И. О. Леошкевич, А. Ю. Тихонов

АНАЛИЗ МЕТОДОВ ИДЕНТИФИКАЦИИ СТРУКТУР ДАННЫХ В ИСПОЛНЯЕМОМ КОДЕ

Введение

Автоматизированный анализ исполняемого кода играет важную роль в информационной безопасности. В нем заинтересованы антивирусные компании, исследующие вредоносные программы, а также сертификационные лаборатории, занимающиеся верификацией программного обеспечения. Его целью является восстановление алгоритма работы программы и выявление определенных его свойств, таких как наличие ошибок и уязвимостей. В общем случае эта проблема является алгоритмически неразрешимой, поэтому при ее рассмотрении осуществляется поиск приближенного решения. На практике приемлемость приближения определяется отсутствием в нем ошибок второго рода, т. е. пропусков возможных состояний программы. Такие ошибки допустимы лишь при условии, что система информирует аналитика о принятии решений, которые могут привести к их появлению. Количество ошибок первого рода — рассмотрений невозможных состояний — должно быть сведено к минимуму. Анализ исполняемого кода актуален даже в тех случаях, когда доступен исходный код. Это обусловлено в числе прочего тем, что в некоторых случаях анализаторы для языков высокого уровня не способны однозначно определить семантику программы [1]. Анализ строится из двух частей: построение некоторого представления машинного кода и его проверка на соответствие заранее определенной модели.

В настоящей работе рассматривается один из аспектов получения представления машинного кода, а именно получение сведений о структуре обрабатываемых данных и ограничениях, накладываемых на их значения. Имея такую информацию, можно более эффективно проводить другие фазы анализа [2]. Зная о том, как реальное приложение реализует сетевой протокол или обрабатывает входной файл, можно, сравнив этот процесс с формальной спецификацией, обнаружить нежелательные отклонения, ошибки и уязвимости. Также можно осуществлять построение сигнатур новых сетевых атак [3] и повтор диалога между двумя программами. В настоящей работе рассматриваются как динамические методы, строящиеся на наблюдении за выполнением программы на конкретных входных данных, так и статические, проводимые без запуска программы.

1. Статические методы

Наиболее популярный на данный момент дизассемблер IDA [4] может определять некоторые локальные переменные, основываясь на том наблюдении, что для доступа к ним обычно используются регистры EBP и ESP. Такой подход в принципе не позволяет выявлять переменные в динамической памяти, а также те стековые переменные, доступ к которым осуществляется через другие регистры.

Методы [2, 6] разработаны в Университете Висконсин-Мэдисон. В их основе лежит алгоритм анализа возможных значений (Value Set Analysis, VSA) [1], который позволяет определять надмножества значений переменных на каждом этапе выполнения программы. VSA основан на абстрактной интерпретации — он эмулирует выполнение программы на абстрактных входных данных, представляющих собой множества конкретных значений. Для представления множеств чисел используются интервалы с шагом $s[l, u] \Leftrightarrow \{x : l \le x \le u,\ \exists q: q \in Z,\ x = l + sq\}$. Для работы с участками памяти, адреса которых становятся известны только во время выполнения, используется понятие «регион». Регионы описывают стековые фреймы, динамически выделенную память и глобальные области. Множества значений переменных задаются в виде (R, s[l, u]), где R — регион (для численных значений используется нулевой регион), а s[l, u] — смещение в нем.

Алгоритм идентификации структур данных (Aggregate Structure Identification, ASI) [7] разработан компанией IBM в рамках решения «проблемы 2000». На вход принимается модель

обращений к памяти, отражающая взаимосвязь между структурами данных. Результатом работы ASI является ориентированный граф без циклов, каждая вершина которого представляет собой структуру данных, а ребра обозначают отношение вложенности. В работе [2] показано, каким образом можно построить такую модель с помощью VSA.

Алгоритм извлечения форматов файлов (File Format Extraction for x86,FFE/x86) [6] определяет структуру данных, записываемых на диск. От пользователя требуется указать, какие функции отвечают за вывод, и описать их параметры. Результатом является иерархический конечный автомат, получаемый из интерпроцедурного графа потока управления. С каждым состоянием автомата с помощью VSA и ASI связываются возможные значения и структуры данных, выводимых соответствующей функцией.

VSA [1], ASI [2] и FFE/x86 [6] реализованы в закрытом исследовательском проекте CodeSurfer/x86 и потому недоступны для непосредственного анализа. Однако применяемые в них подходы видятся перспективными. По утверждению авторов, им удалось восстановить большую часть структур данных в скомпилированных утилитах с открытым исходным кодом размером до 10^5 машинных инструкций.

2. Динамические методы

Idastruct [5] представляет собой расширение IDA для получения информации о полях записей, находящихся в динамической памяти. Он основан на x86emu — эмуляторе процессора x86. Idastruct вызывается после эмуляции каждой машинной команды и функции выделения памяти, сохраняя адреса выделенных блоков памяти и обращения к ним. Затем на основании этой информации база данных записей IDA дополняется описаниями выделенных сэмулированным фрагментом программы записей. Idastruct считает все участки памяти, выделенные в одной точке программы, принадлежащими одному типу, что в отдельных случаях приводит к неверным результатам.

Приведенные ниже методы разработаны компанией Microsoft для анализа политик безопасности [11] и сетевых протоколов [8–10]. Последние основываются на знании того, каким образом устроены современные протоколы. Так, в них используются понятия «сессия», «сообщение», «поле сообщения» и «тип сообщения». Discoverer [10] структурирует трафик сетевого приложения без анализа его кода. Основным недостатком такого подхода является то, что кодировки полей сообщений должны быть известны заранее. В остальных методах используется трассировка, в ходе которой отслеживаются зависимости между данными.

Polyglot [9] предполагает наличие полей фиксированной и произвольной длины. Границы полей произвольной длины задаются либо определяющими полями (direction fields), либо разделителями. Определяющие поля прямо или косвенно задают длину блока данных, а разделители представляют собой последовательности байтов, обозначающие конец блока данных. Определяющие поля ищутся исходя из следующего их свойства: данные, зависящие от определяющих полей, используются для увеличения указателей на данные, зависящих от трафика. Поиск разделителей основывается на том наблюдении, что в ходе выполнения программы они сравниваются с каждым фрагментом входа. Tupni [8] работает аналогично, но использует улучшенные эвристики. Так, он определяет поля как непересекающиеся фрагменты трафика, сумма количеств обращений к которым как к целому максимальна. Tupni предполагает, что существует один или несколько циклов, обрабатывающих сообщения в том порядке, в котором они идут в трафике. Анализируя такие циклы, он определяет границы сообщений. Сообщения, обрабатываемые схожим набором инструкций, считаются принадлежащими одному типу.

ShieldGen [3] строит обобщенные сигнатуры сетевых атак по заранее известному протоколу, уязвимому приложению и конкретной атаке. Он пытается сгенерировать возможные варианты атаки, чтобы сигнатура наиболее полно описывала уязвимость. Атаки проверяются на трассируемом приложении, по трассе определяется их успешность, а также возможность внесения

новых модификаций. Сигнатура представляет собой предикат над полями сообщений протокола, описывающий объединение всех полученных вариаций исходной атаки.

ConfigRE [11] по конфигурационным файлам и коду приложения выявляет его политики безопасности. В частности, определяются возможные субъекты, объекты и правила разграничения доступа. Для этого ConfigRE производит две трассировки: инициализации приложения, дающей представление о синтаксической структуре конфигурационных файлов, и обработки запроса на доступ к защищенному ресурсу, позволяющей определить семантику конфигурации. Синтаксис выявляется аналогично случаю с сетевым трафиком, а семантика определяется исходя из взаимодействия потоков данных из конфигурационных файлов и из запроса пользователя.

По данным их авторов, Discoverer, Polyglot и Tupni способны проанализировать широко используемые протоколы, как бинарные (CIFS), так и текстовые (HTTP). ShieldGen был использован на нескольких известных уязвимостях, и результаты его работы описывали достаточно широкое множество возможных атак. С помощью ConfigRE была получена модель политик безопасности веб-сервера Apache. Следует отметить, что лежащий в основе этих методов динамический анализ зависимостей данных можно обойти, преобразовав зависимости данных в зависимости по управлению, что может быть использовано для обфускации.

Заключение

В настоящей работе были рассмотрены существующие методы идентификации структур данных — статические и динамические. Алгоритмы [8—11] направлены на решение частного случая проблемы, в то время как [2, 6] пытаются решить ее в общем. Доступные на данный момент реализации [4, 5] дают практически значимое, однако далеко не полное представление о структуре данных в памяти программы. Другие методы реализованы в закрытых проектах, поэтому об их эффективности можно судить лишь по данным их авторов.

Идентификация структур данных является составной частью более общей проблемы анализа исполняемого кода и автоматического поиска уязвимостей в нем. Представляются актуальными следующие направления для дальнейших исследований:

• Существующие методы работают с идиомами, свойственными одной архитектуре. Обобщение на большее количество архитектур расширит область применимости, а также улучшит качество работы на программах с нестандартной организацией.

• Улучшение качества работы с обфусцированным кодом позволит идентифицировать структуры данных во вредоносном программном обеспечении.

• Получение как можно более полного представления ввода-вывода программы позволит в лучшей мере оценить его свойства.

• Анализ свойств модели ввода-вывода позволит производить автоматический поиск ошибок и уязвимостей в программном обеспечении.

СПИСОК ЛИТЕРАТУРЫ:

1. *Reps T., Balakrishnan G.* Improved memory-access analysis for x86 executables // Proceedings of the International Conference on Compiler Construction, April 2008.

2. *Balakrishnan G., Reps T.* DIVINE: DIscovering Variables IN Executables // Proceedings of the VMCAI. Nice, France. Jan. 14—16, 2007.

3. *Cui W., Wang H. J., Peinado M., Locasto M.* ShieldGen: Automated Data Patch Generation for Unknown Vulnerabilities with Informed Probing // Proceedings of the 2007 IEEE Symposium on Security and Privacy. May 2007.

4. Interactive Disassembler, http://www.idapro.ru.

5. x86 Disassembler Internals 22c3: Private Investigations. December. 2005.

6. *Lim J., Reps T., Liblit B.* Extracting output formats from executables // Proceedings of the IEEE Working Conference on Reverse Engineering. Benevento, Italy. Oct. 23—27, 2006.

7. *Ramalingam G., Field J., Tip F.* Aggregate structure identification and its application to program analysis // POPL, 1999.
8. *Cui W., Peinado M., Chen K., Wang H. J., Irun-Briz L.* Tupni: Automatic Reverse Engineering of Input Formats // Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS). Alexandria, VA. October 27—31, 2008.
9. *Caballero J., Yin H., Liang Z., Song D.* Polyglot: Automatic Extraction of Protocol Message Format using Dynamic Binary Analysis // Proceedings of the 14th ACM Conference on Computer and Communications Security. Alexandria, VA. October 2007.
10. *Cui W., Kannan J., Wang H. J.* Discoverer: Automatic Protocol Reverse Engineering from Network Traces // Proceedings of the 16th USENIX Security Symposium. Boston, MA. August 2007.
11. *Wang R., Wang X. F., Zhang K., Li Z.* Towards automatic reverse engineering of software security configurations // Proceedings of the 15th ACM conference on Computer and communications security. October 27—31, 2008. Alexandria, Virginia, USA.

Г. Н. Мальцев, И. В. Прохоров

ВОПРОСЫ БЕЗОПАСНОСТИ ПРИ ОРГАНИЗАЦИИ ЗАЩИТЫ СИСТЕМЫ ЭЛЕКТРОННЫХ ПЛАТЕЖЕЙ

В настоящее время широкое распространение получили системы для оплаты повседневных услуг (системы моментальных платежей) с использованием различных технологических платформ: платежные системы в сети Интернет, банкоматы, терминалы самообслуживания, карточки пополнения. Поскольку более половины населения России пользуется услугами операторов сотовой связи или Интернета, как минимум, ежемесячно, а то и ежедневно значительное количество людей сталкивается с необходимостью пополнения баланса или оплаты счета.

Создание современной и защищенной системы, позволяющей принимать платежи от населения и обеспечивать высокий уровень безопасности, является на сегодняшний день новой и еще не изученной задачей. В России рынок подобных систем начал формироваться только после 2000 г., стали активно развиваться два направления: карточки пополнения и платежные системы Интернета. Каждое направление поначалу было ориентировано на различные рынки: карточки пополнения выпускались поставщиками услуг для увеличения баланса пользователя в системе, а платежные системы осуществляли выпуск электронных денег, позволяющих приобретать товары в интернет-магазинах. Однако основной проблемой в обоих случаях была сложность приема денег от населения, платежные системы также выпускали карточки пополнения. Но уже к 2004 г. на рынке появились первые платежные системы, которые предоставляли широкий спектр решений для приема платежей в пользу различных операторов услуг (также именуемых провайдерами), тем самым вытеснив с рынка карточки пополнения, а также предоставляли удобный способ конвертации наличных денег в электронные. На сегодняшний день основным инструментом для зачисления денег на счет является платежный терминал — аппарат, позволяющий совершить платеж наличными деньгами в пользу любого провайдера.

Конечно же новое направление не осталось без внимания различных мошенников с целью получения незаконной выгоды. Сейчас количество попыток незаконно списать деньги со счетов пользователей только увеличивается. Можно выделить два основных направления атак преступников: на сервера платежной системы и на ее пользователей.