Бабенко Людмила Климентьевна – Технологический институт федерального государственного автономного образовательного учреждения высшего профессионального образования «Южный федеральный университет» в г. Таганроге; e-mail: blk@fib.tsure.ru; 347928, г. Таганрог, ул. Чехова, 2; тел.: 88634312018; кафедра безопасности информационных технологий; д.т.н.; профессор.

Ищукова Евгения Александровна – e-mail: jekky82@mail.ru; тел.: 88634371905; кафедра безопасности информационных технологий; к.т.н.; доцент.

Маро Екатерина Александровна – e-mail: marokat@gmail.com; тел.: 88634371905; кафедра безопасности информационных технологий; ассистент.

Сидоров Игорь Дмитриевич – e-mail: idsidorov@gmail.com; тел.: 88634371905; кафедра безопасности информационных технологий; к.т.н.; доцент.

Кравченко Павел Павлович – e-mail: kravch@tsure.ru; 347928, г. Таганрог, пер. Некрасовский, 44; тел.: 88634371673; кафедра математического обеспечения и применения ЭВМ; зав. кафедрой.

Babenko Lyudmila Klimentevna – Taganrog Institute of Technology – Federal State-Owned Autonomy Educational Establishment of Higher Vocational Education “Southern Federal University”; e-mail: blk@fib.tsure.ru; 2, Chekhov street, Taganrog, 347928, Russia; phone: +78634312018; the department of security in data processing technologies; dr. of eng. sc.; professor.

Ischukova Evgeniya Aleksandrovna – e-mail: jekky82@mail.ru; phone: +78634371905; the department of security in data processing technologies; cand. of eng. sc.; associate professor.

Maro Ekaterina Aleksandrovna – e-mail: marokat@gmail.com; phone: +78634371905; the department of security in data processing technologies; assistant.

Sidorov Igor Dmitrievich – e-mail: idsidorov@gmail.com; phone: +78634371905; the department of security in data processing technologies; cand. of eng. sc.; associate professor.

Kravchenko Pavel Pavlovich – e-mail: kravch@tsure.ru; 44, Nekrasovskyi, Taganrog, 347928, Russia; phone: +78634371673; the department of software engineering; head of the department.

УДК 004.491

Л.К. Бабенко, Е.П. Тумоян, К.В. Цыганок, М.В. Аникеев

КЛАССИФИКАЦИЯ ВРЕДОНОСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НА ОСНОВЕ ПОВЕДЕНЧЕСКИХ ПРИЗНАКОВ

Классификация является распространенной задачей при анализе вредоносного программного обеспечения и генерации сигнатур для него. Обычным способом классификации вредоносного кода является экспертная оценка похожести антивирусных образцов квалифицированным вирусным аналитиком.

В данной работе предлагается новый метод классификации вредоносного кода на основе поведенческих признаков – последовательности вызовов WinAPI и их аргументов, а также файлов, создаваемых анализируемым приложением. Метод обеспечивает получение двумерного вектора, характеризующего данную программу. Наборы характеризующих векторов кластеризуются с использованием оригинального алгоритма нечеткой кластеризации. Полученные кластеры отражают группы программ, демонстрирующие сходную, с поведенческой точки зрения, активность. Метод был экспериментально исследован на исполняемых тестовых файлах, защищенных упаковкой и шифрованием, а также реальных образцах вредоносного программного обеспечения.

Обнаружение вредоносных программ; метаморфные преобразования; кластеризация; компьютерная безопасность.

L.K. Babenko, E.P. Tumoyan, K.V. Tsyganok, M.V. Anikeev

CLASSIFICATION OF MALICIOUS SOFTWARE BASED ON BEHAVIOR FEATURES

Classification is a common problem of malware analysis and signature generation. Estimation of similarity measure between malware samples made by virus analyst is a common approach to the classification.

This paper describes the new method of malware classification based on behavior features extracted from WinAPI calls, their arguments and files created by the analyzed application. The method allows to obtain a two-dimensional feature vector for each program. Sets of characteristic features are clusterized with a novel algorithm of fuzzy clusterization. Obtained clusters characterize groups of programs that demonstrate similar behavior. The method was investigated experimentally with packed and encrypted executables as well as real samples of malware.

Malware detection; metamorphic transformations; clustering; computer security.

Классификация вредоносного программного обеспечения (ПО) является одной из наиболее распространенных задач в антивирусной индустрии. Она позволяет выявить идентичность или похожесть вредоносных программ. Производительный и точный метод классификации вредоносного ПО позволит решать следующие задачи:

1. Создание общих сигнатур для семейства вредоносных программ. Одним из критичных параметров антивирусных продуктов является размер антивирусной базы. Классификация позволяет выделять группы вирусов и формировать для них одну общую сигнатуру, обнаруживающую родственные вирусы группы.

2. Исследование развития вредоносного кода. Отслеживание изменений вредоносного кода позволяет упростить его анализ, отследить источники кода и т.д.

В настоящее время типовой способ классификации – это экспертный анализ вредоносного кода с использованием программного обеспечения статического (IDA, bindiff) и динамического анализа (ProcessMonitor, различные виды sandbox). Основной проблемой данного типа анализа является высокая сложность классификации, особенно для полиморфного и метаморфного вредоносного ПО. Таким образом, актуальной задачей является разработка и исследование метода, который позволит выполнять автоматическую (с минимальным участием эксперта) классификацию вредоносных программ на основе признаков времени выполнения.

Целью данной работы является разработка автоматического метода классификации вредоносного программного обеспечения на основе поведенческих признаков. Для достижения данной цели необходимо решить следующие задачи:

1. Получение набора поведенческих признаков программ. Под поведенческими признаками мы понимаем характеристики программы времени исполнения, такие как инструкции микропроцессора, вызов библиотечных функций, генерируемый сетевой трафик и т.д.

2. Разработка модели программного кода на основе наблюдаемых поведенческих признаков программы.

3. Разработка метода сравнения моделей программного кода.

4. Экспериментальное исследование и разработка схем применения.

Ниже приведен анализ результатов аналогичных исследований, а также теоретическое и экспериментальное обоснование выбора динамического анализа.

1. Предыдущие исследования. Существующие в настоящее время работы в области обнаружения вредоносного ПО представляют два основных подхода: статический анализ и динамический анализ. Кратко рассмотрим основные работы обоих направлений.

Статический анализ. К данному направлению относятся исследования статического графа исполнения, статистический и нейросетевой анализ дизасемблированного кода программ и подобные. Работа V. Sai Sathyanarayan и др. [1], опуликованная в 2008 году, основана на анализе критических вызовов API, получаемых статически. Сигнатуры данного типа могут обнаруживать не отдельные образцы, а семейства вредоносного ПО. Однако данный подход не работает для упакованных и шифрованных вредоносных программ. В работе [2] был предложен алгоритм анализа инструкций в связи с последовательностью системных вызовов, результат представлен в виде CFG. Работа Wing Wong и Mark Stamp [3] описывает два метода классификации самомодифицирующихся вирусов. Первый – метод на основе вычисления матрицы подобия для инструкций двух исполняемых файлов. После вычисления матрицы можно сформировать простой геометрический (и численный) критерий для оценки подобия файлов. Второй – метод на основе HMM, заключающийся в обучении скрытой марковской модели на последовательности инструкций нескольких эталонных файлов и вычислении вероятности для классифицируемого файла. Система DOME [4] использует статический анализ для определения системных вызовов определенных мест во время мониторинга, чтобы проверить все системные вызовы сделанные из данного места, выявленные в ходе статического анализа.

Данная работа, а также другие работы, в которых используется статический анализ исполняемых файлов, демонстрирует следующий недостаток. При анализе упакованных и шифрованных вирусов (такие вирусы являются наиболее распространенными) предложенные методы смогут выполнить классификацию только распаковщика или расшифровщика. Идентификация только расшифровщика не имеет существенного практического значения – расшифровщики для одного и того же семейства вирусов часто (несколько раз в месяц) изменяются.

Динамический анализ. Предполагает анализ различных характеристик поведения программы времени исполнения, в том числе трасс инструкций, трасс библиотечных и системных вызовов и т.д. Christodorescu и его соавторы [5] представили подход, основанный на анализе трасс исполнения, построении графов зависимости функций и вычислении на их основе спецификации поведения. В работе [6] был предложена система MEDUSA, выполняющая классификацию с использованием динамического анализа API. В работе [7] Sun и другие предложили метод обнаружения обнаружения червей и другого вредоносного ПО с использованием последовательностей вызовов WinAPI. Достоинством работы является, то что был предложен эффективный механизм для отслеживания вызовов WinAPI. Недостаток метода – обнаружение вредоносного ПО происходит при использовании фиксированных адресов вызовов API.

Общим недостатком работ в области статистического анализа и анализа на основе формальных спецификаций (как динамических, так и статических) является требование достаточно большого количества образцов, что в условии классификации новых видов вирусов невозможно.

2. Предлагаемый метод

2.1. Общие положения. Пусть множество программ $\Pi = \{P_i\}, i = 1...N_p$, где P_i – это программа. Задача классификации данного множества состоит в формировании отношения $F(\Pi)$, разделяющего множество Π на непересекающиеся подмножества ε, для которых выполняется условие $\forall P_j \in \varepsilon_i : P_1 = P_2 = ... = P_N$, где $P_1 = P_2$ – эквивалентность программ. Очевидно, что задачу классификации программ можно свести к установлению попарной эквивалентности программ.

Легко показать также, что задачу установления эквивалентности программ можно свести к решению задачи установления эквивалености алгоритмов, реализующих эти программы.

Пусть есть две программы P_1 и P_2. Каждую программу можно представить в виде алгоритма [A] = {I}, где I – это последовательность инструкций алгоритма. Два алгоритма будут эквивалентны, если для любого слова Q из алфавита Γ оба алгоритма генерируют слово W, также принадлежащее Γ.

Необходимо учитывать, что входными данными для программы являются не только данные, вводимые пользователем или получаемые из файлов данных, но также и любые элементы программного окружения. В данных условиях будем рассматривать *контекст программы*. Под контекстом понимается программное окружение, включая файлы данных, переменные в памяти, другие программы и процессы, конфигурацию аппаратного обеспечения ПЭВМ и т.д.

Выделим входной и выходной контекст программы. Входной контекст CI – это контекст, поступающий в программу, выходной контекст CO – это контекст, который генерирует программа, т.е. $CO_1 = P_1(CI_1)$, $CO_2 = P_2(CI_2)$.

Программы P_1 и P_2. эквивалентны при условии, что $CI_1 = CI_2$ и $CO_1 = CO_2$ для любых CI.

Однако полный контекст программы включает значительное количество элементов. Провести сравнение контекстов почти никогда не представляется возможным. С технической точки зрения есть возможность зафиксировать часть входного контекста программы. Это может быть выполнено с использованием механизма снимков состояния операционной системы (snapshots), которые поддерживаются многими виртуальными машинами. Данный механизм позволяет зафиксировать все состояние виртуальной машины кроме двух элементов, которые мы не считаем принципиальными:

1) текущее время операционной системы;

2) текущее состояние сетевого окружения, в том числе активные узлы сети Интернет.

Для сокращения данных выходного контекста мы учитываем следующие соображения:

1. Работа с регистрами CPU и памятью. Обращение к регистрам микропроцессора, ячейкам памяти, стеку или портам устройств. В настоящее время нет эффективных механизмов отслеживания всех инструкций работы с памятью.

2. Взаимодействие с операционной системой. Взаимодействие программы с операционной системой происходит посредством трех механизмов: вызовы WinAPI (наиболее распространенный механизм), вызовы NativeAPI, прямые вызовы обработчиков syscall (наименее универсальный механизм, требующий дополнительных действий со стороны программиста).

В настоящее время мы рассматриваем случай, когда программа взаимодействует с операционной системой посредством WinAPI.

3. Изменение в файловой системе. Вредоносные программы в большинстве случаев должны закрепиться на ПЭВМ жертвы, т.е. создать в файловой системе файлы (возможно, исполняемые) или записи в реестре. В данной работе мы учитываем созданные исследуемой программой файлы.

4. Сетевое взаимодействие программы. Часть вредоносного ПО взаимодействует с управляющими центрами для передачи несанкционированно полученных данных и приема команд. В настоящее время мы игнорируем данные сетевого взаимодействия и опираемся на информацию о сетевом взаимодействии, которая предоставляется вызовами WinAPI.

5. Перехваты и внедрение кода. Перехваты функций и внедрение (inject) кода или библиотек в другие процессы являются типовыми действиями вредоносного программного обеспечения. В данной работе мы частично учитываем эти действия путем контроля вызовов WinAPI. Несмотря на то, что операции с памятью не контролируются, процесс удаленного внедрения кода прослеживается по вызовам WinAPI.

Рассматривая подмножество информации из контекста процесса, мы не можем предложить формальное доказательство того факта, что эта информация достаточна для установления эквивалентности двух программ. Однако в п. 3 приведены результаты экспериментов по классификации программ, которые статистически подтверждают данную гипотезу. Можно сформулировать следующую модель представления программного кода: $P \approx \{C, F\}$, где C – упорядоченное множество системных вызовов, F – множество файлов, созданных программой.

Таким образом, установление эквивалентности двух программ P_1 и P_2 сводится к установлению эквивалентности множеств системных вызовов и множеств файлов. Если множества равны, то установление их эквивалентности - тривиальная задача. В противном случае – полезно определить меру близости данных множеств.

2.2. Сравнение последовательностей системных вызовов. Сформулируем основные условия для определения данной меры:

Условие 1. Нечувствительность меры к наличию незначащих (“мусорных”) элементов множества. Внесение незначащих вызовов является одним из типовых средств обфускации вредоносного ПО и используется для обхода статических анализаторов и эмуляторов антивирусных систем.

Условие 2. Чувствительность меры к порядку следования вызовов. Зависимость от порядка следования вызовов, поскольку их взаимное расположение определяет функциональность программы.

Условие 3. Чувствительность к аргументам. В WinAPI аргументы вызовов существенно влияют на фактически выполняемые действия.

Для вычисления меры мы не можем использовать статистические параметрические методы, поскольку размер статистики – один образец.

В результате анализа данных требований мы предлагаем следующий алгоритм вычисления близости последовательностей:

Решение условия 1:

$[S, I1, I2] = LongestCommonSubsequence(W1, W2)$,

где

LongestCommonSubsequence – операция вычисления наибольшей общей подпоследовательности, в источнике [8]

W1, W2 – последовательности вызовов WinAPI программ P_1, P_2 соответственно;

S – общая подпоследовательность;

I1, I1 – векторы индексов элементов S в W_1, W_2 соответственно;

причем S, I1, I2 имеют длину N.

Решение условия 2:

K – минимальная из длин последовательностей W_1, W_2;

R = 0;

for idx=1:N:

R=R+(1+CompareArgumens(W1[I1[idx]],W2[I2[idx]])/2;

R=R/K;

Df=1-*R*.

Решение условия 3:

CompareArgumens – функция сравнения аргументов, результат [0…1];

*CompareArgumens (W*1[*i*],*W*2[*j*]);

M – количество аргументов вызова;

L = 0;

for idx = 1:M:

if $W_1[i].arg[idx] == W_2[i].arg[idx];$

L = L + 1;

end

end

return L = L/M.

В данном алгоритме:

R – мера близости двух последовательностей вызовов *W*1, *W*2, $R \in [0...1]$;

Dc – расстояние между двумя последовательностями вызовов *W*1, *W*2, $Dc \in [0...1]$.

2.3. Сравнение множества файлов. Сформулируем условия для определения меры близости по файлам:

Условие. Нечувствительность меры к наличию незначащих (“мусорных”) элементов множества. Назовем незначащими файлы, порожденные незначащими вызовами. Обоснование аналогично обоснованию в подразд. 2.2.

Ограничение. Вредоносное ПО создает файлы различных типов, а кроме того – нетипизированные файлы. Таким образом, анализ формата файла и синтаксический разбор в общем случае представляются сложными (или вообще невозможными).

С учетом предыдущего условия и ограничения можно предложить следующую меру близости:

K – минимальная из длин последовательностей W_1, W_2,

R=0

for idx=1:N:

R=R+(*CompareArgumens(W*1[*I*1[*idx*]],*W*2[*I*2[*idx*]];

R=R/K;

Df=1-*Df.*

При этом:

CompareFiles – функция сравнения файлов $\in [0...1]$.

CompareFiles(*W*1[*i*], *W*2[*j*]);

M – минимальная из длин файлов;

*if W*1[*i*] == *‘CreateFile’;*

L=length(*LongestCommonSubstring*(*W*1[*i*]. *filename, W*2[*j*]. *filename*))

end

return L = L/M,

где *LongestCommonSubstring – операция вычисления наибольшей общей подстроки (в источнике [8]) от содержимого файлов, имена которых представлены аргументами функций* W_1[*I filemane, W*2[*j*] *filemane*

R – мера близости по файлам последовательностей W_1 и W_2 $R \in [0...1]$;

D_f – расстояние по файлам, $D_f \in [0...1]$.

2.4. Анализ результатов. После выполнения алгоритмов 2.2 и 2.3 мы получим множество попарных расстояний между анализируемыми программами по

вызовам и по файлам. Приведем набор попарных расстояний к набору расстояний, характеризующих отдельную анализируемую программу. В данной работе для этого предлагается вычисление нормы вектора в Евклидовом пространстве. Однако мы полагаем, что норма на другом топографическом пространстве будет так же приемлема.

Совместив данные о дистанциях файлов и дистанциях наборов вызовов для каждой анализируемой программы, получим двумерный вектор $f_i, i = 1...N_p$, отражающий расстояние от других программ по вызовам и файлам соответственно. Каждый такой вектор можно представить точкой в двумерном пространстве. Множество точек для всех анализируемых программ образуют области. Компактные области являются признаком того, что анализируемые программы подобны.

Для выделения таких компактных областей мы используем нечеткую кластеризацию (fuzzy clustering). Этот вид кластерного анализа позволяет автоматически определять количество кластеров. В данной работе алгоритм нечеткой кластеризации был модифицирован для получения точного количества кластеров при условиях различных размеров кластеров.

Полученные центры кластеров и маркеры принадлежности вектора к кластеру позволяют сделать вывод о близости программ из кластера. Кроме того, в результате кластеризации мы можем автоматически определить количество кластеров, фактически – количество групп в множестве исследуемых программ.

3. Экспериментальная проверка разработанного метода

3.1. Получение информации времени исполнения. Как было описано выше, часть положений разработанного метода не может быть доказана формально и требует экспериментальной проверки.

Для получения данных об исполнении программы используется система Cuckoo Sandbox [9], которая обеспечивает получение:

- трасс вызовов WindowsAPI;
- множества файлов, создаваемых данной программой;
- сетевых пакетов, генерируемых программой;
- трасс ассемблерных инструкций, выполняемых программой.

Мы используем часть данной информации для автоматической классификации анализируемой программы. В частности, в настоящее время используем:

- трассы вызовов WindowsAPI;
- множество файлов, создаваемых данной программой.

3.2. Архитектура экспериментальной системы. Экспериментальная программная система, реализующая метод, описанный в разд. 2, включает интерфейс к системе Cuckoo, подсистему форматирования данных от Cuckoo на языке Python и подсистему анализа на языке Matlab.

Подсистема анализа и кластеризации данных предоставляет графический интерфейс программы, который позволяет изменять исследуемую выборку образцов, размеры кластеров, просматривать элементы нужного кластера и представлять результаты кластеризации в виде графика.

3.3. Результаты экспериментов. В ходе экспериментов было проверено следующее множество файлов, разбитое на группы:

- UPX-Collection: различные исполняемые файлы, которые были упакованы упаковщиком UPX. Тестовый набор используется для тестирования классификации упаковщика. Это не является первичной задачей, однако позволяет сделать выводы о возможностях и ограничениях метода. Размер коллекции – 9 образцов.

- Simple-Collection: один файл был упакован различными упаковщиками. Тестовый набор используется для классификации программ. Размер коллекции – 14 образцов.
- Malware-Collection: набор из вредоносного ПО, полученного из открытых источников. Коллекция насчитывает 69 файлов трасс и около 150 файлов.

Результаты классификации UPX-Collection представлены на рис. 1. Здесь и далее образцы, принадлежащие разным классам, помечены различными маркерами. Исходя из подразд. 2.4 для анализа данных набора UPX-Collection необходимо оценивать значения только по оси Y – близость трасс WinAPI, поскольку различные проверяемые программы создают различные файлы. Как видно из графика, большинство точек имеет одно значение близости по WinAPI. С учетом метода, описанного в разд. 2, можно показать, что это те вызовы, которые генерируются распаковщиком UPX. Из графика видно, что ошибка классификации составляет около 22 %.

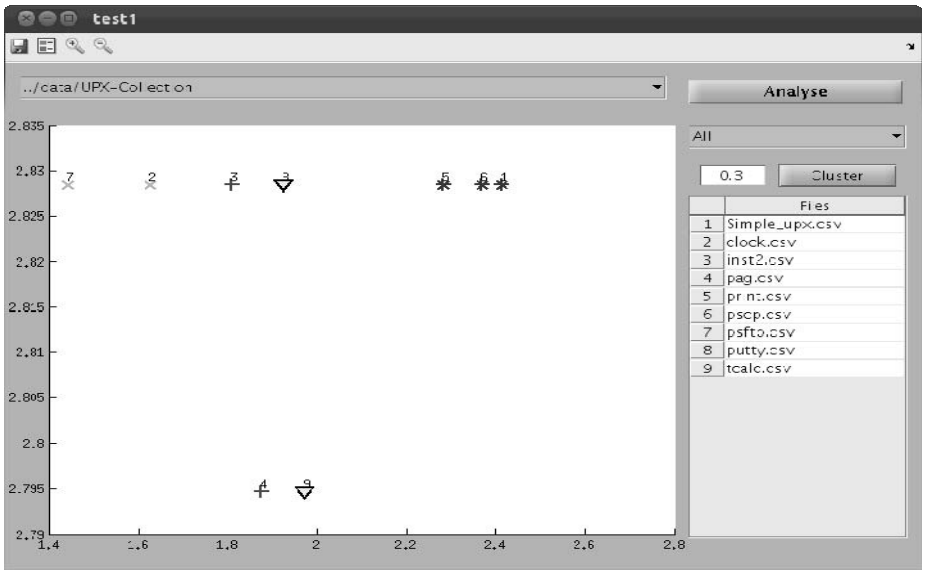

Рис. 1. Результат классификации набора UPX-Collection

На рис. 2 представлены результаты кластеризации набора Simple-Collection. Как видно из графика, при использовании различных упаковщиков последовательность вызовов самой программы незначительно меняется, отсюда и появились некоторые отклонения для упаковщиков upack и morphine. Ошибка классификации составляет 21,4 %.

При тестировании набора Malware-Collection для проверки результатов мы опирались на данные о вирусах, полученные из системы VirusTotal. Ошибка классификации составляет около 23,5 %. Общие результаты тестирования сведены в табл. 1.

Таблица 1

Результаты тестирования тестовых коллекций

Общее количество проверенных образцов	Средняя ошибка классификации, %	Время расчета близости пары образцов, с	Потребление памяти для классификации пары образцов, Мб
92	22 %	от 0,05 до 0,1	от 3 до 5

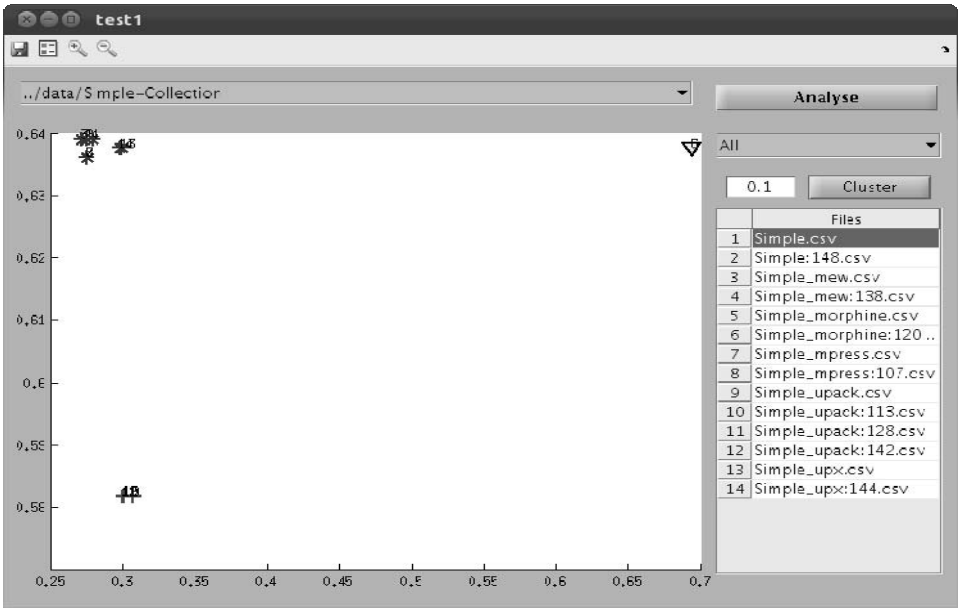

Рис. 2. Результат классификации набора Simple-Collection

Выводы. Предложенный в работе метод обеспечивает оценку близости и кластеризацию образцов вредоносного ПО на основе поведенческих признаков. Полученные кластеры используются для классификации вредоносного ПО. Экспериментальная реализация метода демонстрирует время анализа одной пары образцов от 0,05 до 0,1 секунды, что примерно в 10 раз меньше, чем в представленных в научной печати методах на основе динамического анализа. Разработанный метод обеспечивает не только классификацию образцов метаморфного вредоносного ПО, но и, в отличие от других методов (таких, как метод на основе алгебраических спецификаций), предоставляет информацию, которую эксперт может оценить визуально и интерпретировать в терминах предметной области.

Результаты работы используются при выполнении совместного гранта РФФИ и ДНТ Индии 11-07-92693-ИНД_а “Разработка методов обнаружения и анализа метаморфного вредоносного программного обеспечения”.

Теоретическое исследование метода и проведенные к настоящему времени эксперименты позволяют выделить следующие проблемы и ограничения:

1. Ряд вариантов вредоносного ПО прекращает функционирование, в случае определения виртуальной машины. Необходимо отметить, что данная проблема не является критичной, поскольку в основном это относится к телам вирусов, а основная цель данной работы – идентификация и обнаружение дропперов вирусов. Однако мы планируем провести дополнительные исследования в данной области для уточнения ситуации.

2. Планируется провести широкие экспериментальные исследования разработанного метода. Это станет возможным после запуска разрабатываемой нами активной системы-ловушки во втором квартале 2012 г.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Sai Sathyanarayan V., Kohli P., Bruhadeshwar B.* Signature Generation and Detection of Malware Families // Proceedings of the 13th Australasian conference on Information Security and Privacy, Australia, Wollongong. – 2008. – P. 336-349.

2. *Lee H., Jeong K.* Code graph for malware detection // Proceedings of International conference on Information Networking. – 2008. – P. 1-5.
3. *Stamp M., Wong W.* Hunting for metamorphic engines // Comput Virol. – France: Springer-Verlag France, 2006. – № 2. – P. 221-229.
4. *Rabek J.C., Khazan R.I., Lewandowski S.M., Cunningham R.K.* Detection of injected, dynamically generated, and obfuscated malicious code // Proceedings of the 2003 ACM workshop on Rapid malcode. – USA, Washington. – 2003.
5. *Christodorescu M., Jha S., Kruegel C.* Mining specifications of malicious behavior // Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. – Croatia, Dubrovnik, 2007.
6. *Vinod P., Harshit Jain, 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.
7. *Sun H., Lin Y., Wu M.* Api monitoring system for defeating worms and exploits in ms-windows system // Information Security and Privacy, 11th Australasian Conference, ACISP 2006. – Vol. 4058 of Lecture Notes in Computer Science. – Australia, Melbourne. – 2006.
8. *Cormen T.H., Leiserson C.E., Rivest R.L., Stein C.* Introduction to Algorithms. – 2nd ed. – Boston: MIT Press, McGraw-Hill, 2001. – 1180 p.
9. Сайт проекта Cuckoo Sandbox [Электронный ресурс]. – Режим доступа: http://cuckoobox.org, свободный.

Статью рекомендовал к опубликованию д.т.н., профессор Я.Е. Ромм.

Бабенко Людмила Климентьевна – Технологический институт федерального государственного автономного образовательного учреждения высшего профессионального образования «Южный федеральный университет» в г. Таганроге; e-mail: blk@fib.tsure.ru; 347928, г. Таганрог, ул. Чехова, 2; тел.: 88634312018; кафедра безопасности информационных технологий; д.т.н.; профессор.

Тумоян Евгений Петрович – e-mail: e.tumoyan@gmail.com; кафедра безопасности информационных технологий; к.т.н.; доцент.

Цыганок Ксения Васильевна – e-mail: kleo.148@gmail.com; кафедра безопасности информационных технологий; студентка.

Аникеев Максим Владимирович – e-mail: anikeev@users.tsure.ru; кафедра безопасности информационных технологий; к.т.н.; доцент.

Babenko Lyudmila Klimentevna – Taganrog Institute of Technology – Federal State-Owned Autonomy Educational Establishment of Higher Vocational Education “Southern Federal University”; e-mail: blk@fib.tsure.ru; 2, Chehov street, Taganrog, 347928, Russia; phone: +78634312018; the department of security in data processing technologies; dr. of eng. sc.; professor.

Tumoyan Evgeny Petrovich – mail: e.tumoyan@gmail.com; the department of security in data processing technologies; cand. of eng. sc.; associate professor.

Tsyganok Xenia Vasil’evna – e-mail: kleo.148@gmail.com; the department of security in data processing technologies; student.

Anikeev Maxim Vladimirovich – e-mail: anikeev@users.tsure.ru; the department of security in data processing technologies; cand. of eng. sc.; associate professor.