УДК 004.492.3

СЦЕНАРИИ КАК ФОРМА ПРЕДСТАВЛЕНИЯ ЗНАНИЙ О ПОВЕДЕНИИ ВРЕДОНОСНЫХ ПРОГРАММ

В. М. Рувинская, А. В. Молдавская, А. О. Холовчук
Одесский национальный политехнический университет
просп. Шевченко, 1, г. Одесса, 65000, Украина. E-mail: reklama_nedvizh@ukr.net

Рассмотрена проблема поведенческого анализа вредоносных программ и его автоматизации. Оценены и проанализированы различные трактовки понятия сценария. Описано представление знаний о программном поведении с помощью сценариев с учётом использования этого представления в системе автоматизированного распознавания и с учётом потребности в применении машинного обучения. Предложено формировать иерархии сценариев для обеспечения понятного и эффективного представления знаний. Предложено представлять нижний уровень иерархии сценариев в виде конечного автомата. Приведен пример такого представления для поведения вредоносной программы класса «backdoor». Предложено формировать конечный автомат из набора данных о поведении с помощью алгоритма Ахо-Корасик. Рассмотрен метод динамического обновления конечного автомата, приведен пример его использования и описаны особенности практической реализации.

Ключевые слова: вредоносные программы, поведенческий анализ, сценарии, конечные автоматы.

СЦЕНАРІЇ ЯК ФОРМА ПРЕДСТАВЛЕННЯ ЗНАНЬ ПРО ПОВЕДІНКУ ЗЛОВМИСНИХ ПРОГРАМ

В. М. Рувинська, А. В. Молдавська, А. О. Холовчук
Одеський національний політехнічний університет
просп. Шевченка, 1, м. Одеса, 65000, Україна. E-mail: reklama_nedvizh@ukr.net

Розглянута проблема поведінкового аналізу зловмисних програм та його автоматизації. Розглянуто різні трактування поняття сценарію. Описано представлення знань про програмну поведінку за допомогою сценаріїв з урахуванням використання цього представлення в системі автоматизованого розпізнання та з урахуванням потреби у використанні машинного навчання. Запропоновано формувати ієрархії сценаріїв для забезпечення зрозумілого й водночас ефективного представлення знань. Запропоновано репрезентувати найнижчий рівень в ієрархії сценаріїв у вигляді скінченного автомата. Наведено приклад, що представляє поведінку зловмисної програми класу «backdoor». Запропоновано формувати скінчений автомат з набору даних про поведінку за допомогою алгоритму Ахо-Корасік. Розглянуто метод динамічного оновлення скінченного автомата, наведено приклад його використання та описано особливості реалізації.

Ключові слова: шкідливі програми, поведінковий аналіз, сценарії, скінченні автомати.

ПОСТАНОВКА ПРОБЛЕМЫ. На сегодняшний день одной из главных проблем компьютерных систем остаётся уязвимость к вредоносным программам. Традиционно применяемые статические методы анализа на основе сигнатур стали уязвимы к защитным приёмам, применяемым авторами вредоносных программ. В связи с постоянным ростом количества угроз их применение усложняется. Аналитики антивирусных компаний вынуждены всё шире применять динамические, поведенческие методы исследования «подозрительных» файлов, поскольку поведение вредоносных программ в целом остаётся типичным.

Анализ литературных данных. Сегодня программные системы и средства только частично автоматизируют процесс исследования. Наиболее популярными являются среды эмуляции с функцией генерации отчётов, подробно рассмотренные в [1]. Анализ данных о поведении, полученных с помощью таких систем, и принятие решения о вредоносности этого поведения по-прежнему проводится вручную вирусным аналитиком-экспертом. В [2, 3] описаны варианты подхода, в соответствии с которым данные о вредоносном поведении представляют в виде векторов признаков, которые затем используются для кластеризации с помощью соответствующих алгоритмов. Разработки в этом направлении ведутся касаемо выбора наиболее эффективного алгоритма кластеризации. Как отмечают сами исследователи [2], такой подход не позволяет исследовать многовариантное либо полиморфное поведение вредоносных программ. Также он не даёт возможности учитывать причинно-следственную связь между различными действиями.

Исходя из вышесказанного, существуют две проблемы, связанные с обнаружением вредоносных программ по их поведению. Это недостаточная автоматизация всего процесса исследования и отсутствие метода, который позволил бы отображать причинно-следственные связи между действиями вредоносной программы. Необходимо разработать экспертную систему автоматизированного распознавания вредоносного поведения, в которой представление знаний содержало бы связи. Такая система должна также обладать механизмом обучения на поступающих данных.

Целью статьи является разработка метода автоматизированного представления поведения вредоносных программ на основе формализованных знаний.

МАТЕРИАЛ И РЕЗУЛЬТАТЫ ИССЛЕДОВАНИЙ. Для реализации поставленной цели решаются следующие задачи:

– определение понятия сценария как модели представления знаний в целом и для рассматриваемой предметной области (поведение программ).

– определение способа представления сценариев в рамках практической реализации системы автоматизированного распознавания с учётом потребности в применении машинного обучения для

формирования знаний;

– рассмотрение метода(ов) автоматического формирования и обновления структуры для выбранного способа представления.

1. *Понятие сценария. Представление поведения вредоносной программы в виде сценария.* Сценарий – это одна из моделей представления знаний, первоначально являвшаяся разновидностью фреймового представления. Существует как минимум две классических интерпретации сценарного представления, которые применимы в различных областях. Первая предложена автором теории фреймов – М. Минским [4]. Сценарий по М. Минскому представляет собой типовую структуру, описывающую некоторое событие с учётом его контекста. Единицей сценария, соответствующей фреймовому слоту, является в этом случае ответ на характерный вопрос, связанный с ситуацией. Этот вопрос может касаться любого элемента контекста – предмета, эффекта, намерения. В связи с этим, сценарий по М. Минскому предназначен для широкого описания концептов, связанных с событиями. Например, сценарий «Праздник дня рождения» подразумевает вопросы и ответы вида: «Какова одежда? Самая лучшая», «Что взять с собой? Подарок», «Где происходит? В ресторане; дома» и т.п.

Вторая интерпретация приведена в работах Шенка и Абельсона [5]. В ней сценарий определён как стереотипная последовательность действий, определяющих известную ситуацию. В отличие от подхода Минского, здесь сценарии действительно представляют собой последовательно совершаемые типовые действия. Для них также определены действующие лица (*роли*), и введено понятие *цели* – основного смысла производить действия, составляющие сценарий. Сценарий Шенка-Абельсона структурирован с помощью смыслового разделения на сцены, а действия совершаются с некоторыми аргументами, например: «ПЕРЕДВИНУТЬ себя к кассе», «ПЕРЕДАТЬ деньги продавщице».

Оба варианта представления сценариев предполагают использование данной модели представления знаний в искусственном интеллекте при создании систем распознавания, планирования и т.п. Фреймы и сценарии Минского применялись для роботехнических работ, связывающих задачи распознания окружающей среды и адекватного реагирования машины. Кроме того, фреймовые и фреймово-сценарные представления нашли широкое применение в психологии и лингвистике [6], поскольку базируются на естественном человеческом мировосприятии и передаче смысла. Стоит заметить, что общепринятое формальное определение сценария как термина инженерии знаний так и не было разработано.

Одной из особенностей сценария является его иерархичность. В версии Минского она подобна иерархичности любых других фреймов с наследованием свойств. Слот сценария может включать в себя более подробный сценарий, т.е. можно строить иерархичную структуру из множества сценариев. В версии Шенка-Абельсона сценарий подразделяется на сцены, каждая из которых именована как отдельное обобщённое действие и, в свою очередь, подразделяется на более детальные действия, т.е. он представляет собой цельную структуру с внутренним разделением. Для описания программного поведения обычно не требуется знание широкого контекста. Однако, вместе с тем, идея применения обобщённых действий с иерархичным углублением детализации представляется полезной по ряду причин: позволяет рассматривать типовые классы программ (исходя из определения сценария как отображения типовой последовательности действий), позволяет давать пользователю объяснения о результатах анализа в виде общих и доступных понятий. Сценарное представление в классическом понимании является для поставленных задач избыточным, но подходит для представления знаний о поведении программ, в том числе вредоносных.

Сценарием программного поведения будем называть многовариантную последовательность целей, которых требуется достигать программам определённого класса при их работе. *Цель сценария* – это результат, к которому приводит действие или набор действий. Далее под сценариями будем подразумевать именно сценарии программного поведения.

Опишем теперь иерархии сценариев. На наивысшем уровне расположены все основные цели поведения – базовые цели. На более низких уровнях могут быть расположены сценарии, описывающие подцели [7]. Подцели − это промежуточные этапы для достижения цели. Всякая подцель может являться целью для набора подцелей уровнем ниже. На наиболее низком уровне подцели представлены последовательностью некоторых элементарных *действий*, доступных в данном поведении.

Некоторые цели имеют свойство *необязательности*. Это значит, что их достижение необязательно для достижения последующих целей.

Связи между целями могут быть различны:

а) последовательная связь;

б) «и»-связь: для достижения следующей цели необходимо выполнить все предыдущие, связанные «и»-связью;

в) «и/или»-связь: для достижения следующей цели достаточно выполнить любую из предыдущих, связанных «и/или»-связью;

г) «или»-связь: для достижения следующей цели необходимо выполнить любую одну (и только одну) из целей, связанных «или»-связью.

Подцели могут также выполняться однократно либо в виде цикла с некоторым количеством итераций.

Рассмотрим, как будет выглядеть модель поведения вредоносной программы (на примере программы типа «бэкдор»), представленная в форме иерархии сценариев (рис. 1).

Вредоносные программы типа «бэкдор» относятся к категории троянских вредоносных программ. Они проникают в систему различными путями и открывают для злоумышленника доступ к компьютеру по сети, принимая и выполняя его команды и посылая ответ. Здесь цели «бэкдора» представлены

на различных уровнях абстракции: более обобщённом и более детальном. Верхний уровень соответствует жизненному циклу программ-«бэкдоров», приводимых в [8]. Их ещё недостаточно для технического решения задачи обнаружения. Поэтому введён также нижний уровень, выраженный в элементах, соответствующих элементарным системным действиям вредоносной программы. Такими элементами являются события, происходящие в системе вследствие работы программы.

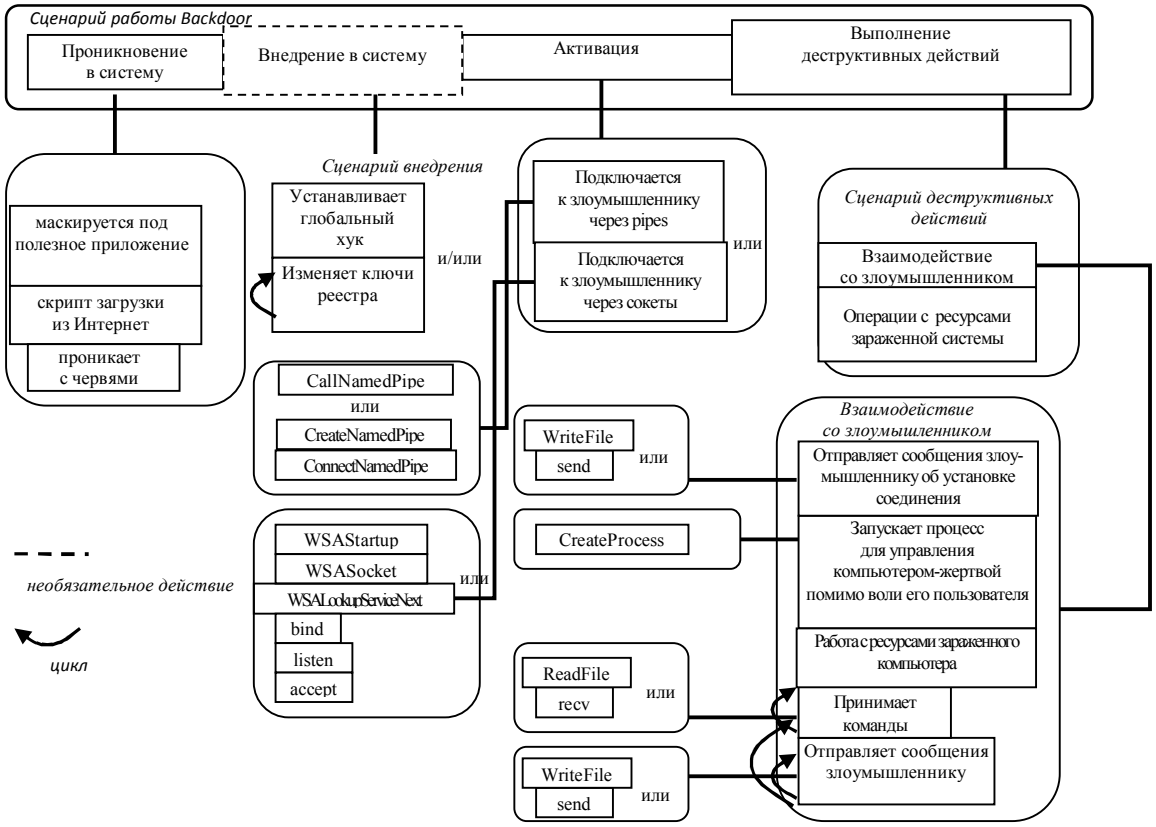


Рисунок 1 – Иерархия сценариев для описания поведения вредоносной программы типа «бэкдор»

Для выполнения одной цели или подцели вредоносной программе часто требуется выполнить несколько элементарных системных действий в определённой последовательности. При этом и сами цели выполняются в определённом порядке, поскольку для достижения каждой последующей цели необходимо завершить выполнение предыдущей и получить результат. На основе этого и можно обнаружить вредоносную активность среди множества происходящих в системе элементарных действий, сличив их с имеющимися сценариями. Однако при этом нужно учитывать, что определённых целей можно достичь несколькими разными действиями или их последовательностями. Все эти варианты должны быть приведены в сценарии.

Примером элементарного действия программы в системе может служить вызов API-функции – это функции ОС, предназначенные для выполнения основных действий в системе. Их вызывают любые прикладные программы или службы ОС. API-функции предназначены для работы с системой на низком уровне, поэтому вредоносные программы могут использовать их для выполнения деструктивных действий. Посредством вызова этих функций они могут проводить операции по записи на диск, самораспространению, порче других программ и т.д.

На рис. 1 нижний уровень представлен для сценария активации и частично для сценария взаимодействия со злоумышленником.

Итак, на самом низком уровне описание действий вредоносной программы будет представлять собой многовариантную последовательность наименований элементарных действий (например, API-функций), которые она вызывает в системе. Наивысший уровень сценария – это последовательность целей с понятными для пользователя названиями. Также установлены соответствия между действиями нижнего уровня и целями и подцелями на более верхних уровнях. Таким образом, цели представляют собой группы из действий или небольших их последовательностей, объединённых по смыслу. Эти действия могут быть различны по своим параметрам и способу выполнения в системе, однако все они позволяют вредоносной программе-бэкдору достичь одной и той же цели.

Формирование верхних уровней сценария потребует ручной обработки информации. Однако формирование нижнего уровня возможно автоматически с помощью средств машинного обучения на статистических данных о поведении уже известных

вредоносов. Далее по сформированному нижнему уровню можно проводить логический вывод, сверяя последовательность событий (например, вызовы API-функций) со стороны проверяемой программы и вариантов последовательностей, выраженные нижним уровнем сценария.

При создании базы сценариев следует накапливать знания не только о поведении вредоносных программ, но и о поведении безопасных, по поведению подобных вредоносным (например, утилиты дистанционного управления частично схожи по функциям с троянскими программами «бэкдор»). Это позволит уменьшить количество ложных срабатываний.

2. *Формы представления нижнего уровня сценариев*. Для практического применения в системе обнаружения вредоносных программ требуется представить нижний уровень сценария в такой форме, которая позволит применять механизмы машинного обучения и логического вывода. Одной из возможных форм представления сценария является байесовская сеть.

Байесовская сеть – направленный ацикличный граф, узлами которого являются случайные переменные. Случайной (вероятностной) переменной называется такая переменная, которая с известной вероятностью принимает одно из значений определённого для неё диапазона [9]. Применимость данного представления и методов машинного обучения, связанных с ним, продемонстрирована в [10]. Преимуществом в применении является возможность вероятностной оценки. Недостатком является ацикличность байесовских сетей, не позволяющая представлять сценарий с циклическими событиями в виде единственной сети.

Другим возможным представлением является конечный автомат. Конечный автомат – модель с конечным числом состояний, входов и выходов. В каждый момент времени автомат находится в одном из своих состояний. Формально конечный автомат задаётся в следующем виде [11]:

$$M=(V,Q,\ q_0,\ F,\ \delta),$$

где V – *входной алфавит* (конечное множество *входных символов*), из которого формируются *входные цепочки*, допускаемые конечным автоматом; Q – *множество состояний;* $q_0 \in Q$ – *начальное состояние;* $F \subset Q$ – множество *терминальных состояний;* δ *– функция переходов.*

Переход осуществляется по допустимому входному символу. В нашем случае входной цепочкой для распознавания будет являться последовательность вызовов API-функций, а алфавитом – набор названий либо кодов всех применимых API-функций. Допустимые автоматом API-функции являются *метками дуг*. *Состояниями автомата* являются состояния процесса работы программы, они включают начальное состояние, промежуточные и конечное (возможно, несколько), определяющее тип программы. В задаче распознания строк смысл состояний автомата иррелевантен. Конечный автомат, в отличие от байесовских сетей, обладает возможностью представлять циклы.

На рис. 2 представлен нижний уровень фрагмента сценария поведения программы-«бэкдора», действие «Взаимодействие со злоумышленником», выраженное в виде диаграммы состояния конечного автомата. Фрагмент с первого по шестое состояния описывает стандартную последовательность инициализации сокета и установку соединения по сети. Фрагмент 1–9 – вариант установки соединения с помощью именованных каналов (*pipe*). Фрагменты 6–11 и 9–11 – разные варианты получения, выполнения и подтверждения команд от злоумышленника.

С помощью команды CreateProcess вызывается утилита командной строки – cmd.exe, но отображается не на компьютере пользователя-жертвы, а на компьютере злоумышленника. При вызове CreateProcess ввод, вывод и сообщения об ошибках перенаправляются на компьютер злоумышленника. Для этого специальным образом настраивается параметры структуры startupinfo.

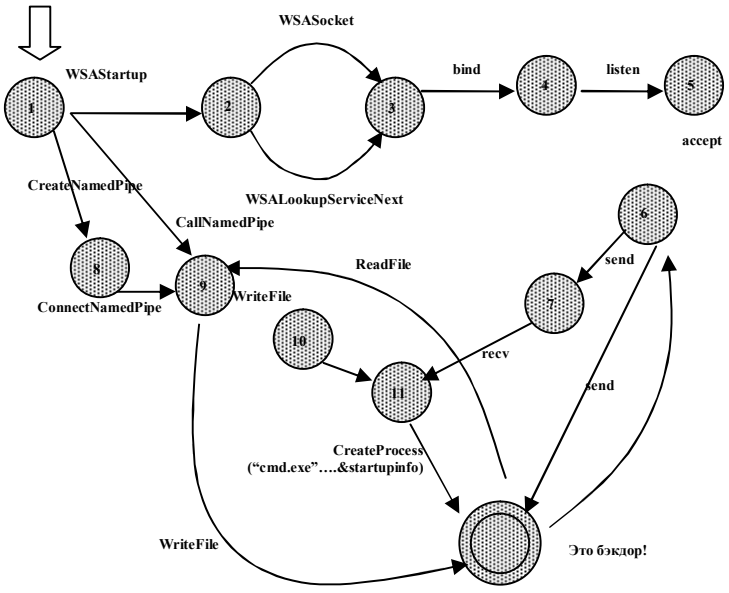


Рисунок 2 – Диаграмма состояний конечного автомата, описывающего фрагмент сценария поведения бэкдора

3. *Формирование сценариев.* Для формирования конечного автомата из набора исходных строк используется модифицированный алгоритм Ахо-Корасик [12]. Данный алгоритм позволяет постро-

ить конечный детерминированный автомат по набору входных строк, которые впоследствии могут быть распознаны этим автоматом, а также могут быть распознаны такие строки, для которых входные строки являются подстроками. Это особенно важно при решении задачи детектирования поведения, т.к. позволяет обнаруживать вредоносное поведение даже в том случае, если до и после действий, описываемых сценарием, программа выполняет и другие действия. Ранее алгоритм Ахо-Корасик уже применялся в области антивирусной защиты для построения автоматов, хранящих сигнатуры [13].

Работа классического алгоритма Ахо-Корасик состоит из трёх фаз:

1. Построение бора на заданном наборе слов. *Бор* – структура данных для хранения набора строк, представляющая собой дерево, в котором каждое ребро помечено некоторым символом [14]. Строки можно распознать, проходя от корня дерева до терминальной вершины. Время работы данной фазы составляет *O(M)*, где *M* – сумма длин всех строк.

2. Построение суффиксных ссылок. Суффиксная ссылка для каждой вершины *p* — это вершина, в которой оканчивается наидлиннейший собственный суффикс строки, соответствующей вершине *p*. Вычисление суффиксных ссылок для *M* вершин займет *O(M)* времени.

3. Бор с суффиксными ссылками и добавленной функцией перехода становится детерминированным конечным автоматом. Узлы бора станут состояниями автомата. На фазе 3 каждая вершина бора рассматривается лишь единожды, вследствие чего затрачивается константное количество времени на каждой из них (считая размер алфавита постоянной величиной).

Исходя из этого, асимптотика по времени построения автомата алгоритмом Ахо-Корасик – *O(M)*.

Алгоритм Ахо-Корасик не является методом машинного обучения: он лишь формирует структуру из всех выданных ему строк, т.е. структурирует данные, а не выводит закономерности и получает знания из данных. Возможны два пути построения метода машинного обучения на основе этого алгоритма:

– использовать предварительную обработку данных для обучения. Подразумевается, что на вход алгоритма будут поданы обобщённые строки, полученные из первичной выборки после отсеивания случайных нехарактерных действий;

– применять вероятностный автомат. Данный подход предполагает построение вероятностного конечного автомата из выборки с последующим учетом либо удалением переходов, охарактеризованных наименьшей вероятностью.

При обоих подходах итоговый конечный автомат будет выражать закономерности между событиями, а следовательно, представлять знания о поведении.

Алгоритм Ахо-Корасик не предусматривает динамического изменения структуры конечного автомата при добавлении новых строк. Однако в области антивирусной защиты это неприемлемо из-за частого появления новых вредоносных программ. При использовании классического алгоритма Ахо-Корасик потребуется построение нового автомата каждый раз, когда появляются новые данные о поведении.

Итак, рассмотренный выше алгоритм решает задачу поиска подстроки из *статического* набора строк. Рассмотрим задачу построения автомата для *динамического* набора строк, то есть в условиях изменения набора строк. Пусть дано множество строк *V*, изначально пустое. Необходимо корректно обрабатывать два типа запросов (всего *N* запросов):

1. Добавить новую строку в V.
2. Принять на вход строку T и найти все вхождения каждого элемента множества V в неё.

Решение задачи «наивным» алгоритмом имеет сложность *O(N^2) + O(N)*, если считать что каждое построение автомата заново производится за *O(N)* времени.

Рассмотрим алгоритм, позволяющий обрабатывать *N* запросов и первого, и второго типа за *O(N*log N)* времени. Общий метод конструирования динамических структур из статических приведен в [15]. Идея алгоритма, основанная на упомянутом методе и примененная для динамического формирования автомата, впервые встречается в [16].

Основная идея алгоритма заключается в следующем. Пусть в каждый момент времени существует не один автомат, а несколько (порядка *O(log N)*, где *N* – количество слов в множестве *V*). Причем объединение множеств слов, распознаваемых каждым из них, будет равняться множеству *V*, а пересечение множеств слов, распознаваемых любой парой автоматов, будет пусто (каждый элемент множества *V* будет содержаться только в одном автомате).

Допустим, что уже построено *O(log N)* автоматов, удовлетворяющих предъявленным выше требованиям. Тогда ответ на запрос второго типа будет производиться следующим образом: пропустим считанную строку *T* через каждый из имеющихся автоматов, объединим все найденные строки и вернем в качестве результата.

Оценка временной асимптотики алгоритма: строка длины *|T|* проходит через *O(log N)* автоматов, затратив на каждом *O(|T|)* времени. Таким образом, получаем сложность алгоритма – *O(|T| * log N)*.

Как поддерживать множество автоматов? Храним *O(log N)* автоматов, притом количество слов в каждом является степенью двойки, и все степени двойки различны. При каждом добавлении нового слова возникает новый автомат, состоящий только из этого слова. Всякий раз, когда получены два автомата, состоящие из равного количества слов, производится объединение их в один, т.е. построение нового автомата, объединяющего их, и разрушение исходных двух. Повторяем это до тех пор, пока не восстановим первоначальное условие – отсутствие двух автоматов с одинаковым количеством слов.

Таким образом, на каждом шаге по-прежнему происходит разрушение некоторых автоматов и построение новых, но из значительно меньшего количества слов. Именно за счет этого достигается асим-

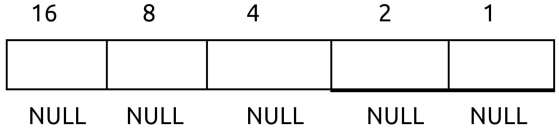
птотика по времени *O(N * log N)*.

Однако такая асимптотика достижима только при правильной реализации идеи. Реализуем следующим образом: будем хранить связный список *L* указателей на автоматы (пускай каждый автомат представляет из себя объект некоторого класса) такой, что в *i*-м элементе (в 0-индексации) списка будет храниться указатель на автомат, состоящий из 2^i слов.

Рассмотрим пример работы алгоритма.

Допустим, последовательно поступают запросы на добавление пяти слов в следующем порядке: *his, he , hers, she,rolled.*

Изначально множество слов пусто, следовательно, в каждом элементе списка хранится NULL-указатель:



Далее происходит добавление слова *“his”*. Автомат занимает ячейку 1, так как она свободна:

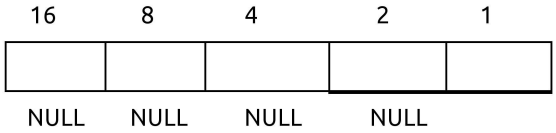
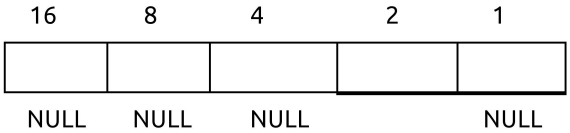


Диаграмма состояний автомата представлена на рис. 3,а.

Затем добавляется слово *“he”*. Ячейка для автомата с одним словом уже занята, поэтому разрушим автомат со словом *“his”* и запишем автомат, распознающий два слова, в ячейку 2.

Диаграмма состояний автомата представлена на рис. 3,б.



Далее добавляется слово *“hers”*. Ячейка для автомата с одним словом свободна, поэтому запишем указатель на этот автомат в ячейку 1. Теперь у нас два автомата:

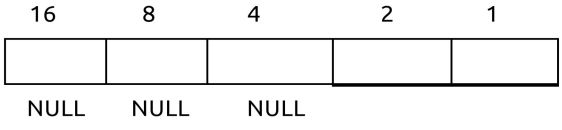


Диаграмма состояний автомата представлена на рис. 3,в.

Далее добавляется слово *“she”*. Ячейки и с одним, и с двумя словами заняты, поэтому разрушим первые два автомата и запишем в ячейку 3 автомат, состоящий уже из 4-х слов:

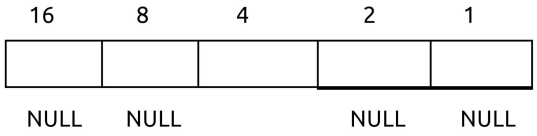


Диаграмма состояний автомата представлена на рис. 3,г.

Далее добавляется последнее слово – *“rolled”*. Так как ячейка для автомата из одного слова свободна, займем её:

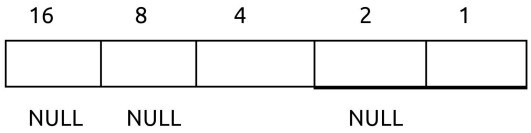


Диаграмма состояний автомата представлена на рис. 3,д.

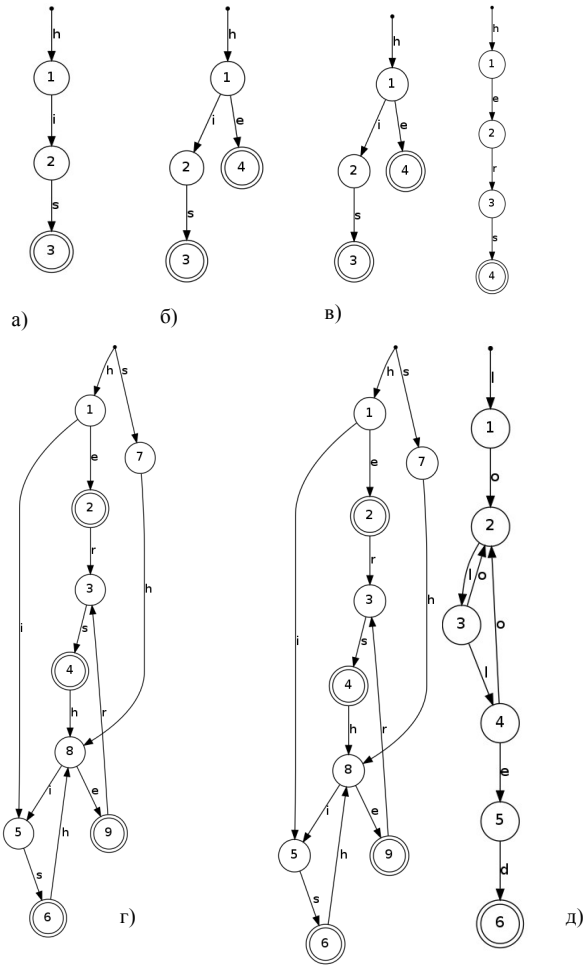


Рисунок 3 – Поэтапное изменение конечного автомата в ходе алгоритма

Добавление пяти слов завершено. Таким образом, присутствует возможность динамически отвечать на запросы добавления слов и распознавания слов в строках.

Существует альтернативный алгоритм добавления строк – алгоритм инкрементальной минимиза-

ции [17], однако полученный в результате автомат позволяет распознавать символы только из начала строки.

Запрос первого типа в нашем случае – это добавление при обучении автомата последовательности событий новой вредоносной программы, а запрос второго типа – это распознавание в анализируемых программах типичного вредоносного поведения. Сам конечный автомат – это представление нижнего уровня иерархии сценариев.

ВЫВОДЫ. Поведенческий анализ вредоносных программ открывает широкие возможности для их обнаружения. Предложено модели поведения вредоносных программ представлять в виде иерархий сценариев. Рассмотрены разные подходы к понятию сценария как модели представления знаний, описано понятие сценария для изучаемой предметной области (программное поведение). Предложено нижний уровень иерархии сценариев представлять в виде конечных автоматов, приведен соответствующий пример. Рассмотрена возможность применения алгоритма Ахо-Корасик для формирования конечных автоматов на основе выборки данных о поведении. Рассмотрен метод динамического обновления конечного автомата, пример его использования и особенности реализации. Таким образом, можно изменять структуру автомата по мере поступления новой информации, не перестраивая его целиком. Методы построения конечного автомата не являются методами машинного обучения, а потому в дальнейшем следует разработать метод машинного обучения для конечных автоматов. В статье упомянуты два возможных пути построения такого метода.

ЛИТЕРАТУРА

1. Survey on Automated Dynamic Malware-Analysis Techniques and Tools / Egele M., Scholte T., Kirda E., Kruegel C. // Journal in ACM Computing Surveys. – 2012. – № 44. – PP. 1–49.

2. Learning and Classification of Malware Behavior / Konrad Rieck, Thorsten Holz, Carsten Willems et all. // Lecture Notes in Computer Science. – 2008. – № 5137. – PP.108–125.

3. Vinod P., Harshit J., Yashwant K. Golecha. MEDUSA: MEtamorphic malware Dynamic analysis Using Signature from API // Proceedings 3rd international conference on Security of information and networks. – New York: ACM New York, 2010.

4. Минский М. Фреймы для представления знаний. – М.: Энергия, 1979. – 151 с.

5. Roger C. Schank and Robert P. Abelson. Scripts, Plans and Goals // IJCAI'75 Proceedings of the 4th international joint conference on Artificial intelligence. – San Francisco, CA, USA, 1975. – Vol. 1. – PP. 151–157.

6. Луев Г.В. Фреймы-сценарии для описания смысловой структуры слова // Теория верификации лингвистических отношений: межвуз. сбор. науч. трудов. – М.: МОПИ им. Н.К. Крупской, 1988. – С. 93–103.

7. Эвристические методы детектирования вредоносных программ на основе сценариев / В.М. Рувинская, Е.Л. Беркович, А.А. Лотоцкий // Искусств. интеллект . – 2008. – № 3. – С. 197–207.

8. Вирусы и средства борьбы с ними. [Электронный ресурс] // Режим доступа: http://www.csid.omsu.omskreg.ru/docs/docs/viruses.pdf

9. Тулупьев А.Л., Николенко С.И., Сироткин А.В. Байесовские сети: логико-вероятностный подход. – СПб.: Наука, 2006. – 608 с.

10. Применение методов машинного обучения для формирования сценариев поведения вредоносных программ / А.В. Молдавская, В.М. Рувинская // Информатика и математические методы в моделировании. – 2014. – № 4(2). – С. 149–157.

11. Введение в теорию автоматов, языков и вычислений / Д.Э. Хопкрофт, Раджив Мотвани, Джеффри Ульман. – 2-е изд. – М.: Вильямс, 2002. – 528 с.

12. Efficient string matching: An aid to bibliographic search / A.V. Aho, M.J. Corasick // Communications of the ACM. – 1975. – Vol. 18(6). – PP. 333–340.

13. Pungila C. A Bray-Curtis Weighted Automaton for Detecting Malicious Code Through System-Call Analysis // SYNASC, 11th International Symposium on Symbolic and Numeric Algorithms for Scientic Computing. – 2009. – Timisoara. – PP. 392–400.

14. Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. Алгоритмы: построение и анализ – 2-е изд. – М.: Вильямс, 2007.

15. Bentley, J.L., Saxe, J.B. Decomposable searching problems I. Static-to-dynamic transformation // Journal of Algorithms. – 1980. – № 1(4). – PP. 301–358.

16. Xui Hao Ran. (2013), "About a few non-classical solutions for data structures", National Olympiad in Informatics 2013 – Proceedings of the Chinese team participants, Chengdu, Sichuan, China, pp. 71–85.

17. Daciuk J., Mihov S., Watson B.W., Watson R.E. Incremental construction of minimal acyclic finite-state automata // Computational Linguistics. – 2000. – Vol. 26, № 1. – PP. 3–16.

SCRIPTS AS A FORM OF REPRESENTATION OF KNOWLEDGE OF MALWARE BEHAVIOR

V. Ruvinska, A. Moldavskaya, A. Kholovchuk

Odessa National Polytechnic University

prosp. Shevchenko, 1, Odessa, 65000, Ukraine. E-mail: reklama_nedvizh@ukr.net

This article is concerned with the problem of malware behavior analysis and its automatization by using scripts for knowledge representation. Different views on the concept of the script are presented. The way to represent knowledge of software behavior via scripts is described, while taking into account the necessity of using machine learning and those scripts further involving into creation of a knowledge base implemented into automated malware detection system. It is proposed to design scenarios hierarchy so to provide both effective and understandable making knowledge representation. An example of the malware behavior script with its finite automaton is presented. The Aho-Corasic algorithm is proposed to be applied when behavior-representing finite automaton designing. An algorithm of dynamic finite automaton structure update is described with its detailed example use is given.

Key words: malicious software, malware behavioral analysis, scripts, finite automaton.

REFERENCES

1. Egele, M., Scholte, T., Kirda, E., and Kruegel, C. (2012), "Survey on Automated Dynamic Malware-Analysis Techniques and Tools", *Journal in ACM Computing Surveys*, no. 44, pp.1–49.

2. Konrad Rieck, Thorsten Holz, Carsten Willems, Patrick Düssel, and Pavel Laskov (2008), "Learning and Classification of Malware Behavior", *DIMVA '08 Proceedings of the 5th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Berlin, pp.108–125.

3. Vinod P., Harshit Jain, and Yashwant K. Golecha (2010), "MEDUSA: Metamorphic malware Dynamic analysis Using Signature from API", *Proceedings of the 3rd International Conference on Security of Information and Networks*, ACM New York, New York, USA.

4. Minskij, M. (1979), *Frejmy dlja predstavlenija znanij* [A Framework for Representing Knowledge], Energija, Moscow, Russia.

5. Roger C. Schank and Robert P. Abelson (1975), "Scripts, Plans and Goals", *IJCAI'75 Proceedings of the 4th international joint conference on Artificial intelligence*, San Francisco, CA, USA, vol. 1, pp. 151–157.

6. Luev, G.V. (1988), "Using script frames for describing the semantic structure of the word", *Teorija verifikacii lingvisticheskih otnoshenij,* Krupskaya Moscow Regional Pedagogical Institute (Moscow State Regional University), Moscow, Russia, pp. 93–103.

7. Ruvinskaja, V.M., Berkovich, E.L., and Lotockij, A.A. (2008), "Heuristic script-based methods of malware detection", *Iskusstv. Intellect*, no. 3, pp. 197–207.

8. "Malwares and the methods of fighting them", available at: http://www.csid.omsu.omskreg.ru/docs/docs/viruses.pdf (accessed July 15, 2014)

9. Tulup'ev, A.L., Nikolenko, S.I., and Sirotkin, A.V. (2006), *Bajesovskie seti: logiko-verojatnostnyj podhod*, Nauka, Saint Petersburg, Russia.

10. Moldavskaja, A.V., Ruvinskaya, V.M. (2014), "Machine learning for malware behavior scenarios", *Informatika i matematicheskie metody v modelirovanii*, vol. 4, no. 2, pp. 149–157.

11. Hopkroft, D.E., Radzhiv Motvani, Ulman, J. (2002), *Vvedenie v teoriju avtomatov, jazykov i vychislenij* [Introduction to Automata Theory, Languages, and Computation], vol. 2, Vil'jams, Moscow, Russia.

12. Aho, A.V., Corasick, M.J. (1975), "Efficient string matching: An aid to bibliographic search", *Communications of the ACM,* vol. 18, no. 6, pp. 333–340.

13. Pungila, C. (2009), "A Bray-Curtis Weighted Automaton for Detecting Malicious Code Through System-Call Analysis", *SYNASC, 11th International Symposium on Symbolic and Numeric Algorithms for Scientic Computing*, Timisoara, pp. 392–400.

14. Kormen, T.H., Leiserson, E.C., Rivest, R.L., and Stein, C. (2007), *Algoritmy: postroenie i analiz* [Introduction to Algorithms], vol. 2, Vil'jams, Moscow, Russia.

15. Bentley, J.L., Saxe, J.B. (1980), "Decomposable searching problems I. Static-to-dynamic transformation", *Journal of Algorithms,* vol. 1, no. 4, pp. 301–358.

16. *Xui Hao Ran.* (2013), "About a few non-classical solutions for data structures", *National Olympiad in Informatics 2013 – Proceedings of the Chinese team participants*, Chengdu, Sichuan, China, pp. 71–85.

17. Daciuk, J., Mihov, S., Watson, B.W., Watson, R.E. (2000), "Incremental construction of minimal acyclic finite-state automata", *Computational Linguistics*, vol. 26, no. 1, pp. 3–16.

Стаття надійшла 30.07.2014.