Evaluación de la capacidad de ejecución de atacantes en sistemas Unix usando Return Oriented Programming

Jaime Alconchel Gallego, Ricardo J. Rodríguez

Grupo de I+D en Computación Distribuida (DisCo)
Instituto de Investigación en Ingeniería de Aragón (I3A)
Universidad de Zaragoza, Mariano Esquillor s/n, 50018, Zaragoza, Spain.
e-mail: 845977@unizar.es

Resumen

Se ha extendido la herramienta ROP3[1], dedicada a la localización de *rop gadgets* en binarios Windows, a sistemas Unix. Gracias a esta extensión ha sido posible obtener métricas significativas sobre la presencia de *gadgets* útiles en las librerías estándar de C de estos sistemas.

Introducción

En 2021 se presentó el artículo *Evaluation of the Executional Power in Windows using Return Oriented Programming*[2], en el que se ahondaba en trabajos e intuiciones previas sobre la técnica de explotación conocida como programación orientada al retorno (ROP, por sus siglas en inglés). Esta es una técnica de reutilización de código basada en encadenar secuencias de instrucciones acabadas en una instrucción *ret* de la sección ejecutable de los binarios, con el objetivo de lograr ejecución arbitraria de código en sistemas que cuentan con protecciones para evitar la ejecución de zonas modificables de memoria, entre otros. Desde su aparición académica[3], ROP y otras técnicas similares como JOP[4] se han convertido en una pieza fundamental en la explotación de software, si bien se han desplegado poderosas contramedidas[5,6] contra ellas.

La creación de secuencias de código Turing completas a partir de fragmentos presentes en el código es fundamental a la hora de realizar un ataque exitoso. Ante este problema, la solución propuesta en el artículo ya mencionado fue crear un lenguaje similar a ensamblador que permitiera modificar arbitrariamente el valor de los registros de un procesador x86, agrupando todos los *gadgets* presentes en un ejecutable Windows que realizan la misma operación en una pseudoinstrucción *roplang*. Los resultados obtenidos fueron favorables, demostrando la posibilidad de realizar ejecución no condicional completa en casi todos los binarios básicos del sistema Windows 10 y ejecución condicional en los binarios con más instrucciones.

Tomando como base el trabajo realizado para Windows, se ha ampliado la herramienta original para que pueda leer y procesar los formatos binarios ELF, propio de Linux y BSD, y Mach-O, específico de macOS. Adicionalmente, se ha diseñado una nueva forma de realizar ejecución condicional para solventar los problemas del lenguaje *roplang* original y se han elaborado *ropchains* complejos como una máquina de Turing o un shellcode.

Una vez extendida la herramienta, se diseñó un banco de pruebas en Python y Bash para realizar mediciones reproducibles sobre las librerías de C presentes en los siguientes sistemas operativos: *Debian, Ubuntu, Archlinux, Fedora, Void Linux, Slackware, FreeBSD, OpenBSD, NetBSD* y *macOS Sonoma*.

Las librerías C de los sistemas Linux se extrajeron de las versiones más actualizadas de los repositorios de cada distribución en su versión estable, salvo para las distribuciones *rolling release*, para las que se utilizó la versión más actualizada en el momento de la descarga. Las librerías de *macOS* se extrajeron de la cache de dyld mediante la herramienta ipsw y del repositorio *homebrew*. Para la ejecución de pruebas, se usó la versión 3.13 de Python en Debian 13 en un ordenador con CPU AMD Ryzen 1600af y 16GB de memoria RAM.

Resultados

Los resultados obtenidos revelaron en los sistemas Unix una cantidad equivalente de operaciones *roplang* a la encontrada en los binarios de Windows. También se obtuvieron resultados considerablemente mejores en las nuevas operaciones para ejecución condicional. Se incluye al final del documento un mapa de calor de instrucciones *roplang* similar al presentado en el artículo original. La antigua instrucción de salto aparece reflejada en el mapa de calor como *jmp2*. Como referencia, se volvieron a realizar mediciones sobre la librería de Windows SHELL32.dll. A la hora de establecer comparaciones, se debe tener en

cuenta que el tamaño de esta *dll* es considerablemente mayor al de las librerías C.

Conclusiones

A pesar de las nuevas contramedidas contra ataques ROP, este vector de ataque seguirá formando parte durante años de las amenazas del futuro, coordinado necesariamente con otros métodos de explotación. ROP3, una vez ampliado a los sistemas operativos más utilizados, constituye por tanto una herramienta de gran utilidad tanto para escoger cuantitativamente los binarios que mayor capacidad ofrecen para esta técnica como para diseñar *ropchains* gracias a su lenguaje de abstracción.

REFERENCIAS

[1]. RME-DisCo Research Group. Herramienta ROP3 [Online: https://github.com/reverseame/rop3/]. 2021. Accedido el 25/05/2026.

[2]. UROZ, Daniel; RODRÍGUEZ, Ricardo J. Evaluation of the Executional Power in Windows using Return Oriented Programming. En *2021 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2021. p. 361-372.

[3]. SHACHAM, Hovav. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). En *Proceedings of the 14th ACM conference on Computer and communications security*. 2007. p. 552-561.

[4]. BLETSCH, Tyler, et al. Jump-oriented programming: a new class of code-reuse attack. En *Proceedings of the 6th ACM symposium on information, computer and communications security*. 2011. p. 30-40.

[5]. LILJESTRAND, Hans, et al. {PAC} it up: Towards pointer integrity using {ARM} pointer authentication. En *28th USENIX Security Symposium (USENIX Security 19)*. 2019. p. 177-194.

[6]. GARRISON, Tom. Intel CET answers call to protect against common malware threats. *Intel, Mountain View, CA, USA, Jun*, 2020, vol. 15.