UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR IT-SICHERHEIT

Übungsblatt 11

Microarchitectural Attacks

Abgabe bis 04. Juli 2023 um 12:00 Uhr im Moodle

Cybersecurity im Sommersemester 2023

Thomas Eisenbarth, Jan Wichelmann, Anja Köhl

Einführung

Die vergangenen Ereignisse haben das Management dazu bewogen, wieder zur alten Abrechnungsmethode zurückzukehren, woraufhin auch die ehemaligen Kunden wieder neue Verträge abschließen – offenbar hatten sie einige Bedenken bezüglich der Vertrauenswürdigkeit von EVE. Das heißt aber natürlich nicht, dass die Systeme von ITS nun perfekt und sicher wären: Kaum eine Woche später häufen sich auf einmal die Beschwerden, dass trotz der TLS-Verschlüsselung wichtige Daten abhanden gekommen wären, die verschiedene Kunden im Kundenportal eingegeben haben.

Hier zeigt sich nun endlich ein kleiner (versehentlicher) Vorteil des neuen Kontextwechsel-Abrechnungsverfahrens: Um die Implementierung des Verfahrens zu debuggen, wurden regelmäßig sämtliche internen *Performance Counter* der verwendeten Serverprozessoren ausgelesen. Dies gibt aber nicht nur Informationen über Kontextwechsel und CPU-Auslastung aus, sondern auch über interne Mechanismen wie spekulative Ausführung und Caches. Techniker Thorsten fällt nun bei der finalen Auswertung der Kontextwechsel-Zähler zufällig auf, dass die Anzahl der Cache Misses extrem erhöht ist.

Da er sich regelmäßig in den Medien über neue Sicherheitslücken informiert, hat er eine Idee was dieser Messwert bedeuten könnte: Möglicherweise handelt es sich um einen Cache-Angriff? Er zieht Sie hinzu, um Ihn im die Thematik tiefer zu erklären und eine geeignete Gegenmaßnahme zu entwerfen.

Aufgabe 1 Angriffe mit dem Cache als Seitenkanal (6 Punkte)

Bevor es los geht, bittet er Sie um eine Erklärung zu den möglicherweise eingesetzten Angriffen; er vermutet entweder *Flush & Reload* oder *Prime & Probe*.

1. Erklären Sie in Stichworten, wie der jeweilige Angriff grundsätzlich abläuft.
2. Welche Voraussetzungen müssen jeweils gegeben sein?
3. Welche Auflösung (zeitlich und auf den Daten) ist jeweils möglich?

Aufgabe 2 Flush & Reload-Angriff implementieren (8 Punkte)

Um den Angriff nachstellen zu können, bittet Thorsten Sie um eine Referenzimplementierung des *Flush & Reload*-Angriffs.

Er hat bereits etwas Vorarbeit geleistet (siehe Moodle): In der Datei `victim.c` befindet sich ein Array `keys` mit zehn Schlüsseln, die jeweils 64 Bytes groß sind (also jeweils eine Cache-Line füllen). Die Funktion `select_key` greift auf einen dieser Einträge zu und kopiert ihn in ein weiteres Array. Ziel ist nun, nur über *Flush & Reload* und Aufrufe von `select_key` herauszufinden, auf welchen Eintrag zugegriffen wird.

Bearbeiten Sie *ausschließlich* die Datei `flush_reload.c`. Sie können das Programm mit `compile.sh` kompilieren; testen Sie Ihre Implementierung mit einem Intel-Prozessor, andernfalls können die Ergebnisse anders ausfallen (falls Sie keinen Intel-Prozessor zur Verfügung haben, können Sie auch auf einen Pool-PC zurückgreifen[1]).

Testen Sie zuerst, ob der Ansatz grundsätzlich funktioniert:

1. Sorgen Sie dafür, dass ein beliebiger Schlüssel aus `keys` nicht im Cache liegt.
2. Messen Sie nun die Zeit, die Sie zum Laden dieses Schlüssels brauchen.
3. Messen Sie unmittelbar danach ein zweites Mal die Zeit, die Sie zum Laden dieses Schlüssels brauchen. Wie verhalten sich die Zugriffszeiten?

Zum Entfernen von Speicher aus dem Cache und zum Messen von Zugriffszeiten sind in dem Code bereits zwei Hilfsfunktionen `clflush()` und `rdtsc()` vorgegeben. Neben der Implementierung des jeweiligen Befehls fügen diese zusätzlich Fencing-Instruktionen ein, um zu verhindern, dass spekulative Ausführung des Messcodes die Ergebnisse verfälscht.

Starten Sie nun eine Messreihe, in der Sie den Aufruf der Funktion `select_key` so einbauen, dass Sie feststellen können, welcher der Schlüssel von ihr geladen wurde. Führen Sie die Messreihe oft genug durch, um statistisch belastbare Werte zu erhalten. Geben Sie Ihren Code sowie eine kurze Zusammenfassung Ihrer Ergebnisse (Messwerte, Schwellwert für den korrekten Schlüssel, Beobachtungen, ...) ab.

Aufgabe 3 Constant-Time Conversion (6 Punkte)

Jetzt muss nur noch der Angriffsvektor identifiziert werden. Da Cache-Angriffe meistens ausnutzen, dass Implementierungen datenabhängige Ausführungspfade oder Speicherzugriffsmuster haben, werfen Sie zuerst einen Blick in die verwendete TLS-Bibliothek. Und tatsächlich – Sie finden die folgende Konstruktion in einer Entschlüsselungsroutine:

```
1  ...
2  for(int i = 0; i < secret_key.length; ++i)
3  {
4      if(secret_key[i] == 0)
5      {
6          a = a * b
7          b = b * b
8      }
9      else
10     {
11         b = a * b
12         a = a * a
13     }
14 }
15 ...
```

`secret_key` ist hierbei ein Array von Bits (`0` oder `1`).

Die sicherste bekannte Maßnahme gegen derartige Cache-Angriffe ist es, *Constant Time Code* zu schreiben – also sämtliche schlüsselabhängigen Sprünge und Speicherzugriffe zu entfernen, oder zu verschleiern. Können Sie Thorsten helfen, die Implementierung abzusichern?

1. Erklären Sie kurz, wie und warum der Angreifer hier Informationen abgreifen kann.
2. Konvertieren Sie den Code so, dass keine schlüsselabhängige Information mehr nach außen dringen kann. Sie können davon ausgehen, dass die verwendeten arithmetischen Operationen *Constant Time*-Eigenschaften haben.

   Hinweis: Nutzen Sie *Masking*.

[1] Sie erreichen Pool-PC `XX` (z.B. `01`) aus dem Internet mittels
`ssh -J <name>@ssh-gate.uni-luebeck.de <name>@64pcXX.pools.uni-luebeck.de`
Überprüfen Sie mittels `lscpu`, dass Sie wirklich auf einem Intel-Prozessor arbeiten.

Aufgabe 4 Spectre und Meltdown (6 Punkte)

Während Sie noch an Ihrem Patch werkeln, macht sich Thorsten auf die Suche nach dem Verursacher der Angriffe. Hierfür beginnt er, die Prozesse der einzelnen Kunden jeweils kurz auf einem physischen Prozessor zu isolieren, um anschließend lokal deren Cache Miss Counter auszuwerten. Die meisten sind unauffällig, aber ein Prozess eines gewissen „Ernst Vincent Ehrlich“ sticht heraus. Bei einer genaueren Analyse stellt Thorsten fest, dass der Prozess zusätzlich eine hohe Zahl Exceptions verursacht, sodass der Prozesser ständig zwischen dem Kernel und dem Prozess hin und her springt. Er vermutet, dass der Besitzer des Prozesses versucht, mittels eines *Spectre*- oder *Meltdown*-Angriffs den Speicher des Kernels auszulesen, welcher unter anderem sensible interne Konfigurationsdaten enthält.

1. Erklären Sie jeweils die grundsätzliche Funktionsweise eines Spectre- und eines Meltdown-Angriffs. Gehen Sie hierzu auf die jeweils nötigen Voraussetzungen ein, und beschreiben Sie das Vorgehen des Angreifers. Wie unterscheiden sich die beiden Angriffe?
2. Welche der beiden Angriffsarten wurde in dem von Thorsten entdeckten Prozess vermutlich eingesetzt? Begründen Sie Ihre Antwort.