Universidad Nacional Autónoma de México

Facultad de Ingeniería

Exploiting Concurrency Vulnerabilities in System Call Wrappers

Alejandro Jesus Antonio Roblero

1. Introducción.

La interposición de llamadas al sistema es una técnica poderosa para regular y monitorear los comportamientos del programa. Brinda a los sistemas de seguridad la capacidad de monitorear toda la interacción del programa con la red, sistema de archivos y otros recursos sensibles del sistema. Muchos sistemas de seguridad como software antivirus comercial aprovechan la interposición de llamas del sistema para detectar comportamientos anómalos del programa. Para lograr identificar un comportamiento normal o anormal se basan en las llamas al sistema que normalmente invoca un programa en ejecución. Con el fin de poder garantizar la efectividad y seguridad, las llamadas al sistema deben ser interceptadas y manejadas de manera segura y completa.

2. Concurrencia y kernel.

Los núcleos del sistema operativo son programas altamente concurrentes que consumen servicios de concurrencia internamente y los ofrecen a los programas, la mayoría de los sistemas de escritorio y servidos suelen soportar multiprocesamiento. La concurrencia significa que múltiples procesos se ejecutan simultáneamente y en paralelo, y el acceso de las unidades de ejecución concurrente a recursos compartidos (recursos de hardware y variables globales y variables estáticas en el software) puede conducir fácilmente a condiciones de carrera.

La gestión de concurrencia es, sin embargo, uno de los problemas centrales en la programación de sistemas operativos. Los errores relacionados con la concurrencia son algunos de los más fáciles de crear y algunos de los más difíciles de encontrar. En un mundo donde se premia el rendimiento en sistemas con cada vez más procesadores y que insiste en que el sistema responda a los eventos de forma rápida, el kernel ha evolucionado hasta un punto en el que suceden muchas cosas de manera simultánea. Esta evolución ha dado como resultado un rendimiento mucho mayor pero también ha complicado de manera significativa la tarea de la programación del kernel.

3. Wrappers y seguridad.

3.1 Wrapper.

Los envoltorios controlan la interacción de la función original con el entorno. El contenedor después de interceptar la llamada puede modificar los argumentos, devolver el valor de la función original o no ejecutar la función en absoluto. El envoltorio, desde la vista de patrón de diseño, se le puede considerar un proxy o un decorador.

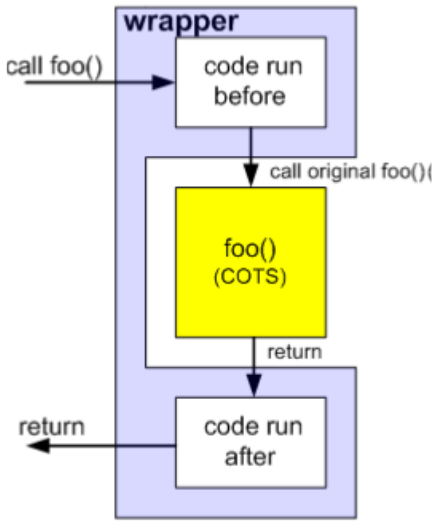
3.2 Interposición a llamadas del sistema con wrappers.

La interposición de llamadas al sistema tiene un monitor de referencia en los servicios del kernel interceptando las llamadas al sistema que permite supervisar y regular prácticamente todas las interacciones de una aplicación con la red, sistema de archivos y otros recursos sensibles del sistema. Los contenedores de llamadas al sistema pueden

cumplir con el rol del monitor ya que se ejecutan en el dominio del kernel, se invocan en la ruta de la llamada del sistema y evitan modificar el kernel de forma compleja.

Los envoltorios de llamadas al sistema implementan el procesamiento de condiciones previas y posteriores alrededor de la llamada al sistema.

- Condición previa: Ocurre antes de pasar el control al kernel, lo que permite que el contenedor inspeccione o sustituya los argumentos antes de pasarlo al kernel.
- Condición posterior: Se lleva a cabo antes de devolver el control al espacio de usuario, lo que ayuda a que el contenedor rastree y registre los resultados, transforma los valores devueltos, etc.



4. Vulnerabilidades de la concurrencia.

4.1 Condición de carrera.

En los sistemas modernos, existen numerosas fuentes de concurrencia y, por lo tanto, posibles condiciones de carrera. Las condiciones de carrera surgen como resultado del acceso compartido a los recursos. Cuando dos subprocesos de ejecución tienen una razón para trabajar con los mismos recursos, siempre existe la posibilidad de que se produzcan confusiones. Entonces, evitar las condiciones de carrera puede ser una tarea intimidante. Resulta que la mayoría de las condiciones de carrera se pueden evitar con un poco de pensamiento, las primitivas de control de concurrencia del kernel, etc.

4.2 Vulnerabilidades debido a la concurrencia.

Impulsados por el auge del hardware de múltiples núcleos, los programas de múltiples subprocesos ya son omnipresentes. Por desgracia, estos programas están plagados de errores de concurrencia como:

- La carrera de datos, que consiste en como dos subprocesos acceden al mismo byte de memoria al mismo tiempo, donde uno de los accesos es de escritura.
- Perdida de información, ya que un hilo sobrescribe la información.
- Problemas de rendimiento, debido al mal manejo de los recursos.

Las vulnerabilidades pueden originarse por no planificar adecuadamente la concurrencia, errores de implementación en el control de la concurrencia, etc.

5. Ataques de concurrencia en wrappers

Los núcleos del sistema operativo al proteger datos críticos son un punto de mira para encontrar vulnerabilidades, el enfoque dado por el artículo [1], es la no atomicidad entre el kernel y las envolturas de llamadas al sistema.

Se descubrió que las vulnerabilidades identificables y explotables con mayor frecuencia se clasifican en tres categorías:

- Time-of-check-to-time-of-use (TOCTTOU): vulnerabilidades, en las que las comprobaciones de control de acceso no son atómicas con las operaciones que protegen, lo que permite a un atacante violar las reglas de control de acceso.
- Time-of-audit-to-time-of-use (TOATTOU): vulnerabilidades, para ocultar los datos de cualquier registro que se pueda realizar, cubriendo las pistas de un exploit de una aplicación de detección de intrusos.
- Time-of-replacement-to-time-of-use (TORTTOU): vulnerabilidades, exclusivas de los envoltorios, en las que los atacantes modifican los argumentos de las llamadas al sistema después de que un envoltorio los ha reemplazado, pero antes de que el kernel haya accedido a ellos.

6. Time-of-check-time-of-use (TOCTTOU).

6.1 Descripción del problema.

La falla ocurre cuando un programa busca una característica particular de un objeto, y luego toma alguna acción que asume que la característica aún se mantiene cuando en realidad no es así. Una subclase de defectos de TOCTTOU surge cuando se asume erróneamente que los identificadores de objeto permanecen ligados a un objeto.

6.2 Demostración del problema.

Muchos sistemas operativos permiten que algunos usuarios de confianza tengan un control completo sobre el sistema ya que alivia los problemas de administración. Pocos ataques explotan fallas específicas del kernel del sistema operativo; la mayoría explota fallas en las utilidades.

La falla de enlace arquetípica de TOCTTOU en un programa privilegiado en el sistema operativo UNIX surge cuando un programa setuid to root requiere guardar datos en un archivo propiedad del usuario que ejecuta el programa. El programa no debe alterar el archivo a menos que el usuario pueda alterar el archivo sin privilegios especiales. Ejemplo genérico de código:

```
if (access(filename, W_OK) == 0)
{
    if ((fd = open(filename, O_WRONLY)) == NULL)
    {
        perror(filename);
        return(0);
    }
    /* now write to the file */
```

Si el objeto al que hace referencia filename cambia entre las dos llamadas al sistema el segundo objeto se abrirá, aunque su acceso nunca fue verificado.

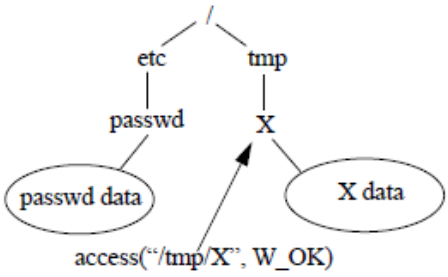


Figure 1a.

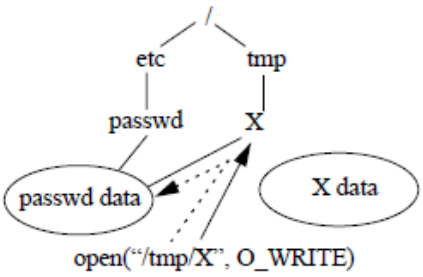


Figure 1b.

En la primera figura 1a se muestra un ejemplo de falla vinculante de TOCTTOU, donde el estado del sistema en el momento del acceso a la llamada del sistema, la flecha sólida indica el acceso a "/tmp/X". Tanto "/tmp/X" como "/etc/passwd" nombran objetos distintos. No obstante, antes de que el proceso realice su llamada al sistema (*open*), se elimina "/tmp/X" y se crea un enlace fijo para "/etc/passwd", que se denomina "/tmp/X". Entonces *open* accede a los datos asociados con "/etc/passwd" cuando abre "/tmp/X", ya que "/tmp/X" y "/etc/passwd" apuntan al mismo archivo. La figura 1b indica con la flecha punteada los datos que se leen realmente mientras la flecha solida muestra que nombre se le paso a open. Como resultado, el proceso sin privilegios puede escribir en el archivo de contraseña protegido. Varias versiones del programa de emulación de terminal (xterm) sufren de esta falla.

6.3 Caracterización de TOCTTOU

Una falla TOCTTOU ocurre en dos eventos donde el segundo depende del primero. Durante el intervalo entre dos eventos, ciertas suposiciones de los resultados de la primera llamada al sistema influyen en el segundo. Si alguna acción durante ese intervalo invalida esos supuestos, es posible que los resultados de la segunda acción no sean los previstos, para esto se requiere que el atacante actúe durante el intervalo. El término general es "condición de carrera" donde los atacantes intentan invalidar las suposiciones antes de que ocurra la segunda acción.

La existencia del intervalo se le llama *condición de programación* y el intervalo mismo se nombra *intervalo de programación*. Habiendo encontrado que esta condición se cumple, el atacante debe poder afectar las suposiciones creadas por la primera acción del programa. Esa condición es la *condición de entorno.* Ambas condiciones deben cumplirse para que exista una falla TOCTTOU explotable.

Retomando el ejemplo anterior, la llamada del sistema *access* crea la suposición de que el usuario está autorizado a modificar el archivo "/tmp/X". De tal manera que *open* actúa sobre esa suposición. Entonces la condición de programación se mantiene. Si el atacante puede alterar la referencia del nombre "/tmp/X", entonces la condición de entorno también

se cumple y existe una falla TOCTTOU explotable. Si el archivo estuviera en un directorio que el atacante no pudiera alterar, la *condición de entorno* no se cumpliría, por lo tanto, no existe falla TOCTTOU.

7. Técnicas de mitigación.

- Lee Badger [2] ha sugerido un enfoque de detectar y mitigar los cambios de explotación en el estado del kernel a través de una condición posterior, tomando medidas correctivas si se ha producido una infracción. El enfoque enfrenta desafíos debido a los complejos efectos secundarios de algunas llamas al sistema, la detección de inconsistencias enfrenta los mismos problemas de atomicidad que otras postcondiciones.
- Provos [3] proporciona facilidades similares en Systrace, copiando argumentos indirectos en el "espacio de pila", un área reservada de la memoria del proceso, lo que permite que los contenedores sustituyan los argumentos indirectos de mayor tamaño que el argumento original. Por otra parte, se ha propuesto que se puede utilizar para resistir los ataques de memoria compartida, ya que el espacio de pila es único para cada proceso. Esta protección no es efectiva con subprocesos, ya que los subprocesos comparten un solo espacio de direcciones. Este enfoque genera copias de datos adicionales para cualquier argumento protegido.
- Ghormley [4] aborda las carreras de argumentos en el marco de interposición de SLIC a través de búferes en el núcleo que extienden el espacio de direcciones. Cada subproceso almacena en caché regiones del espacio de direcciones copiadas por la extensión o kernel, los accesos futuros serán desde el cache, lo que evitará modificaciones adicionales por parte de los hilos del usuario. Sin embargo, este enfoque impone una importante penalización de rendimiento, ya que todos los argumentos indirectos deben copiarse y almacenarse en caché en la memoria del kernel.

8. Conclusión

Se ha presentado el tema de concurrencia, además de su relación con el kernel explicando de manera breve como el avance de la tecnología (en especial el procesador multinúcleo) ha influido fuertemente en ambos conceptos. Por otra parte, se abordó el tema de los wrappers explicando de manera introductoria su funcionamiento, igualmente se explicó uno de sus usos como monitor de los servicios del kernel. Posteriormente se relacionaron las vulnerabilidades de la concurrencia con los contenedores de las llamadas al sistema, igualmente se mencionaron y describieron brevemente las clasificaciones de las vulnerabilidades que son explotadas con mayor frecuencia; profundizando en mayor medida con la vulnerabilidad Time-of-check-time-of-use (TOCTTOU) con el fin de describir y mostrar la manera en la que se pueden usar los errores de envoltura y concurrencia para comprometer la seguridad de un sistema. Continuando, se describieron algunas técnicas de

mitigación que se pueden usar para evitar en la medida de lo posible las vulnerabilidades, describiendo los pros y contras de cada una. Finalmente, este trabajo no es más que un acercamiento al tema, por esa razón se llegan abordar temas que a primera vista no tienen relación pero que buscan poner al lector en contexto para una mejor comprensión por lo que se recomienda leer la bibliografía usada si se desea realizar un estudio más a fondo.

9. Referencias.

[1] Watson, R. N. (2007). Exploiting Concurrency Vulnerabilities in System Call Wrappers. WOOT.

[2] Fraser, T., Badger, L., & Feldman, M. Hardening COTS software with generic software wrappers. IEEE.

[3] Provos, N. (2003, August). Improving Host Security with System Call Policies. In USENIX Security Symposium.

[4] Ghormley, D. P., Rodrigues, S. H., Petrou, D., & Anderson, T. E. (1998, June). SLIC: An Extensibility System for Commodity Operating Systems. In USENIX Annual Technical Conference (Vol. 98).

[5] Bishop, M., & Dilger, M. (1996). Checking for race conditions in file accesses. Computing systems.

[6] Zhao, S., Gu, R., Qiu, H., Li, T. O., Wang, Y., Cui, H., & Yang, J. (2018, June). Owl: Understanding and detecting concurrency attacks. In 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. IEEE.

[7] Snyder, D. (2001). On-line intrusion detection using sequences of system calls.

[8] Garfinkel, T. (2003). Traps and pitfalls: Practical problems in system call interposition based security tools. In In Proc. Network and Distributed Systems Security Symposium.

[9] Edge, J. (2007). Exploiting races in system call wrappers [LWN.net]. Consultado el 19 de Julio de 2021, en https://lwn.net/Articles/245630/.