ОСОБЕННОСТИ ПОЛУЧЕНИЯ ИНФОРМАЦИИ О ХОДЕ ВЫПОЛНЕНИЯ ПРОГРАММЫ (ТРАССЫ) С ИСПОЛЬЗОВАНИЕМ АППАРАТНОГО ОКРУЖЕНИЯ

Поляков С. А.[1], Карасев С. В.[2]

В статье описывается метод, позволяющий получить информацию о ходе выполнения программы с использованием аппаратных возможностей современных микропроцессорных систем. Подробно рассмотрен способ вывода информации на основе стандарта ETMARM. Предложен подход, позволяющий предварительно обрабатывать полученный результат с помощью функций темпоральной логики и определять необходимый путь анализа трассы. Описаны ограничения использования данного метода и направления дальнейших исследований в указанной области. Предложенный метод позволяет снизить временные затраты аналитика при анализе алгоритма работы исследуемого программного обеспечения.

***Ключевые слова**: трасса, модель, анализ программного обеспечения, микропроцессор, интерфейс.*

Введение

Понимание алгоритмов и механизмов, заложенных в исследуемом программном обеспечении в случае анализа хода выполнения программы (трассы), является важной задачей. Исследование трассы просто необходимо при решении практических задач, таких как анализ вредоносного кода, поиск недекларированных возможностей, восстановление протоколов обмена данными и форматов данных. Предлагаемый подход является актуальным при анализе программного обеспечения и в условиях отсутствия исходных текстов.

Существует большое количество работ, посвященных методам получения хода выполнения программ (трассы) для приложений на уровне операционной системы (в частности: Win32/64), основанных на использовании собственного отладчика операционной системы и/или попытках внедрить в исследуемый процесс исполняемый код [1]. Существуют также механизмы получения трассы с помощью методов фазинга [2]. Однако все указанные способы и механизмы имеют ограничения, связанные с достоверностью эмулируемой трассы относительно реальной, и не могут быть применимы при исследовании аппаратной периферии и модулей, для анализа особенности функционирования исполняемого кода на уровне вычислителя (микропроцессора). В статье предлагается подход, позволяющий получать трассу в условиях действия упомянутых выше ограничений.

Постановка задачи на получение информации о ходе выполнения программы

В общем случае под *трассой* понимают последовательность инструкций, выполненных на целевой архитектуре, и значения регистров на каждом шаге работы процессора, которые затем аккумулируются в единую диаграмму. В некоторых случаях анализируемая программа может исполняться несколько раз с изменением входных параметров. Принято разделять процессы получения трассы и ее анализа с целью получения информации. В настоящее время большинство исследователей уделяют основное внимание получению трассы с помощью выполнения исследуемого программного обеспечения на эмуляторах. К примеру, для архитектуры x86 эмулятором Simics [3] компании Virtutech поддерживается разнообразные архитектуры: помимо x86 входят ARM, PowerPC, MIPS, а также имеется возможность эмулировать периферию. Эмулятор обладает удобной документацией и интерфейсом, который позволяет разрабатывать приложения для реализации нестандартных задач. В 2010–2011 гг. произошла реструктуризация компании Virtutech, и ее основным акционером стала компания Intel, что существенно ограничило доступность этого продукта. Также возможно использовать эмулятор SimNow [4] производства AMD, которыйпозволяет работать только с процессорами AMD, однако вместе с тем обеспечивает более высокую достоверность. Кроме того, в

1 Поляков Сергей Александрович, кандидат технических наук, Академия ФСО России, г. Орёл, sats861@yandex.ru

2 Карасев Станислав Владимирович, Академия ФСО России, г. Орёл, ilmaglu@mail.ru

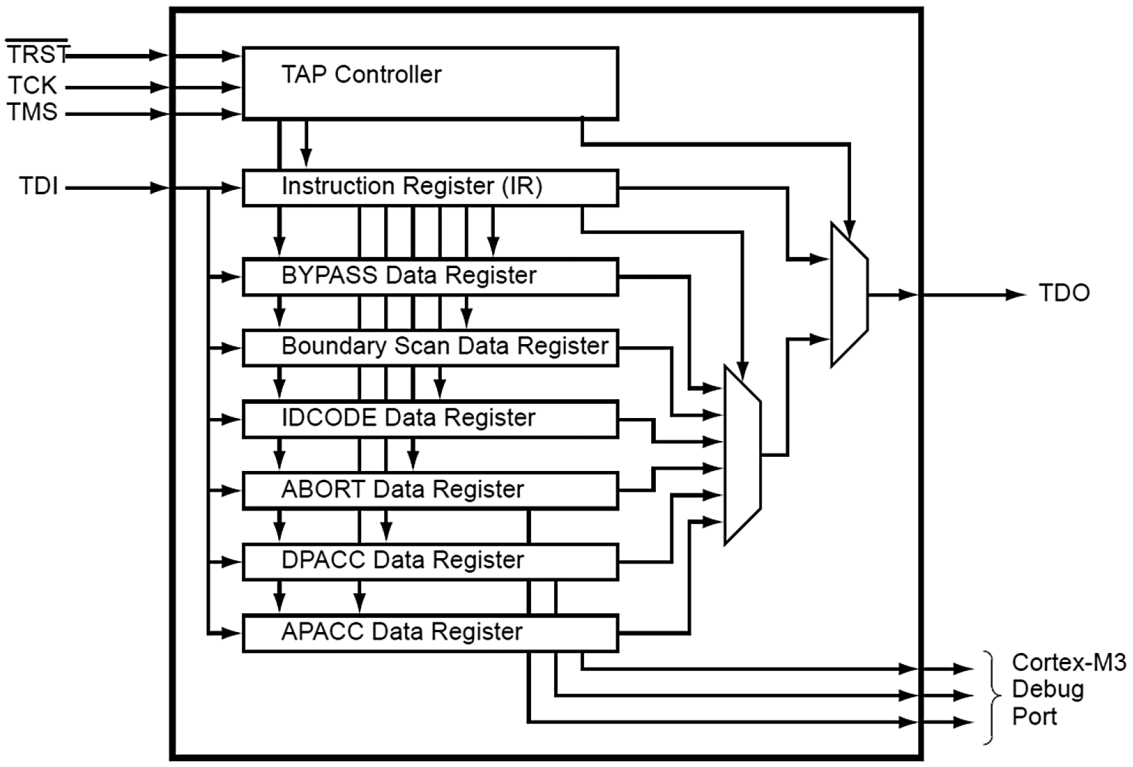

***Рис.1**. Цепь граничного сканирования ARM*

настоящее время активно применяются проекты с открытым исходным кодом, такие как Dynamips [5] и QEMU [6]. Все указанные эмуляторы имеют существенную проблему – низкую скорость работы исследуемого программного обеспечения во время снятия трассы. При исследовании программного обеспечения (ПО) на целевой платформе без взаимодействия с внешним миром такой подход допускается, однако если происходит взаимодействие с другими устройствами или не подконтрольными частями исследуемого программного обеспечения, то замедление однозначно повлияет на трассу ее выполнения.

Производители микропроцессоров для решения описанной проблемы при производстве предусматривают так называемые трассировочные порты. В настоящее время компания ARM стремительно завоевывает рынок встраиваемых систем, по этой причине в статье предлагается рассмотреть механизм получения трассы на основе ее реализаций. К сожалению, доступность устройств, реализующих сбор трассы с исследуемого ПО, оставляет желать лучшего. Компании-производители зачастую предоставляют подобные устройства при условии неразглашения о тонкостях функционирования трассировочных портов, а в обучающих материалах стараются не упоминать об их наличии.

Предлагаемое решение

В соответствии со стандартом IEEE 1149.1 все производители микропроцессоров обязуются реализовывать в своих разработках функционал JTAG (Joint Test Action Group), который изначально предназначался для тестирования качества сборки и корректности соединений. Однако в связи с простотой и большими потенциальными возможностями стандарт перерос в часть обязательную и пользовательскуючасть. В пользовательской части функционал оставляется «на откуп» производителю, который использует его не только для тестирования, но и для конфигурирования устройств, в том числе и для обновления программного обеспечения. Подробности применения стандарта достаточно хорошо описаны в [7]. Типовая схема организации цепи граничного сканирования для ARM микропроцессоров представлена на рисунке 1.

Если сравнить принцип построения стандартной цепи JTAG [8] и цепи, представленной на рисунке 1, становится понятно, что в нее дополнительно введены регистры DPACC (регистр доступа к debugport) и APACC (регистр доступа к accessport). Данные регистры предназначены для конфигурации системы трассировки ETM (EmbeddedTraceMacrocell), следовательно, конфигурация трассировочного порта осуществляется

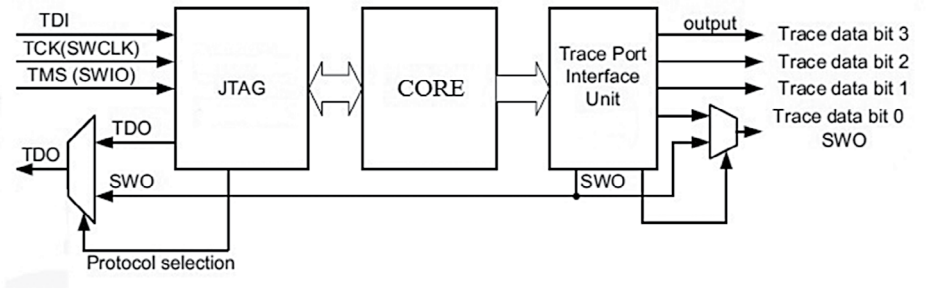

***Рис.2**. Принцип получения трассы через ETM*

через интерфейс JTAG, что стандартом [7] не предусмотрено. Принцип, описывающий механизм получения трассировочных данных, представлен на рисунке 2.

Таким образом, информация о трассе программы предоставляется через отдельный порт в параллельном виде с разными скоростями, разрядностью и различными данными, определяемыми при конфигурировании трассировочного порта через интерфейс JTAG. Получение трассы через интерфейс стандарта ETM [8] имеет следующие отличительные особенности:

1) не задействуются вычислительные ресурсы микропроцессора;

2) доступ к дополнительным ресурсам (системный таймер, память ит.д.);

3) скорость передачи 800 Мбит/с.

Следовательно, использование указанного интерфейса однозначно решает вопрос по установлению достоверности полученной трассы с помощью эмулятора реальной трассе, что позволяет исключить вероятность неправильного определения пути следования программы при получении трассы. В общем случае трасса выполнения программы – T, представляет собой графическую диаграмму (представление), содержащую различное количество операторов, выстроенных в соответствии с последовательностью их выполнения. Пример подобной диаграммы представлен на рисунке 3 а).

При предварительном анализе необходимо определиться с узлом трассы, который подлежит изучению, и произвести отбор в соответствии с рисунком 3б). Предлагается производить отбор по адресации, формируя при этом множество допустимых адресов с определением данного множества и формированием его границ аналитиком. В связи с тем, что в предлагаемом представлении в каждый момент времени может быть несколько альтернатив дальнейшего развития событий, а время представляется ветвящейся структурой, целесообразно для дальнейшего анализа использовать темпоральную логику, в частности логику деревьев вычислений.

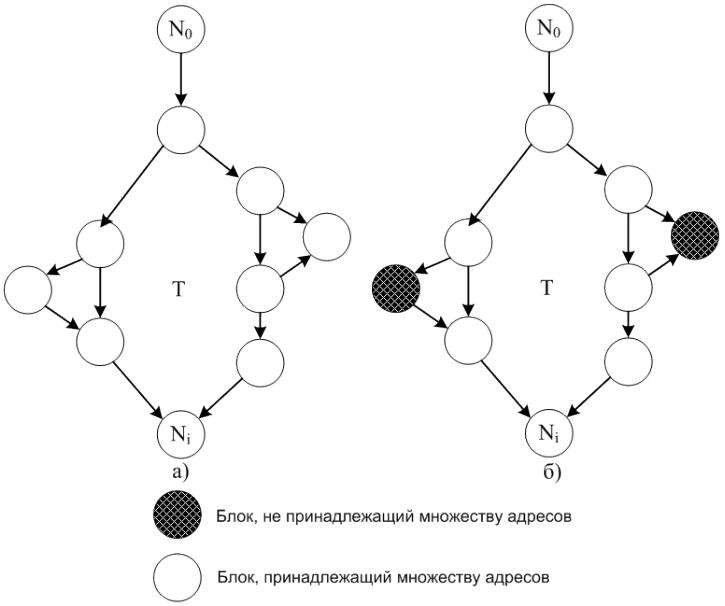

***Рис. 3**. Графическое представление трассы, N_i – оператор трассы* T

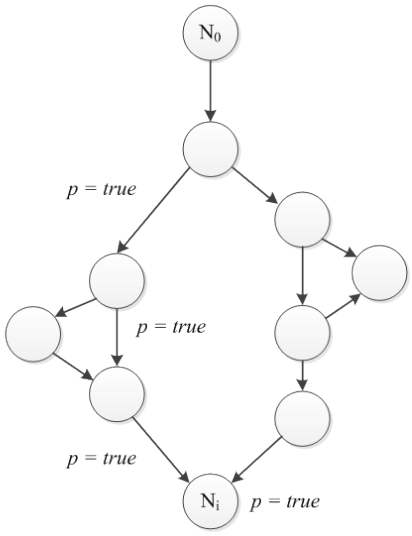

***Рис.4**. Выбор пути для предиката*

Таким образом, с помощью структуры Крипке [9] возможно из множества состояния N определить единственный путь, для которого искомое предположение является истиной. Представим рисунок 3 в значениях темпоральной логики на основе структур Крипке (деревьев вычислений).

Пусть для трассы T формула (1) будет является моделью Крипке:

$$T = \langle N, \boldsymbol{L}, p \rangle, \quad (1)$$

где $N_0 \in N$ - является одним из состояний модели.

Тогда запись формулы будет следующей:

$$T, N \models \exists \Box p. \quad (2)$$

Формула (2) говорит о том, что для трассы T результатом работы модели Крипке будет существовать одна ветвь, исходящая из состояния в каждом состоянии которой верно отношение выполнимости к предикату (3):

$$T, N \models p. \quad (3)$$

В соответствии с предложенным подходом графическое представление трассы отображено на рисунке 4.

Таким образом, предлагаемый подход позволяет получать трассу выполнения программы, проводить ее предварительный анализ и выделять необходимый путь в графическом представлении, что дает возможность повысить результативность в понимании процессов, происходящих в программе.

Ограничения

Существенным ограничением на использование предложенного подхода является то, что множество возможных состояний определяется разрядностью процессора и количество элементов этого множества должно соответствовать набору инструкций целевой исследуемой системы.

Выводы

В материалах статьи предложен один из вариантов решения задачи, касающийся получения трассы выполнения программы в реальном времени. Описаны проблемы, возникающие при получении трассы на эмуляторе, что связано с расхождением реальной трассы с эмулируемой. В связи с этим предлагается способ обработки полученной информации о трассе с помощью модели Крипке. Предложено аналитическое описание модели, позволяющее обрабатывать полученные данные и определять кратчайшие пути в дереве ветвления событий.

В качестве направления дальнейших исследований рекомендуется произвести преобразование модели Крипке в автомат Бюхи для оценки вероятности возникновения коллизий на пересечении узлов трассы.

***Рецензент**: Овсянкин Сергей Владимирович, кандидат технических наук, сотрудник Академии ФСО России, Орел.*

Литература:

1. Падарян В.А., Гетьман А.И., Соловьев М.А. Программная среда для динамического анализа бинарного кода // Труды Института системного программирования РАН. 2009. Т. 16. С. 51-72.
2. Тихонов А.Ю., Аветисян А.И. Комбинированный (статический и динамический) анализ бинарного кода // Труды Института системного программирования РАН. 2012. Т. 22. С. 131-152. DOI: 10.15514/ISPRAS-2012-22-9.
3. Takanen, Ari, Jared D. Demott, and Charles Miller. Fuzzing for software security testing and quality assurance. Boston: Artech House, 2008. Print.
4. FullSystemSimulation. URL: http://www.windriver.com/products/simics/ (дата обращения 01.05.2016).
5. SimNow™ Simulator. URL: http://developer.amd.com/tools-and-sdks/cpudevelopment/simnow/ (дата обращения 01.05.2016).
6. GNS3 / dynamips. URL: http://github.com/GNS3/dynamips (дата обращения 01.05.2016).
7. QEMU – Open Source Processor Emulator. URL: http://wiki.qemu.org/Main_Page (дата обращения 10.05.2016).
8. 1149.1-2013 - IEEE Standard for Test Access Port and Boundary-Scan Architecture.
9. ARM Software development tools. URL: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dud/Chdcdbib.html (дата обращения 01.05.2016).
10. Emerson E. A., Clarke E. M. Using branching time temporal logic to synthesizesynchronisation skeletons // Science of Computer Programming 2. 1982, 241-266 pp.

APPLICATION OF HARDWARE FOR CHECK PROGRAM EXECUTION TRACE

Polyakov S.A.[3], Karasev S.V.[4]

This paper presents a method of using hardware for check information of a program execution trace. It describes the experience of using ARM ETM tracing standard. To solve the problems ofpre–process the re–sult of execution trace with using a temporal logic functions. The described limitations of using of this method and directions for further research in this sciences. Function models are combined into a high level model algorithm that can be used to restore or analyse it.

Keywords*: execution trace, model, dynamic analysis, microprocessor, interface.*

References:

1. Padaryan V.A., Get'man A.I., Solov'yev M.A. Programmnaya sreda dlya dinamicheskogo analiza binarnogo koda, Trudy Instituta sistemnogo programmirovaniya RAN. 2009. T. 16. S. 51-72.
2. Tikhonov A.Yu., Avetisyan A.I. Kombinirovannyy (staticheskiy i dinamicheskiy) analiz binarnogo koda, Trudy Instituta sistemnogo programmirovaniya RAN. 2012. T. 22. S. 131-152. DOI: 10.15514/ISPRAS-2012-22-9.
3. Takanen, Ari, Jared D. Demott, and Charles Miller. Fuzzing for software security testing and quality assurance. Boston: Artech House, 2008. Print.
4. FullSystemSimulation. URL: http://www.windriver.com/products/simics/
5. SimNow™ Simulator. URL: http://developer.amd.com/tools-and-sdks/cpudevelopment/simnow/ (data obrashcheniya 01.12.15).
6. GNS3 / dynamips. URL: http://github.com/GNS3/dynamips (data obrashcheniya 01.12.15).
7. QEMU – Open Source Processor Emulator. URL: http://wiki.qemu.org/Main_Page
8. 1149.1-2013 - IEEE Standard for Test Access Port and Boundary-Scan Architecture.
9. ARM Software development tools. URL: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dud/Chdcdbib.html
10. Emerson E. A., Clarke E. M. Using branching time temporal logic to synthesizesynchronisation skeletons // Science of Computer Programming 2. 1982, 241-266 pp.

3 Sergey Polyakov, Ph.D., The Academy of the Federal Guard Service of the Russian Federation, Orel, sats861@yandex.ru
4 Stanislav Karasev, The Academy of the Federal Guard Service of the Russian Federation, Orel, ilmaglu@mail.ru