О представлении результатов обратной инженерии бинарного кода ⋆

В.А. Падарян <vartan@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25
Московский государственный университет имени М.В. Ломоносова,
119991 ГСП-1, Москва, Ленинские горы

Аннотация. В статье рассматривается вопрос представления кода алгоритмов, извлекаемых из бинарного кода в рамках задачи обратной инженерии: как промежуточные представления для автоматического анализа, так и конечные представления, передаваемые пользователю. Разбираются две ключевых подзадачи в области обратной инженерии: автоматический поиск эксплуатируемых дефектов и выявление НДВ. Описана общая схема системы, реализующей автоматический поиск эксплуатируемых дефектов, указаны ключевые свойства промежуточного представления, позволяющего решать такую задачу с точки зрения эффективной генерации системы уравнений для SMT-решателя. Представлен тракт работы системы по выявлению НДВ, состоящий из трех этапов: локализация алгоритма, его представление в удобной для анализа форме и исследование его свойств. Для автоматизации первого этапа применяется построение статико-динамического представления, выделяются события уровня ОС и вызовы библиотечных функций, являющиеся «якорями», от которых отталкивается аналитик при локализации алгоритма. Дальнейшая поддержка локализации осуществляется за счет построения срезов и средств навигации. После того, как локализация алгоритма выполнена, дальнейшая работа разделяется на два направления: диалоговое построение компактного аннотированного представления алгоритма в виде блок-схемы и автоматизированное изучение свойств алгоритма в части определения декларированных и недекларированных потоков данных. Представление алгоритма в виде блок-схемы базируется на построении упрощенных моделей функций, учитывающих входные и выходные буферы, и автоматически выявляемыми связями по данным между буферами различных вызовов функций. Описан общий сценарий работы аналитика с подобной блок-схемой в контексте задачи выявления НДВ, основанный на аннотировании декларированных потоков данных и автоматическом выявлении недекларированных потоков. В завершение статьи приводится пример получаемого представления и перечисляются направления дальнейшей работы.

⋆ Работа поддерживается грантом РФФИ № 16-29-09632

Ключевые слова: бинарный код; комбинированный анализ; промежуточное представление.

DOI: 10.15514/ISPRAS-2016-29(3)-3

Для цитирования: Падарян В.А. О представлении результатов обратной инженерии бинарного кода. Труды ИСП РАН, том 29, вып. 3, 2017 г., стр. 31-42. DOI: 10.15514/ISPRAS-2016-29(3)-3

1. Введение

Необходимость в анализе безопасности бинарного (исполняемого) кода возникает, когда требуется оценить фактические свойства программы: соответствие заявленным возможностям, отсутствие программных закладок, наличие дефектов и возможности их эксплуатации. Проведение анализа на уровне бинарного кода обусловлено рядом причин. Исходный код может быть потерян или недоступен. Характерные примеры такой ситуации – наследственные системы с частично или полностью утраченной документацией, сторонние библиотеки, вредоносное ПО. Оптимизирующие преобразования кода, проводимые современными компиляторами, способны внести серьезные и потенциально эксплуатируемые дефекты, когда сталкиваются с определенными аспектами поведения программы, выходящими за рамки спецификации языка программирования [1].

Волна исследовательских работ последних 10 лет [2, 3, 4] предлагала новый подход к исследованию безопасности исполняемого кода. Основой разработанных методов стали компиляторные технологии, примененные в обратном направлении, когда входными данными оказывается исполняемый файл, низкоуровневые команды транслируются в промежуточное представление, которое затем анализируется с привлечением классических компиляторных алгоритмов, таких как распространение констант, достижимые определения и др.

Проводимое в 2016 году соревнование Darpa Grand Cyber Challenge [6] выявило победителей, работы которых [5, 4] реализуют приведенный выше подход.

В ИСП РАН на протяжении ряда лет разрабатывается среда анализа бинарного кода [7], базирующаяся на комбинированном подходе, сочетающем динамический и статический анализ. Методы анализа и программные инструменты, реализующие этот подход, широко задействуют компиляторные технологии. В статье рассматриваются некоторые разработанные представления, как промежуточные, рассчитанные на автоматический анализ, так и передаваемые пользователю для интерактивной работы.

2. Выявление эксплуатируемых дефектов

Технологии автоматизированного поиска дефектов в последние годы качественно развились, претендуя на применение в промышленности. Основанием для развития стали два подхода: символьное выполнение на

уровне бинарного кода и фаззинг. Типовой механизм символьного выполнения совмещает конкретное и символьное состояние программы. Символьные значения заводятся только для тех переменных, на которые способны повлиять входные данные, что способствует сохранению относительно небольшого числа переменных и компактности выражений над ними. Отслеживание влияния со стороны входных данных сводится к классической задаче анализа помеченных данных, для которой по-прежнему актуальны проблемы избыточной и недостаточной помеченности.

Поиск дефекта средствами символьной интерпретации заключается проверке совместимости системы условий (уравнений и неравенств) над символьными переменными и нахождением ее решения. Система состоит из предиката пути, задающего условия выполнения определенной последовательности команд, и предиката безопасности, описывающего срабатывания некоторого программного дефекта.

Можно утверждать, что уже устоялась архитектура такого средства поиска дефектов (рис. 1). Выполнение программы, сопровождающееся поддержкой символьного состояния, обеспечивается средствами бинарной инструментации (Valgrind [8], Pin [9], QEMU [10]). Та часть исполняемого кода, которая обрабатывает символьные данные, транслируется в архитектурно независимое представление низкого уровня. Можно описать его как код RISC-машины с минимизированными или полностью исключенными побочными эффектами. Следует отметить, что возможна ситуация и с обратным порядком, когда весь код транслируется в промежуточное представление и уже на нем ведется отслеживание символьных значений.

Операции над символьными переменными приводят к обновлению символьного состояния. Условные переходы формируют предикат пути – систему ограничений над символьными переменными. В целях упрощения последующего решения предикаты формируют в рамках теории QFBV. Предикат пути, дополненный предикатом безопасности, передают SMT-решателю. Возможности современных SMT-решателей, таких как Z3 [11], позволяют на настольном компьютере за приемлемое время оценивать совместность системы из нескольких десятков тысяч формул.

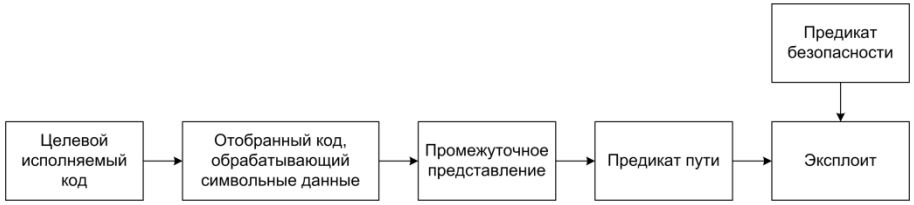

Рис. 1. Последовательность преобразований исполняемого кода при поиске уязвимостей средствами символьного выполнения

Fig. 1. Sequence of transformations of executable code when searching for vulnerabilities by means of symbolic execution

Ключевая особенность описанной архитектуры – трансляция в промежуточное представление (BAP [3], VINE [2], Pivot [12], REIL [13] и др.), которое, затем транслируется в SMT-формулы. Характерная особенность всех упомянутых представлений – крайне малое числом команд, что позволяет задать краткие правила построения SMT-формул. Представление Pivot, например, содержит всего 8 различных операторов. Таким образом, поддержка новой процессорной архитектуры ограничивается разработкой транслятора в промежуточное представление.

Открытой проблемой остается формулировка предикатов безопасности для актуальных видов дефектов [14]. В подавляющем числе открытых публикаций приводятся предикаты, описывающие конкретный механизм эксплуатации дефекта, неприменимый к современным промышленным программам. В научных статьях популярны такие дефекты, как переполнение буфера на стеке или контролируемая форматная строка. При этом игнорируются повсеместно применяемые защитные механизмы, такие как ASLR, DEP, противодействие переполнению буфера со стороны компилятора. Без учета этих факторов точное ранжирование опасности найденных дефектов невозможно.

После подготовки анализируемых материалов поиск программных дефектов не требует участия человека. Тем не менее, для окончательной оценки опасности найденного дефекта необходим ручной экспертный анализ.

3. Выявление НДВ

Возможности автоматизации для поиска программных закладок гораздо скромнее. Исследователь вынужден заниматься ручным анализом бинарного кода, который условно разделяется на три этапа. Необходимо локализовать алгоритм в общей массе кода, представить его в удобной форме и провести исследование свойств. Полная автоматизация первого и последнего этапа невозможна, требуется поддержка действий аналитика.

В ИСП РАН были получены результаты [7], продемонстрировавшие возможность предварительного повышения уровня представления бинарного кода, в рамках комбинированного анализа. По трассам выполнения восстанавливается статическое представление программ с сохранением связи между инструкциями представления и шагами трассы. Предложенный подход не накладывает требований по априорным знаниям об окружении (устройство ОС, язык и система программирования), в котором работает анализируемый алгоритм. Для поддержки анализа кода новой платформы будет достаточно знания о семантике инструкций процессорной архитектуры и карты адресного пространства. Разработанные программные средства позволяют восстановить события уровня ОС и системы программирования, облегчают навигацию. В полученном статико-динамическом представлении выявляются зависимости по управлению, что способствует более точному восстановлению потоков данных и управления.

После проведения предварительного повышения уровня представления трасса размечается, становится доступен поиск вызовов функций, в месте вызова у библиотечных функций восстанавливаются значения параметров. Автоматизированная навигация по вызовам библиотечных функций позволяет найти в трассе выполнения «якорь»: как правило, это некоторая функция ввода/вывода, через нее передаются результаты работы искомого алгоритма.

Посредством отслеживания потока данных в обратном направлении (обратного слайса трассы) выделяется подпоследовательность шагов, участвующих в выработке результата, эти шаги будут отнесены к искомому алгоритму.

Поиск НДВ затрагивает определенные алгоритмы, реализованные в программе; их выбор обусловлен проводимыми тематическими исследованиями. Как правило, выбираются алгоритмы, работающие с чувствительными данными. Для каждого выбранного алгоритма требуется определить, как вырабатывались результаты, какие входные данные были для этого использованы, где именно были размещены результаты, не сопровождалось ли это утечкой чувствительных данных. Помимо того, преобразованиям, проводимым над входными данными необходимо дать некоторую интерпретацию, выраженную в виде аннотации на естественном языке. Поскольку вопрос оценки семантики кода сводится к определению эквивалентных функций, в общем случае он неразрешим. Выработка аннотации остается за человеком. Однако выявление входных и выходных данных, определение зависимостей между ними успешно автоматизируются.

Наиболее близкой работой по высокоуровневому представлению алгоритма является гибридный граф потока данных и управления (Hybrid Information and Control Flow Graph, HI-CFG) [15]. HI-CFG содержит вершины двух типов: вершины, соответствующие фрагментам кода программы, и вершины, соответствующие некоторым структурам данных. Ребра, связывающие вершины первого типа, отражают последовательность передачи управления, а ребра, связывающие вершины второго типа, отражают зависимости по данным. Кроме того, в графе есть специальные виды ребер *производитель*, связывающие участок кода с порождаемой им структурой данных, и *потребитель*, связывающие структуры данных с использующими их участками кода. HI-CFG поддерживает различные уровни детализации: вершины первого типа могут представлять как базовые блоки кода, так и функции, вершины второго типа – как отдельные ячейки памяти, так и крупные ее области.

В отличие от HI-CFG, разработанное в ИСП РАН представление изначально рассчитано на то, что работа с ним будет вестись по двум направлениям. Первое – диалоговое построение компактного аннотированного представления в виде иерархической блок-схемы. Второе – автоматизированное изучение свойств алгоритма в части определения декларированных и недекларированных потоков данных. Помимо того, в представление включается информация о восстановленном интерфейсе структурных компонент алгоритма – так называемых моделей функций.

Построение представления начинается с восстановления графа зависимостей по данным для подмножества команд трассы, относящихся к описываемому алгоритму. Граф – двудольный, вершины в нем – выполнявшиеся команды и использующиеся ими на запись и чтение ячейки памяти и регистры. Группировка узлов графа управляется посредством проекции на него восстановленного дерева вызовов.

Для описания семантики блоков кода, выполняющихся в исследуемой программе, используется понятие моделей. Модель функции – это её формальное упрощенное описание, позволяющее сгруппировать ячейки данных, с которыми взаимодействует функция, в несколько блоков данных (буферов), соответствующих входным и выходным параметрам функции.

Использование моделей функций позволяет аналитику описывать семантику отдельных функций и их параметров. При этом потоки данных между параметрами экземпляров моделей отслеживаются автоматически.

Начальным источником аннотаций выступает набор ранее описанных моделей библиотечных функций. Более формально под моделью понимается полностью восстановленная функция, описываемая следующими атрибутами: имя функции, модуль, в котором она располагается, смещение команды точки входа в функцию от начала модуля, множество команд выхода из функции, задаваемое смещениями от начала модуля, множество именованных параметров, зависимости между параметрами. Каждый параметр, в свою очередь, описывается атрибутами: имя, тип (входной, выходной, входной/выходной), вырабатываемое значение. Последний атрибут представляет собой арифметическое выражение над адресуемыми ячейками памяти и регистрами. Значения регистров и ячеек памяти берутся в точке входа, точке выхода или в явно заданной команде, соответственно для всех типов параметров.

Модель строится либо вручную по результатам изучения функции, либо автоматизировано при наличии заголовочных файлов и документации к ним. На рис. 2 приведен снимок окна, используемого для управления моделями функций. На снимке выделена модель функции CreateFileW, размещенной в динамической библиотеке kernel32. Функция оперирует четырьмя параметрами: тремя входными (помечены зеленым) и одним выходным (помечен красным). Следует отметить, что один из параметров (fname), является указателем; имя fname используется для описания содержимого второго параметра – буфера памяти непосредственно содержащего имя файла.

Схема алгоритма включает в себя вершины кода, данных и вспомогательные вершины.

Вершины кода бывают трех типов:

- вызовы функций с известной семантикой, то есть вызовы функций, для которых в системе были заведены модели;
- вызовы функций, не имеющих моделей;

- более мелкие фрагменты трассы (блоки, разделенные инструкциями вызова функций и возврата).

Вершины данных соответствуют параметрам моделей.

Вспомогательные вершины (соответствуют началу трассы, концу трассы, либо некоторой позиции, на которой был завершен анализ потока данных).

При этом вершины данных (параметры вызова) связаны ребром с соответствующей этому вызову вершиной-моделью, а потоки данных между параметрами представлены либо ребром графа (в случае, если выходные данные одной функции с моделью непосредственно используются другой функцией с моделью), либо же подграфом-гамаком, содержащим вызовы функций без моделей.

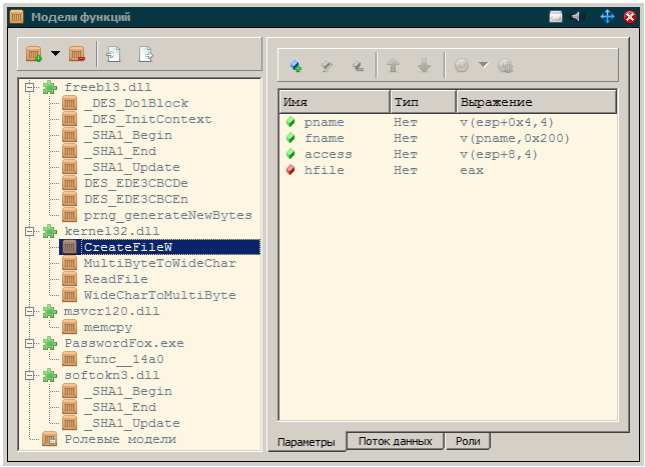

Рис. 2. Окно графического интерфейса, управляющее моделями функций
Fig. 2. The graphical interface window that controls the function models

В случае если какой-либо параметр формировался (использовался) исключительно функциями, не имеющими моделей, в качестве входной (выходной) вершины гамака будет использована вспомогательная вершина.

Особенность используемого представления заключается в том, что одному и тому же фрагменту трассы (вызову функции) может соответствовать несколько вершин, расположенных в разных гамаках.

Уже имеющиеся и дополнительно создаваемые модели применяются к графу зависимостей, сворачивая подграфы, соответствующие вызовам данной функции. В момент свертки выполняется сверка фактических и модельных потоков данных. Для каждого параметра модели выполняется вычисление его значения и происходит их сопоставление с фактическим множеством входных выходных данных. Модельные зависимости между входными и выходными параметрами сопоставляются с фактическими. Свернутый подграф аннотируется именем модели, а его входные и выходные данные – именами

параметров. Процесс изучения кода алгоритма ведется снизу-вверх и заканчивается тогда, когда построена и проверена объемлющая модель для всего отобранного кода.

На рис. 3 приведен фрагмент визуализированного графа, полученного при извлечении из исполняемого кода браузера Firefox алгоритма доступа к локальному хранилищу паролей. Образ исполняемого кода составлял 12МБ, из них 10МБ – системные библиотеки. В результате проведенного обратного слайсинга было отобрано 236 тыс. шагов трассы, при этом учитывались адресные зависимости. Для того чтобы отобранный код стал «обозримым», к представлению было применено 17 моделей функций. В слайсе было обнаружено и свернуто порядка 1500 вызовов функций, описываемых этими моделями. Результирующее представление на верхнем уровне было сведено к графу из 102 узлов.

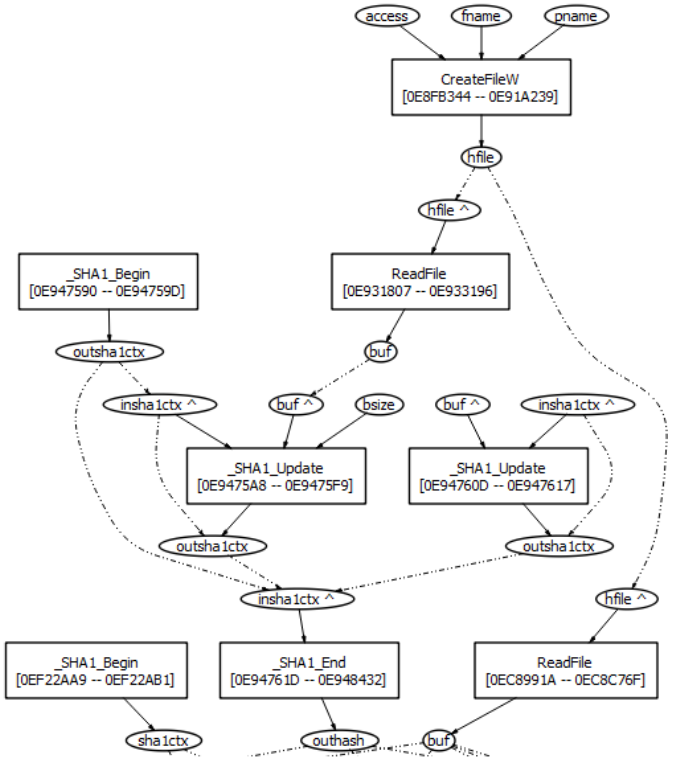

Рис. 3. Фрагмент визуализированного представления алгоритма
Fig. 3. Fragment of the visualized representation of the algorithm

4. Заключение

В работе описаны представления программ, используемые для решения двух основных задач компьютерной безопасности: поиска НДВ и уязвимостей. Представления были разработаны в рамках работ по созданию и развитию среды анализа бинарного кода.

Представления, используемые в поиске уязвимостей, полностью покрывают текущие потребности экспериментальной работы. Это позволило сдвинуть исследовательскую деятельность [14] в область разработки более точных методов оценки критичности найденных программных дефектов.

В части поиска НДВ в настоящее время ведется работа по преодолению ряда ограничений реализации инструмента, поддерживающего описанный способ анализа. В визуализируемом представлении будет добавлена поддержка зависимостей по управлению, а выражения, описывающие значение параметра, планируется развить, добавив возможность описания рекурсивных типов данных.

Список литературы

[1]. Wang X., Zeldovich N., Kaashoek M. F., Solar-Lezama A. A Differntial Approach to Undefined Behavior Detection. ACM Transactions on Computer Systems, 33(1), article 1, 2015, 29 p. DOI: 10.1145/2699678.

[2]. Song D., Brumley D., Yin H. et al. BitBlaze: A new approach to computer security via binary analysis. Information systems security, 2008, pp. 1-25.

[3]. Brumley D., Jager I., Avgerinos T. et al. BAP: A binary analysis platform. International Conference on Computer Aided Verification, 2011, pp. 463-469.

[4]. Shoshitaishvili Y., Wang R., Salls C. et al. Sok: (state of) the art of war: Offensive techniques in binary analysis. Security and Privacy (SP), 2016 IEEE Symposium on, 2016, pp. 138-157.

[5]. Cha S. K., Avgerinos T., Rebert A. et al. Unleashing mayhem on binary code. Security and Privacy (SP), 2012 IEEE Symposium on, 2012, pp. 380-394.

[6]. Defense Advanced Research Projects Agency Program Information: Cyber Grand Challenge (CGC). Доступно по ссылке: http://www.darpa.mil/program/cyber-grand-challenge, 01.06.2017.

[7]. В.А. Падарян, А.И. Гетьман, М.А. Соловьев, М.Г. Бакулин, А.И. Борзилов, В.В. Каушан, И.Н. Ледовских, Ю.В. Маркин, С.С. Панасенко. Методы и программные средства, поддерживающие комбинированный анализ бинарного кода. Труды ИСП РАН, том 26, вып. 1, 2014, стр. 251-276. DOI: 10.15514/ISPRAS-2014-26(1)-8.

[8]. Nethercote N., Seward J. Valgrind: a framework for heavyweight dynamic binary instrumentation. ACM SIGPLAN notices, 42(6), 2007, pp. 89-100.

[9]. Luk C. K., Cohn R., Muth R. et al. Pin: building customized program analysis tools with dynamic instrumentation. ACM SIGPLAN notices, 40(6), 2005, pp. 190-200.

[10]. Bellard F. QEMU, a fast and portable dynamic translator. USENIX Annual Technical Conference, FREENIX Track, 2005, pp. 41-46.

[11]. De Moura L., Bjørner N. Z3: An efficient SMT solver. Tools and Algorithms for the Construction and Analysis of Systems, 2008, pp. 337-340.

[12]. В.А. Падарян, М.А. Соловьев, А.И. Кононов. Моделирование операционной семантики машинных инструкций. Программирование, № 3, 2011, стр. 50-64.

[13]. Dullien T., Porst S. REIL: A platform-independent intermediate representation of disassembled code for static code analysis. CanSecWest, 2009, 7 pp.

[14]. А.Н. Федотов, В.А. Падарян, В.В. Каушан, Ш.Ф. Курмангалеев, А.В. Вишняков, А.Р. Нурмухаметов. Оценка критичности программных дефектов в условиях работы современных защитных механизмо. Труды ИСП РАН, том 28, вып. 5, 2016, стр. 73-92. DOI: 10.15514/ISPRAS-2016-28(5)-4.

[15]. Caselden D., Bazhanyuk A., Payer M., McCamant S., Song D. HI-CFG: Construction by Binary Analysis and Application to Attack Polymorphism. In Computer Security – ESORICS 2013. Lecture Notes in Computer Science, vol 8134. Springer pp. 164-181

On representation used in the binary code reverse engineering

V.A. Padaryan <vartan@ispras.ru>
Institute for System Programming of Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia

Abstract. The paper discusses the problem of representation of algorithms extracted from binary code in course of reverse engineering: both representations for automatic analysis and final representations for the user. Two key subproblems of reverse engineering are focused on: automatic search for exploitable defects and discovery of undeclared capabilities. A principal scheme of system that allows automatically finding exploitable defects is described, along with key features of an internal representation employed by such system from the viewpoint of efficient generation of equations for an SMT solver. A sequence of steps for a system that reveals undeclared capabilities is enumerated: algorithm localization, its representation in a form suitable for analysis, and recovery of its properties. In order to automate the first step a static-dynamic representation is built which includes OS-level events and calls to library functions that serve as “anchor points” for the analyst in course of algorithm localization. Further support for localization is provided by means of code slicing and navigation algorithms. Once the algorithm is localized, further work goes in two directions: dialogue-based building of an annotated representation of the algorithm as a flowchart and automated research of characteristics of the algorithm in terms of declared and undeclared data flows. Flowchart representation of an algorithm is based on building simplified function models which describe input and output buffers, and automatic analysis of data flows between buffers of calls of different functions. The general scenario of interaction between an analyst and such a flowchart in context of the undeclared capability revealing problem is described, based on annotating declared data flows and automatically revealing undeclared ones. The paper concludes with an example of such a representation and an enumeration of further work directions.

Keywords: binary code; combined analysis; intermediate representation

DOI: 10.15514/ISPRAS-2017-29(3)-3

For citation: Padaryan V.A. Automated vulnerabilities exploitation in presence of modern defense mechanisms. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 3, 2017. pp. 31-42 (in Russian). DOI: 10.15514/ISPRAS-2017-29(3)-3

References

[1]. Wang X., Zeldovich N., Kaashoek M. F., Solar-Lezama A. A Differntial Approach to Undefined Behavior Detection. ACM Transactions on Computer Systems, 33(1), article 1, 2015, 29 p. DOI: 10.1145/2699678.

[2]. Song D., Brumley D., Yin H. et al. BitBlaze: A new approach to computer security via binary analysis. Information systems security, 2008, pp. 1-25.

[3]. Brumley D., Jager I., Avgerinos T. et al. BAP: A binary analysis platform. International Conference on Computer Aided Verification, 2011, pp. 463-469.

[4]. Shoshitaishvili Y., Wang R., Salls C. et al. Sok: (state of) the art of war: Offensive techniques in binary analysis. Security and Privacy (SP), 2016 IEEE Symposium on, 2016, pp. 138-157.

[5]. Cha S. K., Avgerinos T., Rebert A. et al. Unleashing mayhem on binary code. Security and Privacy (SP), 2012 IEEE Symposium on, 2012, pp. 380-394.

[6]. Defense Advanced Research Projects Agency Program Information: Cyber Grand Challenge (CGC). Available at: http://www.darpa.mil/program/cyber-grand-challenge, accessed 01.06.2017.

[7]. V.A. Padaryan, A.I. Getman, M.A. Solovyev, M.G. Bakulin, A.I. Borzilov, V.V. Kaushan, I.N. Ledovskich, U.V. Markin, S.S. Panasenko. Methods and software tools for combined binary code analysis. Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 1, 2014, pp. 251-276 (in Russian). DOI: 10.15514/ISPRAS-2014-26(1)-8.

[8]. Nethercote N., Seward J. Valgrind: a framework for heavyweight dynamic binary instrumentation. ACM SIGPLAN notices, 42(6), 2007, pp. 89-100.

[9]. Luk C. K., Cohn R., Muth R. et al. Pin: building customized program analysis tools with dynamic instrumentation. ACM SIGPLAN notices, 40(6), 2005, pp. 190-200.

[10]. Bellard F. QEMU, a fast and portable dynamic translator. USENIX Annual Technical Conference, FREENIX Track, 2005, pp. 41-46.

[11]. De Moura L., Bjørner N. Z3: An efficient SMT solver. Tools and Algorithms for the Construction and Analysis of Systems, 2008, pp. 337-340.

[12]. V.A. Padaryan, M.A. Solov'ev, A.I. Kononov. Simulation of Operational Semantics of Machine Instructions. Programming and Computer Software, 37(3), 2011, pp. 161-170. DOI: 10.1134/S0361768811030030.

[13]. Dullien T., Porst S. REIL: A platform-independent intermediate representation of disassembled code for static code analysis. CanSecWest, 2009, 7 pp.

[14]. A.N. Fedotov, V.A. Padaryan, V.V. Kaushan, Sh.F. Kurmangaleev, A.V. Vishnyakov, A.R. Nurmukhametov. Software defect severity estimation in presence of modern defense mechanisms. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 5, 2016, pp. 73-92 (in Russian). DOI: 10.15514/ISPRAS-2016-28(5)-4.

[15]. Caselden D., Bazhanyuk A., Payer M., McCamant S., Song D. HI-CFG: Construction by Binary Analysis and Application to Attack Polymorphism. In Computer Security – ESORICS 2013. Lecture Notes in Computer Science, vol. 8134. Springer pp. 164-181