Марк М. Ермолов[1], Дмитрий В. Скляров[2], Максим С. Горячий[3]
АО «Позитив Текнолоджиз»,
Щелковское шоссе, 23А, помещение V, комната 30, Москва, 107241, Россия
[1]e-mail: mermolov@ptsecurity.com, http://orcid.org/0000-0001-8039-9557
[2]e-mail: dsklyrov@ptsecurity.com, https://orcid.org/0000-0002-1116-4452
[3]e-mail: m.goryachy@gmail.com, https://orcid.org/0000-0002-8812-7473

НЕДОКУМЕНТИРОВАННЫЕ ИНСТРУКЦИИ X86 ДЛЯ УПРАВЛЕНИЯ CPU НА УРОВНЕ МИКРОАРХИТЕКТУРЫ В СОВРЕМЕННЫХ ПРОЦЕССОРАХ INTEL*

DOI: http://dx.doi.org/10.26583/bit.2022.4.03

Аннотация. В рамках исследования, целью которого стало выявление ранее неизвестных уязвимостей, вызванных ошибками реализации или внедренными программно-аппаратными закладками в системном, прикладном и аппаратном обеспечении, был обнаружен режим отладки Red Unlock, который позволяет извлекать микрокод (ucode) и применим к процессорам Intel Atom. Используя данный режим отладки, удалось исследовать внутреннюю структуру микрокода и реализацию инструкций x86, в результате чего были найдены две недокументированные инструкции x86. Найденные недокументированные инструкции архитектуры x86 в современных процессорах Intel, которые были названы udbgrd и udbgwr, предназначены для чтения и записи данных микроархитектуры. Предполагается, что данные инструкции предназначены для отладки микроархитектуры процессоров инженерами Intel, но при этом их наличие является опасным с точки зрения безопасности, так как в публичном доступе имеется рабочая демонстрация, выполняющая активацию режима Red Unlocked для одной из актуальных платформ Intel. В работе представлены обнаруженные инструкции, приводится объяснение условий, при которых они могут быть использованы на общедоступных платформах. Подобные исследования могут быть использованы в целях развития методов и средств обеспечения информационной безопасности систем и сетей в части выработки решений, направленных на противодействие угрозам, обусловленных вновь выявляемыми уязвимостями, вызванными ошибками реализации или внедренными программно-аппаратными закладками в системном, прикладном и аппаратном обеспечении.

Ключевые слова: архитектура x86, недокументированная инструкция, угроза, уязвимость, Intel.
Для цитирования: ЕРМОЛОВ, Марк М.; СКЛЯРОВ, Дмитрий В.; ГОРЯЧИЙ, Максим С. НЕДОКУМЕНТИРОВАННЫЕ ИНСТРУКЦИИ X86 ДЛЯ УПРАВЛЕНИЯ CPU НА УРОВНЕ МИКРОАРХИТЕКТУРЫ В СОВРЕМЕННЫХ ПРОЦЕССОРАХ INTEL. Безопасность информационных технологий, [S.l.], т. 29, № 4, с. 27–41, 2022. ISSN 2074-7136. URL: https://bit.spels.ru/index.php/bit/article/view/1455. DOI: http://dx.doi.org/10.26583/bit.2022.4.03.

*Информация представлена в работе исключительно в образовательных целях. Ни авторы, ни их работодатель не несут ответственности за любой прямой или косвенный ущерб или убытки, возникшие в результате действий или бездействия любого лица или организации на основе информации, содержащейся в этой статье.

Mark M. Ermolov[1], Dmitry V. Sklyarov[2], Maxim S. Goryachy[3]
“Positive Technologies”
Shchelkovskoe shosse, 8, Moscow, 107061, Russia
[1]e-mail: mermolov@ptsecurity.com, https://orcid.org/0000-0001-8039-9557
[2]e-mail: dsklyrov@ptsecurity.com, https://orcid.org/0000-0002-1116-4452
[3]e-mail: m.goryachy@gmail, https://orcid.org/0000-0002-8812-7473

Undocumented x86 instructions to control the CPU at the microarchitecture level in modern INTEL processors*

DOI: http://dx.doi.org/10.26583/bit.2022.4.03

Abstract. The purpose of this study was to uncover previously unknown vulnerabilities in Intel CPUs caused by implementation errors or backdoors embedded in system firmware, applications, and hardware. The authors have discovered the Red Unlocked debugging mode which allows microcode to be extracted from Intel Atom processors. Using this debugging mode, the internal microcode structure and the implementation of x86 instructions have been examined, and two undocumented x86 instructions were found. These undocumented x86 instructions, udbgrd and udbgwr, can read and write microarchitectural data. These instructions are assumed to be intended for Intel engineers to debug the CPU microarchitecture. However, their existence poses a cybersecurity threat: there is a working demonstration available in the public domain on how to activate the Red Unlock mode for one of the current Intel platforms. This paper presents the analysis of the udbgrd and udbgwr instructions and explains the conditions under which they can be used on commonly available platforms. This kind of research can be used to develop methods, tools, and solutions to ensure information security of systems and networks by countering threats that arise from newly identified vulnerabilities stemming from implementation defects or backdoors in system firmware, applications, and hardware.

Keywords: attack, microcode, undocumented instruction, threat, vulnerability, Intel.

For citation: *ERMOLOV, Mark M.; SKLYAROV, Dmitry V.; GORYACHY, Maxim S. Undocumented x86 instructions to control the CPU at the microarchitecture level in modern INTEL processors. IT Security (Russia), [S.l.], v. 29, no. 4, p. 27–41, 2022. ISSN 2074-7136. URL: https://bit.spels.ru/index.php/bit/article/view/1455. DOI: http://dx.doi.org/10.26583/bit.2022.4.03.*

*All information in this paper is provided for educational purposes only. Neither the authors nor their employer are liable for any direct or consequential damage or loss arising from any person or organization acting or failing to act on the basis of information contained in this article.

Введение

Наличие недокументированных механизмов во внутренней реализации современных центральных процессоров (CPU) всегда волновало как исследователей информационной безопасности, так и простых пользователей. Основным фактором, вызывающим опасения, если допустить существование таких механизмов, является возможность обхода существующих мер защиты современных процессоров, выполняющих контроль доступа к памяти и к периферийным устройствам, содержащим личную информацию пользователей. Считается недопустимым, чтобы как сам производитель процессоров, так и какая-либо другая организация имели возможность получить несанкционированный доступ к данным, которые пользователь признает конфиденциальными. При этом доступ осуществляется в обход задокументированных, хорошо известных механизмов защиты, с применением методов, известных только самому производителю процессоров. Подобные методы могут быть основаны на функциональных особенностях, либо специально спроектированных для этих целей, либо имеющих вполне легитимное назначение, но которые в силу тех или иных архитектурных свойств также позволяют обойти защиту.

К таким недокументированным механизмам во внутренней реализации можно отнести:

1. Специальные режимы функционирования процессора, при которых стандартные правила разграничения доступа не работают или работают иначе, чем описано в документации (для Intel – в Software Developers Manual [1], специальном руководстве для разработчиков приложений, работающих на процессорах, выпускаемых компанией Intel).

2. Специальные микроархитектурные состояния CPU, при которых возможно нарушение тех или иных правил контроля доступа (возникающие, например, при выполнении определенной последовательности инструкций процессора).

3. Наличие недокументированных инструкций процессора, которые позволяют обойти существующие механизмы защиты (например, памяти) и предоставляют особые привилегии для вызывающего кода.

Существование подобных недокументированных возможностей для процессоров компании Intel всегда было предметом споров и спекуляций в среде исследователей информационной безопасности. За более чем 40-летнее существование архитектуры x86 был проведен ряд значимых исследований в этой области [2–4]. В результате некоторые недокументированные возможности, найденные исследователями, стали общедоступными (например, недокументированные инструкции, описанные в [3]). При этом одна из найденных недокументированных инструкций процессора позволяла обходить встроенную защиту памяти, однако ее наличие было установлено только в очень старых процессорах Intel (80286 и 80386). С тех пор в x86 процессорах, производимых самой компанией Intel, не было выявлено каких-либо явных, недокументированных методов получения доступа в обход существующей защиты, если не принимать во внимание множество найденных за последнее время уязвимостей переходного состояния выполнения и утечек по сторонним каналам (Meltdown, Spectre и т.д.), которые никак нельзя отнести к намеренным и которые по общему признанию являются ошибками в проектировании, ввиду стремления к наибольшей производительности в ущерб безопасности. Однако в ходе проводимых исследований доказано наличие как минимум двух недокументированных x86-инструкций и специальный отладочный режим работы процессора, которые в совокупности позволяют полностью обойти существующие механизмы защиты как памяти, так и контроля оборудования. Более того, доступ к процессору на уровне микрокода через найденные недокументированные инструкции, позволяет переопределить семантику основных инструкций x86 (определенного набора самых важных с точки зрения безопасности инструкций), что является куда более опасным, чем, например, только обход защиты памяти (уровней привилегий x86, так называемых колец защиты – rings), так как это допускает реализацию вредоносного программного обеспечения на таком уровне, где его невозможно ни детектировать современными средствами защиты, ни как-то от него защититься.

Можно предположить, что данные инструкции изначально предназначены для отладки на уровне микроархитектуры (аппаратной реализации) процессоров исключительно инженерами компании Intel, однако, несмотря на полное отсутствие открытой документации по этой теме и описания всех возможных способов их применения, очевидно, что эти инструкции являются весьма опасными, а сервисный режим отладки процессоров Red Unlocked – недопустимым в конечных продуктах, реализуемых в магазинах и используемых конечными пользователями в повседневных задачах. Более того, на некоторых платформах указанный отладочный режим работы центральных процессоров может быть активирован аппаратными средствами, к которым имеет доступ подсистема Intel Converged Security and Management Engine (Intel CSME) [5], в которой за последнее время был найден ряд серьезных уязвимостей [6, 7], позволяющих выполнять произвольный код и таким образом включать отладочный режим основного процессора, что и было продемонстрировано в [8]. Это делает отладочный режим Red Unlocked доступным для использования любому желающему, лишая его начального предназначения только для инженеров компании Intel.

Стоит отметить, что данная проблематика ранее исследовалась нами, а результаты были опубликованы в [9]. Однако, в настоящей статье большее внимание уделяется описанию процедуре поиска и условиям выполнения инструкции udbgrd/udbgwr.

1. Исследование безопасности аппаратной платформы компании Intel

Во время проводимых исследований были найдены следующие недокументированные инструкции архитектуры x86 в современных процессорах Intel:

1. Инструкция **udbgrd – 0F 0E – microarchitecture (uarch) debug read**.

Указанная инструкция предназначена для чтения данных микроархитектуры и принимает следующие аргументы:

а) rcx – команда, задающая тип (источник) читаемых данных:

– 0x00 – CRBUS (Control Register Bus), шина, соединяющая все исполнительные блоки внутри ядра процессора, такие как IFU – instruction fetch unit, DCU – data cache unit, MS – microcode sequencer, CORE – execution core и др. Каждый из блоков имеет свой диапазон адресов для регистров, доступных через CRBUS.

– 0x08 – mailbox команд для PCODE[1], косвенный доступ к которому предоставляется через порты ввода/вывода 0xa0/0xa4. Микрокод использует этот mailbox для управления PCODE на разных стадиях работы процессора, например для загрузки заплаток, исправляющих ошибки в микропрограмме (PCODE patch). Данная команда через инструкцию udbgrd (0x08) позволяет читать внутреннюю память PCODE.

– 0x10 – URAM (небольшая SRAM микрокода, которая является закрытой памятью ядра CPU и не разделяется между другими ядрами). Внутреннее имя этой закрытой SRAM микрокода – FSCP (расшифровка аббревиатуры не установлена), и некоторые ее ячейки доступны через MSR, хотя число адресов, которые могут быть считаны и записаны через MSR, очень мало.

– 0x18 – 8-битный порт ввода/вывода, обращение к которому происходит от лица микрокода.

– 0x40 – Staging Buffer, специальная SRAM, по размеру много большая URAM, доступ к которой разделяется между несколькими ядрами процессора и которая расположена в сегменте электропитания AON (Always On), то есть Staging Buffer, в отличие от URAM, не подвергается отключению от электропитания в случае перехода ядер процессора в энергосберегающий режим (в результате процедуры Power Gating).

– 0x48 – 16-битный порт ввода/вывода.

– 0x50 – 32-битный порт ввода/вывода.

– 0x58 – 64-битный порт ввода/вывода.

– 0x80 – Staging Buffer[2].

б) rax – адрес данных (специфичный для типа данных, заданных регистром rcx).

Выходом для данной инструкции являются:

а) rdx – содержит до 64 бит считанных данных (если тип данных, заданный регистром rcx, предполагает данные размером меньше 64 бит, то старшие биты – нулевые).

б) rbx – старшие 32 бита считанных данных, дублирующие старший DWORD из rdx. Для команды 0x08, содержит не старший DWORD данных, так как регистр системного агента всегда 32-битный, а код ошибки (результат чтения) внешней шины.

2. Инструкция **udbgwr – 0F 0F – microarchitecture debug write**.

Данная инструкция предназначена для записи микроархитектурных данных и принимает следующие аргументы:

а) rcx – команда, определяющая тип записываемых данных. Для udbgwr поддерживаются все команды инструкции udbgrd. Кроме того, udbgwr поддерживает дополнительно следующие команды:

---

[1]Микропрограммы, выполняемой на микроконтроллере управления электропитанием процессора, который называется P-unit.

[2]Отличие от 0x40 пока не совсем ясно, возможно обращение к части, зарезервированной для другого ядра.

– 0xc8 – выполнение запроса на шине IOSF-SB от лица микрокода, то есть с использованием Port ID микрокода и его идентификатора безопасности SAI. Например, микрокод через IOSF-SB обращается к так называемой SSRAM (Shared SRAM), которая находится в блоке управления электропитанием платформы Power Management Controller (PMC), и считывает с нее некоторые глобальные настройки платформы, которые задаются через PMC.

– 0xd0 – mailbox PCODE. В отличие от команды 0x08, позволяет задать код операции для mailbox (команда 0x08 использует встроенные коды операций 0x05 для чтения, 0x07 для записи памяти PCODE, однако PCODE поддерживает ряд других операций, например остановку и запуск выполнения PCODE).

– 0xd8 – вызов процедуры в микрокоде через указание произвольного адреса MSROM. Микроархитектурные регистры tmp0-tmp15 передаются через Staging Buffer (которые позволяет записать команда 0x40), начиная с адреса 0xb800 с шагом 0x40. Архитектурные регистры (кроме rcx и rax) передаются напрямую.

б) rax – адрес записываемых данных, который специфичный для типа (команды), заданной в rcx. Для CRBUS – это адрес регистра одного из модулей CPU, для порта ввода/вывода – адрес I/O регистра, для URAM – адрес ячейки памяти и т.д.

в) rdx:rbx – записываемые данные. Если команда подразумевает данные меньше 64 бит, то значение rbx игнорируется (в том числе при обращении к регистрам Системного Агента командой 0x08. Для команды 0xc8, rbx содержит код операции для шины Системного Агента).

Выходом для данной инструкции является:

а) rbx – только для команд обращение к регистрам Системного Агента (0x80, 0xc8) содержит результат записи (код ответа), считанный с внутренней шины. Во всех остальных значение исходного регистра (на входе в инструкцию) не изменяется.

б) rdx – только для команды записи шины IOSF-SB для PCU, содержит код ответа внутренней шины. Во всех остальных случаях – исходное значение.

2. Условия выполнения инструкций udbgrd/udbgwr

Используя механизмы отладки, доступные через LDAT[3] было установлено, что микроархитектурный декодер обрабатывает указанные инструкции во всех режимах (User Mode [10], Kernel Mode [10], SMM [10], VMX Root [11], VMX Non-Root [11], SGX [12] и др.). То есть IFU распознает длину этих инструкций (2 байта), а составной декодер (complex decoder) вызывает соответствующие точки входа в MSROM. Микрокод для найденных инструкций в MSROM выглядит следующим образом (адреса и микрокод приведены для ядра Atom Goldmont):

```
...
U028a: tmp3:= MOVEFROMCREG_DSZ64(0x2e6, 32)
U028c: BTUJNB_DIRECT_NOTTAKEN(tmp3, 0x00000009, generate_#UD)
U028d: tmp3:= MOVEFROMCREG_DSZ64(CTAP_CR_DFX_CTL_STS, 32)
U028e: BTUJB_DIRECT_NOTTAKEN(tmp3, 0x0000000a, generate_#UD)
    SEQW GOTO U27c9
...
udbgwr_xlat:
U0660: tmp2:= CONCAT_DSZ32(rbx, rdx)
U0661: tmp1:= MOVE_DSZ64(0x00000001)
```

[3]Local Direct Access Test, специальный отладочный порт для исполнительных модулей ядра CPU, доступный через регистры CRBUS.

```
U0662: tmp3:= MOVEFROMCREG_DSZ64(0x38c, 32)
      SEQW GOTO U0b5a
...
udbgrd_xlat:
U0b58: tmp1:= MOVE_DSZ64(0x00000000)
U0b59: tmp3:= MOVEFROMCREG_DSZ64(0x38c, 32)
U0b5a: tmp3:= NOTAND_DSZ32(tmp3, 0xa0000000)
U0b5c: UJMPCC_DIRECT_NOTTAKEN_CONDZ(tmp3, U028d)
U0b5d: tmp3:= READURAM(0x005c, 64)
U0b5e: BTUJB_DIRECT_NOTTAKEN(tmp3, 0x00000002, U028d)
      SEQW GOTO U028a
...
U27c9: BTUJB_DIRECT_NOTTAKEN(rcx, 0x00000005, U6bce)
U27ca: BTUJB_DIRECT_NOTTAKEN(rcx, 0x00000002, U27cc)
      SEQW GOTO U27cd
-----------------------------------------------------
U27cc: SAVEUIP_REGOVR(0x01, U27cd, 0x0005)
      SEQW GOTO U32cd
U27cd: tmp2:= CONCAT_DSZ32(rbx, rdx)
U27ce: tmp4:= NOTAND_DSZ32(0x00000001, rax)
U27d0: tmp5:= AND_DSZ32(0x000000c0, rcx)
U27d1: tmp5:= SHR_DSZ32(tmp5, 0x00000001)
U27d2: tmp6:= AND_DSZ32(0x00000018, rcx)
U27d4: tmp8:= OR_DSZ32(tmp6, tmp5)
U27d5: LFNCEMARK-> BTUJNB_DIRECT_NOTTAKEN(tmp1, 0x00000000, U27d6)
      SEQW GOTO U27d9
-----------------------------------------------------
U27d6: NOP
U27d8: tmp9:= ADD_DSZ32(tmp8, 0x00004052)
      SEQW GOTO U27da
-----------------------------------------------------
U27d9: tmp9:= ADD_DSZ32(tmp8, 0x00004392)
U27da: LFNCEWAIT-> UJMP(tmp9)
...
U4052: BTUJB_DIRECT_NOTTAKEN(rax, 0x0000000d, U5902)
U4054: tmp10:= MOVEFROMCREG_DSZ64(rax)
      SEQW GOTO U4065
...
U4065: rdx:= EXTEND_DSZ64(tmp10)
U4066: rbx:= SHR_DSZ64(tmp10, 0x00000020)
      SEQW GOTO U43a4
...
U43a4: LFNCEMARK-> MOVETOCREG_BTS_DSZ64(0x00000001, 0x289)
      SEQW GOTO uend
```

Выше приведена реализация в микрокоде только одной из команд (CRBUS read) для найденных инструкций, которая начинается с адреса U4052.

Точкой входа инструкции udbgrd является адрес в MSROM U0b58, а для udbgwr – U0660. Основные же действия по выполнению заданных команд происходят начиная с адреса U27c9, при этом, как для операций чтения, так и для записи, рассчитывается целевой адрес в ROM микрокода, по которому расположен фактический код, выполняющий команду чтения или команду записи. Для команды чтения CRBUS этот код начинается с адреса U4052: он вычисляется путем прибавления идентификатора команды (по адресу U27d8), переданного в регистр rcx перед началом выполнения инструкции udbgrd, к значению 0x00004052. Отсюда следует шаг, с которым изменяются идентификаторы команд, равный 0x08. Для команды чтения CRBUS, идентификатор которой равен 0x00, получается целевой адрес, равный U4052, которому передается управление микрооперацией UJMP по адресу U27da. Для команд записи начальный адрес, от которого вычисляются обработчики команд, равен U4392 (по адресу U27d9). Таким образом, реализована поддержка нескольких команд инструкциями udbgrd/udbgwr.

Однако основной код, выполняющий проверку условий, при которых возможно выполнение описываемых инструкций, расположен начиная с адреса U028a. Инструкции udbgrd/udbgwr принимаются фронтендом CPU на выполнение во всех режимах работы процессора, и именно микрокод, расположенный по адресам начиная с U028a, решает, допустимо ли выполнение инструкций или нет.

Итак, начиная с адреса U028a, можно видеть проверку двух условий, при соблюдении которых возможно выполнение указанных инструкций:

1. Бит #9 (значение 0x200) установлен в регистре по адресу 0x2e6 на CRBUS.
2. Бит #10 (значение 0x400) не установлен в регистре CTAP_CR_DFX_CTL_STS по адресу 0x285 CRBUS.

При невыполнении любого из этих условий микрооперации BTUJNB_DIRECT_NOTTAKEN/ BTUJB_DIRECT_NOTTAKEN передают управление на подпрограмму generate_#UD, которая в конечном итоге сводится к выполнению микрооперации SIGEVENT и вызову обработчика исключения #UD, Invalid Opcode Exception, имеющего вектор прерываний 0x06. Микрооперации BTUJNB_* выполняют тестирование бита с индексом, заданным в качестве второго операнда, и если он установлен (или не установлен, в зависимости от типа микрооперации B или NB), то передают управление на микрокод, адрес которого задан в третьем аргументе микрооперации, в ином случае управление передается на микрооперацию, следующую непосредственно за BTUJNB_*.

Как уже было отмечено выше, внутренняя шина ядра процессора, которая называется CRBUS, объединяет все модули (исполнительные блоки), расположенные внутри IP-блока процессора, называемого Core, который производит фактическое выполнение x86-инструкций. К таким модулям можно отнести: устройство выборки инструкций (IFU, Instruction Fetch Unit), декодер x86, кэш данных первого уровня (DCU, Data Cache Unit), устройство генерации микрокода (MS, Microcode Sequencer) и др. Каждый модуль имеет зарезервированный за собой диапазон адресов на CRBUS, через которые происходит обращение к управляющим регистрам модуля. Однако иногда адреса перемешиваются, и регистры разных модулей могут иметь смежные адреса на CRBUS. Во внутренней документации Intel (например, в xml-файлах системного программного обеспечения) в названии каждого регистра присутствует префикс, обозначающий модуль, к которому относится регистр. Например, регистр CTAP_CR_DFX_CTL_STS показывает принадлежность к модулю управления портом отладки JTAG процессора (CTAP – Test Access Port Controller).

Регистр CRBUS с адресом 0x2e6 предположительно не связан с конкретным аппаратным модулем ядра процессора: существует группа регистров CRBUS, которые выполняют функцию хранения данных и сохраняются в специальной SRAM внутри ядра процессора. Микрокод обращается к таким регистрам для сохранения информации времени выполнения, которая может храниться независимо от вычислительного ядра. Такие регистры CRBUS имеют префикс UCODE_CR_*. На момент написания работы не все свойства таких регистров известны (например, могут ли они быть читаемы и записываемы другими модулями, подключенными к CRBUS). Однако можно совершенно точно утверждать, что сам микрокод отображает значение регистра CRBUS с адресом 0x2e6 на MSR с адресом 0x1e6. Была получена полная таблица всех MSR, поддерживаемых ядром Goldmont: в каких режимах процессора доступен MSR, его тип (то есть куда он отображается: на URAM, CRBUS или на внешние по отношению к ядру устройства), адрес, специфичный для типа MSR, точки входа в MSROM, где происходит фактическое чтение/запись MSR и т.д. Также было найдено, что запись в MSR 0x1e6 приводит к записи заданного значения в регистр 0x2e6 на CRBUS. Итак, условием #1 для выполнения инструкций можно управлять с помощью MSR 0x1e6.

Однако, как это можно заключить, читая микрокод от точек входа (меток с окончанием _xlat) исследуемых инструкций, условие #1 не всегда является обязательным. Можно увидеть, что микрокод по адресу U0b5c и по адресу U0b5e позволяет пропустить проверку условия #1 и передать управление сразу на условие #2. По адресу U0b5c происходит проверка другого регистра CRBUS, с адресом 0x38c, а по адресу U0b5e происходит проверка бита #2 из ячейки памяти URAM с адресом 0x005c. Точно не известно, как называется регистр с адресом 0x38c и чем он управляет. Однако, предполагаем, что этот регистр относится к модулю IFU, который выбирает инструкции x86 из входного потока инструкций на выполнение. Такое заключение можно сделать по диапазону адресов, в который попадает данный регистр, так как точно известно, что регистр с адресом 0x3c0 является LDAT-портом для IFU – через этот порт было выполнено обращение к внутренним массивам (arrays) IFU и видны исходные x86-инструкции до декодирования. Если данное предположение верно, то регистр 0x38c может отражать либо значение FUSE-битов (однократно программируемой памяти) для IFU, либо специальные архитектурные состояния входного потока инструкций, например, наличие специального префикса или определенного порядка инструкций x86. Микрокод по адресам U0b59-U0b5c проверяет, что биты #29 и #31 установлены в регистре 0x38c. В таком случае возможен пропуск проверки условия #1 из двух обозначенных выше (бита, устанавливаемого через MSR 0x1e6). Таким образом, будет верным утверждение, что, при определенных условиях входного потока инструкций, инструкции udbgrd/udbgwr могут не требовать их предварительного включения через MSR 0x1e6. Также, если бит #2 64-битного значения, записанного в URAM (закрытой, статической памяти времени выполнения для микрокода) по адресу 0x005c, установлен, то условие #1 также не обязательно. Была найдена установка бита #2 в URAM по адресу 0x005c в микрокоде, связанном с обработкой MCE (Machine Check Exception), если возникающая ошибка Machine Check связана с внутренними модулями ядра процессора (в отличие от ошибок, например, внешних устройств). В таком случае, предположительно, в обработчике #MC (вектор прерываний 0x12) возможно использование udbgrd/udbgwr без предварительного их включения через MSR 0x1e6.

Но самым важным условием выполнения udbgrd/udbgwr является условие #2, которое должно быть соблюдено в любом случае (в микрокоде, нет путей его обойти), для успешного выполнения найденных инструкций. Экспериментальным способом было выяснено, что регистр CTAP_CR_DFX_CTL_STS по адресу 0x285 CRBUS является аппаратным (некоторые биты не могут быть записаны микрокодом, а устанавливаются на уровне железа). Бит #11 (0x800) этого регистра точно управляет CTAP (отладочным портом ядра процессора, подключенным к JTAG): при установке этого бита доступ к привилегированным JTAG-командам блокируется (IR-/DR-последовательности сканирования scan chain, которые доступны в режиме Red Unlocked, начинают возвращать 0). Бит #10 (0x400), который поверяется в условии #2, не может быть модифицирован микрокодом и является статусным битом, который выставляется на аппаратном уровне. Было установлено, что бит #10 регистра CTAP_CR_DFX_CTL_STS[4] принимает значение 0 (сброшен) при выполнении процедуры отладочной разблокировки процессора Red Unlock. Стоит отметить, что нельзя с полной уверенностью утверждать, что режим Red Unlocked является единственным способом сброса бита #10 регистра CTAP_CR_DFX_CTL_STS. Возможно, существуют другие условия[5], кроме Red Unlocked, при которых бит #10 также обращается в 0 и появляется возможность исполнения инструкций udbgrd/udbgwr. На данный момент можно сказать точно лишь то, что в состоянии Red Unlocked необходимое условие #2 выполняется и блокировка инструкций udbgrd/udbgwr снимается.

В процессе проводимых исследований у компании Intel нами в online конференции через Skype была запрошена информация по точному описанию всех битов регистра CTAP_CR_DFX_CTL_STS для того, чтобы можно было с определенной уверенностью заявлять, что использование режима Red Unlocked является единственно возможным способом активации инструкций udbgrd/udbgwr, но на момент написания настоящей статьи ответ получен не был. Необходимо отметить, что в процедуре микрокода, обрабатывающей запись в MSR (x86-инструкцию wrmsr) с адресом 0x1e6 (для условия #1), также проверяется, сброшен ли бит #10 регистра CTAP_CR_DFX_CTL_STS, и в случае его отсутствия, инструкция wrmsr для данного адреса генерирует исключение #GP (General Protection). Таким образом, можно предположить, что бит #10 регистра CTAP_CR_DFX_CTL_STS означает состояние DFX блокировки ядра процессора (DfxPolicyLocked).

Поведем итог: инструкции udbgrd/udbgwr требуют активации через бит #9 (значение 0x200) в MSR 0x1e6 и могут быть выполнены, если процессор находится в режиме Red Unlocked. При этом существуют иные способы активации данных инструкций, помимо MSR 0x1e6, для процессора в состоянии Red Unlocked, например автоматическая активация в обработчике #MC. Также нельзя утверждать, что режим Red Unlocked является единственно возможным условием для выполнения инструкций udbgrd/udbgwr и не существует иного способа, программного или аппаратного, с помощью которого процессор может быть переведен в состояние, при котором возможна активация и выполнение данных инструкций.

3. Процедура поиска инструкции udbgrd/udbgwr

Был извлечен микрокод для современных процессоров Atom, построенных на микроархитектуре Goldmont. Это стало возможным для SoC (System on Chip), основанных на Atom Goldmont, благодаря уязвимости произвольного выполнения кода подсистемы Intel CSME, описанной в Intel SA-00086. С помощью данной уязвимости удалось включить режим Red Unlocked на данных SoC не только для чипсета (PCH), но и для центрального процессора. Это стало возможно, так как для SoC аппаратные компоненты, управляющие Unlocked-состояниями платформы, разделяются между IP-модулями системной логики и компонентами центрального процессора, одним из которых и является исполнительное

[4]Имя для регистра было задано по аналогии с другими регистрами CRBUS, которые встречаются в различных источниках, и согласно предположению, что регистр управляет для ядра процессора отладочными механизмами DFX – Designed for Testability, Debugging and Manageability – технологиями аппаратной отладки продуктов Intel посредством JTAG.

[5]Было замечено, что сам микрокод также обращается к регистру CTAP_CR_DFX_CTL_STS на запись, но к битам #0 и #1 при выполнении определенных задач.

ядро x86. Задействование режима Red Unlocked для CPU на платформе Intel Apollo Lake позволило восстановить ряд внутренних регистров JTAG, IR-коды (Instruction Register – терминология стандарта JTAG [13]) которых отсутствуют в публичных версиях программных пакетов Intel DAL и Intel OpenIPC для аппаратной отладки микросхем Intel, через которые (IR-коды) возможно обращение к так называемой шине CRBUS, Control Register Bus, ядра процессора. Доступ к CRBUS позволил найти регистры внутреннего модуля, называемого MS или Microcode Sequencer. Используя регистры MS для доступа к его порту LDAT (Local Direct Access Test), удалось извлечь все 5 внутренних массивов MS (внутренней памяти), одним из которых был MSROM – память только для чтения, в которой хранится микрокод как всех реализуемых через microcode-assist инструкций x86 (некоторые инструкции декодируются напрямую, минуя MSROM), так и начальной инициализации CPU. Затем был осуществлен обратный инжиниринг извлеченного микрокода в целях получения информации о формате микроопераций и их назначения. При этом возникла необходимость разработки дизассемблера микрокода для ядра Atom Goldmont, с целью изучения микроархитектуры современных процессоров Intel и реализации x86-инструкций. На данный момент разработанный дизассемблер микрокода процессоров Intel Atom опубликован [14].

При изучении листинга микрокода, полученного разработанным дизассемблером, был выявлен следующий фрагмент микрокода:

```
U30fd: tmp6:= LDSTGBUF_DSZ64_ASZ16_SC1(0x0000b980)
U30fe: tmp7:= LDSTGBUF_DSZ64_ASZ16_SC1(0x0000b9c0)
U3100: tmp8:= LDSTGBUF_DSZ64_ASZ16_SC1(0x0000ba00)
U3101: tmp9:= LDSTGBUF_DSZ64_ASZ16_SC1(0x0000ba40)
U3102: tmp10:= LDSTGBUF_DSZ64_ASZ16_SC1(0x0000ba80)
U3104: tmp11:= LDSTGBUF_DSZ64_ASZ16_SC1(0x0000bac0)
U3105: tmp12:= LDSTGBUF_DSZ64_ASZ16_SC1(0x0000bb00)
U3106: tmp13:= LDSTGBUF_DSZ64_ASZ16_SC1(0x0000bb40)
U3108: tmp14:= LDSTGBUF_DSZ64_ASZ16_SC1(0x0000bb80)
U3109: tmp15:= LDSTGBUF_DSZ64_ASZ16_SC1(0x0000bbc0)
U310a: LFNCEMARK-> SAVEUIP(0x00, U21fe)
U310c: SAVEUIP(0x01, U21fe)
U310d: NOP
U310e: LFNCEWAIT-> UJMP(rax)
```

В представленном фрагменте полагается странным присутствие микрооперации UJMP(rax), в бинарном виде 0x015d00000800, поскольку это свидетельствует о том, что в микрокоде существует место, которое позволяет передать управление на произвольный адрес в MSROM, который в свою очередь задается архитектурным регистром. Предполагается, что опкод микрооперации UJMP (0x15d) был установлен правильно, так как это было подтверждено экспериментальным путем: используя механизмы match/patch, можно выполнять произвольный микрокод при вызове любой из x86-инструкций, реализуемой с помощью MSROM-assist. Также полагается верным то, что аргумент микрооперации находится именно в регистре rax (селектор 0x20 в поле src1). Все это означает, что существует x86-инструкция, позволяющая вызывать произвольный адрес в MSROM. Также установлено, что микроархитектурное состояние (регистры tmpx), считывается из Staging Buffer (статической памяти, разделяемой всеми ядрами CPU)до передачи управления на заданный адрес в микрокоде, так как эти регистры нельзя задать на уровне архитектуры x86, но они активно используются самим микрокодом. Данный факт подтверждает, что в ходе исследования удалось найти механизм вызова произвольной подпрограммы в MSROM.

При поиске ссылки на данный фрагмент микрокода по всему MSROM было выявлено начало этого фрагмента, на который уже не было прямых ссылок в микрокоде (в операциях передачи управления на адрес в MSROM, условных или безусловных, в которых целевой адрес задается непосредственно в микрооперации):

```
U440a: tmp0:= LDSTGBUF_DSZ64_ASZ16_SC1(0x0000b800)
U440c: tmp1:= LDSTGBUF_DSZ64_ASZ16_SC1(0x0000b840)
U440d: tmp2:= LDSTGBUF_DSZ64_ASZ16_SC1(0x0000b880)
U440e: tmp3:= LDSTGBUF_DSZ64_ASZ16_SC1(0x0000b8c0)
U4410: tmp4:= LDSTGBUF_DSZ64_ASZ16_SC1(0x0000b900)
U4411: tmp5:= LDSTGBUF_DSZ64_ASZ16_SC1(0x0000b940)
    SEQW GOTO U30fd
```

Таким образом, можно заметить, что полный набор микроархитектурных регистров до передачи управления считывается из Staging Buffer. Несмотря на то, что нет прямых ссылок на данный фрагмент микрокода, само его наличие говорит о том, что управление на него должно передаваться тем или иным способом. Полагается верным, что существует место в MSROM, которое передает управление на этот фрагмент косвенно, возможно используя арифметику для вычисления целевого адреса в микрокоде. Чтобы подтвердить данное предположение, был выполнен поиск всех констант размера 16 бит, которые использовались в арифметических микрооперациях сложения или вычитания, результат которых мог дать значение 0x0000440a. Был найден следующий набор микроопераций в микрокоде:

```
...
U27d9: tmp9:= ADD_DSZ32(tmp8, 0x00004392)
U27da: LFNCEWAIT-> UJMP(tmp9)
```

При изучении микрокода, расположенного выше (адреса U27d9), было установлено, что регистр tmp8 может принимать значение 0x78 (0x00000440a–0x00004392), необходимое для передачи управления на требуемый адрес. При изучении логики формирования значения регистра tmp8 было установлено, что, помимо вызова произвольного адреса MSROM, существует целый набор подпрограмм для доступа к самым разным данным микроархитектуры, причем как для записи, так и для чтения. Среди них стоит отметить доступ к CRBUS – шине, через которую возможно управлять регистрами match/patch и памятью Patch RAM в Microcode Sequencer, так как именно через эти механизмы и происходит изменение произвольных областей MSROM данными из microcode patch.

В процессе изучения общего блока, позволяющего вызвать различные подпрограммы в микрокоде, были найдены две точки входа в MSROM для x86-инструкций, которые обе задействуют этот механизм. Точки входа для x86-инструкций расположены в MSROM по младшим адресам с U0000 до U1000, причем адрес должен быть кратен 8 и на него не должно быть ссылок из других областей микрокода в MSROM. Таким образом, были найдены две точки входа неизвестных инструкций (одна запускает описанный выше механизм для записи данных, другая – для чтения):

```
udbgwr_xlat:
U0660: tmp2:= CONCAT_DSZ32(rbx, rdx)
U0661: tmp1:= MOVE_DSZ64(0x00000001)
U0662: tmp3:= MOVEFROMCREG_DSZ64(0x38c, 32)
    SEQW GOTO U0b5a
...
```

```
udbgrd_xlat:
U0b58: tmp1:= MOVE_DSZ64(0x00000000)
U0b59: tmp3:= MOVEFROMCREG_DSZ64(0x38c, 32)
U0b5a: tmp3:= NOTAND_DSZ32(tmp3, 0xa0000000)
U0b5c: UJMPCC_DIRECT_NOTTAKEN_CONDZ(tmp3, U028d)
U0b5d: tmp3:= READURAM(0x005c, 64)
U0b5e: BTUJB_DIRECT_NOTTAKEN(tmp3, 0x00000002, U028d)
SEQW GOTO U028a
```

Разработанный дизассемблер поддерживает метки для микрокода, в связи с этим произвольному адресу может быть назначена текстовая метка, для более удобного его представления в листинге микрокода. Было принято решение называть все метки, относящиеся к точкам входа x86-инструкций, с окончанием _xlat, чтобы отличать их от других меток, описывающих различные подпрограммы в микрокоде. Название XLAT (сокращение от Translate) в микроархитектуре современных процессоров обозначает механизм статического преобразования опкода x86-инструкции в адрес MSROM, по которому расположена точка входа реализации этой инструкции в микрокоде. Это название отражает табличный характер этого механизма. Механизм (табличного преобразования) используется составным декодером для обработки x86-инструкций, которые реализуются с помощью MSROM-assist (большая часть инструкций x86).

Следует отметить, что на данный момент XLAT-таблицы среди различных микроархитектурных массивов, доступных для чтения/записи через JTAG/CRBUS для ядра Atom Goldmont, не обнаружены. Данная информация является особо важной, так как позволяет однозначно сопоставить точки входа в MSROM с x86-инструкциями (их опкодами). Единственный способ найти инструкции x86 для интересующих точек входа в MSROM – это фаззинг (fuzzing) всех опкодов x86, то есть их последовательный перебор.

Задача фаззинга x86 для поиска недокументированных инструкций представляется довольно сложной [15] и в ее классическом представлении есть один недостаток – довольно сложно отличить реально несуществующий опкод x86 (при котором возникает исключение #UD) от, например, неверного формата данной инструкции или неверного режима исполнения процессора, при котором эта инструкция может быть выполнена. Однако существует возможность настройки match/patch-механизма на перехват необходимых точек входа MSROM для неизвестных x86-инструкций с последующим запуском перебора всех возможных x86-опкодов и ожидание срабатывания перехватчиков. Это позволило значительно оптимизировать механизм перебора x86-опкодов при отправке на выполнение до 256 инструкций за раз: если срабатывало исключение #UD, осуществлялся перезапуск оставшихся инструкций из данного набора.

Среди инструкций с двухбайтными опкодами перехватчики сработали для инструкций 0f 0e и 0f 0f, первая – для точки входа операций чтения, вторая – для точки входа операций записи. Таким образом недокументированные инструкции x86 были найдены.

Объединение PoC для Red Unlocked платформы Apollo Lake [8] и PoC на чтение/запись CRBUS с помощью udbgrd/udbgwr [16] позволило обращаться к CRBUS, используя локальный доступ, без необходимости подключения к платформе через JTAG. Данный факт делает режим Red Unlocked особо важным с точки зрения обеспечения безопасности платформы, поскольку ранее он был опасен только с физическим подключением через XDP/DCI.

Поскольку сам режим Red Unlocked существует во всех процессорах Intel уже довольно продолжительное время, то можно предположить, что найденные инструкции также поддерживаются и на десктопных процессорах Intel. Позже был получен ряд косвенных подтверждений существования этих инструкций и на других CPU, которые не относятся к линейке Atom [17]. Ввиду того что по набору поддерживаемых инструкций x86 современные процессоры Atom не сильно отличаются от последних процессоров основной линейки (они поддерживают все, кроме AVX-расширений – VMX, SGX, MPX и др.), можно сделать вывод, что найденные инструкции являются недокументированной частью самой архитектуры x86 и поддерживаются всеми актуальными процессорами Intel.

На данный момент существует доказательство их наличия только на платформах Apollo Lake и Gemini Lake (основанных на процессорах Atom Goldmont и Goldmont Plus)[18], для которых был написан PoC[8], использующий режим Red Unlocked.

Еще одним аргументом в пользу того, что существующие инструкции поддерживаются и на основных процессорах, выпускаемых Intel, является ориентация компании на гибридные процессорные архитектуры, когда на одном кристалле часть ядер относится к семейству процессоров Intel Atom, а другая часть к «большим» процессорным ядрам поколения Intel Core (например, в SoC Alder Lake).

Заключение

В процессе проводимых исследований были найдены две недокументированные x86-инструкции, которые были названы udbgrd и udbgwr. Найденные инструкции позволяют читать и писать микроархитектурные данные. Одной из поддерживаемых команд для этих инструкций является чтение и запись CRBUS, что позволяет модифицировать микрокод MSROM через Patch RAM и регистры match/patch.

Удалось доказать наличие найденных недокументированных инструкций на процессорах поколения Atom Goldmont и Atom Goldmont Plus. Было установлено, что найденные инструкции требуют активации через MSR 0x16e и выполняются процессором, только если предварительно была проведена процедура DFX-разблокировки, называемая Red Unlock.

Предполагается, что данные инструкции предназначены для отладки микроархитектуры процессоров инженерами Intel, но при этом их наличие является опасным с точки зрения безопасности, так как в публичном доступе имеется рабочий PoC, выполняющий активацию режима Red Unlocked для одной из актуальных платформ Intel.

СПИСОК ЛИТЕРАТУРЫ

1. Intel Corp. Intel 64 and IA-32 Architectures Software Developer’s Manual. 2021. URL: https://software.intel.com/content/dam/develop/external/us/en/documents-tps/325462-sdm-vol-1-2abcd-3abcd.pdf (дата обращения: 20.08.2022).
2. Easdon C. Undocumented CPU Behavior: Analyzing Undocumented Opcodes on Intel x86-64. 2020. URL: https://www.cattius.com/images/undocumented-cpu-behavior.pdf (дата обращения: 20.08.2022).
3. Collins R. Undocumented OpCodes. 1995. URL: http://www.rcollins.org/secrets/OpCodes.html (дата обращения: 20.08.2022).
4. Domas C. Breaking the x86 ISA. 2017. URL: https://www.blackhat.com/docs/us-17/thursday/us-17-Domas-Breaking-The-x86-Instruction-Set-wp.pdf (дата обращения: 20.08.2022).
5. Intel Corp. Intel Converged Security and Management Engine (IntelR CSME) [Security White Paper].2020. URL: https://www.intel.com/content/dam/www/public/us/en/security-advisory/documents/intel-csme-security-white-paper.pdf (дата обращения: 20.08.2022).
6. Ermolov M., Goryachy M. How to Hack a Turned-Off Computer, or Running Unsigned Code in Intel Management Engine. 2017. URL: https://www.blackhat.com/docs/eu-17/materials/eu-17-Goryachy-How-To-Hack-A-Turned-Off-Computer-Or-Running-Unsigned-Code-In-Intel-Management-Engine.pdf (дата обращения: 20.08.2022).
7. Intel Corp. The Intel Converged Security and Management Engine (CSME) Delayed Authentication Mode (DAM) vulnerability – CVE-2018-3659 and CVE-2018-3643. 2020.

URL: https://www.intel.com/content/dam/www/public/us/en/security-advisory/documents/the-intel-csme-dam-vulnerability-cve-2018-3659-and-cve-2018-3643-whitepaper.pdf (дата обращения: 20.08.2022).
8. Positive Research. IntelTXE-PoC. 2020. URL: https://github.com/ptresearch/IntelTXE-PoC (дата обращения: 20.08.2022).
9. Ermolov Mark, Sklyarov Dmitry, Goryachy Maxim. Undocumented x86 instructions to control the CPU at the microarchitecture level in modern Intel processors. 2022. DOI: http://dx.doi.org/10.1007/s11416-022-00438-x.
10. Intel Corp. Intel 64 and IA-32 Architectures Software Developer's Manual. Vol 3. Chap 2.2. Modes of Operation. 2022. URL: https://software.intel.com/content/dam/develop/external/us/en/documents-tps/325462-sdm-vol-1-2abcd-3abcd.pdf#page=2881 (дата обращения: 20.08.2022).
11. Intel Corp. Intel 64 and IA-32 Architectures Software Developer's Manual. Vol 3. Chap 23.3. Introduction to VMX Operation. 2022. URL: https://software.intel.com/content/dam/develop/external/us/en/documents-tps/325462-sdm-vol-1-2abcd-3abcd.pdf#page=3733 (дата обращения: 20.08.2022).
12. Intel Corp. Intel 64 and IA-32 Architectures Software Developer's Manual. Vol 3. Chap 33.1 Introduction to Intel Software Guard Extensions. 2022. URL: https://software.intel.com/content/dam/develop/external/us/en/documents-tps/325462-sdm-vol-1-2abcd-3abcd.pdf#page=4035 (дата обращения: 20.08.2022).
13. IEEE 1149.1 Working Group Official Webpage. 2021. URL: https://grouper.ieee.org/groups/1149/1 (дата обращения: 20.08.2022).
14. Ucode Research Team. Ucode Disassembler. 2021. URL: https://github.com/chip-red-pill/uCodeDisasm (дата обращения: 20.08.2022).
15. Easdon C. Undocumented CPU Behavior: Analyzing Undocumented Opcodes on Intel x86-64. 2020. URL: https://www.cattius.com/images/undocumented-cpu-behavior.pdf#page=17 (дата обращения: 20.08.2022).
16. Ucode Research Team. Microarchitecture Debug PoC. URL: https://github.com/chip-red-pill/udbgInstr/tree/main/udebug (дата обращения: 20.08.2022).
17. Boluk C. Speculating The Entire X86-64 Instruction Set In Seconds With This One Weird Trick. 2021. URL: https://blog.can.ac/2021/03/22/speculating-x86-64-isa-with-one-weird-trick (дата обращения: 20.08.2022).
18. Ucode Research Team. Ucode Disassembler. The Structure and the Binary Format of Intel Atom Goldmont Microcode. 2021. URL: https://github.com/chip-red-pill/uCodeDisasm#the-structure-and-the-binary-format-of-intel-atom-goldmont-microcode (дата обращения: 20.08.2022).

REFERENCES:

[1] Intel Corp. Intel 64 and IA-32 Architectures Software Developer's Manual. 2021. URL: https://software.intel.com/content/dam/develop/external/us/en/documents-tps/325462-sdm-vol-1-2abcd-3abcd.pdf (accessed: 20.08.2022).
[2] Easdon C. Undocumented CPU Behavior: Analyzing Undocumented Opcodes on Intel x86-64. 2020. URL: https://www.cattius.com/images/undocumented-cpu-behavior.pdf (accessed: 20.08.2022).
[3] Collins R. Undocumented OpCodes. 1995. URL: http://www.rcollins.org/secrets/OpCodes.html (accessed: 20.08.2022).
[4] Domas C. Breaking the x86 ISA. 2017. URL: https://www.blackhat.com/docs/us-17/thursday/us-17-Domas-Breaking-The-x86-Instruction-Set-wp.pdf (accessed: 20.08.2022).
[5] Intel Corp. Intel Converged Security and Management Engine (IntelR CSME) [Security White Paper].2020. URL: https://www.intel.com/content/dam/www/public/us/en/security-advisory/documents/intel-csme-security-white-paper.pdf (accessed: 20.08.2022).
[6] Ermolov M., Goryachy M. How to Hack a Turned-Off Computer, or Running Unsigned Code in Intel Management Engine. 2017. URL: https://www.blackhat.com/docs/eu-17/materials/eu-17-Goryachy-How-To-Hack-A-Turned-Off-Computer-Or-Running-Unsigned-Code-In-Intel-Management-Engine.pdf (accessed: 20.08.2022).
[7] Intel Corp. The Intel Converged Security and Management Engine (CSME) Delayed Authentication Mode (DAM) vulnerability – CVE-2018-3659 and CVE-2018-3643. 2020. URL: https://www.intel.com/content/dam/www/public/us/en/security-advisory/documents/the-intel-csme-dam-vulnerability-cve-2018-3659-and-cve-2018-3643-whitepaper.pdf (accessed: 20.08.2022).
[8] Positive Research. IntelTXE-PoC. 2020. URL: https://github.com/ptresearch/IntelTXE-PoC (accessed: 20.08.2022).
[9] Mark Ermolov, Dmitry Sklyarov, Maxim Goryachy. Undocumented x86 instructions to control the CPU at the microarchitecture level in modern Intel processors. 2022. DOI: http://dx.doi.org/10.1007/s11416-022-00438-x.

[10] Intel Corp. Intel 64 and IA-32 Architectures Software Developer's Manual. Vol 3. Chap 2.2. Modes of Operation. 2022. URL: https://software.intel.com/content/dam/develop/external/us/en/documents-tps/325462-sdm-vol-1-2abcd-3abcd.pdf#page=2881 (accessed: 20.08.2022).
[11] Intel Corp. Intel 64 and IA-32 Architectures Software Developer's Manual. Vol 3. Chap 23.3. Introduction to VMX Operation. 2022. URL: https://software.intel.com/content/dam/develop/external/us/en/documents-tps/325462-sdm-vol-1-2abcd-3abcd.pdf#page=3733 (accessed: 20.08.2022).
[12] Intel Corp. Intel 64 and IA-32 Architectures Software Developer's Manual. Vol 3. Chap 33.1 Introduction to Intel Software Guard Extensions. 2022. URL: https://software.intel.com/content/dam/develop/external/us/en/documents-tps/325462-sdm-vol-1-2abcd-3abcd.pdf#page=4035 (accessed: 20.08.2022).
[13] IEEE 1149.1 Working Group Official Webpage. 2021. URL: https://grouper.ieee.org/groups/1149/1 (accessed: 20.08.2022).
[14] Ucode Research Team. Ucode Disassembler. 2021. URL: https://github.com/chip-red-pill/uCodeDisasm (accessed: 20.08.2022).
[15] Easdon C. Undocumented CPU Behavior: Analyzing Undocumented Opcodes on Intel x86-64. 2020. URL: https://www.cattius.com/images/undocumented-cpu-behavior.pdf#page=17 (accessed: 20.08.2022).
[16] Ucode Research Team. Microarchitecture Debug PoC. URL: https://github.com/chip-red-pill/udbgInstr/tree/main/udebug (accessed: 20.08.2022).
[17] Boluk C. Speculating The Entire X86-64 Instruction Set In Seconds With This One Weird Trick. 2021. URL: https://blog.can.ac/2021/03/22/speculating-x86-64-isa-with-one-weird-trick (accessed: 20.08.2022).
[18] Ucode Research Team. Ucode Disassembler. The Structure and the Binary Format of Intel Atom Goldmont Microcode. 2021. URL: https://github.com/chip-red-pill/uCodeDisasm#the-structure-and-the-binary-format-of-intel-atom-goldmont-microcode (accessed: 20.08.2022).

Поступила в редакцию – 29 сентября 2022 г. Окончательный вариант – 20 октября 2022 г.
Received – September 29, 2022. The final version – October 20, 2022.