Buffer Overflow

Sebastian Mora Piedrahita
Ingeniería de Sistemas
Universidad de Antioquia
sebastian.mora3@udea.edu.co

Neiver Tapia Escobar
Ingeniería de Sistemas
Universidad de Antioquia
neiver.tapia@udea.edu.co

Carlos Eduardo Castaño
Ingeniería de Sistemas
Universidad de Antioquia
carlose.castano@udea.edu.co

***Abstract*—The concept of "Buffer Overflow" is fundamental in computer security and occurs when a program attempts to write more data than reserved in a memory area, which can overwrite adjacent areas. This can be exploited by attackers to execute arbitrary code, corrupt data or cause system failures, compromising the integrity, confidentiality and availability of information.**

There are several types of attacks that exploit buffer overflow vulnerabilities, each with specific characteristics and methods. This report describes three of the most common and dangerous types:

- **Stack Overflow.**
- **Heap-Based Overflow (Heap-Based Overflow)**
- **String Format Vulnerability**

The objective of the report is to provide a detailed understanding of how these attacks are performed, how they exploit vulnerabilities in program memory management and what measures can be taken to mitigate these risks. Through examples and technical analysis, it will highlight the importance of secure programming and the implementation of robust development practices to prevent these devastating attacks.

Keywords—Buffer Overflow, Computer Security, Exploits, Attack Method

I. Introducción

El concepto de "Buffer Overflow" o desbordamiento de buffer es uno de los fundamentos más críticos y antiguos en el ámbito de la seguridad informática. Esta vulnerabilidad se produce cuando un programa intenta escribir más datos en un área de memoria de la que ha sido reservada para el mismo, causando que los datos sobrescriben áreas adyacentes de la memoria. Este comportamiento anómalo puede ser explotado por atacantes para ejecutar código arbitrario, corromper datos o provocar fallos en el sistema, comprometiendo la integridad, confidencialidad y disponibilidad de la información.

Existen varios tipos de ataques que explotan vulnerabilidades de buffer overflow, cada uno con características y métodos específicos. En este informe, se describen tres de los tipos más comunes y peligrosos de ataques de buffer overflow: Stack Overflow, Heap-Based Overflow y String Format Vulnerability.

El objetivo de este informe es proporcionar una comprensión detallada de cómo se realizan estos ataques, cómo explotan las vulnerabilidades inherentes en la gestión de memoria de los programas y qué medidas se pueden tomar para mitigar estos riesgos. A través de ejemplos y análisis técnicos, se destacará la importancia de la programación segura y la implementación de prácticas de desarrollo robustas para prevenir estos ataques devastadores.

II. Marco Teórico

A. Contexto

El buffer overflow, o desbordamiento de buffer, es una de las vulnerabilidades más antiguas y críticas en la historia de la seguridad informática. Este problema surge principalmente debido a la falta de validación adecuada de la entrada de datos en los programas, permitiendo que se escriba más información en un buffer de la que este puede manejar. Desde los inicios de la programación en lenguajes como C y C++, donde la gestión de memoria es manual, los desbordamientos de buffer han sido una amenaza constante.

Históricamente, los buffer overflows han sido responsables de algunos de los ataques más devastadores y notorios en la informática, como el famoso gusano Morris en 1988, que aprovechó un desbordamiento de buffer para propagarse y causar interrupciones masivas en la red. Este tipo de vulnerabilidad explota la memoria de un sistema, permitiendo a los atacantes sobrescribir datos críticos, corromper programas y ejecutar código malicioso.

En el contexto moderno, a pesar de los avances en técnicas de programación segura y la implementación de mecanismos de protección como Address Space Layout Randomization (ASLR) y Data Execution Prevention (DEP), los desbordamientos de buffer siguen siendo relevantes. Esto se debe a que muchas aplicaciones heredadas y sistemas embebidos aún contienen código vulnerable y porque nuevos errores en la gestión de memoria continúan apareciendo en software actual.

B. Descripción del Problema

El desbordamiento de buffer representa una amenaza significativa debido a la forma en que interactúa con la memoria del sistema. Cuando un programa no controla adecuadamente la cantidad de datos que se escriben en un buffer, estos datos pueden sobrescribir regiones adyacentes de memoria, lo que puede llevar a varios problemas graves:

- Ejecución de Código Arbitrario: Un atacante puede inyectar código malicioso en la memoria y alterar el flujo de ejecución del programa para ejecutar ese código. Esto puede dar lugar a la toma de control del sistema afectado.

- Corrupción de Datos: Los datos críticos en memoria pueden ser sobrescritos, llevando a la

corrupción de archivos, pérdida de datos y comportamiento inesperado del programa.

- Interrupciones del Sistema: La sobrescritura de regiones de memoria puede causar fallos en el sistema, haciendo que el programa o incluso el sistema operativo se bloqueen.

- Escalada de Privilegios: Mediante la manipulación de datos en memoria, un atacante puede elevar sus privilegios en el sistema, obteniendo acceso a recursos y datos que de otro modo estarían protegidos.

El problema se agrava debido a la diversidad de escenarios en los que puede ocurrir un desbordamiento de buffer. Por ejemplo, en un desbordamiento de pila (stack overflow), los datos pueden sobrescribir la dirección de retorno de una función, redirigiendo la ejecución del programa. En un desbordamiento en el heap (heap-based overflow), la estructura de memoria dinámica puede ser corrompida, permitiendo la manipulación de punteros y la ejecución de código arbitrario. Además, las vulnerabilidades de formato de cadena (string format vulnerabilities) pueden ser explotadas para leer o escribir datos arbitrarios en la memoria, utilizando funciones de formateo de cadenas mal diseñadas.

Dado el impacto potencial de estos ataques, es crucial que los desarrolladores comprendan las causas y consecuencias de los desbordamientos de buffer, y adopten prácticas de programación segura, incluyendo la validación rigurosa de entradas y el uso de herramientas y técnicas modernas de mitigación de riesgos.

C. Conceptos Básicos

- EIP: el puntero de instrucción extendido. Cuando se llama a una función, este puntero se guarda en la pila para su uso posterior. Cuando la función regresa,esta dirección guardada se utiliza para determinar la ubicación de la siguiente instrucción ejecutada.

- ESP: el puntero de pila extendido, que apunta a la posición actual en la pila y permite añadir y quitar cosas de la pila mediante operaciones push y pop o manipulaciones directas del puntero de pila.

- Shellcode: Tradicionalmente, shellcode es el código de bytes que ejecuta un shell. Shellcode ahora tiene un significado más amplio, para definir el código que se ejecuta cuando un exploit tiene éxito. El propósito de la mayoría es devolver una dirección de shell, pero existen muchos shellcodes con otros propósitos, como salir de un shell chroot, crear un archivo y delegar llamadas al sistema.

- Exploit: normalmente, un programa muy pequeño que cuando se utiliza provoca que una vulnerabilidad del software se active y sea aprovechada por el atacante.

- Buffer: un búfer es un área de memoria asignada con un tamaño fijo. Se suele utilizar como zona de retención temporal cuando se transfieren datos entre dos dispositivos que no funcionan a la misma velocidad o carga de trabajo. Los buffers dinámicos se asignan en el heap utilizando malloc. Cuando se definen variables estáticas, el búfer se asigna en la pila.

- Debugger: un depurador es una herramienta de software que, o bien se conecta al entorno de ejecución de la aplicación que se está depurando, o bien actúa de forma similar (o como) una máquina virtual para que el programa se ejecute dentro de ella. El software permite depurar problemas dentro de la aplicación que se está depurando. El depurador permite al usuario final modificar el entorno, como la memoria, en el que la aplicación se basa y está presente.

- Little Endian: little y big endian se refiere a aquellos bytes que son los más significativos. En un sistema little-endian, el byte menos significativo se almacena primero. x86 utiliza una arquitectura little-endian.

- Stack: La pila es un área de memoria utilizada para almacenar datos temporales. Crece y decrece a lo largo del tiempo de ejecución de un programa. Los desbordamientos de búfer más comunes se producen en el área de memoria de la pila. Cuando se produce un desbordamiento del búfer, los datos se sobrescriben en la dirección de retorno guardada, lo que permite a un usuario malintencionado hacerse con el control.

- Máquina virtual: Una máquina virtual es una simulación de software de una plataforma que puede ejecutar código.Una máquina virtual permite que el código se ejecute sin estar adaptado al procesador de hardware específico.Esto permite la portabilidad y la independencia de la plataforma del código.

III. Stack Overflow

Stack overflow (desbordamiento de la pila): Ocurre cuando un programa almacena una estructura de datos (como un buffer de cadena) en la pila, la cual también se usa en las arquitecturas de CPU comunes para guardar direcciones de retorno de llamadas a procedimientos, y no verifica la cantidad de bytes copiados en esta estructura. Cuando se copian datos excesivos a la pila, los bytes extra pueden sobrescribir otros datos, incluyendo la dirección de

retorno almacenada. Si el contenido de este buffer ha sido diseñado de una manera específica, esto puede causar que el programa ejecute un código proporcionado por un atacante dentro de este buffer.

Los desbordamientos de buffer en la pila (stack-based buffer overflows) fueron identificados como una clase separada de vulnerabilidad en 1996. Desde la publicación del artículo "Smashing the Stack for Fun and Profit" por Aleph, estos desbordamientos han sido considerados como el tipo más común de errores de programación explotables remotamente encontrados en aplicaciones de software.[1]

Aunque los desbordamientos de pila han sido el enfoque principal de la educación sobre vulnerabilidades de seguridad, estos errores se están volviendo menos comunes en el software convencional. Sin embargo, aún es importante estar consciente de ellos y buscarlos.

A. Conceptos generales de explotación

Explotar una vulnerabilidad en cualquier plataforma requiere planificación. Al controlar el registro EIP (Instruction Pointer), puedes redirigir la ejecución del código. Esto se logra usualmente apuntando el EIP a un código escrito por el atacante, conocido como payload.

- Técnicas de Inyección de Buffer: Para crear un exploit, primero necesitas encontrar una manera de colocar tu buffer grande en el buffer vulnerable. Esto puede implicar llenar un buffer a través de la red o escribir un archivo que luego sea leído por el proceso vulnerable.

- Optimización del vector de inyección: El vector de inyección es el código personalizado que necesitas para controlar el puntero de instrucción en la máquina remota. El payload, por otro lado, es el código que deseas ejecutar y debería funcionar de manera confiable, independientemente de cómo se haya inyectado.

B. Payloads

Un payload (carga útil) es el código que se inyecta en un sistema vulnerable con el objetivo de realizar una acción específica una vez que se ha logrado explotar una vulnerabilidad. La función principal del payload es ejecutar comandos o realizar operaciones que permitan al atacante controlar el sistema o filtrar información. Los payloads pueden variar en complejidad y funcionalidad, desde simples comandos que abren una shell hasta programas más complejos que pueden instalar malware, modificar configuraciones del sistema o robar datos sensibles.

Métodos para ejecutar un payload:

- Salto Directo (Adivinando Desplazamientos): este método implica decirle al código desbordado que salte directamente a una ubicación específica en la memoria. Sin embargo, la dirección de la pila puede cambiar, lo que hace que este método sea menos confiable.

- Retorno Ciego: La instrucción RET carga el registro EIP con lo que apunta el registro ESP (stack pointer), lo que permite redirigir la ejecución a una nueva dirección de código.

- Pop Retorno:Si la dirección en la parte superior de la pila no apunta al buffer del atacante, el EIP inyectado puede apuntar a una serie de instrucciones POP seguidas de un RET. Esto hará que la pila se despliegue varias veces antes de usar un valor para el EIP

- Llamada a Registro: Si un registro ya está cargado con una dirección que apunta al payload, el atacante puede simplemente cargar el EIP con una instrucción que realice una llamada a ese registro.

- Retorno de Empuje:Este método es similar al de llamada a registro pero utiliza una instrucción PUSH seguida de un RET.

C. Offset

El término offset se usa principalmente en desbordamientos de buffer locales (no remotos). En un entorno UNIX, los atacantes suelen compilar sus exploits directamente en la máquina que quieren atacar. Tienen una cuenta de usuario y usualmente buscan obtener permisos de root ejecutando un programa SUID root que abra una shell. El código del inyector para un exploit local calcula la base de su propia pila y asume que el programa atacado tiene la misma base. El atacante puede especificar el offset desde esta dirección para un salto directo. Si todo funciona correctamente, el valor base+offset del código atacante coincidirá con el del código víctima.

D. Sled de No Operación (NOP)

Al inyectar código usando una dirección directa, debes adivinar exactamente dónde está tu payload en la memoria, lo cual es casi imposible. La ubicación del payload no siempre será la misma. Para minimizar este efecto y disminuir la precisión necesaria, se usa el sled de NOP. Un NOP es una instrucción que no hace nada, solo ocupa espacio. Al llenar el buffer con NOPs antes del payload, si adivinas incorrectamente la dirección del payload, no importará mientras adivines una dirección que caiga en el sled de NOP. El buffer lleno de NOPs permite que cualquier dirección que apunte al buffer comience a ejecutar los NOPs hasta llegar al payload real. Cuanto más grande sea el buffer de NOPs, menos precisa debe ser la adivinanza de la dirección del payload.

E. Medidas de mitigación

- Control de Datos en Código: Los desarrolladores pueden especificar la longitud máxima de los datos que pueden ser copiados en el código, basándose en el tamaño del buffer de destino en lugar de en los datos mismos.

- Biblioteca Dinámica Segura: Se pueden crear bibliotecas dinámicas más seguras y hacer que los programas se enlacen dinámicamente a estas funciones en lugar de cambiar el programa original.

- Analizador Estático de Programas: Herramientas que advierten a los desarrolladores sobre patrones en el código que podrían llevar a vulnerabilidades de desbordamiento de búfer.

- Lenguaje de Programación: Lenguajes como Java y Python proporcionan verificación automática de límites, eliminando la necesidad de que los desarrolladores se preocupen por los desbordamientos de búfer.

- Compilador: Los compiladores pueden controlar el diseño de la pila e insertar instrucciones en el binario para verificar la integridad de la pila y eliminar las condiciones necesarias para los ataques de desbordamiento de búfer.

- Sistema Operativo: Implementa medidas como la Aleatorización del Espacio de Direcciones (ASLR), que dificulta a los atacantes adivinar la dirección correcta de la memoria del programa.

- Arquitectura de Hardware: La tecnología NX bit, que separa el código de los datos en la memoria, puede prevenir los ataques de desbordamiento de búfer en la pila. Sin embargo, puede ser superado por ataques de retorno a la librería.

IV. String Format Vulnerability

Las vulnerabilidades de cadena de formato (Format String Vulnerabilities) son un tipo de falla de seguridad en aplicaciones de software que permiten a los atacantes manipular el comportamiento del programa explotando errores en el manejo de las cadenas de formato. Estas vulnerabilidades suelen ocurrir cuando los datos no confiables controlados por el usuario se utilizan en funciones que manejan cadenas de formato, como *printf* en C, sin la validación o el saneamiento adecuado.

A. Contexto y Origen

Las funciones de formato de cadenas, como *printf*, *sprintf*, y similares, se utilizan ampliamente en la programación para generar cadenas de texto con formato. Estas funciones permiten a los desarrolladores especificar cómo deben aparecer los valores de las variables en la salida, utilizando especificadores de formato como *%d* para enteros, *%s* para cadenas, y *%x* para valores en hexadecimal. Cuando se emplean correctamente, estas funciones son herramientas poderosas para la generación de texto dinámico.

Sin embargo, cuando los datos de entrada controlados por el usuario se pasan directamente como la cadena de formato a estas funciones sin una validación adecuada, los atacantes pueden aprovechar esta situación para ejecutar código arbitrario, leer datos sensibles de la memoria del programa, o causar que el programa falle. Este tipo de vulnerabilidad fue identificado por primera vez a finales de la década de 1990 y ha sido una preocupación significativa en la seguridad de software desde entonces.

B. Impacto y Relevancia

La explotación de una vulnerabilidad de cadena de formato puede tener consecuencias graves, incluyendo la ejecución de código arbitrario, la corrupción de datos, la revelación de información sensible, y la interrupción del servicio. Esto se debe a que las cadenas de formato permiten el acceso y la manipulación de la memoria del programa, lo que puede ser utilizado para modificar variables y estructuras de datos críticas.

V. Heap-Based Overflow

El Heap-Based Overflow o desbordamiento basado en el heap, es otro tipo crítico de vulnerabilidad en la seguridad informática. A diferencia del desbordamiento de la pila (stack overflow), que afecta a la memoria utilizada para almacenar variables locales y direcciones de retorno de funciones, el heap-based overflow afecta a la memoria dinámica, que se asigna y gestiona en tiempo de ejecución.

A. Contexto y Funcionamiento

En muchas aplicaciones, la memoria dinámica se gestiona mediante el uso de funciones como malloc en C, que asignan bloques de memoria en el heap. Estos bloques se utilizan para almacenar estructuras de datos que necesitan ser accesibles durante la vida útil del programa. Un desbordamiento de buffer en el heap ocurre cuando un programa escribe más datos de los que se han asignado en un bloque de memoria del heap, sobrescribiendo así datos adyacentes.

B. Consecuencias y Exploits

Un desbordamiento en el heap puede tener varias consecuencias peligrosas:

- Ejecución de Código Arbitrario: Un atacante puede diseñar la entrada de tal manera que sobreescriba punteros de función o estructuras de control en el heap, permitiéndole redirigir la ejecución del programa a su propio código malicioso.
- Corrupción de Datos: El desbordamiento puede sobrescribir datos críticos almacenados en el heap, causando corrupción de datos y comportamientos inesperados del programa.
- Interrupciones del Sistema: La sobrescritura de regiones de memoria puede causar fallos en el sistema, haciendo que el programa o incluso el sistema operativo se bloqueen.

C. Medidas de Mitigación

- Para mitigar los riesgos asociados con el heap-based overflow, los desarrolladores deben adoptar prácticas de programación segura, tales como:

- Validación de Entradas: Validar y limitar la longitud de los datos de entrada antes de copiarlos en buffers.

- Uso de Funciones Seguras: Utilizar funciones seguras para la manipulación de strings, como strncpy en lugar de strcpy.

- Mecanismos de Protección: Implementar mecanismos de protección como canaries y ASLR (Address Space Layout Randomization) para detectar y prevenir sobreescrituras de memoria.

VI. Conclusiones

- Prevenir los desbordamientos de pila, se pueden implementar varias medidas de mitigación, como el control de datos en el código, el uso de bibliotecas dinámicas seguras, el análisis estático de programas, el uso de lenguajes de programación con verificación automática de límites, la optimización del compilador y la implementación de medidas de seguridad a nivel de sistema operativo y arquitectura de hardware, como ASLR y NX bit.

- Los desbordamientos de buffer en la pila han sido identificados como una clase separada de vulnerabilidad desde 1996. Son considerados los errores de programación más comunes y explotables remotamente en aplicaciones de software.

- A pesar de los avances en la seguridad de software, las vulnerabilidades de desbordamiento de buffer, como stack overflow, heap-based overflow y string format vulnerabilities, continúan siendo una preocupación crítica en el campo de la seguridad informática. Estas vulnerabilidades representan una amenaza significativa para la integridad, confidencialidad y disponibilidad de la información en sistemas informáticos.

Bibliografía

[1] Buffer Overflow Attacks. Elsevier, 2005

[2] Computer Security: A Hands-on Approach, Wenliang Du.