Criptografía y Seguridad

Explotación de Vulnerabilidades: Buffer Overflow

Facultad de Ciencias
Universidad Nacional Autónoma de México

2 de marzo de 2023

---

1. Objetivos

Generales:

La explotación de una vulnerabilidad, es el elemento básico del hacking. Todo programa consiste en un conjunto de reglas que, dada una entrada, nos responde una única salida. Estas reglas siguen un cierto flujo de ejecución que le dice a la computadora que hacer. Hallar la vulnerabilidad de un programa y explotarla consiste en hacer que el programa haga lo que deseamos, incluso si este no fue diseñado para no hacerlo en un principio.

Las vulnerabilidades en un programa pueden tener graves consecuencias tanto para las personas como para las organizaciones. Pueden provocar violaciones de datos, mal funcionamiento del sistema o incluso ataques maliciosos. Por lo tanto, es esencial comprender las vulnerabilidades potenciales de un programa y cómo se pueden prevenir.

El objetivo general de esta práctica es que el alumno desarrolle una serie de exploits que rompan una serie de programas escritos en C, de manera que logren aprovecharse de la vulnerabilidad de Buffer Overflow.

Particulares:

2. Requisitos

- **Conocimientos previos:** Conocimientos de arquitectura, organización de computadoras y lenguaje ensamblador. Manejo de bash y C.
- **Tiempo de realización sugerido:**
  8 horas.
- **Número de colaboradores:**
  2
- **Software a utilizar:**
  - gcc, gbd, bash, python y C

3. Planteamiento

La vulnerabilidad de Desbordamiento de Búfer o Buffer Overflow, es un problema de seguridad de la memoria en donde el software o programa no considera o verifica sus límites de almacenamiento. Esta vulnerabilidad puede ser explotada por los piratas informáticos para obtener acceso a un sistema informático, permitiéndoles insertar código malicioso en el sistema y provocar el robo o la destrucción de datos.

Las vulnerabilidades de desbordamiento de búfer han existido desde los primeros días de las computadoras y hasta hoy en día, provocan dolores de cabeza a los desarrolladores de software. La mayoría de los gusanos de Internet utilizan la vulnerabilidad de desbordamiento de búfer para propagarse y las consecuencias de un ataque exitoso pueden ser graves, por lo que es esencial que se tomen medidas contra este tipo de ataques.

C es un lenguaje de programación en el cual se puede generar fácilmente programas que son vulnerables al desbordamiento de búfer y esto se debe a su simplicidad y a que el lenguaje asume que el programador es responsable de la integridad de los datos. Si no hiciera esto y trasladará la responsabilidad al compilador, los archivos binarios resultarían más lentos debido a las comprobaciones de integridad de cada variable, y esto haría más complejo el lenguaje.

Si bien arreglar el problema en C nos provoca más daños que beneficios, se dejó mejor la responsabilidad de generar un código seguro en el diseñador del programa. Por lo tanto, para diseñar un código seguro, primero debemos entender como es que funciona la vulnerabilidad, como explotarla y finalmente arreglarla.

La vulnerabilidad de desbordamiento de búfer se genera cuando la memoria del programa recibe una cantidad de datos mayor a la que realmente puede procesar de acuerdo a como fue desarrollado. Sucede principalmente al usar funciones de C de bajo nivel inseguras como ***gets***, ***strcat*** y ***sprintf***. Estas funciones, principalmente se utilizan para escribir una cadena o variable en una parte de la memoria en una longitud determinada. Si intentamos escribir algo que sea más grande que lo declarado, puede sobreescribir las direcciones de memoria posteriores y con eso inyectar código malicioso.

4. Desarrollo

La práctica consiste en crear una serie de expliots que ataquen a 2 programas en C, los cuales fueron programados de tal manera que son vulnerables, utilizando funciones inseguras despreciadas incluso por el mismo compilador de *gcc*.

Los programas que se van a explotar se encuentran cada uno dentro de una subcarpeta, y dentro de la carpeta *Practica02*.

Lo que se pretende es que se realice un pentesting de caja blanca, en el cual tienen acceso total al código fuente de los siguientes programas:

- simple-password-verification.c
- simple-verification.c

Para crear tu exploit, lo primero que deberás hacer es seguir los pasos para la realización de pentesting. La fase de recopilación de información ya la tienes en su mayoría, ya que cuentas con el código fuente a vulnerar, ahora solo falta el análisis de la vulnerabilidad, él ¿cómo?, y ¿dónde?.

Para esta etapa tendrás que analizar en que parte, el búfer sobreescribe el puntero base, con el código fuente puedes darte una idea de en que momento sucede el desbordamiento. Otro método es utilizar una cadena con "A" repetidas, las n-veces que sea necesario hasta desbordar el búfer. La letra "A" es una gran aliada, ya que es fácil de identificar en las direcciones de memoria, ya que en hexadecimal se representa como "0x41" y si al momento de analizar vemos series de "0x41414141" sabremos que es el búfer sobreescribiendo el stack donde se guarda nuestro código.

Utiliza*gdb* (Gnu Project Debugger) una herramienta que permite entre otras cosas, correr el programa con la posibilidad de detenerlo cuando se cumple cierta condición, avanzar paso a paso, analizar que ha pasado cuando un programa se detiene o cambiar algunas cosas del programa como el valor de las variables.

Con *gdb* podrás correr el programa con tu exploit, y recibir mensajes más explícitos de falla de segmentación y que fue lo que intento hacer el computador cuando eso sucedió (análisis de stack). También podrás hacer puntos de quiebre o breaks, para analizar la pila de llamadas antes y después de ingresar datos y así encontrar el puntero de retorno.

Cuando encuentres el puntero de retorno y logres sobreescribirlo deberás poder realizar un exploit que alimente tu código y corrompa la pila al sobreescribirla, dándote la posibilidad de saltarte las verificaciones de los códigos y garantizar el acceso a las funciones internas.

5. Entrada

Deberás crear dos archivos de Python llamados miniexploit-pass.py y miniexploit-ver.py, que vulnere simple-password-verification.c y simple-verification.c respectivamente. De manera que si al momento de correr el ejecutable y alimentarlo con este archivo, seas capaz corromper la pila al sobreescribirla.

6. Salida

El ejecutable junto con el exploit deberán terminar la ejecución del código y saltarse la validación a pesar del segmentation fault, mostrando el mensaje "Felicidades lograste llegar hasta aquí". También deberás crear los mismos archivos en C una vez analizado sus vulnerabilidades, de manera que no sean vulnerables a Buffer Overflow.

7. Procedimento

Se entregará un pdf debidamente documentado con tu análisis de los archivos vulnerables. Deberás crear el tu reporte con los siguientes apartados:

- **Recopilación**: En esta fase analizarás tus herramientas, el código y todo lo que cuentas para llevar a cabo la practica.
- **Análisis**: Identificarás la vulnerabilidad y como aprovecharla. Documentarás todo lo que llegues a encontrar.
- **Explotación**: En esta etapa documentarás la creación de tu exploit, como llegaste al puntero base, y como usaste las herramientas para lograrlo.

- **Post-explotación**: En esta fase explicarás tu exploit resultante, también las conclusiones a las cuales llegaste. Explicarás como corregir el código de la práctica y anexarás la corrección en su respectiva carpeta.
- **Imagina**: En esta fase intenta imaginar todo lo que podrías hacer utilizando esta vulnerabilidad. Explícalo detalladamente y reflexiona el alcance de esta vulnerabilidad.

Recuerda agregar la bibliografía correspondiente o la practica se evaluará en cero y tus datos completos y debidamente explicados. La tarea deberá ser entregada en tiempo y forma. En el Classroom únicamente uno del equipo subirá un .zip con todos los archivos y el pdf con los datos completos del equipo (nombre y número de cuenta).

8. Preguntas

1. ¿Qué significa que un pentesting sea de Caja Blanca?
2. Crea un diagrama explicando, como funciona el bloque de RAM de una computadora. Donde está el kernel, stack, el heap, etc.
3. Menciona el significado de los siguientes registros.
    - EAX
    - EBX
    - ECX
    - EDX
    - ESI
    - EDI
    - EBP
    - ESP
    - EIP
4. ¿Qué es little endian y big endian? ¿Cuál usa tu procesador?
5. ¿Qué es un segmento y que es un offset?
6. ¿Qué arquitectura tiene tu computadora?
7. ¿Crees que esta vulnerabilidad desapareció con las nuevas funciones seguras como *fgets*?

9. Bibliografía

Erickson, J. (2007). Hacking: The art of exploitation, 2nd edition: The art of exploitation. No Starch Press.

Cómo usar GDB. (s/f). Edu.ar. Recuperado el 25 de febrero de 2023, de https://lihuen.linti.unlp.edu.ar/index.php/Cómo_usar_GDB

(S/f). Softtek.com. Recuperado el 25 de febrero de 2023, de https://blog.softtek.com/es/explotación-de-software-en-arquitecturas-x86-i-introducción-y-explotación-de-un-buffer-overflow