УДК 004.056.57

МЕТОД МАСКИРОВКИ ВИРТУАЛЬНОЙ МАШИНЫ ОТ ЕЁ ОБНАРУЖЕНИЯ ВРЕДОНОСНЫМ ПО

Вишневская Татьяна Ивановна,
канд. физ.-мат. наук, доцент кафедры программного обеспечения ЭВМ и ИТ,
e-mail: iu7vt@bmstu.ru,
Московский государственный технический университет (МГТУ им. Н.Э. Баумана), г. Москва, Россия,

Макаренко Олег Константинович,
студент магистратуры кафедры программного обеспечения ЭВМ и ИТ,
e-mail: jktu2308@gmail.com,
МГТУ им. Н.Э. Баумана, г. Москва, Россия

В работе представлен метод маскировки виртуальной машины, запущенной с использованием систем виртуализации QEMU и KVM. Целью работы является создание метода маскировки виртуальной машины, при использовании которого обнаружение виртуальной машины будет усложнено. Показывается, что затраты процессорного времени на выход и возврат в систему виртуализации являются наиболее характерным признаком, позволяющим вредоносной программе обнаружить использование виртуальной машины. Предложен метод маскировки виртуальной машины на основе подсчёта числа реально прошедших тиков процессора хоста и его замены при возврате в виртуальную машину. Описаны алгоритмы обхода проверок на наличие виртуального окружения. Дано описание результата работы метода маскировки. Продемонстрирована целесообразность использования предложенного метода. Описаны возможные сценарии применения данного метода. Предложенный метод маскировки виртуального окружения может быть использован при анализе поведения вредоносного программного обеспечения. В результате применения разработанного метода тестовые программы не смогли обнаружить наличие виртуального окружения. Полученные результаты исследования будут полезны разработчикам средств анализа вредоносного программного обеспечения.

Ключевые слова: информационная безопасность, маскировка, QEMU, KVM, виртуальная машина, вредоносное ПО

METHOD FOR MASKING A VM FROM BEING DETECTED BY MALICIOUS SOFTWARE

Vishnevskaya T.I.,
PhD physical and mathematical sciences, Associate Professor,
e-mail: iu7vt@bmstu.ru,
Bauman Moscow State Technical University (BMSTU), Moscow, Russia,

Makarenko O.K.,
master's degree student,
e-mail: jktu2308@gmail.com,
Bauman Moscow State Technical University (BMSTU), Moscow, Russia

The article a method of masking for a virtual machine, which is launched utilizing virtualization systems QEMU and KVM are proposed. The aim of the work is to create a method for masking a virtual machine, using which the detection of a virtual machine will be complicated.

It is shown that the cost of CPU time to exit and return to the virtualization system is the most characteristic feature that allows malware to detect the use of a VM. A method for masking a VM is proposed based on counting the number of actually passed ticks of the host processor and replacing it when returning to the VM. Algorithms of evading tests for virtual environment presence are described. The result of the masking method are presented. The relevance of the proposed method implementation is given. Possible application scenarios for the method

are also described. The proposed method of masking the virtual environment can be used to analyze the behavior of malware. As a result of applying the developed masking method, programs could not detect the presence of a virtual environment. The obtained research results will be useful for developers of malware analysis tools.

Keywords: information security, masking, QEMU, KVM, virtual machine, malware

DOI 10.21777/2500-2112-2020-2-48-57

Введение

В настоящее время анализ вредоносного программного обеспечения (ПО) не обходится без использования автоматических средств анализа действий программ. Одним из возможных способов обнаружения опасного ПО является выполнение программы в виртуальных машинах. Для усложнения анализа собственных программ разработчик вредоносного ПО может встроить в программу проверку на то, где она запускается. Если программа обнаруживает виртуальное окружение, она сразу же прекращает своё действие, или выполняет какое-то другое действие. Если проверка показала наличие реальной машины, то программа начинает выполнять вредоносные действия.

Системы виртуализации KVM [7] и QEMU позволяют эмулировать инструкции виртуального процессора на реальном. На таком уровне возможно подменять ответы на каждую инструкцию таким образом, чтобы скрыть факт наличия виртуального окружения. Но при таком режиме производительность виртуальной машины достаточно сильно снижается. Для увеличения производительности виртуальных машин используется технология аппаратного ускорения. У Intel технология имеет название VT-x [2].

Целью данной работы является создание метода маскировки виртуального окружения, который позволит скрывать виртуальную машину, работающую в режиме аппаратного ускорения. Этот метод позволит антивирусным программам значительно ускорить анализ вредоносного ПО.

Научная новизна работы заключается в том, что в работе предложены ранее не реализованные методы обхода проверок на выходы из виртуальной машины, также работающие в режиме аппаратного ускорения.

1. Способы обнаружения виртуального окружения

Существует достаточно много методов проверки, находится ли исполняемая программа в виртуальном окружении или нет. В качестве примеров программ, проверяющих разными способами виртуальное окружение, были взяты специальные инструменты, доступные в публичных репозиториях. Данные программы запускают несколько проверок и на выходе отдают подробный лог файл с результатами проверки.

В репозиториях, упомянутых выше, находятся программы AlKhaser [6] и paFish [8]. Эти программы разработаны под ОС Windows и содержат в себе достаточно большой список проверок системы на факт запуска в виртуальной машине. В обеих программах они разделены по категориям. Выделим категории, которые относятся к обнаружению виртуальной машины, опустив такие проверки как, например, запуск под отладчиком, проверка инъекций и тому подобных, так как они не относятся к обнаружению виртуального окружения. К основным проверкам системы на факт запуска в виртуальной машине можно отнести следующие:

1. Проверки таймера машины:
− проверка на выход из виртуальной машины при вызове RDTSC;
− проверка на выход из виртуальной машины при вызове CPUID.
2. Анти-виртуализация:
− проверки на названия подключенных устройств;
− проверки на PNP ID подключенных устройств.
3. Проверка BIOS:
− проверки на названия гипервизора в ACPI таблицах;

– проверка возможных состояний сна машины;
– проверка наличия датчика температуры материнской платы.

2. Алгоритмы проверок на выход из виртуальной машины

Во время аппаратной виртуализации реальный процессор выполняет команды от виртуальной машины напрямую, без их эмуляций в виде KVM или QEMU. Процессору необходимо предоставить набор регистров виртуальной машины. В процессорах Intel для входа в такой режим реализована команда VM_RESUME [4].

В режиме аппаратной виртуализации предусмотрен выход из режима виртуализации для эмулирования таких команд, как CPUID, RDTSC, RDMSR, WRMSR, и др. Если на виртуальной машине была выполнена одна из таких команд, виртуальный процессор выходит из режима виртуализации для того, чтобы система виртуализации вернула в регистры виртуального процессора необходимые данные. Это может занять достаточно много времени, гораздо больше, чем на обычной, не виртуальной машине. На рисунке 1 представлена схема передачи управления между уровнями виртуализации во время выполнения инструкции CPUID.

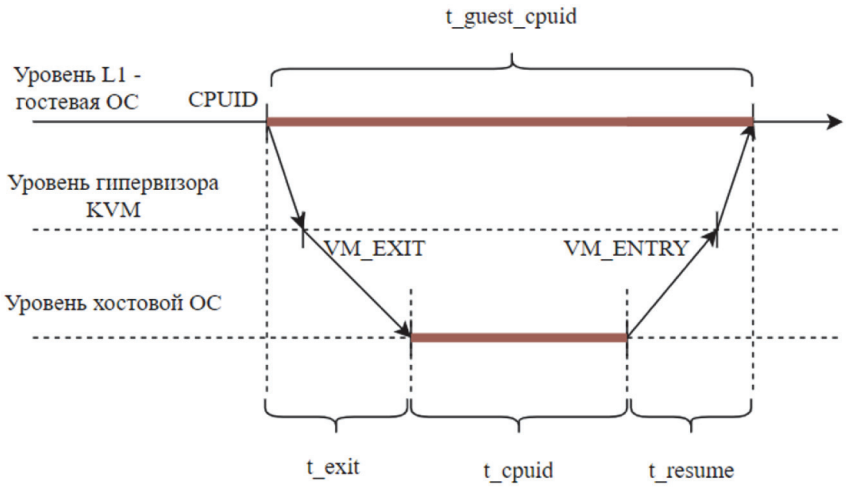

Рисунок 1 – Схема передачи управления между уровнями виртуализации

На рисунке 1 приняты следующие обозначения: t_exit – время, затраченное на выход из виртуальной машины, t_cpuid – время, затраченное на команду cpuid, t_resume – время, затраченное на возврат управления виртуальной машины, t_guest_cpuid – время, затраченное на выполнение команды cpuid на виртуальной машине. Как видно из рисунка 1, большое количество времени тратится именно на выход и возврат в систему виртуализации. На этом и строится проверка на выход из виртуальной машины при вызове CPUID и RDTSC [3]. На рисунке 2 представлены алгоритмы проверок наличия виртуальной машины, использующиеся вредоносным ПО.

Проверки на названия подключенных устройств заключаются в простом поиске среди всех подключенных к машине устройств и их названий, содержащих названия известных систем виртуализации, таких как “QEMU”, “KVM”, “BOCHS”, “VirtualBox” и т.п.

Системы виртуализации при запуске эмулирует BIOS самостоятельно. В зависимости от выбранной конфигурации эмулируемой системы генерируются ACPI (Advanced Configuration and Power Interface) таблицы, в которых расположены описания всех устройств, поддерживаемых компьютером [5]. В реальной машине в ACPI таблицах расположена информация о датчиках материнской платы, о подключенном процессоре, о производителе материнской платы. Система виртуализации, генерируя данные таблицы, оставляет свои названия в них. В настоящее время все реальные машины поддерживают так называемый «спящий режим», или же режимы S3 или S4 процессора. При генерации BIOS-а

QEMU не включает поддержку этих состояний. Эти состояния могут быть получены из операционной системы (ОС) и программы могут это проверять.

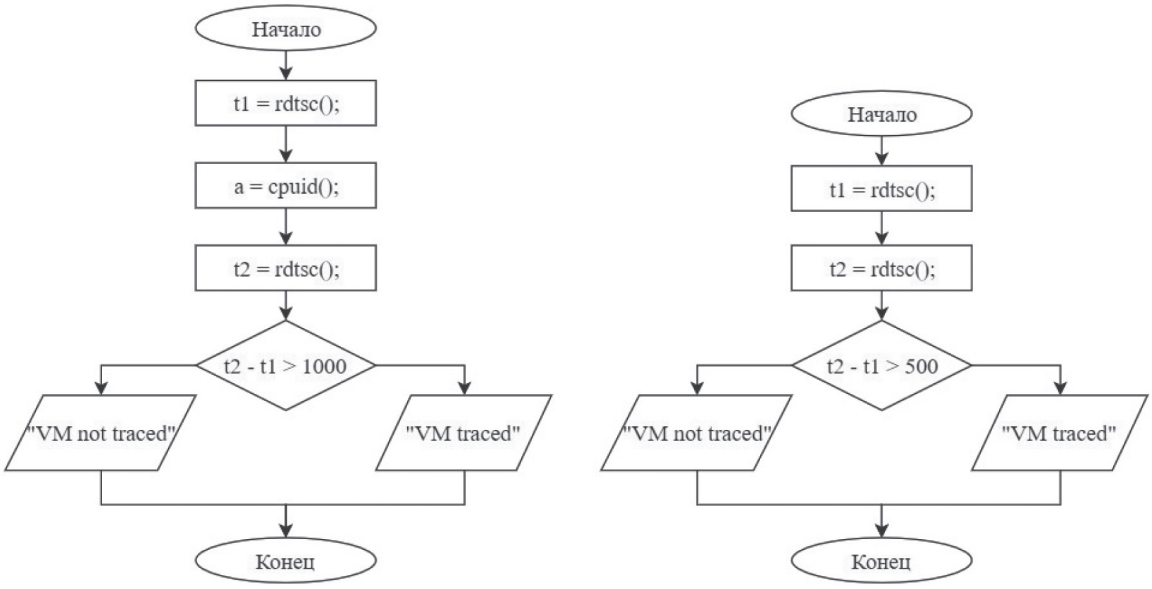

Рисунок 2 – Алгоритмы проверок на выход из виртуальной машины

Также на реальных машинах присутствует датчик температуры, так называемая «Термальная зона». В неё включена система охлаждения процессора. Состояние термальной зоны доступно ОС, её тоже могут проверять программы. QEMU не имеет поддержки термальной зоны.

На процессорах Intel присутствует встроенный датчик температуры. Его данные могут быть получены командой чтения регистров MSR (Model-Specific Registers), т.е. могут быть доступны всем желающим. Если происходит ошибка чтения данных регистров, то это может свидетельствовать об эмуляции используемого процессора.

3. Алгоритм обхода проверок на выход из виртуальной машины

Для обхода проверки на выход из виртуальной машины был разработан алгоритм подсчёта реально прошедших тиков и его замены при возврате в виртуальную машину. В виртуальных машинах с использованием аппаратного ускорения счётчик тиков можно получить двумя способами: через команду RDTSC и через команду RDMSR(0x10) [1]. В виртуальных машинах он эмулируется таким образом: во время запуска виртуальной машины запоминается количество тиков процессора хоста. После этого во время выполнения команды RDTSC в виртуальной машине выполняется команда RDTSC на машине хоста и разница между количеством тиков при старте машины и количеством тиков в текущий момент возвращается как результат выполнения на виртуальной машине. Причем в Intel VMX для этого не требуется VMEXIT, поэтому проверка на выход при RDTSC показывает, что выхода не происходит.

Во время выполнения команды CPUID происходит передача управления Монитору виртуальных машин гипервизора по инструкции VMEXIT. На выход из машины и вход обратно требуется достаточно много времени, около 2 тысяч тиков. На возврат эмулируемого значения CPUID времени нужно примерно ещё столько же, 3-4 тысяч тиков. На реальной машине время, затраченное на выполнение команды CPUID, не превышает ~700 тиков.

Суть алгоритма обхода проверки заключается в увеличении значения запомненного при запуске значения счётчиков тиков процессора на количество затраченных тиков на эмуляцию CPUID. Схема разработанного алгоритма представлена на рисунке 3.

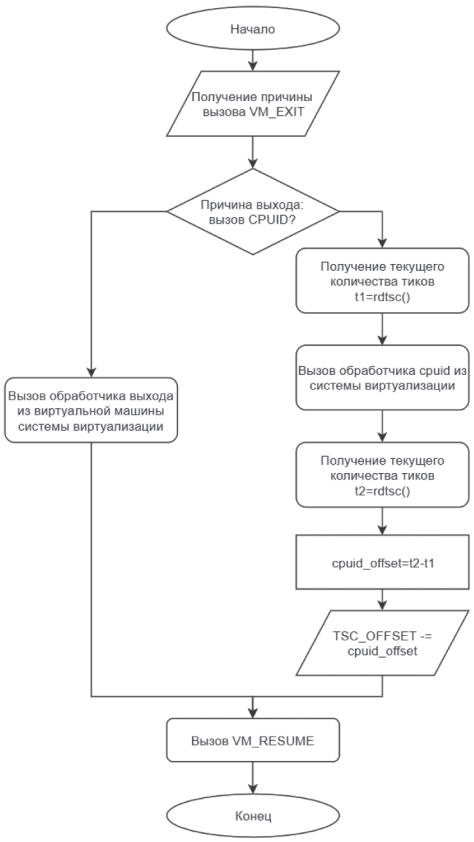

Рисунок 3 – Алгоритм маскировки количества тиков при вызове CPUID

4. Обход проверок на названия и идентификаторы подключенных устройств

Для проверок названий подключенных устройств достаточно изменить все вхождения строк «QEMU», и т.п. в исходниках генерации устройств и ACPI таблиц в QEMU (Quick Emulator – эмулятор различных устройств). То же самое касается сборки BIOS. По умолчанию BIOS собирается с определёнными именами производителей. Также в BIOS, а именно в SMBIOS, байте расширений характеристик, присутствует бит, показывающий, был ли собран BIOS для виртуальной машины, или нет. Этот бит тоже необходимо убрать.

5. Обход проверок на температуру виртуального процессора

Для добавления термальной зоны в ACPI необходимо было реализовать добавление термальной зоны в виртуальные ACPI таблицы в соответствии со спецификациями ACPI. В спецификации указано, что термальная зона описана в структуре с типом 0x85. В исходниках QEMU была генерация похожих структур с другими ID. На их основе был написан метод определения термальной зоны и генератор значений температуры процессора (листинг 1).

В спецификации ACPI описываются требования к написанию собственных ACPI таблиц [8]. Для элемента термальной зоны обязательно нужно описать метод «_TMP», возвращающий текущую температуру термальной зоны. Нам достаточно возвращать здесь какую-либо константу. Также в описании термальной зоны обязательно необходимо описать метод «_CRT», выходными данными которого является критическая температура для процессора. Выходными данными метода «_STR» является строка с описанием термальной зоны.

```
/* DefThermalZone */
Aml *aml_thermalzone(const char *name_format, ...)
{
 va_list ap;
 Aml    *var    =    aml_bundle(0x85    /*    ThermalZoneOp    */,
AML_EXT_PACKAGE);
 va_start(ap, name_format);
 build_append_namestringv(var->buf, name_format, ap);
 va_end(ap);
 return var;
}
```

Листинг 1 – Реализованный генератор термальной зоны ACPI на языке AML

Термальная зона ACPI добавляется к каждому виртуальному процессору отдельно. Реализованное добавление термальной зоны показано в листингах 1 и 2.

```
 Aml* deviceEC;
 deviceEC=aml_device("EC0");
 (deviceEC,  aml_name_decl("_HID",  aml_string("PNP0C09")));  //
PNP0C09

 Aml* tz;

 tz = aml_thermalzone("TZ%d", tzcnt);
 method = aml_method("_TMP", 0, AML_SERIALIZED);
 aml_append(method,
 aml_return(aml_int(0xB73))
 );
 aml_append(tz, method);

 method = aml_method("_CRT", 0, AML_SERIALIZED);
 aml_append(method,
 aml_return(aml_int(0xE93))
 );
 aml_append(tz, method);

 aml_append(tz, aml_name_decl("_STR", aml_string("System thermal
zone (20)")));
```

Листинг 2 – Добавление термальной зоны ACPI

6. Результат работы разработанного метода

С использованием изменённых систем виртуализации с реализованными алгоритмами маскировки, программы, проверяющие факт наличия виртуального окружения, более не смогли определить факт запуска на виртуальной машине. На стабильность работы системы виртуализации внесённые из-

менения не повлияли. На листингах 3, 4 показан вывод программы pafish, определяющей виртуальное окружение различными проверками на не изменённой и на изменённой системе виртуализации соответственно. Данная программа использует проверки, описанные выше в данной статье. При получении среднего значения количества тиков, затрачиваемых на команду CPUID больше 1000, программа выводит «TRACED» в соответствующей проверке.

```
 Patfish (Paranoid fish) *

Some anti(debugger/VM/sandbox) tricks
Used by malware for the general public.

[*] Windows version: 10.0 build 17134
[*] CPU: GenuineIntel
 CPU brand: Intel Core Processor (Broadwell)

[-] Debuggers detection
[*] Using IsDebuggerPresent() ... OK

[-] CPU information based detections
[*]  Checking  the  difference  between  CPU  timestamp  counters
(rdtsc) ... OK
[*]  Checking  the  difference  between  CPU  timestamp  counters
(rdtsc) forcing VM exit ... OK
[*]  Checking  the  difference  between  CPU  timestamp  counters
(rdtsc) forcing VM exit for 4 instructions cpuid ... traced!
[*]  Checking  the  difference  between  CPU  timestamp  counters
(rdtsc) forcing VM exit for 1000 instructions cpuid ... traced!
[*] Checking hypervisor bit in cpuid feature bits ... traced!
[*]  Checking  cpuid  hypervisor  vendor  for  known  VM  vendors  ...
traced!
```

Листинг 3 – Результат работы программы pafish на неизменённой системе виртуализации

```
 Patfish (Paranoid fish) *

Some anti(debugger/VM/sandbox) tricks
Used by malware for the general public.

[*] Windows version: 10.0 build 17134
[*] CPU: GenuineIntel
 CPU brand: Intel Core Processor (Broadwell)

[-] Debuggers detection
[*] Using IsDebuggerPresent() ... OK
```

Листинг 4 – Результат работы программы pafish на изменённой системе виртуализации (начало)

```
[-] CPU information based detections
[*] Checking the difference between CPU timestamp counters
(rdtsc) ... OK
[*] Checking the difference between CPU timestamp counters
(rdtsc) forcing VM exit ... OK
[*] Checking the difference between CPU timestamp counters
(rdtsc) forcing VM exit for 4 instructions cpuid ... OK
[*] Checking the difference between CPU timestamp counters
(rdtsc) forcing VM exit for 1000 instructions cpuid ... OK
[*] Checking hypervisor bit in cpuid feature bits ... OK
[*] Checking cpuid hypervisor vendor for known VM vendors ... OK
```

Листинг 4 – Результат работы программы pafish на изменённой системе виртуализации (окончание)

На рисунке 4 представлены замеры времени выполнения команд без маскировки. На левой диаграмме представлены результаты замеров на реальной машине – хорошо видно, что на реальной машине количество тиков не превышает значения в 300 тиков. На правой диаграмме – результаты, полученные с незамаскированной виртуальной машины. Хорошо видно, что время, затраченное на CPUID, в несколько раз превышает аналогичное на реальной машине. Время выполнения инструкции RDTSC не увеличилось, т.к. при RDTSC не происходит т.н. «выхода».

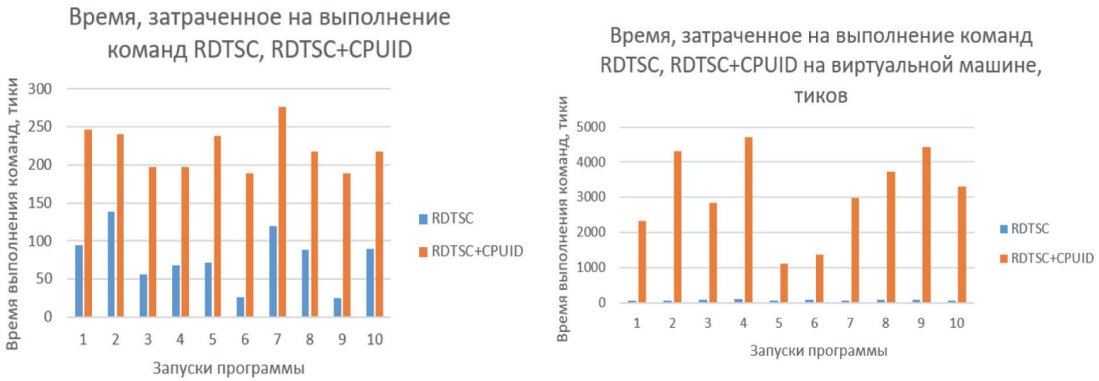

Рисунок 4 – Результаты замеров времени без маскировки

На рисунке 5 представлены результаты замеров времени выполнения команд с маскировкой тиков на изменённой виртуальной машине. Из диаграммы видно, что время выполнения CPUID заметно снизилось – виртуальную машину с такой маскировкой теперь сложнее будет обнаружить при помощи замера времени.

Заключение

Анализ существующих методов обнаружения виртуального окружения показал, что есть достаточное количество методов, применяющихся вредоносным ПО для обнаружения среды виртуализации и дальнейшего изменения собственного поведения. В данной работе предложен метод создания среды виртуализации, позволяющей обойти большинство применяющихся проверок. Анализ результатов работы метода показал успешный обход проверок модели процессора, наличия сенсоров температуры, наличия виртуальной периферии. Замеры времени выполнения инструкций CPUID показали уменьшение подсчета количества тиков во время выхода из виртуальной машины. При реализации метода

маскировки был сохранён основной функционал систем виртуализации, в том числе, и наличие аппаратного ускорения.

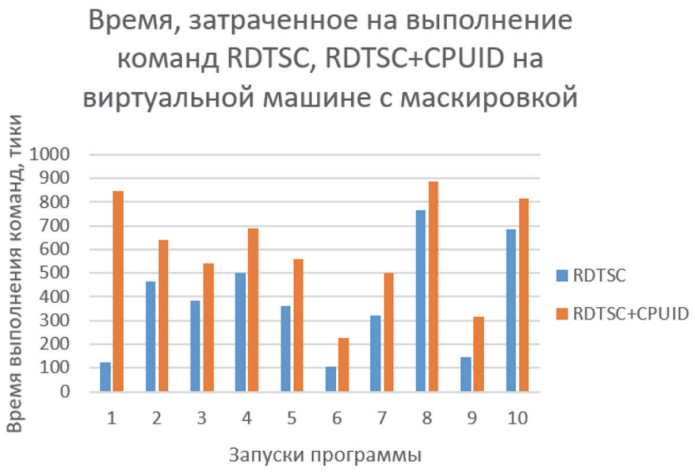

Рисунок 5 – Результаты замеров времени с маскировкой

Список литературы

1. *Гилязов Р.Р.* Оценка времени, прошедшего между двумя событиями, в операционной системе // Вестник РГГУ. Серия: Документоведение и архивоведение. Информатика. Защита информации и информационная безопасность // Российский государственный гуманитарный университет. – 2011. – № 13. – С. 171–181.

2. *Карпов И.В., Егоров В.Ю.* Применение технологий виртуализации Intel Vt-x и Intel Vt-d для повышения защищенности и надежности функционирования рабочей станции // Наукоемкие Технологии // Издательство «Радиотехника». – 2010. – Т. 11. – №4. – С. 46–54.

3. *Коркин И.Ю., Петрова Т.В., Тихонов А.Ю.* Метод обнаружения аппаратной виртуализации в компьютерных системах // Безопасность Информационных Технологий // КлАССное снаряжение. – 2010. – Т. 17. – № 1. – С. 80–82.

4. *Юлюгин Е.А.* Корректное и быстрое исполнение отдельных инструкций архитектуры INTEL® 64 в виртуальном окружении. // Информационные Технологии // Издательство «Новые технологии». – 2019. – Т. 25. – № 3. – С. 157–164.

5. Advanced Configuration and Power Interface Specification [Электронный ресурс]. 2017. – URL: https://uefi.org/sites/default/files/resources/ACPI_6_2.pdf (дата обращения 10.06.2020).

6. Al Khaser [Электронный ресурс]. – 2019. – URL: https://github.com/LordNoteworthy/al-khaser (дата обращения 10.06.2020).

7. *Liu D., Zhang Y.Y.* A research on KVM-based virtualization security // Applied mechanics and materials. – 2014. – Т. 543–547. – С. 3126–3129.

8. PaFish [Электронный ресурс]. 2019. – URL: https://github.com/a0rtega/pafish (дата обращения 10.06.2020).

References

1. *Gilyazon R.R.* Otsenka vremeni, proshedshego mezhdu dvumya sobytiyami, v operatsionnoy sisteme // Vestnik RGGU, Seriya: Dokumentovedeniye i arkhivovedeniye. Informatika. Zashchita informatsii i informatsionnaya bezopasnost’ // Rossiyskiy gosudarstvennyy gumanitarnyy universitet. – 2011. – № 13. – S. 171–181.

2. *Karpov I.V., Yegorov V.Yu.* Primeneniye tekhnologiy virtualizatsii Intel Vt-x i Intel Vt-d dlya povysheniya zashchishchennosti i nadezhnosti funktsionirovaniya rabochey stantsii // Naukoyemkiye Tekhnologii // Izdatel’stvo “Radiotekhnika”. – 2010. – T. 11. – № 4. – S. 46–54.

3. *Korkin I.Yu., Petrova T.V., Tikhonov A.Yu.* Metod obnaruzheniya apparatnoy virtualizatsii v komp'yuternykh sistemakh // Bezopasnost' Informatsionnykh Tekhnologiy // KlASSnoye snaryazheniye. – 2010. – T. 17. – № 1. – S. 80–82.
4. *Yulyugin Ye.A.* Korrektnoye i bystroye ispolneniye otdel'nykh instruktsiy arkhitektury INTEL® 64 v virtual'nom okruzhenii. // Informatsionnyye Tekhnologii // Izdatel'stvo "Novyye tekhnologii". – 2019. – T. 25. – № 3. – S. 157–164.
5. Advanced Configuration and Power Interface Specification [Elektronnyij resurs]. – 2017. – URL: https://uefi.org/sites/default/files/resources/ACPI_6_2.pdf (data obrasheniya 10.06.2020).
6. Al Khaser [Elektronnyij resurs]. – 2019. – URL: https://github.com/LordNoteworthy/al-khaser (data obrasheniya 10.06.2020).
7. *Liu D., Zhang Y.Y.* A research on KVM-based virtualization security // Applied mechanics and materials. – 2014. – T. 543–547. – C. 3126–3129.
8. PaFish [Elektronnyij resurs]. – 2019. – URL: https://github.com/a0rtega/pafish, (data obrasheniya 10.06.2020).