Engineer

Kernel

Kern eines Betriebssystems

Was es ist

Ein Kernel ist das Programm im Zentrum eines Betriebssystems. Der Kernel läuft mit den vollen Prozessorrechten und kontrolliert den Zugriff auf die Hardware. Er entscheidet, welches Programm als Nächstes läuft, verteilt Speicher und trennt den Speicher jedes Programms vom übrigen, liest und schreibt Datenträger über Dateisysteme, bewegt Pakete durch den Netzwerkstack und setzt durch, wer auf welche Datei, welchen Port oder welches Gerät zugreifen darf. Alles andere läuft mit weniger Rechten im Benutzerraum und fordert beim Kernel entsprechende Aktionen an.

Diese Anfragen sind Systemaufrufe, laut Linux-Handbuch „die grundlegende Schnittstelle zwischen einer Anwendung und dem Linux-Kernel“. Programme führen sie selten von Hand aus. Sie rufen Wrapper in einer C-Bibliothek wie glibc auf, die die Argumente weiterreichen und die Antwort des Kernels in einen Rückgabewert und einen Fehlercode umwandeln. Stoppt ein Programm mit „Permission denied“, ist dieser Wortlaut meist die von der Bibliothek übersetzte Antwort des Kernels auf einen solchen Aufruf.

Kernel unterscheiden sich darin, was sie in ihrem Inneren behalten. Linux hält Gerätetreiber, Dateisysteme und den Netzwerkstack im Kernel selbst und kann im Betrieb weitere Komponenten als Module laden. Dieser Aufbau heißt gewöhnlich monolithisch. Jeder Treiber läuft dann mit den vollen Kernelrechten. Ein Mikrokernel enthält nur das Minimum. seL4 stellt Mechanismen bereit, um den Zugriff auf den physischen Adressraum, Interrupts und Prozessorzeit zu kontrollieren. Treiber, Dateisysteme und Netzwerkstacks laufen als gewöhnliche Programme im Benutzermodus. Seine verifizierte 64-Bit-RISC-V-Version umfasst etwa 10.000 Codezeilen, mit einem formalen Beweis, dass die C-Implementierung tut, was ihre Spezifikation beschreibt. Der macOS-Kernel XNU ist ein Hybrid, der den Mach-Mikrokernel mit Teilen von FreeBSD kombiniert.

Eine Kernelaktualisierung wird normalerweise erst wirksam, wenn die Maschine mit diesem Kernel startet. Seine Versionsnummer, etwa 6.12, ist von der Versionsnummer der Distribution, die diesen Kernel enthält, unabhängig.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist ein Kernel?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Server, E-Mail, Konten und Updates werden im Alltag betreut, damit die Dinge, auf die alle angewiesen sind, unauffällig funktionieren.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. 10 PostgreSQL-/MS-SQL-Server auf Ubuntu — 10 bereitgestellte und funktionsfähig gehaltene Datenbankserver

10 PostgreSQL-/MS-SQL-Server auf Ubuntu Kontakt aufnehmen →

Warum es wichtig ist

Für ein Unternehmen entscheidet der Kernel drei Fragen: Wie weit sind seine Arbeitslasten tatsächlich voneinander getrennt, wie viel Ausfallzeit kostet eine Sicherheitskorrektur und wo liegt die Wahrheit über eine Maschine?

Zuerst die Isolation. Ein Container besteht aus gewöhnlichen Prozessen, die der Kernel Namespaces und Kontrollgruppen zugeordnet hat. Die Namespaces geben jeder Gruppe ihre eigene Sicht auf Prozesskennungen, Netzwerkschnittstellen, Einhängepunkte und Nutzer. Kontrollgruppen verteilen Prozessorzeit, Speicher und Datenträgerein- und -ausgabe. Jeder Container eines Hosts läuft auf dessen einem Kernel. Ein Fehler darin betrifft daher alle gleichzeitig. Selbst ein Image mit eigenen Bibliotheken hängt weiterhin von den Kernelfunktionen des Hosts ab. Eine virtuelle Maschine startet ihren eigenen Kernel. Für die Dienste eines Teams sind Container eine vernünftige Grenze. Geben Sie bei Kunden, die einander nicht vertrauen, jedem Mandanten einen eigenen Kernel.

Dann die Korrekturen. Ein gepatchtes Kernelpaket auf dem Datenträger ändert nichts, bis die Maschine damit startet. Auf einem Server ist dieser Neustart der Preis. Livepatching existiert, weil Nutzer laut Kerneldokumentation „nur ungern neu starten“: Es lenkt einzelne Funktionen im laufenden System auf korrigierte Fassungen um. So lassen sich nur Funktionen patchen, die der Kernel nachverfolgen kann. Livepatching verschafft daher Zeit für eine dringende Korrektur, ersetzt aber keine Neustarts. Ein Plan mit regelmäßigen, geprobten Neustarts kostet weniger als einer, der davon ausgeht, dass sie nie nötig sind.

Schließlich hält der Kernel den aktuell wirksamen Zustand. Eine Firewalldatei beschreibt Regeln; die Regeln im Paketfilter des Kernels entscheiden über ein Paket. Eine Datei unter `/etc/sysctl.d` beschreibt eine Einstellung; wirksam ist der Wert unter `/proc/sys`. Ein Dienst kann sich als gesund melden, unabhängig davon, ob seine Aktion den Kernel je erreicht hat. Prüfungen des kernelinternen Zustands erkennen Fehler, die beim Prüfen von Konfigurationsdateien durchkämen.

Wie es in der Praxis aussieht

Eine Härtungsprüfung eines Produktionshosts im August 2026 fragte, ob dessen Schutzmaßnahmen gegen SSH-Brute-Force-Angriffe funktionierten. Alle waren konfiguriert und erschienen in jedem Bericht. Die Prüfung glich sie deshalb damit ab, was der Kernel tatsächlich mit einem Paket macht, statt mit den Dateien über den Sollzustand. Aus Sicht des Kernels hatte keine der drei seit dem Aufbau des Hosts auch nur einmal gehandelt. Die Sperren der Jail nannten Port 22. Dort lauschte kein Socket mehr, seit SSH auf Port 1986 gewechselt war. Die Ratenbegrenzung erreichte den Paketfilter zwar. Er wertet Regeln aber von oben nach unten aus und stoppt beim ersten Treffer, sodass eine allgemeinere Regel darüber jedes Paket abfing, bevor die Grenze geprüft wurde. Die Sperraktion rief über eine Shell ein Paketfilterprogramm auf, das sein Paket nur empfiehlt statt zwingend voraussetzt. Auf einem Host ohne dieses Programm startete die Jail, zählte und entschied, scheiterte dann aber genau beim Übergeben einer Regel an den Kernel. Aus der Prüfung folgten zwei Regeln: Eine Sicherheitsmaßnahme wird durch eine Assertion, also eine ausführbare Prüfaussage, belegt; ein Kommentar kann das nicht leisten. Und eine Firewall wird anhand der Reihenfolge geprüft, in der der Kernel ihre Regeln hält, statt nur anhand ihrer Existenz.

Der Sperrpfad erhielt anschließend einen Test, der bei jeder Anwendung der Konfiguration läuft. Jeder Durchlauf sperrt eine Adresse aus einem für Dokumentation reservierten Block, liest die entstandene Regel aus dem Paketfilter des Kernels zurück und hebt die Sperre anschließend in einem Aufräumschritt auf, der in jedem Fall läuft. Erst das Zurücklesen macht daraus einen Test: Der eigene Status des Sperrdiensts würde auch bei einer Sperre ohne entstandene Regel Erfolg melden. Eine reservierte Adresse gehört zu keinem echten Netzwerk. Eine nach einem fehlgeschlagenen Lauf verbliebene Sperre schließt daher niemanden aus. Der Preis sind wenige Sekunden je Durchlauf und eine jedes Mal in Produktion geschriebene und wieder entfernte Firewallregel.

Ein Host mit 464 MB und ohne Platz für einen Metrikstack erhielt sieben nur lesende Rollen, die den Zustand des laufenden Rechners als Markdown ausgeben. Die Momentaufnahme seiner Systemdaten berechnet die Prozessorauslastung aus zwei Messwerten des kernelinternen Zählers. Der Sicherheitsbericht zeigt die Firewallregeln in der Reihenfolge ihrer Auswertung neben den aktiven Sperrregeln. Jede der sieben Rollen gibt nur Zahlen aus, die ihr eigener Durchlauf gelesen hat, darunter die Kernelzähler, niemals eine Schätzung zum Füllen einer Lücke. Diese Berichte fanden zwei unwirksame Sicherheitsmaßnahmen und einen lauschenden Port, den niemand deklariert hatte.

Wo es schiefgeht

Eine von Hand geänderte Kerneleinstellung wird beim nächsten Neustart zurückgesetzt. `sysctl -w` schreibt den Wert nur in den laufenden Kernel. Er bleibt deshalb nur bis zum Neustart erhalten. Der umgekehrte Fehler: Eine Datei unter `/etc/sysctl.d` wird beim Start eingelesen, sodass die Änderung unbemerkt auf einen späteren Neustart wartet. Schreiben Sie die Einstellung in eine Datei, laden Sie sie jetzt und lesen Sie den aktiven Wert zurück:

```
echo 'vm.swappiness = 10' | sudo tee /etc/sysctl.d/90-local.conf
sudo sysctl --system
sysctl vm.swappiness
```

Eine CPU-Kurve bleibt während einer Lastspitze flach. Die Zeile `cpu` in `/proc/stat` zählt Zeit seit dem Systemstart in Einheiten von USER_HZ, auf den meisten Architekturen Hundertstelsekunden. Eine einzelne Messung ergibt daher den Durchschnitt seit dem Start, der sich auf einer monatelang laufenden Maschine kaum bewegt. Lesen Sie zweimal und teilen Sie die Differenzen:

```
head -1 /proc/stat; sleep 5; head -1 /proc/stat
```

Die aktive Zeit ist die Summe der Änderungen von user, nice, system, irq und softirq, die Leerlaufzeit die Summe der Änderungen von idle und iowait. Die Auslastung ist die aktive Zeit geteilt durch die Summe aus aktiver Zeit und Leerlaufzeit. Das Handbuch nennt iowait unzuverlässig. Behandeln Sie es daher als Hinweis statt als Alarm.

Ein Absturzbericht kommt mit der Bitte zurück, den Fehler mit einem Kernel ohne Taint-Markierung zu reproduzieren. Der Kernel setzt eine Taint-Markierung, bezeichnet als tainted, wenn etwas geschieht, das für eine spätere Untersuchung wichtig sein könnte, etwa das Laden eines proprietären, außerhalb des Kernelbaums entwickelten oder unsignierten Moduls. Seine Dokumentation warnt, dass Entwickler Fehlerberichte von Kerneln mit einer solchen Markierung oft ignorieren. Die Markierung bleibt bis zum Neustart erhalten, auch nachdem das Modul entladen wurde. Lesen Sie sie vor dem Melden aus:

```
cat /proc/sys/kernel/tainted
```

Null bedeutet, dass keine Taint-Markierung gesetzt ist. Jede andere Zahl ist eine Summe von Flags, darunter `1` für ein proprietäres Modul, `4096` für ein außerhalb des Kernelbaums entwickeltes und `8192` für ein unsigniertes. Reproduzieren Sie den Fehler nach einem Start ohne das Modul, das die Markierung gesetzt hat.

Eine neue Netzwerkkarte erscheint in der Hardwareliste, aber das System bietet keine Schnittstelle dafür an. Treiber gehören zum Kernel. Ein Kernel, der älter als die Hardware ist, hat möglicherweise keinen passenden. `lspci -k` zeigt für jedes Gerät den verwendeten Treiber und die geeigneten Module. Fehlt bei einem Gerät die Zeile zum verwendeten Treiber, ist keiner gebunden. Installieren Sie eine neuere Kernelserie der Distribution, statt das Herstellermodul von Hand zu bauen. Ein Modul außerhalb des Kernelbaums muss für jeden neuen Kernel neu gebaut werden und setzt beim Laden eine Taint-Markierung.

Weiterführende Lektüre

- Benutzer- und Administratorhandbuch des Linux-Kernels — Das eigene Handbuch des Kernelprojekts für den Linux-Betrieb.
- syscalls(2) — Jeder Linux-Systemaufruf mit der Kernelversion, die ihn eingeführt hat.
- namespaces(7) — Die Arten von Namespaces und wie daraus Container entstehen.
- proc_stat(5) — Jedes Feld von `/proc/stat`, einschließlich der Warnung zu iowait.
- Kernel mit Taint-Markierungen — Jedes Taint-Flag und wie sich die Zahl dekodieren lässt.

- seL4-FAQ — Was ein Mikrokernel auslässt und welche Annahmen hinter dem Beweis von seL4 stehen.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Bereitstellungsweg, die Betriebsrisiken und die Bereitschaftsverantwortlichen. Wir können eine Änderung mit Prüfungen, einem Wiederherstellungsweg und klaren Zuständigkeiten abgrenzen.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist ein Kernel?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/kernel/