TEC | Tecnológico de Costa Rica

Tarea 2

Vulnerabilidad de Condición de Carrera

Mayo, 2025

Objetivo

Desarrollar una aplicación que permita comprender al estudiante el los riesgos de las condiciones de carrera.

Datos Generales

- **Fecha de Entrega:** Lunes 2 de Junio de 2025 antes de las 23:59:59 GMT-6.
- **Fecha de Revisión:** Asíncrona.
- **Lenguaje:** C
- **Recurso Humano:** Grupos de 3
- **Valor de la asignación:** 10 %

Profesor

Kevin Moraga
kmoraga@tec.ac.cr
Escuela de Computación

Introducción

Una condición de carrera ocurre cuando varios procesos acceden y manipulan los mismos datos al mismo tiempo, y el resultado de la ejecución depende del orden particular en el que se produce el acceso. Si un programa privilegiado tiene un vulnerabilidad de condición de carrera, los atacantes pueden ejecutar un proceso paralelo para "competir" contra el programa privilegiado, con la intención de cambiar el comportamiento del programa.

Definición

Introducción

El siguiente código es parte de un programa **Set-UID** (root es el propietario); agrega un *string* de entrada del usuario al final de un archivo temporal `/tmp/XYZ`. Dado que el código se ejecuta con privilegios de root, se verifica cuidadosamente si el usuario realmente tiene permiso de acceso al archivo `/tmp/XYZ`; ese es el propósito de la llamada `access()`.

Una vez que el programa se ha asegurado de que el usuario tiene el derecho, el programa abre el archivo y escribe la entrada del usuario en él.

Parece que el programa no tiene ningún problema a primera vista. Sin embargo, existe una vulnerabilidad de condición de carrera en este programa: debido a la ventana (el retraso simulado) entre la comprobación (`acceso`) y el uso (`fopen`), existe la posibilidad de que el archivo utilizado por el `access` sea diferente del archivo utilizado por `fopen`, aunque tengan el mismo nombre de archivo `/tmp/XYZ`. Si un atacante malicioso de alguna manera puede hacer que `/tmp/XYZ` sea un enlace simbólico que apunte a `/etc/shadow`, el atacante puede hacer que la entrada del usuario se agregue a `/etc/shadow` (tenga en cuenta que el programa se ejecuta con privilegios de root y, por lo tanto, puede sobrescribir cualquier archivo).

Programa Vulnerable

El siguiente programa es un programa aparentemente inofensivo. Contiene una vulnerabilidad de condición de carrera.

```
/* vulp.c*/

#include <stdio.h>
#include<unistd.h>

#define DELAY 10000

int main()
{
    char * fn = "/tmp/XYZ";
    char buffer[60];
    FILE *fp;
    long int i;

    /* get user input */
    scanf("%50s", buffer );

    if (!access(fn, W_OK)) {

        /* simulating delay */
        for (i = 0; i < DELAY; i++) {
            int a = i^2;
        }

        fp = fopen(fn, "a+");
        fwrite("\n", sizeof(char), 1, fp);
        fwrite(buffer, sizeof(char), strlen(buffer), fp);
```

```
        fclose(fp);
    }
    else printf("No permission \n");
}
```

Tareas

Tarea 1: Explotar las vulnerabilidades de la condición de carrera

Debe explotar la vulnerabilidad de la condición de carrera en el programa Set-UID anterior. Más específicamente, usted necesita lograr lo siguiente:

1. Sobrescriba cualquier archivo que pertenezca a `root`.
2. Obtener privilegios de `root`; es decir, debería poder hacer cualquier cosa que `root` pueda hacer.

Tarea 2: Mecanismo de protección A - Repetición

Deshacerse de las condiciones de carrera no es fácil, porque el patrón de verificar-y-usar a menudo es necesario en los programas. En lugar de eliminar las condiciones de carrera, podemos agregar más condiciones de carrera, de modo que para comprometer la seguridad del programa, los atacantes deben ganar todas estas condiciones de carrera. Si estas condiciones de carrera se diseñan correctamente, podemos reducir exponencialmente la probabilidad de victoria de los atacantes. La idea básica es repetir `access()` y `open()` varias veces; en cada momento, abrimos el archivo, y al final, comprobamos si el mismo archivo está abierto comprobando sus `i-nodes` (deben ser iguales).

Utilice esta estrategia para modificar el programa vulnerable y repita su ataque. Documente lo difícil que es tener éxito, si aún puede tener éxito.

Tarea 3: Mecanismo de Protección B - Principio de Privilegio Mínimo

El problema fundamental del programa vulnerable, es violentar del *Principio de Privilegio Mínimo*. El programador entiende que el usuario que ejecuta el programa puede ser demasiado poderoso, por lo que introdujo `access()` para limitar el poder del usuario. Sin embargo, este no es el enfoque adecuado. Un mejor enfoque es aplicar el *Principio del Privilegio Mínimo*; es decir, si los usuarios no necesitan ciertos privilegios, el privilegio debe deshabilitarse.

Podemos usar la llamada al sistema `seteuid` para deshabilitar temporalmente el privilegio de root y luego habilitarlo si es necesario. Utilice este enfoque para corregir la vulnerabilidad en el programa y luego repita su ataque. ¿Se puede tener éxito en el ataque? Por favor documente sus observaciones y explicación.

Aspectos a considerar

Dos objetivos potenciales

Posiblemente hay muchas formas de explotar la vulnerabilidad de la condición de carrera en `vulp.c`. Una forma es usar la vulnerabilidad para agregar alguna información tanto a `/etc/passwd` como a `/etc/shadow`. Estos dos archivos son utilizados por los sistemas operativos `Unix` para autenticar a los usuarios. Si los atacantes pueden agregar información a estos dos archivos, esencialmente tienen el poder de crear nuevos usuarios, incluidos los superusuarios (cambiando `uid` a **cero**).

El archivo `/etc/passwd` es la base de datos de autenticación para una máquina Unix. Contiene atributos básicos de usuario. Este es un archivo ASCII que contiene una entrada para cada usuario. Cada entrada define los atributos básicos aplicados a un usuario. Cuando se usa el comando `useradd` para agregar un usuario a su sistema, el comando actualiza el archivo `/etc/passwd`.

El archivo `/etc/passwd` tiene que ser legible por todo el mundo, porque muchos programas de aplicación necesitan acceder a los atributos de los usuarios, como nombres de usuario, directorios de inicio, etc. Guardar una contraseña cifrada en ese archivo significaría que cualquier persona con acceso a la máquina podría usar programas para descifrar contraseñas (como `crack`) para irrumpir en las cuentas de otros. Para solucionar este problema, se creó el sistema de contraseña oculta. El archivo `/etc/passwd` en el sistema es legible por todo el mundo pero no contiene las contraseñas en hash. Otro archivo, `/etc/shadow`, que solo es legible por `root` contiene las contraseñas.

Para averiguar qué *Strings* agregar a estos dos archivos, ejecute `useradd` y vea qué se agrega a estos archivos. Por ejemplo, lo siguiente es lo que se ha agregado a estos archivos después de crear un nuevo usuario llamado `smith`:

```
/etc/passwd:
-------------
smith:x:1000:1000:Joe Smith,,,:/home/smith:/bin/bash


/etc/shadow:
-------------
smith:*1*Srdssdsdi*M4sdabPasdsdsdasdsdasdY/:13450:0:99999:7:::
```

La tercera columna del archivo `/etc/passwd` indica el `UID` del usuario. Debido a que la cuenta `smith` es una cuenta de usuario regular, su valor 1000 no es nada especial. Si cambiamos esta entrada a 0, `smith` ahora se convierte en `root`.

Creación de enlaces simbólicos

Puede crear manualmente enlaces simbólicos usando "`ln -s`". También puede llamar a la función C `symlink` para crear enlaces simbólicos en su programa. Dado que Linux no permite crear un enlace si el enlace ya existe, primero debemos eliminar el enlace anterior. El siguiente fragmento de código C muestra cómo eliminar un enlace y luego hacer que `/tmp/XYZ` apunte a `/etc/passwd`:

```
unlink("/tmp/XYZ");
symlink("/etc/passwd","/tmp/XYZ");
```

Mejora de la tasa de éxito

El paso más crítico (por ejemplo, apuntar el `symlink` a nuestro archivo de destino) de un ataque de condición de carrera debe ocurrir dentro de la ventana entre la verificación y el uso; es decir, entre `access` y las llamadas `fopen` en `vulp.c`. Como no podemos modificar el programa vulnerable, lo único que podemos hacer es ejecutar nuestro programa atacante en paralelo con el programa de destino, con la esperanza de que el cambio del enlace ocurra dentro de esa ventana crítica. Desafortunadamente, no podemos lograr el momento perfecto. Por lo tanto, el éxito del ataque es probabilístico. La probabilidad de un ataque exitoso puede ser bastante baja si la ventana es pequeña. Debe pensar en cómo aumentar la probabilidad (Sugerencias: puede ejecutar el programa vulnerable muchas veces; solo necesita lograr el éxito una vez entre todas estas pruebas).

Dado que necesita ejecutar los ataques y el programa vulnerable muchas veces, debe escribir un programa para automatizar el proceso de ataque. Para evitar escribir manualmente una entrada en `vulp`, puede usar la redirección. Es decir, escribe su entrada en un archivo y luego redirige este archivo cuando ejecuta `vulp`. Por ejemplo, puede usar lo siguiente: `vulp < ARCHIVO`.

En el programa `vulp.c`, se agregó intencionalmente un parámetro **DELAY** en el programa. Esto está destinado para hacer el ataque más fácil. Una vez que haya tenido éxito en el ataque, reduzca gradualmente el valor de **DELAY**. Cuando **DELAY** es cero, ¿cuánto tiempo más lleva tener éxito?

Saber si el ataque es exitoso

Dado que el usuario no tiene permiso de lectura para acceder a `/etc/shadow`, no hay forma de saber si se modificó. La única forma posible es ver sus marcas de tiempo. También sería mejor si detenemos el ataque una vez que las entradas se agreguen a los archivos respectivos. El siguiente script bash comprueba si se han cambiado las marcas de tiempo de `/etc/shadow`. Imprime un mensaje una vez que se nota el cambio.

```
#!/bin/bash

old=‘ls -l /etc/shadow‘
new=‘ls -l /etc/shadow‘

while [ "$old" = "$new" ]
do
    new=‘ls -l /etc/shadow‘
done

echo "El archivo shadow ha cambiado"
```

Solución de problemas

Mientras se prueba el programa, debido a la eliminación prematura del programa de ataque, /tmp/XYZ puede entrar en un estado inestable. Cuando esto sucede, el sistema operativo lo convierte automáticamente en un archivo normal con `root` como su propietario. Si esto sucede, el archivo debe eliminarse y el ataque debe reiniciarse.

Precauciones

Puede que accidentalmente el archivo `/etc/shadow` quede vacío durante los ataques. Si pierde el archivo shadow, no podrá volver a iniciar sesión. Para evitar este problema, haga una copia del archivo `shadow` original.

Aspectos Administrativos

Entregables

- Código fuente del programa que cumpla los requerimientos funcionales y técnicos.
- Binario del programa, compilado para una arquitectura x86.
- Fuente de la documentación en Latex o Markdown.
- PDF con la documentación.
- Video con la ejecución del ataque de forma satisfactoria.

Evaluación

- Tarea 1: 25 %
- Tarea 2: 25 %
- Tarea 3: 25 %
- Documentación del Ataque: 25 %

Documentación

Las siguientes son las instrucciones para la documentación:

1. **Introducción**: Presentar el problema.
2. **Instrucciones para ejecutar el programa**: Presentar las consultas concretas usadas para correr el programa para el problema planteado en el enunciado de la tarea y para los casos planteados al final de esta documentación.
3. **Descripción del Ataque**: En esta sección se debe describir el ataque de Evil Maid en su profundidad y incluyendo referencias hacia variantes del mismo.
4. **Documentación del Ataque**: Este es un resumen de como funciona el ataque Evil Maid ejecutado por el estudiante.
5. **Autoevaluación**: Indicar el estado final en que quedó el programa, problemas encontrados y limitaciones adicionales. Por otro lado, también debe incluir una calificación con la rúbrica de la sección "Evaluación".
6. **Lecciones Aprendidas**: Orientados a un estudiante que curse el presente curso en un futuro.
7. **Video**: Enlace al video de demostración del ataque.
8. **Bibliografía** utilizada en la elaboración de la presente asignación.
9. Es necesario documentar el código fuente.

Aspectos Adicionales

Aún cuando el código y la documentación tienen sus notas por separado, se aplican las siguientes restricciones:

1. Sí no se entrega documentación, automáticamente se obtiene una nota de 0.
2. Sí el código no compila se obtendrá una nota de 0, por lo cuál se recomienda realizar la defensa con un código funcional.
3. El código debe ser desarrollado en el lenguaje definido previamente, en caso contrario se obtendrá una nota de 0.
4. Sí no se se firma el archivo de la entrega se obtendrá una nota de 0.
5. La revisión de la documentación será realizada por parte del profesor, no durante la defensa del proyecto.
6. Cada excepción o error que salga durante la ejecución del proyecto y que se considere debió haber sido contemplada durante el desarrollo del proyecto, se castigará con 2 puntos de la nota final de la presente asignación.
7. Durante la revisión podrán participar asistentes, otros profesores y el coordinador del área.
8. Cualquier indicio de copia será calificado con una nota de 0 y será procesado de acuerdo al reglamento.

Licencia