УДК 004.056 + 004.738.5 + 004.491

АНАЛИЗ ВРЕДОНОСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

М.С. Уфимцев, И.С. Антясов

Исследованы проблемы и основные тенденции в области анализа вредоносного программного обеспечения. Рассмотрены методы получения информации о логике работы исследуемых программ посредством частичного восстановления исходных текстов либо с применением модели черного ящика путем воздействия на изучаемый объект и снятия основных показателей активности операционной системы.

Ключевые слова: анализ вредоносного программного обеспечения, дизассемблирование, информационная безопасность.

Определяющая все направления развития инфокоммуникационных технологий парадигма достаточно устойчиво связала почти все сферы деятельности человека с современным информационным миром, однако на данный момент рост значимости информационных технологий в деятельности человека вызывает ряд вопросов и формирует спектр актуальных проблем.

Обзор одной из проблемных областей нашел свое отражение в рамках статьи. На данный момент опасение вызывает стремительно развивающийся рынок вредоносного программного обеспечения (далее по тексту – вредоносное ПО). Анализ крупнейших атак с использованием вредоносного ПО позволил выделить основные тенденции:

- Высокий рост сложности вредоносных программных продуктов за счет применения относительно простых технологических решений. Примером может служить использование шифрования. Относительно простые для реализации алгоритмы гарантируют злоумышленнику при выполнении некоторых простых условий почти полную гарантию того, что зашифрованные им данные пользователь не сможет за приемлемое время восстановить.
- Увеличение числа группировок, имеющих необходимые навыки для создания вредоносного ПО. Данный факт объясняет численный рост количества вредоносных программ. Порог вхождения в данную область со временем упал до критического предела, что позволяет объяснить и спад качества изготавливаемой вредоносной продукции [1].
- Регулярно в сети появляется информация о том, что подавляющее большинство подобного программного обеспечения не детектируются средствами антивирусной защиты. Причины возникновения подобных ситуаций всегда различны, однако их можно классифицировать по следующим признакам:

1) новые модификации вредоносного ПО, сигнатуры которых отсутствуют в базах антивирусных продуктов. Подобное поведение антивирусов достаточно просто объяснить: каждая массовая вирусная атака не обходится без новой модифицированной версии вредоносного продукта. Должно пройти время, чтобы обновленная версия попала в базы антивирусов;

2) вредоносные программы средствами эвристического анализа признаются безопасными. Часто поведенческий модуль антивируса не распознает в действиях программы деструктивного подтекста.

Анализ представленных тенденций позволил нам сформировать ряд выводов:

- Антивирусная защита – необходимый компонент для построения любой информационной системы, который в совокупности с грамотно реализованными организационными мерами и комплексным подходом к конфигурированию всех аспектов системы, обеспечивает безопасность защищаемых ресурсов. Однако, текущая ситуация показала, что полностью доверять данным средствам нельзя. Возникает прямая потребность в формировании новой методологии анализа вредоносного ПО своими силами.
- Работы по самостоятельному изучению вредоносного ПО успешно ведутся. Информация по методам, примерам анализа и алгоритмам нейтрализации регулярно появляется в специализированной литературе, периодических изданиях, интернет-форумах, однако попыток систематизировать воедино подобную информацию и показать возможность самостоятельного изучения вредоносного ПО недостаточно.

Рассмотрены основные этапы анализа для широкого спектра вредоносных программ без привязки к конкретному языку программирования, компилятору, операционной системе, аппаратной платформе.

Основная проблема анализа вредоносного ПО заключается в получении объекта исследования. Однако, в подавляющем большинстве случаев задача сводится к загрузке файла программы (также это может быть скрипт, какой-либо файл с внедренным вредоносным кодом, эксплуатирующим уязвимость программы-просмотрщика, макрос, разработанный на макроязыке, встроенном, например, в пакет прикладного ПО Microsoft Office) из источника. Источником может являться письмо электронной почты с вложением, ссылка на страницу загрузки, инфицированный ранее файл, содержащий тем не менее тело вируса, способное на дальнейшие активные действия.

Далее необходимо выбрать среду, в которой будет проводиться исследование. Среда в данном контексте – это либо реальный компьютер, либо правильно сконфигурированная виртуальная машина, в которую загружены все необходимые объекты исследования, а также инструменты. Настройка параметров виртуальной машины сводится к максимальному ограничению взаимодействия её с системой-хостом, на которой она запуще-

на. Подразумевается отсутствие доступа к ресурсам системы-хоста (отсутствие общих папок, доступа к общим ресурсам и сервисам) и, если есть подозрения на то, что вирус распространяется по сети, максимально ограничить сетевое взаимодействие. Имеет смысл обеспечить запуск виртуальной машины в потенциально невосприимчивой к заражению изучаемым объектом операционной системе. Стоит отметить, что существует большое количество вирусов, способных детектировать свой запуск на виртуальной машине. В случае подобного обнаружения подключаются механизмы усложнения анализа или полностью маскируется деструктивная деятельность. Подобное поведение накладывает определенные условия при выборе среды.

При получении объекта исследования необходимо разобраться в том, что представляет собой файл вируса. Чаще всего это некоторый исполняемый файл, к анализу которого возможно сразу перейти (не рассмотрены случаи, в которых изначально исходный текст программы, например, на скрипт-языке или языке макросов). Большинство попыток разобраться в логике работы программы начинаются с загрузки соответствующего исполняемого файла в дизассемблер. Правильно сконфигурированная программа-дизассемблер позволяет в простейшем случае получить из машинного кода текст программы на языке ассемблера. С полученным текстом можно сразу начать работу по изучению логики работы программы, а также приступить к попыткам привести код в удобный читаемый вид. В данной области широкое признание получили так называемые интерактивные дизассемблеры, позволяющие вмешаться напрямую в процесс дизассемблирования. Исследователь в любой момент может снять дамп дизассемблируемого участка памяти, просмотреть строковые константы с данными, прокомментировать необходимые участки программы. Основная задача, с которой следует справиться на этом этапе – разделить данные программы и сам машинный код [2], постараться найти точку входа в программу. Под точкой входа в программу будем понимать адрес в оперативной памяти, содержащий первую команду программы.

Основные сложности данного этапа заключаются в том, что разработчики вредоносного ПО осведомлены об основных методах анализа и часто пытаются усложнить исследование программы. Основным и самым действенным методом на данный момент является обфускация исходного текста программы (в случаях со скрипт-языками), исполняемого кода, а также алгоритмов работы. Данный метод запутывает код, полностью сохраняя функциональность программы, однако значительно увеличивая трудоемкость анализа. Как правило, подобные действия совершаются с использованием особых компиляторов, эксплуатирующих неочевидные особенности языка и среды выполнения программы. Данная область защиты программ от исследования развивается с высокой скоростью, создаются слож-

нейшие программные комплексы, способные приводить код к абсолютно нечитаемому виду (методы слома графа и применения виртуализации), однако подобный подход с увеличением уровня обфускации также уменьшает производительность программы, в некоторых случаях при неумелом использовании снижает надежность. Часто доходит до того, что разработчик жертвует необходимым уровнем защиты в угоду работоспособности программы, ведь на определенном уровне усложнения программа может вовсе перестать работать или стать жестко привязанной к определенному компилятору или платформе. Обычно используется максимально разумный уровень воздействия на исполняемый код. Процесс обхода обфускации возможно решить программными методами. Существуют автоматические деобфускаторы, способные восстановить «шаблонно испорченный» код. На данном этапе всё зависит от опыта исследователя, который уже в ручном режиме будет трассировать программу, удалять или игнорировать некоторые куски кода, а также создавать свои правила обхода сложных мест и при необходимости переписывать необходимые фрагменты, отслеживать поведение программы при подаче различных входных параметров [3]. Данные действия подразумевают использование дополнительного инструмента – отладчика.

Следующим необходимым этапом, который чаще всего выполняется параллельно дизассемблированию, является загрузка исследуемого исполняемого файла в отладчик. Данный класс программ позволяет изучать код в процессе его выполнения. Возможности каждого отладчика сводятся к следующим: пошаговое выполнение программы, просмотр данных в регистрах, просмотр состояния стека, отслеживание передачи параметров в конкретные функции, детектирование вызовов конкретных процедур и установка точек останова в необходимые места, чтобы при повторных запусках программы сразу останавливаться на конкретном участке кода и заниматься его изучением [4]. На данном этапе уже следует найти опытным путем точку входа программы, а далее всё зависит от исследуемого объекта: существует большое количество возможных сценариев развития.

Часто дизассемблирование провести не удается, а трассировка в отладчике не позволяет за приемлемое время получить даже точку входа. Такие ситуации возникают преимущественно в тех случаях, когда программа запакована. Обычно в подобных случаях происходит запуск приложенного к коду распаковщика, который приступает к расшифровке (распаковке) программного кода. После завершения происходит прыжок к необходимой нам точке входа в программу, откуда уже начинается выполнение распакованного программного кода. На этом этапе можно снимать дамп памяти, позволяющий в некотором виде получить листинг программы. Обычно проблемы распаковки можно решить автоматическими распаковщиками, когда формат упаковки известен, либо вручную с созданием дампа, вос-

становлением таблиц импорта и заменой точки входа на необходимую (оригинальная точка входа, с которой бы начиналась работа программы в том случае, если бы она не была упакована) [5].

Стоит заметить, что на данном этапе мы получаем полный набор необходимых сведений и инструментов для успешного анализа кода вируса, но возникают случаи, когда полное восстановление логики работы вируса невозможно по тем или иным причинам, либо появляется необходимость в ускорении процесса исследования. Для подобных случаев существует неформальная методология, основанная на следующем предположении: вредоносная программа является некоторым черным ящиком, о внутреннем устройстве которого нет информации, однако имеется возможность подавать на вход ящика некоторые параметры и снимать определенный массив выходных данных, характеризующих реакцию операционной системы.

Утилиты, позволяющие отслеживать поведение процессов, изучать обращение процесса к диску, исследовать сетевую активность. В рамках данного этапа создаются снимки файловой системы, реестра, оперативной памяти до вмешательства вредоносной программы и сравнение со снимками после совершения деструктивных действий. Ко всему прочему, всегда имеется возможность запустить программы для создания лог-файлов, в которые будет записана любая активность изучаемого объекта операционной системы. Также возможен сбор статистики сетевой активности интересующих нас процессов. Полученная таким образом информация позволит определить, какие данные вредоносная программа принимает по сети. К тому же, в некоторых случаях появляется возможность увидеть, какие данные с зараженной машины передаются злоумышленнику (данные пользователей, собранные данные активности, данные кейлоггеров). Метод носит обзорный характер, но уже за короткое время позволяет получить список атакуемых ресурсов операционной системы.

Изучение взаимодействия программы с операционной системой средствами мониторов API-вызовов. Работа большого числа приложений (пример рассмотрен для операционной системы Windows) основана на системе динамически подгружаемых библиотек. Использование специальных программ позволяет перехватывать вызовы API-функций, посредством которых выполняется взаимодействие программ с операционной системой. В простейшем случае мы способны перехватить обращение к функции и получить результаты ее выполнения. Часто можно узнать также и входные параметры, а определенный класс утилит позволяет влиять на выполнение вызываемых API-функций. Данный метод позволяет наиболее быстро создать базовую схему работы программы, которой достаточно для понимания основных принципов функционирования.

Представленный в статье порядок и методика анализа описывают основные этапы исследования вредоносного программного обеспечения. Данный подход позволил выделить на каждом из этапов ряд проблем, затрудняющий анализ исследуемых объектов. В процессе описания проблемных областей приведены возможные пути решения. Рассмотренный материал обеспечивает базовое представление о вирусном анализе и при наличии определенных навыков позволяет приступить к глубокому изучению вредоносного ПО с целью проведения самостоятельных исследований в данной области.

Библиографический список

1. Касперски, К. Техника и философия хакерских атак / К. Касперски. – М.: СОЛОН-Р, 2004.
2. A Survey on Tools for Binary Code Analysis / Shengying Li // Computer Science Department Stony Brook University, 2004.
3. О методах деобфускации программ // Ш.Ф. Курмангалеев, К.Ю. Долгорукова, В.В. Савченко и др. // Труды Института системного программирования. – 2013. – Т. 24. – С. 145–160.
4. Молоков, К.А. Реализация основных алгоритмических конструкций средствами отладчика / К.А. Молоков // Вологдинские чтения. – 2010.
5. Искусство дизассемблирования / К. Касперски, Е. Рокко; пер. с англ. – СПб.: БХВ-Петербург, 2009.

К содержанию