Spectre и Meltdown

Начнем с истории, в нюне 2017 года, производителям процессоров (intel, amd, arm), прислали информацию о нахождении в архитектуре процессоров, фундаментальных дыр. Начинаются закрытые обсуждения в компаниях по исправлению ситуации.

Первая публикация данного инцидента в сми была сделана в январе следующего года. Производители ОС выпускают патч, который препятствует воспроизведению данных уязвимостей.

После этого эксперты гугл пишут статью где они успешно смогли реализовать данные уязвимости, так благодаря spectre мы можем читать виртуальную память других процессов, а благодаря Meltdown – появляется возможность доступа чтения привилегированной памяти ядра.

-----------------------------------------------------------------------------------------------------------------------------------------------------------

Обе уязвимости схожи на логическом уровне и предоставляют возможность проведения атаки по стороннему каналу – кэш процессора. Meltdown использует задержку в обработке исключений. Spectre – обманывает блоки предсказания. Мы вынуждаем процессор в свой кэш данные по определенному адресу, который нам не доступен. Подходы по разному кэшируют данные

-----------------------------------------------------------------------------------------------------------------------------------------------------------

Последовательное исполнение команд

Изначально процессоры выполняли команды последовательно – Процессор с последовательным выполнением команд начинает выполнение очередного цикла команды только после того, как будет закончен предыдущий, то есть в каждый момент времени выполняется только одна команда., однако такой подход имеет ряд минусов

• допустим у нас есть две команды первая обращается к памяти и из за этого очень долго находится на процессоре, а вторая маленькая – все это время ждет

• Ступор конвейера – команда ждет результат предыдущей

Спекулятивное выполнение команд

Основан на том, чтобы не ждать долгих расчетов предыдущих команд, а выполнять то что можно выполнить. Так после декодера инструкций появился буфер переупорядочивания микроопераций. Который сам решает какая команда будет исполнена следующей

-----------------------------------------------------------------------------------------------------------------------------------------------------------

Суперскалярность

Производители процессоров поняли, что могу добиться большей скорости выполнения сделав процессоры суперскалярными. Суперскалярный – у нас есть параллелизм в конвейере. Современные процессоры могут одновременно делать 4 целочисленные инструкции и 2 с плавающей точкой в одном такте.

Поэтому процессор может выполнять команды в таком порядке, так как между этими переменными нет связи.

-----------------------------------------------------------------------------------------------------------------------------------------------------------

Таблицы страниц

Напомним как работают программы в ОС. Каждая программа работает в своем виртуальном адресном пространстве для реализации технологии виртуальной памяти в ОС есть таблица страниц - содержат сопоставления между виртуальной памятью, которую использует процессор и физической памятью, которую использует диспетчер памяти.

Код ядра современной ОС отображается в виртуальное адресное пространство пользовательского процесса. Это было сделано для повышения производительности. В нормальных условиях память ядра не может быть прочитана из пользовательского пространства, будет создано исключение.

-----------------------------------------------------------------------------------------------------------------------------------------------------------

Meltdown

Из за того что код ядра находится в виртуальном адресном пространстве, то это породило уязвимость Meltdown.

* Цель – прочитать какое то значение про какому то адресу в ядре. Которое недоступно в обычном исполнении.
* Ограничение – только свое адресное пространство.
* Из за спекулятивного выполнения команд процессором, проверка проходит на финальном этапе

-----------------------------------------------------------------------------------------------------------------------------------------------------------

Атака

Итак, перед вами код, в котором идет какое-то вычисление, потом условный оператор, в котором также есть код. Это очень похоже на то, что я вам показывал вам в примере, так что думаю вы уже догадываетесь как все будет происходить. CPU может подумать что возможно v действительно равно 0 и сдвинуть команды.

Обратите внимание что этот код пытается считывать информацию из памяти ядра. Произойдет ошибка, которая должна завершить процесс.

Однако, что на самом деле произойдет, так это то спекуляция, мы можем запустить все это здесь, потому что у нас есть запасной конвейер. Хорошо, тогда все что происходит в этих командах может и не должно происходить, т.к. мы не знаем сработает условие или нет. При считывании информации в адресе ядра установлен бит защиты и вопрос в том, где мы проверяем этот бит, даже если мы сразу его проверяем мы не можем выкинуть ошибку так как мы не знаем придем мы сюда или нет. В целях производительности мы это выполним. По-хорошему необходимо было ставить проверку на считывание данных, однако этого не делали в целя производительности. Мы запускаем проверку позже.

Когда к нам придет v, который не равен нулю то все расчеты должны быть стерты и эти операции не должны быть раскрыты.

Итак у нас есть какой то секрет в w, который мы не должны были видеть, но мы действуем злонамеренно и хотим его увидеть.

В начали мы делаем логическое и с 1 последнего бита числа. Поэтому ответом будет 1 или 0.

Затем мы умножаем это на 4096

Затем мы загружаем в пользовательскую память или 0 или 4096

Сделайте так, чтобы user\_mem[0...4096] не находился в кэше.

Запустите код (либо с v != 0, либо обработайте ошибку). Затем проверьте, сколько времени требуется для чтения user\_mem[0] и user\_mem[4096]. Один из них будет кэширован из-за спекулятивного выполнения. Какой из них показывает наименее значимый бит kernel\_mem[addr].

Либо нулевая пользовательская память находится к кэше процессора или 4096. 4096 размер страницы кэша гарантирует что туда не попадет другой мусор. Хотя данную уязвимость можно реализовать и без if.

-----------------------------------------------------------------------------------------------------------------------------------------------------------

Производители ОС выпустили патч, который удаляет отображение ядра из пользовательского адресного пространства. Поскольку ядро боль не отображается его память нельзя прочитать с помощью meltdown. Однако это сильно снизило производительность системы так как системные вызовы стали обрабатываться намного дольше. Многие БД зафиксировали снижение производительности на 30%.

-----------------------------------------------------------------------------------------------------------------------------------------------------------

Spectre

Пытается обмануть предсказатель ветвлений. Предсказатель ветвлений - Устройство, входящее в состав микропроцессоров, имеющих конвейерную архитектуру, предсказывающее, будет ли выполнен условный переход в исполняемой программе. Предсказание ветвлений позволяет сократить время простоя конвейера за счёт предварительной загрузки и исполнения инструкций, которые должны выполниться после выполнения инструкции условного перехода. Первая уязвимость семейства Spectre направлена на то, чтобы обмануть предсказатель переходов. Вторая – предсказатель адреса.

-----------------------------------------------------------------------------------------------------------------------------------------------------------

Атака Spectre

По сути это ещё одна проверка защиты, которую мы обходим. Это не проверка программного обеспечения а аппаратного.

-----------------------------------------------------------------------------------------------------------------------------------------------------------

Методы защиты от spectre

Производители процессов решили добавить новый моделезависимый регистр, который ограничивает выполнение косвенных переходов, ограничивает обучение предсказателя переходов, и умеет его очищать. Однако тем самым это сделало процессоры более медленными.