Kernel- und Treiberprogrammierung mit dem Linux-Kernel – Folge 97

Kern-Technik

In Kenntnis von Meltdown greift der unprivilegierte Code von Crackern schonungslos auf den geschützten Kernelspace zu. Gepatchte Linux-Systeme setzen der Hardware-Sicherheitslücke Software-seitig die Kernel Pagetable Isolation (KPTI) entgegen. Diese Kern-Technik erklärt beides. Eva-Katharina Kunst, Jürgen Quade

Nach Bekanntwerden der für viele Beobachter „größten Sicherheitslücken aller Zeiten" [1], namentlich Meltdown und Spectre, am 4. Januar 2018 hat das Linux-Magazin einen ersten Überblick und einen lesenswerten Kommentar darüber veröffentlicht ([2], [3]). Diese Kern-Technik entschlüsselt nun die technische Seite des Problems genauer sowie Linux' Entgegnungsstrategie.

Die Sicherheitslücken selbst sind nicht nur deshalb so gefährlich, weil Angreifer damit auch ohne Superuser-Rechte Klartextpasswörter und andere Daten aus dem Hauptspeicher eines Rechners auslesen dürfen. Vielmehr sind mehr oder minder sämtliche PC-Systeme (Desktops, Server, Notebooks) seit zirka 1995, aber auch viele Tablets, Smartphones, NAS oder aktive Netzkomponenten betroffen.

Als Ursache entpuppt sich nicht wie gewöhnlich die Software, sondern die CPU selbst. Darum gelingt der Schutz davor – wenn überhaupt – nur auf Kosten der Performance. Software kann die Lücken nämlich nicht vollständig schließen, ein Löchlein bleibt. Daher sprechen die Linux-Entwickler bei ihren Schutzmaßnahmen auch nur von Mitigation, also Reduzierung. Die ganze Sache ist und bleibt der größte anzunehmende Unfall.

Die Mitigation-Patches sind mit Kernel 4.15 fester Bestandteil von Linux. Greg Kroah-Hartmann hat entscheidende Patches auf ältere Kernel zurückportiert, sodass aktuell gehaltene Systeme einen Basisschutz bieten (siehe Kasten „Ist mein System geschützt?").

Wettlauf als Hürde

Meltdown und Spectre lassen sich als klassische Race Conditions einordnen, die im Zuge der Parallelverarbeitung auftreten. Race Conditions, also Situationen, bei denen das Ergebnis einer Berechnung vom zeitlichen Verlauf oder – praxisnäher formuliert – vom Zufall abhängt, gehören zu den schwierigsten Facetten der Informatik.

Die Codesequenzen, welche die Berechnung oder Verarbeitung mit letztlich ungewolltem Zufallsergebnis durchführen, nennt der Informatiker „kritische Abschnitte". Es gibt mehrere Formen von Race Conditions und entsprechend auch unterschiedlichste Methoden, kritische Abschnitte zu schützen.

Gefährliche Spekulanten

Bei Meltdown und Spectre nennen sich die kritischen Abschnitte „Out of Order Execution" und „spekulative Befehlsausführung" [5]. Neu ist die Erkenntnis, dass es sich bei diesen Technologien um kritische Abschnitte handelt, wahrlich nicht. Auf Konferenzen mindestens seit 2016 war die Schwäche Thema, und im Blogpost [6] hatte ein Mitarbeiter von G Data Advanced Analytics den misslichen Sachverhalt ausführlich dargelegt.

Die Autoren

Eva-Katharina Kunst ist seit den Anfängen von Linux Fan von Open Source. Jürgen Quade ist Professor an der Hochschule Niederrhein. Ihr gemeinsames Buch „Linux-Treiber entwickeln" ist Ende 2015 in vierter Auflage erschienen.

Ist mein System geschützt?

Ob der eigene Kernel geschützt ist oder nicht, lässt sich schnell von jedermann überprüfen (Abbildungen 1 und 2). Bei wem das Verzeichnis »vulnerabilities« erst gar nicht existiert, dessen System ist generell bezüglich Meltdown und Spectre anfällig. Eine Ausgabe wie in Abbildung 1 dagegen darf den Admin beruhigen.

Für die bedenkliche Situation in Abbildung 2 hat der Autor die Schutzmaßnahme beim Booten bewusst deaktiviert. Besitzer eines Raspberry Pi müssen sich übrigens keine Sorgen machen. Die dort eingesetzte CPU ist für Meltdown und Spectre nicht anfällig [4].

```
quade@ezs-mobil:~$ grep . /sys/devices/system/cpu/vulnerabilities/*
/sys/devices/system/cpu/vulnerabilities/meltdown:Mitigation: PTI
/sys/devices/system/cpu/vulnerabilities/spectre_v1:Mitigation: OSB (observable speculation barrier, Intel v6)
/sys/devices/system/cpu/vulnerabilities/spectre_v2:Mitigation: Full generic retpoline
quade@ezs-mobil:~$
```

Abbildung 1: Dieses Ubuntu-16.04-System besitzt einen KPTI-gepatchten Kernel.

```
quade@ezs-mobil:~$ dmesg | grep isolation
[    0.000000] Kernel/User page tables isolation: disabled on command line.
quade@ezs-mobil:~$ grep . /sys/devices/system/cpu/vulnerabilities/*
/sys/devices/system/cpu/vulnerabilities/meltdown:Vulnerable
/sys/devices/system/cpu/vulnerabilities/spectre_v1:Mitigation: OSB (observable speculation barrier, Intel v6)
/sys/devices/system/cpu/vulnerabilities/spectre_v2:Mitigation: Full generic retpoline
quade@ezs-mobil:~$
```

Abbildung 2: Wer KPTI per Bootoption deaktiviert hat (oben), macht Linux verwundbar (unten).

Aber in dem damals bequemen und – wie wir heute wissen – falschen Glauben, niemand könne die kritischen Abschnitte ausnutzen, haben die Prozessorschmieden diese Erkenntnis geflissentlich ignoriert. Erst mit den Ergebnissen des Google-Projekts Zero [7] und weiterer parallel arbeitender Forscher [8] erhielt das Thema die notwendige Aufmerksamkeit.

Mit der Out of Order Execution versuchen Prozessoren seit Mitte der 90er Jahre ihre Verarbeitungsleistung signifikant zu steigern. Dabei nutzt die CPU Wartezeiten, die beim Zugriff auf den Hauptspeicher entstehen, und führt die nächsten anstehenden Befehle, die sie aus ihrer Pipeline entnimmt, prophylaktisch aus.

Die Ergebnisse der außer der Reihe ausgeführten Operationen legt sie in Schattenregistern ab. Ist die Operation gemäß programmierter Abarbeitungsreihenfolge tatsächlich an der Reihe, übernimmt der Prozessor die Ergebnisse zeitsparend aus den Schattenregistern, ansonsten verwirft er sie. Nichts zu verlieren also, aber viel zu gewinnen?

Nicht ganz. Denn pfiffige Hacker nutzen diese Techniken, um Daten aus dem für Otto-Normal-User nicht erreichbaren Kernelspace auszulesen. Um deren Vorgehen zu verstehen, sind allerdings Basiskenntnisse bezüglich der Speicherverwaltung und Prozessorcaches nötig.

Pagetables

Unter anderem auch aus Sicherheitsgründen hat jeder Rechenprozess und auch der Kernel seinen eigenen virtuellen Spei-

cher. Das regelt die Memory Management Unit (MMU), die virtuelle Adressen auf physische Adressen umsetzt. Die Umsetzungsvorschriften liegen in so genannten hierarchisch organisierten Pagetables, wovon jeder einzelne Rechenprozess seinen eigenen Satz hat.

Die Wurzel der Pagetables, der Anker, ist das so genannte Page Global Directory (PGD) oder kurz Page Directory (PD). Der Wechsel von einem Satz Pagetables auf den nächsten beim Übergang zwischen Rechenprozessen (Kontextswitch) ist sehr zeitaufwändig. Der Kernel hat ebenfalls seinen eigenen Speicher, das macht insbesondere bei einem Systemcall, also wenn etwa ein Rechenprozess Daten liest oder schreibt, den Wechsel der Pagetable nötig. Das Ganze passiert sogar zweimal, denn am Ende des Systemcalls, wenn die Kontrolle wieder an den Rechenprozess zurückgeht, muss das Betriebssystem ja die ursprüngliche Pagetable reaktivieren.

Da das Performance-technisch desaströs ist, fügen die Entwickler von Linux, Windows und Mac OS X dem Page Directory eines jeden Rechenprozesses die immer gleichen Pagetable-Einträge für den Kernelspace hinzu. Damit reduziert sich zwar der maximale virtuelle Adressraum, aber der Performancegewinn macht den Verlust mehr als wett. Bei einem 32-Bit-System verringert sich der Bereich von 4 auf 3 GByte, auf 64-Bit-Systemen erreicht der Verlust ohnehin keine praktische Bedeutung.

Mit diesem Kniff muss das System die Pagetables nicht mehr wechseln, wenn ein Rechenprozess einen Systemcall startet. Der normale (unprivilegierte) Prozess erlangt durch die Adressumsetzung in seinen Pagetables aber formell Zugriff auf den Kernelspace. Daher benutzen die Entwickler ein Zugriffsflag, welches das Schreiben und Lesen im nicht-privilegierten Modus verhindert, aber beim Abarbeiten von Kernelcode erlaubt.

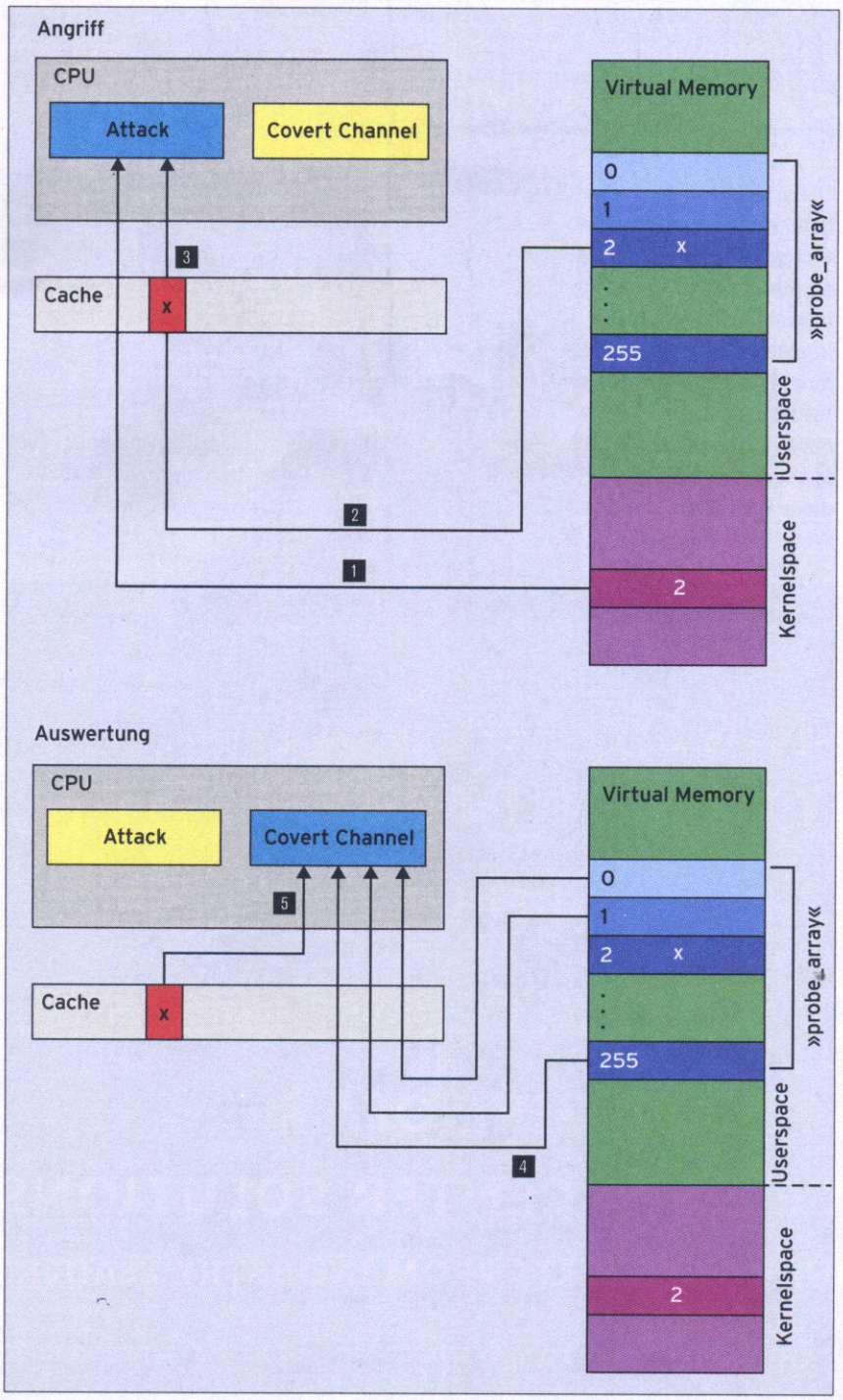

Abbildung 3: Der Meltdown-Mechanismus beim Auslesen des Speichers: 1 Ein Lesezugriff auf die verbotene Speicherzelle, der eine Exception auslöst. 2 Ein Out-of-Order-Speicherzugriff im erlaubten Bereich mit dem Out of Order gelesenen Speicherinhalt als Index. 3 Die CPU verwirft die Out of Order gelesenen Daten, lässt dabei aber den Cache aus. 4 Die Dauer der Blockzugriffe auf den Testspeicher (»probe_array«) ausmessen. 5 Der Zugriff auf den Out of Order gelesenen Wert gelingt schneller, da er auf dem Cache erfolgt.

Schutz abtauen

Der Name Meltdown (Abschmelzen) bezeichnet das Überwinden der Barriere zwischen User- und Kernelspace. Dank dieser Sicherheitslücke greift ein normaler Rechenprozess mit ein paar Tricks auf den Kernelspace zu und liest dort schonungslos Daten aus. Mehr noch: Da im Adressraum des Kernels klassischerweise der ganze physische Speicher eingeblendet ist, liest der Angreifer auch den Speicher anderer Programme aus.

Im Zentrum des Geschehens steht der CPU-Cache. Da Zugriffe auf den normalen Hauptspeicher aus Sicht der CPU schnarchlangsam sind, haben moderne Prozessoren superschnellen Zwischenspeicher (Cache) eingebaut. Die CPU bedient sich nach Möglichkeit aus diesem Puffer. Liegen benötigte Daten dort nicht, transferiert sie der Prozessor aus dem

Hauptspeicher in diesen Cache. Die CPU ist so lange ausgebremst und beschäftigt sich mit Out of Order Executions.

Zum Angriff!

Eine von Meltdown-Angreifern geschriebene Applikation versucht nun (Out of Order) auf eine Speicherzelle im Kernelspace zuzugreifen, deren Inhalt sie lesen möchte. Da der Zugriff illegal ist, löst die CPU eine Exception (Segmentation Fault) aus, aber dennoch liest sie den Inhalt der Speicherzelle – eben Out of Order, die CPU hat nichts Besseres zu tun.

Der Angreifer hat in einem weiteren Out-of-Order-Befehl sogar noch Zeit, um mit dem ausgelesenen Inhalt einen Lesezugriff auf einen im Vorfeld reservierten Speicherbereich durchzuführen. Der reservierte und im Vorfeld aus dem Cache verbannte Speicherbereich besteht aus 256 Blöcken zu je 4096 Bytes. Jeder Block repräsentiert einen der 256 möglichen Werte, die an einer Speicherstelle abgelegt sein können. Die Blockgröße von 4096 stellt sicher, dass sich später über den Cache eindeutig feststellen lässt, auf welchen Block des Speichers die Applikation zugegriffen hat.

Der Trick besteht darin, den Inhalt der Speicherstelle des Kernelspace als Index in den reservierten Bereich zu nutzen. Der Zugriff dient nur dazu, Daten vom Hauptspeicher in den Cache zu transferieren. Sobald die CPU merkt, dass der Zugriff nicht legitim war, verwirft sie die Ergebnisse der Out-of-Order-Operationen, insbesondere den gelesenen Inhalt der Kernelspace-Adresse. Und genau das macht die Intel-CPU nicht komplett – die Designer haben vergessen, Spuren, die bei Speicherzugriffen im Cache zurückbleiben, zu beseitigen (Abbildung 3).

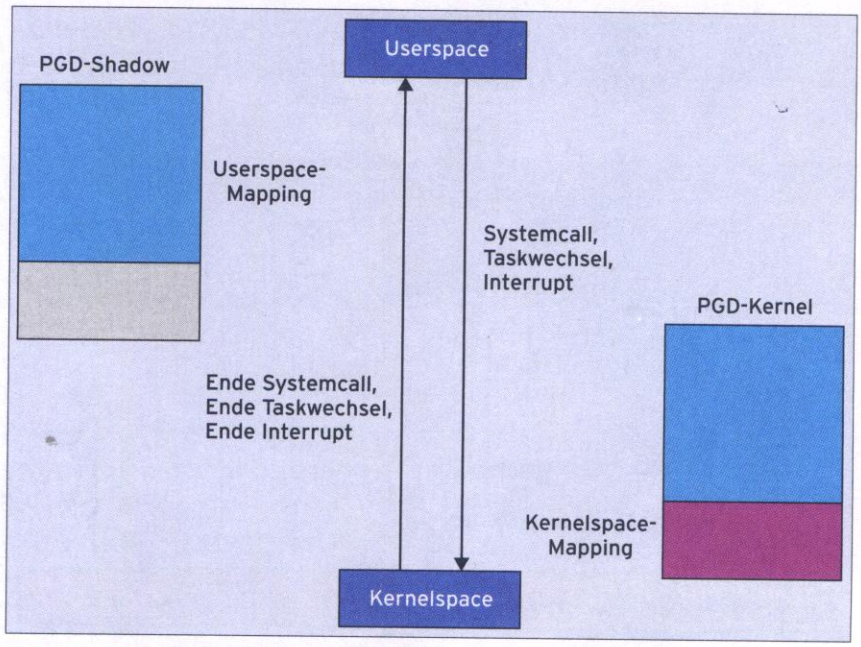

Abbildung 4: Je nach Betriebsmodus aktiviert der Kernel unterschiedliche Versionen der Pagetables.

Timing ist alles

Im Nachgang muss die neugierige Applikation nur nach und nach auf die 256 Blöcke zugreifen und die dafür benötigte Zeit ausmessen. Auf einen der 256 Blöcke hatte der Programmablauf ja Out of Order zugegriffen, und der Inhalt dieses Blocks befindet sich bereits im Cache.

Folglich fällt für diesen die Zugriffszeit signifikant kürzer aus als bei den übrigen Blöcken. Bingo! Da der Wert der Kernelspace-Adresse als Index fungiert hat, entspricht die Blocknummer dem Inhalt der Out of Order gelesenen Kernelspace-Adresse.

Dieses Durchprüfen des Speicherbereichs wird ein Angreifer übrigens typischerweise als einen separaten Thread realisieren. Die Experten sprechen von einem Covert Channel, einem verborgenen Kanal. Bekannter sind aber die Begriffe Seitenkanal und Seitenkanalangriff.

Listing 1 zeigt den Pseudocode-Schnipsel [9], der den Angriff repräsentiert. Zeile 1 reserviert den Speicherbereich, Zeile 2 sorgt dafür, dass der Speicher nicht im Cache ist, und in Zeile 3 findet der Zugriff auf die verbotene Speicherzelle im Kernelspace statt. Er löst die Ausnahmebehandlung aus.

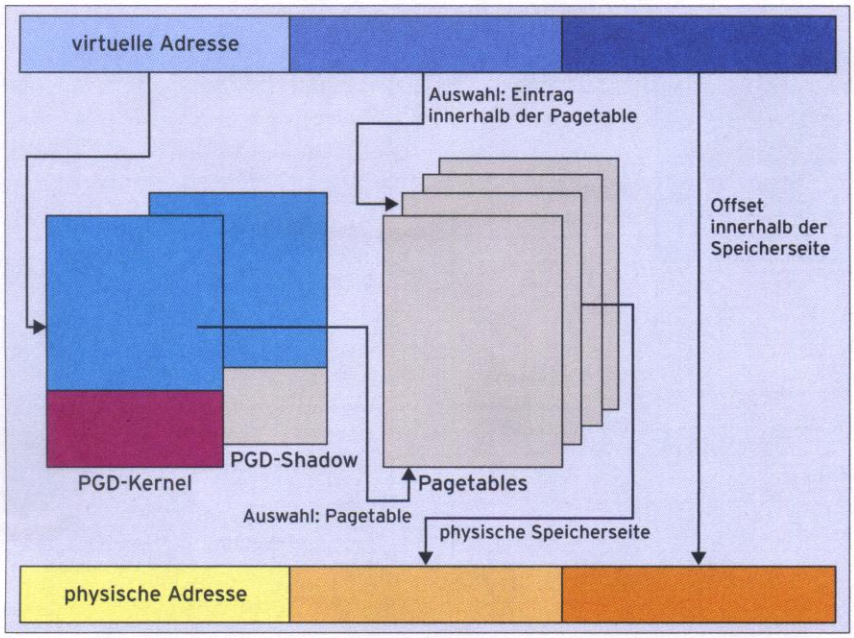

Abbildung 5: Vereinfachte Darstellung der mehrstufigen Adressumsetzung mit KPTI.

Die Zeilen 4 und 5 führt der Prozessor Out of Order aus. Der Code multipliziert den gelesenen Inhalt mit der Blockgröße und erhält den Index auf den zuvor reservierten Bereich. Eine Lese-Operation holt den zugehörigen Inhalt aus dem Hauptspeicher. Der Inhalt selbst ist irrelevant, es geht nur darum, dass das Ganze über den Cache läuft. Jetzt muss der Angreifer den Cache (»probe_array«) Block für Block überprüfen. Die Nummer des Blocks, bei dem die Zugriffszeit kurz ist, entspricht dem Wert an der Adresse im Kernelspace.

Da die Applikation nur Byte für Byte liest, dauert der Angriff nach dem besagten Schema zwar etwas, funktioniert auf ungepatchtes Systemen aber tadellos.

Isolierstation

Zur Abwehr dieses Angriffs implementiert Linux die KPTI (Kernel Pagetable Isolation). Anstelle des gemeinsamen Page Directory verwendet Linux zwei, ein Kernel- und ein Shadow-PGD (Abbildungen 4 und 5). Sie unterscheiden sich dadurch, dass im Shadow-PGD die Einträge für den Kernelspace aufs Minimum (Interrupt-Descriptor-Tabelle, Interrupt-Code-Segment, Per-CPU-Speicherbereiche und der Thread-Stack) reduziert sind, während im Kernel-PGD der gesamte Kernel-Adressraum abgebildet ist.

Dadurch kann aus dem Userspace heraus kein Zugriff auf den Kernelspace erfolgreich sein. Setzt sich dagegen ein Systemcall in Bewegung, aktiviert der Kernel das Kernel-PGD, und der Zugriff ist wie gewohnt möglich. Nach der Bearbeitung und vor dem Wechsel zurück zur eigentlichen Applikation aktiviert der Kernel wieder das Shadow-PGD.

Listing 1: Meltdown-Angriff als Pseudocode

```
01 uint8_t* probe_array = new uint8_t[256 * 4096];
02 // ... Make sure probe_array is not cached
03 uint8_t kernel_memory = *(uint8_t*)(kernel_address);
04 uint64_t final_kernel_memory = kernel_memory * 4096;
05 uint8_t dummy = probe_array[final_kernel_memory];
06 // ... catch page fault
07 // ... determine which of 256 slots in probe_array
   is cached
```

Unsicher, aber schnell

Wie bereits angedeutet, kostet das Auswechseln des PGD Rechenzeit und verlangsamt das System. Das Ausmaß hängt von der Häufigkeit von Taskwechseln und praktizierten Systemcall-Aufrufen ab. Umfangreiche Messungen sind unter [10] zu finden. Um eine Größenordnung zu nennen, die zweifelsfrei nicht repräsentativ ist: KPTI kostet knapp 5 Prozent Performance.

Wer Systeme in einer gesicherten Umgebung betreibt und meint, auf den Schutz verzichten zu können, kann beim Booten die Kommandozeilenoption »nopti« oder alternativ »pti = off« übergeben. Dann arbeitet ein KPTI-gepatchtes Linux wie früher mit einem Page Global Directory pro Task. Linus Torvalds arbeitet übrigens daran, die Option auf Taskebene aktivieren zu können. Damit wird der Admin die Möglichkeit bekommen, einzelne Rechenprozesse mit einer und andere mit zwei PGDs ablaufen zu lassen.

Der Schutz per KPTI gilt als effektiv. Ein Schutz vor den beiden Spectre getauften Sicherheitslücken ist demgegenüber eine Klasse diffiziler. Dazu mehr in der nächsten Kern-Technik. *(jk)*

Infos

[1] Thiemo Hegg, „Größte Sicherheitslücke aller Zeiten“: Frankfurter Allgemeine Zeitung, 7.1.2018, [http://www.faz.net/aktuell/wirtschaft/diginomics/wie-forscher-die-groesste-sicherheitsluecke-aller-zeiten-entdeckten-15379125.html]

[2] Ulrich Bantle, Christian Kißling, „Kernschmelze“: Linux-Magazin 03/18, S. 18

[3] Jan Kleinert, „Chips fressende Köter“: Linux-Magazin 03/18, S. 3

[4] Ebon Upton, „Why Raspberry Pi isn't vulnerable to Spectre or Meltdown“: [https://www.raspberrypi.org/blog/why-raspberry-pi-isnt-vulnerable-to-spectre-or-meltdown/]

[5] Cyberus Technology, „Meltdown and Spectre“: [https://spectreattack.com]

[6] Anders Fogh, „Negative Result: Reading Kernel Memory From User Mode“: [https://cyber.wtf/2017/07/28/negative-result-reading-kernel-memory-from-user-mode/]

[7] Google Project Zero, „Reading privileged memory with a side-channel“: [https://googleprojectzero.blogspot.de/2018/01/reading-privileged-memory-with-side.html]

[8] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, M. Hamburg, „Meltdown“: [https://meltdownattack.com/meltdown.pdf]

[9] Matt Klein, „Meltdown and Spectre, explained“: [https://medium.com/@mattklein123/meltdown-spectre-explained-6bc8634cc0c2]

[10] Brendon Gregg, „KPTI/KAISER Meltdown Initial Performance Regressions“: [http://www.brendangregg.com/blog/2018-02-09/kpti-kaiser-meltdown-performance.html]