МЕТОДЫ И АЛГОРИТМЫ ЗАЩИТЫ ИСПОЛНЯЕМОГО ПРОГРАММНОГО КОДА ОТ ДИНАМИЧЕСКОГО И СТАТИЧЕСКОГО АНАЛИЗА

Мусаев Ш.С

ГУП "Центр кибербезопасности", Узбекистан

Курёзов Р.Б

Степень магистра, Факультет кибербезопасности, Ташкентский университет информационных технологий имени Мухаммада аль-Хоразмий, Узбекистан

Аннотация: В последнее время системы обнаружения вторжений (IDS) были внедрены для эффективного защищенные сети. Использование нейронных сетей и машинного обучения для обнаружения и классификации вторжения являются мощными альтернативными решениями. В этой исследовательской работе оба метода Gradient спуск с импульсом (GDM) на основе обратного распространения (BP) и градиентный спуск с импульса и адаптивного усиления (GDM / AG) используются для обучения нейронные сети для работы как IDS. Чтобы проверить эффективность двух предложенных обучения, IDS на основе нейронной сети строится с использованием предложенного алгоритма обучения - ритмы. Эффективность обоих алгоритмов проверяется с точки зрения скорости сходимости к достичь системного обучения и затраченного времени обучения, используя различные настройки нейронной сети параметры. Результат показал, что алгоритм обучения BP на основе GDM/AG превосходит алгоритм обучения BP на основе GDM.

Ключевые слова Системы обнаружения вторжений (IDS) Нейронные сети (NN)

О Б Р А Т Н О Е Р А С П Р О С Т Р А Н Е Н И Е (В Р)

Абстракт: Основная идея статьи заключается в том, что методы динамического и статического анализа необходимо использовать в сочетании друг с другом для повышения эффективности анализа бинарного кода. В статье авторы вносят свой вклад в методы декомпиляции Д В О И Ч Н О Г О кода и динамического анализа выполнения. Результаты применяются к проблеме защиты программного обеспечения от несанкционированного обратного инжиниринга. Авторы использовали анализ основного алгоритма потока управления программой, чтобы запутать П Р О Г Р А М М У и защитить ее от исследований.

Ключевые слова: Динамический, статический анализ, поток управления, защита программного обеспечения, обфускация.

В в е д е н и е . Компании-разработчики программного обеспечения сталкиваются с серьезными рисками, связанными с безопасностью программного обеспечения, поскольку процесс разработки программного обеспечения всегда содержит ошибки и ошибки. В свою очередь, эти ошибки могут создать благоприятные условия для появления серьезных уязвимостей в ПО. Эксплуатация уязвимостей влияет не только на безопасность системы, использующей уязвимый продукт, но и на конкурентные преимущества компании, занимающейся разработкой программного обеспечения. Следует отметить, что процесс исследования уязвимостей сильно усложняется, поскольку в этом процессе возникает ряд принципиальных технологических проблем. Также на практике любой метод атаки на безопасность программного обеспечения основан на предварительной дизассемблировании и анализе механизма защиты бинарного кода. Целью данного исследования является восстановление алгоритма защиты, выявление его слабых мест и недокументированных особенностей для их дальнейшей модификации и (или) автоматизации процесса атаки. Для решения этих задач широко используются методы статического и динамического бинарного анализа. В этих статьях авторы продемонстрировали методы динамического и статического бинарного анализа для обнаружения уязвимостей в бинарном коде. Динамический анализ основан на выполнении программы на ЦП. В свою очередь, статический анализ основан на методике анализа PE-модуля в памяти. Метод динамического анализа используется для восстановления потока выполнения программного обеспечения. Авторы использовали метод статического бинарного анализа для обнаружения шаблонов уязвимостей в модулях PE (portable executable). Авторы продемонстрировали, что запутывание двоичного потока выполнения эффективно для защиты программного обеспечения от несанкционированного исследования. Авторы используют методы динамического выполнения бинарных файлов для запутывания программного обеспечения. В свою очередь, автор использует статический анализатор для анализа алгоритмов выполнения и потока данных. В статье представлены оригинальные методы динамического и статического анализа, а также оптимизация существующих. Мы использовали комбинацию методов статического и динамического анализа, чтобы максимально повысить эффективность защиты и обнаружения уязвимостей. Для реализации статического анализатора мы предлагаем использовать наши предыдущие работы по декомпиляции и реализации на промежуточном языке (IL) . Таким образом, в этой статье мы делаем следующие вклады:

1. Методика анализа ИЛ. Мы вводим новую схему статического бинарного анализа, позволяющую искать уязвимости в промежуточном языке (IL).

2. Техника анализа динамического исполнения. В статье представлены алгоритмы оптимизации для выполнения динамического двоичного кода.

3. Методика гибридного анализа. Методика сочетает в себе статические и динамические алгоритмы защиты программного обеспечения от несанкционированных исследований.

М а т е р и а л ы. Для реализации метода статического анализа двоичный код необходимо интерпретировать на промежуточный язык. В нашей предыдущей работе мы сформулировали и описали эту ИЛ. Для перевода мнемоники x86 в IL мы использовали базис, состоящий из 16 основ н ы х операций. Эта нотация позволяет описывать код в ресурсоориентированном виде. Ресурс реестра обозначается скобками, в которых указано количество ресурсов. Ресурс памяти обозначается скобками, в которых указан адрес ресурса. Количество таких ресурсов вын е с е н о за скобки. Результат отделяется знаком «=». Операции и константы записываются «как есть».

М е т о д ы. Следует отметить, что статический анализ является ресурсоемким процессом, так как исследование выполняется во всем PE-модуле анализируемого программного обеспечения. Динамический анализ обычно более точен, чем статический, поскольку он работает с р е а л ь н ы м и значениями в режиме выполнения. По той же причине динамический анализ часто намного проще статического анализа.

П о л у ч е н н ы е результаты. Эти два подхода (статический и динамический) дополняют друг друга. Комбинация методов используется потому, что статический анализ не позволяет анализировать значения времени выполнения программы, так называемая проблема: «То, что вы в и д и т е, не является тем, что вы выполняете». В случае динамического анализа невозможно проанализировать все возможные пути выполнения и состояния программы. Таким образом, в данной работе мы используем комбинацию описанных выше методов, последовательность бинарного кода тестируется статическим и динамическим анализатором. Следует отметить, что основной проблемой при реализации описанных выше технологий является производительность. При статическом анализе основная проблема производительности заключается в т о м, что необходимо анализировать весь программный код, но в случае больших программ это проблематично. Процесс динамического анализа также имеет серьезные проблемы с производительностью, поскольку метод используется для последовательного анализа каждой инс т р у к ц и и в программном обеспечении. Рассмотрим эту проблему подробнее. Для выполнения анализа потока данных эффективный анализатор должен собрать как можно больше данных, он должен выполнять один шаг определенного пути выполнения и сохранять значения регист р о в на каждом шаге выполнения программы. Существует несколько способов выполнения одного

шага в исполняемом модуле. Следует отметить, что описанные технологии реализованы в виде программных средств.

1. Отладка Win 32 API через «официальные» интерфейсы. Для этого мы должны запустить автоматический отладчик, форсирующий пошаговое выполнение путем установки бита TF в исполнительном контексте и сбора информации о регистрах, памяти и флагах. Процессор Intel x86 использует архитектуру компьютера со сложным набором команд (CISC), что означает небольшое количество регистров специального назначения вместо большого количества регистров общего назначения. Если флаг TF установлен в TRUE, процессор вызовет исключение STATUS_SINGLE_STEP после выполнения одной инструкции. Однако это медленная техника, потому что после каждой инструкции происходит переключение контекста, а отладчику необходимо получить контекст и возобновить выполнение.

2. Техника динамической бинарной аппаратуры (DBI). Двоичное инструментирование — это метод, который изменяет двоичную программу либо перед выполнением, либо во время выполнения, чтобы наблюдать, отслеживать и изменять двоичную программу. Как описано, на первом этапе лаунчер загружает тестовое приложение, внедряет DBI dll с инструментальным кодом, вычисляет точку входа тестового приложения, вводит команду перехода в DBI dll и запускает программу. После перехода код DBI выполняет некоторый анализ и вводит второй переход во вторую инструкцию. Это позволяет проводить эффективный анализ в контексте процесса с возможностью динамического изменения двоичного кода. Для сравнения методов отладки и DBI была написана простая программа, которая запускает цикл заданное количество раз для сбора некоторых эталонных показателей. Методики статического и динамического анализа через DBI были опробованы в задаче защиты бинарного кода. В этом методе протектор вводит дополнительные операции и инструкции в защищенное программное обеспечение.

Следует отметить, что статический анализ является ресурсоемким процессом, так как исследование выполняется во всем PE-модуле анализируемого программного обеспечения. Динамический анализ обычно более точен, чем статический, поскольку он работает с реальными значениями в режиме выполнения. По той же причине динамический анализ часто намного проще статического анализа. При описании метода динамического бинарного анализа рассмотрим понятие линейного участка бинарного кода. Участок двоичного кода является линейным, если он не содержит инструкций по передаче управления другому участку двоичного кода. Инструкции, которые являются нелинейными.

Основной подход, описанный в этой статье, заключается в предоставлении информации о выполнении программы, такой как: вызовы функций, ресурсы, которые использовались на линейных графиках. Такой подход позволяет

выполнять некоторые блоки программы и осуществлять контроль над регистрами, флагами и памятью процессора после каждого выполненного линейного блока. Результаты этого выполнения позволяют отслеживать управление и поток данных программы и описывать последовательность шагов программы с сохранением результатов каждого из них.

ССЫЛКИ :

1. Шудрак М. Лубкин И. «Метод и техника защиты кода от несанкционированного анализа». Журнал «Программное обеспечение и системы», Тверь , вып. 4. 2022.

2. Санг Кил Ча, Танассис Averginos , Alexandre Rebert и David Brumley «Unleashing Mayhem on Binary Code» в Proc. Симпозиума IEEE 2020 года по безопасности и конфиденциальности.

3. Чжи Лю; Сяосун Чжан; Сюнда Ли; «Проактивный поиск уязвимостей с помощью отслеживания информационных потоков» Мультимедийные информационные сети и безопасность (MINES), Международная конференция, 2020 г., том, №, стр. 481-485.

4. Марко Кова; Виктория Фельметсгер ; Грег Бэнкс; Джованни Винья ; «Статическое обнаружение уязвимостей в исполняемых файлах x86», Конференция по приложениям компьютерной безопасности, 2019 г. ACSAC '06. 22-й ежегодник, том, №, стр. 269-278.

5. Дарвиш , С.М.; Гиргис , Словакия; Залат , МС; «Техника обфускации скрытого кода для обеспечения безопасности программного обеспечения» Вычислительная техника и системы (ICCES), Международная конференция 2020 г. , стр. 93-99.

6. Хайбо Чен; Ливэй Юань; Си Ву; Бинью Занг ; Бо Хуанг; Пенчунг Ю ; «Обфускация потока управления с отслеживанием потока информации» Микроархитектура, 2019. МИКРО-42. 42-й ежегодный международный симпозиум IEEE/ACM , том, №, стр. 391-400.