# СИСТЕМА ЗАЩИТЫ ТЕРМИНАЛЬНЫХ ПРОГРАММ ОТ АНАЛИЗА НА ОСНОВЕ ВИРТУАЛИЗАЦИИ ИСПОЛНЯЕМОГО КОДА

*Маркин Д. О.[1], Макеев С. М.[2]*

***Целью работы*** *является повышение защищенности терминальных программ от анализа и восстановления алгоритма их функционирования.*

***Метод.*** *Предлагаемое решение основано на применении технологии виртуальных машин с секретной архитектурой. Под архитектурой виртуальной машины в данной работе понимаются используемый алфавит байт-кода и функциональные объекты, используемые в защищаемых терминальных программах.*

***Результат.*** *В работе предложена система защиты активных данных в распределенной вычислительной системе. Проведен анализ существующих технических решений, выделена научная проблема, заключающаяся в необходимости эффективной защиты терминальных программ от анализа и восстановления алгоритмов функционирования. Объектом защиты в представленной работе являются терминальные программы или активные данные, функционирующие в узлах распределенной вычислительной системе.*

*Разработаны и описаны модели угроз и нарушителя безопасности программного обеспечения. Сформированы требования на основе проведенного анализа условий функционирования, а также моделей угроз и нарушителя безопасности. Предложена математическая модель системы защиты активных данных от анализа на основе виртуализации исполняемого кода и ее программная реализация в виде приложений на C/C++ и Java. Экспериментально проверена эффективность системы по частным показателям качества (оперативности) предложенной системы, подтверждающая ее результативность. Предложенное решение позволяет эффективно затруднять анализ и восстановление алгоритмов функционирования терминальных программ в распределенной вычислительной системе.*

***Ключевые слова:*** *система защиты информации, активные данные, методы обфускации, виртуальная машина, байт-код.*

**DOI:**10.21681/2311-3456-2020-01-29-41

## Введение

Недостаточная защищенность программного обеспечения от анализа, нарушения его работоспособности, модификации и других угроз может привести к серьезным неблагоприятным последствиям различного характера и масштаба. Таким образом, проблема обеспечения защищенности программного обеспечения (ПО) является актуальной и требует применения новых методов и средств, способных обеспечить требуемый уровень защиты в современных условиях.

Проблема защиты программных реализаций на основе применения технологий виртуализации и других методов защиты затрагивалась в научных трудах Аранова В. Ю., Петрова А. С., Петрова А. А. [1–3], Речистова Г. С.[4], а также Казарина О. В., Варнавского Н. П., Захарова В. А., Кузюрина Н. Н. и Шокурова А. В. [5, 6], Зегжды П. Д., Бойко В. П., Заборовского В. С., Подловченко Р. И., Иванникова В. П., Сомборсона К., Викстромома Д. и других. В указанных работах отмечалась необходимость защиты программных реализаций от технологий обратной разработки, основанных, в том числе на методах статического и динамического анализа ПО.

В ряде автоматизированных систем ПО доступно исследователям в виде исходного или байт-кода. Подобные системы используют интерпретаторы либо виртуальные машины для исполнения кода. Защита такого типа ПО от анализа требует применения подходов, обеспечивающих максимальное затруднение анализа исходных текстов – т. е. применения методов обфускации [6].

Одним из наиболее эффективных способов защиты исходного текста ПО от анализа, не исключающего применения других методов, является внедрение технологии виртуализации кода. Такой подход позволяет сформировать ПО, состоящее архитектурно из двух компонентов – виртуальной машины/интерпретатора или их множества и байт-кода. При этом в зависимости от наличия лицензии у пользователя (права на использование ПО) исполняемый байт-код может преобразовываться к виду, который сможет быть обработан встроенной виртуальной машиной или

1 Маркин Дмитрий Олегович, кандидат технических наук, сотрудник Академии ФСО России, г. Орёл, Россия. E-mail: admin@nikitka.net

2 Макеев Сергей Михайлович, кандидат технических наук, сотрудник Академии ФСО России, г. Орёл, Россия. E-mail: maksm57@yandex.ru

интерпретатором без ошибок. Данный подход описан в работах [7 , 8 ].

**Объект защиты**

В данной работе объектом защиты являются терминальные программы, передаваемые между узлами системы, обеспечивающей их исполнение на данных узлах [9 , 10 , 14 ]. Для защиты от анализа исполняемого кода таких терминальных программ в данной работе предлагается система на основе виртуализации исполняемого кода с применением луковичной и чесночной архитектуры.

*Терминальные программы или активные данные* – это исполняемый код в виде исходного текста или байт-кода, передаваемые на вход ПО, способный их выполнить.

Примерами активных данных являются:

- самораспаковывающийся архивный файл;
- приложения, разработанные на скриптовых языках программирования и передаваемые удаленной среде в качестве данных;
- специализированная клиент-серверная система передачи активных данных, в том числе на основе виртуальных сред исполнения.

Преимущества использования технологии активных данных:

- значительное снижение объема исполняемого кода клиентских приложений;
- возможность исполнения на клиентских приложениях произвольного кода, т. е. универсальность.

Недостатки использования технологии активных данных:

- необходимость обеспечения аутентичности кода, защиты клиентского приложения от анализа, авторизации активных данных, т. е. защиты от компрометации клиентского приложения;
- необходимость использования интерпретаторов;
- передача кода в виде исходных текстов;
- необходимость диспетчеризации.

Реализация концепции активных данных базируется на применении скриптовых языков программирования, таких как *PHP*, *Perl*, *Python*, *Ruby*, *JavaScript* и других, а также способности приложений, выполняющих роль интерпретатора, обрабатывать содержание данных (например, тела *HTTP(s)*-запросов) как программный код [11 , 12 ]. Иллюстрация такого применения скриптовых языков в контексте реализации концепции активных данных представлена на рисунке 1.

Эффективность такой реализации была проверена в работах [11 , 12 ] на базе сети веб-прокси серверов с помощью ряда разработанных программных средств [15 , 16 ], а также с использованием испытательного стенда на основе МАУ под управлением ОС Android [17 ] и управляющего веб-сервера для решения типовой задачи факторизации [18 ].

Типовой алгоритм обработки активных данных представлен на рисунке 2 .

Для реализации подобного алгоритма требуется поддержка скриптового языка программирования и возможность его вызова из приложения для МАУ с ОС *Android*, *iOS* или другой системы.

Недостатками использования технологии активных данных являются:

- необходимость обеспечения аутентичности кода, защиты клиентского приложения от анализа, авторизации исполняемых активных данных, т. е. защиты от компрометации клиентского приложения;
- необходимость использования интерпретаторов;
- передача кода в виде исходных текстов; необходимость диспетчеризации.

Проблема обеспечения авторизации исполняемого кода известна очень давно, однако в контексте решения задачи построения защищенной системы туманных вычислений, реализующей концепцию активных данных, возникает задача авторизации полученного извне кода не столько самим устройством, сколько программным обеспечением агента системы туманных вычислений в условиях, когда к самому устройству нет доверия.

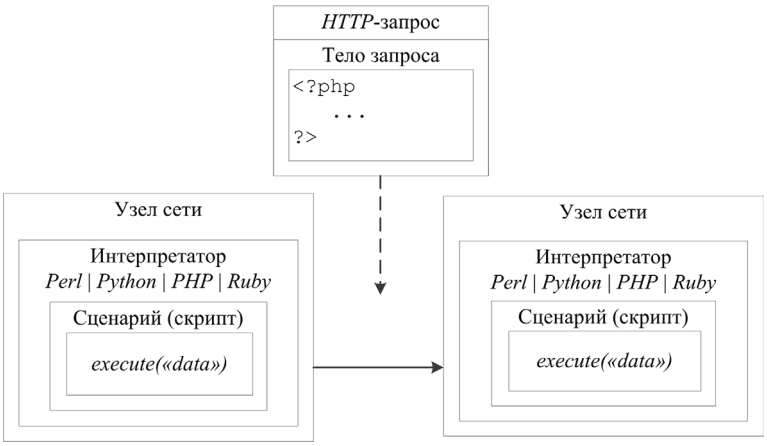


*Рис. 1 . Реализация концепции активных данных на основе применения скриптовых языков*


*Рис. 2 . Алгоритм обработки активных данных*

Данное ограничение предъявляет особые требования к системе защиты приложения, выполняющего активные данные. К ним можно отнести следующее:

- исполнение активных данных возможно только на тех устройствах, на которых имеется соответствующе аутентичное программное обеспечение (иными словами, нелегитимный пользовать не должен иметь возможность пользоваться вычислительными ресурсами системы туманных вычислений);
- анализ ПО агента не должен позволять исследователю (нарушителю) использовать ПО данного агента для решения его вычислительных задач, а также ПО иных агентов системы туманных вычислений (т.е. должна обеспечиваться аутентичность активных данных программному обеспечению агента);
- анализ содержания и структуры активных данных не должен позволять исследователю (нарушителю) формировать собственные активные данные, которые могли бы быть обработаны системой туманных вычислений;
- модификация активных данных, а также ПО агента должно приводить к отказу в доступе к вычислительным ресурсам устройства (т. е. должна обеспечиваться целостность вычислительного процесса и авторизация исполняемого программного кода).

В ряде автоматизированных систем ПО доступно исследователям в виде исходного или байт-кода. Подобные системы используют интерпретаторы либо виртуальные машины для исполнения кода. Защита такого типа ПО от анализа требует применения подходов, обеспечивающих максимальное затруднение анализа исходных текстов – т.е. применения методов обфускации[3] [6 ].

Одним из наиболее эффективных способов защиты исходного текста ПО от анализа, не исключающего применения других методов, является внедрение технологии виртуализации кода [4 , 8 , 7 , 14 ]. Такой подход позволяет сформировать ПО, состоящее архитектурно из двух компонентов – виртуальной машины/интерпретатора или их множества и байт-кода. При этом в зависимости от наличия лицензии у пользователя (права на использование ПО) исполняемый байт-код может преобразовываться к виду, который сможет быть обработан встроенной виртуальной машиной или интерпретатором без ошибок.

В данной работе предлагается усовершенствовать технологию виртуальных машин за счет использования луковичной архитектуры пакета с активными данными, содержащего код для разных агентов, и использования уникальной архитектуры виртуальной машины для каждого из агентов системы туманных вычислений.

### Модель нарушителя безопасности программного обеспечения

Модель нарушителя безопасности ПО предполагает использование всего инструментального комплекса средств анализа ПО, к которым в настоящее время относятся автоматизированные статические средства анализа ПО, основанные на методах компьютерной лингвистики, средства динамического анализа, основанные на методах динамической бинарной инструментации (*DBI – dynamic binary instrumentation*), а также средства фаззинга и профилирования ПО.

Кроме того, нарушитель (исследователь) имеет полный доступ к ресурсам (ЦП, ПЗУ, ОЗУ, устройства ввода/вывода и т. п.), ПО и ОС данной ПЭВМ, на которой реализуется указанная схема исследования; способен создавать копии исследуемого экземпляра ПО агента и активных данных.

Обобщенная схема «атаки» на программную реализацию показана на рисунке 3 .

Наиболее известными средствами исследования ПО являются дизассемблеры и декомпиляторы, отладчики и эмуляторы. Проведенный анализ существующих средств исследования ПО [13 ] показал, что наиболее распространенными дизассемблерами являются: *IDAPro*, *x64dbg*, *Radare 2*, *DeDe*, *Sourcer*, *PeExplorer*, *Cutter*, *Hopper* (*OSx Linux*), *Scratch ABit*, *Plasma*, *D3-28*, *ODA – Online*, *WinHex*, *Hiew*, *DB52ASM*, *W32DASM*, *Binary Ninja*, *Capstone*, *Zydis*, *Objconv*, *HT Editor*, *distorm 64*, *Crudasm*.

Наиболее распространенными отладчиками являются [13 ]: *IDA Pro*, *OllyDbg*, *GDB*, *SoftIce*, *AQtime*, *DBX*, *Dtrace*, *Electric Fence*, *GNU Debugger*, *LLDB*, *MDB*, *MS Visual Studio*, *Immunity Debugger*, *Dr. Watson*, *TotalView*, *WinDbg*, *FlexTracer*, *YDbg*, *x64dbg*, *Trace replayer*, *PIN tracer*.

Эмуляторы: *VMware*, *VirtualBox*, *QEMU*, *Nox*, *Bochs*, *JVM*, *Colinux*, *AlphaVM-Free(-Pro)*, *CHARON-AXP(-VAX)*, *Denali*, *DOSBox*, *DOSEMU*, *Icore virtual accounts*, *Jail*, *KVM*, *OpenVZ*, *Parallels Workstation*, *PearPC*, *Virtual PC*, *Hyper-V*, *Virtuozzo*, *VMware ESX Server*, *SimNow*, *Solaris Zones*, *Xen*, *z/VM*.

Наиболее распространенными декомпиляторами являются: *Boomerang*, *DCC*, *RecStudio*, *Hex-Rays*, *Cavaj Java Decompiler*, *JD*.

3 Обфускация или запутывание кода — приведение исходного текста или исполняемого кода программы к виду, сохраняющему её функциональность, но затрудняющему анализ.

DOI:10.21681/2311-3456-2020-01-29-41

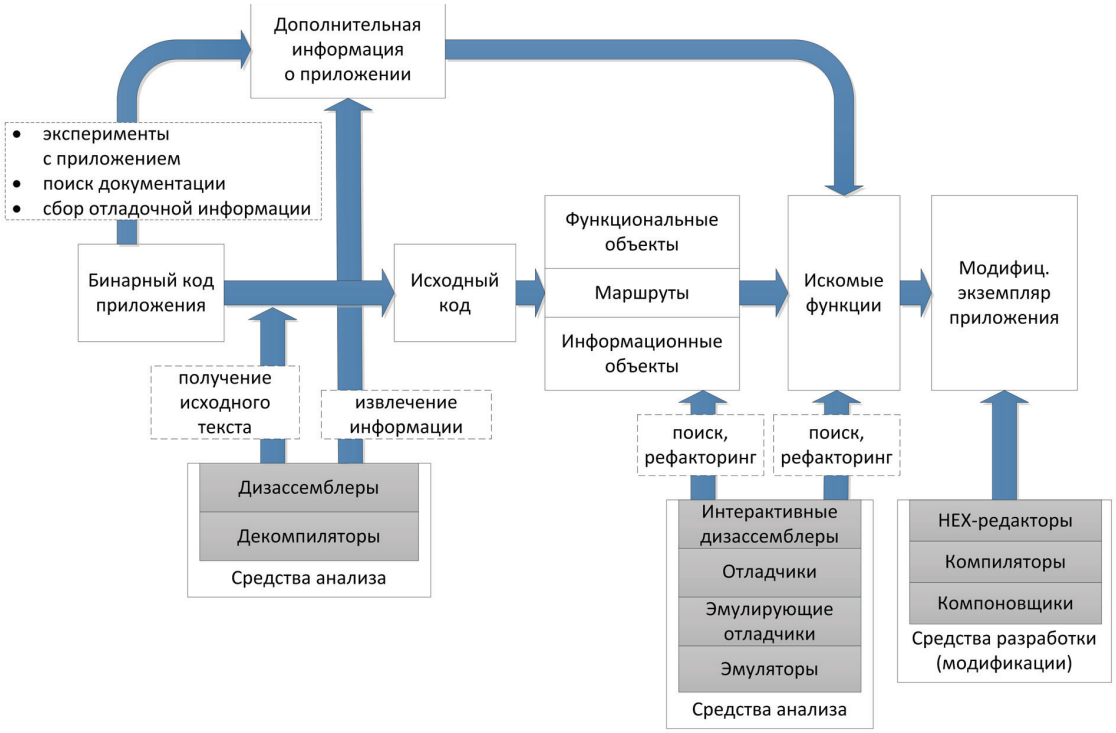


*Рис. 3. Схема исследования и модификации экземпляра приложения*

Среди средств автоматизированного статического анализа известны: *SVACE*, АК-ВС 2, АИСТ-С, КСАИТ, *Project Viewer*, «Бурундук», а также *AppChecker*, *RATS*, *Yasca*, *Cppcheck*, *graudit*, *Klocwork Insight*, *SWAAT*, *PHP Bug Scanner*, *Pixy*, *Ounce* 6, *OWASP Code Crawler*, *Coverity Prevent Static Analysis*.

К средствам автоматизированного динамического анализа можно отнести: *SVACE*, АК-ВС 2, *EMU*, *IRIDA*, КСАИТ, «Бурундук». Наиболее известные средства динамической бинарной инструментации[4]: *frida*, *PIN*, *DinamoRIO*, *Dyninst, Valgrind*.

### Модель угроз безопасности программного обеспечения

Исходя из особенностей применения концепции активных данных, существующих технических возможностей по исследованию программного обеспечения и описанной модели нарушителя предлагаемая система защиты программного обеспечения распределенной вычислительной системы на основе активных данных должна обеспечивать защиту от следующих угроз:

1. несанкционированное использование вычислительных ресурсов агента системы распределенных вычислений на основе активных данных;
2. исполнение произвольного кода средствами агента системы распределенных вычислений на основе активных данных;
3. статический анализ исходного текста терминальных программ (активных данных);
4. статический анализ исходного текста виртуальной машины агента системы распределенных вычислений на основе активных данных;
5. динамический анализ исходного текста терминальных программ (активных данных);
6. динамический анализ приложения виртуальной машины агента системы распределенных вычислений на основе активных данных, в том числе фаззинг и профилирование;
7. восстановление алгоритма программной реализации агента системы распределенных вычислений;
8. компрометация системы распределенных вычислений;
9. восстановление средства формирования пар (виртуальная машина, байт-код).

### Математическая модель системы защиты активных данных от анализа на основе виртуализации исполняемого кода

Под защитой ПО авторы [1 , 19 ] понимают такой метод выполнения программы, при котором невозможно выявить процесс обработки данных и восстановление алгоритма работы программы, либо максимально затруднить решение данных задач. Формально, данную задачу можно представить как

$$Y = F(X, K),$$

4 Динамическая бинарная инструментация (Dynamic Binary Instrumentation, DBI), которая заключается во вставке в бинарный исполняющийся код анализирующих (в общем случае) процедур. При данном подходе нет необходимости в исходном коде анализируемого приложения – работа происходит непосредственно с бинарным файлом в отличие от статической бинарной инструментации кода, которая заключается во вставке, изменении или удалении определенной функциональности в бинарном исполняемом файле и сохранении получившегося файла в виде отдельной копии.

где $X$ – множество входных данных, $Y$ – множество выходных данных, $K$ – множество данных, определяющих право на лицензионное использование экземпляра ПО, $F$ – отображение, определяющее множество процессов обработки входных данных. Таким образом, задача защиты ПО заключается в затруднении восстановления отображения $F$.

Применяемые в настоящее время методы защиты условно делятся на организационно-инфраструктурные и функциональные. Первые направлены на формирование доверенной вычислительной среды (например, операционные системы (ОС) *Android*, *iOS* используют для этих целей изолированную программную среду (ИПС) на основе встроенных служб сертификации и лежащих в их основе несимметричных криптосистем), тогда как вторые – на блокирование действий, разрушающих существующие средства защиты программ от копирования и обратного проектирования (методы защиты от дизассемблеров и декомпиляторов, отладчиков и эмуляторов).

Ряд функциональных методов защиты основан на применении алгоритмов и методов преобразования исполняемого кода прикладных программ к виду, затрудняющему (запутывающему) анализ алгоритмов – *обфускации*. При этом обфускации может подвергаться исходный текст ПО, получаемый из него машинный код, а также ее алгоритм (поток управления) [3 ]. Формально, задача обфускации состоит в получении отображения (обфускатора) $F':(X,K)\xrightarrow{F'}Y$, функционально эквивалентного отображению $F$ в смысле получения эквивалентного множества выходных данных $Y$ при таких же множествах входных данных $(X,K)$. Определение такого обфускатора и требования к нему приводятся в работе [4], а он сам носит название обфускатора в модели виртуального «черного ящика».

В ряде работ отмечалось, что обфускаторы могут использоваться для создания криптосистем с открытым ключом [7], гомоморфных систем шифрования [7], схем дезавуируемого шифрования и односторонних функций с секретом [8].

К недостаткам применения методов обфускации можно отнести следующие:

- код программы после обработки обфускатором может стать более зависимым от программно-аппаратной платформы или компилятора;
- обфускатор затрудняет анализ кода с одной стороны для исследователя, с другой стороны – для разработчика; это приводит к тому, что на этапе отладки ПО систему защиты приходится отключать;
- ни один из существующих известных обфускаторов не гарантирует достаточного уровня сложности декомпиляции (деобфускации) и не обеспечивает безопасности, сопоставимой с уровнем современных криптографических алгоритмов;
- обфускаторы могут содержать ошибки, не учитывающие некоторые особенности модели приложения, обфускацию которого они выполняют, поэтому существует ненулевая вероятность того,

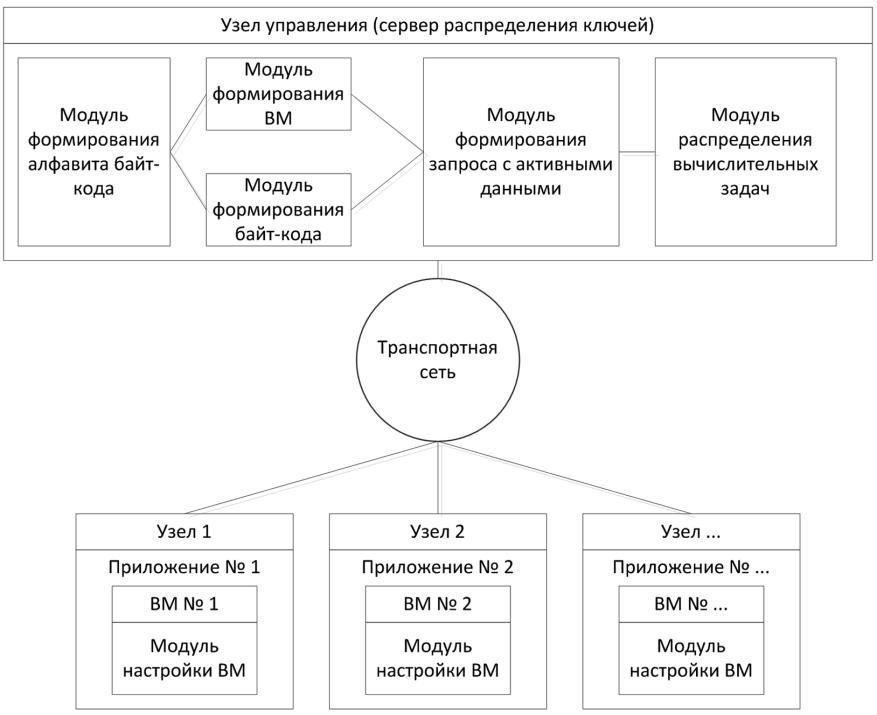


*Рис. 4.Структурная схема система обмена терминальными программами*

что прошедший через обфускатор код потеряет работоспособность (чем сложнее разрабатываемая программа, тем больше эта вероятность).

Общая структурная схема системы, в которой осуществляется обмен и исполнение активных данных (терминальных программ), представлена на рисунке 4, где ВМ – виртуальная машина.

Система состоит из:

1. Множества узлов, в состав которых входят приложения на основе различных реализаций виртуальных машин и модуля настройки виртуальной машины.
2. Узла управления, в состав которого входят:

1) модуль формирования алфавита байт-кода;
2) модуль формирования реализации виртуальной машины;
3) модуль формирования байт-кода;
4) модуль распределения вычислительных задач;
5) модуль формирования запроса с активными данными.

Таким образом, в состав каждого из узлов системы включено приложение в виде виртуальной машины с уникальной архитектурой, отличающейся словарем и алфавитом байт-кода, а также шаблоном исходного текста ПО виртуальной машины. Для того чтобы защищаемая терминальная программа могла исполниться на данной виртуальной машине, ее код должен быть сформирован на основе соответствующей данной виртуальной машине архитектуры.

Виртуализация кода, основанная на технологиях виртуальных машин, позволяет преобразовать машинные команды исходного приложения в произвольный, как правило, неизвестный никому, кроме разработчика, байт-код, который будет выполняться на соответствующей ему (байт-коду) виртуальной машине (интерпретатору), который, в свою очередь, будет входить в состав защищаемого приложения. При этом процесс функционирования защищаемого ПО можно представить как криптосистему [6, 20 ], в которой алгоритм обфускатора и сформированный ключ являются криптограммами, а защищаемый алгоритм и исходные данные – открытым текстом

$$(F_{VM}, X_B) = E(F, X, K),$$

где $F$ – отображение, определяющее множество процессов обработки входных данных, $X$ – множество входных данных, $K$ – секретный ключ криптосистемы, $X_B$ – множество входных данных (байт-код) для виртуальной машины $F_{VM}$, $F_{VM}$ – отображение, определяющее множество процессов обработки входных данных, представленных в виде байт-кода, $E$ – шифр, удовлетворяющий принципам Керкгоффса[5] для криптосистем [21 ].

Отображения $F$ $F_{VM}$, $F_S$ и $E$ в общем случае могут быть представлены в виде конечного автомата:

$$A(X, S, Y, h, f),$$

где $X, S, Y$ – непустые множества элементов входного, внутреннего (состояния автомата) и выходных символов алфавитов, а $h$, $f$ – отображения, определяющие функция переходов между состояниями и функциями выходов автомата.

Таким образом, процесс отображения исходного (защищаемого) алгоритма и набора входных данных в защищенный может быть представлен как:

$$X \xrightarrow{E_B} X_B,$$

$$F \xrightarrow{E_F} F_{VM},$$

при условии, что $Y' = Y$, где $Y' = F_{VM}(X_B)$, а $Y = F(X, K)$, а $(E_B, E_F) \in E$.

Секретным ключом $K = (K_X, K_B, K_{VM})$ в предлагаемой криптосистеме являются:

- множество подстановок $K_X$, определяющих соответствие между элементами множества $X$ и элементами множества $X_B$ – элементы алфавита байт-кода (словаря байт-кода);
- множество подстановок $K_B$, определяющих соответствие между инструкциями или операциями (правилами преобразования) из множества $F$ и элементами подмножества $F_{VM}$ – элементы алфавита байт-кода (словаря байт-кода);
- множество $K_{VM}$ элементов $F_{VM}$, определяющих архитектуру виртуальной машины.

Схема формирования байт-кода $X_B$ с учетом его алфавита $K_B$ и архитектуры виртуальной машины $K_{VM}$ представлена на рисунке 5 .

В свою очередь, исполняемый машинный код виртуальной машины также должен быть сформирован с учетом используемого словаря байт-кода $(K_X, K_B) \subset K$, а также избранной архитектуры виртуальной машины. С учетом изложенного схема формирования виртуальной машины будет выглядеть так, как представлено на рисунке 6 .

Суть предлагаемого решения заключается в том, что каждый экземпляр реализации виртуальной машины и исполняемый на нем байт-код являются уникальными, строго соответствующими друг другу. Иными словами, байт-код и виртуальная машина, сформированные с использованием разных ключей $K = (K_X, K_B, K_{VM})$, будут несовместимы.

Обобщенная схема программной реализации такой криптосистемы представлена на рисунке 7 .

На рисунках 5 –6 секретным ключом являются:

- словарь байт-кода bytecode.json;
- архитектура виртуальной машины vm-template.cpp и получаемая на ее основе виртуальная машина vm.cpp.

Процедуру формирования vm.cpp можно описать отображением

$$F \xrightarrow{E_F(K_{VM}, K_B)} F_{VM},$$

или $F_{VM} = E_F(F, K_{VM}, K_B)$.

5 Принцип Керкгоффса — правило разработки криптографических систем, согласно которому в засекреченном виде держится только определённый набор параметров алгоритма, называемый ключом, а сам алгоритм шифрования должен быть открытым

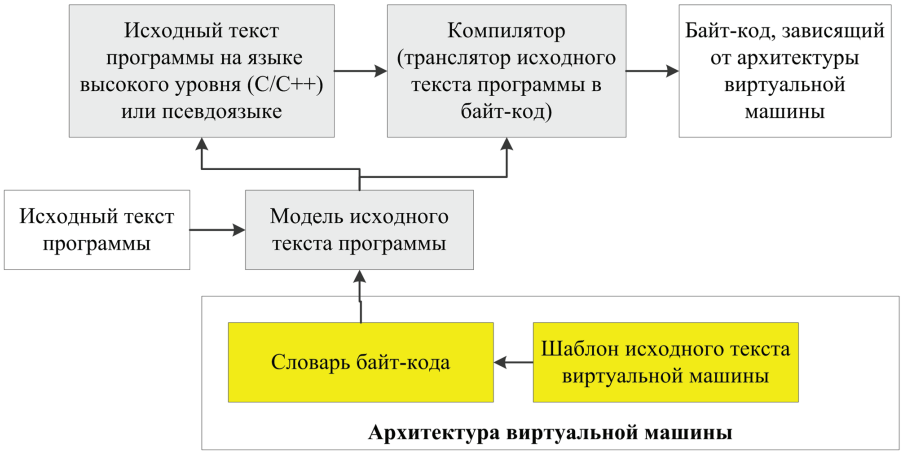


*Рис. 5. Схема формирования байт-кода виртуальной машиной*

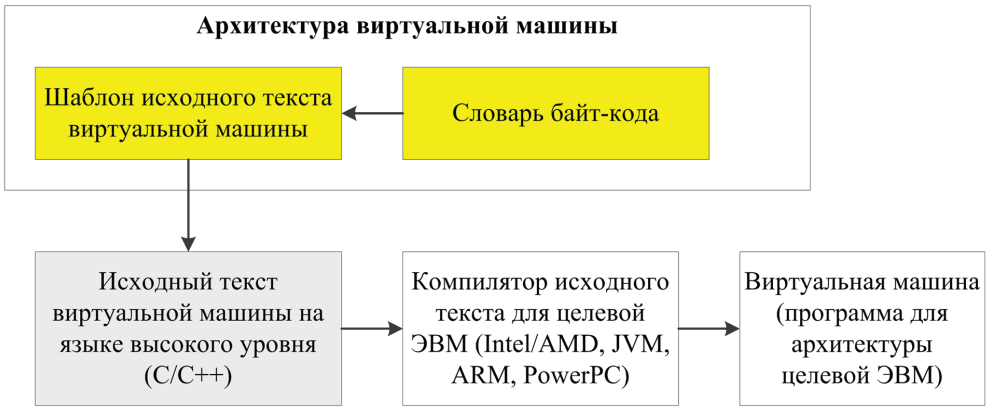


*Рис. 6 . Схема формирования виртуальной машиной*

Процедуру формирования байт-кода, способного выполняться только в заданной виртуальной машине, можно описать отображением

$$X \xrightarrow{E_B(K_X,K_B)} X_B,$$

$$\text{или } X_B = E_B\left(X, K_{VM}, K_B\right).$$

Таким образом, представленная криптосистема обеспечит получение выходных данных $Y' = Y$, где $Y' = F_{VM}\left(X_B\right)$, а $Y = F\left(X,K\right)$, а $\left(E_B, E_F\right) \in E$, но при этом процесс анализа экземпляра программной реализации $F_{VM}\left(X_B\right)$ по сравнению с реализацией $F\left(X,K\right)$ будет значительно затруднен.

Таким образом, сущность защиты программного кода на основе обфускации с использованием виртуальных машин заключается в том, чтобы заставить исследователя перейти от анализа машинных инструкций известной архитектуры (например, x86) к незнакомому набору виртуальных команд, которые увеличат сложность и, соответственно, затрачиваемое время на анализ защиты.

Поскольку защищаемый программный код может исполняться только на соответствующей виртуальной машине, то подобная система аналогична несимметричной криптографической системе, в которой защищаемой информацией является исполняемый машинный код, а ключевой парой (открытый – секретный ключ) является архитектура виртуальной машины и две ее производные – байт-код и виртуальная машина.

Схематично такая криптосистема представлена на рисунке 7.

В рассматриваемой в работе несимметричной криптосистеме процесс зашифрования на открытом ключе эквивалентен процессу трансляции (отображению) $E_B$ защищаемого исходного текста $X$ в байт-код $X_B$:

$$X \xrightarrow{E_B(K_X,K_B)} X_B,$$

$$\text{или } X_B = E_B\left(X, K_{VM}, K_B\right),$$

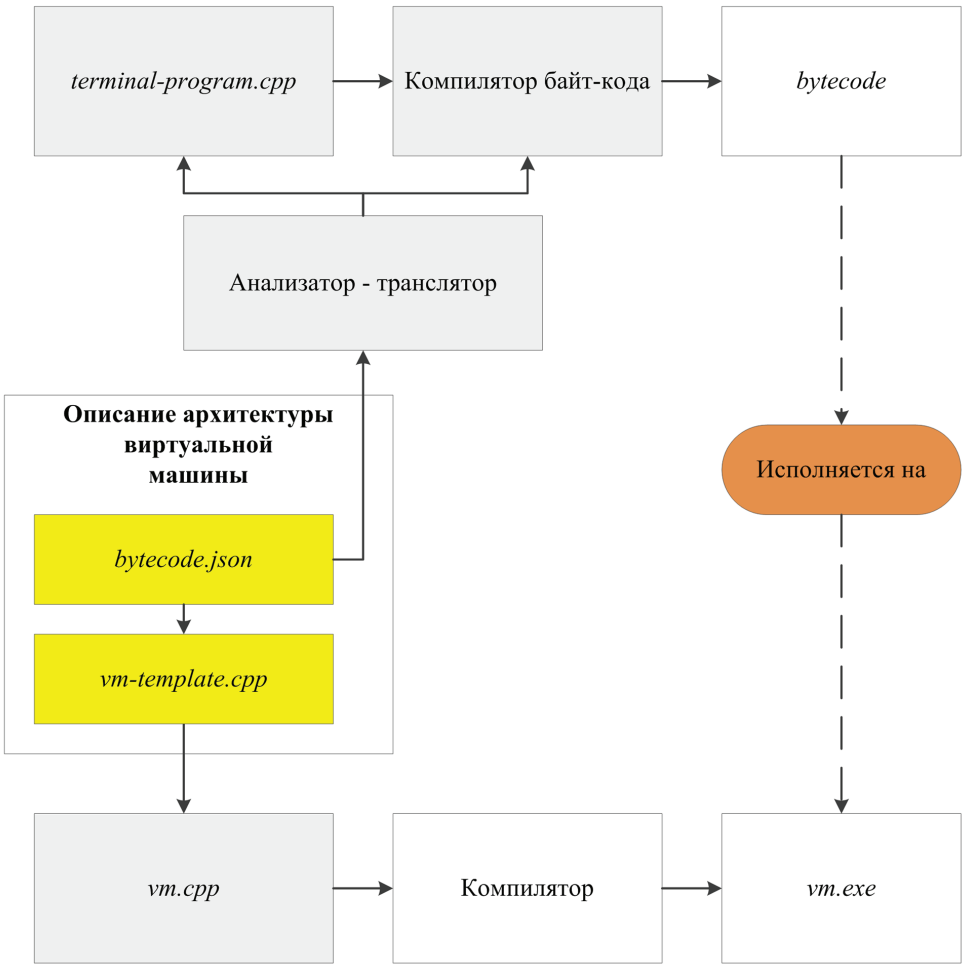


*Рис. 7 . Структурная схема программной реализации криптосистемы на основе виртуальных машин*

где $K_X$ – множество подстановок, определяющих соответствие между элементами множества $X$ и элементами множества $X_B$ – элементы алфавита байт-кода (словаря байт-кода); $K_B$ – множество подстановок, определяющих соответствие между инструкциями или операциями (правилами преобразования) из множества $F$ (множество инструкции исходного текста защищаемого приложения) и элементами подмножества $F_{VM}$ – элементы алфавита байт-кода (словаря байт-кода), используемые в виртуальной машине.

При этом открытым ключом является словарь байт-кода $K_X$, а закрытым – архитектура виртуальной машины, скомпилированная на основе того же словаря байт-кода $(K_X, K_B) \subset K$ .

**Эффективность системы защиты активных данных в распределенной вычислительной системе**

Для того, чтобы преодолеть защитные механизмы, исполняемые встроенной в приложение виртуальной машиной как инструкции байт-кода, необходимо решить следующие задачи:

1. проанализировать работу компонентов интерпретатора виртуальной машины:
   - выделить машинные коды инструкций байт-кода;
   - определить фактические инструкции, исполняемые при выполнении обнаруженных инструкций байт-кода;
2. выделить семантику инструкций байт-кода на основе анализа фактически исполняемых машинных инструкций;
3. восстановить логику работы целевого кода (алгоритм).

Для оценивания эффективности предлагаемой системы было разработано два прототипа ее программной реализации [16 ]: на языке *C/C++* и на языке *Java* и проведены эксперименты по сравнительному анализу вычислительной сложности данных прототипов.

В качестве секретного ключа была выбрана простейшая структура виртуальной машины [22 ] с набором примитивных инструкций без стека и регистров. В процессе исполнения защищаемого приложения сформированный байт-код передается как входные данные модулю приложения – виртуальной машине, которая его и обрабатывает.

Фрагмент исходного текста виртуальной машины на *Java* представлен ниже.

```
public class BCclass {
    static void BCclass(String bytecode) {
        bytecode = bytecode.toLowerCase();
        byte[] b = bytecode.getBytes();
        for (byte bytec : b) {
            switch (bytec) {
                case (bc0):
                    func0();
                    break;
                case (bc1):
                    func1();
                    break;
                      . . .
                default:
                    break;
            }
        }
    }

    public void func0() {
              . . .
    }

    public void func1() {
              . . .
    }
```

Схема функционирования виртуальной машины представлена на рисунке 8 . В процессе исполнения защищаемого приложения сформированный байт-код передается как входные данные модулю приложения – виртуальной машине, которая его и обрабатывает.

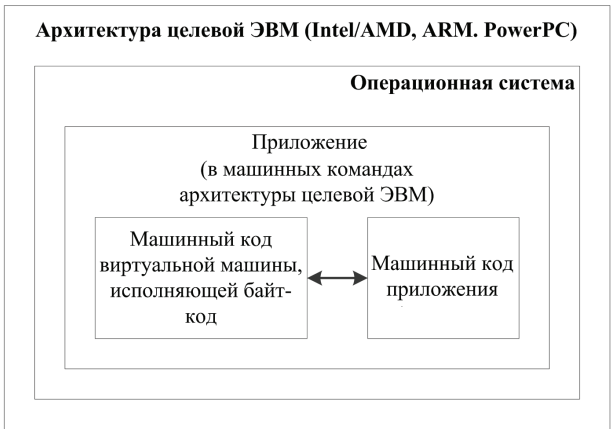


*Рис. 8. Функционирование виртуальной машины в целевой среде*

Работа прототипа, разработанного на языке *Java*, в целевой ЭВМ представлена на рисунке 9 .

Порядок взаимодействия элементов прототипа, реализующих тестирование удаленного веб-ресурса методом фаззинга[6], представлен на рисунке 10 .

Этапы выполнения задачи в распределенной системе следующие:

1. Отправка запроса о готовности к работе от ПО агента (Java-приложения) к серверу управления с его последующей обработкой на сервере.
2. Отправка запроса на получения вычислительной задачи от сервера управления к средству распределения задач.
3. Выбор очередной задачи средством распределения задач и отправка ответа серверу управления.
4. Формирования и отправка байт-кода агенту с *Java*-приложением.
5. Исполнение вычислительной задачи (тестирование методом фаззинга с отправкой запроса тестируемому веб-ресурсу).
6. Получение ответа от исследуемого веб-ресурса и его обработка.
7. Отправка результата выполнения вычислительной задачи серверу управления.
8. Отправка отчета о выполнении вычислительной задаче средству распределения задач.

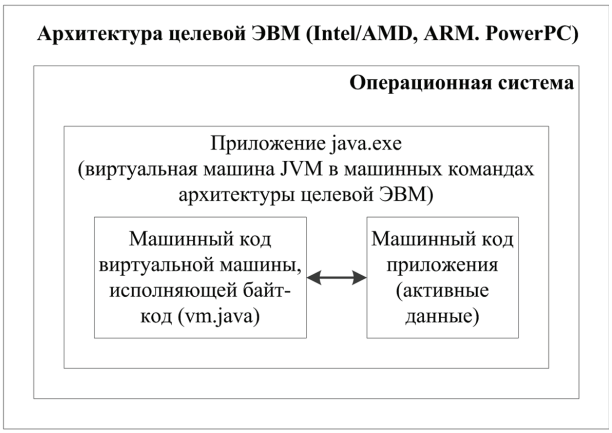


*Рис. 9. Функционирование виртуальной машины vm.java*

Результаты оценки эффективности прототипа на базе приложения, разработанного на *C/C++*.

1. Объем кода:

1) защищаемого приложения (исходная программа): 195 строк;

2) защищенного приложения (программа, выполняемая в виртуальной машине): 4061 строка;

3) байт-код: 1000000 байт.

Среднее время выполнения:

1) защищаемого приложения: 2,3922 мс;

2) защищенного приложения: 4,0976 мс.

Результаты оценки эффективности прототипа на базе приложения, разработанного на *Java*.

1. Объем кода:

1) защищаемого приложения (исходная программа): 422 байта;

2) защищенного приложения (виртуальная машина): 927 байта;

3) байт-код: 1000000 байт.

Среднее время выполнения:

1) защищаемого приложения: 7 мс;

2) защищенного приложения: 1003 мс.

Полученные результаты позволяют сделать следующие выводы:

6 Фаззинг (англ. fuzzing) — техника тестирования программного обеспечения, заключающаяся в передаче приложению на вход неправильных, неожиданных или случайных данных.

DOI:10.21681/2311-3456-2020-01-29-41

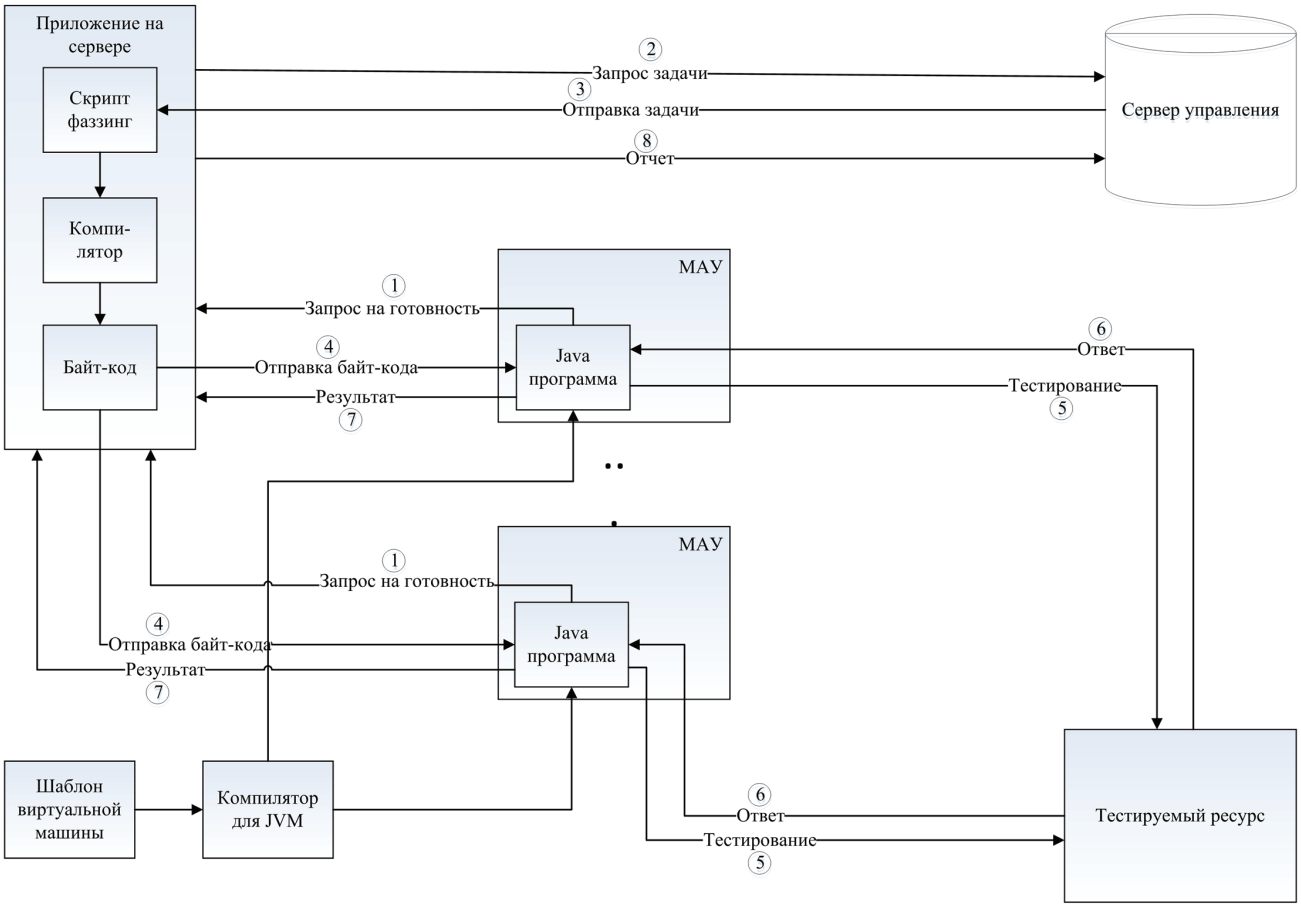


*Рис. 10 . Порядок выполнения вычислительных задач распределенной системой на базе агентов с ПО, разработанном на Java*

1. Предлагаемая система защиты имеет недостаток, выраженный в том, что защита ПО с использованием предлагаемой системы предъявляет повышенные требования к вычислительной системе, в которой исполняется защищаемое ПО и снижает его производительность.

2. Достоинствами предложенной системы является:
   - повышение сложности анализа ПО за счет вынуждения исследователя проводить анализ кода с неизвестными кодами виртуальных команд;
   - невозможность выполнения байт-кода на виртуальной машине, если байт-код и виртуальная машина сформированы с использованием различных секретных ключей.

## Заключение

В работе предложена система защиты активных данных в распределенной вычислительной системе на основе применения виртуальных машин. На основе проведенного анализа условий функционирования сформированы требования к таким виртуальным машинам. Разработаны и описаны модели угроз и нарушителя безопасности программного обеспечения распределенной вычислительной системы на основе активных данных.

Предложена математическая модель системы защиты активных данных от анализа на основе виртуализации исполняемого кода и ее программная реализация в виде приложений на *C/C++* и *Java*.

Экспериментально проверена эффективность системы по частным показателям качества (оперативности) предложенной системы, подтверждающая ее результативность.

**Рецензент:** *Марков Алексей Сергеевич, доктор технических наук, старший научный сотрудник, профессор Московского государственного технического университета им. Н.Э. Баумана, г. Москва, Россия. E-mail: a.markov@bmstu.ru*

**Литература**


1. Петров А.А. Методы защиты программного кода // Информатика и безопасность современного общества. 2019. Т. 1. № 1. С. 24–28.
2. Петров А.С., Петров А.А. Технология защиты программного кода посредством применения виртуальной машины // Вестник ВНУ. 2009. № 9 (103), часть 1. С. 117–122.
3. Аранов В.Ю. Метод и средства защиты исполняемого программного кода от динамического и статического анализа : автореф. дис. ... канд. техн. наук : 05.13.19 Санкт-Петербург, 2014. 18 с.
4. Речистов Г.С., Юлюгин Е. А. Моделирование инструкций поддержки транзакционной памяти в современных центральных процессорах // Прикладная информатика. 2014. № 5 (53). С. 16–24.
5. Казарин О.В., Шубинский И.Б. Надежность и безопасность программного обеспечения / Москва: Издательство Юрайт. 2018. 342 с.
6. Варнавский Н.П., Захаров В.А., Кузюрин Н.Н., Шокуров А.В. Современное состояние исследований в области обфускации программ: определение стойкости обфускации // Труды ИСП РАН. Т. 26. Вып. 3. 2014. С. 167–198. DOI: 10.15514/ISPRAS-2014-26(3)-9.
7. Kaiyuan Kuang, Zhanyong Tang, Xiaoqing Gong, Dingyi Fang, Xiaojiang Chen, Zheng Wang Enhanced virtual-machine-based code obfuscation security through dynamic bytecode scheduling // Computers & Security. 2018. № 74. P. 202–220. DOI: 10.1016/j.cose.2018.01.008.
8. Barak B., Goldreich O., Impagliazzo R., Rudich S., Sahai A., Vadhan S., Ke Yang On the (im)possibility of obfuscating programs // Advances in Cryptology – CRYPTO'01, Lecture Notes in Computer Science, v. 2139, 2001, p. 1–18. DOI: 10.1145/2160158.2160159.
9. Кулешов С.В., Цветков О.В. Активные данные в цифровых программно-определяемых системах // Информационно-измерительные и управляющие системы. 2014. Т. 12. № 6. С. 12-19.
10. Александров В.В., Кулешов С.В., Цветков О.В., Зайцева А.А. Концепция построения инфотелекоммуникации (прототип SDR) // Труды СПИИРАН. 2008. № 6. С. 51–57. DOI: 10.15622/sp.6.5.
11. Маркин Д.О., Галкин А.С., Архипов П.А. Организация анонимного доступа с помощью веб-прокси // Наукоемкие технологии в космических исследованиях Земли. 2016. Т. 8, № 5. С. 44–49.
12. Маркин Д.О., Галкин А.С., Архипов П.А. Исследование устойчивости анонимной сети на основе технологий веб-прокси // Вопросы кибербезопасности. 2016. № 2 (15). С. 21–28. DOI: 10.21681/2311-3456-2016-2-21-28.
13. Маркин Д.О, Павлов Д.И., Звягинцев С.А.. Анализ методов и средств исследования программного обеспечения // Актуальные направления развития систем охраны, специальной связи и информации для нужд органов государственной власти Российской Федерации: XI Всероссийская межведомственная научная конференция: материалы и доклады (Орёл, 5–6 февраля 2019 г.) В 10 ч. Ч. 10 / под общ. ред. П. Л. Малышева. Орёл: Академия ФСО России, 2019. 200 с. С. 34–37.
14. Маркин Д.О. Макеев С.М., Вихарев А.Н. Комплекс алгоритмов защищенных туманных вычислений на основе технологии активных данных // Известия Тульского государственного университета. Технические науки. 2019. Выпуск 3. С. 263–269.
15. Маркин Д.О., Макеев С.М., Ленчук В.Д., Охрименко А.А., Шапкин Р.В. Автоматизированная система управления содержанием удаленных информационных ресурсов: свидетельство о государственной регистрации программы для ЭВМ № 2016617298 Российская Федерация / авторы и правообладатели Д.О. Маркин, С.М. Макеев, В.Д. Ленчук, А.А. Охрименко, Р.В. Шапкин. № 2016614475; заявл. 04.05.2016; зарегистрировано в Реестре программ для ЭВМ 29.06.2016 г.
16. Маркин Д.О., Трохачёв М.А., Земцов А.Э., Юркин А.А. Программный агент обеспечения распределенных вычислений на основе технологии активных данных для узла вычислительной сети на базе мобильных устройств под управлением операционной системы Android : свидетельство о государственной регистрации программы для ЭВМ № 2018660343 Российская Федерация / авторы и правообладатели Д.О. Маркин, М.А. Трохачёв, А.Э. Земцов, А.А. Юркин. № 2018617165 ; заявл. 10.07.2018; зарегистрировано в Реестре программ для ЭВМ 22.08.2018 г.
17. Маркин Д.О., Трохачев М.А. Разработка агента туманных вычислений для мобильных устройств под управлением операционной системы Android // Информационная безопасность и защита персональных данных: Проблемы и пути их решения: материалы XI Межрегиональной научно-практической конференции / под ред. О.М. Голембиовской, М.Ю.Рытова. Брянск: БГТУ, 2019. 190 с. С. 136–141.
18. Маркин Д.О., Земцов А.Э. Алгоритм решения задачи факторизации средствами туманных вычислений // Информационная безопасность и защита персональных данных. Проблемы и пути их решения : материалы XI Межрегиональной научно-практической конференции [Электронный ресурс] / под ред. О. М. Голембиовской, М. Ю. Рытова. Брянск : БГТУ, 2019. 190 с. С. 126–129.
19. Петров А.С., Петров А.А. Технология защиты программного кода посредством применения виртуальной машины // Вестник ВНУ. 2009. № 9 (103), часть 1. С. 117–122.
20. Sahai A., Waters B. How to Use Indistinguishability Obfuscation: Deniable Encryption, and More // CRYPTO ePrint, 2013. DOI: 10.1145/2591796.2591825.
21. Auguste Kerckhoffs. La cryptographie militaire. Journal des sciences militaires, 1883.
22. Интерпретаторы байт-кодов своими руками [Электронный ресурс] : [сайт] / Блог компании Badoo. Режим доступа: https://habr.com/company/badoo/blog/425325/. Дата обращения: 17.09.2019.

# PROTECTION SYSTEM OF TERMINAL PROGRAMS AGAINST ANALYSIS BASED ON CODE VIRTUALIZATION

*Markin D. O.*[7], *Makeev S. M.*[8]

***Abstract.*** *The paper proposes a protection system of active data in a distributed computing system. The analysis of existing technical solutions is carried out, the scientific problem consisting in necessity of effective protection of terminal programs against the analysis and restoration of algorithms of functioning is allocated. The object of protection in the presented work are terminal programs or active data functioning in nodes of the distributed computing system. The aim of the work is to increase the security of terminal programs against the analysis and restoration of the algorithm of their functioning. The proposed solution is based on the application of virtual machine technology with a secret architecture. The architecture of the virtual machine in this paper refers to the used alphabet of byte-code and functional objects used in protected terminal programs. The models of threats and software security violator are developed and described. The requirements are formed on the basis of the analysis of the operating conditions, as well as models of threats and security violator. A mathematical model of the active data protection system against analysis based on code virtualization and its software implementation in the form of applications in C/C++ and Java is proposed. The efficiency of the system was experimentally tested according to particular indicators of quality (efficiency) of the proposed system, confirming its effectiveness. The proposed solution makes it possible to effectively complicate the analysis and restoration of algorithms for the functioning of terminal programs in a distributed computer system.*

***Keywords:****information security system, active data, obfuscation methods, virtual machine, byte-code.*

## References

1. Petrov A.A. Metodi zashhity programmnogo koda // Informatika i bezopasnost' sovremennogo obshchestva. 2019. Vol. 1. No 1. P. 24–28.
2. Petrov A.S., Petrov A.A. Tehnologija zashhity programmnogo koda posredstvom primenenija virtual'noi mashini // Vestnik VNU. 2009. № 9 (103), No 1. pp. 117–122.
3. Aranov V.Y. Metod i sredstva zashhity ispolnjaemogo programmnogo koda ot dinamicheskogo i staticheskogo analiza: avtoref. dis. ... kand. tehn. nauk : 05.13.19 Sankt-Peterburg, 2014. 18 p.
4. Rechistov G.S., Yulyugin E.A. Simulation of instruction set extension for transactional memory of modern central processors // Prikladnaya informatika. 2014. No 5 (53). P. 16–24.
5. Kazarin O.V., Shubinskii I.B. Nadegnost' i bezopasnost' programmnogo obespechenija / Moscow: Izdatel'stvo Yurait. 2018. 342 p.
6. Varnavskii N.P., Zaharov V.A., Kuzyurin N.N., Shokurov A.V. Sovremennoe sostojanie issledovanii v oblasti obfuskacii programm : opredelenie stoikosti obfuskacii // Trudi ISP RAN. Vol. 26. No 3. 2014. pp. 167–198. DOI: 10.15514/ISPRAS-2014-26(3)-9.
7. Kaiyuan Kuang, Zhanyong Tang, Xiaoqing Gong, Dingyi Fang, Xiaojiang Chen, Zheng Wang Enhanced virtual-machine-based code obfuscation security through dynamic bytecode scheduling // Computers & Security. 2018. No 74. P. 202–220. DOI: 10.1016/j.cose.2018.01.008.
8. Barak B., Goldreich O., Impagliazzo R., Rudich S., Sahai A., Vadhan S., Ke Yang On the (im)possibility of obfuscating programs // Advances in Cryptology – CRYPTO'01, Lecture Notes in Computer Science, Vol. 2139, 2001, pp. 1–18. DOI: 10.1145/2160158.2160159.
9. Kuleshov S.V., Cvetkov O.V. Aktivnye dannye v cifrovyh programmno-opredeljaemyh sistemah // Informacionno-izmeritel'nie i upravljayushhie sistemy. 2014. Vol. 12. № 6. pp. 12–19.
10. Aleksandrov V.V., Kuleshov S.V., Cvetkov O.V., Zaiceva A.A. Koncepcija postroenija infotelekommunikacii (prototip SDR) // Trudi SPIIRAN. 2008. No 6. pp. 51–57. DOI: 10.15622/sp.6.5.
11. Markin D.O., Galkin A.S., Arhipov P.A. Organizatsija anonimnogo dostupa s pomosch'ju veb-proksi // Naukoemkie tehnologii v kosmicheskih issledovanijah Zemli. 2016. No 8, № 5. pp. 44–49.
12. Markin D.O., Galkin A.S., Arhipov P.A. Issledovanie ustojchivosti anonimnoj seti na osnove tehnologij veb-proksi // Voprosy kiberbezopasnosti. 2016. No 2 (15). pp. 21–28. DOI: 10.21681/2311-3456-2016-2-21-28.
13. Markin D.O, Pavlov D.I., Zvjagintsev S.A. Analiz metodov i sredstv issledovanija programmnogo obespechenija // Aktual'nye napravlenija razvitija sistem ohrany, spetsial'noj svjazi i informatsii dlja nuzhd organov gosudarstvennoj vlasti Rossijskoj Federatsii: XI Vserossijskaja mezhvedomstvennaja nauchnaja konferentsija: materialy i doklady (Orjol, 5–6 feb. 2019) Vol. 10 / pod obsch. red. P. L. Malysheva. Orjol: Akademija FSO Rossii, 2019. 200 p. pp. 34–37.
14. Markin D.O. Makeev S.M., Viharev A.N. Kompleks algoritmov zaschischennyh tumannyh vychislenij na osnove tehnologii aktivnyh dannyh // Izvestija Tul'skogo gosudarstvennogo universiteta. Tehnicheskie nauki. 2019. No 3. pp. 263–269.
15. Markin D.O., Makeev S.M., Lenchuk V.D., Ohrimenko A.A., Shapkin R.V. Avtomatizirovannaja sistema upravlenija soderzhaniem udalennyh informatsionnyh resursov № 2016617298 ; declared 29.06.2016.

7 Dmitrij Markin, Ph.D., Academy of FSS of Russia. Oryol city, Russia. E-mail: admin@nikitka.net

8 Sergey Makeev, Ph.D., Academy of FSS of Russia. Oryol city, Russia. E-mail: maksm57@yandex.ru

16. Markin D.O., Trohachjov M.A., Zemtsov A.E., Jurkin A.A. Programmnyj agent obespechenija raspredelennyh vychislenij na osnove tehnologii aktivnyh dannyh dlja uzla vychislitel'noj seti na baze mobil'nyh ustrojstv pod upravleniem operatsionnoj sistemy Android № 2018660343 ; declared 22.08.2018 g.
17. Markin D.O., Trohachev M.A. Razrabotka agenta tumannyh vychislenij dlja mobil'nyh ustrojstv pod upravleniem operatsionnoj sistemy Android // Informatsionnaja bezopasnost' i zaschita personal'nyh dannyh: Problemy i puti ih reshenija: materialy XI Mezhregional'noj nauchno-prakticheskoj konferentsii / pod red. O.M. Golembiovskoj, M.Ju.Rytova. Brjansk: BGTU, 2019. pp. 136–141.
18. Markin D.O., Zemtsov A.E. Algoritm reshenija zadachi faktorizatsii sredstvami tumannyh vychislenij // Informatsionnaja bezopasnost' i zaschita personal'nyh dannyh. Problemy i puti ih reshenija : materialy XI Mezhregional'noj nauchno-prakticheskoj konferentsii / pod red. O. M. Golembiovskoj, M. Ju. Rytova. Brjansk : BGTU, 2019. p. 126–129.
19. Petrov A.S., Petrov A.A. Tehnologija zaschity programmnogo koda posredstvom primenenija virtual'noj mashiny // Vestnik VNU. 2009. No 9 (103), Vol. 1. pp. 117–122.
20. Sahai A., Waters B. How to Use Indistinguishability Obfuscation: Deniable Encryption, and More // CRYPTO ePrint, 2013. DOI: 10.1145/2591796.2591825.
21. Auguste Kerckhoffs. La cryptographie militaire. Journal des sciences militaires, 1883.
22. Interpretatory bajt-kodov svoimi rukami / Blog kompanii Badoo. URL: https://habr.com/company/badoo/blog/425325/. Accessed: 17.09.2019.