoriza las direcciones de la memoria, evitando que un atacante no sepa hacia dónde redirigir la ejecución, ya que las direcciones cambian cada vez que se ejecuta el programa. La prevención de la ejecución de los datos (DEP/NX bit), la cual, impide la ejecución de código áreas de la memoria, donde solo deberían contener solo datos. También está los canarios en pila (stack canaries), los cuales permiten detectar y prevenir los ataques de stack buffer overflow antes de que se pueda sobreescribir la dirección de retorno de una función. Todavía hay muchísimas aplicaciones modernas que aún contienen código vulnerable o que ejecutan bibliotecas inseguras.

En la tabla 1, se puede ver una comparativo entre los ataques mencionados anteriormente y las mitigaciones típicas:

Tipo de ataque	Memoria afectada	Impacto común	Mitigaciones típica
Stack Overflow	Pila	Ejecución de shellcode	Canarios, DEP
Heap-Based Overflow	Memoria dinámica	Corrupción de estructuras	Safe unlinking, ASLR
String Format Vulnerability	Memoria arbitraria	Lectura/escritura de memoria	Validación de formatos

Tabla 1: Comparación de los tipos de ataques

De acuerdo a lo visto en el curso de sistemas operativos, este proyecto se enfoca en comprender cómo las vulnerabilidades de memoria pueden ser aprovechadas para comprometer la seguridad de un sistema.

La gestión de procesos, la asignación de memoria, la ejecución de instrucciones y la protección de memoria son conceptos fundamentales, que son aplicables a la comprensión y explotación de los Buffer Overflow. Además, se analizaran las contramedidas implementadas por los sistemas operativos, como la aleatorización de espacio de direcciones (ASLR) y la protección de no ejecución (NX/DEP) para mitigar los ataque de desbordamiento de buffer.

4. Diseño e Implementación

4.1. Diseño de la Solución

Describa cómo diseñó su proyecto. Incluya diagramas (de bloques, de flujo, de estados) para ilustrar la arquitectura o el funcionamiento interno. Explique las decisiones de diseño clave que tomo y por qué.

ID	Tarea	Duración	Fecha de Inicio	Fecha de Fin
1	Fase 1: Investigación y Configuración			
1.1	Revisión Bibliográfica y Marco Teórico	7	20/05/2025	26/05/2025
1.2	Preparación del Entorno Virtual (VirtualBox, OS)	4	27/05/2025	30/05/2025
1.3	Instalación y Configuración de Herramientas de Ataque	5	31/05/2025	04/06/2025
2	Fase 2: Análisis y Explotación del Binario Vulnerable			
2.1	Fuzzing y Detección de Crash	6	05/06/2025	10/06/2025
2.2	Depuración con Immunity Debugger/GDB	7	11/06/2025	17/06/2025
2.3	Identificación de Offsets y Badchars	4	18/06/2025	21/06/2025
2.4	Generación de Shellcode (msfvenom)	3	22/06/2025	24/06/2025
2.5	Desarrollo del Exploit Script (Python)	5	25/06/2025	29/06/2025
2.6	Ejecución y Validación del Exploit	4	30/06/2025	03/07/2025
3	Fase 3: Análisis de Mitigaciones y Documentación			
3.1	Investigación y Análisis de Técnicas de Mitigación	2	04/07/2025	05/07/2025
3.2	Pruebas con Mitigaciones Activas (si aplica)	1	06/07/2025	06/07/2025
3.3	Redacción de Resultados y Conclusiones	2	07/07/2025	08/07/2025
3.4	Revisión y Edición Final del Documento	2	09/07/2025	10/07/2025

Tabla 2: Diagrama de Gantt

Para ver en mejor detalle Diagrama de Gantt

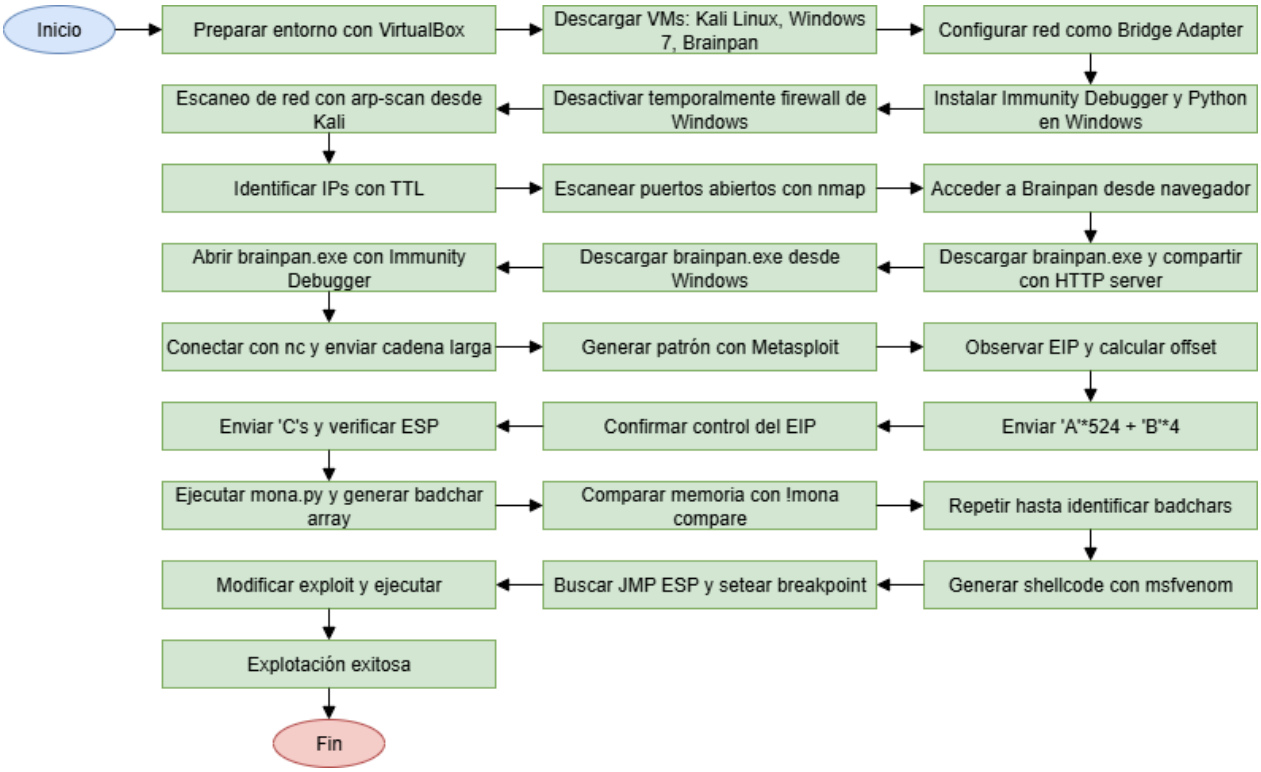

Figura 2: Diagrama de flujo

Para ver en mejor detalle: [Diagrama de Flujo]

4.2. Tecnologías y Herramientas

Enumere los lenguajes de programación, herramientas, entornos de desarrollo, o cualquier otra tecnología que utilizó para implementar su proyecto.

Durante el desarrollo del proyecto, se emplearon diversas tecnologías y herramientas especializadas para lograr los objetivos.

- Lenguaje de programación: Python (para los scripts de explotación)
- Sistema operativo anfitrión: Windows y Linux
- Máquinas virtuales: VirtualBox con imágenes de Kali Linux, Windows 7 y Brainpan
- Herramientas utilizadas:
    - gcc con flags -fno-stack-protector y -z execstack
    - GDB para debugging en Linux
    - Immunity Debugger en Windows
    - mona.py para análisis automatizado en Immunity Debugger
    - Metasploit (pattern_create.rb, pattern_offset.rb) para encontrar offset y badchars
    - msfvenom para generación de shellcodes
    - nmap, netcat, python3 -m http.server para escaneo y transferencia de archivos
    - Gobuster para fuzzing de directorios

4.3. Detalles de Implementación

Describa los aspectos más importantes de la implementación, es decir, hable sobre detalles técnicos sin ahondar mucho. ¿Cómo codificó las funcionalidades clave? Explique los aspectos básicos sobre archivos de configuración, las estructuras de datos o algoritmos principales que utilizo según aplique en la implementación de su proyecto. (Evite pegar todo el código aquí, para ello referencias a secciones en los anexos).

- **Desactivación de ASLR:** Se ejecutó echo 0 > /proc/sys/kernel/randomize_va_space para evitar la aleatorización de la memoria en los entornos de Linux.
- **Compilación del binario vulnerable:** Con la desactivación de protecciones como stack protector (gcc -fno-stack-protector -z execstack).
- **Análisis del binario vulnerable:** Mediante GDB e Immunity Debugger para identificar punto de crash.
- **Uso de patrones personalizados:** pattern_create.rb y pattern_offset.rb para identificar la longitud precisa hasta EIP.
- **Identificación de caracteres peligrosos (badchars):** Para evitar bytes que corrompan el payload.
- **Construcción del exploit final:** En Python, con sled de NOP, shellcode generado con msfvenom, y dirección de salto obtenida con mona.

5. Pruebas y Evaluación

5.1. Metodología de Pruebas

Describe cómo probaste tu proyecto. ¿Qué enfoque de pruebas utilizaste? ¿Creaste casos de prueba específicos?

Se realizaron las pruebas paso a paso, buscando replicar un entorno real de explotación en laboratorio. Se aplicaron algunas técnicas de fuzzing, depuración y la validación de los resultados esperados en cada etapa.

5.2. Casos de Prueba y Resultados

Presente los casos de prueba más importantes que ejecutó y los resultados obtenidos. Puede usar para ello una tabla como la siguiente:

desarrollo del proyecto de Buffer Overflow:

ID	Descripción del caso de prueba	Resultado esperado	Resultado obtenido	Éxito/Fallo
CP-001	Ejecutar brainpan.exe con Immunity Debugger y enviarlo a 'Running'	El binario debe quedar en estado Running listo para recibir datos	brainpan.exe en ejecución con Immunity y esperando conexión en puerto 9999	Éxito

CP-002	Conectarse desde Kali con netcat y enviar 1000 'A's	El programa debe crashear y pausar en Immunity	Se observa el cambio a 'Paused', y crash por desbordamiento de buffer	Éxito
CP-003	Enviar cadena patrón con pattern_create.rb	El EIP debe ser sobreescrito con un valor identificable del patrón	EIP muestra 35724134 (o valor derivado del patrón)	Éxito
CP-004	Calcular el offset con pattern_offset.rb	El offset al EIP debe calcularse correctamente	Offset calculado como 524	Éxito
CP-005	Enviar el payload: 524 'A' + 4 'B'	El EIP debe ser sobreescrito con 42424242 (ASCII de 'BBBB')	EIP = 42424242 → Confirmado control del EIP	Éxito
CP-006	Verificar la ubicación del ESP enviando 524 'A' + 4 'B' + 250 'C'	El registro ESP debe apuntar a la zona de los 'C'	ESP apunta correctamente → zona válida para shellcode	Éxito
CP-007	Ejecutar !mona jmp -r esp y establecer la dirección de salto (JMP ESP)	Encontrar la dirección válida sin badchars para redirigir flujo	Dirección encontrada: 311712f3	Éxito
CP-008	Generar bytearray con !mona bytearray -cpb "\x00" y comparar con !mona compare	Detectar los caracteres problemáticos (badchars)	Se identifica \x00 como badchar principal	Éxito
CP-009	Generar el shellcode con msfvenom sin badchars y añadirlo al payload	Shellcode debe poder insertarse correctamente en la pila	Shellcode inyectado sin problemas	Éxito
CP-010	Ejecutar exploit.py con shellcode y JMP ESP	Se debe recibir shell reversa en Kali	Reverse shell exitosa → conexión entrante recibida	Éxito
CP-011	Activar las mitigaciones (DEP/ASLR) y volver a ejecutar	El exploit no debe funcionar correctamente con las mitigaciones activas	El exploit falla → comportamiento esperado	Éxito
CP-012	Documentar cada etapa del ataque	Registro claro de pruebas y pasos realizados	Reporte técnico y las pruebas documentadas correctamente	Éxito

5.3. Evaluación del Rendimiento (si aplica)

Si su proyecto tiene aspectos de rendimiento (ej: tiempos de ejecución, uso de CPU/memoria), presente y analiza las métricas obtenidas.

5.4. Problemas Encontrados y Soluciones

Describa los problemas o errores significativos encontrados durante el desarrollo y las pruebas, y cómo los resolvió.

- Uno de los problemas, a los cuales se enfrentó, fue la dirección de salto invalida (jmp esp), para darle solución, se utilizó !mona jmp -r esp para poder encontrar una dirección válida sin badchars.
- Por otra parte los badchars en el shellcode impidan la ejecución y la solución para esto, fue que se ajustó la generación del shellcode excluyendo los bytes problemáticos

6. Conclusiones

- Resuma los logros de tu proyecto. ¿Cumpliste los objetivos iniciales?
- Discuta lo que aprendió al realizar el proyecto, especialmente en relación con los conceptos de sistemas operativos del curso.
- Evalúe el éxito general de su proyecto.

- El desarrollo de este proyecto, permitió

El desarrollo de este proyecto, permitió alcanzar de una manera satisfactoria, los objetivos planteados. A través de la simulación controlada, de un ataque de tipo buffer Overflow (Stack Overflow) en la aplicación vulnerable Brainpan, se logró:

- Comprender en profundidad la mecánica de una vulnerabilidad de desbordamiento de búfer.
- Manipular directamente los registros del sistema (EIP y ESP) para redirigir el flujo de ejecución del programa.
- Crear un exploit funcional, empleando técnicas como cálculo de offset, detección de badchars, uso de NOP sled y shellcode personalizado.
- Estudiar y aplicar contramedidas de protección modernas como ASLR, DEP y stack canaries.
- Validar la efectividad de dichas contramedidas al bloquear la ejecución del exploit bajo condiciones seguras.

Desde un punto de vista del aprendizaje, este trabajo, no solo refuerza los conceptos teóricos vistos en clase como la gestión de la memoria, los procesos y la seguridad en los sistemas operativos, sino que también potencia las habilidades prácticas como el análisis binario, la depuración, el scripting en Python y el manejo de entornos virtuales complejos.

Este ejercicio también permitió adquirir una mayor conciencia sobre la importancia de escribir código seguro, considerando que los errores como estos, pueden comprometer gravemente la integridad y confidencialidad de sistemas reales.

7. Trabajo Futuro (Opcional)

Sugiera posibles extensiones o mejoras que se podrían realizar en el proyecto en el futuro.

Como mejoras o extensiones futuras a este proyecto, se propone:

- Analizar las técnicas modernas de evasión de las mitigaciones como Return-Oriented Programming (ROP).

- Automatizar la generación del exploit mediante las herramientas personalizadas.
- Integrar el análisis con los motores de detección de intrusos (IDS), para poder registrar los intentos de explotación.
- Simular el ataque en arquitecturas distintas como ARM o MIPS.
- Explorar otras vulnerabilidades similares en los servicios reales, ya sea con fines educativos y éticos, en los entornos CTF.

8. Referencias

Enumere todas las fuentes que consultaste (libros, artículos, sitios web, documentación de herramientas, etc.) utilizando un formato de citación consistente.

- Cloudflare. (s.f.). *Buffer overflow*. Cloudflare. https://www.cloudflare.com/es-es/learning/security/threats/buffer-overflow/
- Foster, J. , Osipov, V., Bhalla, N., Heinen, N., & Liu, Y. (2005). *Buffer overflow attacks: Detect, exploit, prevent*. Syngress Publishing. https://repo.zenk-security.com/Techniques%20d.attaques%20%20.%20%20Failles/Buffer%20Overflow%20Attacks%20-%20Detect%20Exploit%20Prevent.pdf
- Rapid7. (s.f.). *Metasploit Framework*. Rapid7. https://docs.rapid7.com/metasploit/
- Immunity Inc. (s.f.). *Immunity Debugger*. https://www.immunityinc.com/products/debugger/
- GNU Project. (s.f.). *GDB: The GNU Project Debugger*. Sourceware. https://www.gnu.org/software/gdb/

9. Anexos (Opcional)

Incluya cualquier material adicional que sea relevante pero que no encaje en el cuerpo principal del reporte, como:

- Código fuente completo del proyecto.
- Diagramas detallados.
- Capturas de pantalla de la ejecución o resultados.
- Archivos de configuración.
- Resultados extensos de pruebas.

Link del repositorio: Repositorio
Link del video: Video
Link de Instalación de dependencias: Instalación de dependencias