Sistemas Operativos

Práctica 5 - Seguridad - Parte 1

Notas:

1. Utilizar un kernel completo (no el compilado en las prácticas 1 y 2)
2. Compilar el código C usando el Makefile provisto a fin de deshabilitar algunas medidas de seguridad del compilador y generar un código assembler más simple.
3. Acceda al código necesario para la práctica en el repositorio de la materia.

A - Introducción

1. Defina política y mecanismo.
2. Defina objeto, dominio y right.
3. Defina POLA (Principle of least authority).
4. ¿Qué valores definen el dominio en UNIX?
5. ¿Qué es ASLR (Address Space Layout Randomization)? ¿Linux provee ASLR para los procesos de usuario? ¿Y para el kernel?
6. ¿Cómo se activa/desactiva ASRL para todos los procesos de usuario en Linux?
7.

B - Ejercicio introductorio: Buffer Overflow simple

El propósito de este ejercicio es que las y los estudiantes tengan una introducción simple a un stack buffer overflow a fin de poder abordar el siguiente ejercicio. Las y los estudiantes aprenderán a identificar la vulnerabilidad, analizar la disposición de la memoria y construir una entrada que aproveche la vulnerabilidad para obtener acceso no autorizado a una función privilegiada.

Nota: Puede ser de ayuda ver el código assembler generado al compilar (00-stack-overflow.s) o utilizar gdb para depurar el programa pero no es obligatorio.

1. Compilar usando el makefile provisto el ejemplo 00-stack-overflow.c provisto en el repositorio de la cátedra.
2. Ejecutar el programa y observar las direcciones de las variables `access` y `password`, así como la distancia entre ellas.
3. Probar el programa con una password cualquiera y con “big secret” para verificar que funciona correctamente.

4. Volver a ejecutar pero ingresar una password lo suficientemente larga para sobreescribir `access`. Usar distance como referencia para establecer la longitud de la password.
5. Después de realizar la explotación, reflexiona sobre las siguientes preguntas:
    a. ¿Por qué el uso de gets() es peligroso?
    b. ¿Cómo se puede prevenir este tipo de vulnerabilidad?
    c. ¿Qué medidas de seguridad ofrecen los compiladores modernos para evitar estas vulnerabilidades?

C - Ejercicio: Buffer Overflow reemplazando dirección de retorno

Objetivo: El objetivo de este ejercicio es que las y los estudiantes comprendan cómo una vulnerabilidad de desbordamiento de búfer puede ser explotada para alterar la dirección de retorno de una función, redirigiendo la ejecución del programa a una función privilegiada. Además, se explorará el mecanismo de seguridad ASLR y cómo desactivarlo temporalmente para facilitar la explotación.

Nota: Puede ser de ayuda ver el código assembler generado al compilar (`01-stack-overflow-ret.s`) o utilizar gdb para depurar el programa pero no es obligatorio.

1. Compilar usando el makefile provisto el ejemplo 01-stack-overflow-ret.c provisto en el repositorio de la cátedra.
2. Configurar setuid en el programa para que al ejecutarlo, se ejecute como usuario root.
3. Verificar si tiene ASLR activado en el sistema. Si no está, actívelo.
4. Ejecute 01-stack-overflow-ret al menos 2 veces para verificar que la dirección de memoria de `privileged_fn()` cambia.
5. Apague ASLR y repita el punto 3 para verificar que esta vez el proceso siempre retorna la misma dirección de memoria para `privileged_fn()`.
6. Suponiendo que el compilador no agregó ningún padding en el stack tenemos los siguientes datos:
    a. El stack crece hacia abajo.
    b. Si estamos compilando en x86_64 los punteros ocupan 8 bytes.
    c. x86_64 es little endian.
    d. Primero se apiló la dirección de retorno (una dirección dentro de la función `main()`). Ocupa 8 bytes.
    e. Luego se apiló la vieja base de la pila (`rbp`). Ocupa 8 bytes.
    f. `password` ocupa 16 bytes.

    Calcule cuántos bytes de relleno necesita para pisar la dirección de retorno.

7. Ejecute el script payload_pointer.py para generar el payload. La ayuda se puede ver con: `python payload_pointer.py --help`
8. Pruebe el payload redirigiendo la salida del script a `01-stack-overflow-ret` usando un pipe.
9. Para poder interactuar con el shell invoque el programa usando el argumento `--program` del script `payload_pointer`.

   Por ejemplo:
   ```
   python payload_pointer.py --padding <padding> --pointer
   <pointer> --program ./01-stack-overflow-ret
   ```

10. Pruebe algunos comandos para verificar que realmente tiene acceso a un shell con UID 0.
11. Conteste:
    a. ¿Qué efecto tiene setear el bit setuid en un programa si el propietario del archivo es `root`? ¿Qué efecto tiene si el usuario es por ejemplo `nobody`?
    b. Compare el resultado del siguiente comando con la dirección de memoria de `privileged_fn()`. ¿Qué puede notar respecto a los octetos? ¿A qué se debe esto?

```
python payload_pointer.py --padding <padding> --pointer <pointer> | hd
```

    c. ¿Cómo ASLR ayuda a evitar este tipo de ataques en un escenario real donde el programa no imprime en pantalla el puntero de la función objetivo?
    d. ¿Cómo podría evitar este tipo de ataques en un módulo del kernel de Linux? ¿Qué mecanismo debería estar habilitado?

C - Ejercicio SystemD

Objetivo: Aprender algunas restricciones de seguridad que se pueden aplicar a un servicio en SystemD.

- https://www.redhat.com/en/blog/cgroups-part-four
- https://www.redhat.com/en/blog/mastering-systemd

1. Investigue los comandos:
   a. systemctl enable
   b. systemctl disable
   c. systemctl daemon-reload
   d. systemctl start
   e. systemctl stop
   f. systemctl status

   g. systemd-cgls
   h. journalctl -u [unit]
2. Investigue las siguientes opciones que se pueden configurar en una unit service de systemd:
   a. IPAddressDeny e IPAddressAllow
   b. User y Group
   c. ProtectHome
   d. PrivateTmp
   e. ProtectProc
   f. MemoryAccounting, MemoryHigh y MemoryMax
3. Tenga en cuenta para los siguientes puntos:
   a. La configuración del servicio se instala en:
      `/etc/systemd/system/insecure_service.service`
   b. Cada vez que modifique la configuración será necesario recargar el demonio de systemd y recargar el servicio:
      i. systemctl daemon-reload
      ii. systemctl restart insecure_service.service
4. En el directorio `insecure_service` del repositorio de la cátedra encontrará, el binario `insecure_service`, el archivo de configuración `insecure_service.service` y el script `install.sh`.
   a. Instale el servicio usando el script install.sh.
   b. Verifique que el servicio se está ejecutando con systemctl status.
   c. Verifique con qué UID se ejecuta el servicio usando psaux | grep insecure_service.
   d. Abra localhost:8080 en el navegador y explore los links provistos por este servicio.
5. Configure el servicio para que se ejecute con usuario y grupo no privilegiados (en Debian y derivados se llaman nouser y nogroup). Verifique con qué UID se ejecuta el servicio usando psaux | grep insecure_service.
6. Limite las IPs que pueden acceder al servicio para denegar todo por defecto y permitir solo conexiones de localhost (127.0.0.0/8).
7. Explore el directorio /home y el directorio /tmp usando el servicio y luego:
   a. Reconfigurelo para que no pueda visualizar el contenido de /home y tenga su propio /tmp privado.
   b. Recargue el servicio y verifique que estas restricciones surgieron efecto.
8. Limite el acceso a información de otros procesos por parte del servicio.
9. Establezca un límite de 16M al uso de memoria del servicio e intente alocar más de esa memoria en la sección “Memoria” usando el link “Aumentar Reserva de Memoria”