Bloqueos mutuos y estados de carrera: Las vulnerabilidades más explotables generadas por hilos y Procesos en los sistemas informáticos

Días atrás, inicié un post sobre este tema en blog de DebianHackers y dado que hay mucho que investigar sobre la implementación de Hilos y Procesos en la ingeniería de sistemas y sus consecuentes riesgos de seguridad, decidí agregarlo en esta edición, para así poder dar inicio a una exhaustiva investigación al respecto.

No conozco programador **Python** que no alucine trabajando con **«threads»** y **«process»** y a pesar de ello, muy pocos son los que realmente saben de qué se trata. Pero esto no es lo peor. Lo peor, es que desconocen los **riesgos de seguridad que una mala implementación de hilos y procesos puede generar en todo un sistema informático**.

Si ya trabajas con **«threads»** y **«process»** este artículo te aclarará algunas dudas con respecto a la seguridad del sistema. Pero si aún no haz llegado hasta allí, **es una buena forma de comenzar, puesto que te lo explicaré de forma que lo entiendas, sin rollos ni tecnicismos inútiles**.

El principio ¿Qué es un proceso?

Primero que nada, vamos a tratar de entender qué es un proceso. Un proceso, no es más que una parte de un programa que se está ejecutando. Técnicamente es un **conjunto de instrucciones haciendo uso de un conjunto de recursos del sistema**.

> *Cuando hablamos de **recursos del sistema** nos referimos a uso de memoria RAM, Swap, CPU, tiempo de ejecución, entre otros recursos menos relevantes*

Por este motivo, cuando se dice que un programa hace un mal uso de los recursos del sistema, por lo general es debido a la ejecución desorganizada que hace de sus propios procesos.

¿Qué es un hilo?

Un hilo (*thread* en inglés) es similar a un proceso en cuanto a que también representa un conjunto de instrucciones. Sin embargo, los hilos se diferencian enormemente de los procesos, paradójicamente, por la administración que hacen de los recursos del sistema. Y esta última, es la característica que diferencia a los hilos de los procesos, tal como te lo explico en el siguiente párrafo.

Threads vs. Procesos: ¿cuál es la diferencia entre hilos y procesos?

La diferencia fundamental, es que ambos administran los recursos de formas diferentes. **Mientras que los diferentes procesos de un mismo programa ocupan diferentes espacios en la memoria, diferentes hilos comparten un mismo espacio**.

A primera vista, esto nos puede hacer suponer que los threads son una mejor alternativa que los procesos, puesto que utilizarán menor espacio de memoria. Esto es muy cierto, pero no son la panacea. Pueden acarrear múltiples problemas de seguridad tal como explicaré más adelante. No obstante, el uso de procesos en vez de threads, también solucionará problemas -a nivel seguridad de la aplicación- que estos últimos no tienen capacidad de resolver. Pero vayamos de a poco.

Problemas de seguridad generados por los threads

Todo conjunto de hilos o de procesos, probablemente necesite compartir información entre ellos. Sin lugar a dudas, será mucho más fácil compartir información si se comparte un mismo espacio de memoria, que hacerlo si hay que estar saltando de un puntero hacia otro. Por consiguiente, **compartir información será mucho más fácil para los threads que para los procesos**.

Sin embargo, compartiendo grandes cantidades de información y haciendo un uso significativo de tareas concurrentes a través de threads, **se pueden producir dos vulnerabilidades trágicas**: Una de ellas, la llamada **«bloqueo mutuo»** (*deadblock* en inglés); la otra y a la vez más peligrosa, la denominada **«condición de carrera»** (*race condition* en inglés).

> ***Bloqueo mutuo (deadlock):** Es el **bloqueo irreversible de un conjunto de hilos o procesos**.*

Un bloqueo mutuo es lo que sucede cuando un programa se te queda “colgado” y te ves en la obligación de “matar un proceso” ya que el conjunto de hilos o procesos bloqueados, no tiene solución.

> ***Condición (o estado) de carrera:*** *Es aquel que se produce cuando* ***varios hilos o procesos intentan modificar de forma simultánea a un mismo recurso.***

Si más de un hilo o recurso intenta modificar el estado o valor de un mismo recurso al mismo tiempo, los datos dejan de ser confiables y por consiguiente, es correcto afirmar que **los datos quedan corruptos**.

Por ejemplo, si Juan y Ana intentan modificar la contraseña del usuario admin al mismo tiempo. Juan decide colocar 123456 y Ana, 223344 ¿cuál de las dos claves será finalmente, la clave del usuario admin? La respuesta es que si el programador no contempló está posibilidad, será imprevisible conocer la clave. Incluso, podría quedar corrupta a tal punto de quedar sin clave. Esta concurrencia mal implementada es la que se denomina condición de carrera, donde los actores "compiten" por ver quien "gana" la modificación del dato.

Los estados de carrera en un sistema informático **son fácilmente explotables mediante «*exploits*» locales**. Por ese motivo, representan uno de los riesgos de seguridad más graves en un sistema.

Linux y GNU vs Sistemas Operativos privativos

En más de una oportunidad habrás escuchado decir que Linux (el kernel) y los sistemas operativos GNU que lo implementan, hacen una mejor administración de recursos que los sistemas operativos privativos comúnmente usados por el usuario promedio y que no se encuentran basados en Unix. Esto simplemente hace referencia al costo que la creación y destrucción de procesos, implica en los distintos sistemas.

Tanto en Linux y Unix a nivel kernel como en GNU a nivel sistema operativo, el costo de creación y destrucción tanto de hilos como de procesos puede ser casi imperceptible, mientras que en sistemas operativos privativos con núcleos diferentes, **el costo es muy elevado** y por ese motivo, en estos sistemas, se utilizan más los threads que los procesos. De allí que comúnmente se escuche decir que estos sistemas son más vulnerables que los que implementan Linux o Unix.

Uso de procesos que evitan errores de diseño explotables

Un buen uso de procesos y de preferencia de estos frente a los hilos, se da cuando una aplicación requiere efectuar tareas simultáneas con usuarios diferentes. Puede que no te hayas planteado emplear diferentes usuarios para diferentes tareas de tu aplicación y esta es una excelente oportunidad. **Apache** nos da un ejemplo concreto de ello.

Apache utiliza múltiples procesos con diferentes usuarios para la ejecución de sus tareas. El enlace a los puerto de bajo número -como el puerto 80, por ejemplo- lo realiza a través el usuario root mientras que el procesamiento de las solicitudes Web, lo efectúa con el usuario www-data quien tiene permisos muy inferiores a los de root.

Casos como el de Apache, son los que evitan que frente a un error producido en el código de las tareas ejecutadas por usuarios con permisos escuetos, ejecuten acciones solo permitidas al usuario root. La

única forma de que esto suceda, sería escalando privilegios para lo cual, otro error sería necesario de forma simultánea pero en el código de las tareas asignadas al usuario root.

Trabajo distribuido y portabilidad

Otra de las grandes ventajas que los procesos facilitan frente a los hilos, es la de permitir una mayor portabilidad en las aplicaciones y posibilidad de compartir el trabajo de forma simple (característica por excelencia en las arquitecturas cliente-servidor).

Esto significa que si lo que se necesita es una aplicación que permita distribuir/compartir el trabajo a través de una red local o externa, la ejecución de tareas mediante diferentes procesos, será la única alternativa. Caso contrario, la aplicación podría responder de forma impredecible facilitando así la corrupción de los datos y su consecuente explotación.

Nota final sobre threads: hilos del kernel vs hilos de usuario

Es muy importante que como programador o programadora, entiendas que **diferentes hilos comparten un mismo PID** (*process identifier* o identificación de proceso) mientras que diferentes procesos, poseen sus propios PID. De allí que frecuentemente puedas escuchar frases como *"los diferentes hilos de un proceso"*.
Sin embargo, **esto no es así a nivel del kernel**. Los hilos del kernel tienen sus propios PID. Esto es debido a la forma en la que el Kernel es ejecutado.

El kernel en sí mismo no se ejecuta como un proceso sino que sus tareas corren como parte de otros procesos. Al ser una cantidad de tareas significativas, **el kernel se ve obligado a implementar acciones alternativas que operen de forma similar a los procesos**, a fin de abaratar costos. Estas acciones alternativas **son los famosos demonios** (*daemon* en inglés) los cuales se ejecutan constantemente en segundo plano, asegurando de esta forma, un menor uso de los recursos ya que en caso contrario, de no estar disponibles de forma constante al usuario, éste debería invocar a dichos servicios cada vez que necesitara hacer uso de ellos. Esto implicaría nuevos procesos creándose y destruyéndose de forma permanente.
Cuando hablamos de caro o barato refiriéndonos al uso de recursos del sistema, estamos diciendo que **"a mayor uso de RAM y mayor tiempo de ejecución, más cara es la implementación"**.

En entregas futuras, procuraré avanzar en cuestiones más específicas sobre la implementación de hilos y procesos en la Ingeniería de aplicaciones. Pero mientras tanto, **si programas en Python, desde ahora en más, antes de decidir utilizar *threads*, piensa en lo que acabas de leer.**