УДК 004.056:061.68

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

Технологический институт Южного федерального университета, Таганрог, Россия

ПОВЕДЕНЧЕСКАЯ КЛАССИФИКАЦИЯ ПОЛИМОРФНОГО И МЕТАМОРФНОГО ВРЕДОНОСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В работе предлагается новый метод классификации вредоносного кода на основе поведенческих признаков. Была разработана мера близости программ, учитывающая последовательности вызовов WinAPI программ и их аргументов, а также файлов создаваемых анализируемым приложением. Для выделения групп программ используется кластерный анализ. Метод был экспериментально исследован на реальных образцах вредоносного программного обеспечения. Работа выполнена при поддержке Российского фонда финансирования фундаментальных исследований, грант 11-07-92693-ИНД_а.

Ключевые слова: *антивирусная защита, классификация, кластерный анализ, полиморфизм программ, метаморфизм программ.*

Проблема классификации вредоносных программ

Классификация вредоносного программного обеспечения (ПО) является одной из наиболее распространенных задач в антивирусной индустрии при подготовке и генерации сигнатур. В данном контексте – классификация представляет собой задачу определения похожести нескольких программ. Классификация вредоносных программ позволяет:

1. Выделять группы вирусов и формировать для них одну общую сигнатуру, обнаруживающую родственные вирусы группы. Это уменьшает размер антивирусной базы и увеличивает обобщающие способности сигнатур.

2. Отслеживать изменения вредоносного кода, что упрощает его анализ, отследить источники кода и т.д.

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 методов, которые позволят выполнять автоматическую (с минимальным участием эксперта) классификацию вредоносных программ.

Целью данной работы является разработка автоматического метода классификации вредоносного программного обеспечения. В основном – решаемая задача связана с обнаружением т.н. дропперов вирусов – программ, выполняющих первоначальное получение управления и загрузку тела вируса на целевую ПЭВМ. При формализации решаемых задач, необходимо учитывать, что полиморфное вредоносное программное обеспечение зашифровано, что исключает возможность его статического анализа. Таким образом, мы приходим к необходимости анализа признаков времени исполнения. Для достижения поставленной цели необходимо решить следующие основные 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, а также метода сравнения моделей программного кода.

3. Программная реализация разработанных моделей и методов и их экспериментальное исследование.

В следующих разделах приведен анализ результатов аналогичных исследований, а также описание и экспериментальное обоснование разработанного метода динамического анализа.

1. Предыдущие работы

Существующие в настоящее время исследования 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 дизассемблированного кода программ и подобные. Работа V. Sai Sathyanarayan и др. [1], опубликованная в 2008 году, основана на анализе критических вызовов API, получаемых статически. Сигнатуры такого типа могут обнаруживать не отдельные образцы, а семейства вредоносного ПО. В работе [2] был предложен алгоритм анализа инструкций в связи с последовательно-

стью 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, состоит 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.

У всех работ использующих статический анализ исполняемых файлов присутствует общий 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, а один и тот же публичный расшифровщик (например, UPX) может использоваться при защите сотен различных версий вредоносного программного обеспечения.

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 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.

Приведенные методы обеспечивают возможности обнаружения и опознавания вирусов в случае наличия значительного количества образцов. Это с необходимостью вытекает из того факта что, используемые модели являются статистическими или другим способом обобщают общие признаки нескольких программ. В условии классификации новых видов вирусов невозможно. Более того, часто в распоряжении аналитиков есть только один образец вредоносной программы. Ожидание нескольких образцов может привести к тому, что вирус будет проанализирован слишком поздно и успеет заразить большое количество пользовательских компьютеров.

2. Предлагаемый метод

Исходя из вышеописанного сформулируем следующее требование к разрабатываемым модели и методу. Необходима модель программы, которая может быть сформирована на основании поведенческих признаков одного образца вируса.

Пусть имеется множество программ $\Pi = \{P_i\}, i = 1..N_p$, где P_i - это отдельная программа. Задача классификации данного множества состоит в формировании отношения $F(\Pi)$, разделяющего множество Π на непересекающиеся подмножества ε, для которых выполняется условие:

$$\forall P_i, P_j \in \varepsilon_k : P_i = P_j,$$

где $P_1 = P_2$ - означает эквивалентность программ. Очевидно, что задачу классификации программ можно свести к установлению попарной эквивалентности программ.

Рассмотрим возможные способы установления эквивалентности программ. Данную задачу можно свести к решению задачи установления эквивалентности 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 инструкции алгоритма. Два алгоритма будут эквивалентны, если для любого слова Q из алфавита Γ оба алгоритма генерируют одно слово W также принадлежащее Γ:

$$\forall Q : W_i = W_j, \qquad (1)$$

где $W_i = A_i(Q)$.

Необходимо учитывать, что входными данными для программы являются не только данные вводимые пользователем или получаемые из фалов данных, но также и любые элементы программного окружения. В данных условиях расширим понятие входного слова алгоритма на понятие *контекста 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. $CO_i = P_i(CI)$. Таким образом, будем считать, что программы P_i и P_j эквивалентны если:

$$\forall CI : CO_i = CO_j, \qquad (2)$$

Данное утверждение легко доказать. Поскольку $Q \in CI$, а $W \in CO$, то выполнение условия (2) с необходимостью приводит к выполнению условия (1).

Необходимо отметить, однако, что полный контекст программы включает значительное количество элементов. Выполнение полного сравнения контекстов для программ в современных операционных системах не представляется возможным. Достаточно указать, что в общем случае контекст включает виртуальную память системы, образ жесткого диска, результаты взаимодействия с пользователем и сетевого взаимодействия программы. 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 одного элемента, а именно текущего состояния сетевого окружения, в том числе - активные узлы сети Интернет. Мы, по необходимости, игнорируем данный элемент.

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. Существует широкий класс инструкций микропроцессора могут выполнять явную запись в ячейку памяти (MOV, XLAT, ADD, SUB), а также такие, которые выполняют это косвенным образом (PUSH, POP). В настоящее время нет эффективных механизмов отслеживания *всех* инструкций работы с памятью. Поэтому инструкции обращения к памяти мы не учитываем.

2. Взаимодействие с операционной системой. Взаимодействие программы с операционной системой происходит посредством трех механизмов: вызовы WinAPI (наиболее распространенный и универсальный механизм), вызовы NativeAPI, прямые вызовы обработчиков syscall (наименее универсальный механизм, требует 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 объекты файловой системы – директории, файлы, Alternate Data Stream.

4. Сетевое взаимодействие. 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. Однако, при учете этих данных предложенная нами модель изменится незначительно.

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 данные действия путем контроля вызовов WinAPI. Несмотря на то, что операции с памятью не контролируются, процесс удаленного внедрения кода прослеживается по вызовам WinAPI.

В данной работе рассматривается подмножество информации из контекста программы.

В работе отсутствуют формальные доказательства того факта, что эта информация достаточна для установления эквивалентности двух программ. Более того, подобные формальные доказательства в общем случае получить невозможно. Несмотря на это, в пункте 3 приведены результаты экспериментов по классификации программ, которые статистически подтверждают данную гипотезу. Мы предлагаем следующую модель представления программного кода:

$$P \approx \{C, F\},$$

где C - упорядоченное множество вызовов WinAPI, F - множество объектов файловой системы, созданных программой.

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

Подобие по системным вызовам. 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 P1,P2 соответственно;
S - общая подпоследовательность;
I1,I2 - векторы индексов элементов S в W1, W2 соответственно;
Причем S,I1,I2 имеют длину N
```

Решение условия 2:

```
K - минимальная из длин последовательностей W1, W2
    R = 0;
    for idx=1:N:
        R = R + (1 + CompareArguments(
      W1[I1[idx]],W2[I2[idx]]))/2;
       R = R/K;
       Dc = 1- R;
    end
```

Решение условия 3:

```
CompareArguments - функция сравнения аргументов, результат в интервале [0..1];
M - количество аргументов вызова;
CompareArguments(W1[i],W2[j]):
      L = 0;
      for idx = 1:M:
          if W1[i].arg[idx] == W2[j].arg[idx]:
                    L = L + 1;
          end
      end
      return L = L/M;
```

В данном алгоритме:

R - мера близости двух последовательностей вызовов *W1,W2, R в интервале [0..1];*

Dc - расстояние между двумя последовательностями вызовов *W1, W2, Dc в интервале [0...1];*

Подобие по файлам. Сформулируем условие для определения меры подобия по файлам:

Условие. Нечувствительность меры к наличию незначащих («мусорных») элементов множества. Назовем незначащими файлы, порожденные незначащими вызовами. Обоснование аналогично обоснованию в п/п 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 W1,W2
    R = 0;
    for idx=1:N:
        R = R + (CompareFiles(W1[I1[idx]], W2[I2[idx]]));
       R = R/K;
       Df = 1 – R;
    end
```

При этом:

```
CompareFiles - функция сравнения файлов, R в интервале [0..1];
M - минимальная из длин файлов;
CompareFiles(W1[i],W2[j]):
    if W1[i] == 'CreateFile':
        L=length(LongestCommonSubstring(
        W1[i].filename, W2[j].filename));
    end
    return L = L/M;
```

где LongestCommonSubstring - операция вычисления наибольшей общей подстроки (описана в источнике [8]) от содержимого файлов, имена которых представлены аргументами функций W1[i].filename, W2[j].filename

После выполнения предыдущих алгоритмов было получено множество попарных расстояний между анализируемыми программами по вызовам и по файлам. Далее мы приводим набор попарных расстояний к набору расстояний, характеризующих отдельную анализируемую программу. В данной работе для этого предлагается вычисление нормы вектора в Евклидовом пространстве. Мы не исключаем, 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 используется алгоритм нечеткой кластеризации (fuzzy clustering). Данный вид кластерного анализа позволяет автоматически определять количество кластеров.

В данной работе алгоритм нечеткой кластеризации был модифицирован. Мы используем иерархическое разбиение множества на кластеры, для получения точного количества кластеров при условиях их различных размеров.

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

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:

- Трассы вызовов WindowsAPI.
- Множество файлов, создаваемых данной программой.
- Сетевые пакеты, генерируемые программой.
- Трассы ассемблерных инструкций выполняемых программой

Для автоматической классификации в данной работе используется только часть данной информации, в частности:

- Трассы вызовов WinAPI.
- Множество файлов создаваемых данной программой.

Мы проводим анализ вредоносных программ в виртуальных машинах, управляемых Cuckoo Framework.

При обработке нового образца виртуальная машина восстанавливается из снимка состояния.

После обработки образца – состояние машины (которая, может быть к этому моменту заражена) сбрасывается.

Результаты обработки передаются в систему Cuckoo.

Разработанная экспериментальная программная система, основана на методе и алгоритмах, описанных в разделе 2.

Процесс классификации показан на рис. 1.

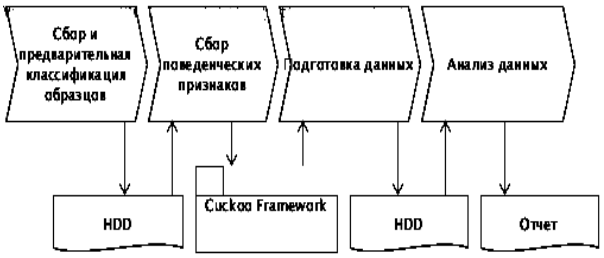

Рис. 1. Процесс классификации

Система включает интерфейс к Cuckoo Framework, подсистему форматирования данных 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.

В ходе экспериментов было проверено следующее множество файлов, разбитое на три 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 проведения экспериментов по классификации программ с различными упаковщиками. Размер коллекции - 14 образцов.

Malware-Collection: набор из вредоносного ПО, полученного из открытых источников. Коллекция насчитывает 1080 образцов трасс программ.

Необходимо обратить внимание, что для анализа данных набора UPX-Collection необходимо оценивать значения только по оси Y - близость последовательностей 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 2, можно показать, что это те вызовы, которые генерируются распаковщиком UPX. Из графика (рис. 2) видно, что ошибка классификации составляет около 22%. Здесь и далее образцы, принадлежащие разным классам помечены различными маркерами.

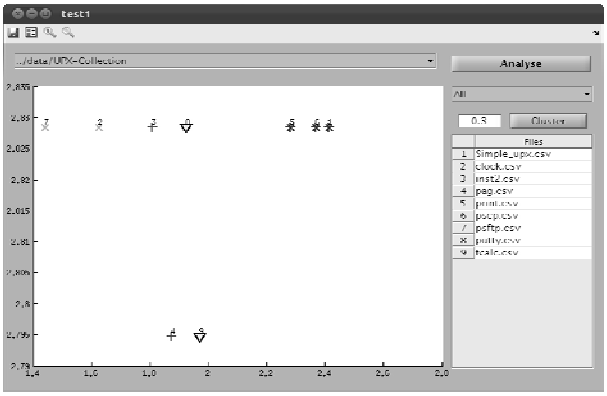

Рис. 2. Результат классификации набора UPX

Как видно из графика (рис. 3) 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%.

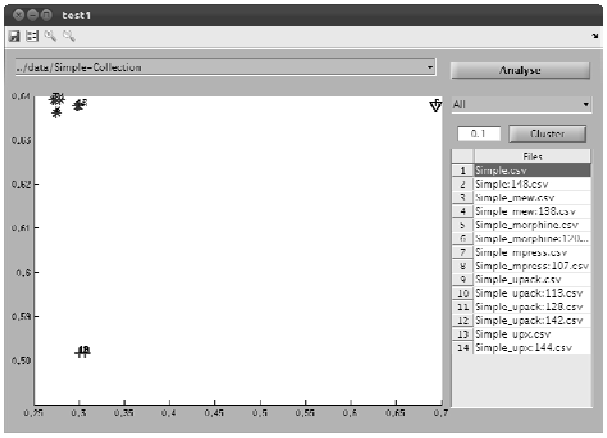

Рис. 2. Результат классификации тестовой программы

При тестировании набора Malware-Collection для проверки результатов мы опирались на данные о вирусах, полученные из системы VirusTotal. Ошибка классификации составляет около 18.5%. На графике (рис. 4) видно, что образцы сформированы в компактные группы принадлежащие различным вирусам. В частности выделенная группа включает образцы вируса Zeus.

Общие результаты тестирования сведены в табл. 1 (время расчетов приведено для одной пары образцов).

Таблица 1

Результаты экспериментов

Кол-во	Ошибка	Время, с	Память, Мб
1103	20%	0.05 - 0.1	5 - 10

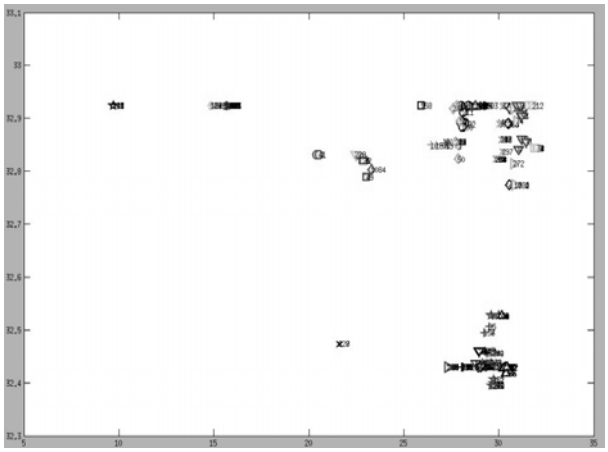

Рис. 3. Результат классификации набора вредоносных программ

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. Мы планируем провести широкие экспериментальные исследования разработанного метода. Это станет возможным после запуска разрабатываемой нами активной системы-ловушки во втором квартале 2012 года.

Список литературы

1. V. Sai Sathyanarayan, Pankaj Kohli, Bezawada Bruhadeshwar, 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 . - NY: ACM New York .- 2010.

7. Sun H., Lin Y., Wu M., Api monitoring system for defeating worms and exploits in ms-windows system // In Information Security and Privacy, 11th Australasian Conference, ACISP 2006 / Volume 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.

Поступила в редколлегию 22.03.2012

Рецензент: д-р. техн. наук, проф. Я.Е. Ромм, ТГПИ им. А.П.Чехова, Таганрог, Россия.

ПОВЕДІНКОВА КЛАСИФІКАЦІЯ ПОЛІМОРФНОГО І МЕТАМОРФНОГО ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Є.П. Тумоян, Л.К. Бабенко, К.В. Циганок, М.В. Аникєєв

Представлені модель і метод класифікації шкідливого програмного забезпечення, захищеного поліморфними і метаморфними, перетвореннями на основі поведінкових ознак.

Ключові слова: *антивірусний захист, класифікація, кластерний аналіз, поліморфізм програм, метаморфізм програм.*

BEHAVIORAL CLASSIFICATION POLYMORPHIC AND METAMORPHIC OF HARMFUL SOFTWARE

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

A model and method of classification of harmful software, protected polymorphic and metamorphic is presented, by transformations on the basis of behavioral signs.

Keywords: *anti-virus defence, classification, cluster analysis, polymorphism of the programs, metamorphicof the programs.*