Kurz erklärt: Kernel Address Randomized Link

Frisch gemischt

Michael Plura

Mit der neuen Technik KARL würfelt OpenBSD die Kernel-Komponenten für jeden Systemstart neu zusammen und will damit den Herstellern von Kernel-Exploits ein Schnippchen schlagen.

Mitte Juni überraschte OpenBSD-Gründer Theo de Raadt die Mitglieder der Mailingliste *openbsd-tech* mit einem Posting zu einem neuen Zufallsmechanismus namens KARL. Das steht für Kernel Address Randomized Link und soll dem Schutz des Betriebssystemkerns dienen (siehe *iX*-Link [a]).

Der Ausgangspunkt seiner Überlegungen: OpenBSD hat einen Release-Zyklus von etwa sechs Monaten. Da sich bei der Aktualisierung auch eine Version überspringen lässt, ist ein bestimmter OpenBSD-Kernel weltweit für rund sechs bis zwölf Monate im Einsatz. Potenzielle Angreifer hätten dadurch viel zu viel Zeit, Kernel-Exploits zu erarbeiten und diese über Monate einzusetzen. De Raadts Idee: Bootet bei jedem Systemstart ein anderer Kernel, laufen Exploits, die Kernel-Funktionen an bestimmten Speicher-Offsets erwarten, ins Leere.

Kernelbau in wenigen Sekunden

Die daraus entstandene Technik des „Instant-Kernel-Backens“ ist weit weniger kompliziert, als man denken würde. Normalerweise übersetzt der Compiler alle zum Kernel gehörenden Quellcode-Dateien einzeln in Objektdateien. Hinzu gesellt sich ein winziger Bootstrap-Loader oder Urlader, der in der BSD-Familie *locore.s* heißt. Der lädt den eigentlichen Kernel in den Speicher und startet damit das System. Alle Objektdateien des OpenBSD-Standard-Kernels liegen im Verzeichnis */usr/share/compile/GENERIC/*. Hieraus erzeugt der Linker, beginnend mit *locore.o*, anhand festgelegter Regeln unter anderem den OpenBSD-Kernel */bsd* oder */bsd.mp* auf SMP-Systemen.

Während der Compiler-Lauf eine Weile braucht, dauert das Linken auf einem halbwegs aktuellen System nur ein bis zwei Sekunden. Das ebnet den Weg, den Theo de Raadt als KARL bezeichnet: Bei jedem Systemstart erzeugt der Linker einen neuen OpenBSD-Kernel, indem er die Objektdateien in zufälliger Reihenfolge an den *locore*-Bootstrap anhängt. Zusätzlich modifiziert er das Kernel-Layout ein wenig. Das geschieht über die neue Funktion *reorder_kernel()* des Startskripts */etc/rc*.

Die überraschend geringen Änderungen veröffentlichte Theo de Raadt ebenfalls als *diff*-Datei, die die Shellskript-Ergänzungen in *src/etc/rc*, *src/etc/Makefile* und in den architekturspezifischen Makefiles sowie kleine Anpassungen in den *mi*- und *disktab*-Dateien zusammenfasst [b]. Da de Raadt den Weg über das Startskript geht, generiert das während des Hochfahrens bereits den Kernel für den nächsten Systemstart. Dieser neue OS-Kern ersetzt den alten */bsd*, der den Namen */bsd.booted* bekommt.

Die Kernels unterscheiden sich sowohl im Aufbau als auch ein wenig in der Größe. Da beim Durchlaufen der Startskripte der „Einweg-Kernel“ bereits geladen ist, können sie bei dieser Gelegenheit den Bootloader aus dem Arbeitsspeicher kicken, da dessen Position bekannt ist. Je nach Hardwarearchitektur wird er aus dem Speicher gemappt oder mit *trap*-Befehlen überschrieben.

Ein Angreifer, der einen Teil des OpenBSD-Kernels im Speicher identifizieren kann, weiß trotzdem nicht, wo sich dessen übrige Funktionen befinden. Das verhindert auch sogenannte ROP-Angriffe (Return Oriented Programming), bei denen ein Angreifer die Rücksprungadressen des Stacks manipuliert, um direkt in bekannten Kernel-Code zu springen und Techniken wie NX (No eXecute) auszuhebeln. Ein versuchter Exploit könnte bei einem mit KARL generierten Kernel leicht in einem Systemcrash enden – was zu einem Neustart mit einem neuen Kernel führt. Eine verlässliche Informationssuche wird für den Angreifer unmöglich.

Stochern im Nebel

Ähnlich muten Sicherheitstechniken wie das inzwischen verbreitete ASLR (Address Space Layout Randomization) und das darauf basierende KASLR (Kernel Address Space Layout Randomization) an. Bei ASLR weist der Kernel den Anwendungen zufällige Adressbereiche zu, was Pufferüberläufe erschweren soll. Beim KASLR von Linux 4.12 lädt der Bootloader den immer gleichen Kernel in zufällig ausgesuchte Speicherbereiche. KARL hingegen lädt einen Kernel mit zufällig angeordneten Komponenten, ohne die Speicheradresse zu ändern – beides ließe sich auch kombinieren.

Bei jedem Start einen neuen Kernel vorzufinden, mag manchem Systemverwalter merkwürdig vorkommen. Das erinnert an Viren mit Mutation Engines, die auf diese Weise Virenscanner mit Signaturerkennung an der Nase herumführen. In der Entwicklerversion OpenBSD 6.1-current funktioniert KARL bereits reibungslos und verlängert den Startvorgang kaum spürbar. Einzig das Aufwachen aus dem Hibernate-Modus gelingt nicht, weil OpenBSD hier etwas voreilig mit einem neuen Kernel startet, der nicht zum Hibernate-Speicherabbild des vorherigen Kernels passt – hieran arbeiten die Entwickler aber. (sun)

Michael Plura

lebt in Schweden und ist freier Autor mit den Schwerpunkten IT-Sicherheit, Virtualisierung und freie Betriebssysteme.

Alle Links: www.ix.de/ix1710124