ОБОСНОВАНИЕ ВОЗМОЖНОСТИ ПРИМЕНЕНИЯ ВЕРИФИКАЦИИ ПРОГРАММ ДЛЯ ОБНАРУЖЕНИЯ ВРЕДОНОСНОГО КОДА

Козачок А.В.[1], Кочетков Е.В.[2]

Целью проводимого исследования является обоснование возможности применения метода формальной верификации по моделям для обнаружения вредоносного программного обеспечения. Данный подход предлагается в качестве дополнения к существующим механизмам обнаружения вредоносного программного обеспечения. Его использование позволит осуществлять верификацию программ на предмет наличия в них вредоносного кода. В статье показано, что основным средством распространения вредоносного программного обеспечения являются исполняемые файлы. Рассмотрены базовые подходы к построению механизмов обнаружения вредоносного кода при отсутствии априорных сведений об их функциональном предназначении. Дано понятие формальной верификации программ с точки зрения обнаружения вредоносного программного обеспечения. Предложен подход к построению аксиоматических теорий и моделей безопасного исполнения программного кода. Описана функциональная схема верификации исполняемого кода на предмет соответствия моделям безопасного исполнения программного кода.

***Ключевые слова**: вредоносное программное обеспечение, формальная верификация, model checking, стратегии защиты от вредоносного программного обеспечения, механизмы обнаружения.*

Введение

В последние несколько лет особое внимание в исследованиях, посвященных защите информации, уделяется вопросу, касающемуся обеспечения информационной безопасности корпоративных информационно-вычислительных сетей (КИВС). Эти исследования касаются, главным образом, формальной верификации свойств безопасности. Основной целью при этом является разработка формальной математической модели свойств безопасности системы, а также верификация этой модели с помощью математических доказательств.

Целью данного исследования является обоснование возможности применения математического аппарата формальной верификации для защиты инфокоммуникационных систем от проникновения вредоносного программного обеспечения (ВПО). Предлагаемый подход ограничивает использование в КИВС только таких исполняемых файлов, уровень доверия к которым позволяет гарантировать их безопасное функционирование, что особенно актуально в условиях постоянного совершенствования средств и методов самозащиты ВПО.

Под формальной верификацией понимается процесс математического доказательства соответствия определенной модели некоторым заданным свойствам (спецификации), основанный на строгих математических принципах. При этом для описания свойств модели и задания требований к ней применяются специализированные языки, а для проведения верификации – специализирован-ные программные средства, называемые верификаторами.

1. Тенденции развития и распространения ВПО

На современном этапе развития ВПО преобладает тенденция к созданию программ, имеющих канал связи со злоумышленниками и способных успешно скрывать свое присутствие от существующих средств защиты, в рамках проведения комплексной долговременной APT-атаки (Advanced Persistent Thread) [1]. Суть данной атаки заключается в проведении злоумышленниками ряда подготовительных мероприятий по комплексной оценке защищенности информации, циркулирующей в КИВС организации, включающих идентификацию используемых программно-аппаратных средств защиты информации, маршрутов е е движения, порядка обработки и хранения, а также проводимых организационных мероприятий по ее защите от несанкционированного доступа. Такой подход к организации атаки злоумышленниками позволяет обнаружить в системе защиты информации наиболее уязвимые элементы и использовать

1 Козачок Александр Васильевич, кандидат технических наук, Академия ФСО России, Орёл, alex.totrin@gmail.com

2 Кочетков Евгений Викторович, Академия ФСО России, Орёл, mr.Koch91@mail.ru

именно их для проникновения в информационную инфраструктуру организации. При этом однозначная идентификация наименования и версии программно-аппаратных средства защиты информации дает возможность злоумышленникам составить характерный для этих средств перечень уязвимостей, эксплуатация которых позволяет осуществлять проникновение в КИВС на регулярной основе. Однако слабым звеном в комплексе защиты информации становятся также сотрудники организации, одновременно осуществляющие доступ к ресурсам КИВС и ресурсам сети Интернет. Одним из наиболее успешных путей проникновения в компьютеры сотрудников организации является применение методов социальной инженерии, а именно отправка большого количества электронных писем с вложениями, содержащими ВПО или ссылку на него в сети Интернет [2].

Согласно [3], можно сделать вывод, что основным средством распространения ВПО являются исполняемые файлы операционных систем (ОС) семейства Windows. Авторы работы связывают этот факт с подавляющей долей в мире устройств под ее управлением, а также с «разрешающей» политикой безопасности данной ОС. Значительно реже ВПО распространяется в документах офисных приложений, системных компонентах, а также файлах изображений. Данный факт обусловлен более трудоемким процессом встраивания ВПО в файлы этих форматов, заключающимся в необходимости идентификации версии программного обеспечения (ПО), предназначенного для выполнения заданных форматов файлов. Все файлы, содержащие исполняемый код, можно разделить на два класса по виду среды исполнения.

К первому классу относятся исполняемые файлы ОС, для которых перед запуском выделяется память, создается процесс с передачей ему управления. Такие файлы, содержащие вредоносный код, наиболее опасны для ОС, так как процесс, получивший управление, имеет возможность, используя уязвимости ОС и антивирусных средств (АВС), нейтрализовать заложенные в них механизмы обнаружения ВПО и успешно скрывать от пользователя свое присутствие продолжительное время, выполняя цели злоумышленников.

Второй класс файлов, содержащих исполняемый код, составляют файлы документальных форматов, имеющих возможность программирования взаимодействия с ОС, и скрипты командных интерпретаторов. Такие файлы помимо информации включают в себя программы (скрипты), написанные на специфичном для данных приложений языке программирования. Выполнение вложенных программ или сценариев осуществляется прикладным приложением (интерпретатором) в своем адресном пространстве. В этом случае для получения доступа к ресурсам ОС вредоносному коду необходима реализация уязвимости самого приложения изнутри. Этот класс файлов, содержащих исполняемый код, более безопасный по сравнению с исполняемыми файлами ОС.

В настоящее время основным средством защиты информационной инфраструктуры организации от проникновения и функционирования ВПО являются АВС. Исследования показывают, что применяемые ими механизмы обнаружения позволяют достичь вероятностей ошибок первого и второго рода, близких к нулю.

Согласно данным тестирования коммерческих АВС [4], самую высокую вероятность обнаружения равную 0,974, показало АВС «Avast Internet Security». Даже при таком высоком уровне обнаружения количество образцов ВПО, которые остались необнаруженными, составляет 467.

Данный факт порождает необходимость проведения дальнейших исследований в области защиты КИВС от заражения ВПО, создания новых механизмов их обнаружения, применение которых позволило бы компенсировать ряд ограничений существующих механизмов [5] и повысить защищенность конфиденциальной информации в КИВС, так как проникновение даже одного образца ВПО может повлечь за собой получение контроля злоумышленниками над всей информационной инфраструктурой организации в целом.

2. Базовые стратегии защиты от ВПО

Для защиты компьютерных систем от заражения ВПО применяются две базовых стратегии. Обозначим все множество исполняемых файлов ОС – X. Тогда применение любой из стратегий приводит к разделению всего множества X на два подмножества: A и B (рис. 1), критерий отнесения исполняемых файлов будет определяться выбранной стратегией. При этом на рисунке 1 заштрихованная область обозначает множество исполняемых файлов, моделирование которого производится при реализации соответствующей стратегии.

Первая стратегия предполагает разделение исполняемых файлов на основе выявления в них признаков наличия ВПО (рис. 1, a). Изначально все поступающие исполняемые файлы входят во множество A, то есть множество программ, исполнение которых является безопасным с позиции АВС.

В случае обнаружения АВС у исполняемого файла признаков наличия ВПО данный исполняемый файл переносится во множество B, то есть множество программ, исполнение которых вызывает потенциально опасные последствия для пользователя. Наиболее характерными признаками наличия ВПО в исполняемом файле, обнаруживаемыми АВС, являются:

– наличие определенной последовательности байт (сигнатуры), соответствующей одному или нескольким образцам ВПО;

– изменение контрольной суммы исполняемого файла без участия пользователя;

– совершение последовательности действий, похожих на поведение ранее изученных образцов ВПО;

– совершение явно запрещенных операций в процессе функционирования ОС.

Данная стратегия используется в большинстве коммерческих АВС благодаря ее универсальности.

В основе применения второй стратегии к разделению всего множества исполняемых файлов X лежит принцип «запрещающей» политики, то есть все, что не разрешено, то запрещено (рис. 1, *б*). Изначально все поступающие исполняемые файлы входят во множество B, то есть множество программ, безопасность исполнения которых ничем не подтверждена или в них присутствуют признаки наличия ВПО, обнаруженные АВС. Если после проведения соответствующего исследования будет подтверждена безопасность исполнения определенной программы, то ее переносят во множество A. Множество A включает в себя только исполняемые файлы, уровень доверия к которым достаточен для безопасной работы с ними. Данная стратегия не нашла широкого распространения в коммерческих продуктах ввиду сложности реализации и отсутствия соответствующих механизмов анализа.

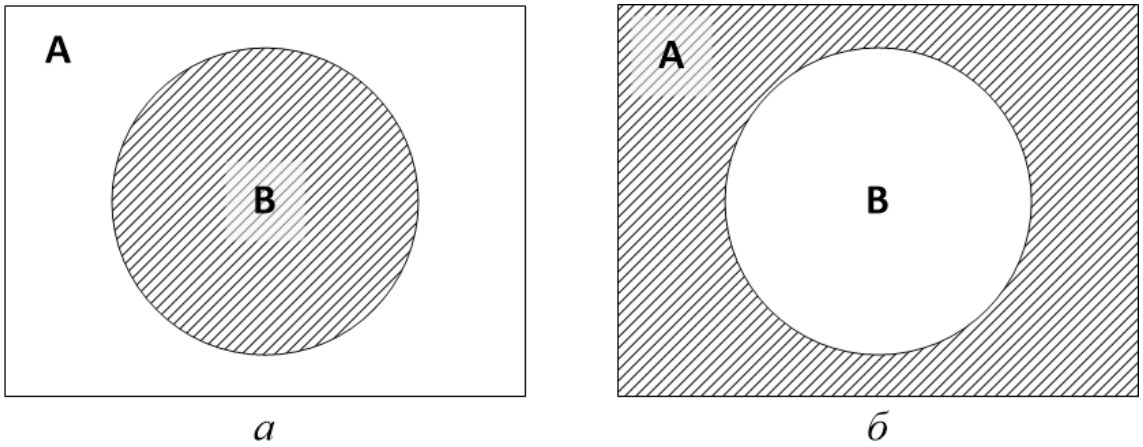
По мнению авторов, наиболее перспективным является совершенствование поведенческого анализа обнаружения ВПО, так как существующие технологии самомодифицирующихся вирусов и полиморфизма значительно затрудняют их обнаружение детерминированными методами и эвристическими механизмами.

Применение поведенческого анализа позволяет использовать обе стратегии защиты от ВПО, решающие соответствующие им взаимосвязанные задачи. Существуют успешные примеры построения моделей вредоносных программ в целях реализации первой стратегии защиты от ВПО [6, 7].

3. Понятие формальной верификации программы

Формальная верификация – это метод проверки того, что аппаратная или программная система соответствует заявленной спецификации (то есть обладает необходимыми свойствами). Он основан на строгих математических принципах [8, 15 с.] и был разработан для верификации систем критически важных инфраструктур. Для проведения верификации без вмешательства человека применяется автоматическая формальная верификация.

Классический подход к формальной верификации последовательных и параллельных алгоритмов описан в [8, 15–20 с.]. Для решения задачи по верификации программы необходимо задать описание каждой отдельной функции, включающее описание допустимых состояний входа, выхода и правила перехода между ними в терминах логики предикатов. После формализации пред- и постус-



а *б*

***Рис. 1**. Стратегии разделения множества исполняемых файлов*

ловий алгоритм функционирования программы кодируется в псевдокоде и по шагам доказывается, что он удовлетворяет спецификации.

В настоящее время основное внимание уделяется формальной верификации реактивных систем [8, 11 с.]. Они характеризуются взаимодействием с окружением, то есть непрерывно получают значения входов и реагируют на них. Примерами таких систем являются сетевые контроллеры, драйверы устройств, сетевые сенсоры, на вход которых поступают извне сетевые пакеты.

Основным недостатком классического подхода к формальной верификации параллельных и реактивных систем является то, что он базируется на описании перехода в функции от входа к выходу. Для параллельных систем корректность верификации при этом зависит не только от конечного результата вычисления, но и от поведения системы во времени. Достаточно затруднительно сформулировать глобальные свойства системы в терминах отношений между входами и выходами. Попытки обобщения классического подхода к формальной верификации реальных параллельных и реактивных систем приводят к большим и сложным доказательствам ввиду того, что эти системы могут взаимодействовать путем использования разделяемых переменных или обмениваться сообщениями. Необходимо также учесть факт взаимодействия с окружающей средой и реакцию на команды пользователя. Работать с доказательствами в подобной форме затруднительно, так как их сложно представить в «удобном» виде, что порождает множество ошибок.

Для верификации корректности реактивных систем во времени, а не только во время перехода от входов к выходам, возможно применение темпоральных логик. Согласно [8, 21 с.], темпоральные логики – логики, расширенные операторами, которые позволяют выражать свойства о вычислениях (в частности, свойства о взаимном порядке между событиями, т. е. последовательностью смены состояний исследуемой программы). Под состоянием программы понимается мгновенное значение определенного набора заданных переменных. Аналогично классическому подходу можно построить правила вывода для темпоральной логики и доказать корректность системы. Данный подход также требует больших временных ресурсов исследователя.

Для выражения взаимного порядка следования между событиями программы необходимо задать соответствующую формулу на основе синтаксиса темпоральной логики. Данный синтаксис можно определить в нотации Бэкуса–Наура ($p \in AP$):

$$\varphi ::= p|\neg\varphi|(\varphi \vee \varphi)|\boldsymbol{EX}\varphi|\boldsymbol{E}[\varphi \boldsymbol{U} \varphi]|\boldsymbol{A}[\varphi \boldsymbol{U} \varphi],$$

где φ – предикат, выражающий свойство системы; $\boldsymbol{X}$ и $\boldsymbol{U}$ – линейные темпоральные операторы, выражающие свойства алгоритма на фиксированном пути, в то время как квантификатор пути $\boldsymbol{E}$ выражает свойство на некотором пути, а универсальный квантификатор пути $\boldsymbol{A}$ – свойства на всех путях; AP – множество атомарных предикатов.

Стоит отметить, что квантификаторы пути $\boldsymbol{E}$ и $\boldsymbol{A}$ могут быть использованы только в комбинации с $\boldsymbol{X}$ или $\boldsymbol{U}$. Булевы операторы $true$, $false$, $\wedge$, $\rightarrow$, $\leftrightarrow$ определяются, как обычно. Примерами простейших высказываний темпоральной логики являются:

– $\boldsymbol{EF}\varphi$ произносится как «φ выполняется потенциально»;

– $\boldsymbol{EG}\varphi$ произносится как «потенциально всегда φ»;

– $\boldsymbol{AG}\varphi$ произносится как «безусловно всегда φ»;

– $\boldsymbol{AX}\varphi$ произносится как «для всех путей в следующий момент φ».

Существует метод формальной верификации, требующий меньшего участия человека в процессе верификации. Он называется «проверка моделей» («Model checking»), базируется на применении темпоральной логики и заключается в использовании исполняемых компьютером алгоритмов для проверки корректности системы. Исходными данными для применения данного метода являются описание модели (возможное поведение) и спецификация (требуемое поведение). Метод «проверка моделей» позволяет существенно быстрее и удобнее получить результат формальной верификации.

Существует два метода проверки по моделям, ключевым отличием которых является способ описания спецификации требуемого поведения:

– логический, описывающий поведение системы путем формулировки множества требуемых свойств в виде спецификации на языке темпоральной логики. Алгоритм функционирования системы при этом моделируется как конечный автомат, в котором состояния содержат сведения о переменных, а переходы указывают, как система может переходить из одного состояния в другое. Таким образом, можно считать систему корректной, если все множество требований выполняется для всего множества начальных состояний;

– поведенческий. В этом случае и моделируемое поведение, и требуемое задаются в одной и той же нотации (конечным автоматом), а для корректности используются отношения эквивалентности или предпорядка. При этом отношение эквивалентности можно интерпретировать как «ведет себя так же, как», а отношение предпорядка «ведет себя как минимум так же, как».

Процесс верификации произвольной системы состоит из трех этапов:

1) моделирование поведения заданной системы (преобразование проекта, аппаратной схемы или программы в формальную модель);

2) построение спецификации (для проверки определенных свойств системы их необходимо сформулировать и записать на специализированном языке);

3) верификация (использование специальных методов и алгоритмов, позволяющих проверить соответствие модели системы заданной для нее спецификации).

Стоит отметить, что исполнение программы в ОС представляет собой реактивную систему, так как данный процесс соответствует ее описанию, то есть под окружением понимается среда ОС, в которой выполняется программа, а под значениями входов – взаимодействие с ней.

Для применения метода «проверка моделей» на практике используются специальные программы – верификаторы. На вход подаются модель системы в формализованном виде и спецификация требований к функционированию этой модели. Результат представляет собой сообщение о согласованности модели системы заданной спецификации в случае успеха, или контрпример, показывающий рассогласованность модели системы и спецификации в случае неудачи [9].

Таким образом, метод автоматической формальной верификации «проверка моделей» является мощным и эффективным средством для доказательства работы исследуемой программы в рамках заданных требований при всех возможных вариантах поведения этой программы. На данный момент существует большое разнообразие программных средств верификации, что объясняется широкой областью применения метода верификации моделей в различных областях науки и техники.

4. Описание подхода к реализации процедуры верификации ПО на предмет наличия в нем вредоносного кода

Основная идея формальной верификации применительно к обнаружению вредоносного кода состоит в том, чтобы построить формальную (математическую) модель исследуемой программы, которая отражает (моделирует) ее возможное поведение в ОС. Требования по корректности безопасного поведения при этом описываются в виде спецификации, которая отражает рамки разрешенного поведения программы. На основе спецификаций и модели исполняемого файла с помощью метода «проверка моделей» можно проверить, насколько возможное поведение согласуется с разрешенным. Поскольку происходит именно математическая верификация, решение о согласованности, то есть соответствии возможного поведения требуемому, является корректным.

Алгоритмы проверки моделей, как правило, базируются на исчерпывающей достижимости всего множества состояний модели [10, 29 с.]. Таким образом, для каждого состояния проверяется, удовлетворяет ли оно заданным в спецификации требованиям и свойствам программы. В самой простой форме алгоритмы проверки моделей позволяют дать ответ на вопрос о достижимости заданных состояний. В таком случае необходимо определить все запрещенные состояния, достижение которых является небезопасным, и выяснить, существует ли такая последовательность смены состояний, которая приводит к одному из них. Если такая последовательность существует, то принимается решение о небезопасности использования исследуемой программы. Стоит отметить, что исчерпывающая достижимость множества состояний гарантированно завершается ввиду конечности модели. При эксплуатации уязвимости программа может перейти к выполнению действий, в том числе не описанных ни в одном состоянии. В таких случаях считается, что программа переходит в запрещенное состояние.

Определение 1. Аксиомы безопасного исполнения программного кода (БИПК) – это описание параметров состояний программы и ОС, а также логики их взаимодействия во времени, позволяющие исключить потенциальную возможность выполнения программой вредоносных действий.

Определение 2. Модель безопасного исполнения программного кода (МБИПК) – это модель поведения программы, исключающая возможность реализации программой вредоносных действий на протяжении всего процесса функционирования ее в ОС. Она построена на основе аксиом БИПК.

На рисунке 2 представлена функциональная схема проверки модели исполнения программы на соответствие МБИПК. Процесс верификации исполня-

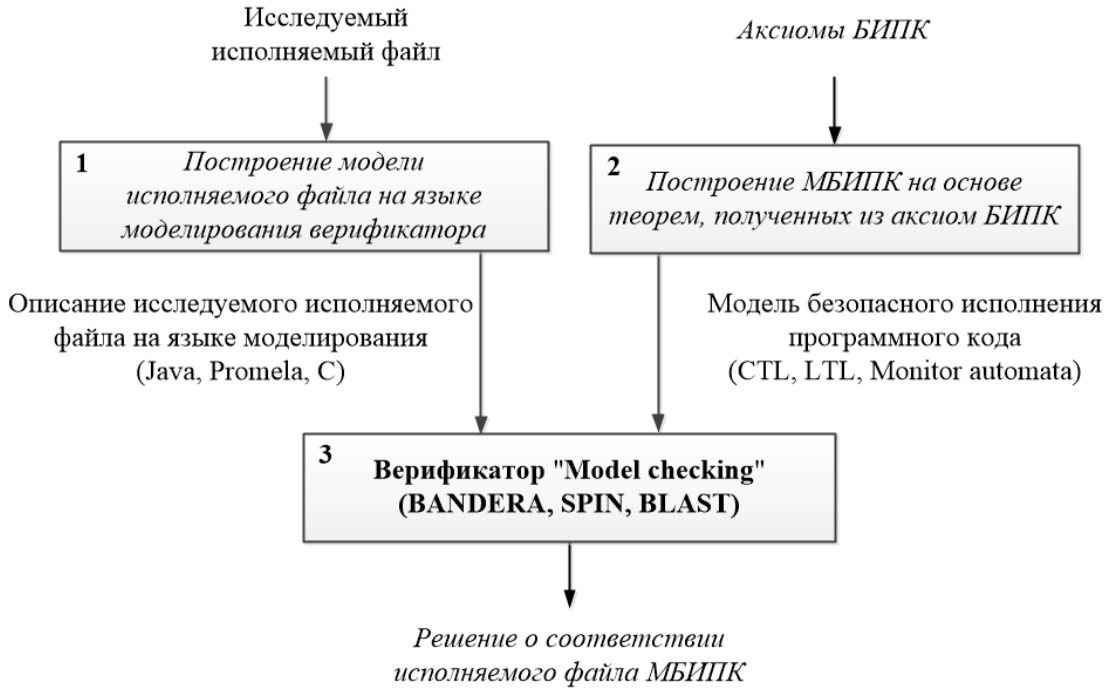


***Рис. 2**. Функциональная схема проверки модели программы на соответствие МБИПК*

емого файла на соответствие МБИПК представляет собой следующую последовательность операций:

1. Исследуемый исполняемый файл поступает на вход блока 1, в котором происходит преобразование исполняемого файла из бинарного представления в модель, описанную на одном из специализированных языков моделирования. Преобразование из бинарного представления в последовательность ассемблерных команд осуществляется с помощью статического дизассемблера «IDA pro» (рис. 3).

При этом важно осуществить полный доступ ко всему коду программы, что обуславливает допущение в виде отсутствия конструкций самомодификации и механизмов защиты от анализа в коде программы. При наличии подобных конструкций и механизмов исполняемый файл считается небезопасным и процесс верификации прекращается.

2. В блоке 2 происходит построение МБИПК, обладающей свойствами, приведенными в определении. Данный этап производится один раз для каждого набора аксиом БИПК. Средством описания МБИПК является формула темпоральной логики или формализованное описание требуемых свойств на специальном языке.

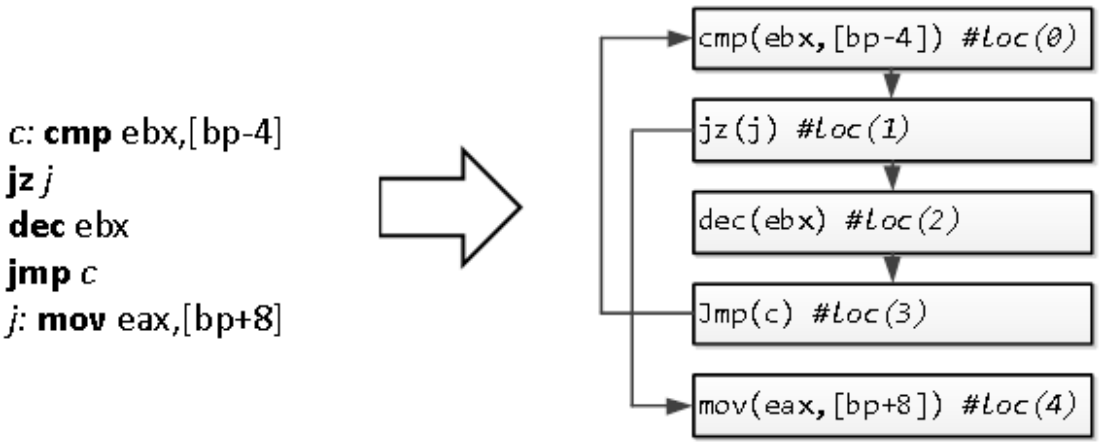


***Рис. 3.** Преобразование последовательности машинных команд в модель Крипке*

Примером простейшего описания требуемого поведения является выражение: «*Где-то в коде программы присутствует инструкция mov, которая помещает константное значение 302 в регистр eax, при этом обязательно в дальнейшем значение этого регистра помещается в стек*». В терминах темпоральной логики оно описывается как

$$EF(mov(eax, 302) \wedge AF(push(eax)),$$

где $mov(x, y)$ – предикат помещения значения x в регистр y; $push(x)$ – предикат помещения значения x в стек.

3. В блоке 3 происходит процесс верификации модели исполняемого файла на соответствие МБИПК верификатором в автоматическом режиме. Выход блока 3 представляет собой бинарное решение: программа является безопасной или потенциально способна нанести вредоносные действия.

Таким образом, постановка задачи по обнаружению вредоносного кода с применением метода верификации моделей сводится к построению модели исполняемого файла и последующей верификации данной модели на соответствие спецификации МБИПК. В случае успешного прохождения верификации данный файл классифицируется как доверенно безопасный, в противном случае считается, что файл потенциально может содержать вредоносный код, и дальнейшая работа с ним запрещается. Данный подход позволяет существенно повысить защищенность КИВС от проникновения ВПО, так как в ее основе лежит вторая (запрещающая) стратегия защиты от ВПО.

Заключение

Методу обнаружения ВПО на основе анализа поведения в настоящее время уделяется большое внимание, так как он позволяет компенсировать ряд ограничений, возникающих при использовании существующих методов, применяемых в коммерческих АВС. В настоящей статье предложен метод формальной верификации «проверка моделей» для доказательства соответствия возможного поведения исполняемого файла в ОС МБИПК. Данную проверку необходимо проводить один раз для каждого исполняемого файла и МБИПК. Использование предложенного метода существенно поднимет уровень доверия к исполняемым файлам, что, в свою очередь, позволит повысить эффективность защиты КИВС от проникновения ВПО в целом. Направлением дальнейших исследований станут практическая реализация и экспериментальная проверка качества предложенного подхода к обнаружению ВПО с помощью метода «проверка моделей».

***Рецензент**: Мацкевич Андрей Георгиевич, кандидат технических наук, сотрудник Академии ФСО России, mag3d@rambler.ru*

Литература:

1. Яремчук С. APT: реальность или паранойя? // Системный администратор. 2012. № 7–8 (116–117). С. 52–56.
2. K. Krombholz, H. Hobel, M. Huber, E. Weippl Advanced social engineering attacks // Journal of Information Security and Applications, Volume 22, June 2015, pp. 113–122.
3. Dong-Her Shih, Hsiu-Sen Chiang, David C. Yen, Shin-Chuan Huang An intelligent embedded system for malicious email filtering // Computer Standards & Interfaces, Volume 35, Issue 5, September 2013, pp. 557–565.
4. Бекбосынова А. А. Тестирование и анализ эффективности и производительности антивирусов // Теория и практика современной науки. 2015. № 5 (5). С. 53–56.
5. A. Moser, C. Kruegel, E. Kirda. Limits of static analysis for malware detection // Secure Systems Lab Technical University Vienna, 2007, pp. 421–430.
6. J. Kinder, S. Katzenbeisser, C. Schallhart, H. Veith. Detecting Malicious Code by Model Checking // Munchen Institut fur Informatik Garching bei Munchen, 2005, pp. 174–180.
7. J. Amador, Jesus R. Artalejo Modeling computer virus with the BSDE approach // Computer Networks, Volume 57, Issue 1, 16 January 2013, pp. 302–316.
8. Вельдер С. Э., Лукин М. А., Шалыто А. А., Яминов Б. Р. Верификация автоматных программ. СПб. – Наука, 2011. – 244 с.
9. A. Emerson. The beginning of model checking: a personal perspective? Department of Computer Sciences. Computer engineering research center the university of Texas at Austin, Austin, USA, 2008, pp. 27–45.
10. Кларк Э. М., Грамберг О., Пелед Д. Верификация моделей программ. Model Checking. М. – МЦНМО, 2002. – 416 с.

УДК 004.056.57

USING PROGRAM VERIFICATION FOR DETECTING MALWARE

Kozachok A.V.[3], Kochetkov E.V.[4]

The purpose of the research is to proof possibility of using formal verification method «Model checking» for detecting malware. This approach is proposed as an additional mechanism to existing mechanisms for detecting malicious software, it will verify programs for the presence of malicious code. The article shows that main method for spreading malware is based on using executable files. Basic approaches for malicious code detecting mechanisms construction without any information about malware purposes are proposed. In the article is described the definition of formal program verification in terms of malware detection. The approach to axiomatic theories and safe code execution models construction is proposed. The article describes functional scheme of executable code verification for compliance with program code execution security models.

***Keywords**: malware, formal verification, model checking, protection strategies against malicious software, detection mechanisms, antivirus protection.*

References:

1. Yaremchuk S. APT: real'nost' ili paranojya? Sistemnyj administrator. 2012. No 7–8 (116–117), pp. 52–56.
2. K. Krombholz, H. Hobel, M. Huber, E. Weippl Advanced social engineering attacks, Journal of Information Security and Applications, Volume 22, June 2015, pp. 113–122.
3. Dong-Her Shih, Hsiu-Sen Chiang, David C. Yen, Shin-Chuan Huang An intelligent embedded system for malicious email filtering, Computer Standards & Interfaces, Volume 35, Issue 5, September 2013, pp. 557–565.
4. Bekbosynova A. A. Testirovanie i analiz ehffektivnosti i proizvoditel'nosti antivirusov, Teoriya i praktika sovremennoj nauki. 2015. No 5 (5), pp. 53–56.
5. A. Moser, C. Kruegel, E. Kirda. Limits of static analysis for malware detection, Secure Systems Lab Technical University Vienna, 2007, pp. 421–430.
6. J. Kinder, S. Katzenbeisser, C. Schallhart, H. Veith. Detecting Malicious Code by Model Checking, Munchen Institut fur Informatik Garching bei Munchen, 2005, pp. 174–180.
7. J. Amador, Jesus R. Artalejo Modeling computer virus with the BSDE approach, Computer Networks, Volume 57, Issue 1, 16 January 2013, pp. 302–316.
8. Vel'der S. E., Lukin M. A., Shalyto A. A., Yaminov B.R. Verifikaciya avtomatnyh programm. SPb. – Nauka, 2011. – 244 p.
9. A. Emerson. The beginning of model checking: a personal perspective? Department of Computer Sciences. Computer engineering research center the university of Texas at Austin, Austin, USA, 2008, pp. 27–45.
10. Klark E.M., Gramberg O., Peled D. Verifikaciya modelej programm. Model Checking. M. – MCNMO, 2002. – 416 p.

3 Aleksandr Kozachok, Ph.D., The Academy of the Federal Guard Service, Orel, alex.totrin@gmail.com
4 Evgeniy Kochetkov, The Academy of the Federal Guard Service, Orel, mr.Koch91@mail.ru