Software-basierte Mikroarchitekturangriffe[1]

Daniel Gruss[2]

Abstract: Moderne Prozessoren sind hoch optimierte Systeme, bei denen jeder einzelne Rechenzeitzyklus von Bedeutung ist. Viele Optimierungen hängen von den Daten ab, die verarbeitet werden. Mikroarchitekturangriffe greifen diese Daten unbefugt ab (Seitenkanäle) oder nutzen physikalische Unregelmäßigkeiten aus, um die Kontrolle über das gesamte System zu übernehmen (Fault-Angriffe). In meiner Dissertation [Gru17] habe ich den Stand der Technik bei Mikroarchitekturangriffen und Abwehrmechanismen in drei Dimensionen verbessert. Ich behandle diese kurz in dieser Zusammenfassung. Erstens zeige ich, dass Angriffe vollständig automatisiert werden können. Zweitens präsentiere ich einige neue, bisher unbekannte Seitenkanäle. Drittens zeige ich, dass Angriffe in stark eingeschränkten Umgebungen wie JavaScript in Websites sowie auf jedem Computersystem (Smartphones, Tablets, PCs und kommerzielle Cloud-Systeme) durchgeführt werden können.[3]

1 Einleitung

Die Idee, den Geheimcode für einen Safe zu lernen, indem man auf die Klickgeräusche des Schlosses hört, ist wahrscheinlich so alt wie Safes selbst. Das Klickgeräusch ist ein unbeabsichtigter Einfluss auf die Umgebung, der geheime Informationen preisgibt. 1996 beschrieb Kocher [Koc96] Seitenkanalangriffe, eine Technik, die es erlaubt, geheime Werte, die in einer Berechnung verwendet werden, aus unbeabsichtigten Einflüssen, die die Berechnung auf ihre Umgebung hat, abzuleiten. Diese bahnbrechende Arbeit war der Beginn eines ganzen Forschungsbereichs über Seitenkanäle. Kocher führte durch, was wir jetzt als Timing-Angriff bezeichnen – einen Angriff, der Unterschiede in der Ausführungszeit eines Algorithmus ausnutzt. In den folgenden Jahren wurden Seitenkanalangriffe basierend auf praktisch jeder messbaren Umgebungsänderung demonstriert, die durch verschiedene Arten von Berechnungen verursacht wurden. Beispiele sind etwa der Energieverbrauch, die elektromagnetische Abstrahlung, die Temperatur sowie photonische oder akustische Emissionen. All diese Angriffe haben jedoch gemeinsam, dass ein Angreifer physischen Zugriff auf das Zielgerät haben muss.

Im Gegensatz zu Seitenkanalangriffen, die dem Zielgerät keinen Schaden zufügen, gibt es auch Fault-Angriffe. Bei einem Fault-Angriff versucht ein Angreifer, die Berechnungen eines Geräts zu manipulieren, um Sicherheitsmechanismen des Geräts zu umgehen oder, um Geheimnisse des Geräts preiszugeben. Zu diesem Zweck manipuliert der Angreifer die Umgebung auf eine Weise, die das Zielgerät beeinflusst. Typischerweise können so Fehler induziert werden, wenn die Umgebung an den Rand des Spezifikationsbereichs

[1] Englischer Titel der Dissertation: “Software-based Microarchitectural Attacks”

[2] TU Graz, daniel.gruss@iaik.tugraz.at

[3] Die in meiner Dissertation veröffentlichten Ergebnisse spielten eine zentrale Rolle bei der Entdeckung der Meltdown [Li18] und Spectre [Ko19] Angriffe, die wir im Januar 2018 veröffentlicht haben.

des Zielgeräts oder darüber hinaus bewegt wird. Wie bei Seitenkanalangriffen wurden in der Vergangenheit verschiedene Umgebungsmanipulationen untersucht, z.B. Spannungsstörungen, Taktstörungen, extreme Temperaturen oder der Beschuss mit Photonen. Um einen Fault-Angriff auszuführen, ist ebenfalls eine Form von physischem Zugriff auf das Zielgerät erforderlich.

Moderne Computersysteme sind hochkomplex und hochoptimiert. Durch diese Optimierungen entstehen Informationslecks, unbeabsichtigte Einflüsse auf die Umgebung, die vom verarbeiteten Geheimnis abhängen. Optimierungen basieren auf spezifischen Datenwerten, die verarbeitet werden, dem Speicherort der Daten, der Häufigkeit von Zugriffen auf Speicherorte und vielen anderen Faktoren. Hier wird klar, dass ein Angreifer, der die Auswirkungen dieser Optimierungen durch einen Seitenkanal beobachtet, Schlüsse über die spezifische Ursache der Optimierungen ziehen kann. Das bedeutet, dass der Angreifer Informationen über die geheimen Datenwerte, die verarbeitet werden, lernt.

In meiner Dissertation [Gru17] habe ich softwarebasierte Mikroarchitekturangriffe untersucht. Softwarebasierte mikroarchitekturelle Seitenkanalangriffe nutzen Zeit- und Verhaltensunterschiede aus, die (teilweise) durch mikroarchitekturelle Optimierungen verursacht werden, d.h. Unterschiede, die nicht architekturell definiert oder dokumentiert sind. Softwarebasierte mikroarchitekturelle Fault-Angriffe nutzen physikalische Unregelmäßigkeiten im Gerät aus, um Fehler zu induzieren, d.h. sie operieren Elemente moderner Computersysteme am Rande oder außerhalb ihres Spezifikationsbereichs. Im Allgemeinen erfordern softwarebasierte Mikroarchitekturangriffe keinen physischen Zugriff, sondern nur eine Form der Codeausführung auf dem Zielsystem.

Zusammenfassend betrachten wir also Angriffe, die im Extremfall aus einer Website heraus sensitive Daten eines Systems abgreifen, ohne dass der Nutzer dies bemerkt (Seitenkanalangriffe), oder gar die komplette Kontrolle über das System übernehmen (Fault-Angriffe). Derartige Angriffe aufzuspüren ist essentiell, um die Angriffsfläche zu ermitteln und Gegenmaßnahmen zu entwerfen. Mikroarchitekturangriffe sind hoch-komplexe Angriffe, die weitreichende Grundlagen im Bereich der Betriebssystementwicklung und Prozessorarchitekturen benötigen. Während meiner Dissertation habe ich zu 13 Konferenz-Publikationen (davon sieben bei Top-Tier-Konferenzen) in diesem Themengebiet beigetragen von denen sechs (davon drei bei Top-Tier-Konferenzen) in meiner Dissertation vollumfänglich enthalten sind. Die Relevanz und den wissenschaftlichen Beitrag meiner Dissertation haben wir auch in der jüngeren Vergangenheit eindrucksvoll durch die Angriffe Meltdown [Li18] und Spectre [Ko19] bewiesen, für die meine Dissertation einen wichtigen Grundpfeiler darstellt.

2 Hintergrund

Cache-Angriffe sind die bekannteste Klasse softwarebasierter Mikroarchitekturangriffe. Die Möglichkeit, Timing-Unterschiede, welche durch Prozessor-Caches entstehen, für Angriffe auszunutzen, wurde erstmals von Kocher [Koc96] beschrieben. Cache-Timing-Angriffe wurden zuerst hauptsächlich auf kryptografische Algorithmen in softwarebasierten Angriffen angewendet.

Die meisten Cache-Angriffe in neueren Arbeiten sind Instanzen von drei generischen Cache-Angriffstechniken. Diese Techniken wurden bei gezielten Angriffen auf kryptographische Algorithmen verwendet und später von Osvik et al. [OST06] und Yarom et al. [YF14] verallgemeinert. Diese generischen Techniken sind unabhängig vom spezifischen Cache und der Hardware, auf der sie ausgeführt werden. Osvik et al. [OST06] beschrieben zwei generalisierte Cache-Angriffstechniken. Erstens den *Evict+Time*, bei der ein Angreifer misst, wie die Ausführungszeit eines Algorithmus beeinflusst wird, indem ein Cache-Set – eine Menge von kongruenten Cache-Zeilen – aus dem Cache entfernt wird. Zweitens, *Prime+Probe*, bei der ein Angreifer misst, ob eine Berechnung des Zielprogramms einen Einfluss darauf hat, wie lange es dauert, auf jeden Weg eines gewählten Cache-Sets zuzugreifen.

Bei beiden Angriffen erfährt der Angreifer, dass das gewählte Cache-Set vom Zielprogramm benutzt wurde. Yarom et al. [YF14] führten die dritte generische Angriffstechnik ein namens *Flush+Reload* ein. Bei einem *Flush+Reload*-Angriff löscht der Angreifer einen gemeinsam genutzten Speicherort aus dem Cache und misst anschließend, wie lange es dauert, um darauf zuzugreifen. Wenn das Zielprogramm in der Zwischenzeit den gemeinsam genutzten Speicher wieder in den Cache geladen hat, ist der erneute Zugriff schneller. Bei einem *Flush+Reload*-Angriff erkennt der Angreifer nicht nur, welches Cache-Set vom Zielprogramm verwendet wurde, sondern auch den exakten Speicherort (mit einer Genauigkeit der Cache-Zeilen-Größe).

Basierend auf diesen drei Angriffsprimitiven wurden verschiedenste Berechnungen angegriffen, zum Beispiel kryptografische Algorithmen [YF14], Webserver Funktionsaufrufe [ZJRR14] und Betriebssystemoperationen [HWH13]. Es war jedoch unklar, ob mit diesen Angriffen auch sensitive Informationen wie Nutzerpasswörter und Nutzereingaben ausspioniert werden können.

Alle Cache-Angriffe, die vor meiner Dissertation publiziert wurden, erforderten einen signifikanten Anteil an manueller Arbeit eines Experten, um den Angriff zu entwickeln und durchzuführen. Dies steht im Gegensatz zu klassischer Malware, die voll automatisiert agiert. Eine interessante Fragestellung ist also, inwieweit sich Cache-Angriffe automatisieren lassen. Auf modernen Prozessoren funktionieren außerdem die vor meiner Dissertation publizierten *Prime+Probe*-Angriffe nicht mehr. Die Herausforderung hier lag also darin, Angriffe so weit zu automatisieren, dass sie sich auch auf modernere Systeme automatisch anpassen können.

Softwarebasierte Fault-Angriffe sind in der Praxis erheblich schwieriger durchzuführen, da hier Fehler in der Hardware induziert werden müssen. Daher muss die Software eine Systemkomponente so weit an den Rand ihres Spezifikationsbereichs bringen – oder darüber hinaus –, sodass ein Fehler auftritt. Erst im Jahr 2014 haben sich softwarebasierte Fault-Angriffe als praktisch erwiesen, in dem sogenannten Rowhammer-Angriff [Ki2014; SD15]. Folglich war eine offene Frage die Anwendbarkeit von Rowhammer-Angriffen, d.h. ob Rowhammer beispielweise aus einer Website heraus durchgeführt werden könnte.

3 Beiträge der Dissertation zum Stand der Wissenschaft

Um mögliche Gegenmaßnahmen gegen softwarebasierte Mikroarchitekturangriffe zu entwickeln und zu bewerten, ist es notwendig, die Angriffsfläche detailliert abzubilden und zu verstehen. In meiner Dissertation [Gru17] habe ich dazu beigetragen, das allgemeine Verständnis der Angriffsfläche softwarebasierter Mikroarchitekturangriffe zu verbessern und habe neue Einblicke in softwarebasierte Mikroarchitekturangriffe und Angriffsvektoren geliefert. Meine Forschung umfasst die Minimierung von Anforderungen, die Automatisierung früherer Angriffe und die Identifizierung bisher unbekannter Seitenkanäle.

Ich begann mit der Arbeit an softwarebasierten Mikroarchitekturangriffen, indem ich die *Flush+Reload* Cache-Angriffstechnik [YF14] verbesserte. Frühere Cache-Angriffe erforderten die manuelle Identifikation von Schwachstellen (z.B. spezifische Datenzugriffe oder die Ausführung von Anweisungen, abhängig von geheimen Informationen). In meiner Dissertation [Gru17] habe ich die Technik **Cache-Template-Angriffe** vorgestellt. Cache-Template-Angriffe ermöglichen es, cache-basierte Informationslecks jedes Programms ohne vorherige Kenntnis bestimmter Softwareversionen oder Systeminformationen automatisch zu identifizieren und auszunutzen. Cache-Template-Angriffe können online auf einem Remote-System ohne vorherige Offline-Berechnungen oder Messungen ausgeführt werden. Ich habe diese Technik auf der USENIX Security 2015 Konferenz [GSM15] in Zusammenarbeit mit Raphael Spreitzer und Stefan Mangard veröffentlicht. Cache-Template-Angriffe bestehen aus zwei Phasen. In der Vorbereitungsphase bestimmen wir Abhängigkeiten zwischen der Verarbeitung von geheimen Informationen – z.B. bestimmten Schlüsseleingaben oder privaten Schlüsseln von kryptografischen Primitiven – und spezifischen Cache-Zugriffen. In der Angriffsphase leiten wir die geheimen Werte basierend auf beobachteten Cache-Zugriffen ab.

Wir haben verschiedene Anwendungen von Cache-Template-Angriffen untersucht. Unser automatisierter Angriff auf die T-Tabellen-basierte AES-Implementierung von OpenSSL ist genauso effizient wie die manuellen Cache-Angriffe nach dem Stand der Technik. Unsere Ergebnisse zeigen jedoch auch, dass ein Angreifer sowohl unter Linux als auch unter Windows, die Zeitpunkte sowie weitere Informationen aller Tastenanschläge mit hoher Genauigkeit mitprotokollieren kann. Für Linux-Distributionen demonstrierten wir sogar einen vollautomatischen Keylogger, der die Entropie von Passwörtern von $\log_2(26) = 4{,}7$ Bits pro Zeichen auf 1,4 Bits pro Zeichen signifikant reduziert. Daraus können wir schließen, dass cache-basierte Seitenkanalangriffe eine noch größere Bedrohung für die heutigen Computerarchitekturen darstellen als bisher angenommen. Tatsächlich können selbst sensible Benutzereingaben wie Passwörter, auf Maschinen die Prozessor-Caches verwenden, nicht als sicher betrachtet werden. Grundlegende Konzepte von Computerarchitekturen und Betriebssystemen ermöglichen diese automatische Ausnutzung von cache-basierten Sicherheitslücken. Unsere Beobachtungen haben gezeigt, dass viele der bestehenden Gegenmaßnahmen solche Angriffe nicht wie erwartet verhindern. Insbesondere reicht es nicht aus, nur bestimmte kryptographische Algorithmen wie AES zu schützen. Um der Bedrohung durch automatisierte Cache-Angriffe entgegenzuwirken, müssen wir generische Gegenmaßnahmen finden. Die Tatsache, dass Cache-Angriffe automatisch gestartet werden

können, stellt einen Perspektivwechsel dar, von einem eher akademischen Interesse hin zu praktischen Angriffen, die von weniger versierten Angreifern gestartet werden können.

Während Caches den vergleichsweise langsamen Arbeitsspeicher puffern, puffert der Arbeitsspeicher selbst die noch langsamere Festplatte. Daher sind Seitenkanalangriffe auch auf der Arbeitsspeicher-Ebene möglich. Suzaki et al. [SIYA11] demonstrierte einen Seitenkanal-Angriff auf Seitendeduplizierung, die vom Betriebssystem oder Hypervisor durchgeführt wird. Mit diesem Angriff kann festgestellt werden, ob ein Zielprogramm bestimmte Daten im Speicher hält. Folglich wurde Seitendeduplizierung in öffentlichen Clouds als schädlich betrachtet, aber in einer privaten Umgebung, z.B. privaten Clouds, PCs und Smartphones, immer noch als sicher eingestuft. Wir haben als Erste gezeigt, dass **Seitendeduplizierungsangriffe** sogar **in JavaScript** ausgeführt werden können. Im Gegensatz zu früheren Angriffen erfordert unser Angriff nicht, dass der Zielrechner das Programm des Angreifers ausführt, sondern einfach, dass eine Webseite auf dem Zielrechner geöffnet wird, die den JavaScript-Code des Angreifers enthält. Mit diesem Angriff lässt sich nicht nur feststellen, welche Anwendungen laufen, sondern es lassen sich auch bestimmte Benutzeraktivitäten beobachten, zum Beispiel ob der Benutzer bestimmte Webseiten geöffnet hat. Der Angriff funktioniert auf Servern, PCs und Smartphones und über die Grenzen von virtuellen Maschinen hinweg. Unsere Ergebnisse wurden auf der ESORICS 2015 Konferenz [GBM15] in Zusammenarbeit mit David Bidner und Stefan Mangard veröffentlicht. Die Ergebnisse unserer Arbeit zeigen, dass Systeme mit aktivierter Seitendeduplizierung nicht mehr allgemein als sicher betrachtet werden können. Die Tatsache, dass Seitendeduplizierungsangriffe über Websites gestartet werden können, stellt außerdem einen Paradigmenwechsel dar: von einem gezielten Angriff auf ein bestimmtes System hin zu groß angelegten praktischen Angriffen, die gleichzeitig auf einer großen Anzahl von Geräten ausgeführt werden können.

Basierend auf den beiden zuvor beschrieben Arbeiten untersuchte ich Möglichkeiten **Rowhammer-Angriffe** [Ki2014; SD15] **in JavaScript** durchzuführen. Rowhammer steht im Widerspruch zur grundlegenden Sicherheitsannahme, dass ein Speicherort nur durch Prozesse verändert werden kann, die darauf schreiben dürfen und dies auch tun. Parasitäre Effekte im Arbeitsspeicher können jedoch den Inhalt einer Speicherzelle ändern, ohne darauf zuzugreifen, indem auf andere Speicherstellen mit einer hohen Frequenz zugegriffen wird. Dieser sogenannte Rowhammer-Fehler tritt in den meisten heutigen Speichermodulen auf und hat fatale Folgen für die Sicherheit aller betroffenen Systeme, z.B. für Angriffe mit denen ein einfaches unprivilegiertes Programm Root-Privilegien erhält. Alle vorherigen Rowhammer-Studien und Angriffe beruhten auf der Verfügbarkeit der Cache-Flush-Anweisung, um Zugriffe auf Arbeitsspeicher-Module mit einer ausreichend hohen Frequenz zu verursachen.

Wir überwinden diese Einschränkung, indem wir komplexe Cache-Ersetzungsalgorithmen besiegen. Unsere Ergebnisse zeigen, dass ein Angreifer mit regulären Speicherzugriffen Caches sehr effizient dazu bringen kann, Daten aus dem Cache zu entfernen, und zwar effizient genug, um dann den Rowhammer-Fehler auszulösen. Unsere Arbeit ist die erste Arbeit, die Cache-Ersetzungsalgorithmen aus der Angreiferperspektive genauer untersucht und infolge die komplexen Cache-Ersetzungsrichtlinien erfolgreich umgeht. Dies ermöglicht nicht nur die Implementation von Rowhammer in JavaScript, sondern auch eine bessere

Erforschung von Cache-Angriffen, da nun Angriffe auf aktuelle und unbekannte CPUs schnell und zuverlässig ausgeführt werden können. Früher publizierte Gegenmaßnahmen schützen gegen diesen neuen Rowhammer-Angriff nicht.

Unser vollautomatischer Angriff läuft in JavaScript über eine Website und kann uneingeschränkten Zugriff auf Systeme erhalten. Die Angriffstechnik ist unabhängig von der CPU-Mikroarchitektur, der Programmiersprache und der Ausführungsumgebung. Da reguläre Computersysteme mit DDR3-Modulen und DDR4-Modulen anfällig sind, ist es wichtig, alle Angriffsvektoren von Rowhammer zu finden. Automatisierte Angriffe durch Websites stellen eine enorme Bedrohung dar, da sie auf Millionen von Zielmaschinen gleichzeitig ausgeführt werden können. Die Ergebnisse dieser Forschung wurden auf der DIMVA 2016 Konferenz [Gr16b] in Zusammenarbeit mit Clémentine Maurice und Stefan Mangard veröffentlicht.

Vorgeschlagene Gegenmaßnahmen gegen Cache-Angriffe treffen die Annahme, dass Cache-Angriffe mehr Cache-Hits und Cache-Misses verursachen als gutartige Anwendungen. Daher werden Hardware-Performance-Counter zur Erkennung verwendet. Um zu zeigen, dass diese Annahme nicht gilt, habe ich einen neuen Cache-Angriff namens ***Flush+Flush*** entwickelt. Der *Flush+Flush*-Angriff nutzt lediglich die Ausführungszeit der Flush-Anweisung aus, die davon abhängt, ob Daten im Cache sind. Der *Flush+Flush*-Angriff führt im Gegensatz zu anderen Cache-Angriffen keine Speicherzugriffe aus. Somit verursacht der Angriff überhaupt keine Cache-Misses und die Anzahl der Cache-Hits wird aufgrund der konstanten Cache-Flushes auf ein Minimum reduziert. Aus demselben Grund löst *Flush+Flush* keine Prefetches aus und ist daher in mehr Situationen als andere Angriffe anwendbar. Da der Angriff keine Cache-Fehler verursacht, schlagen Erkennungsmechanismen, die auf Hardware-Performance-Counter zur Überwachung der Cache-Aktivität setzen, fehl, da ihre zugrunde liegende Annahme falsch ist. Der *Flush+Flush*-Angriff kann in einer höheren Frequenz ausgeführt werden und ist daher schneller als jeder vorherige Cache-Angriff. Mit 496 KB/s in einem verdeckten Kanal, der CPU-Kern-übergreifend funktioniert, ist er 6,7 mal schneller als jeder zuvor veröffentlichte verdeckte cache-basierte Kanal. Um den *Flush+Flush*-Angriff zu verhindern, haben wir kleine Hardware-Modifikationen vorgeschlagen: Hätte die `clflush`-Anweisung eine konstante Laufzeit, würden keine messbaren Auswirkungen auf heutige Software entstehen, gleichzeitig der *Flush+Flush*-Angriff aber verhindert werden. Daher ist es eine wirksame Gegenmaßnahme, die implementiert werden sollte. Die Ergebnisse unserer Arbeit wurden auf der DIMVA 2016 Konferenz in Zusammenarbeit mit Clémentine Maurice, Klaus Wagner und Stefan Mangard veröffentlicht. Die Experimente in diesem Papier erweiterten das Verständnis der Interna moderner CPU-Caches. Neben den Ergebnissen zu Erkennungsmechanismen profitiert das Feld der Cache-Angriffe von Erkenntnissen rund um die `clflush`-Anweisung.

Intel x86-CPUs haben in der wissenschaftlichen Gemeinschaft eine beachtliche Aufmerksamkeit erhalten und es wurden leistungsfähige Techniken zum Ausnutzen cache-basierter Kanäle entwickelt. Moderne Smartphones verwenden jedoch eine oder mehrere Multicore-ARM-CPUs, die eine andere Cache-Organisation und einen anderen Befehlssatz als Intel x86-CPUs haben. Diese ARM-CPUs haben normalerweise keine Flush-Anweisung, die für reguläre Programme zugreifbar ist und im Gegensatz zu Intel x86-CPUs teilen sie

auch die Last-Level Caches nicht. Folglich wurden noch keine CPU-Kern-übergreifenden Cache-Angriffe auf nicht gerooteten Android-Smartphones gezeigt. Um hier den Stand der Forschung voranzutreiben, haben wir die wichtigsten Herausforderungen, die diese Angriffe bislang verhindert haben, gelöst und ***Prime+Probe*, *Flush+Reload*, *Evict+Reload* und *Flush+Flush* auf nicht gerooteten ARM-basierten Geräten** ohne jegliche Privilegien demonstriert. Unsere Angriffe sind die ersten kernübergreifenden und prozessorübergreifenden Angriffe auf ARM-CPUs. Basierend auf unseren Techniken haben wir verdeckte Kanäle gebaut, die vorherige verdeckte Kanäle auf Android um mehrere Größenordnungen übertreffen. Darüber hinaus bieten unsere Angriffstechniken eine hohe Auflösung und eine hohe Genauigkeit, die es ermöglicht, einzelne Ereignisse wie Touch- und Swipe-Aktionen auf dem Bildschirm, Touch-Aktionen auf der Soft-Tastatur und die Zeitabstände zwischen Tastenanschlägen zu überwachen. Folglich können wir auch die Länge der auf dem Touchscreen eingegebenen Wörter ableiten. Schließlich zeigen wir den ersten Cache Angriff auf eine kryptografische Primitive, die in Java implementiert ist. Wir zeigen, dass effiziente Angriffe gegen die Standard-AES-Implementierung, die Teil des Java-Bouncy-Castle-Crypto-Providers ist, durchgeführt werden können. Wir zeigen auch, dass die Cache-Aktivität der ARM TrustZone von der normalen Welt aus überwacht werden kann. Unsere Ergebnisse wurden auf der USENIX Security 2016 Konferenz [Li16] in Zusammenarbeit mit Moritz Lipp, Raphael Spreitzer, Clémentine Maurice und Stefan Mangard veröffentlicht. Wir sind davon überzeugt, dass mit unserem neuen Angriff auf Bibliotheken und Apps zahlreiche weitere ausnutzbare Informationslecks aufgedeckt werden. Unsere Angriffe sind auf Hunderte von Millionen von heute verfügbaren Standard-Smartphones anwendbar, da sie alle sehr ähnliche, wenn nicht sogar identische Hardware haben. Dies ist besonders beängstigend, da Smartphones zu den wichtigsten persönlichen Computern geworden sind und unsere Techniken den Umfang und die Auswirkungen von Cache-Angriffen erheblich erweitern.

Um neue unbekannte Seitenkanäle zu finden, habe ich das Verhalten von Prefetch-Anweisungen untersucht. Hierbei habe ich festgestellt, dass diese Anweisungen verwendet werden können, um moderne Betriebssysteme anzugreifen. Moderne Betriebssysteme verwenden Hardware-Unterstützung, um sich vor Kontrollfluss-Manipulation und Code-Manipulation zu schützen. Beispielsweise werden Schreibzugriffe auf ausführbare Seiten verhindert, und die Ausführung im Kernelmodus ist nur auf Kernel-Code-Seiten beschränkt. Gegenwärtige CPUs bieten jedoch keinen Schutz gegen Code-Recycling-Angriffe wie ROP (return-oriented programming). ASLR (Address-Space-Layout Randomization) wird verwendet, um diese Angriffe zu verhindern, indem alle Adressen für einen Angreifer unvorhersehbar gemacht werden. Somit basiert die Kernelsicherheit unter anderem auch auf dem Verhindern des Zugriffs auf Adressinformationen.

Im Rahmen meiner Dissertation [Gru17] entwickelte ich **Prefetch-Seitenkanalangriffe**, eine neue Klasse von generischen Angriffen, die gravierende Schwächen in Prefetch-Anweisungen ausnutzen. Die Timing-Unterschiede bei Prefetch-Anweisungen stammen von einer zweiten Cache-Hierarchie in modernen Prozessoren für Seitentabelleneinträge, die neben der regulären Cache-Hierarchie existiert. Ich habe herausgefunden, dass die Ausführungszeit von Prefetch-Anweisungen vom Zustand dieses Caches zur Seitenübersetzung abhängt. Ein noch gravierenderes Problem ist, dass die x86-Prefetch-Anweisungen

unprivilegierten Prozessen erlauben, privilegierten Speicher in den Cache zu laden. Damit erlauben diese neuen Angriffe nicht-privilegierten lokalen Angreifern, die Zugriffskontrolle auf Adressinformationen vollständig zu umgehen und somit ein gesamtes physisches System zu kompromittieren, indem sie Sicherheitsmechanismen wie SMAP, SMEP und Kernel-ASLR umgehen. Prefetch-Seitenkanalangriffe funktionieren in nativen und virtualisierten Umgebungen gleichermaßen.

Wir haben zwei Primitive eingeführt, die die Basis unserer Angriffe bilden. Erstens das Übersetzungsebenenorakel, welches die Ausführungszeit von Prefetch-Anweisungen ausnutzt. Zweitens, das Adressübersetzungsorakel, welches die fehlenden Privilegien-Überprüfungen ausnutzt.

Das Übersetzungsebenenorakel ermöglicht ASLR zu besiegen und Bibliotheken und Treiber in unzugänglichen Speicherbereichen zu lokalisieren. Mit dem Adressübersetzungsorakel können wir auf 64-Bit-Linux-Systemen aus unprivilegierten Benutzerprogrammen und sogar in einer virtuellen Amazon EC2-Maschine, virtuelle Adressen in physikalische Adressen übersetzen.

Wir bauen drei Angriffe aus, die diese Primitiven ausnutzen. Unser erster Angriff erstellt ein exaktes Bild der gesamten Seitentabellenhierarchie eines Prozesses und hebelt damit alle ASLR Varianten aus. Bei unserem zweiten Angriff lösen wir virtuelle auf physikalische Adressen auf, um dann SMAP auf 64-Bit-Linux-Systemen mithilfe von Angriffen im Stil von ret2dir zu umgehen. Auf der Grundlage beider Orakel haben wir demonstriert, wie Kernel ASLR unter Windows 10 besiegt werden kann – eine Grundlage für ROP-Angriffe auf Kernel- und Treiber-Binärcode. Als Gegenmaßnahme habe ich schließlich eine neue Form von Kernelisolierung vorgeschlagen. Dieser Vorschlag ist mittlerweile auch bekannt durch seine praktischen Implementationen, z.B. KAISER [Gr17] oder KPTI, die den Adressraum von Programmen vom Adressraum des Betriebssystemkernels trennt. Diese Gegenmaßnahme erfordert nur wenige – jedoch tiefgreifende – Änderungen in Betriebssystemkerneln. Die Leistungseinbuße auf aktueller Hardware und bei realistischen Anwendungsszenarien liegt zwischen 0,06% und 5,09%. Mittlerweile haben alle verbreiteten Betriebssysteme meinen Vorschlag aufgenommen und Varianten dieser Sicherheitsmaßnahme implementiert. Unsere Ergebnisse wurden auf der CCS 2016 Konferenz [Gr16a] in Zusammenarbeit mit Clémentine Maurice, Anders Fogh, Moritz Lipp und Stefan Mangard veröffentlicht.

4 Schlussfolgerungen

Aus meiner Dissertation und den Publikationen, die sie umfasst, können wir Rückschlüsse auf vier verschiedene Achsen ziehen.

Erstens können Mikroarchitekturangriffe weitgehend automatisiert werden. Die Automatisierung macht es deutlich einfacher, Mikroarchitekturangriffe durchzuführen. Sie ermöglicht aber auch eine Großzahl von Angriffen mit geringem zusätzlichem Aufwand.

Zweitens: Unbekannte und neuartige Seitenkanäle existieren sehr wahrscheinlich und werden sehr wahrscheinlich auch gefunden.[4] Wir haben bereits gezeigt, dass moderne Mikroar-

chitekturen mehrere zuvor unbekannte Seitenkanäle haben, wie beispielweise der `clflush`-Anweisung [GMWM16], den Arbeitsspeicher [Pe16] oder Prefetch-Anweisungen [Gr16a].

Drittens ist es möglich, die Anforderungen bekannter Angriffe auf einen Punkt zu reduzieren und zu minimieren, an dem sie in stark eingeschränkten und Sandbox-Umgebungen durchgeführt werden können. Wir haben dies in unserer Arbeit zu Rowhammer-Angriffen in JavaScript [Gr16b] und in unserer Arbeit zu Seiten-Deduplizierungsangriffen in JavaScript [GBM15] gezeigt.

Viertens ist es eine schwierige Aufgabe, effektive und effiziente Gegenmaßnahmen zu entwickeln. Die Forschung versucht oft überambitioniert, universelle Gegenmaßnahmen gegen Mikroarchitekturangriffe zu finden und ignoriert, dass die verschiedenen Angriffe sehr unterschiedliche Anforderungen und Eigenschaften haben [GMWM16; Pe16]. Im Mittelpunkt von Mikroarchitekturangriffen steht meist ein zeitlicher oder verhaltensbedingter Unterschied, der vom Prozessorhersteller zur Optimierung der Performance angestrebt wird. Daher kann es schwierig sein, immer eine universelle Gegenmaßnahme zu finden, die die Leistung nicht beeinträchtigt, da Sicherheit und Leistung sich oft widersprechen [Gr16a].

Literaturverzeichnis

[GBM15] D. Gruss, D. Bidner und S. Mangard. “Practical Memory Deduplication Attacks in Sandboxed JavaScript”. In: *ESORICS*. 2015.

[GMWM16] D. Gruss, C. Maurice, K. Wagner und S. Mangard. “Flush+Flush: A Fast and Stealthy Cache Attack”. In: *DIMVA*. 2016.

[Gr16a] D. Gruss, C. Maurice, A. Fogh, M. Lipp und S. Mangard. “Prefetch Side-Channel Attacks: Bypassing SMAP and Kernel ASLR”. In: *CCS*. 2016.

[Gr16b] D. Gruss, C. Maurice und S. Mangard. “Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript”. In: *DIMVA*. 2016.

[Gr17] D. Gruss, M. Lipp, M. Schwarz, R. Fellner, C. Maurice und S. Mangard. “KASLR is Dead: Long Live KASLR”. In: *ESSoS*. 2017.

[Gru17] D. Gruss. “Software-based Microarchitectural Attacks”. Diss. Graz University of Technology, 2017.

[GSM15] D. Gruss, R. Spreitzer und S. Mangard. “Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches”. In: *USENIX Security Symposium*. 2015.

[HWH13] R. Hund, C. Willems und T. Holz. “Practical Timing Side Channel Attacks against Kernel Space ASLR”. In: *S&P*. 2013.

[Ki2014] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai und O. Mutlu. “Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors”. In: *ISCA*. 2014.

[4] Diese Vorhersage aus meiner Dissertation hat sich in einem unvorhergesehenen Ausmaß bewahrheitet, und zwar mit Meltdown [Li18] und Spectre [Ko19].

[Ko19] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz und Y. Yarom. “Spectre Attacks: Exploiting Speculative Execution”. In: *S&P*. 2019.

[Koc96] P. C. Kocher. “Timing Attacks on Implementations of Diffe-Hellman, RSA, DSS, and Other Systems”. In: *Crypto*. 1996.

[Li16] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice und S. Mangard. “ARMageddon: Cache Attacks on Mobile Devices”. In: *USENIX Security Symposium*. 2016.

[Li18] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom und M. Hamburg. “Meltdown: Reading Kernel Memory from User Space”. In: *USENIX Security Symposium*. 2018.

[OST06] D. A. Osvik, A. Shamir und E. Tromer. “Cache Attacks and Countermeasures: the Case of AES”. In: *CT-RSA*. 2006.

[Pe16] P. Pessl, D. Gruss, C. Maurice, M. Schwarz und S. Mangard. “DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks”. In: *USENIX Security Symposium*. 2016.

[SD15] M. Seaborn und T. Dullien. “Exploiting the DRAM rowhammer bug to gain kernel privileges”. In: *Black Hat Briefings*. 2015.

[SIYA11] K. Suzaki, K. Iijima, T. Yagi und C. Artho. “Memory Deduplication as a Threat to the Guest OS”. In: *EuroSec*. 2011.

[YF14] Y. Yarom und K. Falkner. “Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack”. In: *USENIX Security Symposium*. 2014.

[ZJRR14] Y. Zhang, A. Juels, M. K. Reiter und T. Ristenpart. “Cross-Tenant Side-Channel Attacks in PaaS Clouds”. In: *CCS*. 2014.

Daniel Gruss, geboren am 16. September 1986 in Brühl, Deutschland. Er begann sein Informatikstudium 2008 an der Technischen Universität Graz und promovierte 2017 mit Auszeichnung. Er erhielt eine Auszeichnung für die beste Bachelorarbeit am Institut für Angewandte Informationsverarbeitung und Kommunikation der Technischen Universität Graz. Er hält regelmäßig Vorträge bei akademischen und industriellen IT-Sicherheitskonferenzen. Mit durchschnittlich drei Top-Tier-Publikationen pro Jahr zählt Daniel zu den produktivsten System-Security-Forschern. Seine Forschung ist nicht nur in der Wissenschaft, sondern auch in der Industrie bekannt, wie zum Beispiel die erste Rowhammer-Attacke aus Sandbox-JavaScript oder die Meltdown- und Spectre-Attacken, die enorme Auswirkungen auf die gesamte digitale Welt hatten.