**Лекция 10.** [**Обзор существующих методов идентификации программного**](file:///C:\Users\721114400623\Downloads\Способы-идентификации-вредоносного-программного-обеспечения..doc#_Toc88448) [**обеспечения**](file:///C:\Users\721114400623\Downloads\Способы-идентификации-вредоносного-программного-обеспечения..doc#_Toc88449)

[Методы идентификации, основанные на статическом характере анализа характеристик программного обеспечения](file:///C:\Users\721114400623\Downloads\Способы-идентификации-вредоносного-программного-обеспечения..doc#_Toc88450). [Методы идентификации, основанные на динамическом характере сбора характеристик программного обеспечения](file:///C:\Users\721114400623\Downloads\Способы-идентификации-вредоносного-программного-обеспечения..doc#_Toc88451). [Методы идентификации, основанные на сборе информации путем обращения к встроенным функциям операционной системы или задействования собственного программного агента](file:///C:\Users\721114400623\Downloads\Способы-идентификации-вредоносного-программного-обеспечения..doc#_Toc88452).

В задаче идентификации объект *o* ∈ *O* (где *O* – множество объектов) описывается при помощи некоторого выделенного признакового пространства *f*1, *f*2,…, *fn* (где *i* = 1, 2,…, *n* – число градаций признака и значение для *fn*∈ *Df*– множеству допустимых значений признака), способного охарактеризовать основное свойство объекта *p(o)*, т.е. его класс, и записывается в виде вектора значений признака

*o* = (*f*1(*o*), *f*2(*o*) ,…, *fn*(*o*)),

именуемого в дальнейшем сигнатурой программы. Предполагается, что существует функциональная связь между признаками *f*1, *f*2, …, *fn*и основным свойством объекта *p(o),* позволяющая определить значение свойства *p(o)*. В данном исследовании в качестве основного свойства выступает имя программы, полученное по ранее составленной информации о других объектах *o*1,*o*2,…,*on*(называемых обучающей выборкой) и их свойствах *p*(*o*1), *p*(*o*2) ,…, *p*(*on*).

Идентификация происходит путем применения выбранного алгоритма сравнения двух векторов значений признака объектов: *o* – идентифицируемого и *o’* – эталонного.

Различают следующие типы признаков:

− количественные признаки – признаки, измеренные в определенных метрических (числовых) шкалах;

− качественные – признаки, измеренные в неметрических шкалах и используемые для выражения терминов и понятий, не имеющих цифровых значений.

Также, в зависимости от множества допустимых значений признака, выделяют следующие типы:

− бинарный – принимающий значение нуля или единицы, *Df*= {0,1};

− номинальный – где множество допустимых значений признака конечно, *|Df*| = *const*;

− порядковый – где множество допустимых значений признака конечно и упорядочено, *|Df*| = *const* и *Df*= {*df*| *dfj*≤ *dfj*+1, *j* = 1, …, *const*-1};

− количественный – где в качестве допустимых значений признака выступает множество действительных чисел *Df*∈ ℝ.

В свою очередь, представление признаков может носить разнообразный характер, в частности, являться перечислением признаков, векторным описанием файла, или же иметь графическое представление, записанное в древовидной или графовой модели. В данном пособии будут в основном рассмотрены номинальные (номинативные) признаки – это признаки, измеренные в неметрической шкале наименований, представляющие собой набор числовых характеристик.

На сегодняшний день существует ряд подходов к идентификации программного обеспечения на электронных носителях информации. Процесс распознавания (или идентификации) программы можно разделить на два этапа: первый - выделение признакового пространства и второй - применение метода распознавания программы.

Подходы к распознаванию программ могут быть основаны как на структурной характеристике файла, описывающей его содержимое, так и на семантической составляющей, характеризующей смысл анализируемого кода программы. К последним относится интеллектуальная обработка данных, в которую входят:

− формальные методы – оперирующие фиксированной моделью объекта и использующие предопределенные алгоритмы обработки;

− эвристические методы – способные, в некоторой степени, воспроизвести процесс познания человека; они направлены на выявление неизвестных объектов.

Методы интеллектуальной обработки данных, как правило, включают проверку «корректности» исполняемых файлов, вычисление статистического распределения инструкций процессора или энтропии кода, а также выявление последовательностей инструкций, характерных для вредоносных программ. В связи с этим использование данной группы методов в целях идентификации невредоносного программного обеспечения не представляется возможным (достаточно эффективным).

Ниже представлены наиболее распространенные подходы к задаче идентификации исполняемого файла.

*Таблица 1 – Подходы, используемые для идентификации программ*

|  |  |
| --- | --- |
| Способы сбора характеристик программы | Методы распознавания программ |
| − По *статическим* характеристиками рассматриваемого объекта  − По *динамическим* характеристиками рассматриваемого объекта  − Задействующие *встроенные функции операционной системы* или функциональность внедряемого программного агента | − Задание метрики схожести  − Статистический анализ  − Машинное обучение  − Эвристический анализ |

**Методы идентификации, основанные на статическом характере анализа характеристик программного обеспечения**

*Статические методы* оперируют двоичным образом программы, хранящемся на электронном носителе информации, будь то внешний жесткий диск или оперативная память компьютера. Данный метод подразумевает под собой работу с исполняемым файлом или его дампом памяти как с массивом побайтовой последовательности.

Статические методы, по способу сбора характеристик программы, являются наиболее распространенными. Данный класс методов подразделяется на *несигнатурные* (методы проверки целостности), такие как сравнение с полной копией данных, сравнение контрольных сумм, контроль CRC (Cyclic Redundancy Check), хеширование, имитовставка, цифровая подпись, и *сигнатурные* (методы сравнения признаковых последовательностей), такие как выделение «магического числа», сравнение характерных последовательностей байтов и др.

В средствах защиты информации активно используется сигнатурный (синтаксический) метод, заключающийся в сравнении байтовых последовательностей с эталонной сигнатурой.

Идентифицировать файл можно с разной степенью конкретности, так, самой простой идентификацией будет являться определение расширения файла по его сигнатуре, известной также как «магическое число», представляемое в виде строго определенного набора байт, как правило, внесенных в начало тела файла. Так, например, для ELF файла эта последовательность задана следующим набором байт – 7F 45 4C 46; а для EXE файлов – 4D 5A.

Несигнатурные методы также популярны в настоящее время, но сфера их применения ограничена, и ниже будут описаны их достоинства и недостатки применительно к задаче идентификации ПО.

Одним из первых и наиболее простых методов обеспечения целостности файла является разработка метода контрольной суммы. Первоначально под контрольной суммой понималось некоторое число, занимающее позицию младшего разряда байта и вычисляемое при помощи сложения остальных 7 бит. Сегодня под данное определение попадает огромное число разнообразных алгоритмов, некоторые из которых имеют широкое применение, а некоторые используются только для специфических задач. Самыми известными и популярными алгоритмами являются SHA-1, SHA-2, SHA-3, CRC и MD5, при этом стоит отметить, что последние два алгоритма применяются все реже, т.к. не способны обеспечить достаточный уровень стойкости к коллизиям.

Хеширование является простейшим подходом к сравнению файлов. При вычислении хеш-функции в качестве исходных данных принимается исходная двоичная последовательность файла, далее в зависимости от выбранного алгоритма она разбивается на блоки фиксированной длины с последующим вычислением значения хешфункции.

Достоинством хеш-функции является ее простота, скорость выполнения операции и размер получаемого значения хеш-функции, который всегда имеет одну и ту же длину. Однако, существенным недостатком в задаче сравнения двух файлов является то, что вычисляемое значение хеш-функции зависит от модификации, произведенной над файлом, при этом оно ни в коей мере не зависит от степени произведенного вмешательства над оригинальным файлом. Т.е. если сравнивать значение хеш-функции оригинального файла со значением хеш-функции оригинального файла с измененным хотя бы одним битом – эти значения будут совершенно различными.

Чтобы снизить влияние модификаций и сравнивать схожие файлы, был предложен подход с использованием контекстно-зависимого кусочного хеширования (Context Triggered Piecewise Hashing), который использует плавающее окно с фиксированным значением шага, таким образом, контекстно-зависимое кусочное хеширование представляет собой конкатенацию значений хеш-функций блоков файла. Данный подход использовался в работах. Модификации алгоритма выбора длины окна, шага и границ образуемых блоков приводят к существенному улучшению при идентификации схожих файлов, однако остается ограничение на количество возможных вносимых изменений в оригинальный файл.

Цифровая подпись в исполняемых файлах используется для верификации программы ее автором, позволяя идентифицировать издателя файла (подлинность файла) и определить, не подвергался ли он изменениям (целостность файла), тем самым гарантируя безопасность программы для

АС. Таким образом, если файл содержит некорректную цифровую подпись (или ее нет совсем), то это может означать, что данный файл опубликован ненадежным издателем или был изменен, однако существуют способы обхода проверки цифровой подписи и запуска недостоверного программного обеспечения.

**Методы идентификации, основанные на динамическом характере сбора характеристик программного обеспечения**

*Динамический* *анализ* основан на исполнении анализируемой программы. Отсутствие каких-либо предположений о ходе исполнения программы, а также проверка её в процессе выполнения (или сразу по окончании) является значимым преимуществом динамического анализа. Очевидным требованием, предъявляемым при реализации динамического анализа, является проведение динамического анализа с наименьшим влиянием на ход исполнения программы. При определённых условиях на детерминированность программы динамический анализ позволяет избежать проблемы ложных срабатываний.

Большинство современных вредоносных программ используют методы запутывания (обфускации), чтобы помешать статическому анализу двоичных файлов, в частности бинарные упаковщики. Поэтому динамический анализ таких вредоносных программ зачастую бывает намного эффективнее статического анализа. Мониторинг поведения программ в ходе их исполнения включает, например, сбор выполняемых программой бинарных операций и предлагает потенциально более глубокое понимание самого кода. Сегодня подходы такого типа имеют свои ограничения, например – может быть трудно вызвать определенное поведение программ, а некоторые из них могут требовать определенные условия окружающей среды, тем не менее, использование динамического анализа считается более эффективным по сравнению с использованием только лишь статических подходов. По этой причине, динамическому анализу программ уделяется большее внимание в научном сообществе. Системы анализа, такие как CWSandbox, Anubis, BitBlaze , AVG и Advanced Threat Protection, выполняют запуск образцов вредоносных программ в защищенной среде и отслеживают их поведение для анализа и разработки защитных механизмов.

Основными считаются подходы по:

− выявлению последовательности необычных путей исполнения кода программы или вызываемым системным функциям (API, system calls,

Windows native API);

− выявлению нанесенных повреждений окружающей среде в изолированной системе;

− отслеживанию попыток получения высоких привилегий;

− выявлению изменений параметров вызываемых функций;

− определению способов обработки данных программой;

− анализу трассировки машинных инструкций программы.

Динамический анализ является более эффективным в плане анализа программы, однако для задачи идентификации большого числа версий невредоносных программ его применение становится нецелесообразным, поскольку его главный минус связан с большими временными затратами, требуемыми на выполнение и тщательное исследование программ.

**Методы идентификации, основанные на сборе информации путем обращения к встроенным функциям операционной системы или задействования собственного программного агента**

В настоящее время множество компаний обращаются к использованию систем управления ИТ-активами (ITAM), которые представляют собой комплексные решения, нацеленные на физический учёт, финансовый контроль и соблюдение контрактных обязательств, связанных с ИТ-активами, на протяжении всего их жизненного цикла. Здесь под ИТ-активами подразумеваются все аппаратные и программные элементы ИТ-инфраструктуры, обеспечивающие деятельность бизнес-среды.

В свою очередь ITAM подразделяется на управление аппаратными активами (HAM), охватывающее управление материальными составляющими ИТ-инфраструктуры: пользовательские компьютеры, сервера, телефоны и т.д.; и на управление программными активами (SAM), охватывающее управление нематериальными составляющими ИТинфраструктуры: программное обеспечение, лицензии, версии, конечные точки инсталляции и т.д.

На рынке услуг представлено немало решений, позволяющих идентифицировать программные активы, управлять учетом, а также производить контроль их изменений и др. Из числа наиболее известных программных продуктов можно выделить:

1. Работающие на основе агента:

− Samanage – поддерживает работу с Windows, Linux, Mac ОС.

− Kaspersky Systems Management – поддерживает работу с Windows, Linux, Android, iOS.

1. Работающие без агента:

− Microsoft Assessment and Planning Toolkit – поддерживает работу с Windows ОС [46].

1. Работающие как с агентом, так и без него:

− AIDA64 – поддерживает работу с Windows, Linux, Android.

− Lansweeper – поддерживает работу с Windows, Linux, Mac ОС и устройствами, поддерживающими IP адресацию.

Агентом является программа, устанавливаемая на компьютер пользователя АС и ведущая скрытую деятельность по выполнению своих функций, в частности инвентаризации ПО. Недостатком данного подхода является необходимость установки агента на каждый исследуемый компьютер, однако преимуществом агента является возможность его установки на корпоративные ноутбуки и смартфоны, используемые пользователями за пределами внутренней сети организации. Таким образом, решения, работающие на основе агента, имеют самый высокий уровень детализации, могут быть использованы для вмешательства в компьютер пользователя. Использование подхода без установки агента не требует наличия подходящих для данного агента систем, его обновлений, не повышает производительные затраты ресурсов компьютера, а также имеет удобное централизованное управление. Однако уровень детализации получаемой информации ниже, чем при задействовании агента, а также появляется необходимость контроля того, что исследуемые активы доступны для их обнаружения.

Подход к оценке ресурсов с(без) агентом имеет свои преимущества и недостатки, особенно стоит учитывать число самих ресурсов в организации.

Однако отметим, что большая часть предоставляемой информации собирается посредством встроенных технологий инвентаризации программного и аппаратного окружений, операционной системы:

− Windows Management Instrumentation – является одним из базовых инструментариев централизованного управления и слежения за работой различных частей компьютерной инфраструктуры под управлением платформы Windows;

− Active Directory Domain Services – служба каталогов Windows, которая централизованно сохраняет все данные и настройки среды в базе данных;

− SMS Provider – поставщик инструментария управления Windows, назначающий доступ на чтение и запись к базе данных Configuration Manager на сайте сервера;

− lshw – утилита Linux, которая выводит полный список аппаратных компонентов системы вместе с информацией об устройствах;

− dpkg -l – утилита Linux, которая выводит полный список программных компонентов системы;

− и других технологий.

Недостатки такого подхода очевидны. Отсутствие должного уровня оценивания роли информационной безопасности в бизнес-процессах организации приводит к недооценке кадровой политики со стороны руководства при подборе ИТ персонала. В то же время пользователи АС имеют возможность достичь достаточного уровня компетенции в сфере компьютерных технологий, позволяющего им обходить установленные политики безопасности. Все это предоставляет возможность внесения изменений в конфигурационные данные устанавливаемого программного обеспечения.

Ниже на рисунках 1-3 приведены примеры по изменению версий программ (*ABBYY Lingvo* и *Htop*) в операционных системах Windows (версия: 10) и Linux (версия: Ubuntu) соответственно. Из них становятся очевидны пути обмана средств аудита программного обеспечения как для установленного ранее программного обеспечения, так и для устанавливаемого впервые.



Рис. 1 – Изменение версии программы ABBYY Lingvo x5 15.0.511 на ABBYY Lingvo x5 15.0.510

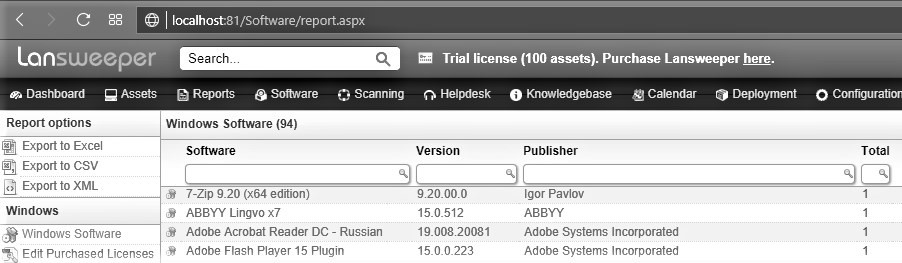


Рис. 2 – Изменение версии программы ABBYY Lingvo x5 15.0.511 на ABBYY Lingvo x7 15.0.512

Из рисунков 1 и 2 видно, что путем манипуляций в реестре операционной системы Windows версия программы *ABBYY Lingvo* была изменена дважды. Целью данного вмешательства являлось намеренное влияние на результаты выдачи запроса по инвентаризации программного обеспечения с помощью средств Windows Management Instrumentation (WMI) (результат выдачи представлен на рисунке 1) и распространенного программного средства Lansweeper (результат выдачи представлен на рисунке 2).

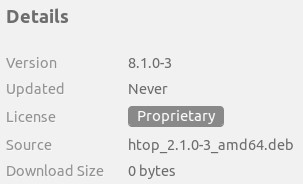
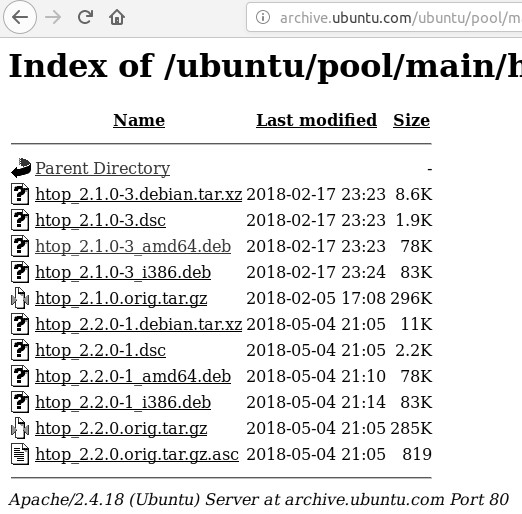
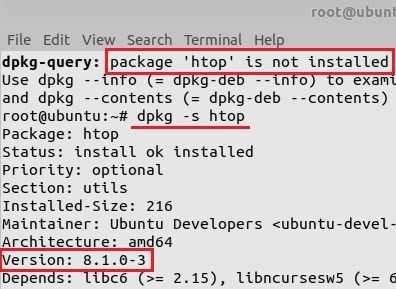


Рис. 3 – Установка измененной версии программы htop 8.1.0-3 вместо htop 2.1.0-3

На рисунке 3, в свою очередь, представлена установка программы *Htop* в операционной системе Ubuntu с заведомо измененной версией, исправленной путем вмешательства в конфигурационный файл устанавливаемого пакета \*.deb.

В данной главе был произведен обзор существующих методов идентификации программного обеспечения, из которого можно выделить ряд ограничений, не позволяющий использовать их в задаче идентификации версий программного обеспечения:

1) Для статического сбора характеристик файла:

− Методы, основанные на оценке целостности файла, не позволяют исследовать схожесть программ, не задействованных в обучающей выборке.

− Использование особенностей PE формата программ ОС Windows, отсутствующие в ELF-файлах.

− Большинство существующих подходов к классификации исполняемых файлов направлены на бинарное разделение тестовой выборки и выделение класса вредоносных программ.

− Часть подходов рассматривает мульти-классификацию с ограниченным набором классов.

− Работы же, направленные на мульти-классификацию с неограниченным набором классов, обладают низкой точностью идентификации исполняемых файлов.

2) Для динамического сбора характеристик файла:

− Использование данного типа подходов является нецелесообразным по причине задействования огромного числа ресурсов.

− Для выполнения программы бывает необходимо создавать особую среду исполнения.

1. Для сбора информации путем обращения к встроенным функциям операционной системы или задействования собственного программного агента:

− Главным недостатком является сам принцип получения информации с АС, где факт наличия способов изменения информации о программе был проиллюстрирован на рисунках 1-3.

Приведенный анализ методов идентификации различных версий исполняемых файлов позволяет сделать вывод, что идентификация на основе статического анализа характеристик программы и использование статистических и машинных алгоритмов классификации является довольно перспективным направлением деятельности в рамках задачи идентификации невредоносного ПО.

Можно сказать, что в современных условиях не существует решения, позволяющего с достаточной точностью производить идентификацию различных версий ПО в АС в задаче по выявлению факта нарушения пользователем установленных правил по несанкционированной установке ПО. Отсюда вытекала необходимость в проведении автором данного пособия исследования, направленного на достижение максимальной точности идентификации исполняемых файлов, путем совершенствования существующих подходов, поиска новых, ранее не применявшихся решений в области выделения признакового пространства файлов, методов идентификации, а также способов постобработки результатов.