УДК 004.056.57 **DOI** 10.15622/sp.51.4

А.В. Козачок, Е.В. Кочетков

ФОРМАЛЬНАЯ МОДЕЛЬ ФУНКЦИОНИРОВАНИЯ ПРОЦЕССА В ОПЕРАЦИОННОЙ СИСТЕМЕ

Козачок А.В., Кочетков Е.В. **Формальная модель функционирования процесса в операционной системе.**

Аннотация. В статье представлена формальная модель функционирования процесса в операционной системе, построенная на основе применения субъектно-объектного подхода к разделению основных элементов операционной системы. Особенностью представленной модели является высокоуровневая абстракция описания взаимодействия процесса с ресурсами операционной системы, что позволяет применить полученные на ее основе результаты к широкому классу аналогичных систем. Применение данной модели необходимо для совершения перехода от реального процесса к его формальной модели, позволяющей учитывать значимые свойства поведения процесса как на статическом этапе анализа бинарного исполняемого файла, так и на динамическом этапе контроля за его выполнением. Предложена структура системы безопасного исполнения программного кода, являющаяся расширенной композицией таких подходов к обнаружению вредоносного программного обеспечения, как применение метода формальной верификации «Model checking» и использования автомата безопасности для контроля за выполнением исследуемой программы. Применение данной системы позволит использовать в корпоративных информационно-вычислительных сетях только программное обеспечение, уровень доверия к которому подтверждается формальным математическим доказательством и непрерывным контролем за его функционированием.

Ключевые слова: формальная модель процесса, model checking, вредоносное программное обеспечение.

1. Введение. В последние несколько лет в исследованиях, посвященных защите информации, особое внимание 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.

Как показывает практика, большинство угроз информационной безопасности КИВС обусловлено использованием глобальной сети Интернет, в частности, использованием в работе недоверенных программных средств. 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 вредоносное программное обеспечение (ВПО) или ссылку на него в Интернете. Путем использования множества методов злоумышленники побуждают сотрудников организации активировать вредоносную программу, не-

смотря на отсутствие достоверной информации о ее происхождении и функциональном содержании [1].

В данной ситуации разумно предложить обеспечить необходимый строгий экспертный контроль за всеми приходящими извне в КИВС файлами. Однако реализация такого подхода сводится к принятию решения аналитиком о допуске или запрете использования конкретного файла в КИВС. Данное решение может быть принято на основе анализа множества факторов, основными из которых являются:

– отнесение антивирусным средством (АВС) исследуемого файла к классу ВПО;

– совпадение контрольной суммы файла с эталонной из доверенного источника;

– проверка сертификата, которым подписан данный файл;

– доверие каналу передачи и источнику, откуда прибыл файл.

Однако, как можно заметить, даже успешное соответствие всем перечисленным признакам не дает гарантии безопасного использования исследуемой программы в КИВС ввиду отсутствия исследования потенциальных функциональных возможностей программы, а также учитывая, что данная проверка будет производиться лишь один раз на пограничном шлюзе организации.

В настоящее время решение о допуске исполняемых файлов в КИВС в основном принимается на основе результатов их проверки АВС. Исследования показывают, что применяемые в них механизмы обнаружения позволяют достичь вероятностей ошибок первого и второго рода, удовлетворяющих современным требованиям. Но стоит учитывать, что даже при вероятности обнаружения равной 0,974, показанной средством «Avast Internet Security», остаются необнаруженными 468 образцов ВПО, уровень потенциальной опасности которых неизвестен [2]. При этом успешный запуск даже одной такой программы в КИВС может нанести существенный ущерб всей организации в целом.

Отказ от использования глобальной сети в повседневной деятельности организации невозможен, что обусловливает поиск разумного компромисса между удобством работы пользователей и требуемым уровнем информационной безопасности КИВС.

По мнению авторов, таким компромиссом является применение системы безопасного исполнения программного кода. Предлагаемая система является расширенной композицией двух активно развивающихся подходов к обнаружению ВПО, а именно: применение методов формальной верификации модели исследуемой программы [3-7] и использование автомата безопасности для контроля за исполнением исследуемой программы в реальном времени [8-12]. В основе функцио-

нирования системы безопасного исполнения программного кода лежит предположение о том, что если безопасность неизвестной ранее программы при заданных функциональных ограничениях не подтверждена, то ее использование запрещено. Реализация данного утверждения требует разработки формальной модели функционирования процесса в операционной системе, которая бы позволила выражать функциональные ограничения в единой форме независимо от способа проверки их достижения.

Целью настоящего исследования является разработка формальной модели функционирования процесса в ОС с перспективой ее использования для формулирования функциональных ограничений, накладываемых на выполнение программы, в единой форме независимо от способа проверки их выполнения в рамках построения системы безопасного исполнения программного кода.

2. Краткий обзор исследований в области обнаружения ВПО. Формальную постановку задачи обнаружения ВПО можно представить следующим выражением:

$$F : X \to Y, \tag{1}$$

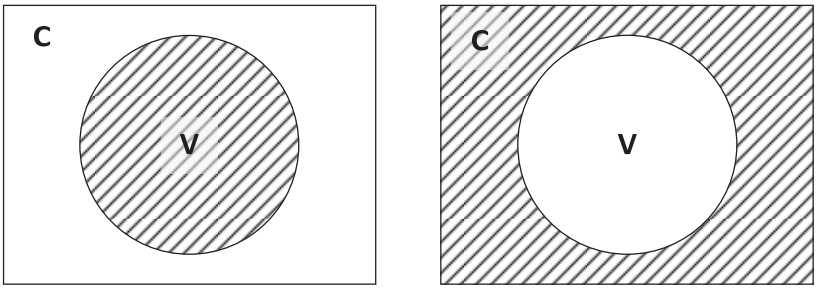
где X — множество всех программ; $Y = \{\{C\},\{V\}\}$ — множество индексов классов программ; C — множество индексов класса программ, безопасных для запуска в ОС; V — множество индексов класса программ, небезопасных для запуска в ОС (ВПО); F — функция, отображающая элементы множества X во множество Y.

Как следует из постановки задачи обнаружения ВПО, индикаторная функция F может быть построена множеством различных способов, однако ее построение всегда происходит в рамках одной из двух стратегий защиты компьютерных систем от проникновения ВПО.

При применении первой стратегии разделение исполняемых файлов происходит на основе выявления в них признаков наличия вредоносного функционала (рисунок 1а). Изначально все исполняемые файлы входят во множество C, то есть множество программ, исполнение которых является безопасным. В случае обнаружения у исполняемого файла признаков наличия ВПО, с точки зрения индикаторной функции F, данный исполняемый файл переносится во множество V, то есть множество программ, исполнение которых вызывает потенциально опасные последствия для пользователя. Данная стратегия используется в большинстве коммерческих АВС ввиду универсальности ее применения.

В основе применения второй стратегии к разделению всего множества исполняемых файлов X лежит принцип «запрещающей» политики — все, что не разрешено, то запрещено (рисунок 1б).

Изначально все исполняемые файлы входят во множество V, то есть множество программ, безопасность запуска которых не подтверждена, а также отсутствует априорная информация о степени риска их выполнения в ОС. Если после проведения соответствующего исследования подтверждается безопасность запуска определенной программы, то она переносится во множество C . Данная стратегия не нашла широкого распространения в коммерческих продуктах ввиду сложности реализации и отсутствия соответствующих механизмов анализа.



а) *б)*

Рис. 1. Стратегии обнаружения ВПО: а) разрешающая; б) запрещающая

В настоящее время выделяются следующие подходы к выявлению признаков вредоносного кода в исполняемых файлах:

– сигнатурный анализ;

– эвристические анализаторы;

– поведенческие анализаторы;

– на основе методов формальной верификации «Model checking».

Поиск на основе сигнатур является основным механизмом, применяемым коммерческими АВС. Сигнатура представляет собой шаблон (регулярное выражение), который однозначно идентифицирует определенную вредоносную программу. Сигнатура позволяет однозначно определить конкретный штамм ВПО, если он присутствует в базе сигнатур. Главными достоинствами этого механизма являются высокая скорость работы и точность обнаружения. Однако он обладает рядом ограничений [13]:

– отсутствует возможность обнаружения новых и модифицированных вредоносных программ до тех пор, пока сигнатура штамма не будет добавлена в базу сигнатур;

– многие современные семейства ВПО используют методы самомодификации в процессе работы [14, 15], что делает поиск на основе сигнатур против них малоэффективным.

Методы эвристического анализа для обнаружения ВПО строятся на основе анализа накопленных знаний об особенностях функционирования различных классов вредоносных программ и применении методов машинного обучения, позволяющих отличать незараженные файлы от ВПО на основе статистических взаимосвязей различных признаков и характеристик [16].

В настоящее время также активно развиваются методы обнаружения ВПО на основе анализа поведения исполняемых файлов. Такой подход является более универсальным, так как позволяет описать поведение вредоносной программы на абстрактном уровне без привязки к бинарному представлению исполняемого файла, которое легко модифицировать [17-19].

Методы обнаружения ВПО на основе анализа поведения непрерывно отслеживают поведение программ в ОС на предмет соответствия некоторой политике безопасности. В течение фазы обучения для детектора администратором формируются правила поведения каждого конкретного приложения. Основное ограничение при применении этого метода — сложность определения множества правил безопасного поведения программ, которые могут быть установлены в защищенной системе. Это, в свою очередь, порождает необходимость вовлечения пользователя в процесс принятия решения. Обзор существующих подходов к обнаружению ВПО на основе анализа поведения представлен в работе [20].

Идея применения метода формальной верификации «Model checking» для решения задачи обнаружения ВПО состоит в том, чтобы построить формальную (математическую) модель вредоносной программы, которая отражает (моделирует) ее возможное поведение в ОС. Поведение штамма ВПО при этом описываются в виде спецификации. На основе этой спецификации и модели исполняемого файла, используя метод «Model checking», можно проверить, согласуется ли возможное поведение с вредоносным.

Впервые использование метода «Model checking» для обнаружения ВПО было предложено в 2005 году в работе [3]. Группа авторов предложила подход, отличительной особенностью которого от существующих на тот момент было обнаружение ВПО на основе анализа его поведения на предмет соответствия заданной модели поведения, характерной для ВПО. Предложенный ими подход заключается в сопоставлении одной спецификации (формулы темпоральной логи-

ки) определенному семейству ВПО, представители которого не имели отличий в поведении, однако существенно отличались по бинарному представлению, вследствие чего не могли быть обнаружены сигнатурным методом до соответствующего изучения каждого образца. Каждый исследуемый бинарный файл преобразовывался в автоматическом режиме в модель, описанную на языке верификатора. На основе этой модели верификатор определял, подходит ли она хотя бы одной из множества заданных спецификаций, соответствующих семействам ВПО. Для сокращения записи спецификаций авторами была введена логика CTPL (Computation Tree Predicate Logic), являющаяся расширением известной логики CTL.

Достоинство предложенного подхода — возможность обнаружения целых семейств вредоносных программ. Недостатком является то, что он не учитывает работу исследуемой программы со стеком, а также необходимость ручного составления спецификации поведения для каждого класса ВПО.

В 2012 году Fu Song и Tayssir Touili предложили использовать метод «Model checking» для обнаружения ВПО с учетом его поведения, включая работу со стеком [4]. Для описания модели поведения вредоносных программ авторами была введена логика SCTPL (Stack CTPL), позволяющая учитывать работу со стеком. Применение данного подхода позволило существенно повысить точность обнаружения вредоносных программ. В результате развития данного подхода авторами была предложена логика SLTPL (Stack Linear TPL) [5].

Суть метода, предлагаемого авторами настоящей статьи, заключается в извлечении информации о поведении исследуемой программы как на этапе хранения, так и на этапе исполнения. Полученная информация сравнивается со спецификацией поведения в соответствии с заданными функциональными требованиями. В случае их соответствия исследуемая программа признается безопасной.

Для реализации предложенного подхода на практике необходимо решение следующих частных задач:

– анализ (извлечение и преобразование информации из исследуемой программы в вид, пригодный для дальнейшей обработки — спецификации);

– синтез (построение модели безопасного исполнения программного кода в соответствии с заданными функциональными требованиями);

– верификация (алгоритм, получающий на вход спецификацию анализируемой программы и принимающий решение о ее соответствии модели безопасного исполнения программного кода);

– контроль исполнения (реализация монитора исполнения программы, позволяющего перехватывать все сигналы взаимодействия процесса с ОС, отслеживать соответствие состояния программы заданной конфигурации и завершить целевую программу в случае необходимости).

Предлагаемые подходы к решению обозначенных выше частных задач рассмотрены в следующем разделе.

3. Функциональная модель системы безопасного исполнения программного кода. Для решения задачи обнаружения ВПО предлагается реализовать индикаторную функцию F (выражение 1) в виде системы безопасного исполнения программного кода, функциональная схема которой представлена на рисунке 2.

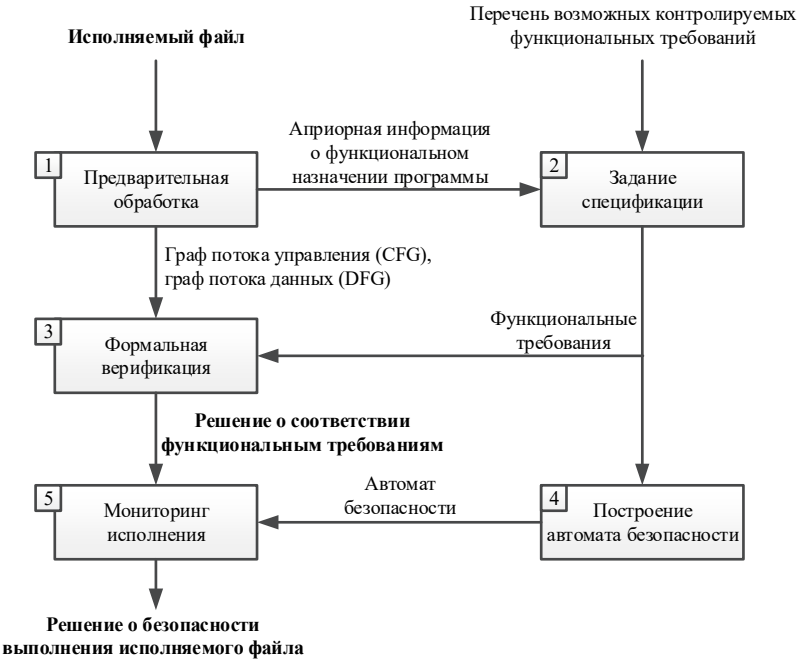


Рис. 2. Функциональная модель системы безопасного исполнения программного кода

На вход блока 1 подается исследуемый исполняемый файл, безопасность использования которого необходимо установить. В данном блоке происходит проверка на наличие в коде программы конструкций самомодификации (включая упаковщики) и механизмов защиты кода программы от анализа. Выполнение этих требований является необходимым условием для дальнейшего исследования исполняемого файла.

В случае их нарушения файл признается небезопасным и его дальнейшая проверка прекращается. Далее происходит преобразование исполняемого файла из бинарного представления в последовательность ассемблерных команд и данных, на их основе строятся графы потока управления (Control Flow Graph) и графы потока данных (Data Flow Graph). Также на этом этапе осуществляется сбор и систематизация априорных сведений о функциональном предназначении программы, которые подаются на вход блока 2.

В блоке «Задание спецификации» на основе априорных сведений о функциональном предназначении программы формируется перечень ограничений на ее функциональные возможности, выполнение которых необходимо для безопасного использования программы. Данный перечень включает в себя функциональные требования, выполнение которых может быть обеспечено в рамках работы системы безопасного исполнения программного кода. Они могут быть разделены на группы в зависимости от категории исследуемой программы. Выходом данного блока являются формализованные функциональные ограничения на работу программы в виде формулы темпоральной логики и конфигурации автомата безопасности.

В блоке 3 осуществляется процесс формальной верификации модели исполняемого файла, построенной на основе графов потоков управления и данных, на соответствие функциональным требованиям статического этапа проверки с помощью метода «Model checking». Требования по корректности безопасного поведения при этом описываются в виде спецификации, которая отражает рамки разрешенного поведения программы. В связи с тем, что 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 [21]. 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 их смены, которая приводит к одному из запрещенных. Если такая последовательность существует, то принимается решение о запрете использования исследуемой программы. Стоит отметить, что исчерпывающая достижимость множества состояний гарантируется ввиду конечности состояний модели [22]. В случае рассогласованности модели программы и спецификации безопасности ис-

полняемый файл признается небезопасным, и дальнейшее исследование прекращается. Выходом данного блока является решение о безопасности использования данной программы по результатам верификации на статическом этапе проверки.

Для контроля над выполнением функциональных ограничений во время работы программы предлагается использование системы, схожей с системой предотвращения вторжений на уровне компьютера (host-based intrusion detection system). Структура предлагаемой системы представлена на рисунке 3.

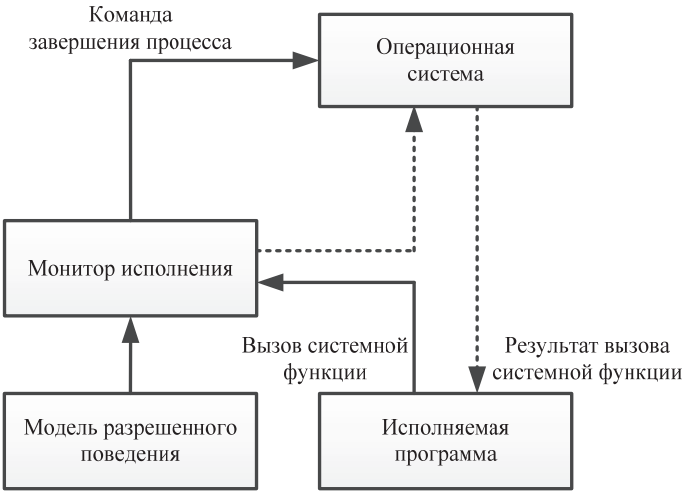


Рис. 3. Структура системы мониторинга исполнения

Монитор исполнения запускается параллельно с исполняемой программой и перехватывает все совершаемые ею системные вызовы. Изначально монитор исполнения загружает модель разрешенного поведения и устанавливает начальное состояние. Каждый вызов системной функции сопоставляется с моделью разрешенного поведения, и происходит переход в новое состояние или, в случае отсутствия такого перехода, подается команда на завершение процесса. В основу монитора исполнения положен автомат безопасности (Security Automata) [8], который строится, как правило, на основе автомата со стеком, определяемого следующим выражением:

$$SA = \left(Q, \Sigma, \Gamma, \delta, q_{0,} z_0, F\right),$$

где Q — множество состояний; Σ — входной алфавит, включая «пустой» входной символ ε ; Γ — алфавит стека; δ — функция перехода;

$q_0 \in Q$ — начальное состояние автомата; $z_0 \in \Gamma$ — начальное состояние стека (стартовый символ); $F \supseteq Q$ — множество конечных состояний.

Таким образом, входными символами автомата *SA* является множество событий функционирования процесса. Конфигурация автомата *SA* определяет множество разрешенных операций в любом состоянии.

В блоке 4 происходит преобразование функциональных требований к программе в конфигурацию конечного автомата со стеком. В блоке 5 производится непрерывный мониторинг за исполнением программы в рамках заданной модели безопасного исполнения. Выходом данного блока является решение о безопасном выполнении исполняемого файла на динамическом этапе проверки.

Для построения формальной модели функционирования процесса в ОС с учетом перечисленных требований необходимо рассмотреть порядок функционирования процесса в ОС. К разрабатываемой модели предъявляются следующие требования [23]:

– целенаправленность (модель должна обеспечивать выделение значимых свойств процесса, взаимодействующего с ОС и ее ресурсами, с учетом момента совершения операций и их критичности, достаточных для построения модели безопасного исполнения программного кода);

– адекватность (модель должна точно отображать свойства исследуемого объекта, а именно функционирование процесса в ОС);

– универсальность (модель должна позволять описывать функционирование любого процесса в большинстве мультипрограммных ОС общего назначения);

– адаптивность (модель должна позволять расширять возможности описания функционирования процесса в ОС с учетом его особенностей);

– интегрируемость (форма описания модели должна позволять преобразования в синтаксис описания спецификации верификатора, конфигурацию автомата безопасности и обратно).

Для построения формальной модели функционирования процесса в ОС с учетом перечисленных требований необходимо рассмотреть порядок взаимодействия процесса с ОС и ее ресурсами.

4. Формальная модель функционирования процесса в ОС. Операционная система является необходимой частью любого вычислительного устройства общего назначения. Она представляет собой интерфейс взаимодействия между пользователем, прикладными программами и аппаратным обеспечением. Большинство прикладных программ не имеют прямого доступа к аппаратному обеспечению вычислительного устройства, а взаимодействуют с ним только через обращения к ОС. Взаимодействие пользователей с программами также

осуществляется только через интерфейсы ОС, передающей воздействие пользователей на устройства ввода и отображающей отклик на устройствах вывода.

Базовыми понятиями при рассмотрении работы ОС являются процесс и ресурс. Согласно [24], процесс — это контейнер для набора ресурсов, используемых при выполнении экземпляра программы. К основным видам ресурсов ОС относятся следующие элементы [25]:

– процессорное время;

– оперативная память;

– внешняя память;

– устройства ввода-вывода.

В основе построения предложенной модели функционирования процесса лежит принцип разделения всех взаимодействующих элементов ОС на субъекты и объекты. К субъектам относятся процессы, совершающие действия по отношению к объектам. Объектами являются ресурсы ОС и процессы, в отношении которых совершаются действия другими субъектами:

– «процесс» (P);

– «оперативная память» (M);

– «внешняя память» (E);

– «периферийные устройства» (D);

– «сетевая подсистема» (N).

Для осуществления доступа к ресурсу процесс выполняет соответствующую функцию ОС, то есть подает заявку на выполнение некоторых действий. ОС на основе внутренних механизмов распределения ресурсов, а также на основе политики безопасности принимает решение о доступе исполняемого кода данного процесса к запрашиваемому ресурсу.

Прикладные программы в процессе работы обладают полным доступом к своему виртуальному адресному пространству для выполнения операций чтения и записи. Для ввода или вывода данных за пределы своего адресного пространства прикладной программе необходимо вызывать соответствующие функции ОС, если она имеет соответствующие привилегии для осуществления таких операций. К таким функциям ОС можно отнести операции чтения, записи, запуск или завершение процесса, выделение дополнительной области памяти, ее освобождение и др.

Определение 1. Под описанием процесса в ОС S_i будем понимать последовательность действий $\{S_i\}_{i\in\mathbb{N}}$, выполняемых субъектом $p_i^{k_p}$. Каждое действие описывается тройкой элементов:

$$\mathrm{s}_i = \left\langle \mathrm{p}_\mathrm{i}^{k_p}; \mathrm{a}_\mathrm{O}; \mathrm{o}_\mathrm{y}^{k_o} \right\rangle, \tag{2}$$

где $\mathrm{p}_\mathrm{i}^{k_p}$ — субъект категории k_p с идентификатором i; $\mathrm{a}_\mathrm{O} \in A_o$ — некоторое действие субъекта по отношению к объекту; $\mathrm{o}_\mathrm{y}^{k_o}$ — некоторый объект категории k_o с идентификатором y.

Операторное представление выражения (2) имеет следующий вид:

$$\mathrm{s}_i = \mathrm{p}_\mathrm{i}^{k_p} \overset{\mathrm{a_O}}{\rightarrow} \mathrm{o}_\mathrm{y}^{k_o}.$$

При выполнении процессом некоторого действия большое значение имеет его категория, так как она определяет возможность совершения им привилегированных операций по отношению к объектам различных категорий.

Важно отметить, что прикладная интерпретация каждой операции зависит от объекта, к которому она применяется и его категории (таблица 1).

Таблица 1. Возможные операции в рамках модели

Обозначение	Наименование операции	Описание
c	***c**reate*	Создание в общем виде нового экземпляра объекта определенной категории.
o	***o**pen*	Получение в общем виде сведений об экземпляре объекта определенной категории.
d	***d**elete*	Удаление экземпляра объекта определенной категории.
r	***r**ead*	Чтение информации из экземпляра объекта определенной категории.
w	***w**rite*	Запись информации в экземпляр объекта определенной категории.

В таблице 2 указаны категории субъектов и объектов представленной модели.

Таблица 2. Категории объектов и субъектов в рамках модели

Обозначение	Описание
Категории субъекта «Процесс»	
P^1	системный процесс
P^2	привилегированный процесс
P^3	пользовательский процесс
Категории объекта «Оперативная память»	
M^1	адресное пространство системного процесса
M^2	адресное пространство другого процесса
M^3	собственное адресное пространство процесса
Категории объекта «Внешняя память»	
E^1	исполняемые файлы
E^2	системные каталоги и конфигурация системы
E^3	конфигурация пользовательского окружения
E^4	файлы и каталоги других пользователей
E^5	собственные файлы и каталоги
Категории объекта «Периферийные устройства»	
D^1	устройства вывода
D^2	устройства ввода
Категории объекта «Сетевая подсистема»	
N^1	сервисы узлов глобальной сети
N^2	сервисы узлов локальной сети
N^3	локальные сетевые сервисы

Схематичное представление модели функционирования процесса в ОС изображено на рисунке 4.

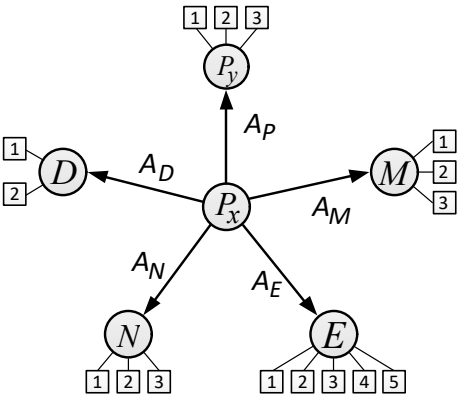


Рис. 4. Модель взаимодействия процесса с ресурсами ОС

В общем виде каждое действие субъекта можно описать набором параметров. Важным отличием при этом от аналогичных моделей является учет параметров осуществляемых действий.

5. Применение модели функционирования процесса в ОС для описания модели безопасного исполнения программного кода. Модель безопасного исполнения программного кода строится на основе аксиом безопасного исполнения программного кода.

Определение 2. Под аксиомой безопасного исполнения программного кода AX будем понимать описание параметров разрешенных действий $\{s_k\}_{k\in\mathbb{Q}}$ для процесса $p_i^{k_p}$ во время его функционирования S_i, позволяющее исключить потенциальную возможность выполнения программой вредоносных действий:

$$AX_j = \bigcup_k s_k, k \in \mathbb{Q}_j,$$

где $\mathbb{Q}_j$ — множество индексов безопасных действий для процесса $p_i^{k_p}$ при заданных функциональных требованиях FR_j.

Пример задания аксиомы «Запрет на создание пользовательским процессом привилегированного или системного процесса» представлен в следующем выражении:

$$AX_j = \neg EF\left[p^3 \xrightarrow{c} p^2\right] \vee \neg EF\left[p^3 \xrightarrow{c} p^1\right]$$

Определение 3. Моделью безопасного исполнения программного кода SM будем называть совокупность аксиом безопасного исполнения программного кода AX_j для процесса $p_i^{k_p}$ при заданных функциональных требованиях FR:

$$SM = \bigcup_{j\in M} AX_j|_{FR=\bigcup_{j\in M} FR_j},$$

где M — множество индексов аксиом безопасного исполнения программного кода, выбранных с учетом функциональных требований.

Главным достоинством описываемой модели функционирования процесса в ОС является возможность описания функционирования процесса на высоком уровне абстракции (без конкретизации выполняемых операций), что позволяет применять полученные на ее основе выводы к широкому классу операционных систем такой же архитектуры.

Представленная модель может быть использована для решения двух взаимосвязанных задач анализа и синтеза при моделировании функционирования процесса в ОС.

Применительно к задаче анализа она позволяет построить модель функционирования процесса в ОС путем наблюдения за его поведением или исследованием бинарного исполняемого файла. В этом случае после моделирования будет получена последовательность событий, порожденных процессом, или граф потока управления. Данные сведения позволяют определить характерные свойства программы и могут быть использованы для оценки потенциальной опасности, которую может принести ее использование.

Применительно к задаче синтеза она позволяет построить модель функционирования процесса в ОС с заранее заданными свойствами, то есть описать такую модель функционирования процесса в ОС, которая соответствует определенному классу безопасности.

6. Заключение. В результате проделанной работы разработана формальная модель функционирования процесса в ОС, позволяющая описать поведение процесса без конкретизации операций или элементарных действий (на высоком уровне абстракции), что позволяет в обобщенной математической форме выразить субъектно-объектное отношения процессов с ресурсами ОС различных категорий.

Рассматриваемая модель может быть использована для описания функциональных ограничений на исполнение программы с использованием операторов темпоральной логики, что позволит выразить их в единой форме в независимости от способа доказательства их выполнения как на статическом, так и на динамическом этапе анализа.

Предложена система безопасного исполнения программного кода, построенная с использованием данной модели, которая позволяет осуществить проверку выполнения функциональных требований. Результатом данной проверки является решение о соответствии исследуемой программы заявленным функциональным ограничениям или принятие решения о запрете ее использования в КИВС.

Направлением дальнейших исследований является построение аксиом безопасного исполнения программного кода с использованием формальной модели функционирования процесса в ОС, практическая реализация системы безопасного исполнения программного кода и экспериментальная проверка качества предложенных решений при решении задачи обнаружения ВПО.

Литература

1. *Krombholz K., Hobel H., Huber M., Weippl E.* Advanced social engineering attacks // Journal of Information Security and Applications. 2015. vol. 22. pp. 113–122.
2. *Бекбосынова А.А.* Тестирование и анализ эффективности и производительности антивирусов // Теория и практика современной науки. 2015. № 5(5). С. 53–56.

3. *Kinder J. et al.* Detecting malicious code by model checking // International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer Berlin Heidelberg. 2005. pp. 174–187.
4. *Song F., Touili T.* Efficient malware detection using model-checking // International Symposium on Formal Methods. Springer Berlin Heidelberg. 2012. pp. 418–433.
5. *Song F., Touili T.* PoMMaDe: pushdown model-checking for malware detection // Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. ACM. 2013. pp. 607–610.
6. *Jasiul B., Szpyrka M., Śliwa J.* Formal Specification of Malware Models in the Form of Colored Petri Nets // Computer Science and its Applications. Springer Berlin Heidelberg. 2015. pp. 475–482.
7. *Козачок А.В., Кочетков Е.В.* Обоснование возможности применения верификации программ для обнаружения вредоносного кода // Вопросы кибербезопасности. 2016. № 3(16). С. 25–32.
8. *Schneider F.B.* Enforceable security policies // ACM Transactions on Information and System Security (TISSEC). 2000. vol. 3. no. 1. pp. 30–50.
9. *Feng H.H. et al.* Formalizing sensitivity in static analysis for intrusion detection // Security and Privacy. Proceedings. 2004 IEEE Symposium on. IEEE. 2004. pp. 194–208.
10. *Basin D. et al.* Enforceable security policies revisited // ACM Transactions on Information and System Security (TISSEC). 2013. vol. 16. no. 1. pp. 3.
11. *Feng H.H. et al.* Anomaly detection using call stack information // Security and Privacy. Proceedings. 2003 Symposium on. IEEE. 2003. pp. 62–75.
12. *Basin D., Klaedtke F., Zălinescu E.* Algorithms for monitoring real-time properties // International Conference on Runtime Verification. Springer Berlin Heidelberg. 2011. pp. 260–275.
13. *Anderson B. et al.* Graph-based malware detection using dynamic analysis // Journal in computer Virology. 2011. vol. 7. no. 4. pp. 247–258.
14. *Devesa J. et al.* Automatic Behaviour-based Analysis and Classification System for Malware Detection // ICEIS. 2010. vol. 2. pp. 395–399.
15. *Козачок А.В., Бочков М.В., Фаткиева Р.Р., Туан Л.М.* Аналитическая модель защиты файлов документальных форматов от несанкционированного доступа // Труды СПИИРАН. 2015. Вып. 6(43). С. 228–252.
16. *Zhang B. et al.* Malicious codes detection based on ensemble learning // International Conference on Autonomic and Trusted Computing. Springer Berlin Heidelberg, 2007. pp. 468–477.
17. *Козачок А.В., Туан Л.М.* Обоснование возможности применения неразличимой обфускации для защиты исполняемых файлов // В сборнике: Перспективные информационные технологии (ПИТ 2015) труды Международной научно-технической конференции. СГАУ. 2015. С. 269–272.
18. *Preda M.D. et al.* A semantics-based approach to malware detection // ACM SIGPLAN Notices. 2007. vol. 42. no. 1. pp. 377–388.
19. *Козачок А.В., Мацкевич А.Г.* Модификация структурного метода распознавания вирусов // Информация и безопасность. 2010. Т. 13. С. 33.
20. *Jacob G., Debar H., Filiol E.* Behavioral detection of malware: from a survey towards an established taxonomy // Journal in computer Virology. 2008. vol. 4. no. 3. pp. 251–266.
21. *Кларк Э., Грамберг О., Пелед Д.* Верификация моделей программ: Model Checking // М.: МЦНМО. 2002.
22. *Вельдер С.Э., Лукин М.А., Шалыто А.А., Яминов Б.Р.* Верификация автоматных программ // СПб. Наука. 2011. 244 с.
23. *Рябов О.А.* Моделирование процессов и систем: учебное пособие // Красноярск. 2008. 122 с.

24. *Руссинович М., Соломон Д.* Внутреннее устройство Microsoft Windows. 6-е изд. // СПб.: Питер. 2013. 800 с.
25. *Гордеев А.В.* Операционные системы: по направлению подгот. «Информатика и вычисл. техника» // Издательский дом «Питер». 2009.
26. *Бабенко Л.К., Буртыка Ф.Б., Макаревич О.Б., Трепачева А.В.* Обобщенная модель системы криптографически защищенных вычислений // Известия ЮФУ. Технические науки. 2015. № 5(166). С. 77–86.

Козачок Александр Васильевич — к-т техн. наук, сотрудник, Академия Федеральной службы охраны Российской Федерации. Область научных интересов: информационная безопасность, защита от несанкционированного доступа, математическая криптография, теоретические проблемы информатики. Число научных публикаций — 80 alex.totrin@gmail.com; Приборостроительная, 35, Орёл, 302034; р.т.: +7(486) 254-99-33.

Кочетков Евгений Викторович — сотрудник, Академия Федеральной службы охраны Российской Федерации. Область научных интересов: информационная безопасность, защита от несанкционированного доступа, защита от вредоносных программ, теоретические проблемы информатики. Число научных публикаций — 10. alex.totrin@gmail.com. 302034, Орёл, ул. Приборостроительная, 35. р.т. +7(486) 254-99-33.

DOI 10.15622/sp.51.4

A.V. KOZACHOK, E.V. KOCHETKOV

FORMAL MODEL OF PROCESS FUNCTIONING IN THE OPERATING SYSTEM

Kozachok A.V., Kochetkov E.V. **Formal Model of Process Functioning in the Operating System.**

Abstract. The article presents a formal model of the functioning of the process in the operating system, created on the basis of a subject-object approach to the separation of the main elements of the operating system. A feature of the presented model is a high-level abstraction of the interaction between the operating system processes and resources, which allows applying the obtained results to a wide range of similar systems. The use of this model is necessary for carrying out the transition from the real world object (process) to a formal model to take into account the significant properties of the behavior of the process both during the static analysis phase of a binary executable file and the dynamic phase of monitoring its implementation. The system of safe execution of code is an extension of the composition of such approaches to the detection of malicious software as the application of the formal verification method «Model checking» and the use of machine safety to monitor the implementation of the studied program. This system allows using in corporate information and computer networks only such software, reliability of which is confirmed by a formal mathematical proof and continuous monitoring of its execution.

Keywords: formal model of process, model checking, malware.

Kozachok Alexander Vasilevich — Ph.D., researcher, The Academy of Federal Security Guard Service of the Russian Federation. Research interests: information security, unauthorized access protection, mathematical cryptography, theoretical problems of computer science. The number of publications — 80. alex.totrin@gmail.com; 35, Priborostroitelnaya Street, Orel, 302034, Russia; office phone: +7(486) 254-99-33.

Kochetkov Evgeniy Victorovich — researcher, The Academy of Federal Security Guard Service of the Russian Federation. Research interests: information security, unauthorized access protection, malware detection, theoretical problems of computer science. The number of publications — 10. alex.totrin@gmail.com; 35, Priborostroitelnaya Street, Orel, 302034, Russia; office phone: +7(486) 254-99-33.

References

1. Krombholz K., Hobel H., Huber M., Weippl E. Advanced social engineering attacks. Journal of Information Security and Applications. 2015. vol. 22. pp. 113–122.
2. Bekbosynova A.A. [Testing and analysis of the effectiveness and efficiency of antivirus]. *Teorija i praktika sovremennoj nauki – The theory and practice of modern science*. 2015. vol. 5(5). pp. 53–56. (In Russ).
3. Kinder J. et al. Detecting malicious code by model checking. International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer Berlin Heidelberg. 2005. pp. 174–187.
4. Song F., Touili T. Efficient malware detection using model-checking. International Symposium on Formal Methods. Springer Berlin Heidelberg. 2012. pp. 418–433.
5. Song F., Touili T. PoMMaDe: pushdown model-checking for malware detection. Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. ACM. 2013. pp. 607–610.

6. Jasiul B., Szpyrka M., Śliwa J. Formal Specification of Malware Models in the Form of Colored Petri Nets. *Computer Science and its Applications*. Springer Berlin Heidelberg. 2015. pp. 475–482.
7. Kozachok A.V., Kochetkov E.V. [Using program verification for detecting malware] *Voprosy kiberbezopasnosti – Cybersecurity issues*. 2016. vol. 3(16). pp. 25–32. (In Russ.).
8. Schneider F.B. Enforceable security policies. *ACM Transactions on Information and System Security (TISSEC)*. 2000. vol. 3 .no. 1. pp. 30–50.
9. Feng H.H. et al. Formalizing sensitivity in static analysis for intrusion detection. Security and Privacy. Proceedings. 2004 IEEE Symposium on. IEEE. 2004. pp. 194–208.
10. Basin D. et al. Enforceable security policies revisited. *ACM Transactions on Information and System Security (TISSEC)*. 2013. vol. 16. no. 1. pp. 3.
11. Feng H.H. et al. Anomaly detection using call stack information. Security and Privacy. Proceedings. 2003 Symposium on IEEE. 2003. pp. 62–75.
12. Basin D., Klaedtke F., Zălinescu E. Algorithms for monitoring real-time properties. International Conference on Runtime Verification. Springer Berlin Heidelberg. 2011. pp. 260–275.
13. Anderson B. et al. Graph-based malware detection using dynamic analysis. *Journal in computer Virology*. 2011. vol. 7. no. 4. pp. 247–258.
14. Devesa J. et al. Automatic Behaviour-based Analysis and Classification System for Malware Detection. ICEIS. 2010. vol. 2. pp. 395–399.
15. Kozachok A.V., Bochkov M.V., Fatkieva R.R., Tuan L.M. [Analytical Model for Protecting Documentary File Formats from Unauthorized Access]. *Trudy SPIIRAN – SPIIRAS Proceedings*. 2015. vol. 6(43). pp. 228–252. (In Russ.).
16. Zhang B. et al. Malicious codes detection based on ensemble learning. International Conference on Autonomic and Trusted Computing. Springer Berlin Heidelberg. 2007. pp. 468–477.
17. Kozachok A.V., Tuan L.M. [Rationale for the possibility of using an indiscernible obfuscation to protect executable files]. *Perspektivnye informacionnye tehnologii – Perspective information technologies (PIT 2015) trudy Mezhdunarodnoj nauchno-tehnicheskoj konferencii. SGAU* [Advanced Information Technologies and Scientific Computing]. 2015. pp. 269–272. (In Russ.).
18. Preda M.D. et al. A semantics-based approach to malware detection. ACM SIGPLAN Notices. 2007. vol. 42. no. 1. pp. 377–388.
19. Kozachok A.V., Mackevich A.G. [Modification of the structural method of recognition of viruses]. *Informacija i bezopasnost' – Information and security*. 2010. vol. 13. pp. 33. (In Russ.).
20. Jacob G., Debar H., Filiol E. Behavioral detection of malware: from a survey towards an established taxonomy. *Journal in computer Virology*. 2008. vol. 4. no. 3. pp. 251–266.
21. Klark Je., Gramberg O., Peled D. *Verifikacija modelej programm: Model Checking* [Verification programs models: Model Checking]. M.: MCNMO. 2002. 416 p. (In Russ.).
22. Vel'der S.Je., Lukin M.A., Shalyto A.A., Jaminov B.R. *Verifikacija avtomatnyh programm* [Verification of the automatic programs]. SPb. Nauka. 2011. 244 p. (In Russ.).
23. Rjabov O.A. *Modelirovanie processov i system: uchebnoe posobie* [Modelling of processes and systems: tutorial]. Krasnojarsk. 2008. 122 p. (In Russ.).
24. Russinovich M., Solomon D. *Vnutrennee ustrojstvo Microsoft Windows. 6-e izd*. [Internal Microsoft Windows device. 6th ed.] SPb.: Piter. 2013. 800 p. (In Russ.).
25. Gordeev A.V. Operacionnye sistemy [Operating Systems]. Izdatel'skij dom «Piter». 2009. 412 p. (In Russ.).
26. Babenko L.K., Burtyka F.B., Makarevich O.B., Trepacheva A.V. [A general model of cryptographically secure computing system]. *Izvestija JuFU. Tehnicheskie nauki – Izvestiya SFedU. Engineering sciences*. 2015. vol. 5(166). pp. 77–86. (In Russ.).