Universidad Nacional Autónoma de México

Facultad de Ingeniería

Pointer Authentication Codes

Sistemas Operativos

Roel Alejandro Perez Candanoza
José Carlos Alcántara Hoyos

1. Problema

Un problema común en seguridad informática son las vulnerabilidades de corrupción de memoria, como el desbordamiento de búfer (buffer overflow), o técnicas como Return Oriented Programming (ROP) y Jump Oriented Programming (JOP). Estos consisten en modificar áreas de la memoria en los que se espera que esté un apuntador a la siguiente instrucción. Un atacante puede modificar el apuntador(por ejemplo, desbordando el buffer, como se muestra en la figura 2) de tal forma que la ejecución del programa salte a un área de código controlado por el atacante.

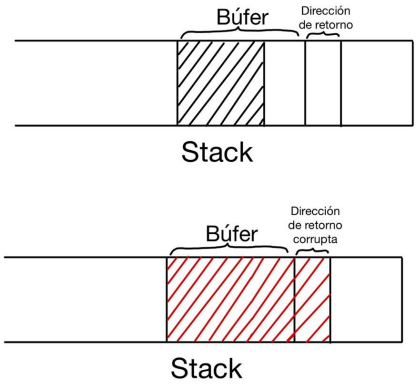

Figura 1: Desbordamiento de búfer. Se observa que al hacer una llamada a una función, un desbordamiento del búfer puede provocar la corrupción del apuntador de retorno.

Existen tres formas comunes de protegerse contra estas vulnerabilidades, las cuales son:

- Almacenar datos sensibles y apuntadores en memoria de solo lectura.
- Verificar la integridad de un apuntador antes de utilizarlo.
- Aleatorizar algún aspecto de la administración de memoria (la pila o el heap)

Las implementaciones de estas soluciones (como Software Stack Protection, o Address Space Layout Randomization) son efectivas y ampliamente utilizadas. Sin embargo, para ciertas arquitecturas, las implementaciones avanzadas se vuelven imprácticas. Por esta razón, se vio la necesidad de implementar en ARM un método para validar la integridad de un apuntador con un impacto mínimo en rendimiento.

2. Pointer Authentication en ARM

La arquitectura del ARMv8.3 añadió una nueva funcionalidad llamada "Pointer Authenticationçon el objetivo de detectar apuntadores creados por alguien externo a la ejecución. Esto se hace agregando una firma criptográfica al valor de los apuntadores, las cuales son validadas antes de usarse. Mientras el atacante no posea la llave usada para crear la firma de seguridad, no podrá ser capaz de crear apuntadores válidos.

Colocar la firma criptográfica dentro del mismo apuntador es posible gracias a que, a pesar de que los apuntadores de los procesadores actuales sean de 64 bits, solo 40 bits tienen información significativa, dejando 24 bits sobrantes para almacenar a la firma criptográfica (bits de extensión de signo).

ARMv8.3 provee cinco llaves para la autentificación de apuntadores: dos son para el uso de apuntadores de funciones, dos para apuntadores de datos, y uno para uso genérico. Esta herramienta solo se puede usar en el espacio de usuario; aún no puede ser usado dentro del kernel. Cuando un proceso es creado, el kernel genera esta llave que se asocia con el contexto del proceso, y es usada para validar los apuntadores, pero el proceso no puede leer esta llave por si solo. Esto para tener mayor seguridad en la llave.

2.1. Autenticación vs. Encripción

Un aspecto importante de diseño acerca de los PACs es que son códigos de autenticación, y no de encripción. En un contexto de comunicación entre un emisor y receptor a través de un mensaje, podemos diferenciar los conceptos de la siguiente manera:

- La encripción sirve para asegurarse de que solo el receptor intencionado recibirá el mensaje del emisor; el mensaje es indescifrable para cualquier otra entidad.
- La autenticación sirve para que el receptor se asegure de que el mensaje es del emisor correcto; el mensaje es público.

Usar la autenticación tiene ventajas sobre la encripción, pues ya que los apuntadores son públicos, pueden aprovecharse de mejoras en el rendimiento durante la ejecución, además de que se puede distinguir una corrupción de memoria de un simple error.

2.2. Instrucciones

Para la implementación de Pointer Authentication se definen dos conjuntos de instrucciones: PAC* y AUT*. El conjunto de instrucciones PAC* sirve para calcular el PAC en sí y ponerlo en el apuntador que deseamos proteger, mientras que el conjunto AUT* se encarga de verificar el PAC del apuntador. En microinstrucciones, la instrucción PAC* se utilizaría antes colocar el apuntador en la pila, mientras el AUT* se utilizaría antes de realizar el return.

Si la autenticación es exitosa durante la instrucción AUT*, se remueve el código asociado al apuntador. En caso contrario, se modifica el PAC para hacer que el apuntador sea una dirección inválida, la cual es atrapada como una excepción. Gracias a AUT*, el manejador de excepciones puede distinguir entre excepción de dirección inválida y un error en la autenticación del apuntador.

	No stack protection	With Pointer Authentication

Function Prologue	`SUB sp, sp, #0x40`
	`STP x29, x30, [sp,#0x30]`
	`ADD x29, sp, #0x30`
	…
		`PACIASP`
		`SUB sp, sp, #0x40`
		`STP x29, x30, [sp,#0x30]`
		`ADD x29, sp, #0x30`
		…

Function Epilogue	…
	`LDP x29,x30,[sp,#0x30]`
	`ADD sp,sp,#0x40`
	`RET`
		…
		`LDP x29,x30,[sp,#0x30]`
		`ADD sp,sp,#0x40`
		`AUTIASP`
		`RET`

Figura 2: Ejemplo de uso de PACs para implementar protección al stack. Imagen de Qualcomm.

2.3. Criptografía

Para generar el PAC es necesario un algoritmo criptográficamente fuerte, aún cuando sea truncado, pero sin comprometer la agilidad de cálculo. Para este fin, se desarrolló QARMA, una familia de unidades de cifrado por bloques.

El código de autenticación es calculado con tres valores: el apuntador, la clave secreta anteriormente mencionada y un tercer valor que puede ser el apuntador actual al stack (memoria stack) o algún otro valor obtenido del entorno o contexto. El objetivo de la llave es hacer que al atacante le sea imposible generar códigos válidos, mientras que el tercer valor se utiliza para asegurarse de que el código fue creado en ese punto de la pila o contexto, previniendo así la reutilización de un código válido, en caso de que se filtrase un apuntador firmado al atacante.

El valor asociado a un código de autenticación no se puede desreferenciar directamente, ya que sin los bits de extensión de signo no se reconoce como una dirección válida. De este modo, para recuperar un apuntador, se requiere utilizar la instrucción AUT*, la cual recalculará el código de autentificación y se comparará con la firma contenida en el apuntador. Si los dos son iguales, se removerá la firma. En caso contrario, se procederá a modificar el apuntador de manera que se garantice que ocurra un error.

2.4. Propiedades de la seguridad

La autenticación de punteros no es sensible tanto a las lecturas como a las escrituras arbitrarias de memoria. Esto quiere decir que aunque un atacante pudiera leer y escribir en la memoria, la seguridad no se vería afectada. En el caso de una lectura arbitraria de memoria, obtener el código de apuntadores con firma no ayudaría a falsificar más códigos, ya que el algoritmo criptográfico que los genera es fuerte. En el caso de escrituras arbitrarias, modificar un apuntador causaría un fallo, dificultando así ataques de fuerza bruta.

3. iOS y PAC

El sistema operativo de los iPhones y iPads es iOS, el cual es un sistema cerrado ya que delimita los permisos al cliente porque no le permite acceder al usuario administrador del sistema. Cuando estos bloqueos son retirados, se habla de un *iOS jailbreak* el cual se lleva a cabo modificando a iOS. El 12 de septiembre de 2018, Apple anunció su proxima generacion de iPhones, los modelos iPhone X [s], X [s] Max y X [r] los cuales implementarían el nuevo modelo de arquitectura ARM v3.8 la cual brinda un conjunto de instrucciones que implementan los PACs (Pointer Authentificacion Codes). Esto representaría una nueva dificultad para hacer *Jailbreak* ya que estos procedimientos para modificar iOS emplean buffer overflows, integer overflows, return oriented programing y otras técnicas enfocadas a la modificación de memoria

3.1. Debilidad del PAC

Existen diversas formas de lograr irrumpir los PACs en iOS, cada una mas compleja que la otra, y lo que buscan es abrir paso a la modificacion del sistema operativo considerando el cambio en la arquitectura en ARM v3.8. Los niveles de privilegios se utilizan para asignar memoria del sistema o para acceder a recursos del procesador, estos se dividen en 4:

- **EL0:** Es llamado espacio de usuario, es el menos privilegiado, es el espacio destinado a las aplicaciones.
- **EL1:** Tiene mayor privilegio que el userspace y contiene al S.O.
- **EL2:**es usado por el Hypervisor, el cual se encarga de soportar maquinas virtuales ofreciendo aislamiento entre ellas.
- **EL3:** Destinado para el monitor seguro, es el que tiene mayor privilegio.

El ataque mas simple a los PACs es leer las llaves desde la memoria del kernel y después de forma manual calcular PACs para usarse en apuntadores arbitrarios del kernel, la documentación especifica que las llaves se encuentran en los registros del procesador y que estos no se encuentran disponibles desde el *user-mode(EL0)*. Si bien, esto es aplicable a los ataques de programas en el espacio de usuario, no toma en cuenta el ataque al kernel. Los dispositivos recientes

de iOS parecen no correr un hypervisor(EL2) o un monitor seguro(EL3), eso significa que el kernel está en EL1 por lo que administra sus propias llaves para PAC. Cuando el kernel se va a dormir, los registros del procesador se borrarán(incluyendo a las llaves del PAC), es por ello que las claves se deben almacenar en algún espacio de memoria del kernel, de este modo un atacante con permisos de lectura de la memoria del kernel podría leer estas llaves y crear sus propios códigos de autentificación. El único problema es conocer el algoritmo usado para generar los códigos, para después implementarlo en espacio de usuario, además, es muy alta la probabilidad de que Apple utilizara un algoritmo modificado o totalmente diferente de QARMA, lo que hace aún mas difícil este método de ataque, requiriendo usar ingeniería inversa.

Referencias

ARMdeveloper. (s.f.). Privilege and exception levels. Descargado de `https://developer.arm.com/architectures/learn-the-architecture/exception-model/privilege-and-exception-levels`

Jailbreak: romper las limitaciones del fabricante en dispositivos ios. (2020). Descargado de `https://www.ionos.mx/digitalguide/paginas-web/desarrollo-web/jailbreak-ios/`

Liljestrand, H., Nyman, T., Wang, K., Perez, C. C., Ekberg, J.-E., y Asokan, N. (2019). Pac it up: Towards pointer integrity using arm pointer authentication. Descargado de `https://arxiv.org/pdf/1811.09189.pdf`

Qualcomm. (2017). Pointer authentication on armv8.3. Descargado de `https://www.qualcomm.com/media/documents/files/whitepaper-pointer-authentication-on-armv8-3.pdf`

R., I. (2018). What do pointer authentication codes mean for ios jailbreaking? Descargado de `https://ivrodriguez.com/pointer-authentication-on-armv8-3`

R., I. (2019). Examining pointer authentication on the iphone xs. Descargado de `https://googleprojectzero.blogspot.com/2019/02/examining-pointer-authentication-on.html`

Rutland, M. (2017). Pointer authentication in aarch64 linux. Descargado de `https://www.kernel.org/doc/Documentation/arm64/pointer-authentication.rst`