УДК 007.52:621.39 DOI: 10.14529/mmp250306

ТЕХНОЛОГИЯ АКТИВНЫХ ДАННЫХ В ВИРТУАЛЬНЫХ МАШИНАХ С ДИНАМИЧЕСКОЙ СИСТЕМОЙ КОМАНД

С.В. Кулешов[1]***, А.А. Зайцева***[1]***, А.Ю. Аксенов***[1]
[1]Санкт-Петербургский Федеральный исследовательский центр РАН,
г. Санкт-Петербург, Российская Федерация

В ряде прикладных задач, таких как разработка защищенных банковских терминалов или приложений, систем группового управления БПЛА, требуется обеспечить защиту исполнимого кода от реверс-инжиниринга путем копирования области памяти и последующего его изучения путем дизассемблирования и восстановления алгоритма, содержащего фрагмент исполнимого кода. Данный вид реверс-инжиниринга может быть осуществлен при физическом перехвате одного из узлов распределенной системы. Для достижения поставленной цели в статье рассматривается подход к организации защиты от такого типа атаки, основанный на использовании активных данных совместно с виртуальными машинами, использующими динамическую систему команд. В результате проведенных исследований на основе технологии активных данных предложен подход к динамической реконфигурации виртуальных машин, позволяющий осуществить дополнительную защиту устройств от реверс-инжиниринга программного кода. Практическая значимость предложенного подхода состоит в реализации программно-определяемых и программно-управляемых систем на основе технологии активных данных.

Ключевые слова: виртуальная машина; динамическая система команд; активные данные.

Введение

Существует класс устройств, содержащих «чувствительные данные», утечка и тиражирование которых может нанести существенный ущерб. Предлагаемая технология может быть применена для защиты «чувствительных данных» в устройствах, которые могут быть перехвачены с последующим созданием «дампа» постоянной и оперативной памяти и последующим их анализом, декомпиляцией программ, сохранением ключей шифрования и т. д.

Суть предлагаемой технологии заключается в использовании динамической уникальной системы машинных команд для реализации целевой программы и исполнителя (виртуальной машины), использующего эту систему команд. Защита от такого типа атаки осуществляется путем генерации пары программ – целевого программного кода и программного кода виртуальной машины под каждый конкретный экземпляр узла при его инициализации (например, непосредственно перед взлетом БПЛА, являющегося узлом распределенной системы).

В этом случае при получении физического доступа злоумышленника к устройству (предположим, что злоумышленник обладает всеми необходимыми знаниями и техническими устройствами) получение дампа памяти программ из оперативной памяти встроенного компьютера не позволит произвести декомпиляцию программы, так как ему принципиально неизвестна система команд, на котором она написана.

В основу технологии реализации предлагаемого метода защиты положен разработанный авторами формализм активных данных. Под активными данными в рамках работы понимаются данные, передаваемые между компонентами системы и включающие фрагменты исполнимого кода [1, 2].

1. Состояние проблемы

В основном задача защиты конфигурации ПЛИС связана с защитой от утечки данных при программировании управляющих систем в недоверенном окружении либо при обновлении встроенного ПО. Основные подходы такой защиты заключаются в использовании криптографических приемов [3]. Существуют методы защиты кода проекта на уровне функционального взаимодействия между компонентами системы [4]. Существуют исследования, рассматривающие вопросы защиты конфигурации ПЛИС [5,6]. Существуют различные методы защиты от считывания из внешней памяти [7,8], в том числе с использованием микросхем специальной памяти [9]. Отдельным направлением таких атак является кража самих алгоритмов с целью поиска в них уязвимостей, разработки эксплойтов или иных целевых атак на информационные системы или устройства. Такой вид атак используется, как правило на системы защиты информации, и на устройства специального назначения. При этом, последние являются наиболее уязвимыми, так как ценность извлекаемых алгоритмов на первых этапах их эксплуатации является несоизмеримо высокой по сравнению со стоимостью работ по организации атаки на них. Анализ таких ситуаций не получил широкого распространения в научно-технической литературе, в основном исследователи сосредоточены на решении частных проблем.

2. Таксономия уровней активных данных

Технология активных данных в зависимости от решаемой задачи может использовать несколько уровней доступа к ресурсам системы. Простейший вариант использования активных данных – формирование битового потока путем выполнения последовательности исполнимых инструкций на виртуальной машине (ВМ). Такой вариант последовательности инструкций называется терминальной программой [10] то есть вырожденным случаем программы без входных данных, либо активными данными нулевого уровня (уровень AD_0), принцип работы терминальной программы проиллюстрирован на рис. 1. Активные данные нулевого уровня могут использоваться как эффективный вариант сжатия данных в тех случаях, когда данные можно представить алгоритмически, возможности реконфигурации на этом уровне отсутствуют.

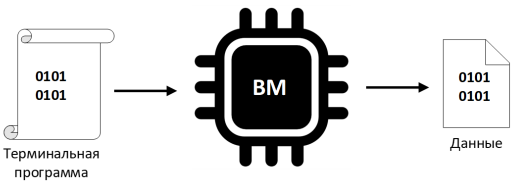

Рис. 1. Иллюстрация принципа работы терминальной программы (уровень AD_0)

Более универсальным вариантом реализации коммуникационных функций при работе с данными является использование кодеков (кодеров/декодеров) – то есть передача совокупности «данные + программа», в которой программа требуется исключительно для декодирования приложенных к ней данных (как правило, сжатых) в том случае, когда передача такой совокупности имеет более компактное представление, чем исходные данные. Такой вариант представления относится к активным данным уровня один (уровень AD_1), возможности реконфигурации сводятся к замене «на лету» программы-декодера для обработки соответствующего фрагмента данных (рис. 2).

Более расширенные возможности реконфигурации предоставляются на уровнях AD_2 и AD_3, где активные данные могут управлять периферийными устройствами

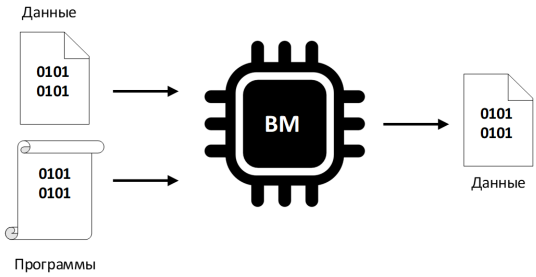

Рис. 2. Иллюстрация принципа работы программы декодера (уровень AD_1)

(технологии, подобные обновлению встроенного программного обеспечения – firmware периферийных устройств) или переконфигурировать сам процессор, используя технологии, подобные обновлению микрокода [11]. В качестве рабочей среды для исполнения активных данных в работах [12–14] было предложено использовать виртуальную машину активных данных (Active Data Virtual Machine – ADVM). Для разработки ADVM и исполнения активных данных подходят как стек-ориентированные, так и регистр-ориентированные архитектуры виртуальных машин [15, 16], выбор архитектуры производится с учетом решаемых задач. Стек-ориентированные (стековые) виртуальные машины традиционно используются в реализации платформ для языков программирования (ActionScript, Java, .Net), благодаря чему такая реализация виртуальных машин достаточно широко распространена [16, 17]. Основным преимуществом стековых виртуальных машин является то, что при их реализации не нужно указывать адреса операндов в явном виде, что может оказаться полезным в случае распределенной реализации при отсутствии общей памяти. Если говорить о регистр-ориентированных (регистровых) виртуальных машинах, то необходимо отметить их преимущество в плане быстродействия при выполнении команд за счет отсутствия обязательных операций работы со стеком POP и PUSH, а также возможности оптимизации при выполнении нескольких одинаковых команд. При этом вследствие необходимости явного указания операндов инструкция регистр-ориентированной виртуальной машины в среднем оказывается длиннее, чем в стек-ориентированной. Примером стековой виртуальной машины может служить виртуальная машина DALVIK, использовавшаяся в ОС Android до версии 5.0 [16, 17].

3. Формализация ADVM

В процессах софтверизации важную роль играет гибкость и возможность реконфигурируемости программно-аппаратных платформ, чаще всего обеспечиваемая аппаратными способами, реже на программном уровне. Идеология активных данных разрабатывалась с целью облегчения программного реконфигурирования соответствующих программно-аппаратных платформ, без необходимости прерывания их работы. При этом чтобы снизить риски нарушения информационной безопасности, неизбежно сопутствующие повышению гибкости программно-определяемых систем, предложено описание ADVM на базе формального исполнителя, частично рассмотренного в [18]. Поскольку в настоящее время понятие машины Тьюринга является общеупотребимым, а большинство современных компьютерных систем обладает Тьюринг-полнотой, принимая нотации и определения авторов [5], можно определить формальный исполнитель, выполняющий функции ADVM, как

$$M = (S, I, \delta, s_0, S_f)\,, \tag{1}$$

где S – множество состояний, которые может иметь ADVM; I – множество инструкций, которые может исполнять ADVM; δ: $S \times I \to S \times I$ – оператор выполнения

инструкций I; s_0 – начальное состояние машины; S_f – множество допустимых финальных состояний машины [5]. Кроме того, если определить последовательность операторов как $\delta^{(m)}$, можно записать

$$\delta^{(m)}(s_n, i_n) = \underbrace{\delta \circ \ldots \circ \delta}_{m}(s_n, i_n) = \delta(\ldots \delta(s_n, i_n)) = (s_{n+m}, i_{n+m}). \tag{2}$$

Соответственно, активные данные определяются как

$$A = (i_n i_{n+1} i_{n+2} \ldots, d_k d_{k+1} d_{k+2} \ldots), \tag{3}$$

где $i_n i_{n+1} i_{n+2} \ldots$ – последовательность исполнимых инструкций, $d_k d_{k+1} d_{k+2} \ldots$ – последовательность фрагментов данных, $d_i \in D$, где D – множество допустимых данных. Для уровня AD_0 использование формального исполнителя имеет вид:

$$(i_n i_{n+1} i_{n+2} \ldots) \xrightarrow{M} (d_k d_{k+1} d_{k+2} \ldots). \tag{4}$$

Если использовать набор инструкций, рассмотренный в [17], состоящий из 4 команд: вывода «0», вывода «1», цикла с конечным количеством итераций и группировки битов, то такой код гарантированно выполняется за конечное время и не может приводить к «тупикам» и бесконечным циклам. Для уровня AD_1 использование формального исполнителя M расширяется возможностью использовать входные (кодированные) данные $d_l d_{l+1} d_{l+2} \ldots$ для формирования выходных (декодированных) данных $d_k d_{k+1} d_{k+2} \ldots$ имеет вид

$$(i_n i_{n+1} i_{n+2} \ldots, d_l d_{l+1} d_{l+2} \ldots) \xrightarrow{M} (d_k d_{k+1} d_{k+2} \ldots). \tag{5}$$

Использование принципа гомоиконности (единство формы представления инструкций и данных) [12] позволяет построить объединенный алфавит T:

$$T = I \cup D. \tag{6}$$

На уровне AD_2 допускается переопределение фрагментов исполнимого кода, определяемого в виде сервисных функций, либо встраиваемого программного обеспечения периферийных устройств. Описанные уровни активных данных AD_0, AD_1 не могут модифицировать M, а значит не требуют расширения формализма машины Тьюринга [20,21]. Для описания уровней AD_2, AD_3 внесены следующие изменения в исходное описание формального исполнителя:

1) входной символ переопределен как набор T (множество инструкций и машинных слов, представляющих данные), который фактически содержит все символы, которые может принимать виртуальная машина;

2) в процессе интерпретации входных символов T формальный вычислитель M может изменяться как $M \to M^*$, правила такого изменения рассмотрены ниже.

В зависимости от разрешений на действия исполнимого кода активных данных возможны 2 варианта реализации ВМ:

- активным данным запрещено модифицировать ВМ (соответствует уровню AD_2):

$$\forall i \in I \delta(s, i) : M \to M, \tag{7}$$

- активным данным разрешено модифицировать ВМ (соответствует уровню AD_3):

$$\exists i \in I \delta(s,i) : M \to M^*, \tag{8}$$

где M^* – модифицированная ADVM.

При втором варианте модификация машины может выполняться как после каждой выполненной инструкции (непосредственный режим), так и только после серии модифицирующих инструкций (транзакционный режим):

$$M \xrightarrow{i^\alpha i_n \ldots i_{n+m} \circ i^\beta} M^*, \tag{9}$$

где i^α и i^β – специальные инструкции начала и конца блока модификации ADVM соответственно, определяющие транзакцию. Следует заметить, что логика интерпретатора инструкций должна обеспечивать следующие условия:

$$\forall i, i \notin \left\{i^\alpha, i^\beta\right\} \delta(s,i) : M \to M. \tag{10}$$

Транзакционный режим модификации ADVM является более безопасным, но при этом более ресурсоемким, так как требует создания после инструкции i^α дополнительных экземпляров ADVM, работающих в непосредственном режиме, и переноса процесса выполнения на новый модифицированный экземпляр после выполнения инструкции i^β. С точки зрения ADVM инструкцию i^α будем называть инструкцией запуска реконфигурации, инструкцию i^β – инструкцией применения новой конфигурации. Иллюстрация транзакционного и непосредственного режимов модификации ADVM приведена на рис. 3.

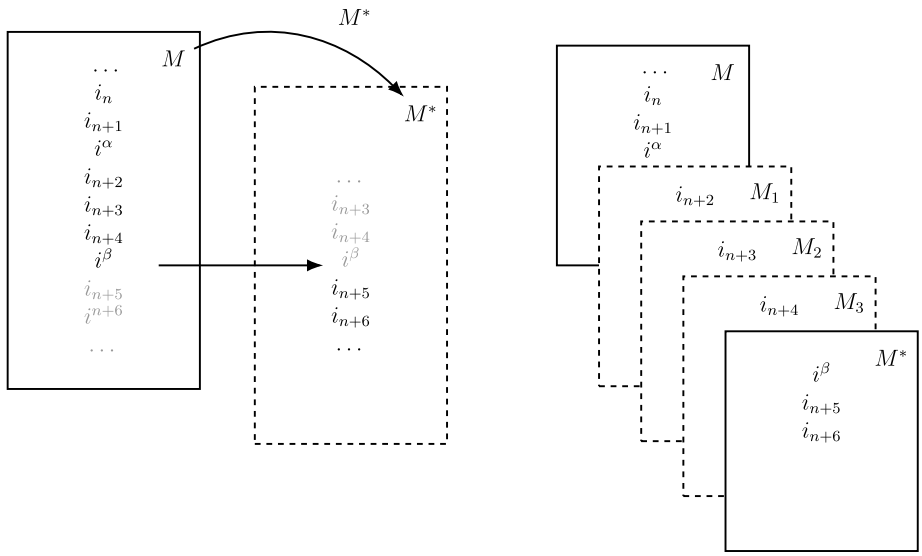

Рис. 3. Транзакционный и непосредственный режимы модификации ADVM

В процессе интерпретации входных символов из множества T формальный вычислитель M может изменяться, при этом конечный формальный вычислитель $M^* = (S^*, I^*, \delta^*, s_0^*, {S_f}^*)$ может как однозначно определяться только набором входных символов независимо от состояний, так и зависеть от состояния $s_{last} \in S$, в котором находился формальный вычислитель M непосредственно перед выполнением инструкции начала реконфигурации i^α. Первый режим можно назвать S-инвариантным

режимом реконфигурации, а второй, соответственно, S-зависимым режимом. Существенное значение для снижения рисков возникновения ошибок при разработке программного обеспечения имеет состояние s_0^* в котором будет находиться ADVM после выполнения инструкции применения новой конфигурации i^β:

$$\delta\left(s_n, i^\beta\right) = \left(s_0^*, i_{n+1}^*\right), \tag{11}$$

при этом инструкция i_{n+1} после выполнения инструкции применения конфигурации интерпретируется как инструкция i_{n+1}^* из нового множества инструкций I^* машины M^*, что также должно учитываться программистом. Для реализации защиты исполнимого кода от реверс-инжиниринга путем копирования области памяти, содержащей фрагмент исполнимого кода при инициализации узла, производится псевдослучайная генерация бинарного описания набора инструкций I^* и формируется пара программ с данным набором инструкций – программа, реконфигурирующая ADVM в формальный вычислитель M^* и целевая программа, использующая набора инструкций I^*.

4. Описание жизненного цикла устройства, использующего исполнитель с динамическим набором команд

Рассмотрим вариант практического применения описанной технологии на примере устройства, которое предварительно подготавливается в доверенной среде [2], а затем переносится во враждебную среду. Этапы жизненного цикла такого устройства с применением уникального исполнителя с динамической системой команд приведены в таблице.

Таблица

Этапы жизненного цикла устройства

Этап	Описание этапа	Место выполнения
0	Генерация системы команд, подготовка конфигурации уникального исполнителя и трансляция целевой программы	Защищенный контур, вне устройства
1	Загрузка конфигурации стандартного исполнителя после начальной инициализации устройства	Защищенный контур, на устройстве
2	Получение пакета активных данных, содержащих конфигурацию уникального исполнителя и целевую программу. Инициализация уникального исполнителя	На устройстве
3	Нормальная работа устройства путем выполнения целевой программы на уникальном исполнителе	На устройстве
4	Перехват мобильного устройства и создание дампа памяти, содержащей целевую программу	На устройстве

На подготовительном этапе (этап 0) по Random Seed на основе генератора псевдослучайных чисел производится генерация пары: машинных кодов для системы команд и конфигурации ПЛИС, содержащий описание исполнителя (уникального исполнителя, U-CPU), работающего с этой системой команд. Далее производится трансляция целевой программы в байт-код в сгенерированное уникальное представление (рис. 4).

На первом этапе после включения питания устройства производится инициализация стандартного процессорного ядра (это может быть процессорное ядро на базе

Рис. 4. Этап 0: генерация случайной системы команд и конфигурации ПЛИС

ресурсов ПЛИС, инициализированное конфигурацией ПЛИС, загруженной из ПЗУ, либо ядро аппаратного ядра внешнего микроконтроллера). Программный код содержит обработку пакетов активных данных для возможности дальнейшей реконфигурации системы (рис. 5).

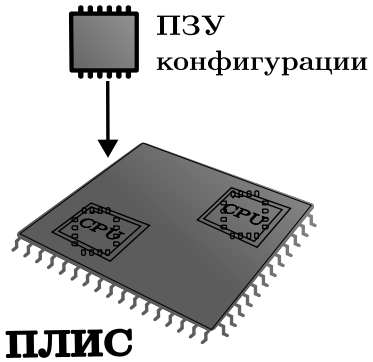

Рис. 5. Этап 1: загрузка конфигурации стандартного исполнителя после начальной инициализации устройства

На этапе 2 производится передача в устройство пакета активных данных (АД), содержащего конфигурацию уникального исполнителя и целевую программу. Пакет активных данных производит реконфигурацию ПЛИС, формируя ядро уникального исполнителя (уникального CPU) с уникальной системой команд и передает ему управление для исполнения целевой программы с уникальной системой команд (рис. 6). Стандартный исполнитель либо становится недоступным после реконфигурации ПЛИС, либо не используется (в случае отдельного физического ядра).

Дальнейшая нормальная работа устройства производится с использованием уникального исполнителя (рис. 7). Целевая программа, код и данные который находятся в оперативной памяти, выполняется на уникальном исполнителе.

В случае физического захвата устройства с возможностью его последовательной разборки и изучения злоумышленник может получить дампы с внешней оперативной памяти (если захват произошел без выключения питания устройства), а также с ПЗУ. Снятие конфигурации уникального процессора с ПЛИС технологически невозможно [4, 18]. При этом изучение дампа программного кода целевой программы не

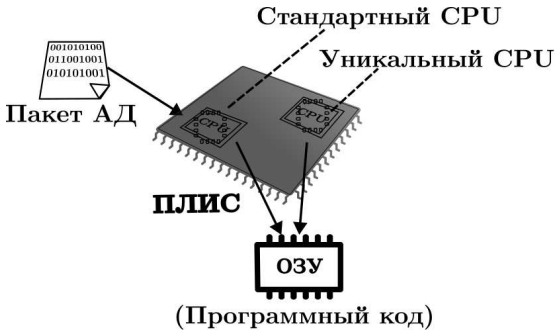

Рис. 6. Этап 2: получение пакета активных данных, содержащих конфигурацию уникального исполнителя и целевую программу. Инициализация уникального исполнителя

позволит ее декомпилировать и провести обратный инжиниринг, так как злоумышленник в принципе не может знать уникальную систему команд, используемую для целевой программы (рис. 8). Байт-код конфигурации ПЛИС в устройстве также не сохраняется.

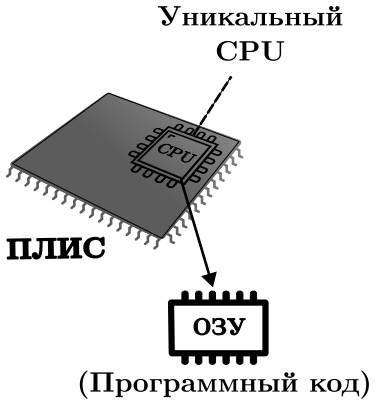

Рис. 7. Этап 3: нормальная работа устройства

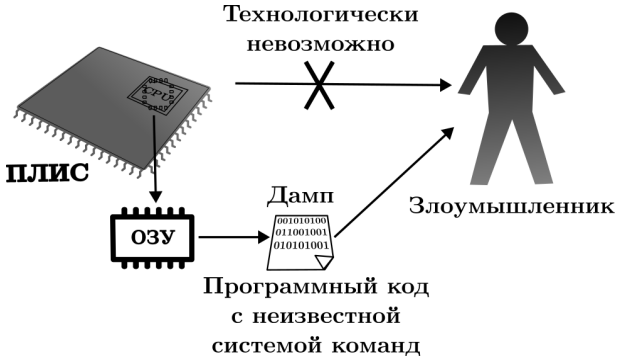

Рис. 8. Этап 4: перехват мобильного устройства и создание дампа памяти, содержащей целевую программу

Заключение

В статье рассмотрен подход к защите от реверс-инжиниринга программного кода путем получения физического доступа к одному из узлов распределенной системы и получения дампов всех видов памяти устройств, основанный на использовании

активных данных совместно с виртуальными машинами, использующими динамическую систему команд. Показано, что, в отличие от известных подходов повышения степени защищенности программного кода путем усложнения реверс-инжиниринга (включая обфускацию кода), предлагаемый подход делает реверс-инжиниринг потенциально невозможным. Соответственно, потенциал предложенного подхода основан на незнании злоумышленником системы команд и невозможности считывания текущей конфигурации из микросхемы ПЛИС. Кроме того, предложенная формализация ADVM открывает возможности для реализации программно-определяемых и программно-управляемых систем на основе технологии активных данных в рамках развития самоорганизующихся сетей и Интернета вещей. Проведена формализация и описаны уровни активных данных по степени воздействия на виртуальную машину, обеспечивающие различную степень гибкости программно-определяемых систем. Дальнейшие исследования предполагают экспериментальную апробацию предложенных подходов и методов.

Работа выполнена в рамках реализации Государственного задания на FFZF-2025-0003.

Литература

1. Кулешов, С.В. Активные данные в цифровых программно определяемых системах / С.В. Кулешов, О.В. Цветков // Информационно-измерительные и управляющие системы. – 2014. – № 6. – С. 12–19.
2. Alexandrov, V.V. Active Data in Digital Software Defined Systems Based on SEMS Structures / V.V. Alexandrov, S.V. Kuleshov, A.A. Zaytseva // Smart Electromechanical Systems. Studies in Systems, Decision and Control. – Cham: Springer, 2016. – P. 61–69.
3. Самоделов, А. Создание защищенных пользовательских приложений на базе СнК SmartFusion2 компании Microsemi. Часть 7. Доверенное программирование микросхем в недоверенном окружении. Общие положения / А. Самоделов // Компоненты и технологии. – 2018. – № 2. – С. 56–62.
4. Гладкий, В.П. Защита кода проекта для ультразвуковых медицинских приборов, построенных на ПЛИС типа FPGA, при помощи генератора ПСП / В.П. Гладкий // Вестник Астраханского государственного технического университета. Серия: Управление, вычислительная техника и информатика. – 2009. – № 2. – С. 166–171.
5. Niranjana, R. FPGA Security: Challenges and Best Practices / R. Niranjana. – URL: https://fpgainsights.com/fpga/fpga-security-challenges-and-best-practices/ (дата обращения: 18.10.2024)
6. Коковин, В.А. Аппаратные методы кибербезопасности при взаимодействии распределенных устройств автоматизированной системы / В.А. Коковин, П.Н. Кривцов, А.Н. Сытин, С.У. Увайсов // Инновационные, информационные и коммуникационные технологии: сборник трудов XX Международной научно-практической конференции. – Махачкала, 2023. – С. 226–231.
7. Baetoniu, C. FPGA IFF Copy Protection Using Dallas Semiconductor/Maxim DS2432 Secure EEPROMs / C. Baetoniu. – URL: https://docs.amd.com/v/u/en-US/xapp780 (дата обращения: 18.10.2024)
8. Paul, S. PGA Configuration Protection and Control Using Hardware Watchdog Timer / S. Paul, A. Garcia, M. Capellaro. – Patent U.S. № US7971051B2, 28.06.2011.
9. Комолов, Д. Использование микросхем специальной памяти для обеспечения защиты FPGA от копирования / Д. Комолов, Р. Золотухо // Компоненты и технологии. – 2008. – № 12. – С. 24–26.

10. Александров, В.В. Этерификация и терминальные программы / В.В. Александров, С.В. Кулешов // Информационно-измерительные и управляющие системы. – 2008. – № 10. – С. 50–53.

11. Koppe, P. Reverse Engineering x86 Processor Microcode / P. Koppe, B. Kollenda, M. Fyrbiak, at el. // Proceedings of the 26th USENIX Security Symposium. – Vancouver, 2017. – 19 p.

12. Kuleshov, S. Distributed System of Virtual Machines for Self-Organized Networks / S. Kuleshov, A. Zaytseva, I. Shalnev // Information and Control Systems. – 2019. – № 5. – P. 30–37.

13. Архипов, П.А. Алгоритм распределенного тестирования веб-приложений на основе технологий ВЕБ-прокси и активных данных / П.А. Архипов, А.С. Галкин, Д.О. Маркин // Информационные системы и технологии. – 2018. – № 1. – С. 93–101.

14. Шальнев, И.О. Подход к построению распределенных систем на основе балансировки объема исполняемого кода между узлами / И.О. Шальнев // Технологическая перспектива в рамках Евразийского пространства: новые рынки и точки экономического роста: материалы 4-й Международной научной конференции. – СПб, 2018. – С. 165–172.

15. Craig, I.D. Virtual Machines / I.D. Craig. – London: Springer, 2006.

16. Yunhe Shi. Virtual Machine Showdown / Yunhe Shi, D. Gregg, A. Beatty, M.A. Ertl // ACM Transactions on Architecture and Code Optimization. – 2008. – № 4. – P. 1–36.

17. Sinnathamby, M. Stack Based vs Register Based Virtual Machine Architecture, and the Dalvik VM / M. Sinnathamby // Engineering The Code. – URL: https://markfaction.wordpress.com/2012/07/15/stack-based-vs-register-based-virtual-machine-architecture-and-the-dalvik-vm/ (дата обращения: 18.10.2024)

18. Kuleshov, S.V. The Analysis of Cybersecurity Problems in Distributed Infocommunication Networks Based on the Active Data Conception / S.V. Kuleshov, A.Y. Aksenov, I.I. Viksnin // Cyber-Physical Systems and Control. Lecture Notes in Networks and Systems. – Cham: Springer, 2020. – P. 491–499.

19. Xiaojing Zhu. PULP: Inner-process Isolation based on the Program Counter and Data Memory Address / Xiaojing Zhu, Mingyu Chen, Yangyang Zhao, Zonghui Hong, Yunge Guo // Arxiv: Cryptography and Security. – 2018. – URL: https://arxiv.org/abs/1804.03379

20. Turing, A.M. On Computable Numbers, with an Application to the Entscheidungsproblem / A.M. Turing // Proceedings of the London Mathematical Society. – 1937. – № 1. – P. 230–265

21. Turing, A.M. The Essential Turing: Seminal Writings in Computing, Logic, Philosophy, Artificial Intelligence, and Artificial Life Plus the Secrets of Enigma. – Oxford: Oxford University Press, 2004.

Сергей Викторович Кулешов, доктор технических наук, профессор РАН, лаборатория автоматизаци научных исследований, Санкт-Петербургский Федеральный исследовательский центр РАН (г. Санкт-Петербург, Российская Федерация), kuleshov@iias.spb.su.

Александра Алексеевна Зайцева, кандидат технических наук, лаборатория автоматизаци научных исследований, Санкт-Петербургский Федеральный исследовательский центр РАН (г. Санкт-Петербург, Российская Федерация), cher@iias.spb.su.

Алексей Юрьевич Аксенов, кандидат технических наук, лаборатория автоматизаци научных исследований, Санкт-Петербургский Федеральный исследовательский центр РАН (г. Санкт-Петербург, Российская Федерация), a_aksenov@iias.spb.su.

Поступила в редакцию 12 декабря 2024 г.

MSC 68P30 **DOI: 10.14529/mmp250306**

ACTIVE DATA TECHNOLOGY IN VIRTUAL MACHINES WITH DYNAMIC COMMAND SYSTEM

S.V. Kuleshov[1]***, A.A. Zaytseva***[1]***, A.Y. Aksenov***[1]
[1]St. Petersburg Federal Research Center of the Russian Academy of Sciences,
Saint-Petersburg, Russian Federation
E-mail: kuleshov@iias.spb.su, cher@iias.spb.su, a_aksenov@iias.spb.su

In some application tasks it may be necessary to protect executable code from reverse engineering by copying a memory area containing a fragment of executable code. This type of reverse engineering can be realized by physical interception of one of the nodes of a distributed system. The paper discusses a defense approach against this type of attack based on the use of active data in conjunction with virtual machines using a dynamic instruction system.

Keywords: virtual machine; dynamic command system; active data.

References

1. Kuleshov S., Tsvetkov O. Active Data in Digital Software Defined Systems. *Information-Measuring and Control Systems*, 2014, no. 6, pp. 12–19.
2. Alexandrov V.V., Kuleshov S.V., Zaytseva A.A. Active Data in Digital Software Defined Systems Based on SEMS Structures. *Studies in Systems, Decision and Control*, 2015, vol. 49, pp. 61–69. DOI: 10.1007/978-3-319-27547-5_6
3. Samodelov A. Creating Secure User Applications Based on Microsemi's SmartFusion2 SnC. Part 7. Trusted Chip Programming in an Untrusted Environment. General Provisions. *Komponenty i Tekhnologii* [Components and Technologies], 2018, vol. 199, no. 2, pp. 56–65. (in Russian)
4. Gladkiy P.V. Protecting the Project Code for Ultrasonic Medical Devices Built on FPGA Using Psp Generator. *Vestnik of Astrakhan State Technical University. Series: Management, Computer Science and Informatics*, 2009, no. 2, pp. 166–171.
5. Niranjana R. *FPGA Security: Challenges and Best Practices.* Available at: https://fpgainsights.com/fpga/fpga-security-challenges-and-best-practices/ (accessed on 17.10.2024)
6. Kokovin V.A., Krivtsov P.N., Sytin A.N., Uvaysov S.U. Hardware Methods of Cybersecurity in the Interaction of Distributed Devices of an Automated System. *Innovative, Information and Communication Technologies: Proceedings of the XX International Scientific and Practical Conference*, Makhachkala, 2023, pp. 226–231.
7. Baetoniu C. *FPGA IFF Copy Protection Using Dallas Semiconductor/Maxim DS2432 Secure EEPROMs.* Available at: https://docs.amd.com/v/u/en-US/xapp780 (accessed on 17.10.2024)
8. Paul S., Garcia A., Capellaro M. *PGA Configuration Protection and Control Using Hardware Watchdog Timer.* Patent U.S. no. 7971051B2, 28.06.2011.
9. Komolov D., Zolotukhov R. Using Special Memory Chips to Provide Copy Protection for FPGAs. *Komponenty i Tekhnologii* [Components and Technologies], 2008, vol. 89, no. 12, pp. 24–26. (in Russian)
10. Aleksandrov V.V., Kuleshov S.V. Etherification and Terminal Programs. *Information-Measuring and Control Systems*, 2008, vol. 6, no. 10, pp. 50–53. (in Russian)
11. Koppe P., Kollenda B, Fyrbiak M, et al. Reverse Engineering x86 Processor Microcode. *Proceedings of the 26th USENIX Security Symposium*, Vancouver, 2017, 19 p.

12. Kuleshov S., Zaytseva A., Shalnev I. Distributed System of Virtual Machines for Self-Organized Networks. *Information and Control Systems*, 2019, no. 5, pp. 30–37. DOI: 10.31799/1684-8853-2019-5-30-37.
13. Arkhipov P.A., Galkin A.S., Markin D.O. Distributed Web Application Testing Algorithm Based on Web Proxy and Active Data Technologies. *Information Systems and Technologies*, 2018, vol. 105, no. 1, pp. 93–101.
14. Shal'nev I.O. Building Distributed Systems Based on Balancing the Amount of Executable Code Between the Nodes. *The 4th International Scientific and Practical Conference "Technological Perspective: New Markets and Points of Economic Growth"*, Saint-Petersburg, 2018, pp. 165–172.
15. Craig I.D. *Virtual Machines*. London, Springer, 2006. DOI: 10.1007/978-1-84628-246-1
16. Shi Yunhe, Gregg D., Beatty A., Ertl M. Virtual Machine Showdown. *ACM Transactions on Architecture and Code Optimization*, 2008, vol. 4, no. 4, pp. 1–36. DOI: 10.1145/1064979.1065001
17. Sinnathamby M. Stack Based vs Register Based Virtual Machine Architecture, and the Dalvik VM. *Engineering The Code*. Available at: https://markfaction.wordpress.com/2012/07/15/stack-based-vs-register-based-virtual-machine-architecture-and-the-dalvik-vm/ (accessed on 18.10.2024)
18. Kuleshov S.V., Aksenov A.Y., Viksnin I.I., Laskus E.O., Belyaev V.V. The Analysis of Cybersecurity Problems in Distributed Infocommunication Networks Based on the Active Data Conception. *Lecture Notes in Networks and Systems*, 2019, vol. 95, pp. 491–499. DOI: 10.1007/978-3-030-34983-7_48
19. Xiaojing Zhu, Mingyu Chen, Yangyang Zhao, Zonghui Hong, Yunge Guo. PULP: Inner-Process Isolation Based on the Program Counter and Data Memory Address. *Arxiv: Cryptography and Security*, 2018. Avialable at: https://doi.org/10.48550/arXiv.1804.03379. DOI: 10.48550/arXiv.1804.03379
20. Turing A.M. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 1937, no. 1, pp. 230–265. DOI: 10.1112/plms/s2-42.1.230
21. Turing A.M. *The Essential Turing: Seminal Writings in Computing, Logic, Philosophy, Artificial Intelligence, and Artificial Life Plus the Secrets of Enigma*. Oxford, Oxford University Press, 2004.

Received December 12, 2024