О некоторых ограничениях полносистемного анализа помеченных данных ⋆

[1]М.А. Климушенкова <maria.klimushenkova@ispras.ru>
[2]М.Г. Бакулин <bakulinm@ispras.ru>
[2,3]В.А. Падарян <vartan@ispras.ru>
[1]П.М. Довгалюк <pavel.dovgaluk@ispras.ru>
[1]Н.И. Фурсова <Natalia.Fursova@ispras.ru>
[1]И.А. Васильев <vasiliev@ispras.ru>
[1]Новгородский государственный университет имени Ярослава Мудрого
173003, Россия, г. Великий Новгород, ул. Большая Санкт-Петербургская, д. 41
[2]Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25
[3]Московский государственный университет имени М.В. Ломоносова,
119991 ГСП-1, Москва, Ленинские горы

Аннотация. Анализ помеченных данных неоднократно пытались применять для исследования безопасности бинарного кода, но все попытки наталкивались на ряд нерешенных вопросов. В данной работе рассматриваются ограничения анализа помеченных данных на уровне бинарного кода, когда он проводится в рамках всей системы. Предлагается подход, способный преодолеть такие ограничения, как высокие накладные расходы на анализ, разрыв в уровне абстракций бинарного и исходного кода и сложности переноса на другие процессорные архитектуры и ОС. Подход позволяет смягчить негативное влияние недостаточной и избыточной помеченности. В подходе используется полносистемный эмулятор, использующий бинарную трансляцию. Возможности анализа помеченных данных обеспечиваются тремя встроенными в эмулятор механизмами: детерминированным воспроизведением, плагинами инстроспекции ВМ и инструментированием промежуточного представления. Приводятся экспериментальные результаты, показывающие лучшую скорость работы в сравнении с аналогичными программными инструментами.

Ключевые слова: анализ помеченных данных; динамический анализ; QEMU.

DOI: 10.15514/ISPRAS-2016-28(6)-1

⋆ Работа поддержана грантом РФФИ № 16-29-09632

Для цитирования: Климушенкова М.А., Бакулин М.Г., Падарян В.А., Довгалюк П.М., Фурсова Н.И., Васильев И.А. О некоторых ограничениях полносистемного анализа помеченных данных. Труды ИСП РАН, том 28, вып. 6, стр. 11-26, 2016 г. DOI: 10.15514/ISPRAS-2016-28(6)-1

1. Введение

С развитием информационных технологий задача противодействия вредоносному ПО принимает все новые формы. Изначально она ограничивалась поиском и удалением вирусов и троянов, заражающих ПК посредством дискет. В настоящее время защита информационных систем обеспечивается сложными программными и аппаратными комплексами. Один из востребованных аспектов противодействия – выявление вредоносного ПО, распространяемого посредством магазинов приложений. Приложения, при подаче в Google Play, проходят обязательную проверку инструментом Bouncer с целью не допустить попадания на устройства пользователей вредоносного ПО. Устройство Bouncer не раскрывалось, известно, что в нем используется динамический анализ – приложение выполняется в контролируемом окружении. Один из известных критериев оценки поведения отслеживает потоки данных в работающей программе: недопустимы ситуации утечки конфиденциальных данных в открытые каналы. Для Android источники и каналы заранее известны в силу четкой спецификации архитектуры системы и многих интерфейсов.

Схожие механизмы выявления вредоносного ПО используются в honeypot-системах. Такие системы, как Argos [1] и Timescope [2] применяют анализ помеченных данных для выявления вредоносного кода, причем последняя система, Timescope, совмещает это с механизмом детерминированного воспроизведения. Практическое применение этого механизма двойственно. Первая решаемая задача заключается в пометке входных данных и отслеживании ситуаций в последующей работе контролируемой программы, когда помеченные данные начнут использоваться в недопустимых операциях, требующих «доверия». Наиболее наглядные примеры – выполнение помеченных данных или пометка счетчика команд. Обратная задача, когда помечаются конфиденциальные данные и отслеживаются ситуации попадания помеченных данных в «недоверенные», открытые каналы связи. Эта задача наиболее актуальна для мобильных приложений, которым доступны различные чувствительные данные: SMS-переписка, голосовые и видео звонки, почта, данные геолокации и многое другое.

Несмотря на то, что за последние десять лет было опубликовано множество работ, посвященных анализу помеченных данных на уровне исполняемого кода, исследования так и не перешли на промышленный уровень. Относительным исключением является Bouncer, но в силу закрытости системы невозможно провести анализ принятых в ней решений. Предлагаемые системы столкнулись с рядом принципиальных проблем, не

имеющих общего решения: учет зависимостей по управлению на уровне исполняемого кода, определение эквивалентности кода и т.д. Следствием этих причин обусловлены проблемы излишней и недостаточной помеченности, выражающиеся в ложных срабатываниях и ложных пропусках соответственно.

В данной статье исследуются причины, препятствующие широкой эксплуатации анализа помеченных данных для выявления вредоносного кода. Авторами статьи была разработана система анализа, основанная на полносистемном эмуляторе Qemu. Реализация системы поддерживает гостевую архитектору x86, позволяет выявлять срабатывание вредоносного кода и обеспечивает интроспекцию виртуальной машины, совмещенную с детерминированным воспроизведением. Главной целью разработки стало нахождение компромисса между несколькими различными критериями: высокой скоростью работы, гибкостью настройки политик анализа помеченных данных и расширяемости системы на другие процессорные архитектуры и гостевые ОС.

Дальнейший материал организован следующим образом. Во втором разделе дается обзор близких работ, рассматриваются проблемы, с которыми сталкивается анализ помеченных данных на уровне бинарного кода. В третьем разделе описываются детали разработанного подхода к выявлению вредоносного кода, в следующем разделе – некоторые аспекты его реализации и результаты экспериментов. В Заключении делаются итоговые выводы и предлагаются дальнейшие улучшения системы.

2. Обзор близких работ

Динамический анализ помеченных данных – крайне популярная техника, которой посвящено множество публикаций. Рассмотрение представительного набора работ является отдельной задачей, в данном обзоре приведены только некоторые, наиболее показательные работы.

Известные инструменты, работающие на уровне бинарного кода, делятся на две категории: полносистемные и поддерживающие только пользовательский код. Анализ в пределах одного процесса, как правило, выполняется гораздо быстрей. Среди таких инструментов стоит упомянуть libdft [3], Dytan [4], Minemu [5]. Libdft и Dytan реализованы как Pin-инструменты и не предлагают достаточно высокого семантического уровня, работая с отдельными командами. Наибольшую гибкость в настройке правил анализа помеченных данных предоставляет Dytan, но ценой значительной деградации производительности: на примере архиватора gzip было зафиксировано 50 кратное замедление. Противоположность ему – система Minemu, показывающая высокую скорость работы, но имеющая множество ограничений, в частности: поддерживается только x86, «цветные» пометки и самомодифицирующийся код не поддерживаются, а XMM-регистры задействованы для хранения служебной информации. Libdtf пытается достичь

компромисса в скорости и гибкости, показанное им замедление не превосходило шестикратного.

Полносистемные инструменты анализа помеченных данных базируются на бинарной трансляции кода и интроспекции виртуальных машин. Исключение – TaintBochs [6], основанный на эмуляторе Bochs и программно интерпретирующий инструкции x86. TaintBosch был создан для отслеживания времени жизни чувствительных данных в памяти приложений. Интерпретация инструкций уступает в скорости бинарной трансляции, а отслеживание помеченных данных вносит дополнительное замедление в 2-10 раз.

Инструмент Timescope [2] предлагает использовать механизм детерминированного воспроизведения для детального анализа атак, происходящих по сети. Применяемый метод состоит в следующем: во время работы полносистемного эмулятора все события, приходящие извне, записываются, что позволяет в дальнейшем точно повторить состояние эмулятора в каждый момент воспроизведения. При воспроизведении используются различные методики для более точного анализа происходящего: сохраняется журнал системных вызовов и их параметров для, например, более детального определения создания вредоносного процесса из-за эксплуатации уязвимости; файлы, созданные или изменённые за время предполагаемой атаки, сохраняются на хозяйской машине; также все данные, полученные по сети, помечаются, пометки отслеживаются, все помеченные инструкции сохраняются, чтобы иметь возможность анализировать шелл-код. Благодаря детерминированному воспроизведению удаётся исполнять исследуемые приложения при незначительном замедлении (1-1,5 раз по сравнению с обычным эмулятором), а тяжеловесный анализ (в том числе анализ помеченных данных) производить без необходимости беспокоиться о вносимом замедлении. Следует заметить, однако, что при таком подходе анализ помеченных данных покажет наличие атаки уже только после её осуществления. Другой недостаток инструмента – механизм воспроизведения не поддерживает графический интерфейс гостевой системы.

DECAF [7] – полносистемный эмулятор с поддержкой битовой и байтовой точности пометок. Эмулятор предоставляет API к базовым средствам анализа: интроспекции, анализу помеченных данных и инструментированию. API используют плагины, нацеленные на решение практических задач: трассировку системных вызовов, выявление логгеров клавиатуры, сбора трассы. Авторами заявлена поддержка двух процессорных архитектур: x86 и ARM и современных ОС: Windows 7 и 8. Поскольку DECAF основан на устаревшей версии Qemu 1.0 благополучно работают только 32-х разрядные версии ОС. На основе DECAF построена DroidScope – специализированная система динамического анализа для платформы Android. В ней обеспечивается интроспекция на уровне виртуальной машины Dalvik и сопоставление

событий разных семантических уровней: уровня машинных команд, ОС Linux, Dalvik VM.

ARGOS [1] – ещё один инструмент, основанный на QEMU (версия 1.1). Эмулятор представляет собой honeypot, он предназначен для обнаружения атак и автоматической генерации сигнатур угроз. В системе автоматически помечаются входящие сетевые пакеты с последующим обнаружением попадания помеченных данных в счетчик команд или использования их в качестве аргументов функции execve. Поддерживаются только архитектуры i386 и x86-64. Основная особенность реализации – ручное инструментирование всех инструкций архитектуры x86, что, с одной стороны, крайне трудоёмко, с другой стороны теоретически должно повысить скорость анализа.

Следующий подход к отслеживанию помеченных данных применяется в другом эмуляторе на основе QEMU – PANDA [8]. Существенной проблемой для поддержки различных архитектур оказываются вспомогательные функции. Если простые команды успешно транслируются TCG, то для сложных, например, системных команд используется программная интерпретация во вспомогательных функциях, написанных на языке Си. Поддержка анализа помеченных данных для каждой новой архитектуры будет требовать модификации вспомогательных функций. Для решения этой проблемы в PANDA весь код гостевой системы транслируется в биткод LLVM [9] и уже на полученном биткоде проводится анализ. Для внутреннего представления TCG такую трансляцию пришлось единожды реализовывать, а код вспомогательных функций различных процессорных архитектур транслируется компилятором clang [10]. Такой подход двухуровневой трансляции позволяет расширяться практически на любую платформу, поддерживаемую Qemu, но приводит к значительному замедлению, оценки которого авторы избегают. Несущественность этой характеристики, по их мнению, обусловлена механизмом детерминированного воспроизведения. Заявлено, что замедление при записи журнала ограничивается 20% и не оказывает влияния на работу гостевой системы. Выполнение тяжеловесного анализа помеченных данных перенесено на этап воспроизведения, когда работа гостя в принципе не чувствительна к замедлению эмулятора.

Идея использовать LLVM как промежуточный язык почерпнута авторами PANDA у системы выборочного символьного выполнения S2E, имеющей близкое архитектурное устройство. S2E реализована на основе двух машин: Qemu для конкретного выполнения и KLEE – для символьного. Поскольку KLEE работает с LLVM, TCG и код вспомогательных функций транслируется. В S2E отслеживание символьных переменных аналогично задаче анализа помеченных данных и страдает от аналогичных проблем, а именно, от избыточной и недостаточной помеченности. Чтоб ограничить распространение пометок (т.е. число символьных переменных) авторы S2E предложили выборочно включать продвижение пометок для определенных

программных модулей. Следует отметить, что S2E демонстрирует высокую производительность, замедляясь по сравнению с Qemu в 3 раза в режиме конкретного выполнения и в 78 раз в режиме символьного выполнения. Несмотря на высокий интерес к S2E со стороны исследовательского сообщества, продуктизации данной технологии за 5 лет не произошло, показанные результаты представляют по большей части академический интерес.

Перечислим основные затруднения, с которыми сталкиваются инструменты анализа, как рассмотренные выше, так и оставшиеся за рамками обзора:

- неприемлемое замедление;
- выявление исследуемой программой факта работы в контролируемом окружении;
- недостаточная помеченность;
- избыточная помеченность;
- семантический разрыв;
- нерасширяемость на другие процессорные архитектуры и гостевые ОС.

В дальнейшем материале делается попытка преодолеть или смягчить указанные затруднения, как на уровне подхода к применению анализа помеченных данных в задачах безопасности ПО, так и при его реализации в рамках полносистемного эмулятора.

3. Полносистемный анализ помеченных данных на уровне бинарного кода

Проблемы, с которыми сталкивается практическое применение анализа помеченных данных на уровне бинарного кода, обусловлены причинами разного уровня. Часть перечисленных проблем решается путем комбинирования различных механизмов, встроенных в эмулятор, и определением порядка их применения. Предлагается подход, нацеленный на достижение максимальной универсальности (Рис. 1) в рамках полносистемного анализа. Выбранная постановка означает, что в качестве среды контролируемого выполнения для исследуемого кода будет использован полносистемный программный эмулятор.

Рис. 1. Порядок применения эмулятора в онлайн и оффлайн анализе.

Fig. 1. Using full-system emulator for online and offline analysis.

Безальтернативной кандидатурой выступает эмулятор с открытым исходным кодом Qemu, поскольку он совмещает в себе целый ряд преимуществ: широкая практика промышленного использования, большое число поддерживаемых процессорных архитектур и периферийных устройств, встроенный, начиная с версии 2.5, механизм детерминированного воспроизведения. Последнее позволяет применять эмулятор как для онлайн, так и для оффлайн анализа, когда собранные журналы недетерминированных событий и снимки состояний используются для воспроизведения работы гостевой системы. Для применения эмулятора в онлайн анализе (фиксация и предотвращение недопустимых ситуаций) необходима эффективность анализа помеченных данных. Существует два подхода к реализации: системы PANDA, весь гостевой код транслируется в LLVM и анализ ведется на его уровне, и системы DECAF, когда инструментируется TCG. Второй подход имеет большую трудоемкость, т.к. для каждой новой архитектуры необходимо разрабатывать инструментирование новых вспомогательных функций, но не дает такого замедления, которое нарушает работу гостевой системы в онлайн режиме.

Другой важный аспект – обеспечение интроспекции произвольной ОС, работающей в гостевой системе для преодоления семантического разрыва. Перехват событий уровня ОС должен быть изолирован в пределах подключаемого модуля, что избавит от необходимости встраивать его в эмулятор на этапе сборки. Такого рода модули должны образовать расширяемую систему плагинов, способных взаимодействовать между собой и вести анализ, как во время реальной работы виртуальной машины, так и при ее воспроизведении. В настоящей версии Qemu должной инфраструктуры плагинов не имеется, ее разработка и реализация необходима для решения всей задачи в целом.

Механизм плагинов обеспечивает взаимодействие независимых модулей на основе API, предоставленный со стороны Qemu. Взаимодействия организованы согласно модели «сигнал-подписчик», схожей с моделью сигналов в Qt. Некоторый источник информации (Qemu или плагин) объявляет, что он предоставляет определенные данные с некоторым назначенным идентификатором, регистрируя новый сигнал в Qemu. Приемник информации (плагин), зная идентификатор интересующего его сигнала, «подписывается» на него, передавая указатель на функцию обработки сигнала. В момент готовности данных источник будет генерировать «сигнал», что является обращением к механизму плагинов. Передаваемые данные оформляются в виде параметров вызова и снабжаются должным идентификатором. В Qemu содержится список зарегистрированных сигналов; для каждого сигнала поддерживается список «подписчиков», т.е. список функций-обработчиков, которые Qemu вызывает с полученными из источника данными. Таким способом устроено оповещение плагинов как о служебных событиях (трансляция инструкции, трансляция базового блока, смена процесса), так и о событиях, возникающих в других плагинах.

Данный подход дает ряд преимуществ. От разработчика плагина скрываются особенности внутренней реализации Qemu, что значительно облегчает написание прикладного функционала. Такая организация взаимосвязи между плагинами позволяет установить многоуровневую систему повышения уровня абстракции событий без жесткой привязки к определенной версии ОС. С данными низкого уровня, получаемыми непосредственно от Qemu работают только т.н. плагины «первого уровня», плагины последующих уровней работают с данными, полученными от плагинов предыдущего уровня, что делает их независимыми от особенностей гостевой системы и процессорной архитектуры. Это существенно упрощает перенос уже существующих плагинов на новые версии Qemu или новые процессорные архитектуры, в идеале потребуется только доработка плагинов первого уровня.

Внутреннее устройство эмулятора, отвечающего требованиям, приведено на Рис. 2. В эмулятор встроено три независимо работающих механизма: детерминированное воспроизведение, анализ помеченных данных и механизм подключаемых плагинов анализа.

Механизм воспроизведения играет сугубо вспомогательную роль. Анализ помеченных данных совмещен с трансляцией гостевого кода в промежуточное представление TCG. Пометки хранятся в теневой памяти и регистрах. В код транслированных блоков добавляются инструментальные инструкции, продвигающие пометки.

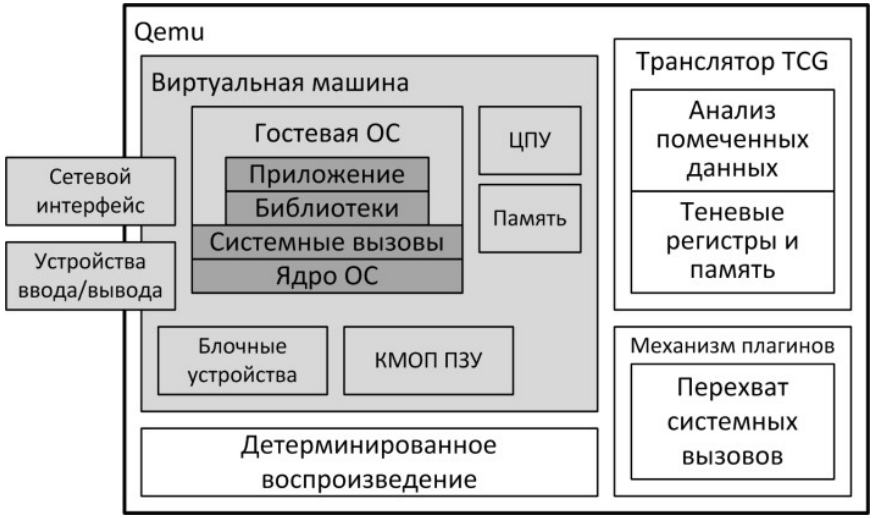

Рис. 2. Архитектура эмулятора с механизмом анализа помеченных данных.

Fig. 2. Architecture of the emulator with dynamic taint analysis support.

Пометки вносятся плагином, перехватывающим системные вызовы, такие как, открытие файла, отображение файла в память, работа с сетевым интерфейсом.

Недопустимые ситуации фиксируются двумя способами: инструментальными командами, встроенными в гостевой код, и плагинами. Первый способ работает для фиксирования низкоуровневых ситуаций, связанных с передачей управления на помеченную память или пометки счетчика команд. Второй способ – для фиксирования событий более высокого уровня, таких как передача помеченных данных в определенные системные вызовы и библиотеки.

Для исчерпывающего контроля потоков данных в системе необходимо отслеживать не только память и регистры процессора, но и потоки, проходящие через периферийные устройства. Блочные устройства, такие как жесткие диски или Flash-накопители хранят в себе данные, которые записываются и считываются программами. Более того, каждое устройство имеет служебные регистры, хранящие некоторое состояние, которое может быть записано, а затем считано. Вопрос отслеживания таких потоков сталкивается с необходимостью корректной интерпретации низкоуровневых обращений к устройствам, а это, в свою очередь, сопоставимо по сложности с разработкой всего комплекта драйверов системы. Тем не менее, должный контроль должен быть обеспечен для блочных устройств и КМОП ПЗУ, хранящей конфигурацию загрузчика.

При работе анализа помеченных данных неизбежно проявляются проблемы недостаточной и избыточной помеченности. Они обусловлены

принципиальными причинами: отсутствием на уровне бинарного кода информации о зависимостях по управлению, отсутствием универсальных правил продвижения пометок, когда помеченные данные влияют на адрес и т.п. Обобщить эти сложности можно в виде задачи определения, вырабатывает ли произвольный фрагмент кода константное значение или нет. В случае константного выражения пометки продвигаться на результат не должны. Известный и единственный способ борьбы с такими ситуациями – эвристическое улучшение правил продвижения пометок.

Стоит упомянуть, что известны работы, посвященные выявлению факта выполнения в эмуляторе Qemu [11]. Применяемая техника основывается на соотнесении неточностей эмуляции определенных машинных команд с эталонным поведением. Противодействие выявлению сводится к исправлению неточностей, открытый исходных код позволяет свободно это делать. Вопрос автоматического поиска неточностей и их диагностики в данной работе не рассматривается.

4. Особенности реализации подхода

Предложенный подход был реализован в виде прототипного инструмента, при разработке которого использовался инструмент DECAF. Одной из практических проблем стало то, что DECAF базируется на весьма старой версии Qemu 1.0. В процессе разработки было проведено перебазирование на актуальную версию Qemu 2.7. Основные усилия были потрачены на портирование вспомогательных функций на новую инфраструктуру эмулятора.

Поддержка битовой гранулярности не переносилась, оставшийся код байтовой гранулярности был специализирован для лучшей производительности. Поддерживается 8 независимых пометок. Пометка регистров сопроцессора с плавающей точкой не поддерживается. Т.к. не отслеживаются зависимости по управлению, регистр флагов тоже не помечается.

Устройство теневой памяти было сохранено, оно нацелено на экономию расхода памяти хостовой машины. Память представляет собой трехуровневое дерево, где корень указывает на промежуточные вершины, а промежуточные вершины содержат листья, отождествляющиеся с участками физической памяти гостевой системы. Листья и промежуточные вершины создаются только в том случае, если в соответствующем участке гостевой памяти содержится пометка.

В результате тестирования было выяснено, что битовые операции (такие как сдвиг и вращение) приводят к избыточной помеченности, поэтому результат таких операций считается помеченным только если исходное значение было помечено независимо от статуса пометки операнда, определяющего величину сдвига.

Для внесения пометок используется плагин, перехватывающий системные вызовы работы с файлами. Поддерживаются следующие функции Windows XP и Windows 7: NtCreateFile, NtOpenFile, NtReadFile, NtCreateSection, NtMapViewOfSection, NtOpenSection, NtWriteFile, NtUnmapViewOfSection, NtClose. Идентификация выбранного процесса делается на основе значения регистра CR3, т.к. в нем содержится база каталога таблиц виртуальной памяти. Это значение остается неизменным на всем протяжении жизни процесса.

5. Результаты экспериментов

Реализованный прототип был протестирован на нескольких примерах. В качестве гостя использовались операционные системы Windows XP SP2 и Windows 7, запуск производился в ОС Ubuntu 14.04LTS на компьютере с процессором Intel Core i5 2500 и 16 гигабайтами оперативной памяти. Примеры были скачаны с базы данных уязвимостей exploit-db.com. Результаты приведены в табл. 1.

Табл. 1. Результаты тестирования на известных уязвимостях.

Table 1. Evaluation of detection rate on known exploits.

Приложение	Поведение	Помеченный переход/код
Mediacoder 0.8.45.5852	Падение	+/+
Easy RM to MP3 Converter 2.7.3.700	Падение	+/-
Coolplayer 2.19.4	Падение	+/-
AllPlayer 5.8	Запуск calc.exe	-/-
MicroP 0.1.1.1600	Падение	-/-
Calavera Uploader! 3.5	Запуск calc.exe	+/+
AlReader 2.5	Запуск calc.exe	+/-
AudioCoder 0.8.22	Падение	+/+
BlazeDVD Pro 7.0	Падение	+/+

В первой колонке указано приложение, в котором проверялась известная уязвимость. Во второй колонке указано поведение системы при попытке эксплуатации уязвимости, например, аварийное завершение приложения при попытке открытия сформированного файла. В третьей колонке указано, выводилось ли предупреждение о подозрительной активности: первый знак (плюс или минус) сообщает о том, был ли совершён переход по помеченному адресу, второй – было ли исполнение помеченного кода. В трех случаях произошёл запуск стандартного приложения «Калькулятор», то есть

однозначно произошла эксплуатация уязвимости. В шести случаях произошло падение тестируемого приложения, что свидетельствует о неправильно обработанных входных данных; скорее всего, эксплуатации при этом не произошло из-за несоответствия версий операционных систем, в которых создавался эксплоит и проводилась проверка, что привело к неправильным адресам переходов в созданных файлах, или ошибок при формировании файла для эксплуатации уязвимости. Например, при попытке эксплуатации приложения Coolplayer был совершен переход на адрес 0x41414141, что соответствует последовательности символов “AAAA”, которые содержатся в файле как заполнитель для переполнения буфера на стеке; это позволяет предположить, что создателями эксплоита было неверно рассчитано расположение адреса возврата.

Также для некоторых приложений создавались «доброкачественные» файлы и проверялось, будет ли выводиться предупреждение при их открытии. В большинстве случаев предупреждение не выводилось, но, например, в приложении BlazeDVD Pro выводились сообщения о подозрительной активности и при открытии обычного файла.

В других инструментах (таких как Panda, Decaf) точность анализа не проверялась, так как в них не имеется механизмов, позволяющих осуществлять пометку файлов или выводить предупреждения, реализованные в разработанном прототипе.

Чтобы примерно оценить замедление, вносимое анализом помеченных данных, было проведено два теста: сжатие папки с различными файлами общим объемом 27 мегабайт при помощи архиватора 7-zip, шифрование папки объёмом 700 мегабайт по алгоритму AES-256, запуск операционной системы.

Табл. 2. Оценка замедления от анализа помеченных данных.

Table 2. Slowdown of different dynamic taint analysis engines.

Среда выполнения	Время, с	
	7-zip	AES-256
Argos	630	1118
Panda	646	730
Decaf	340	675
Разработанный прототип	***315***	***579***
QEMU	88	103
Native	6	12

Для сравнения использовались различные доступные эмуляторы с включенным анализом помеченных данных, эмулятор QEMU без внесенных

модификаций, а также исполнение на реальном оборудовании (обозначено в таблице как Native). Результаты приведены в табл. 2.

По результатам этих двух тестов видно, что предложенная реализация обладает наименьшим замедлением. Эмулятор DECAF, на котором базируется наш прототип, отстаёт, скорее всего, из-за двух своих особенностей реализации: использование битовой точности и старая версия базового эмулятора QEMU (1.0). Остальные эмуляторы показывают значительно большее замедление.

По результатам экспериментов можно сделать следующие выводы: разработанный прототип оказывается способен обнаруживать некоторые из угроз, а также показывает наибольшую скорость работы из доступных инструментов полносистемного анализа помеченных данных. В то же время требуется доработка в плане точности анализа (необнаруженные эксплуатации в двух приложениях) и расширение класса обнаруживаемых угроз.

6. Заключение

В статье исследованы ограничения, с которыми сталкивались попытки применить анализ помеченных данных на уровне бинарного кода. Среди них были указанны те, что представляют наибольшую актуальность при проведении анализа ПО в рамках всей вычислительной системы. Предложен расширяемый подход к организации анализа помеченных данных с целью выявления активности вредоносного ПО, как в виде недопустимого выполнения кода, так и в виде утечки чувствительных данных.

Дальнейшие работы предполагают поддержку таких распространенных архитектур, как ARM и MIPS. Целесообразно расширить поддержку источников помеченных данных на сетевые и USB-устройства, клавиатурный ввод.

Интересен вопрос выявления в исследуемом коде необратимых функций и их учет при продвижении пометок, а также расширение эвристик нахождения константных выражений. Оба улучшения нацелены на снижение доли ложных срабатываний.

На текущий момент не используется статико-динамический подход для отслеживания зависимостей по управлению на уровне бинарного кода. Потенциально он способен улучшить результаты в ситуациях, когда требуется изучить свойства фиксированной программы на большом числе различных входных данных. Потребуется исследовать и разработать эвристики выборочного учета зависимостей по управлению, схожие с теми, что были предложены в работе DTA++ [12].

Список литературы

[1]. Portokalidis G., Slowinska A., Bos H. Argos: An emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation. Proceedings of the 1st ACMSIGOPS/EuroSys European Conference on Computer Systems. New York. 2006. pp. 15-27.

[2]. Srinivasan D., Jiang X. Timescope: Time-Traveling Forensic Analysis of VM-Based High-Interaction Honeypots. In: Security and Privacy in Communication Networks: 7th International ICST Conference, SecureComm 2011, London, UK, September 7-9, 2011, Revised Selected Papers. Springer Berlin Heidelberg, 2012. pp. 209-226.

[3]. Kemerlis V.P., Portokalidis G., Jee K., Keromytis A.D. libdft: practical dynamic data flow tracking for commodity systems. VEE. 2012. pp. 121-132.

[4]. Clause J.A., Li W., Orso A. Dytan: a generic dynamic taint analysis framework. ISSTA. 2007. pp. 196-206.

[5]. Bosman E., Slowinska A., Bos H. Minemu: The World's Fastest Taint Tracker. RAID. 2011. Vol. 6961. pp. 1-20.

[6]. J. Chow, B. Pfaff, T. Garfinkel, K. Christopher, and M. Rosenblum, “Understanding data lifetime via whole system simulation,” in Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13, ser. SSYM’04. Berkeley, CA, USA: USENIX Association [Электронный ресурс]

[7]. Henderson A., Prakash A., Yan L.K., Hu X., Wang X., Zhou R., Yin H. Make it work, make it right, make it fast: building a platform-neutral whole-system dynamic binary analysis platform. The International Symposium on Software Testing and Analysis (ISSTA'14). San Jose. 2014.

[8]. R. Whelan T.L.D.K. Architecture-independent dynamic information flow tracking. Proceedings of the 22Nd International Conference on Compiler Construction. Berlin. 2013.

[9]. Lattner C., Adve V. LLVM: A compilation framework for lifelong program analysis & transformation. Proceedings of the 2004 International Symposium on Code Generation and Optimization. 2004.

[10]. Официальный сайт «Clang» [Электронный ресурс]. http://clang.llvm.org/ (дата обращения: 29.09.2015).

[11]. Raffetseder T., Kruegel C., Kirda E. Detecting system emulators. Proceedings of the 10th International Conference on Information Security. 2007. pp. 1 - 18.

[12]. Kang M.G., McCamant S., Poosankam P., Song D. DTA++: dynamic taint analysis with targeted control-flow propagation. Proceedings of the Network and Distributed System Security Symposium. San Diego. 2011.

On Some Limitations of Information Flow Tracking in Full-system Emulators

[1]M.A. Klimushenkova <maria.klimushenkova@ispras.ru>
[2]M.G. Bakulin <bakulinm@ispras.ru>
[2,3]V.A. Padaryan <vartan@ispras.ru>
[1]P.M. Dovgalyuk <pavel.dovgaluk@ispras.ru>
[1]N.I. Fursova <Natalia.Fursova@ispras.ru>
[1]I.A. Vasiliev <vasiliev@ispras.ru>
[1]Novgorod State University
41 B. St. Petersburgskaya Str., Veliky Novgorod, 173003, Russian Federation
[2]Institute for System Programming of the Russian Academy of Sciences,
25 Alexander Solzhenitsyn Str., Moscow, 109004, Russian Federation
[3]Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russian Federation

Abstract. Tracking and verification of data flows includes set of techniques that can be applied to make applications more secure, to perform software analysis for debugging or reverse engineering, and so on. Taint analysis is one of the techniques used to control data flows. This paper presents an approach for system-wide lightweight platform-aware taint analysis. We implemented proof-of-concept tool based on our approach for i386 platform upon the multi-platform simulator QEMU. Our approach uses instrumentation of QEMU intermediate representation of binary code and processes up to 8 taints simultaneously. Most of the taint analysis code is target-independent and may be used with other target platforms. For some platforms (i386 with Windows XP and Windows 7) we present Virtual Machine Introspection plugins for automatic file tainting. We demonstrate how our taint analysis system may be used to detect exploitation of software vulnerabilities.

Keywords: taint analysis; dynamic analysis; QEMU.

DOI: 10.15514/ISPRAS-2016-28(6)-1

For citation: Klimushenkova M.A., Bakulin M.G., Padaryan V.A., Dovgalyuk P.M., Fursova N.I., Vasiliev I.A. On Some Limitations of Information Flow Tracking in Full-system Emulators. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 6, 2016, pp. 11-26 (in Russian). DOI: 10.15514/ISPRAS-2016-28(6)-1

References

[1]. Portokalidis G., Slowinska A., Bos H. Argos: An emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation. Proceedings of the 1st ACMSIGOPS/EuroSys European Conference on Computer Systems. New York. 2006. pp. 15-27.

[2]. Srinivasan D., Jiang X. Timescope: Time-Traveling Forensic Analysis of VM-Based High-Interaction Honeypots. In: Security and Privacy in Communication Networks: 7th

International ICST Conference, SecureComm 2011, London, UK, September 7-9, 2011, Revised Selected Papers. Springer Berlin Heidelberg, 2012. pp. 209-226.

[3]. Kemerlis V.P., Portokalidis G., Jee K., Keromytis A.D. libdft: practical dynamic data flow tracking for commodity systems. VEE. 2012. pp. 121-132.

[4]. Clause J.A., Li W., Orso A. Dytan: a generic dynamic taint analysis framework. ISSTA. 2007. pp. 196-206.

[5]. Bosman E., Slowinska A., Bos H. Minemu: The World's Fastest Taint Tracker. RAID. 2011. Vol. 6961. pp. 1-20.

[6]. J. Chow, B. Pfaff, T. Garfinkel, K. Christopher, and M. Rosenblum, “Understanding data lifetime via whole system simulation,” in Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13, ser. SSYM’04. Berkeley, CA, USA: USENIX Associati [web-resource]

[7]. Henderson A., Prakash A., Yan L.K., Hu X., Wang X., Zhou R., Yin H. Make it work, make it right, make it fast: building a platform-neutral whole-system dynamic binary analysis platform. The International Symposium on Software Testing and Analysis (ISSTA'14). San Jose. 2014.

[8]. R. Whelan T.L.D.K. Architecture-independent dynamic information flow tracking. Proceedings of the 22Nd International Conference on Compiler Construction. Berlin. 2013.

[9]. Lattner C., Adve V. LLVM: A compilation framework for lifelong program analysis & transformation. Proceedings of the 2004 International Symposium on Code Generation and Optimization. 2004.

[10]. Clang [web-resource]. http://clang.llvm.org/ (last visit: 29.09.2015).

[11]. Raffetseder T., Kruegel C., Kirda E. Detecting system emulators. Proceedings of the 10th International Conference on Information Security. 2007. pp. 1 - 18.

[12]. Kang M.G., McCamant S., Poosankam P., Song D. DTA++: dynamic taint analysis with targeted control-flow propagation. Proceedings of the Network and Distributed System Security Symposium. San Diego. 2011.