

СПОСОБ И СИСТЕМА ВЫЯВЛЕНИЯ ВРЕДОНОСНЫХ ФАЙЛОВ В НЕИЗОЛИРОВАННОЙ СРЕДЕ

ОБЛАСТЬ ТЕХНИКИ

Настоящее техническое решение относится к области вычислительной техники, в частности, к компьютерно-реализуемому способу и системе выявления вредоносных файлов в неизолированной среде.

УРОВЕНЬ ТЕХНИКИ

Из уровня техники известно решение US 2019/0005239 A1, 03.01.2019, в котором раскрыт способ анализа вредоносного кода с помощью электронного устройства, где анализируют вредоносный код с помощью электронного устройства, причем способ содержит этапы, на которых: принимают исполняемый файл; перед установкой полученного исполняемого файла анализируют исполняемый файл на предмет сбора данных о подозрительном вредоносном коде из исполняемого файла; нормализуют собранные данные о подозрительном вредоносном коде и анализируют их на основе алгоритма вероятностной модели, чтобы определить данные о предполагаемом вредоносном коде и вывод результата определения.

Из уровня техники известно еще одно решение, выбранное в качестве наиболее близкого аналога, RU 2654146 C1. В данном решении раскрыта система признания файла вредоносным, которая содержит: а) средство извлечения ресурсов, предназначенное для: извлечения ресурсов из анализируемого файла; передачи извлеченных ресурсов средству формирования правил и средству поиска правил; б) средство формирования правил, предназначенное для: формирования по меньшей мере одного правила, устанавливающего функциональную зависимость между полученными ресурсами (далее - правило), при этом правило формируется путем создания из полученных ресурсов искусственной нейронной сети, где узлами искусственной нейронной сети являются средства анализа полученных ресурсов, а связи между узлами, сформированные во время создания нейронной сети, указывают на функциональную зависимость между полученными ресурсами; передачи каждого сформированного правила средству сравнения правил; в) средство поиска правил, предназначенное для: осуществления поиска по меньшей мере одного правила в базе ресурсов вредоносных файлов на основании полученных ресурсов; передачи каждого найденного правила средству сравнения правил; г) средство сравнения правил, предназначенное для: вычисления степени схожести между полученными от средства формирования правил и средства поиска правилами; передачи вычисленной степени схожести средству вынесения решения; д) средство вынесения решения, предназначенное

для: признания анализируемого файла вредоносным, в случае, когда полученная степень схожести превышает заранее заданное пороговое значение.

Приведенные выше известные из уровня техники решения направлены на решение проблемы определения вредоносных файлов в системе до их запуска. Однако стоит
5 отметить, что разработчики постоянно совершенствуют свое вредоносное программное обеспечение, что уменьшает вероятность его обнаружения в системе.

Предлагаемое решение направлено на устранение недостатков современного уровня техники и отличается от известных ранее тем, что предложенный способ, основанный на статическом анализе происходит на большем количестве извлеченных из файла данных, так
10 как для их получения используются как бинарный вид исполняемого файла, так и дизассемблированный, кроме того нами предложено использование графов потоков, которые помогают при выявлении аномалий в поведении файла без его запуска. Дополнительно, стоит отметить, что в заявленном решении использован ансамбль классификаторов, который служит для улучшения эффективности прогнозирования по
15 сравнению с отдельными классификаторами, а также является менее громоздким решением, чем нейронная сеть.

СУЩНОСТЬ ИЗОБРЕТЕНИЯ

Технической проблемой, на решение которой направлено заявленное решение,
20 является создание компьютерно-реализуемого способа и системы выявления вредоносных файлов в неизолированной среде, которые охарактеризованы в независимых пунктах формулы. Дополнительные варианты реализации настоящего изобретения представлены в зависимых пунктах изобретения.

Технический результат заключается в повышении точности выявления вредоносных
25 файлов в неизолированной среде.

Заявленный результат достигается за счет осуществления компьютерно-реализуемого способа выявления вредоносных файлов в неизолированной среде, где способ содержит:

подготовительный этап, на котором:

30 формируют подборку файлов, которая содержит по меньшей мере один вредоносный исполняемый файл и по меньшей мере один не вредоносный исполняемый файл;

анализируют по меньшей мере один исполняемый файл, при этом:

35 извлекают данные из бинарного и дизассемблированного видов исполняемого файла на основе которых создаются параметры для дальнейшего

обучения классификатора, причем дополнительно статистическим методом определяют параметры, характерные вредоносным файлам и/или наоборот не вредоносным;

причем дополнительно извлекают первый и второй граф потоков;

5 на основе полученных параметров строят первый и второй вектор признаков; создают ансамбль классификаторов из: первого обученного классификатора

на основе первого вектора признаков, второго обученного классификатора на основе второго вектора признаков; третьего классификатора, обученного на основе первого графа потоков, четвертого классификатора, обученного на основе второго графа потоков, и

10 причем для каждого классификатора заранее определяют решающий приоритет;

рабочий этап, на котором:

получают, по меньшей мере один исполняемый файл;

15 запускают обученный на подготовительном этапе ансамбль классификаторов, для выявления вредоносных исполняемых файлов;

выводят результат анализа.

В частном варианте реализации описываемого способа, первый вектор признаков содержит все числовые параметры, полученные при анализе, и в котором второй вектор

20 содержит все строковые параметры, полученные при анализе.

В частном варианте реализации описываемого способа, извлекают, по меньшей мере, следующие свойства файла: первый граф потоков (Control Flow Graph), второй граф потоков (Data Flow Graph), метаданные (Metadata), таблицу импорта (Import Table), таблицу экспорта (Export table), энтропии (Byte/Entropy Histogram).

25 В другом частном варианте реализации описываемого способа, из по меньшей мере одного бинарного исполняемого файла извлекают, по меньшей мере, следующие параметры:

N-граммы байт;

Информация из полей исполнимого файла;

30 Энтропия секций файла;

Метаданные;

Гистограммы распределения длин строк.

В другом частном варианте реализации описываемого способа, информация, извлекаемая из полей по меньшей мере одного бинарного исполняемого файла, содержит

35 по меньшей мере: количество секций в файле, размер заголовков, размер кода, наличие

цифровой подписи, наличие импортируемых dll и функций, где были использованы эти библиотеки, название экспортируемых функций, данные из каталога ресурсов, версия сборщика.

5 В другом частном варианте реализации описываемого способа, энтропия, извлекаемая из полей по меньшей мере одного бинарного исполняемого файла, рассчитывается с помощью применения метода скользящего окна, с последующим применением ряда статистических мер для характеристики полученного вектора энтропийных мер.

10 В другом частном варианте реализации описываемого способа, извлекаемые из заголовка по меньшей мере одного бинарного исполняемого файла, содержат: количество секций в файле, размер секций, наличие контрольной суммы, размер файла, время создания, версия операционной системы, версия сборщика.

В другом частном варианте реализации описываемого способа, из по меньшей мере одного дизассемблированного исполняемого файла извлекают, по меньшей мере, 15 следующие параметры:

Символы;

Регистры;

Операционные коды;

Частота обращения к системным интерфейсам (Windows APIs);

20 Секции;

Метаданные.

В другом частном варианте реализации описываемого способа, дополнительно определяют частоту появления символов, извлекаемых из по меньшей мере одного дизассемблированного исполняемого файла, причем символы могут представлять по 25 меньшей мере: “-“, “+“, “*“, ”]”, ”[“, ”?”, ”@”.

В другом частном варианте реализации описываемого способа, дополнительно вычисляют отношение количества известных регистров, извлекаемых из по меньшей мере одного дизассемблированного исполняемого файла, к количеству неизвестных регистров, извлекаемым из дизассемблированного файла.

30 В другом частном варианте реализации описываемого способа, секции, извлекаемые из по меньшей мере одного дизассемблированного исполняемого файла, определены заранее и представляют собой: .text, .data, .bss, .rdata, .edata, .idata, .rsrc, .tls и .reloc.

В другом частном варианте реализации описываемого способа, метаданные, извлекаемые из по меньшей мере одного дизассемблированного исполняемого файла, 35 представляют собой размер файла и количество строк кода.

В другом частном варианте реализации описываемого способа, результат анализа может быть бинарным или вероятностным.

В другом частном варианте реализации описываемого способа, решающий приоритет каждого из ансамбля классификаторов определяется на основе обучения на тестовой выборке, причем для оценки используются по меньшей мере: точность, полнота, F-мера.

Заявленный результат также достигается за счет системы выявления вредоносных файлов в неизолированной среде, которая содержит следующие модули:

модуль извлечения данных, выполненный с возможностью извлечения данных, по меньшей мере из одного исполняемого файла;

вычислительное устройство, выполненное с возможностью осуществления вышеуказанного способа;

модуль вывода.

ОПИСАНИЕ ЧЕРТЕЖЕЙ

Реализация изобретения будет описана в дальнейшем в соответствии с прилагаемыми чертежами, которые представлены для пояснения сути изобретения и никоим образом не ограничивают область изобретения. К заявке прилагаются следующие чертежи:

Фиг.1, иллюстрирует систему выявления вредоносных файлов в неизолированной среде.

Фиг.2, иллюстрирует компьютерно-реализуемый способ выявления вредоносных файлов в неизолированной среде.

Фиг. 3, иллюстрирует блок схему анализа файла на основе графа потока управления.

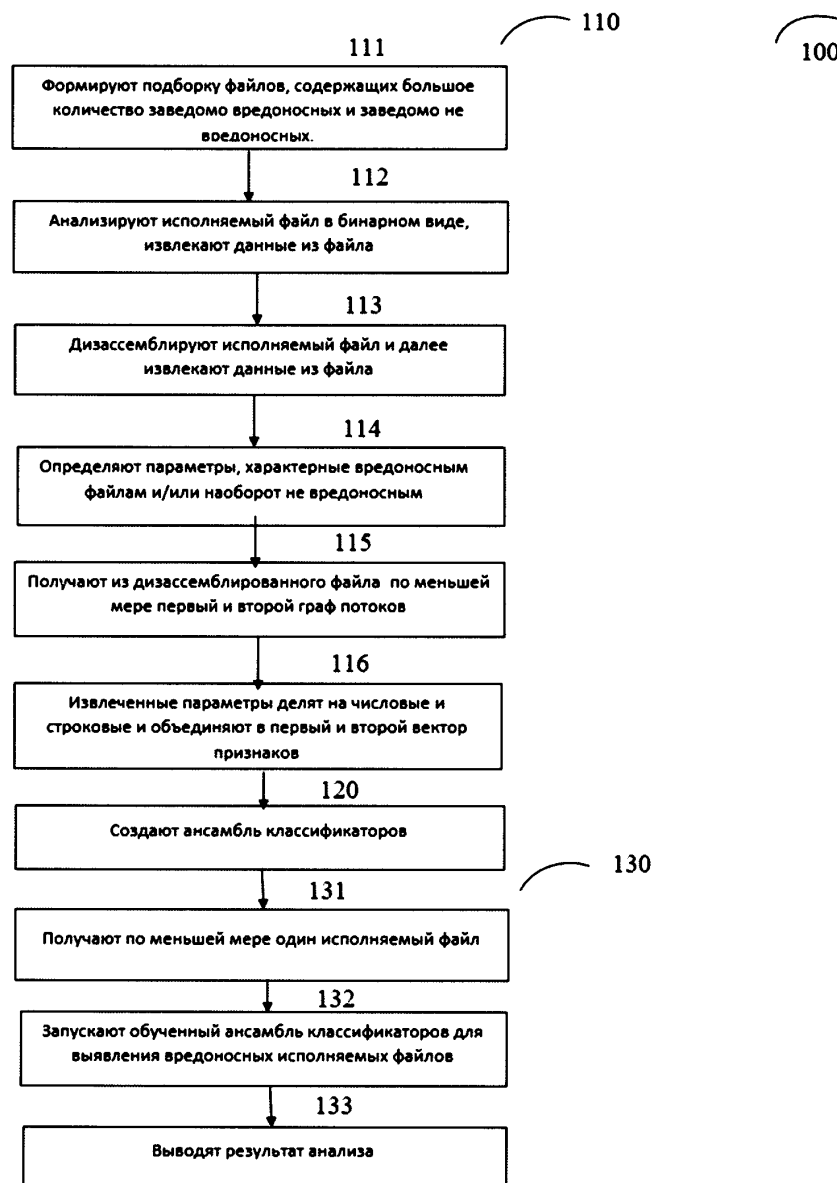
Фиг. 4а, иллюстрирует пример по определению точек в программе.

Фиг. 4b, иллюстрирует пример по определению точек в программе.

Фиг. 5, иллюстрирует пример общей схемы вычислительного устройства.

ДЕТАЛЬНОЕ ОПИСАНИЕ ИЗОБРЕТЕНИЯ

В приведенном ниже подробном описании реализации изобретения приведены многочисленные детали реализации, призванные обеспечить отчетливое понимание настоящего изобретения. Однако, квалифицированному в предметной области специалисту, будет очевидно каким образом можно использовать настоящее изобретение, как с данными деталями реализации, так и без них. В других случаях хорошо известные



Фиг.2

- 111 Générer une présélection de fichiers comprenant un grand nombre de fichiers manifestement malveillants et manifestement non malveillants
- 112 Analyser le fichier exécutable sous forme binaire et extraire des données du fichier
- 113 Désassembler le fichier exécutable puis extraire des données du fichier
- 114 Déterminer des paramètres caractéristiques de fichiers malveillants et/ou inversement non malveillants
- 115 Obtenir à partir du fichier désassemblé au moins un premier et un second vecteurs de flux
- 116 Les paramètres extraits sont divisés en numériques et linéaires et regroupés dans un premier et un second vecteurs d'indices
- 120 Générer un ensemble de classificateurs
- 131 Obtenir au moins un fichier exécutable
- 132 Lancer l'ensemble instruit de classificateurs pour révéler des fichiers exécutables malveillants
- 133 Editer le résultat de l'analyse

данные из таблицы импорта для указания, какие внешние функции и из каких модулей требуются исполняемому файлу для нормальной работы;

данные из таблицы экспорта, определяющие имена и/или номера экспортируемых функций с их положением в виртуальной памяти процесса;

- 5 метаданные (Metadata) из заголовка исполняемого файла, например, количество секций в файле, время создания, тип процессора, версия операционной системы, версия сборщика, суммарный размер секций, контрольная сумма;

данные из таблицы импорта для указания, какие внешние функции и из каких модулей требуются исполняемому файлу для нормальной работы;

- 10 данные из таблицы экспорта, определяющие имена и/или номера экспортируемых функций с их положением в виртуальной памяти процесса;

метаданные (Metadata) из заголовка исполняемого файла, например, количество секций в файле, время создания, тип процессора, версия операционной системы, версия сборщика, суммарный размер секций, контрольная сумма;

- 15 гистограммы (Byte/Entropy Histogram), на основе которой вычисляются бинарные значения двумерной гистограммы энтропии байт, моделирующей распределение байт в файле (чтобы построить гистограмму энтропии байт, используется 1024-байтовое окно, которое смещается с шагом 256 байт по анализируемому бинарному файлу).

Более подробно могут быть рассмотрены, следующие извлекаемые параметры:

- 20 N-граммы байт, причем в данном случае берется последовательность из n байтов, извлеченных из исполняемого файла и каждый элемент байтовой последовательности может иметь одно из 256 значений. Таким образом, рассматриваются гистограммы частотности байт, которые представлены в виде векторов длины 256, которые отражают частоту появления каждого значения.

- 25 Информация из полей исполнимого файла, например количество секций в файле; размер заголовков; размер кода; наличие цифровой подписи; названия импортируемых dll и функций, где были использованы эти библиотеки; названия экспортируемых функций; данные из каталога ресурсов; версия сборщика; версия операционной системы; тип процессора; суммарный размер секций; контрольная сумма.

- 30 Энтропия секций файла, причем подсчет энтропии файла используется в статическом анализе для обнаружения обфускаций кода. Также стоит отметить, что вредоносное ПО часто передается в зашифрованном виде, соответственно частота байт выравнивается, тем самым увеличивая энтропию.

- 35 В описываемом решении может применяться метод скользящего окна для представления исполняемого файла в виде ряда энтропийных мер. Энтропия окна

рассчитывается и после подсчетов в целях характеристики данного ряда рассматривается ряд статистических мер, например, таких как: математическое ожидание, дисперсия, квантиль, перцентиль. Таким образом, например, если энтропия близка к 1, то есть встречаемость различных символов примерно одинаковая, то код обфусцирован и/или зашифрован.

Гистограммы длин строк, причем из исполняемых файлов извлекаются все ASCII строки и фиксируются их длины, поскольку при данном извлечении в выборку попадает много «мусора», то использование полученных строк в качестве параметров является не совсем корректно, поэтому в описываемом решении используются гистограммы распределения длин строк.

Далее на этапе 113 дизассемблируют исполняемый файл и далее, посредством модуля извлечения данных, из него извлекают, по меньшей мере, следующие параметры для анализа.

Символы, причем под символами понимается частота появления по меньшей мере следующих символов: -, +, *,], [, ?, @, поскольку они характерны для кода, который пытается обойти систему обнаружения, например выполняя косвенные вызовы или загрузку динамической библиотеки.

Регистры, так как большинство названий регистров характеризуют свое предназначение, но в некоторых вредоносных программах, для усложнения анализа происходит их переименование. Поэтому в описываемом решении считается, что частота использования неизвестных регистров может быть полезна для определения степени вредоносности ПО. Кроме того, для дальнейшего анализа вводятся еще такие параметры: количество известных регистров, количество неизвестных (переименованных) регистров, отношение количества известных регистров к неизвестным.

Операционные коды (опкоды), причем наиболее часто использующиеся вредоносным ПО выявляются заранее, а также заранее определяется частота их появлений в самом коде. Так как список всех инструкций достаточно велик, то для анализа берется наиболее популярные опкоды, сохраненные во внутренней базе данных системы.

Частота обращения к системным интерфейсам (Windows APIs), причем для анализа можно также использовать частоту обращений к Windows APIs, но поскольку количество подобных интерфейсов слишком велико, то для повышения эффективности отобраны только наиболее характерные для вредоносного ПО и сохранены во внутренней базе данных системы.

Секции исполняемого PE файла, который состоит из нескольких заранее определенных секций таких как .text, .data, .bss, .rdata, .edata, .idata, .rsrc, .tls и .reloc.

Например, в таблице 1 представлены характеристики этих секций, которые используются при анализе.

Таблица 1.

Название параметра	Значение
bss_lines_number	количество строк в .bss секции
data_lines_number	количество строк в .data секции
edata_lines_number	количество строк в .edata секции
idata_lines_number	количество строк в .idata секции
rdata_lines_number	количество строк в .rdata секции
rsrc_lines_number	количество строк в .rsrc секции
text_lines_number	количество строк в .text секции
tls_lines_number	количество строк в .tls секции
reloc_lines_number	количество строк .reloc в секции
known_sections_number	общее количество секций
unknown_sections_number	общее количество неизвестных секций
known_sections_pr	доля известных секций от общего количества
unknown_sections_pr	доля неизвестных секций от общего количества
.text_pr	доля .text секции от всего файла
.data_pr	доля .data секции от всего файла
.bss_pr	доля .bss секции от всего файла
.rdata_pr	доля rdata секции от всего файла
.edata_pr	доля .edata секции от всего файла
.idata_pr	доля .idata секции от всего файла
.rsrc_pr	доля .rsrc секции от всего файла
.tls_pr	доля .tls секции от всего файла
.reloc_pr	доля .reloc секции от всего файла

5

Метаданные, а именно после операции дизассемблирования извлекаются размер получившегося файла и количество строк кода.

Далее на этапе 114 статистическим методом определяют все параметры, характерные вредоносным файлам и/или наоборот не вредоносным.

10

Например, возможно осуществить проверку на наличие контрольной суммы, как параметра, причем если он отсутствует - то подобное характерно для вредоносного

программного обеспечения; или если доля неизвестных секций от общего количества близка к 1, то очень много обфускации в коде, что характеризует вредоносное программное обеспечение или если доля известных секций близка к 0, то все более-менее стандартно и такое характерно легитимному ПО, или очень частый вызов определенных API характерен вредоносному ПО.

Кроме того, на этапе 115 из дизассемблированного вида файла получают по меньшей мере первый граф потоков (Control Flow Graph) и второй граф потоков (Data Flow Graph).

Причем первый граф потока (Control Flow Graph), представляет собой множество всех возможных путей исполнения программы, представленное в виде графа, причем использование графа потока управления в статическом анализаторе кода обусловлено его способностью быстро и оптимально находить уникальные пути исполнения.

Причем второй граф потоков (Data Flow Graph) служит для представления зависимостей одних переменных от других с учетом их области видимости и контекста исполнения, поэтому анализ потока данных напрямую связан с анализом потока управления, так как знания о контексте выполнения берутся оттуда.

Кроме того, стоит также отметить, что первый граф потоков (Control Flow Graph) и второй граф потоков (Data Flow Graph) получают посредством построения конечного автомата по всевозможным вариантам последовательности системных вызовов программы. Таким образом, алгоритм фиксирует циклические и ветвящиеся структуры программы, позволяя распознавать вариации поведения и выявлять аномалии. Способность фокусироваться на поведении программ способствует более коротким периодам обучения системы и меньшему количеству ложных срабатываний, а также меньшим требованиям к памяти. Алгоритм обнаружения аномалий, который использован в описываемом решении, основан на представлении последовательности системных вызовов в виде конечного автомата (FSA).

После способ переходит на этап 116, на котором все извлеченные параметры делятся на числовые и строковые и объединяются в первый и второй вектор признаков соответственно.

Таким образом, все числовые параметры, например, энтропия секций, размер каждой секции, отношение количества известных секций к неизвестным, доля .text секций от всего файла, количество строк в каждой секции, частота встречаемости символов обфусцирующих код (-, +, *,], [, ?, @) и т.п., записываются в вектор содержащий float значения (то есть числа с плавающей точкой).

Выглядит числовой вектор, следующим образом: у каждой ячейки есть свое назначение, то есть она отвечает за изначально заданный атрибут.

Например:

1 ячейка - known_sections_number = 10;

2 ячейка - unknown_sections_pr = 0.93 (доля неизвестных секций от общего количества);

5 3 ячейка - some_section_lines_number = 505 (количество строк в какой-то секции);

...

n-1 ячейка - lines_number = 5623 (количество строк кода в дизассемблированном файле);

n ячейка - file_size = 10893 (размер файла в байтах).

10 При этом стоит отметить, что длина вектора зависит от количества разнообразных признаков, найденных на этапе извлечения признаков.

Составленный вектор подается в бинарный первый классификатор (например, Random Forest), выход которого – вероятность вредоносности файла (0 – чистый файл, 1 – вредоносный).

15 Стоит отметить, что все строковые параметры, например, названия экспортируемых функций, данные из каталога ресурсов, названия импортируемых DLL и функций, где были использованы эти библиотеки и т.п., записываются в вектор, содержащий булевы значения, то есть встретились данные или нет.

Выглядит строковый вектор следующим образом:

20 имя ячейки – извлеченные данные;

значение ячейки - True или False (встретились или нет).

Далее после этапа извлечения всевозможных строковых данных следующим образом создается вектор признаков. Так, все найденные строковые атрибуты передаются в кодировщик. После этого данные преобразуются в вектор, состоящий из 0 и 1, булевый вектор (bool), для уменьшения требуемой памяти хранения.

25 Таким образом, на описываемом этапе получают первый и второй вектор признаков.

После завершения этапа 116 система переходит на этап 120, на котором создается ансамбль классификаторов.

Для создания ансамбля классификаторов, сначала обучают первый классификатор
30 на основе первого вектора признаков, второй классификатор на основе второго вектора признаков, третий классификатор на основе первого потока данных, четвертый классификатор на основе второго потока данных.

Стоит отметить, что в дополнительном варианте осуществления в ансамбль классификаторов может быть добавлен пятый классификатор, ориентированный на анализ
35 по n-граммам опкодов.

Стоит отметить, что каждый из классификаторов может представлять собой, например, классификаторы типа RandomForest и/или SVM.

Стоит отметить, что каждый из классификаторов в ансамбле классификаторов дополнительно характеризуется решающим приоритетом, причём решающий приоритет
5 определяется на основе точности (accuracy) работы классификатора на обоих классах (вредоносное и чистое ПО) на тестовой выборке.

Кроме того, основой проверки является тестовая выборка в которой проставлено соответствие между объектами и их классами. Для оценки используются следующие метрики: точность, полнота и F- мера. А именно используется: точность системы в
10 пределах класса – это доля объектов, действительно принадлежащих данному классу относительно всех объектов которые система отнесла к этому классу; полнота системы – это доля найденных классификатором объектов, принадлежащих классу относительно всех документов этого класса в тестовой выборке и F – мера, которая представляет собой гармоническое среднее между точностью и полнотой, причем она стремится к нулю, если
15 точность или полнота стремится к нулю.

Далее будет представлен подробный пример статического анализа на основе диаграмм первого и второго потока данных.

Стоит отметить, что третий классификатор обучается на основе первого потока. При чем ниже представлено подробное описание анализа исполняемых файлов на основе
20 первого потока (графа потока управления) (см. фиг. 3). На фиг. 3 изображена блок схема анализа файла на основе графа потока управления, в котором: дизассемблируют принятые исполняемые файлы (113), выделяют первый поток (граф потока управления) (115), каждому найденному параметру присваивается некое число (статистический анализ, например, насколько часто встречается данный параметр в обучающей выборке), далее
25 опираясь на которое выбирается только наиболее характерные параметры для обучения третьего классификатора. На фиг. 3 сплошной линией отмечены обучающие операции, а прерывистой – рабочие операции.

Граф потока управления (Control Flow Graph) – множество всех путей исполнения программы, представленное в виде графа.

30 На этапе 113 дизассемблируют файл, так как прежде чем анализировать PE-файл, необходимо его дизассемблировать (например, с помощью IDA Pro) и при необходимости распаковать (если был запакован каким-либо упаковщиком). Распаковывать можно при помощи, например, стандартного упаковщика исполняемых файлов UPX.

Далее необходимо произвести построение графа потока управления (CFG) (этап
35 115). В CFG каждый узел (вершина) графа соответствует базовому блоку -

прямолинейному участку кода, не содержащему в себе ни операций передачи управления, ни точек, на которые управление передается из других частей программы. Первая инструкция в блоке – точка на которую совершается переход, последняя инструкция – инструкция перехода в другой блок.

- 5 Стоит отметить, что граф является направленным. Направленные дуги используются в графе для представления инструкций перехода. Также, в большинстве реализаций добавлено два специализированных блока:

входной блок, через который управление входит в граф;

выходной блок, который завершает все пути в данном графе.

- 10 Для извлечения графа потока управления из дизассемблированного PE-файла написан скрипт на Python, который использует стандартные API, например, IDA Pro для извлечения информации о каждом блоке:

$$ClassTreeNode \left\{ \begin{array}{l} BlockID(Numeric) \\ StartAddr(Hex) \\ EndAddr(Hex) \\ LeftChild(Pointer) \\ RightChild(Pointer) \end{array} \right\} \quad (1),$$

- где BlockID - уникальный id блока, StartAddr - стартовый адрес, EndAddr - конечный
15 адрес, LeftChild - все ассемблерные инструкции, содержащиеся в базовом блоке, RightChild - блоки-наследники (их уникальные id).

После извлечения информации о базовом блоке создается граф для отображения потока управления, при этом избегают циклов, т.к. рассмотрение их приведет к бесконечно длинным путям выполнения.

- 20 Далее необходимо представить граф в виде последовательности вызовов опкодов. Для этого используется модифицированный алгоритм обхода графа в глубину с добавлением проверки на циклы и их отбрасывания по мере обнаружения:

Input : $Start_{Node}(Basic\ BlockID)$
Output : $ArrayOfExecutionTrace(Basic\ BlockIDs)$
Define Arrays : $Stack[], Curr_{ExecPath}[], Total_{ExecPath}[]$

```

Counter = 0
Push StartNode on Stack
While Stack != Empty do
Pop Stack
CurrExecPath = CurrExecPath  $\cup$  Basic BlockID
Counter = Counter + 1
If Exists(RightChild) and RightChild  $\notin$  CurrExecPath
Push RightChild on Stack
PathLen = PathLen + Counter
endif
If NotExists(LeftChild) and LeftChild  $\in$  CurrExecPath
TotalExecPath = TotalExecPath + CurrExecPath
Delete_nodes_from CurrExecPath till_last_added RightChild
Adjust_Counter
Elseif Exists(LeftChild) and LeftChild  $\notin$  CurrExecPath
Push LeftChild on Stack
endif
endwhile
Return TotalExecPath

```

В результате работы алгоритма получается последовательность id блоков, то есть представление графа в виде последовательности вершин при обходе в глубину.

- 5 Далее необходимо пройти по каждому блоку в порядке его выполнения, извлекая опкоды и отбрасывая операнды при них. В итоге получается профиль исполняемого файла, представленный последовательностью опкодов, как показано ниже.

```

'xor', 'inc', 'mov', 'cmp', 'jz', 'mov', 'test', 'jnz', 'mov', 'push', 'push', 'call', 'pop', 'pop',
'pop', 'mov', 'test', 'jnz', 'push', 'push', 'call', 'cmp', 'jnz', 'mov', 'mov', 'mov', 'cmp',
, 'call', 'pop', 'test', 'jnz', 'inc', 'jmp', 'call', 'push', 'push', 'call', 'xor', 'mov', 'mov', 'le
, 'push', 'push', 'call', 'test', 'jnz', 'xor', 'inc', 'mov', 'cmp', 'jz', 'mov', 'test', 'jnz', 'mov'
'cmp', 'jnz', 'push', 'push', 'call', 'pop', 'pop', 'mov', 'test', 'jnz', 'push', 'push', 'call', 'ci
, 'jz', 'cmp', 'jnz', 'cmp', 'jz', 'movzx', 'push', 'call', 'pop', 'test', 'jnz', 'inc', 'mov', 'jmp',

```

- 10 Далее проводится анализ по триграммам опкодов. Таким образом имеем профиль на каждый исполняемый файл и теперь можно анализировать на основании извлеченных 3-грамм операционных кодов.

```

'cmp jbe test', 'jbe test jz', 'test jz cmp', 'jz cmp jnz', 'cmp jnz mov', 'jnz mov test', 'mov test jz', 'test
'test jnz push', 'jnz push pop', 'push pop push', 'pop push push', 'push push push', 'push push push', 'push pus
jz', 'cmp jnz cmp', 'jnz cmp jnz', 'cmp jnz mov', 'jnz mov mov', 'mov mov jmp', 'mov jmp call', 'jmp call ret',
, 'push push call', 'push call xor', 'call xor mov', 'xor mov mov', 'mov mov lea', 'mov lea push', 'lea push call

```

Каждая такая триграмма впоследствии будет являться параметром в нашей системе.

Перед тем как переходить к классификации, данные документы (профили в виде триграмм) должны быть сконвертированы из текстовой формы в векторное представление.

- 15 Исходя из набора профилей исполняемых файлов для обучения системы, составим матрицу, где каждая строка — это отдельный документ, а каждая колонка — уникальная 3-

грамма, найденная по мере составления всех профилей. Таким образом число колонок в матрице – число уникальных параметров (т.е. 3-грамм) извлеченных из документов.

После выполнения предыдущего действия создается третий классификатор ансамбля классификаторов. При этом для уменьшения количества признаков и отбора
5 наиболее характерных будем пользоваться, например, стандартным алгоритмом tf-idf, который нужен для оценки важности слова в контексте документа (профиль одного РЕ-файла), являющегося частью коллекции документов (обучающая выборка).

После того как получили уменьшенный вектор признаков, его можно подавать в классификатор, например, Random Forest Tree Classifier. На этом подготовительный этап
10 для анализа на графе первого потока завершается и способ переходит к этапу 120.

Стоит отметить, что четвертый классификатор обучается на основе второго потока данных (Data Flow Graph). Причем ниже представлено подробное описание анализа исполняемых файлов на основе потоков данных.

Анализ потока данных – это метод сбора информации о возможном наборе значений,
15 вычисленных в различных точках компьютерной программы, который определяет, какие данные проходят через программу.

Кроме того, способ выполнения анализа потока данных, например, может состоять в том, чтобы задавать уравнения потока данных для каждого узла графа потока управления (CFG) и решать их путем многократного вычисления выхода из входа и преобразований с
20 данными локально в каждом узле, пока вся система не стабилизируется.

Для анализа потока данных необходимо решить задачу по определению точек в программе, где переменная была определена, причем каждое присвоение – это определение, изначально и до какой точки она использовалась, пока ее не перезаписали или не удалили.

Таким образом, переменная d достигает точки p , если существует путь от точки, непосредственно следующей за d , до точки p , такой, что d не перезаписывается вдоль этого
25 пути. Каждое определенное в программе или полученное при входе в программу значение, перемещается с множеством других переменных между блоками графа потока управления (фиг. 4a-4b).

Так, на фиг. 4a и 4b представлен пример описанной выше задачи по определению
30 точек в программе. Причем под B0-B2 понимают блоки программы, D (definition)-определение новой переменной, x, y – значения переменных, p – это блок, в котором d исчезает (то есть где-то определили переменную, далее переменная попадает из in в out по блокам, а затем попадаем в блок p , где удаляется), а f (transfer function of the block) – это функция, которая работает над входом in, трансформируя его в выход out.

35 Далее, определяем общий вид уравнения для каждого блока :

$$out[b] = Gen[b] \cup (in[b] - Kill[b]) \quad (2)$$

причем где:

Gen[b] – множество генерируемых в блоке значений;

In[b] – входное множество переменных в блок b;

5 Kill[b] – множество убитых переменных в блоке b;

Out[b] – выходное множество значений.

При этом, $in[b] = out[p_1] \cup out[p_2] \cup \dots \cup out[p_n]$, где b – каждый блок, $p_1 \dots p_n$ – все предшествующие блоки, ведущие в данный.

Примерный алгоритм для получения всех вход-выходов данных в блоках CFG может
10 выглядеть так:

```
// Boundary condition
out[Entry] = ∅

// Initialization for iterative algorithm
For each basic block B other than Entry
  out[B] = ∅

// iterate
While (Changes to any out[] occur) {
  For each basic block B other than Entry {
    in[B] = ∪ (out[p]), for all predecessors p of B
    out[B] = fB(in[B])    // out[B]=gen[B]∪(in[B]-kill[B])
  }
```

После аналогичного алгоритма прохода в глубину с удалением циклов, получают вектор из номеров блоков в графе CFG. К каждому блоку относятся 2 множества переменных (in и out). Получают вектора входных и выходных множеств переменных, в
15 каждой ячейке которых содержится множество входных и выходных переменных каждого блока соответственно. Так как анализ самих переменных не является информативным, по причине большого разнообразия переменных, рационально опираться на количественные изменения наборов переменных в каждом блоке.

Так, например, многие типы вредоносных программ реплицируют себя, заражая
20 другие двоичные файлы или вводя код в другие системные процессы. Это часто приводит к почти равномерному распределению объемов данных исходящих ребер соответствующих узлов графа.

Таким образом, из получившихся векторов множеств можно составить более информативный вектор, в каждой ячейке которого содержится разность мощностей двух
25 множеств: входных и выходных переменных соответствующего блока.

Далее используя ряд статистических мер (стандартизованная оценка, стандартное отклонение, математическое ожидание, интерквантильное расстояние) над полученным вектором, а также посчитав оконную энтропию данного вектора и применив к получившемуся вектору те же статистические меры, можно получить параметры, необходимые для подачи на анализ классификатору по второму потоку.

На этом подготовительный этап для анализа на графе второго потока завершается и способ переходит к этапу 120.

После завершения подготовительного этапа, способ переходит к рабочему этапу 130. Так на этапе 131 получают, по меньшей мере один исполняемый файл для анализа. Посредством модуля извлечения данных (S10), из исполняемого файла извлекают вышеуказанные данные, далее на этапе 132, запускают обученный ансамбль классификаторов, для выявления вредоносных исполняемых файлов; и на этапе 133, выводят результат анализа.

Стоит отметить, что решение ансамбля классификаторов может быть:

- бинарным (вредоносный/нет);
- вероятностным (если вероятность более 50%, то файл считается вредоносным).

Кроме того, стоит отметить, что в дополнительных вариантах реализации пользователю может быть выведен отчет об обнаруженных вредоносных файлах.

На Фиг. 5 далее будет представлена общая схема вычислительного устройства (500), обеспечивающего обработку данных, необходимую для реализации заявленного решения.

В общем случае устройство (500) содержит такие компоненты, как: один или более процессоров (501), по меньшей мере одну память (502), средство хранения данных (503), интерфейсы ввода/вывода (504), средство В/В (505), средства сетевого взаимодействия (506).

Процессор (501) устройства выполняет основные вычислительные операции, необходимые для функционирования устройства (500) или функциональности одного или более его компонентов. Процессор (501) исполняет необходимые машиночитаемые команды, содержащиеся в оперативной памяти (502).

Память (502), как правило, выполнена в виде ОЗУ и содержит необходимую программную логику, обеспечивающую требуемый функционал.

Средство хранения данных (503) может выполняться в виде HDD, SSD дисков, рейд массива, сетевого хранилища, флэш-памяти, оптических накопителей информации (CD, DVD, MD, Blue-Ray дисков) и т.п. Средство (503) позволяет выполнять долгосрочное хранение различного вида информации, например, вышеупомянутых файлов с наборами

данных пользователей, базы данных, содержащих записи измеренных для каждого пользователя временных интервалов, идентификаторов пользователей и т.п.

Интерфейсы (504) представляют собой стандартные средства для подключения и работы с серверной частью, например, USB, RS232, RJ45, LPT, COM, HDMI, PS/2, Lightning, FireWire и т.п.

Выбор интерфейсов (504) зависит от конкретного исполнения устройства (500), которое может представлять собой персональный компьютер, мейнфрейм, серверный кластер, тонкий клиент, смартфон, ноутбук и т.п.

В качестве средств В/В данных (505) в любом воплощении системы, реализующей описываемый способ, должна использоваться клавиатура. Аппаратное исполнение клавиатуры может быть любым известным: это может быть, как встроенная клавиатура, используемая на ноутбуке или нетбуке, так и обособленное устройство, подключенное к настольному компьютеру, серверу или иному компьютерному устройству. Подключение при этом может быть, как проводным, при котором соединительный кабель клавиатуры подключен к порту PS/2 или USB, расположенному на системном блоке настольного компьютера, так и беспроводным, при котором клавиатура осуществляет обмен данными по каналу беспроводной связи, например, радиоканалу, с базовой станцией, которая, в свою очередь, непосредственно подключена к системному блоку, например, к одному из USB-портов. Помимо клавиатуры, в составе средств В/В данных также может использоваться: джойстик, дисплей (сенсорный дисплей), проектор, тачпад, манипулятор мышь, трекбол, световое перо, динамики, микрофон и т.п.

Средства сетевого взаимодействия (506) выбираются из устройства, обеспечивающий сетевой прием и передачу данных, например, Ethernet карту, WLAN/Wi-Fi модуль, Bluetooth модуль, BLE модуль, NFC модуль, IrDa, RFID модуль, GSM модем и т.п. С помощью средств (505) обеспечивается организация обмена данными по проводному или беспроводному каналу передачи данных, например, WAN, PAN, ЛВС (LAN), Интранет, Интернет, WLAN, WMAN или GSM.

Компоненты устройства (500) сопряжены посредством общей шины передачи данных (510).

В настоящих материалах заявки было представлено предпочтительное раскрытие осуществление заявленного технического решения, которое не должно использоваться как ограничивающее иные, частные воплощения его реализации, которые не выходят за рамки испрашиваемого объема правовой охраны и являются очевидными для специалистов в соответствующей области техники.

Формула

- 5 1. Компьютерно-реализуемый способ выявления вредоносных файлов в
неизолированной среде, содержит:
- подготовительный этап, на котором:
- формируют подборку файлов, которая содержит по меньшей мере один
10 вредоносный исполняемый файл и по меньшей мере один не вредоносный исполняемый
файл;
- анализирует по меньшей мере один исполняемый файл, при этом:
- извлекают данные из бинарного и дизассемблированного видов
исполняемого файла на основе которых создаются параметры для дальнейшего
обучения классификатора, причем дополнительно статистическим методом
15 определяют параметры, характерные вредоносным файлам и/или наоборот не
вредоносным;
- причем дополнительно извлекают первый и второй граф потоков;
- на основе полученных параметров строятся первый и второй вектор
признаков;
- 20 создают ансамбль классификаторов из: первого обученного классификатора
на основе первого вектора признаков, второго обученного классификатора на основе
второго вектора признаков; третьего классификатора, обученного на основе первого
графа потоков, четвертого классификатора, обученного на втором графе потоков
причем для каждого классификатора заранее определяют решающий
25 приоритет;
- рабочий этап, на котором:
- получают, по меньшей мере один исполняемый файл;
- запускают обученный на подготовительном этапе ансамбль классификаторов,
для выявления вредоносных исполняемых файлов;
- 30 выводят результат анализа.
2. Компьютерно-реализуемый способ по п.1, причем в котором первый вектор
признаков содержит все числовые параметры, полученные при анализе, причем в котором
второй вектор содержит все строковые параметры, полученные при анализе.
3. Компьютерно-реализуемый способ по п.1, отличающийся тем, что извлекают, по
35 меньшей мере, следующие свойства файла: первый граф потоков (Control Flow Graph,

второй граф потоков (Data Flow Graph) , метаданные (Metadata), таблицу импорта (Import Table), таблицу экспорта (Export table), гистограмму энтропии (Byte/Entropy Histogram).

4. Компьютерно- реализуемый способ по п.1, отличающийся тем, что из по меньшей мере одного исполняемого файла бинарного вида извлекают, по меньшей мере, следующие
5 параметры: N-граммы байт; информация из полей исполнимого файла; энтропия секций файла; метаданные; гистограммы распределения длин строк.

5. Компьютерно- реализуемый способ по п.3, в котором информация из полей по меньшей мере одного исполняемого файла бинарного вида, содержит по меньшей мере: количество секций в файле, размер заголовков, размер кода, наличие цифровой подписи,
10 наличие импортируемых dll и функций, где были использованы эти библиотеки, название экспортируемых функций, данные из каталога ресурсов, версию сборщика.

6. Компьютерно- реализуемый способ по п. 3, в котором энтропия, извлекаемая из полей по меньшей мере одного исполняемого файла бинарного вида, рассчитывается с помощью применения метода скользящего окна с последующим применением ряда
15 статистических мер для характеристики полученного вектора энтропийных мер.

7. Компьютерно- реализуемый способ по п. 3, в котором метаданные, извлекаемые из заголовка по меньшей мере одного бинарного исполняемого файла, содержат: количество секций в файле, размер секций, наличие контрольной суммы, размер файла, время создания, версия операционной системы, версия сборщика.

8. Компьютерно- реализуемый способ по п.1, отличающийся тем, что из по меньшей мере одного дизассемблированного исполняемого файла извлекают, по меньшей мере, следующие параметры: символы; регистры; операционные коды; частота обращения к системным интерфейсам (Windows APIs); секции; метаданные.
20

9. Компьютерно- реализуемый способ по п. 7, в котором дополнительно определяют частоту появления символов, извлекаемых из по меньшей мере одного дизассемблированного исполняемого файла, причем символы могут представлять по меньшей мере: “-“, “+”, “*”, “]”, “[”, “?”, “@”.
25

10. Компьютерно- реализуемый способ по п. 7, в котором дополнительно вычисляют отношение количества известных регистров, извлекаемых из по меньшей мере одного дизассемблированного исполняемого файла, к неизвестным регистрам, извлекаемым из дизассемблированного файла.
30

11. Компьютерно- реализуемый способ по п. 7, отличающийся тем, что секции, извлекаемые из по меньшей мере одного дизассемблированного исполняемого файла, определены заранее и представляют собой: .text, .data, .bss, .rdata, .edata, .idata, .rsrc, .tls и
35 .reloc.

12. Компьютерно- реализуемый способ по п. 7, отличающийся тем, что метаданные, извлекаемых из по меньшей мере одного дизассемблированного исполняемого файла, представляют собой размер файла и количество строк кода.

5 13. Компьютерно- реализуемый способ по п.1, отличающийся тем, что результат анализа может быть бинарным или вероятностным.

14. Компьютерно- реализуемый способ по п. 1, в котором решающий приоритет каждого из ансамбля классификаторов определяется на основе обучения на тестовой выборке, причем для оценки используются по меньшей мере: точность, полнота, F-мера.

10 15. Система выявления вредоносных файлов в неизолированной среде по меньшей мере содержит:

модуль извлечения данных, выполненный с возможностью извлечения данных, по меньшей мере из одного исполняемого файла;

вычислительное устройство, выполненное с возможностью осуществления способа выявления вредоносных файлов в неизолированной среде по пп. 1 -14;

15 модуль вывода.

20

25

30

35