Syscall Top: Estrategias de monitoreo de llamadas al sistema en sistemas GNU/Linux

Fabián A. Gibellini, Sergio Quinteros, Germán N. Parisi, Milagros N. Zea Cárdenas, Leonardo Ciceri, Federico J. Bertola, Ileana M. Barrionuevo, Juliana Notreni, Analía L. Ruhl, Marcelo Auquer

Laboratorio de Sistemas / Dpto. de Ingeniería en Sistemas de Información/
Universidad Tecnológica Nacional / Facultad Regional Córdoba
Maestro M. Lopez esq. Cruz Roja Argentina S/N, Ciudad Universitaria (X5016ZAA) - Córdoba, Argentina
{fabiangibellini, ser.quinteros, germannparisi, milyzc, leonardorciceri, federicorbertola ilebarrionuevo, julinotreni, analialorenaruhl, marcelo.auquer}@gmail.com

Resumen. Los procesos que se ejecutan en un sistema GNU/Linux interactúan con el kernel por medio de llamadas al sistema, incluso los malwares. Existen distintas categorías de malware y en base a su comportamiento se puede inferir que existen patrones de llamadas al sistema, o syscalls, que permitirían descubrir qué tipo de malware se está ejecutando sobre un GNU/Linux. El presente trabajo pretende introducir formalmente la herramienta syscall top, la cual permite visualizar las llamadas al sistema interceptadas de todos los procesos y administrar reglas que permitan configurar acciones automáticas en contra de los procesos que no cumplan con dichas reglas. También se presentarán distintas estrategias reactivas frente a posibles ejecuciones de procesos sospechosos de ser ransomwares.

Palabras Claves: Seguridad. Syscalls. Kernel. Linux. Security. Malware. Ransomware.

1 Introducción

En los últimos años ha crecido la ciberdelincuencia y, con esta, la variedad de malwares o programas maliciosos. Según Raymond et al, el mayor desafío de crear un esquema completo de nombrado de malwares, se debe al número de muestras existentes de malware y a la frecuencia con la que nuevas muestras son descubiertas [1]. Si se considera la clasificación basada en comportamiento propuesta por C. Elisan [2], se puede distinguir entre ransomwares, keyloggers, spywares, gusanos, troyanos, etc. A su vez, un mismo malware puede comportarse como un virus cuando se propaga por un dispositivo de cómputo, como un gusano cuando se propaga a través de una red, mostrar comportamiento de botnet cuando se comunica con servidores de comando y control o cuando sincroniza con otras máquinas infectadas, y comportarse como un rootkit al ocultarse de un sistema de detección de intrusiones (IDS) [3].

Cada uno de estos malwares intentan generar algún daño y para lograrlo es normal que utilicen el kernel para acceder a los recursos que necesitan. Un ransomware, por ejemplo, es una forma de software malicioso utilizado en ataques, en los que no se

busca destruir irreversiblemente los datos, sino cifrar y cobrar por el servicio de recuperación de los datos cifrados [4] y para esto realiza operaciones de lectura y escritura sobre el disco, utilizando el kernel. Otro ejemplo es un keylogger, que es un software que se ubica entre el hardware y el sistema operativo e intercepta cada pulsación de tecla y la almacena, para lo cual también ejecuta estas operaciones por medio del kernel.

Una consultora de cibereconomía y ciberseguridad estima que los daños del cibercrimen tendrán un costo anual y global de seis mil millones de dólares en 2021 [5]. Estos costos incluyen daños y destrucción de datos, dinero robado, pérdida de productividad, robo de propiedad intelectual, robo de datos personales y financieros, malversación, fraude, interrupción posterior al ataque en el curso normal de los negocios, investigación forense, restauración y eliminación de datos perjudicados y sistemas, y daño a la reputación [6].
Es debido al impacto asociado a estos malwares que se hace necesaria una búsqueda permanente para encontrar nuevas técnicas de prevención, o actualizar continuamente las ya existentes, de forma tal que se minimice el impacto de estas amenazas. Entre estas técnicas podemos encontrar las técnicas de análisis de comportamiento o análisis dinámico.
Las técnicas de análisis de comportamiento utilizan las características del software en ejecución para identificar malware potencial [7]. Una de estas técnicas es el análisis de llamadas al sistema, en el que los comportamientos maliciosos se identifican mediante sus rastros de llamadas al sistema [8].

Las llamadas al sistema son muy utilizadas por los programas y los procesadores actuales optimizan su latencia de invocación. Por ejemplo, la llamada al sistema getpid() se completa en 61 nanosegundos [9].

Una llamada al sistema es solo una solicitud de espacio de usuario de un servicio del kernel. Cuando un programa quiere escribir o leer desde un archivo, comenzar a escuchar conexiones en un socket, eliminar o crear un directorio, o incluso terminar su trabajo, el programa usa llamadas al sistema (Figura 1). En otras palabras, una llamada al sistema es solo una función de espacio del núcleo que los programas de espacio del usuario llaman para manejar alguna solicitud. El kernel de Linux proporciona un conjunto de estas funciones y cada arquitectura proporciona su propio conjunto. Por ejemplo: el x86_64 proporciona 322 llamadas al sistema y el x86 proporciona 358 llamadas al sistema diferentes [10].

Es válido aclarar que la necesidad de acceder a los servicios que brinda el kernel a través de llamadas al sistema no es algo exclusivo de los malwares, sino que cualquier proceso lo realiza. La diferencia se encuentra en la manera de acceder, tanto hacia qué llamada, como la frecuencia y los parámetros con los que cada una es solicitada [11].

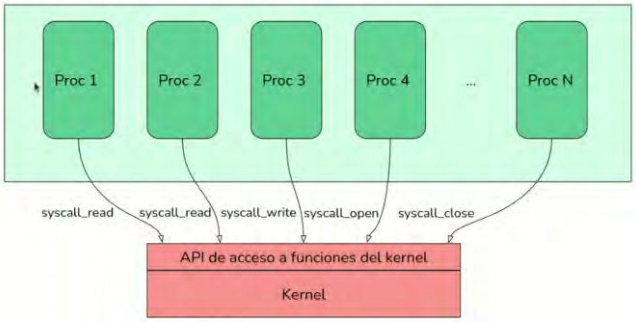

Figura 1. Representación gráfica de syscalls desde distintos procesos.

¿Por qué GNU/Linux? Linux es una opción popular para servidores [12] y sistemas integrados [13], por sus beneficios en rendimiento, fiabilidad y facilidad de desarrollo. Estos tipos de sistemas tienen un mayor riesgo de seguridad, ya que pueden ser una base de datos servidor, equipo de red o una unidad de control de un dispositivo crítico de seguridad.

El presente trabajo utiliza un módulo kernel ya presentado anteriormente como contador de llamadas al sistema en la edición anterior, el cual intercepta las syscalls y lleva un registro de cuántas llamadas al sistema se ejecutan por proceso permitiendo un monitoreo de las mismas [14]. A este contador de syscalls se lo denominó "Módulo kernel del Syscall Top" y para el cual se desarrolló una aplicación Syscall Top que es su Interfaz Gráfica de Usuario (GUI) que se ejecuta en el espacio de usuario. Además de permitir una mejor interpretación de la interacción de las llamadas al sistema de los programas con el kernel también permite configurar reglas con acciones definidas en el caso de que las syscalls superen ciertos umbrales definidos en dichas reglas.
Esta aplicación Syscall Top y sus posibles configuraciones como estrategias de monitoreo frente a procesos sospechosos de ser ransomwares son el eje central del presente trabajo.
Este trabajo está enmarcado dentro del proyecto de I+D "Sistema de detección de malware basado en patrones de llamadas al sistema en GNU/Linux.", código SIUTNCO0007850.

2 Desarrollo

Como se mencionó anteriormente la herramienta Syscall Top está compuesta por dos módulos: el módulo kernel que ya se presentó en la edición 2021 [14] y la aplicación Syscall Top que se presenta ahora.

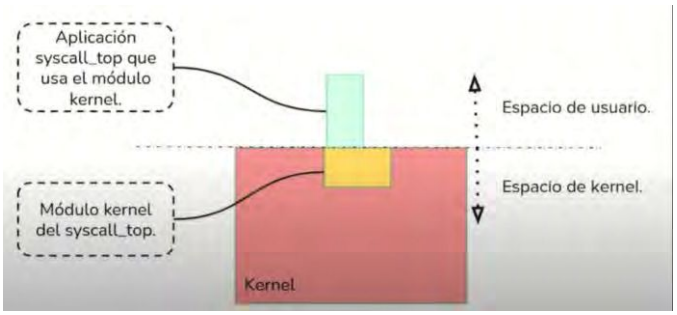

Figura 2. Arquitectura de Syscall Top. Módulo kernel y Aplicación.

El objetivo general de esta herramienta Syscall Top es monitorear ciertas llamadas al sistema revisando cada X tiempo, en base a su tasa de refresco, si dichas llamadas superan un umbral definido, y si lo superan, ejecutar una acción previamente definida. Cuando se inicia el Syscall Top se puede apreciar la siguiente salida (Figura 3) en la consola.

SyscallTop

PID	read	write	open	close	other	total	name
1	2	0	2	2	2	8	systemd(0)
1798	398	397	0	0	0	795	sshd(0)
391	326	0	176	176	204	882	systemd-journald(0)
1544	3	6	3	6	3	21	pickup(1)
1418	4	0	0	2	0	6	nscd(0)
1419	14	6	1	7	5	33	nscd(0)
1420	4	0	0	2	0	6	nscd(0)
1421	4	0	0	2	0	6	nscd(0)
1422	4	0	0	2	0	6	nscd(0)
845	1	0	0	1	0	2	systemd-logind(0)
848	0	0	81	27	378	486	dbus-daemon(4)
1593	23	0	0	0	0	23	slapd(0)
1624	19	12	2	10	10	53	nscd(0)
1625	13	12	0	2	0	27	slapd(0)
1626	12	11	0	3	0	26	slapd(0)
860	72	66	0	0	0	138	gmain(0)
1629	10	10	0	2	0	22	slapd(0)
1125	162	0	27	54	0	243	vmtnfo(0)
1009	3	0	0	0	0	3	in:imklog(0)
809	30	66	0	0	0	96	avahi-daemon(1)
1543	6	6	0	3	0	15	master(0)
1010	0	31	2	0	0	33	rs:main Q:Reg(1)
1000	266	209	62	149	189	875	bash(0)
818	0	0	0	0	21	21	cron(4)
1017	2	0	2	2	0	6	postgres(0)
1594	10	12	0	2	0	24	slapd(1)

Figura 3. Syscall Top en ejecución.

En la Figura 3 podemos visualizar la siguiente información:

- PID: Process Id del proceso en cuestión.
- read: Cantidad de llamadas al sistema de tipo read. Esta llamada al sistema lee bytes de un archivo referenciado por un file descriptor a un buffer.
- write: Cantidad de llamadas al sistema de tipo write. Esta llamada al sistema escribe bytes desde un buffer al archivo referenciado por el file descriptor.

- open: Cantidad de llamadas al sistema de tipo open. Esta llamada al sistema abre un archivo.
- close: Cantidad de llamadas al sistema de tipo close. Esta llamada al sistema cierra un file descriptor, por lo tanto, el archivo referenciado ya no puede ser accedido.
- other: Sumatoria de cantidad de llamadas al sistema de tipos stat, ptrace, fstat.
- stat y fstat: Estas llamadas al sistema obtienen información sobre un archivo, como por ejemplo tiempo.
- ptrace: La llamada al sistema permite observar y controlar la ejecución de un proceso.
- name: Nombre del proceso asociado al PID

Como se puede apreciar la interfaz es familiar a la herramienta Top [15] de un GNU/ Linux para lograr fácil apreciación de la información. Por otro lado, si se identificó un proceso y se quiere hacer énfasis en el mismo, puede hacérselo a través de la ejecución del siguiente comando (Tabla 1) y su salida será la de la Figura 4.

```
$ syscalltop ping
```

Tabla 1. Comando para ejecutar el syscall top monitoreando un solo proceso. Ejemplo con el proceso ping

Figura 4. Salida en consola del syscall top top monitoreando un solo proceso. Ejemplo con el proceso ping.

Una vez que el Syscall Top está ejecutándose, además de brindar la información de llamadas al sistema en tiempo cuasi real, también está monitoreando si alguno de los procesos no cumple con reglas definidas en su archivo reglas.ini, el cual debe tener la siguiente estructura (Figura 5).

Figura 5. Estructura de archivo rules.ini. Archivo que define las reglas de acción del Syscall Top.

El ejemplo de la Figura 5 determina que el syscall top debe detener el/los procesos que superen al mismo tiempo las 10000 llamadas al sistema de write y read. Es decir, si un proceso X superó las 10000 llamadas de write, como también de read en los últimos Z segundos este proceso será detenido.
Generalizando esta estructura el archivo de reglas quedaría:

```
# Estructura de rules.ini
[<SEÑAL>]
syscallM=<umbralM>
…
syscallN=<umbralN>
```

Donde:
- <SEÑAL>: Señales (Signals) que interpreta el kernel Linux [16]. Representan las acciones a tomar cuando un proceso supere los umbrales definidos.
- <syscallM o N>: Llamadas al sistema que se definen para la regla.
- <umbralM o N>: Umbral que representa el máximo permitido de interacciones entre la llamada al sistema M o N.

Si un proceso cumple con todas esas reglas, a dicho proceso se le aplicará la SEÑAL definida.
En cuanto a las llamadas al sistema que pueden ser definidas como reglas, por el momento, solo se pueden definir las de *read* y *write*, debido a que por el momento, esta herramiento está siendo probada con ransomwares y el patrón detectado para este malware son este tipo de llamadas.
En lo referido a las señales, puede configurarse cualquier señal que interprete el kernel de linux, las que se pueden listar a través de un kill -l [17]. Entre las más conocidas podemos nombrar la SIGKILL y SIGSTOP.
Todo lo mencionado anteriormente permite pensar y diseñar distintas estrategias de reacción ante posibles ataques de ransomwares. A continuación se listan algunas estrategias, siempre teniendo en cuenta la tasa de refresco del Syscall top de T segundos.:
1. Detención de procesos: manteniendo el ejemplo anterior, una primera estrategia sería la de detener los procesos que superen las 10000 interacciones de write y read en los últimos T segundos para una revisión posterior manual por parte de una persona que decidirá si el proceso continúa o no (Tabla 2).

```
# Estructura de rules.ini
[SIGSTOP]
read=10000
write=10000
```

Tabla 2. Configuración de rules.ini para una estrategia de detención.

En esta estrategia, el usuario debe revisar manualmente los *jobs* detenidos para identificar los procesos en cuestión.

2. Matar procesos: se puede tener como premisa que todo proceso que supere las 10000 de *write* y *read* en los últimos T segundos es considerado malicioso y, por lo tanto, directamente se envía la señal de kill a ese proceso (Tabla 3).

```
# Estructura de rules.ini
[SIGKILL]
read=10000
write=10000
```

Tabla 3. Configuración de rules.ini para una estrategia de matar procesos.

Esta estrategia tiene el propósito de asegurar que el proceso sospechoso no se iniciará nuevamente. Sin embargo, al ser tan radical, tampoco permite dejar rastros de qué procesos fueron matados, por lo que los usuarios nunca se enterarán si hubo intentos de ejecución de código malicioso.

3. Detener y matar procesos: sirve contra malwares que han encontrado la manera de ejecutarse, en el caso que detecten que han sido detenidos. Es importante considerar en en este caso que los valores de *read* y *write* de la señal SIGKILL deben ser mayores a los de SIGSTOP (Tabla 4).

```
# Estructura de rules.ini
[SIGSTOP]
read=10000
write=10000

[SIGKILL]
read=11000
write=11000
```

Tabla 4. Configuración de rules.ini para una estrategia de detener y matar proceso.

Supongamos que en los últimos T segundos El proceso P se ha detenido cuando alcanzó las condiciones para el SIGSTOP, pero se volvió a iniciar y alcanzó las condiciones para el SIGKILL En este caso, al proceso P se le envía la señal de matar.

4. Notificar estado de procesos: asume que hay un proceso N en el espacio de usuario esperando esta notificación para tomar distintas acciones. También hay que aclarar que, para este caso, se debe adaptar el código del Syscall Top agregando el nombre de dicho proceso (Tabla 5).

```
# Estructura de rules.ini
[SIGUSR1]
read=10000
```

```
write=10000
```

Tabla 5. Configuración de rules.ini para una estrategia de notificar estados de procesos.

Suponiendo que en los últimos 5 segundos un proceso ha superado este umbral, entonces se enviará la señal SIGUSR1 al procesos N, para que éste desencadene sus acciones programadas. Estas acciones podrían ser enviar notificaciones a usuarios, ya sea a través de mail o algún otro medio, enviar datos del procesos a cierta aplicación que los procese en tiempo cuasi real, enviar datos a una persistencia de datos para su posterior análisis, por mencionar algunos ejemplos. Esto ya depende de la aplicación personalizada que se quiera añadir a este ecosistema del Syscall Top.

5. Estrategia combinada: se podrían combinar las anteriores, dando lugar a una estrategia más robusta (Tabla 6).

```
# Estructura de rules.ini
[SIGSTOP]
read=10000
write=10000

[SIGKILL]
read=11000
write=11000

[SIGUSR1]
read=10000
write=10000
```

Tabla 5. Configuración de rules.ini para una estrategia combinada.

Esta configuración permite detener los procesos que superen las 10000 interacciones de *read* y *write* en los últimos T segundos, como así también si este proceso se vuelve a iniciar, lo detiene cuando alcance las 11000 interacciones de *read* y *write*. Además, esta estrategía notifica a un proceso N, si los procesos sospechosos superan las 10000 interacciones también en los últimos T segundos.

3 Conclusiones

La herramienta presentada fue diseñada inicialmente para tener un mayor entendimiento del comportamiento de distintos malwares que se ejecutan sobre un sistema GNU/Linux.

En el proceso recorrido por el proyecto que enmarca este trabajo, se fue descubriendo y agregando funcionalidad para que Syscall Top no solo monitoree los procesos

sospechosos, sino que también podría tomar acciones reactivas frente a estos, tal como se comentan en las primeras tres estrategias descritas en la sección anterior.
El fuerte de Syscall Top es que es adaptable a cualquier entorno actualmente existente en el campo del análisis de llamadas al sistema como se explicó en las estrategias cuatro y cinco.
Esta herramienta pasa a formar parte de la última línea de defensa en el modelo de ciberseguridad de una infraestructura de red, ya que puede ser ejecutada en segundo plano en servidores Linux.
Syscall Top también puede ser utilizado en escritorios personales, no necesariamente servidores. El hecho de que sea de código abierto le permite ofrecer a quien necesite, la posibilidad de ser adaptado para cumplir con las necesidades de cada infraestructura.
El hecho de poder enviar señales a cualquier a cualquier aplicación de usuario, permite a la comunidad sumar módulos satélites con acciones definidas que interpretan estas señales, como por ejemplo un módulo de notificación por mail.
En cuanto a proyecciones futuras, esta herramienta da la posibilidad de explotar las diferentes señales que puede interpretar el kernel linux que sean consideradas que ayuden al análisis de malwares.
Si bien hasta el momento solo se han analizado y experimentado con procesos de ransomwares, el paso siguiente es estudiar procesos de otros malwares y experimentar y analizar posibles estrategias a ser configuradas en el Syscall Top para los mismos.

Referencias

1. Canzanese R. (2015). Detection and Classification of Malicious Processes Using System Call Analysis. Recuperado el 28 de Mayo de 2019 https://pdfs.semanticscholar.org/8060/eae74c98a66cfcc736f4fca61d46f4dbc1d4.pdf.
2. Elisan C. (2015) Advanced Malware Analysis. McGraw-Hill. Capítulo 2. ISBN: 9780071819756.
3. Ethan Rudd, Andras Rozsa, Manuel Gunther, and Terrance Boult. 2017. A survey of stealth malware: Attacks, mitigation measures, and steps toward autonomous open world solutions. IEEE Communications Surveys & Tutorials 19, 2 (2017), 1145–1172.
4. K. Savage, P. Coogan, and H. Lau, (2018). The Evolution of Ransomware. Secur. Response, p. 57, 2015.
5. Morgan S. (Mayo 2017). 2018 Cybersecurity Market Report. Recuperado el 28 de Mayo del 2019 de https://cybersecurityventures.com/cybersecurity-market-report/.
6. Morgan S. (Diciembre 2018). Cybercrime Damages $6 Trillion By 2021. Recuperado el 28 de Mayo del 2019 de https://cybersecurityventures.com/cybercrime-damages-6-trillion-by-2021
7. M. Egele, T. Scholte, E. Kirda, and C. Kruegel, “A survey onautomated dynamic malware-analysis techniques and tools,” ACMComputing Surveys, 2008.
8. S. Forrest, S. Hofmeyr, A. Somayaji, and T. Longstaff, “A sense of selffor unix processes,” in IEEE Security and Privacy, 1996.
9. Haiquan Xiong, Zhiyong Liu, Weizhi Xu, Shuai Jiao LibVMI: a library for bridging the semantic gap between guest os and vmm 12th International Conference on Computer and Information Technology, IEEE (2012), pp. 549-556

10. System calls in the Linux kernel. Part 1. GitBooks. https://0xax.gitbooks.io/linux-insides/content/SysCall/linux-syscall-1.html
11. Searchable Linux Syscall Table for x86 and x86_64. https://filippo.io/linux-syscall-table/. Última visita 08/08/2021.
12. "Usage Statistics and Market Share of Operating Systems for Websites, August 2020", W3Techs, accessed 09.09.2020, https://w3techs.com/technologies/overview/operating_system.
13. "2019 Embedded Markets Study", AspenCore, accessed 09.09.202 0, https://www.embedded.com/wpcontent/uploads/2019/11/EETimes_Embedded_2019_Embedded_Markets_Study.pdf.
14. Gibellini F., Quinteros S., Parisi G., Zea Cárdenas M., Ciceri L., Bertola F., Barrionuevo I., Notreni J., Ruhl A. Monitoreo de Llamadas al Sistema como Método de Prevención de Malware. WSI - SEGURIDAD INFORMÁTICA, CACIC 2021, 27th Congreso Argentino de Ciencias de la Computación, CACIC 2021, Salta, Argentina.
15. Página oficial de Man. Ayuda de Top de Linux. https://man7.org/linux/man-pages/man1/top.1.html
16. Bovet D., Cesati M. Understanding the Linux Kernel. Capítulo 11. Third Edition. O'Reilly Media. ISBN 0-596-00565-2
17. Bovet D., Cesati M. Understanding the Linux Kernel. Capítulo 11. Third Edition. O'Reilly Media. ISBN 0-596-00565-2