Гужанков Евгений Геннадьевич
студент
Уваровский Владислав Вадимович
студент
Воробьев Данил Сергеевич
студент
ФГБОУ ВО «Омский государственный
технический университет»
г. Омск, Омская область

АЛГОРИТМ АТАКИ ЧЕРЕЗ МИКРОАРХИТЕКТУРНУЮ УЯЗВИМОСТЬ ПРОЦЕССОРОВ MELTDOWN

Аннотация*: в данной статье будет приведён возможный алгоритм атаки с использованием микроархитектурной уязвимости Meltdwon. В работе выделены этапы, из которых состоит Meltdown.*

Ключевые слова: *микропроцессор, информационная безопасность, спекулятивное исполнение, алгоритм Томасуло, Meltdown.*

Подготовка к атаке начинается с того, что злоумышленник может запустить любой код, имея привилегии пользователя, не имея при этом физического доступа к атакуемой машине.

Предположим, что система полностью защищена современными программными средствами защиты, такими как ASLR и KASLR, а также функциями ЦПУ, такими как SMAP, SMEP, NX и PXN. То есть, не существует уязвимостей программного обеспечения. Целью атаки является получение пользовательских данных, например, паролей и закрытых ключей, или другой ценной информации.

Content is licensed under the Creative Commons Attribution 4.0 license (CC-BY 4.0)

```
1 ; rcx = адрес ядра
2 ; rbx = определение таблиц
3 retry:
4 mov al, byte [rcx]
5 shl rax, 0xc
6 jz retry
7 mov rbx, qword [rbx + rax]
```

Рис. 1. Основная последовательность команд Meltdown

Meltdown базируется на исполнении временных команд и передаче состояния микроархитектуры. В первую очередь, атакующий заставляет процессор выполнить последовательность временных команд, использующую значение секрета, хранящееся в физической памяти.Временная последовательность команд действует как передатчик скрытого канала, по которому и происходит утечка данных.

Meltdown состоит из 3 этапов:

Шаг 1. Содержимое выбранной памяти, которое недоступно для атакующего, загружается в регистр.

Шаг 2. Временная команда обращается к строке кэша, основанной на секретном содержании регистра.

Шаг 3. Нападающий использует Flush + Reload для получения доступа к кэшу и, следовательно, секрету, хранящемуся в выбранной памяти.

Повторяя эти шаги в разных отделах памяти, атакующий может добраться до дампа памяти ядра. На рисунке 1 показана реализация последовательности временных команд и передающей части скрытого канала, использующих 86-разрядные сборки команд. Заметим, что эта часть атаки также может быть полностью реализована на языках более высокого уровня, таких как C.

Шаг 1: *Чтение секрета*. Для загрузки данных из памяти в регистр, они отправляются в виртуальный адрес. Параллельно с переводом виртуального адреса на физический, ЦПУ также проверяет, доступен ли этот виртуальный адрес пользователю или только ядру. Данная аппаратная изоляция через разрешение бита считается безопасной. Следовательно, современные операционные системы всегда отображают всю память ядра в виртуальном адресном пространстве каждого

Содержимое доступно по лицензии Creative Commons Attribution 4.0 license (CC-BY 4.0)

пользовательского процесса. Впоследствии, все открывшиеся при переходе адреса ядра ведут к физическим адресам, а ЦПУ получает доступ к содержимому этих адресов. Единственное различие в доступе к адресному пространству пользователя заключается в том, что ЦПУ должен создать исключение для получения доступа. Следовательно, из пользовательского пространства невозможно прочесть содержимое этого адреса.

В строке 4 рисунка 1 мы загружаем значение байта, расположенного по адресу выбранного ядра, сохраненному в регистре RCX, в младшее значение байта регистра RAX, представленного от AL. Команда MOV извлекается ядром, декодируется в μOP, выделается и отправляется в буфер переупорядочения. В нем архитектурные регистры (например, RAX и RCX) сопоставляются с базовыми физическими регистрами, осуществляющими внеочередное исполнение. При попытке использования перехода таким же образом, последовательность команд (строки 5–7) уже декодируются и выделяются как μOP. μOP далее отправляются в блок резервирования, где хранятся μOP, во время ожидания их исполнения соответствующим блоком. Выполнение μOP может быть отложено, если исполнительные устройства уже используются, или значения операнда еще не рассчитаны. Когда адрес ядра загружается в строку 4, скорее всего, ЦПУ уже выпустил последовательность команд как часть внеочередного исполнения, и что их соответствующие μOP ждут на станции резервирования для получения содержимого адреса ядра. Как только извлеченные данные отображаются на общей шине данных, μOP могут начать выполнение. Когда μOP заканчивают свое исполнение, они устраняются, и, таким образом, их результаты соответствуют на архитектурном уровне. Во время устранения любые прерывания и исключения, возникшие во время выполнения команды, обрабатываются. Таким образом, если команда MOV, загружаемая в адрес ядра, устраняется, исключение регистрируется и переход исчезает, чтобы устранить все результаты последующих команд, исполненных вне очереди. Предварительная выборка адреса ядра может усилить атаку на некоторые системы.

Content is licensed under the Creative Commons Attribution 4.0 license (CC-BY 4.0)

Шаг 2: *Передача секрета*. Последовательность команд из шага 1, исполненного вне очереди, должна выбираться таким образом, чтобы она являлась временной последовательностью команд. Если эта последовательность временных команд начнёт выполняться до того, как инструкция MOV удалится (т. е. будет исключена), а последовательность временных команд выполняет вычисления секрета, то ее можно использовать для передачи секретной информации злоумышленнику. Последовательность временных команд должна кодировать секрет в микроархитектурный кэш так, чтобы атака через кэш была способна построить быстрый и незаметный канал утечки.

При распределении массива в памяти никакая часть этого массива не кэшируется. Для передачи секрета последовательность временных команд содержит в себе косвенную память доступа к адресу, которая рассчитывается на основе секретного (недоступного) значения. В строке 5 рисунка 2 секретное значение шага 1 умножается на размер страницы, т. е.4 КБ. Умножение секрета гарантирует, что доступ к массиву будет иметь большее пространственное расстояние относительно других. Это предотвращает его от загрузки памяти по стороннему каналу в кэш. Один байт считывается сразу, поэтому массив составляет 256 × 4096 байт, то есть 4 КБ на страницу. При внеочередном исполнении идет смещение к значению регистра «0». По этой причине была введена логика повтора в последовательности временных команд. В случае, если считывается «0», секрет читается снова (шаг 1). В строке 7 умноженный секрет добавляется к базовому адресу массива, образуя целевой адрес скрытого канала. Этот адрес считывается для оптимизации соответствующей строкой кэша. Следовательно, последовательность временных команд влияет на состояние кэша секретного значения, считанного ещё в шаге 1. Поскольку последовательность временных команд в шаге 2 работает против появления исключения,, значительно улучшается эффективность атаки. Если преобразование адресов для массива кэшируется в TLB, это усиливает атаки на некоторые системы.

Содержимое доступно по лицензии Creative Commons Attribution 4.0 license (CC-BY 4.0)

Шаг 3: *получение секрета*.

На шаге 3 атакующий восстанавливает секретное значение (шаг 1) за счет использования микроархитектуры, которая передает состояние кэша (шаг 2) обратно в архитектурное состояние. Meltdown использует Flush +Reload, чтобы передать состояние кэша на архитектурный уровень. Когда последовательность временных команд шага 2 выполняется, только одна строка кэша пробного массива кэшируются. Положение кэшированной строки кэша в пробном массиве зависит только от секрета, считанного в шаге 1. Таким образом, атакующий выполняет итерацию по всем 256 страницам пробного массива и измеряет время доступа для каждой первой строки кэша (т. е. смещения) на странице. Количество страниц, содержащих оптимизированную строку кэша, соответствует непосредственно секретному значению.

Дамп физической памяти. Повторяя все 3 шага, атакующий может получить доступ к дампу всей памяти, итерируя по всем различным адресам. Поскольку все основные операционные системы также отображают всю физическую память ядра в адресе каждого пользовательского процесса, Meltdown может также считать память ядра и физическую память.

Список литературы

1. Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, Mike Hamburg «Meltdown». – June 2017.

2. Meltdown and Spectre – Vulnerabilities in modern computers leak passwords and sensitive data [Электронный ресурс]. – Режим доступа: https://meltdownattack.com/ (дата обращения: 15.06.2018).

Content is licensed under the Creative Commons Attribution 4.0 license (CC-BY 4.0)