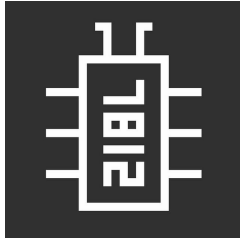


Introduction to software-based microarchitectural side-channel attacks

Abc Xyz
@dura_lex



DCG#7812
2018

Agenda

Introduction

Theory

Basic attacks

Software-based Microarchitectural Fault Attacks

Meltdown & Spectre

Summary

Agenda

Introduction

- Side-channel attacks

- Microarchitectural attacks

Agenda

Introduction

Side-channel attacks

Side-channel attacks



Example of target for side-channel attack

Идея атак по сторонним каналам **весьма стара**, ещё в 1980-х годах было о них известно. Но широкое распространение данный вид атак получил только после публикации **Пола Кохера в 1996 году**.

Самый примитивный пример атаки по сторонним каналам — **определение нажатых кнопок сейфа по звуку** при введении секретного кода.

Такого рода атаки обычно основываются на вычислении изменений в окружающей среде, например, изменения в **потреблении тока устройством, электромагнитном излучении, температуре, по издаваемым акустическим сигналам, по времени, затрачиваемому на выполнение тех или иных операций и другие**.

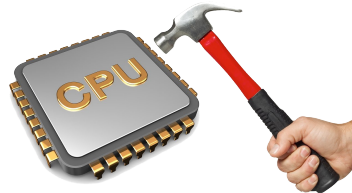
Agenda

Introduction

- Microarchitectural attacks

Microarchitectural attacks

```
code1a:  
  mov (X), %eax  
  mov (Y), %ebx  
  clflush (X)  
  clflush (Y)  
  jmp code1a
```



The DRAM cells get permanently damaged if hammered for a long time

Атаки по сторонним каналам на микроархитектуру, основанные на использовании программного обеспечения, как правило, даже **не требуют физического доступа** к вычислительному устройству.

Также существуют атаки, **основанные на дефектах микроархитектуры**, например, ошибки, происходящие **во время оптимизации**.

Атаки на микроархитектуру, которые используют аппаратные дефекты, **сложно воссоздать на практике**. Примеров таких атак не много, но все они широко известны, это например, **Rowhammer атака**, которая, в случае успешно разработанного потока инструкций, может дестабилизировать работу процессора и даже нанести **неисправимые физические повреждения**, если атака будет проводиться в течении нескольких недель.

Все уязвимости можно найти, просто почитав главу оптимизаций в **спецификации процессора**, даже на Wiki есть раздел про оптимизацию работы CPU, в которой перечислены все элементы, в которых были найдены уязвимости.

Agenda

Theory

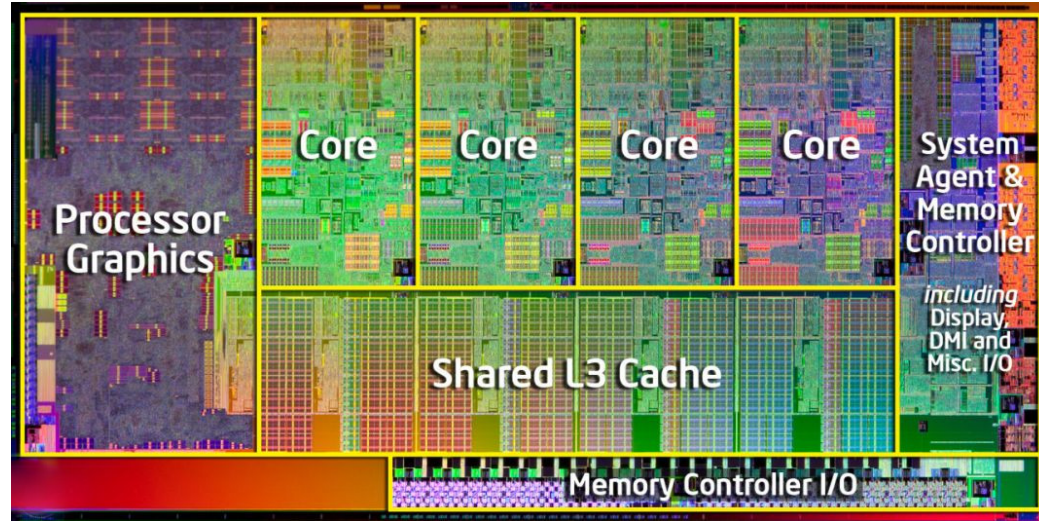
- CPU
- Cache
- DRAM

Agenda

Theory

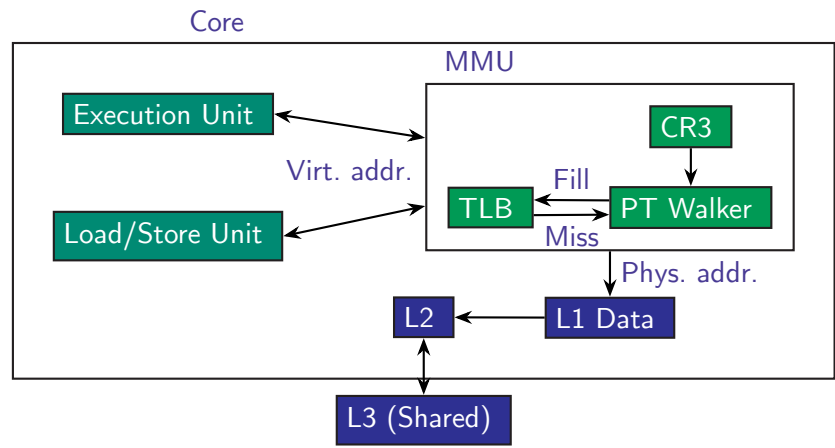
CPU

- Pipelining
- Branch Prediction and Speculation
- Multicore



Architecture of multicore CPU

Современные процессоры состоят из множества крупных элементов: ядер, графического процессора, общего кэша, контроллера памяти и других.

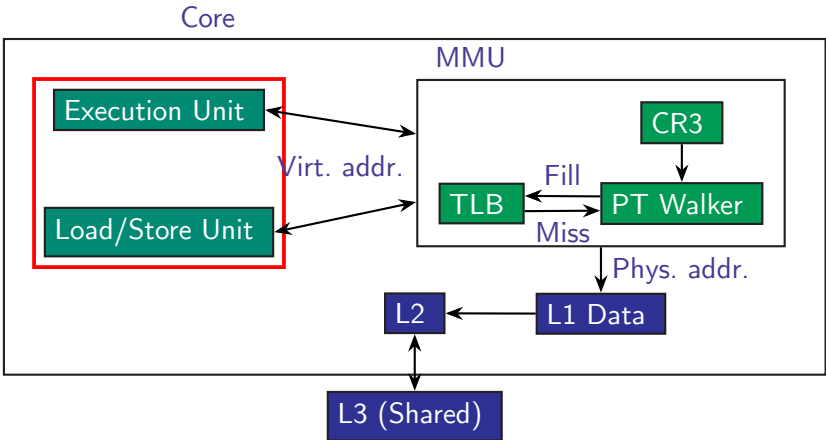


Abstract architecture of core and memory organization

На рисунке 4 представлен общий план работы ядра с памятью и кэшем в Intel процессорах. Более подробно об алгоритмах работы будет рассказано ниже.

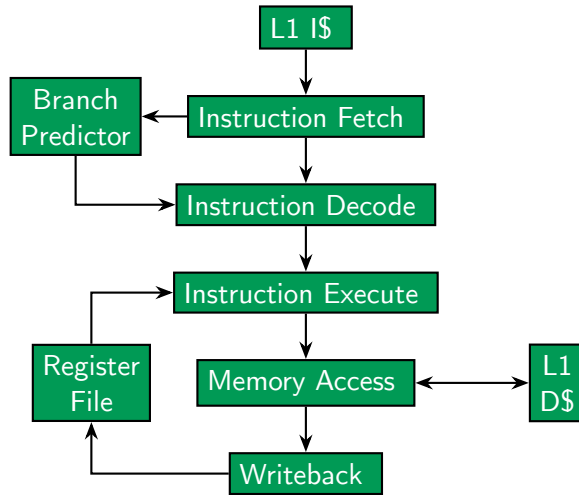
Современные процессоры представляют из себя **сильно распараллеливаемые машины**, которые оперируют данными на высоких скоростях. Размеры процессоров уменьшаются, уменьшается потребление памяти используемое для вычисления одной и той же операции, что позволяет **увеличивать тактовую частоту**. Однако, существуют и другие способы уменьшить время, затрачиваемое на выполнение инструкций — **различные оптимизации**, типы которых зависят от данных и состояния процессора.

Рассмотрим некоторые **системы оптимизации, применяемые в ядрах и процессоре**.



Abstract architecture of core and memory organization

Pipelining. In-Order

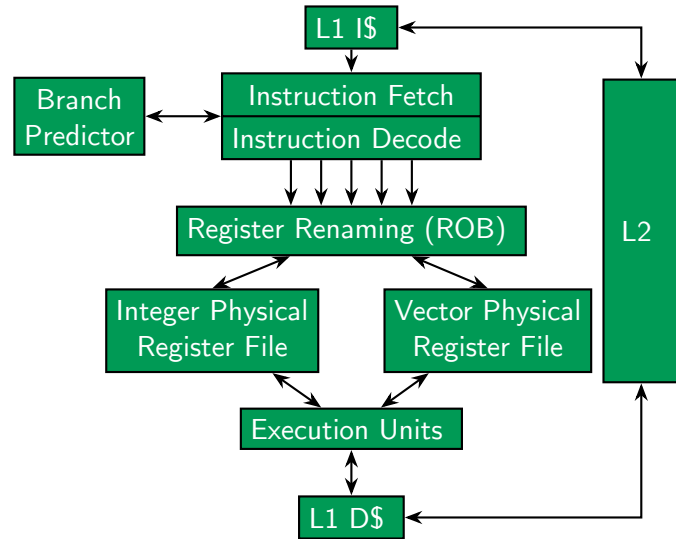


Elements of a modern in-order core

Конвейеризация — одна из главных причин высокой скорости работы процессора. В результате данного процесса **работа с инструкциями разделяется** на несколько этапов (рисунок 5, **выполнение инструкций по порядку**):

- **этап получения**, в результате которого код операции инструкции загружается в процессор;
- **этап декодирования**, в результате которого опкод декодируется во внутреннее представление процессора;
- **этап выполнения** — инструкция исполняется.

Pipelining. Out-of-Order



Elements of a modern out-of-order core

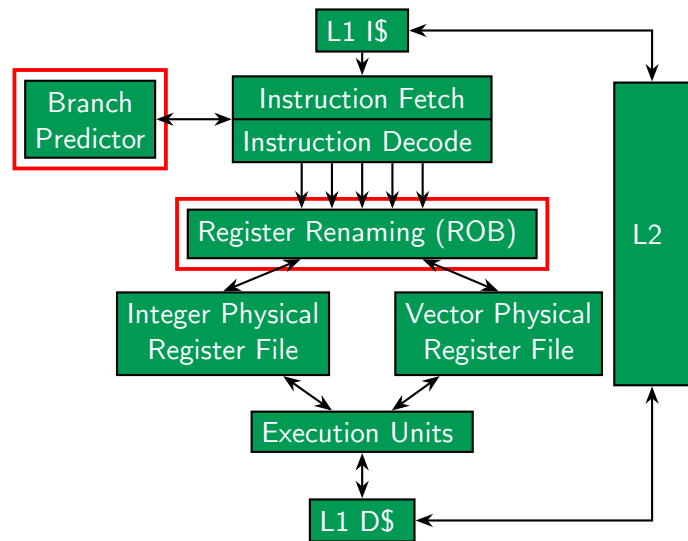
Рисунок 6, выполнение инструкций не по порядку.

Инструкции получаются и декодируются по порядку во **front-end**.

ROB — Re-Order Buffer.

Именно поэтому процессор может выполнять несколько инструкций **одновременно**, при этом не обязательно в порядке их следования. Современные процессоры также могут параллельно выполнять одни и те же стадии для оптимизации вычислений.

Pipelining. Out-of-Order

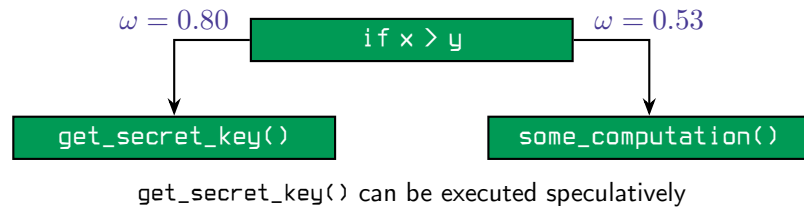


Elements of a modern out-of-order core

О работе Reorder Buffer **рассказано не будет**, существует множество его реализаций.

Ниже **будет рассказано** о работе этих элементов.

Branch Prediction and Speculation

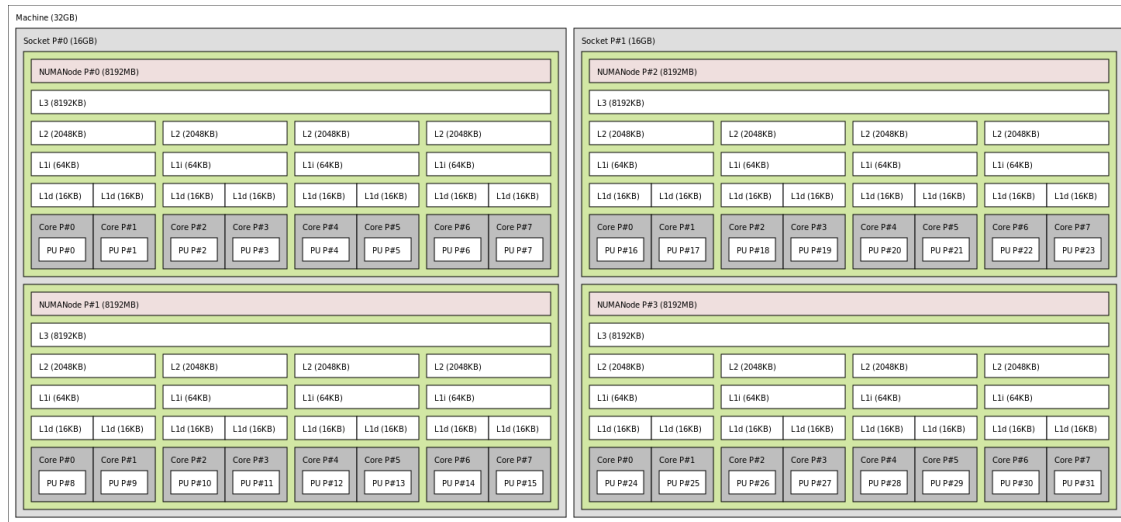


Ещё одна идея для повышения производительности процессора — **исполнение инструкций спекулятивно**. С помощью такой оптимизации процессор **угадывает возможный переход и исполняет его** прежде, чем он может выполняться на самом деле. Если угаданный путь был верным, то процессор просто берёт информацию, которую получил заранее, в противном случае **информация** о ходе спекулятивного выполнения просто **удаляется**.

Существуют:

- Статическое предсказание
- Динамическое предсказание
 - счётчик с насыщением
 - адаптивный двухуровневый предсказатель
 - локальный предсказатель перехода
 - глобальный предсказатель перехода
 - гибридный предсказатель перехода
 - предсказатель для цикла
 - предсказатель косвенных переходов
 - предсказатель инструкций возврата
 - предсказатель, основанный на машинном обучении

Multicore



Architecture of multicore CPU AMD Bulldozer

Вместо оптимизации скорости выполнения на единственном ядре, также существует возможность **увеличивать количество этих самых ядер**. Особенно часто много ядер установлено в процессорах, работающих на серверах, это позволяет выполнять многие независимые друг от друга задачи параллельно. Однако, если задачу невозможно распараллелить, то прироста в производительности, конечно же, не будет. В настоящее время, **почти любое устройство имеет несколько ядер**, в том числе IoT устройства и домашние компьютеры. К тому же, многие языки программирования позволяют без лишних сложностей писать приложения, которые будут исполняться на нескольких ядрах. **Каждое** из таких **ядер имеет свои приватные ресурсы**, например, регистры и конвейеры выполнения, а также **общие ресурсы**, например, основной доступ к памяти.

Agenda

Theory

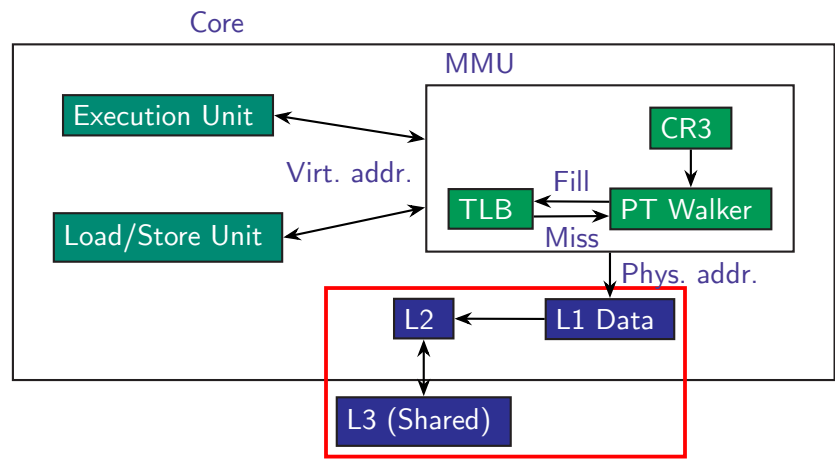
Cache

- Types of cache

- Two-way set associative cache

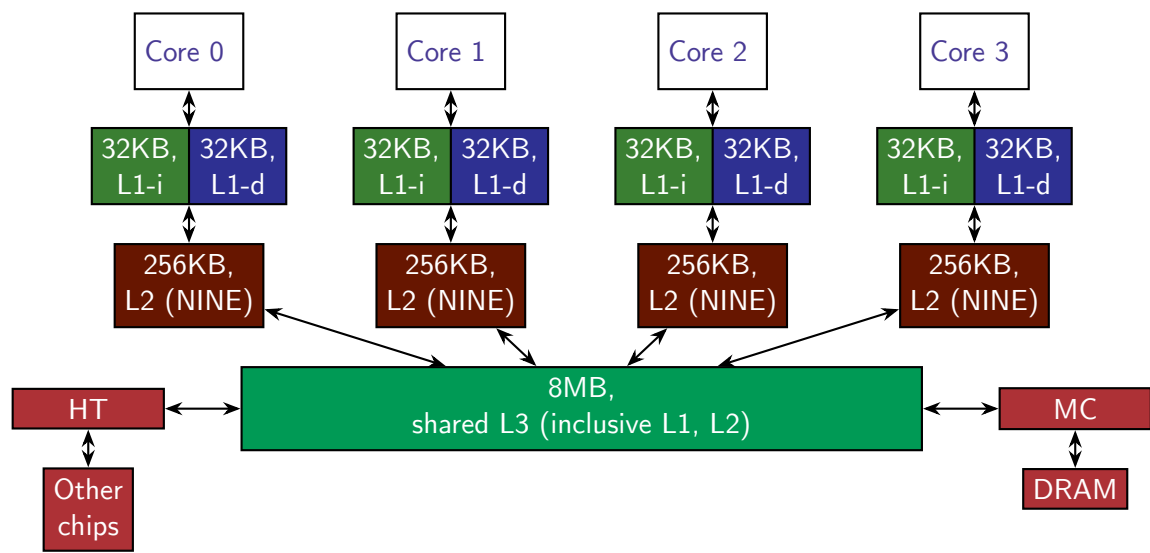
- Cache replacement policies

- Addressing modes



Abstract architecture of core and memory organization

Cache

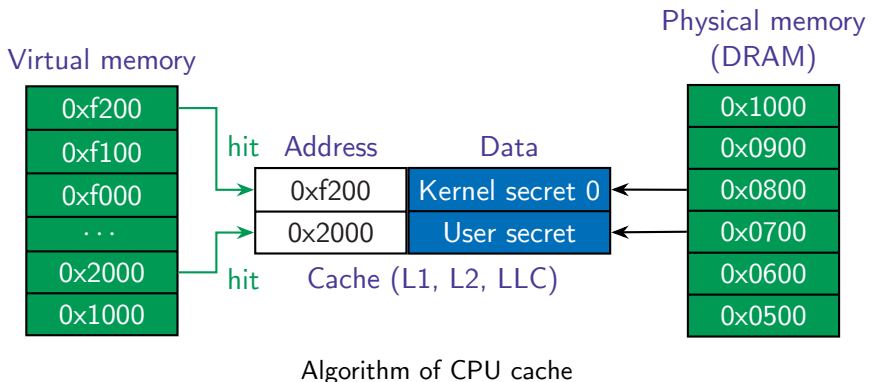


Современные процессоры имеют целую **иерархию кэшей** с различными размерами и скоростью обращения. Некоторые кэши приватные и работают в контексте **только одного процессора**, некоторые **общие**, их могут читать и писать все процессоры. Существует несколько **правил включения (инклюзивности)** кэша один в другой: правило **инклюзивности**, **эксклюзивности**, **NINE**. HT — HyperTransport; MC — Memory Controller.

The flow of data through a modern platform

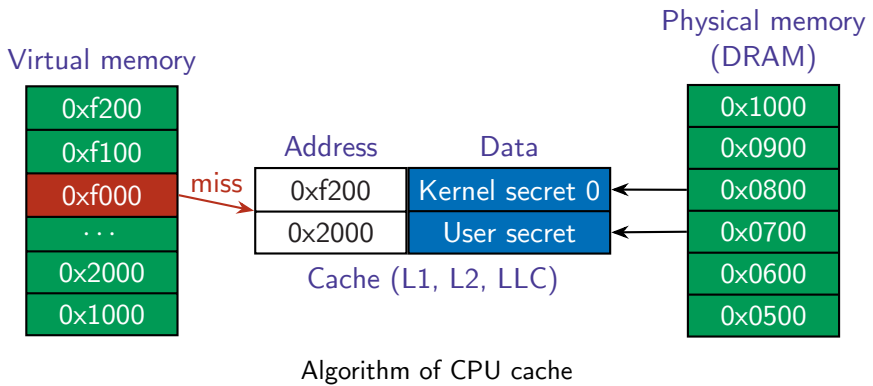
Cache

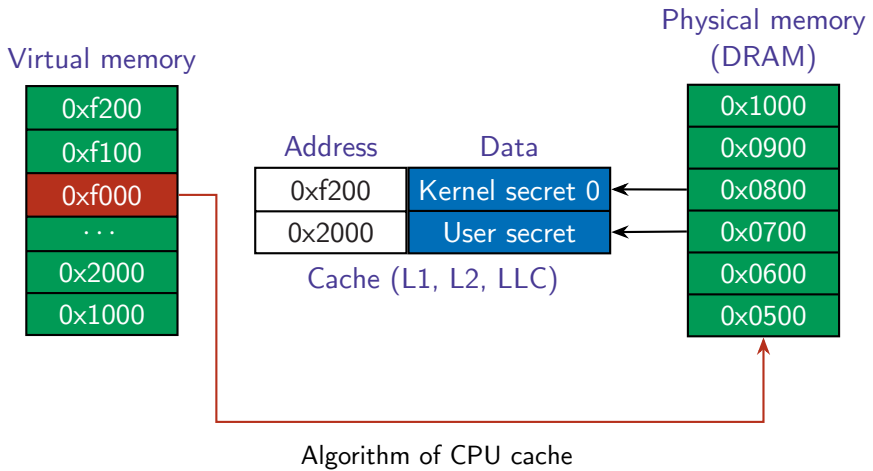
В общем случае, все доступы к памяти происходят через кеш. Если доступ к памяти происходит через кеш, то это называется **попаданием кэша** (*cache hit*).



Cache

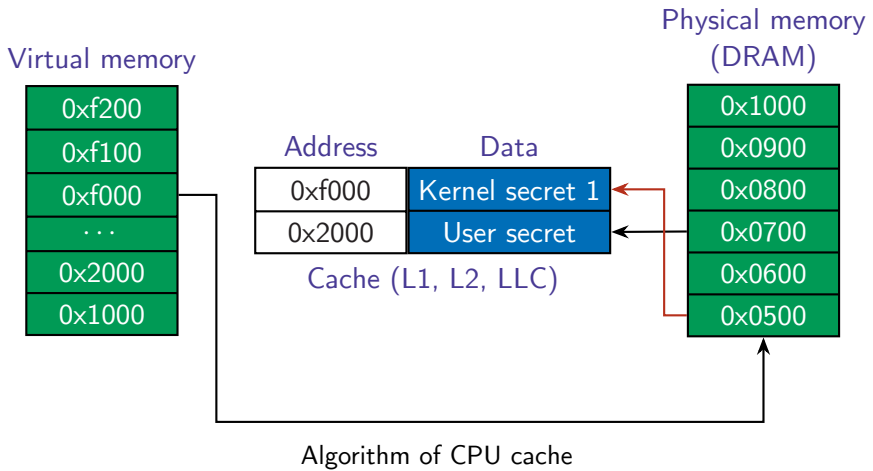
В противном случае происходит **промах кэша** (*cache miss*).





Cache

В кэш записываются новые данные.

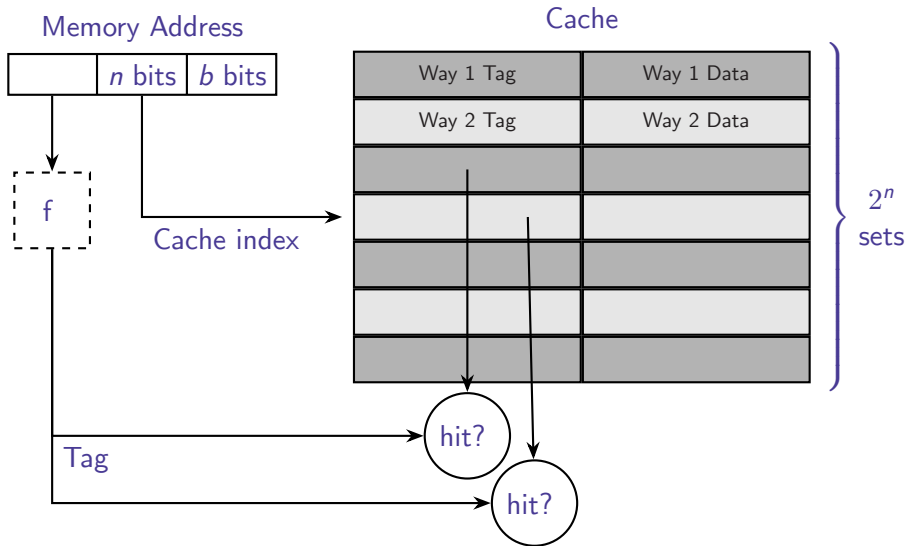


Types of cache

- ▶ Direct-mapped cache
- ▶ Fully-associative cache
- ▶ 2/4/8/12-way set associative cache

1. Главная **проблема** такого вида кэша — это то, что кэш может **хранить единственную** линию кэша из всех **конгруэнтных**. Следовательно, если процессору требуется работать с двумя или более конгруэнтными линиями кэша, то такого рода кэш будет совершать **множество промахов**.
2. Такие кэши становятся более **дорогими с увеличением путей**. Поэтому они обычно содержат небольшое количество путей, например, в современных процессорах используются **буферы ассоциативной трансляции (translation-lookaside buffers TLB)** с 64 путями.

Two-way set associative cache



Компромиссом между этими двумя видами кэша оказывается **кэш с наборами**, а не с линиями кэша. Данные кэши широко используются в современных процессорах, где их называют ***m*-путейные (или *m*-входовые) кэши с ассоциативным набором**. Рисунок отображает абстрактную модель 2-путейного кэша данного вида. Кэш делится на 2^n набора. **Индекс набора** в кэше определяется средними n битами адреса. Каждый набор имеет ***m* путей** для возможности хранения местоположения ***m* конгруэнтных адресов**. Наборы кэша могут быть также представлены в виде крошечного полностью ассоциативного кэша с m путями для набора конгруэнтных адресов. Поэтому **тег** снова используется для определения какой путь кэша содержит определённый адрес.

Cache replacement policies

- ▶ FIFO
- ▶ LIFO
- ▶ least recently used, LRU
- ▶ time aware least recently used, TLRU
- ▶ most recently used, MRU
- ▶ pseudo-LRU, PLRU
- ▶ random replacement, RR
- ▶ segment LRU, SLRU
- ▶ least frequently used, LFU
- ▶ least frequent recently used, LFRU
- ▶ LFU with dynamic aging, LFUDA
- ▶ low inter-reference recency set, LIRS
- ▶ adaptive replacement cache, ARC
- ▶ clock with adaptive replacement, CAR
- ▶ multi queue, MQ
- ▶ and etc.

Количество путей или линий в кэше ограничено, а конгруэнтных адресов, которые требуется хранить, — **достаточно много**, требуется производить замены данных в кэше на новые, полученные из главной памяти.

Производители процессоров хранят детали этих правил в **секрете**, так как данные правила очень сильно влияют на скорость работы процессора в целом.

Самое широкое распространение получили правила вытеснения «**вытеснение давно неиспользуемых**» (**least-recently used, LRU**).

Процессоры **ARM** обычно используют правила **случайного вымещения**, так как такие правила просто реализовать на аппаратных средствах, и в ходе своей работы они потребляют мало энергии, а также показывают себя высокопроизводительными.

Addressing modes

- ▶ Virtually indexed, virtually tagged (VIVT)
- ▶ Physically indexed, virtually tagged (PIVT)
- ▶ Virtually indexed, physically tagged (VIPT)
- ▶ Physically indexed, physically tagged (PIPT)

Кэши могут использовать как **виртуальные адреса**, так и **физические для вычисления индекса кэша и тега**. На практике используется три способа вычисления данных.

Позволяет использовать **тег из физического адреса**, при этом небольшая задержка, так как для поиска в первую очередь и чаще всего требуется определить номер набора, который задан виртуальным адресом.

- уникальный тег — **возможность применять разделяемые данные**
- всё происходит быстро, потому что **трансляция** адреса происходит **параллельно** поиску **индекса кэша**

Agenda

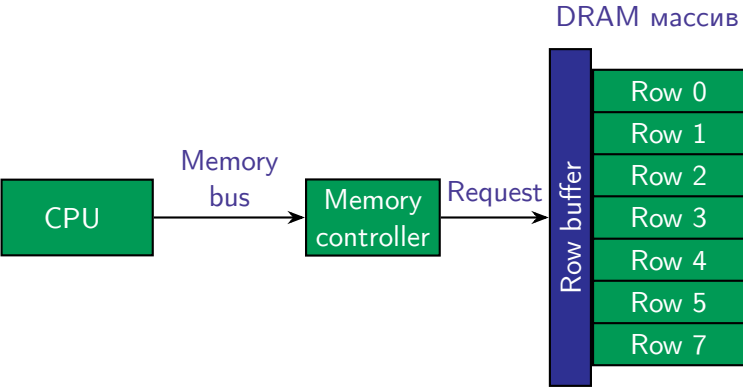
Theory

DRAM

How DRAM works

DRAM organization

How DRAM works



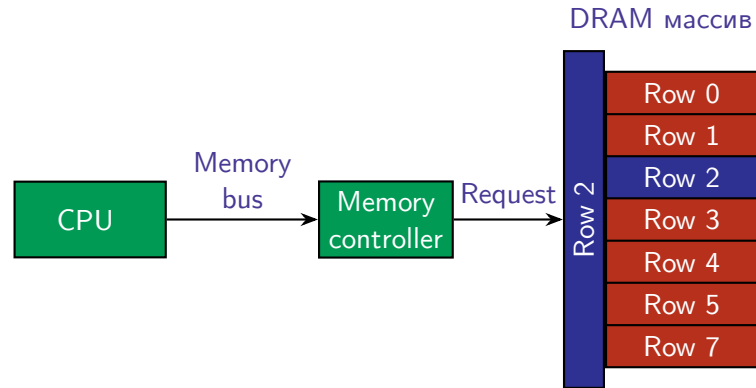
A very simple computer system, with a single DRAM array

DRAM (dynamic random-access memory, динамическая память с произвольным доступом).

DRAM имеет большую задержку в сравнении с кэш-памятью. Причина большой задержки не только в том, что ячейки DRAM имеют меньшую тактовую частоту, но и в том, как DRAM организован и подключён к процессору. Современные процессоры используют чипы-контроллеры памяти, которые позволяют передавать/получать данные в/с DRAM.

DRAM содержит: **строки (row)** и **колонки (columns)** (обычно 1024).

How DRAM works

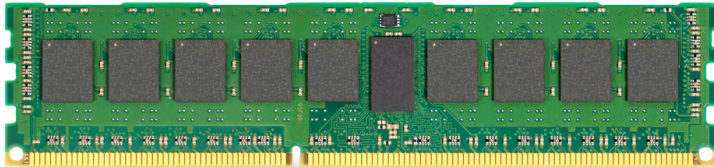


A very simple computer system, with a single DRAM array

Строка может быть **открытой** и **закрытой**. Если какая-либо строка открыта, то она вся сохраняется в **буфер строки (row buffer)**. Если текущая открытая строка содержит необходимые данные, то контроллер памяти просто берёт их из буфера строки. Эта ситуация очень похожа на кэш попадание и называется **попадание строки**. Если текущая открытая строка не содержит нужных данных, то это называется **промах строки**. Контроллер памяти в таком случае сначала **закрывает строку**, т. е. **записывает буфер строки обратно в DRAM**, а затем **открывает нужную строку** и считывает данные из буфера строки. Также как и промахи кэша, промахи строки вызывают повышение задержки.

DRAM organization

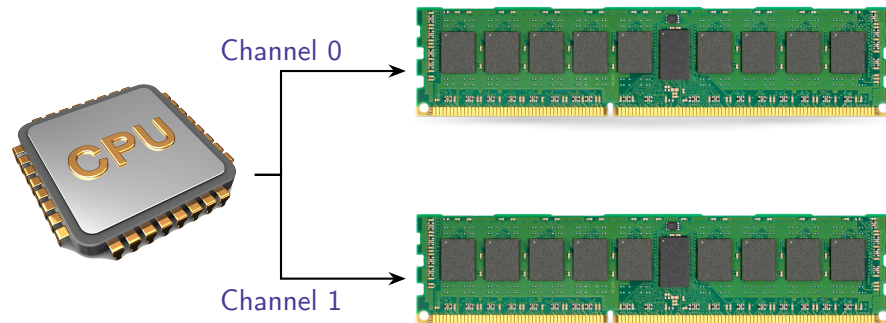
DIMM



Для повышения производительности работы с DRAM были использованы те же методы, что и в случае с кэшем. Современные компьютерные системы организуют DRAM в виде **каналов**, **DIMM (Dual Inline Memory Modules)**, **рангов** и **банков**.

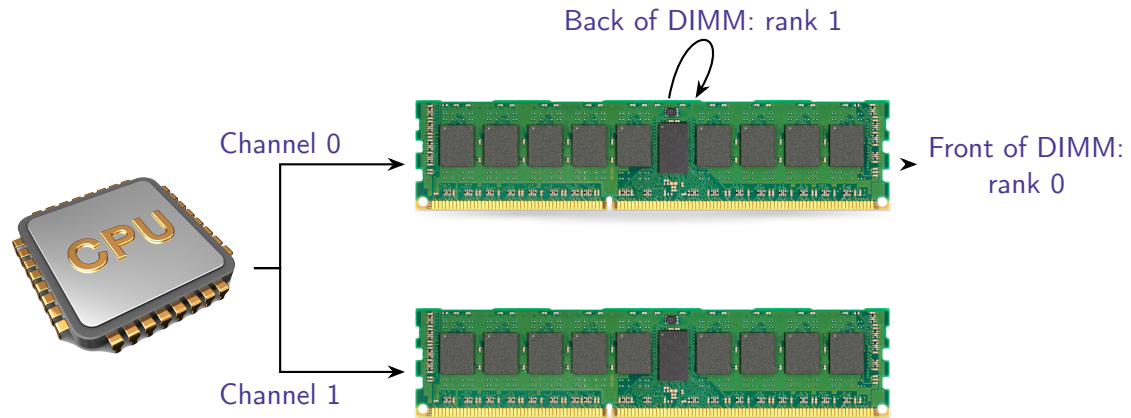
Количество памяти увеличивается в разы при перемножении количества банков на количество **DIMM модулей**.

DRAM organization



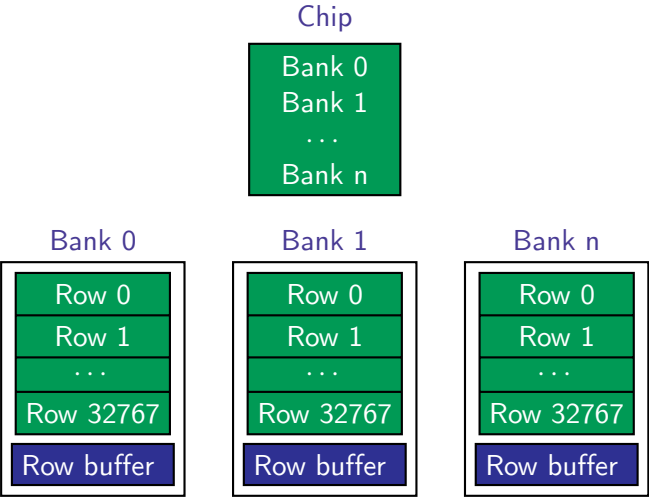
Для **увеличения ширины потока данных** и увеличения количества параллельных потоков современные компьютерные системы используют **несколько каналов**. Каждый канал управляется независимо и параллельно через DRAM шину.

DRAM organization



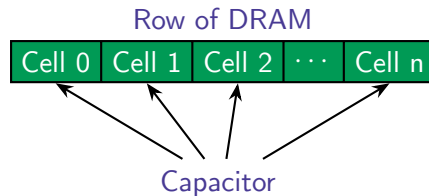
Современные модели DRAM имеют обычно от **1 до 4 рангов**, количество которых увеличивается при перемножении с количеством банков. Такого рода параллелизм позволяет **уменьшить промах строки**.

DRAM organization



Каждый из этих банков имеет своё собственное состояние и может иметь **независимо от других банков** открытую строку.
Современная DDR3 DRAM память имеет 8 банков, а DDR4 DRAM 16 банков (на ранг).

DRAM organization



В случае если два адреса **отображаются на пространство одного и того же** DIMM модуля, ранга и банка, то эти адреса физически **расположены рядом друг с другом** в DRAM. В таких случаях два адреса оказываются в банке с одним и тем же номером. В случае, если адреса отображаются на пространство банков с одним и тем же номером, но разных рангов или DIMM'ов, то они не расположены физически рядом.

Также как и с кэшем, существуют функции, которые производят отображение физического адреса на конкретные канал, DIMM, ранг и банк. Как работают эти функции, публично известно только у AMD, Intel не публиковала никакой информации. Однако, относительно недавно (2015, 2016) был произведён реверс-инжиниринг данных функций. Знание того, **как производится отображение физического адреса на физические структуры** даёт нам новый вектор атак — **атаки по сторонним каналам на DRAM**.

Agenda

Basic attacks

- Cache attacks
- Branch-prediction attacks
- TLB-based attacks
- Exception-based attacks
- DRAM-based attacks
- Covert channels

Agenda

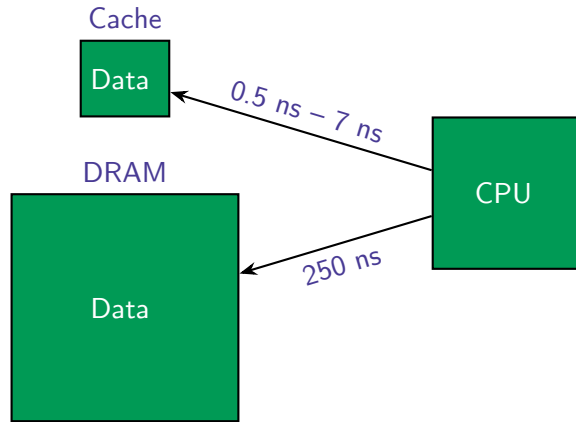
- Basic attacks

 - Cache attacks

 - Flush + Reload

 - Cache attacks

Cache attacks



Timing attack — attack exploiting differences in the execution time of an algorithm

Главное **предназначение кэш-памяти** — **нивелировать задержки** при работе с медленной главной физической памятью. При работе с данными через кэш задержки значительно уменьшаются. Kocher в 1996 году описал возможность использования разницы во времени при обращении к данным для совершения атаки, которая стала называться **атака на кэш по времени**.

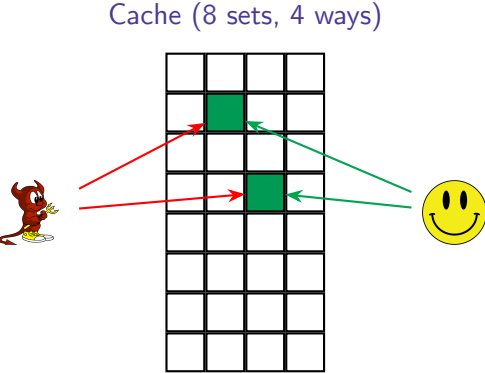
Flush + Reload

1. Map binary (e.g., shared object) into address space
2. Flush a cache line (code or data) from the cache
3. Schedule the victim's program
4. Check if corresponding cache line from step 2 has been loaded by the victim's program

Данная атака считается **наиболее эффективной** атакой на кэш. Целью данной атаки является не просто набор кэша, а **отдельная линия кэша**, более того, у атакующего существует возможность проверить закэширована ли та или иная область памяти. Атака Flush + Reload выполняется в три фазы.

Flush + Reload

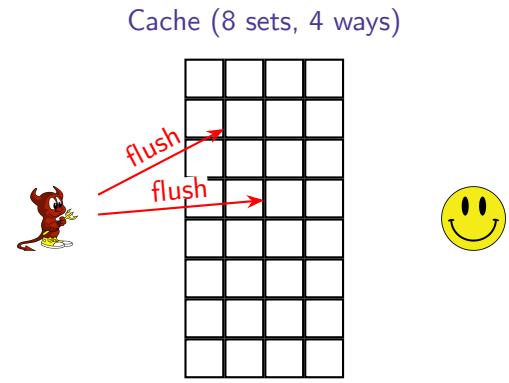
Map binary (e.g., shared object) into address space



Flush + Reload атака работает при условии **общей памяти**, ярким примером может служить общая библиотека, которую использует и атакующий и программа-жертва. По этой причине, в случае, если нет такого элемента, как общая память между атакующим и жертвой, придётся использовать схему атаки Prime + Probe.

Flush + Reload

Flush a cache line (code or data) from the cache



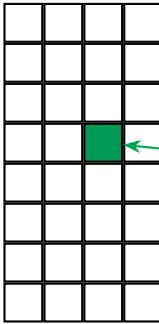
Во время первой фазы контролируемый участок памяти удаляется из структуры кэша (удаляется линия кэша с помощью инструкции **clflush** в случае с Intel).

Flush + Reload

Во второй фазе программа-шпион находится в режиме ожидания, **давая жертве время воспользоваться** участком памяти.

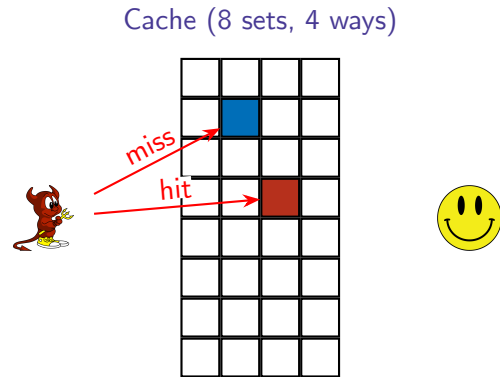
Schedule the victim's program

Cache (8 sets, 4 ways)



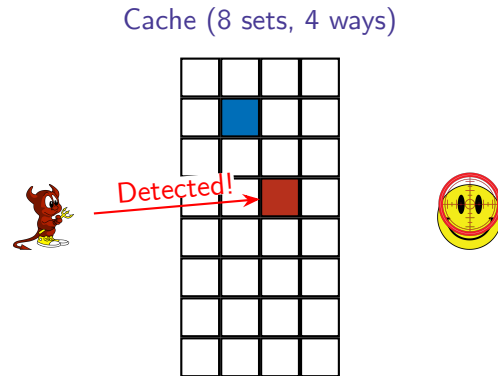
Flush + Reload

Check if corresponding cache line from step 2 has been loaded by the victim's program



Во время третьей фазы программа-шпион **перезагружает** участок памяти и **замеряет время** загрузки. Если во время второй фазы жертва **воспользовалась** участком памяти, то этот участок будет доступен из кэша и операция перезагрузки **пройдёт быстро**. С другой стороны, если линия кэша **осталась неиспользованной**, понадобится время на загрузку участка и операция перезагрузки пройдёт **значительно дольше**.

Flush + Reload



Такие методы атаки, как Flush + Reload и Flush + Flush (описан ниже), используют **непривилегированную** x86-инструкцию сброса clflush для удаления строки данных из кеш-памяти. Однако, за исключением процессоров ARMv8-A, **ARM-платформы не имеют непривилегированных инструкций сброса кеша**, и поэтому в 2016 году был предложен **косвенный метод вытеснения кеша**, с использованием эффекта Rowhammer.

Cache attacks

- ▶ Evict + Time
- ▶ Prime + Probe
- ▶ Prime + Abort
- ▶ Flush + Flush
- ▶ Evict + Reload
- ▶ AnC (ASLR \oplus Cache)
- ▶ and etc.

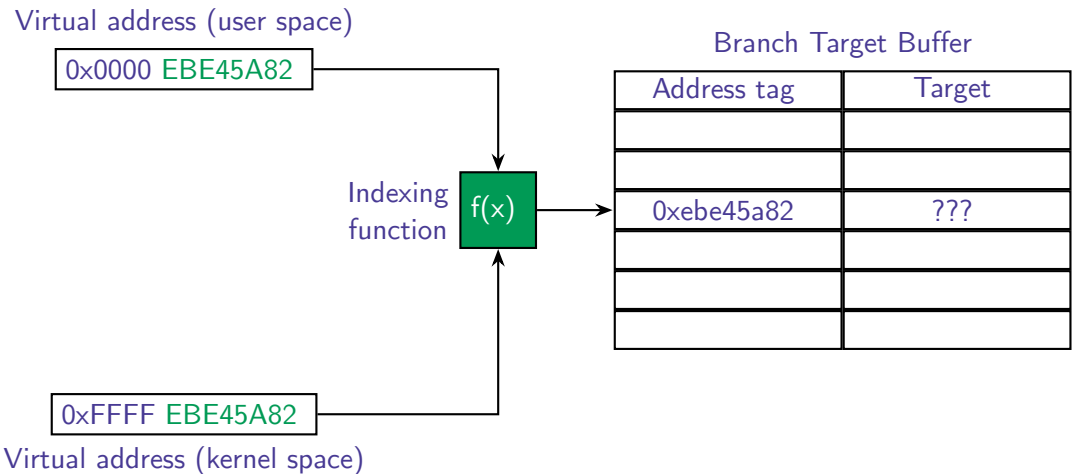
Существуют другие атаки на кэш, которые применяются в различных ситуациях, позволяют проводить атаку в стелс-режиме, направлены на различные уровни кэшей и т. д.

Agenda

- Basic attacks

 - Branch-prediction attacks

Branch-prediction attacks



Branch Target Buffer addressing scheme in Haswell processor

Буфер адресов перехода (branch target buffer, BTB) кэширует информацию о ранее выбранных переходах выполнения для быстрого угадывания будущих. Кэш использует индексацию на **базе виртуального адресного пространства**, таким образом **атакующему не обязательно знать физический адрес** для выполнения атаки. **Атакующий заполняет буфер адресов перехода** путём выполнения последовательности различных переходов. Если жертва-программа будет выполнять ту ветвь, которой не было в кэше, то она добавится туда, вытеснив тем самым существующую запись. **Атакующий может вычислить** какая ветвь была вытолкнута из буфера по сравнительно **большому времени выполнения** этой ветви. Пример атаки позволяет взломать KASLR из пользовательского процесса. Основывается на возникающих **коллизиях** в кэше branch target buffer. По времени выполнения своего кода атакующий имеет возможность **вычислить адрес перехода в ядерном пространстве** (значение берётся из BTB), тем самым вычислить смещение, полученное в результате KASLR. Возникает два типа коллизий:

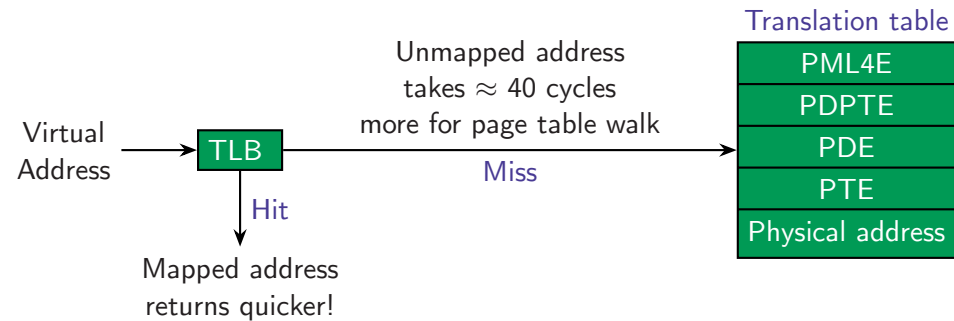
- 1. cross domain collisions — user и kernel space;
- 2. same domain collisions — разные user-space процессы, позволяет

Agenda

Basic attacks

- TLB-based attacks

TLB-based attacks



A translation lookaside buffer (TLB) is a memory cache that is used to reduce the time taken to access a user memory location

Трансляция адресов должна происходить очень быстро. С использованием таблиц трансляций, расположенных в памяти, данная операция быстро выполняться не может. По этой причине был введён кеш для трансляции адресов, который помогает уменьшить задержку при процессе трансляции — **буфер ассоциативной трансляции (translation lookaside buffer, TLB)**.

Атака впервые была представлена Ralf Hund в **2013 году**. Попытка чтения или записи памяти, к которой нет доступа по причине того, что данный участок памяти **используется ядром, занимает меньше времени**, если бы память не была размечена вовсе, т. к. используемые адреса памяти попадают в кэш независимо от уровня привилегий. Это позволяет узнать, какие адреса используются, и более того, узнать какие адреса **используются той или иной частью ядра**, т. е. данный вид атаки позволяет обойти технику рандомизации памяти в ядерном пространстве (kernel address-space-layout randomization, **KASLR**).

Agenda

Basic attacks

- Exception-based attacks

Exception-based attacks

- ▶ Scheduler interrupts
- ▶ Instruction aborts
- ▶ Page faults
- ▶ Behavioral differences (e.g, error code)

Данного рода атаки получают необходимую информацию из исключительных ситуаций, которые происходят при работе процессора. Обычными для процессора исключительными ситуациями являются: прерывание планировщика, прерывания инструкции, ошибка страницы памяти, а также поведенческие изменения, например, инструкции предоставляют пользователю код ошибки.

Во время возникновения исключительных ситуаций можно получить информацию о работе процессора либо **напрямую** (основываясь на поведении самого процессора), либо **косвенно (через измерения времени, при возникновении исключительных ситуаций)**.

Одним из ярких примеров атак подобного рода является **атака на систему дедупликации памяти**.

Agenda

Basic attacks

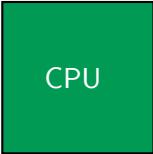
DRAM-based attacks

- Reading from DRAM

- Complex DRAM-based attacks

Reading from DRAM

Ещё раз о том, как работает **row buffer**.



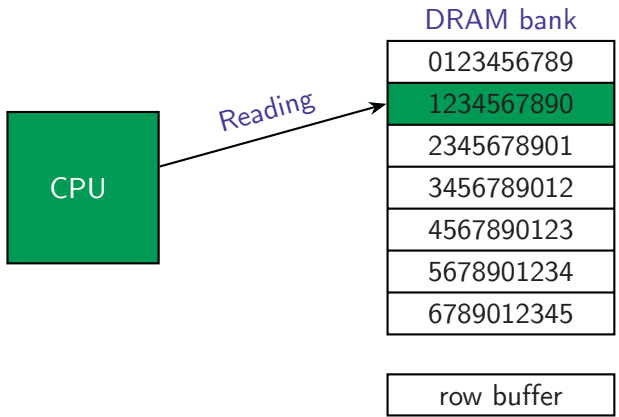
DRAM bank

0123456789
1234567890
2345678901
3456789012
4567890123
5678901234
6789012345

row buffer

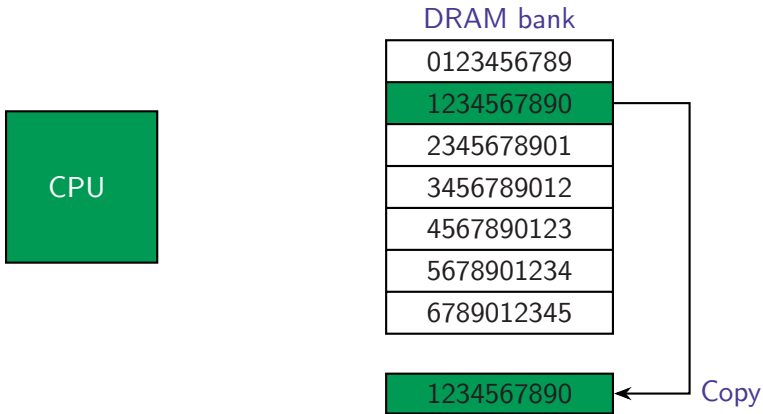
Reading from DRAM

Reading from DRAM

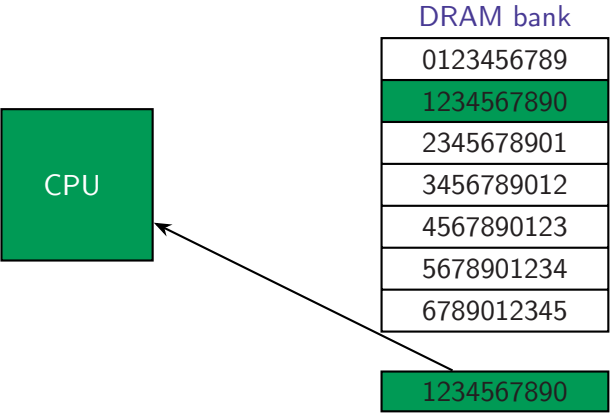


CPU reads row 1, row buffer empty

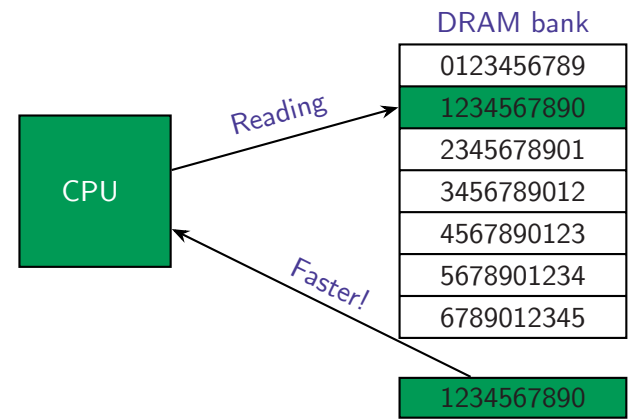
Reading from DRAM



Reading from DRAM



Reading from DRAM



CPU reads row 1, row buffer now full

При попадании строки чтение данных происходит быстрее, при промахе строки — медленнее, что похоже на поведение при работе с кэшем.

Complex DRAM-based attacks

- ▶ DRAMA
- ▶ Row hit (Flush + Reload)
- ▶ Row miss (Prime + Probe)
- ▶ and etc.

Существуют атаки непосредственно на DRAM (DRAM addressing, **DRAMA**).

Атака при **промахе строки** похожа на атаку на кэш **Prime + Probe**, атака при **попадании строки** сравнима с атакой **Flush + Reload**. Оба типа атаки работают и при **отсутствии разделяемой памяти**. DRAMA эксплуатирует буфер строки DRAM, как будто это **кэш с прямым отображением используемый банком**.

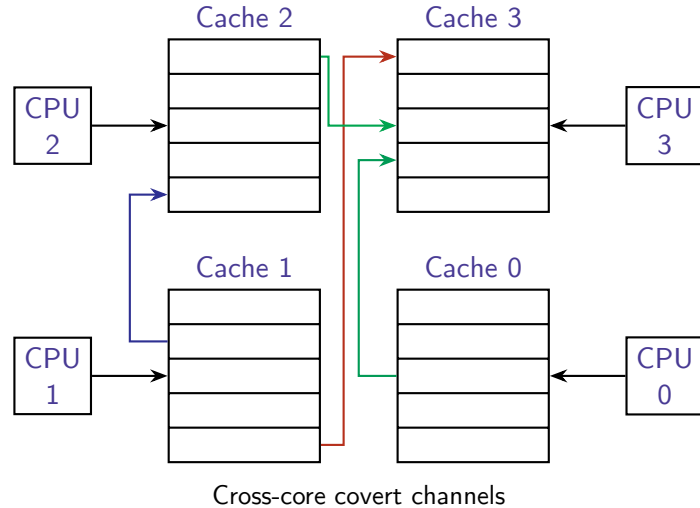
Во время подготовки атаки DRAMA применяли методы реверс-инжиниринга, основанные на подобном поведении DRAM, для того, чтобы **выявить местоположение и архитектуру банков**.

Agenda

Basic attacks

Covert channels

Covert channels



Современные облачные системы часто имеют **несколько процессоров**, установленных в материнскую плату с **мультисокетом**. Кэши процессоров содержатся в когерентном состоянии с помощью **межпроцессорных протоколов, обеспечивающих когерентность**. Однако, это обеспечивает эффект **разделяемой кэш линии**.

Кэш-атаки позволяют реализовать **высокопроизводительные межъядерные и межпроцессорные скрытые кэш-каналы** на современных смартфонах, используя Flush+Reload, Evict+Reload или Flush+Flush. Скрытый канал позволяет двум **непривилегированным приложениям** взаимодействовать друг с другом **без использования** каких-либо системных **механизмов передачи данных**. Благодаря этому можно вырваться из песочницы и обойти систему «ограниченных разрешений». В частности, на Android злоумышленник может использовать одно приложение, которое имеет **доступ к личным контактам** владельца устройства, для **отправки данных по скрытому каналу** другому приложению, имеющему доступ к **интернет**.

Covert channels

- ▶ Cache-based covert channels (shared libraries)
- ▶ Row miss attack (DRAM)
- ▶ Thermal covert channels
- ▶ Radio covert channels

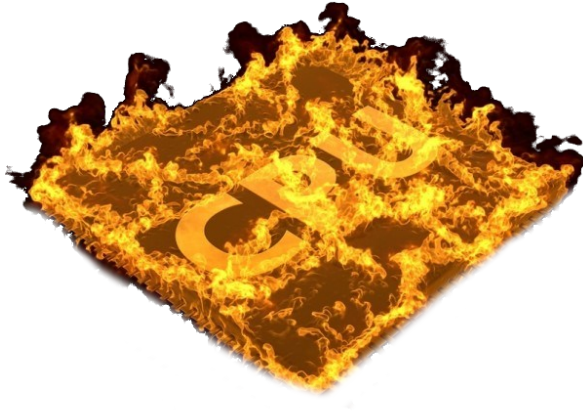
Wu в 2012 и в 2014 годах обнаружил **разницу во времени при доступе к памяти**, возникающую из-за **задержки в шине памяти**, что позволяет наладить скрытый канал передачи данных между **близко расположенными виртуальными машинами**. В облаке Amazon EC2 был налажен канал скоростью 13.5 КБ/с при 0.75 % ошибок. В 2016 Inci также обнаружил задержку в шине памяти, позволяющую наладить канал в облаках Microsoft Azure.

В 2017 году была представлена первая в своём роде реализация **скрытого канала, работающего по протоколу SSH**, с относительно высокой пропускной способностью (45 Кбит/с); эта реализация обеспечивает отказоустойчивые коммуникации между двумя виртуальными машинами даже в условиях экстремальной зашумлённости кеша.

Agenda

- Software-based Microarchitectural Fault Attacks
- Rowhammer

Software-based Microarchitectural Fault Attacks



Software-based microarchitectural fault attacks do not require physical access, but instead only some form of code execution on the target system

Обычно предполагается, что безопасность системы и программная безопасность опирается на безопасность аппаратную и на то, что в аппаратном средстве нет ошибок. Однако, это не так, и **аппаратные средства не идеальны**, особенно часто дефекты встречаются в случаях, когда работа производится **за границами спецификации**. Уникальность атак, основанных на дефектах микроархитектуры в том, что они используют эффекты, вызванные микроархитектурными элементами или операциями, которые реализованы на микроархитектурном уровне. В атаках, которые основаны на использовании программного обеспечения все микроархитектурные эффекты и операции вызываются из программного обеспечения.

Agenda

Software-based Microarchitectural Fault Attacks

Rowhammer

- Rowhammer. Exploitation primitives

- Variations of Rowhammer

Rowhammer. Exploitation primitives

- ▶ Fast uncached memory access
- ▶ Physical memory massaging
- ▶ Physical memory addressing

Быстрый некэшируемый доступ к памяти. Данный примитив не так-то легко получить, т. к. **контроллер памяти** на CPU может **недостаточно быстро обрабатывать запросы** на чтение памяти. Нерасторопность контроллера, как правило нивелируется загрузкой данных в кэши. **Использование кэшей также необходимо предотвратить** для того, чтобы было постоянное обращение к DRAM.

Определение местонахождения уязвимых строк. Кроме того, требуется, чтобы жертва использовала нужные атакующему строки DRAM для хранения важной информации.

Знание функций адресации физической памяти. Требуется для определения методов трансляции виртуальных адресов в физические и трансляции физических адресов на аппаратное средство.

Variations of Rowhammer

- ▶ Flip Feng Shui — targeted Rowhammer
- ▶ Throwhammer — remote Rowhammer
- ▶ Nethammer — better remote Rowhammer
- ▶ Drammer, RAMpage — exploitation ARM-based hardware
- ▶ Glitch — better exploitation ARM-based hardware

Очень много НО.

- дедупликация памяти, **разделяемая память**.
- удалённый прямой доступ к памяти (**remote direct memory access, RDMA**)
- **Intel CAT**, **некэшируемая память**, инструкции очистки кэша в сетевых драйверах.
- ARM — медленная запись в память, инструкция очистки кэша — привилегированная, **Android ION аллокатор памяти**.
- **GPU на мобильных устройствах**, т. к. CPU и GPU используют одну оперативную память, используется WebGL.

Agenda

Meltdown & Spectre

- Derived attacks and not only

- Abstract example of exploitation

Agenda

Meltdown & Spectre

Derived attacks and not only

Derived attacks and not only

Spectre-NG

- ▶ MeltdownPrime & SpectrePrime
- ▶ SgxPectre
- ▶ SMM Speculative Execution Attacks
- ▶ BranchScope
- ▶ LazyFP
- ▶ ...

OpenBSD — LazyFP, отключение Hyper-Threading (TLBleed).

- **другой способ атаки на кэш** + задействование двух ядер CPU — утечка данных работы **протокола согласования содержимого кэша для разных ядер CPU** (Invalidation-Based Coherence Protocol)
- позволяет обойти средства изоляции кода и данных, предоставляемые технологией **Intel SGX** (Software Guard Extensions)
- SMM — **режим системного управления** — запускается специальная программа в привилегированном режиме.
- **variant 2** + вместо BTB — **направления ветвления для спекулятивного перехода (directional branch predictor)** и манипулирует содержимым **таблицы с историей шаблонов переходов (PHT, Pattern History Table)**.
- **variant 3a** — «ленивое» режим **переключения контекста FPU**, при котором реальное восстановление состояния регистров производится **не сразу** после переключения контекста, а только при выполнении первой инструкции

Derived attacks and not only

- ▶ Spectre 1.1, 1.2 (Speculative Buffer Overflows)
- ▶ SpectreRSB
- ▶ NetSpectre
- ▶ L1TF (Foreshadow)
- ▶ and etc.

TotalMeltdown? and other patches...

- Spectre 1 + Buffer Overflows, спекулятивная запись.
- Засорение RSB (return stack buffer), спекулятивное выполнение после return.
- Производится **поиск leak и transmit гаджетов**, низкая производительность (1-3 байта за 3-8 часов).
- При доступе к памяти по виртуальному адресу, приводящему к исключению из-за отсутствия флага Present в таблице страниц памяти, процессоры Intel спекулятивно рассчитывают физический адрес и загружают данные, если они имеются в TLB? кэше.

Agenda

Meltdown & Spectre

Abstract example of exploitation

Abstract example of exploitation

The four components of speculation techniques

1. Speculation primitive

А так ли всё легко и просто? Представим, что мы хотим совершить атаку.

Рассмотрим на примере Spectre v2 (variant 2).

Процессор **должен поддерживать** возможность спекулятивного выполнения.

Требуется **найти место** в коде, где может происходить спекулятивное выполнение по желанию атакующего.

Abstract example of exploitation

The four components of speculation techniques

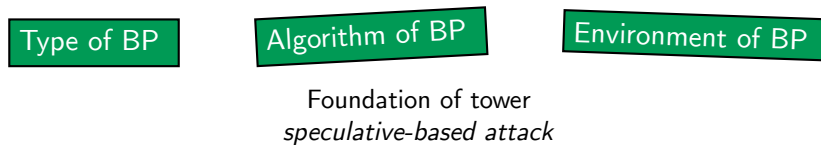
1. Speculation primitive

- ▶ Bypass out of bounds checks
- ▶ Training of branch predictor
- ▶ Speculatively read an earlier value of the data
- ▶ Pending exceptions
- ▶ Exploit branch history table
- ▶ Exploit the Return Stack Buffer
- ▶ Speculatively write to register (buffer overflow)

Microarchitecture — ?

На данный момент существует несколько техник, позволяющих производить спекулятивное выполнение по желанию атакующего.
Для того, чтобы использовать те или иные техники **требуется досконально знать микроархитектуру процессора.**

Abstract example of exploitation



Выберем тренировку предсказателя переходов. Мы столкнёмся:

- требуется знать **вид предсказателя переходов**
- требуется знать **алгоритм работы предсказателя переходов**
- для **разных процессоров** — **разные условия**, например, в i7 два буфера предсказателя переходов.

В whitepaper и в PoC даны **примеры для конкретных процессоров**.

Abstract example of exploitation

Требуется найти гаджеты, которые позволят создать достаточно **длительное по времени выполнения окно** для спекулятивного выполнения.

The four components of speculation techniques

1. Speculation primitive
2. **Windowing gadget**

Abstract example of exploitation

Не **езде** есть такие **цепочки** в окружении. В некоторых случаях **требуется** знать, какие **данные** **сейчас** в **кэше**.

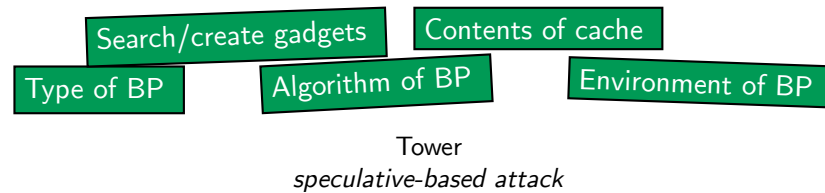
The four components of speculation techniques

1. Speculation primitive

2. **Windowing gadget**

- ▶ Non-cached loads
- ▶ Dependency chain of loads
- ▶ Dependency chain of integer ALU operations

Abstract example of exploitation



Выберем гаджеты для загрузки некешированных данных. Мы столкнёмся:

- в случае JIT — создание гаджетов, в других случаях гаджеты следует искать,
- что хранится в кэше на данный момент.

В whitepaper и в PoC даны **примеры для конкретных процессоров**.

Abstract example of exploitation

Требуется найти гаджеты, которые позволят **считать необходимую закрытую информацию** в ходе спекулятивного выполнения.

The four components of speculation techniques

1. Speculation primitive
2. Windowing gadget
3. Disclosure gadget

Abstract example of exploitation

The four components of speculation techniques

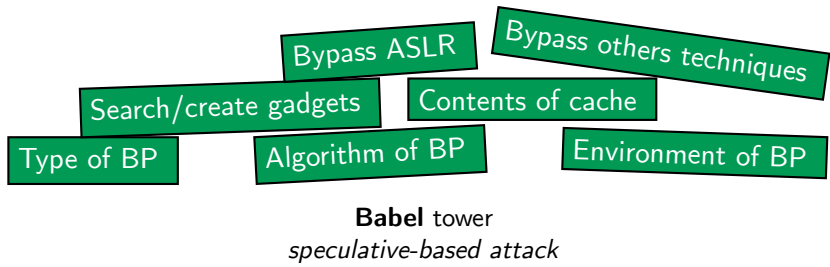
1. Speculation primitive
2. Windowing gadget
3. Disclosure gadget

- ▶ ASLR
- ▶ CFI
- ▶ SMAP
- ▶ DEP/NX
- ▶ retpoline
- ▶ and others.

Для применения необходимых гаджетов требуется обойти некоторые системы защиты.

Abstract example of exploitation

В whitepaper и в PoC все защиты отключены.



Abstract example of exploitation

Требуется возможность **считать полученную** через сторонний канал информацию или **удостовериться**, что она была считана.

The four components of speculation techniques

1. Speculation primitive
2. Windowing gadget
3. Disclosure gadget
4. Disclosure primitive

Abstract example of exploitation

The four components of speculation techniques

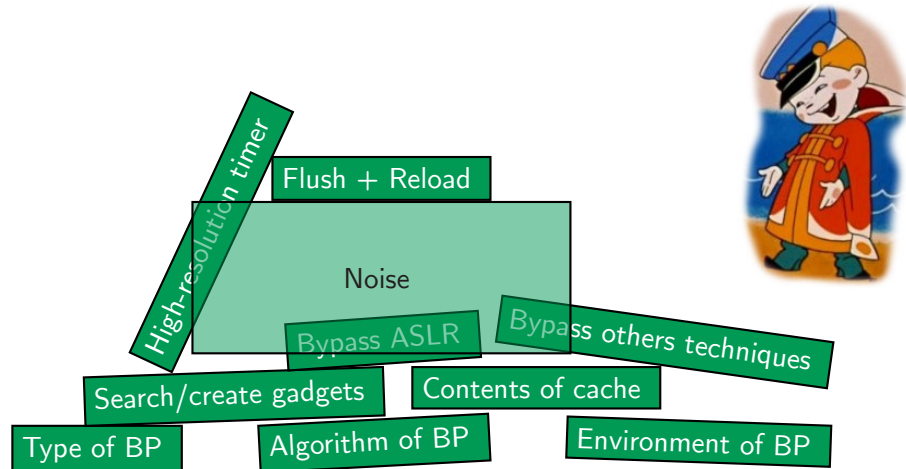
1. Speculation primitive
2. Windowing gadget
3. Disclosure gadget
4. Disclosure primitive

- ▶ Architecture of cache
- ▶ Replacement policies
- ▶ Exclusive and inclusive
- ▶ Type of cache attack
- ▶ Noise
- ▶ High-resolution timer
- ▶ and etc.

- Требуется знать, какой тип кэша используется.
- Требуется знать, какие данные сейчас хранятся в кэше, как выталкивать.
- На какой кэш будет направлена атака.
- Возможность проведения той или иной атаки.
- Возможность многократного повторения атаки.
- Для измерения времени требуются высокоточные счётчики.

Abstract example of exploitation

Что? Ещё одна атака?



Babel tower
speculative-based attack



Agenda

Summary

Summary

- ▶ Software-based microarchitectural attacks become **a very popular**

Всё больше исследований проводится в этой области, всё больше обычных обывателей интересуются данной проблемой. Уязвимости в ПО **всё сложнее эксплуатировать, переходим к железу.**

Уязвимости находят, **прочитав и разобравшись в спецификации архитектуры**, процессора. Обратную разработку производят с помощью базовых атак по сторонним каналам.

Summary

Не смотря на все многочисленные плюсы (для атакующего), **разработка эксплоита** для широкого спектра программного и аппаратного ПО — **весьма затруднительна**.

- ▶ Software-based microarchitectural attacks become **a very popular**
- ▶ Requires **a lot of resources** to develop working exploit

Summary

- ▶ Software-based microarchitectural attacks become **a very popular**
- ▶ Requires **a lot of resources** to develop working exploit
- ▶ Microarchitectural attacks may be **automated**

Представлено множество работ и инструментов, позволяющих провести атаку практически на любую популярную архитектуру. **Создаются эксплоит-паки**, содержащие атаки на микроархитектуру.

Summary

- ▶ Software-based microarchitectural attacks become **a very popular**
- ▶ Requires **a lot of resources** to develop working exploit
- ▶ Microarchitectural attacks may be **automated**
- ▶ Many attacks have **not yet been published**

Описание атак, использующих спекулятивное выполнение, ещё **не до конца опубликованы**.

Множество возможных **изъянов микроархитектуры не найдены**.

Summary

- ▶ Software-based microarchitectural attacks become **a very popular**
- ▶ Requires **a lot of resources** to develop working exploit
- ▶ Microarchitectural attacks may be **automated**
- ▶ Many attacks have **not yet been published**
- ▶ Countermeasures come with a **performance impact**

Для исправления сложившейся ситуации требуются **фундаментальные изменения** в ходе работы процессора.

Исправления, **разработанные на уровне ОС**, требуют **детального изучения уязвимости** и алгоритма противодействия, к тому же **не всегда возможно предотвратить** эксплуатацию на уровне ОС, а если и удаётся, то в большинстве случаев приносят **в жертву процессы оптимизации**.


Questions?


References I


 [Daniel Gruss](#)
Software-based Microarchitectural Attacks.

 [Moritz Lipp, Daniel Gruss](#)
ARMageddon: Cache Attacks on Mobile Devices.


 [D. Page](#)
MASCAB: a Micro-Architectural Side-Channel Attack Bibliography.


 [Pessl P., Gruss D. and others](#)
DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks.


 [Bos H., Fratantonio Y. and others](#)
Drammer: Determenistic Rowhammer Attacks on Mobile Platforms.


 [Microsoft](#)
Mitigating speculative execution side channel hardware vulnerabilities.


References II

 [Google Project Zero](#)
Reading privileged memory with a side-channel.

 [Daniel Gruss, Moritz Lipp](#)
KASLR is Dead: Long Live KASLR.

 [Daniel Gruss, Clémentine Maurice and others](#)
Flush+Flush: A Fast and Stealthy Cache Attack.

 [Fangfei Liu, Yuval Yarom and others](#)
Last-Level Cache Side-Channel Attacks are Practical.

 [Caroline Trippel, Daniel Lustig, Margaret Martonosi](#)
MeltdownPrime and SpectrePrime: Automatically-Synthesized Attacks Exploiting Invalidation-Based Coherence Protocols.

References III



[Michael Schwarz, Clémentine Maurice, Daniel Gruss, Stefan Mangard](#)

Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript.



[Moritz Lipp, Misiker Tadesse Aga and others](#)

Nethammer: Inducing Rowhammer Faults through Network Requests.



[Andrei Tatar, Radhesh Krishnan and others](#)

Throwhammer: Rowhammer Attacks over the Network and Defenses.



[Giovanni Camurati, Sebastian Poeplau and others](#)

Screaming Channels: When Electromagnetic Side Channels Meet Radio Transceivers.



[Julian Stecklina, Thomas Prescher](#)

LazyFP: Leaking FPU Register State using Microarchitectural Side-Channels.



[Mordechai Guri, Assaf Kachlon and others](#)

GSMem: Data Exfiltration from Air-Gapped Computers over GSM Frequencies.

References IV

-  [Dean Sullivan, Orlando Arias, Travis Meade, Yier Jin](#)
Microarchitectural Minefields: 4K-Aliasing Covert Channel and Multi-Tenant Detection in IaaS Clouds.
-  [Gras B., Razavi K., Bosman E., Bos H., Giuffrida C.](#)
ASLR on the Line: Practical Cache Attacks on the MMU.
-  [van Schaik S., Giuffrida C., Bos H., Razavi K.](#)
Malicious Management Unit: Why Stopping Cache Attacks in Software is Harder Than You Think.
-  [Daniel Gruss, Anders Fogh and others](#)
Prefetch Side-Channel Attacks: Bypassing SMAP and Kernel ASLR.
-  [Esmaeil Mohammadian Koruyeh, Khaled N. Khasawneh and others](#)
Spectre Returns! Speculation Attacks using the Return Stack Buffer.

References V

 [Giorgi Maisuradze, Christian Rossow](#)

ret2spec: Speculative Execution Using Return Stack Buffers.

 [Guoxing Chen, Sanchuan Chen and others](#)

SgxPectre Attacks: Leaking Enclave Secrets via Speculative Execution.

 [Moritz Lipp, Michael Schwarz and others](#)

Meltdown.

 [Paul Kocher, Daniel Genkin and others](#)

Spectre Attacks: Exploiting Speculative Execution.

 [ARM Whitepaper](#)

Cache Speculation Side-channels.

 [Michael Schwarz, Martin Schwarzl, Moritz Lipp, Daniel Gruss](#)


NetSpectre: Read Arbitrary Memory over Network.


References VI


 [Sophia D'Antoine](#)
Out-of-Order Execution and Its Applications.

 [Vladimir Kiriansky, Carl Waldspurger](#)
Speculative Buffer Overflows: Attacks and Defenses.

 [Gras B., Razavi K., Bos H., Giuffrida C.](#)
Translation Leak-aside Buffer: Defeating Cache Side-channel Protections with TLB Attacks.

 [Craig Disselkoen, David Kohlbrenner, Leo Porter, Dean Tullsen](#)
Prime+Abort: A Timer-Free High-Precision L3 Cache Attack using Intel TSX.

 [Moritz Lipp, Michael Schwarz](#)
Meltdown & Spectre Side-channels considered hARMful.

 [Jon Masters](#)
Exploiting modern microarchitectures: Meltdown, Spectre, and other attacks.

References VII



Moritz Lipp

Cache attacks on ARM.



Evtyushkin D., Ponomarev D., Abu-Ghazaleh N.

Jump over ASLR: attacking branch predictors to bypass ASLR.