тельные средства защиты информации. Исследование возможностей существующих методов и средств защиты информации, а также их доработка в случае неудовлетворительных результатов, на предмет возможности противодействия скрытым каналам передачи информации данного типа является темой будущих исследований.

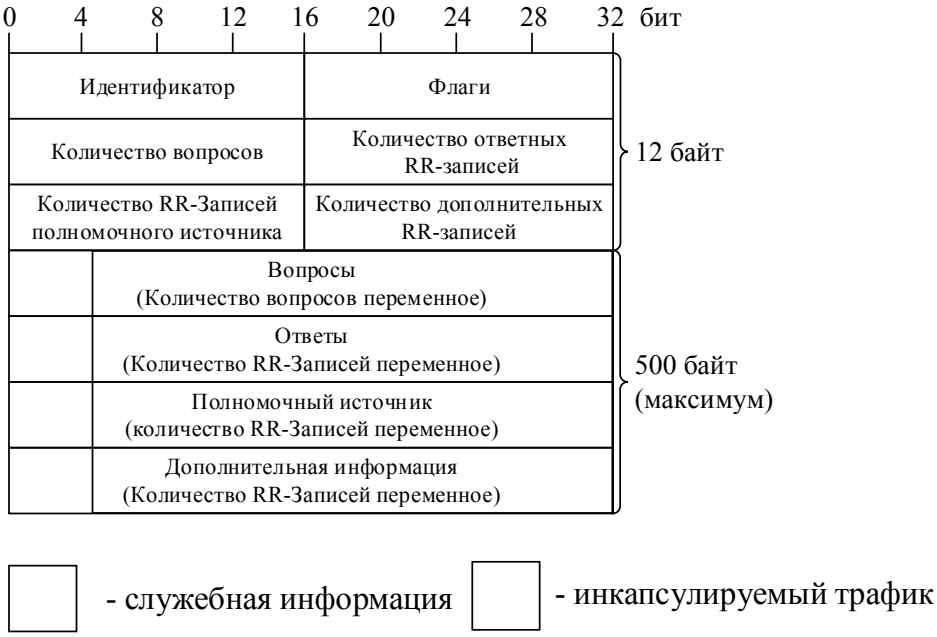

Рис. 2. Структура DNS-запроса

Литература

1. ГОСТ Р 53223.1-2008. Информационная технология. Защита информационных технологий и автоматизированных систем от угроз информационной безопасности, реализуемых с использованием скрытых каналов. Часть 1. Общие положения. – Введ. 2009-09-30. – М. : Изд-во стандартов, 2008, 12 с.

2. ГОСТ Р 53223.2-2009. Информационная технология. Защита информационных технологий и автоматизированных систем от угроз информационной безопасности, реализуемых с использованием скрытых каналов. Часть 2. Рекомендации по организации защиты информации, информационных технологий и автоматизированных систем от атак с использованием скрытых каналов. – Введ. 2009-12-01. – М. : Изд-во стандартов, 2008, 12 с.

3. Безукладников И.И. Скрытые каналы в распределенных автоматизированных системах / Безукладников И.И., Кон Е.Л. – Уфа: УГАТУ, 2010. - с. 245 – 250.

Д.И. Тихонов, А.В. Дорфман

ПРИМЕНЕНИЕ PROGRAM SLISING В ЗАДАЧАХ REVERSE ENGINERRING

(Самарский государственный технический университет)

Reverse engineering (или обратная разработка, далее RE) –давно применяемая техника низкоуровневого исследования программ. RE в основном ис-

пользуется для анализа программ без исходных текстов. RE так же применяется как инструмент восстановления программных алгоритмов из проприетарного программного обеспечения для создания его дальнейших бесплатных аналогов. В компьютерной безопасности RE чаще всего применяется как инструмент исследования программных продуктов на наличие уязвимостей.

Большинство задач RE не зависят от конечной цели применения, а остаются общими для всего процесса обратной разработки, будь то анализ ПО на наличие уязвимостей, либо же восстановление программного алгоритма. Говоря о задачах, которые встают перед исследователями во время процесса обратной разработки, можно выделить следующие:

1) нахождение целевого алгоритма

2) нахождение структур данных, которые обрабатываются целевым алгоритмом

3) нахождение зависимостей процесса обработки от поступивших на вход целевого алгоритма данных

Выделенные задачи носят ключевой характер в процессе исследования программного обеспечения на предмет безопасности, поскольку именно благодаря их решению возможна локализация, дальнейший анализ и понимание исключительно тех участков программного обеспечения, которые потенциально могут содержать ошибки и возможные уязвимости.

Упоминая об анализе программного обеспечения на выявление уязвимостей стоит так же отметить тенденцию постоянного роста объемов программного кода, который используется при написании современного ПО, в особенности промышленного и проприетарного. В таких условиях, когда даже при одном запуске исследуемой программы выполняются огромные объемы машинного кода одним из важных факторов успешного применения RE встает так же задача автоматизации процессов обратной разработки.

Одним из подходов к автоматизации процессов RE, так же позволяющим решать некоторые из ключевых задач, является технология program slicing. Program slicing (программный срез, далее PS) – это метод автоматической декомпозиции программ, реализуемый благодаря анализу потока данных и потока управления программы.[1]

Основная идея данной технологии заключается в выборке из целевой программы только тех вычислений, которые участвовали в формировании определенной переменной. Выбор оператора, который формирует эту переменную и самой переменной называется slicing criterion (критерий среза).[2] Такая усеченная версия исследуемой программы, которая воспроизводит поведение оригинальной программы при вычислении определенной переменной, удовлетворяющей критерию среза, называется слайсом или срезом исследуемой программы.[2] Процесс построения усеченной версии оригинальной программы называется слайсингом или срезом программы. Слайсы представляют собой мысленную абстракцию над исследуемой программой, которая помогает исследователю сконцентрироваться исключительно на определенном аспекте семантики вычислений исследуемой программы.[1]

Слайсинг может быть реализован на базе статического и динамического анализа программного кода. Подход, реализующий наблюдение за программой во время её выполнения, называется динамический анализ (dynamic analysis).[] В противоположность ему существует так же подход работающий над всем её статическим кодом и получивший название статический анализ (static analysis).

Существует важное различие между статическим и динамическим слайсингом программы. Статический слайсинг производится независимо от ввода, поступившего в программу над её статическим кодом, в то время как динамический слайс строится во время выполнения программы на основе поступившего ввода.[2]

Для построения статических слайсов существует несколько подходов, первый из которых (предложенный Майком Вейсером [1] в 1984 году) предполагает последовательную выборку из кода программы множества транзитивно зависимых операторов, в соответствии с зависимостями по данным и потоку управления программы, основываясь на критерии среза программы. Это означает, что из исследуемой программы будут исключены все операторы не влияющие на формирование переменной выбранной в критерии среза. Оставшаяся часть программы будет представлять собой копию поведения оригинальной при вычислении искомой переменной.

Альтернативный подход предполагает использование графа зависимостей программы (ГЗП) для построения статического слайса. ГЗП представляет собой орграф в котором каждый оператор и управляющая конструкция представляют собой вершины, а ребра представляют собой зависимости программы по данным и управлению. Критерием среза выбирается одна из вершин, и слайс представляет собой все вершины графа, через которые можно достичь искомой.
В обоих случаях при построении слайса используется только статически известная информация.

Подход построения слайса программы в динамике был предложен позже статического и основывается на анализе зависимостей потока данных в программе. В данном виде слайсинга используются только сведения полученные при непосредственном выполнении программы. К критерию среза помимо оператора и искомой переменной добавляется еще и фиксированный ввод, на основе которого выполнялась программа. Основная идея заключается в ответе на вопрос о том, как переменные обрабатывались программой, при получении искомого значения. Данный подход является разновидностью так называемого анализа обратного потока (flowback analysis).

В зависимости от направления в котором строится слайс, начиная от критерия среза, различают обратный и прямой слайсинг. Обратный слайсинг подразумевает под собой выборку всех операторов, которые привели к формированию искомой переменной в критерии среза. Прямой же слайсинг отражает какие переменные и ветви программы будут в дальнейшем зависеть от переменной, выбранной в критерии среза. Существует так же понятие чоппинга (chopping) – это случай генерализации прямого и обратного слайсинга, когда в слайсе отмечается не только история формирования искомой переменной в крите-

рии среза, но и дальнейшие порождаемые ею зависимости в исследуемой программе.

В решении ключевых задач, которые встают перед исследователями в процессе RE динамический анализ играет значительную роль.

Во-первых, такой подход не требует исходного кода или специальной компиляции исследуемой программы, и таким образом, является привлекательным для более широкого круга программного обеспечения, в том числе и для промышленного ПО.[4] Этот результат достигается благодаря внедрению дополнительного кода (инструментации [3]) в код исследуемой программы, который собирает и обобщает требуемую иформацию. В связи с этим, динамический анализ – возможность исследовать код во время его выполнения – стал одним из фундаментальных инструментов исследования в области компьютерной безопасности.

Во-вторых, во время динамического анализа не требуется дополнительного анализа потока управления программы поскольку во время такого типа анализа выполнение ветвей условий программы можно назвать предопределенным. В отличие от статического анализа, когда требуется просчитать все возможные условия выполнения тех или иных базовых блоков (или операторов) программы.

С точки зрения трудоемкости при разработке инструментов исследования, а так же затратности самого процесса анализа – такой подход сильно экономит время исследователя.

В-третьих, наибольший плюс динамического анализа заключается в том, что во время выполнения программы становится доступной масса полезной информации о выполнении программы, которая не доступна в статике. Эта информация может представлять собой, как минимум, значения регистров процессора или областей памяти, при выполнении программы, а так же аргументы различных системных вызовов, которые порождает.

Большинство инструментов исследования такого типа реализуется на основе фреймворков динамической бинарной инструментации (dynamic binary instrumentation frameworks) или DBI, таких как Pin, DynamoRIO, и Valgrind. Они предоставляют базовый функционал, который помогает инструментировать и запускать исследуемые программы, а так же окружение, набор функций и структур, которые позволяют писать инструменты исследования.

Разрабатываемый автором данной статьи инструмент исследования базируется на фреймворке Valgrind, чему пособствовали следующие несколько причин:

1. Реализации Valgrind существуют для множества платформ, таких как x86/Linux, AMD64/Linux, and PPC{32,64}/{Linux,AIX}[3]

2. Внутренняя реализация фреймворка позволяет дизассемблировать машинный код исследуемой программы во время его выполнения не в ассемблерные мнемоники, как это делают множество других инструментов, а в архитектуро-независимое промежуточное представление (IR), которое позволяет пол-

ностью отслеживать изменения в состоянии процессора после выполнения каждой инструкции.

Наличие такого внутреннего представления машинного кода, независимо от архитектуры, очень сильно облегчает работу программиста, который разрабатывает инструменты исследования. Поскольку, в итоге, программист пишет инструментацию для промежуточного представления количество видов операций которого заранее известно и несоизмеримо мало по сравнению с количеством различных видов инструкций даже в одной марке процессора. IR так же предполагает под собой большие возможности для переносимости кода инструмента исследования на другие платформы, для которых написан фреймворк.

На основе приведенной методологии авторами был разработан подход, позволяющий решать некоторые из указанных выше задач, возникающих в процессе RE.

Разработанный подход базируется на примененнии технологий program slicing и taint analysis [5] в связке. Он подразумевает использование принципов построения динамических слайсов, основываясь только на введенных в программу данных, которые и представляют собой критерий среза для построения слайса. Разработанный подход еще не был формализован нигде в научной литературе, однако на данный момент уже разработан прототип инструмента, который реализует такую технологию и находится в стадии доработки.

Данный инструмент позволяет собрать информацию об адресах инструкций, которые так или иначе обрабатывали введенные в программу данные и в свою очередь порождали новые данные, обработка которых так же отслеживалась далее (taint analysis). Это позволяет сделать мысленную абстракцию от всего кода программы и изучать только семантику обработки данных программой. Можно без преувеличения сказать, что разработанный инструмент позволяет выполнять динамический чоппинг программы по данным или динамический чоппинг данных (dynamical data chopping), если использовать терминологию применимую в PS. Данный подход позволять решать задачи RE 1 и 3 соответственно. Однако для решения задачи RE 3 требуется дальнейшая доработка визуализации зависимостей процесса выполнения обработки от полученных программой данных.

Таким образом, дальнейшая формализации разработанного нами подхода, а так же разработка визуализации, отображающей зависимости процесса обработки от полученных на входе программы данных являются предметом для дальнейшей работы.

Литература

1. Weiser, M. Program slicing / M. Weiser // IEEE Transactions on Software Engineering. – 1984. -P 352−357.

2. Tip, F. A survey of program slicing techniques / F. Tip // Journal of Programming Languages. – 1995. -P 121−189.

3. Nethercote, N. Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation / N. Nethercote, J. Seward // Proceedings of ACM SIGPLAN 2007

Conference on Programming Language Design and Implementation (PLDI 2007). - 2007.

5. Newsome, J. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software / J. Newsome, D. Song // In Proceedings of the 12th Annual Network and Distributed, System Security Symposium (NDSS 2005), -2005.

6. Тихонов, Д.И Введение в технологию taint analysis / Тихонов Д.И, Буканов Д.Ф // Актуальные проблемы информационной безопасности. Теория и практика использования программно-аппаратных средств: материалы VII всероссийской научно-технической конференции. -2014.

В.П. Цветов

ОБ ОДНОМ СИНТАКСИЧЕСКОМ АЛГОРИТМЕ НА ГРАФАХ

(Самарский государственный университет)

Алгоритмы на графах представляют интерес для различных прикладных направлений, в частности, они могут быть полезны при проектировании информационных систем или построении систем безопасности. Существующие алгоритмы используют различные подходы, опирающиеся на специальные представления и структуры данных, обширная библиография по которым представлена, например в [1].

В докладе рассматривается алгоритм нахождения оптимальных маршрутов на реберно-размеченных графах. Результат не претендует на практическую новизну, но демонстрирует возможный абстрактно алгебраический метод решения задач, связанных с построением степеней элементов ограниченной сверху структурно упорядоченной полугруппы [2] в матричном представлении.

Введем следующие обозначения.

$\mathbb{N}$-множество натуральных чисел;

$1..n := \{1,2,\dots,n\} \subset \mathbb{N}$;

$\mathcal{A} := \{\alpha_1, \alpha_2, \dots, \alpha_n\}$ - алфавит мощности n;

$\mathbb{W}_{\mathcal{A}}^{k}$- множество слов длины k над алфавитом $\mathcal{A}$;

$\mathbb{W}_{\mathcal{A}} := \bigcup_{k=0}^{\infty} \mathbb{W}_{\mathcal{A}}^{k}$- множество слов над алфавитом $\mathcal{A}$;

$\langle \mathbb{W}_{\mathcal{A}}, (\bullet) \rangle$- моноид слов над алфавитом $\mathcal{A}$ с операцией сцепления;

$\mathbb{R}$- множество вещественных чисел;

$\mathbb{R}_{+}$- множество неотрицательных вещественных чисел;

$[0,1] \subset \mathbb{R}$ - замкнутый интервал вещественных чисел от 0 до 1;

$\langle \mathbb{R}, (\cdot, +) \rangle$ - поле вещественных чисел;

$\langle \mathbb{R}, (\wedge, \vee,) \rangle$ - решетка вещественных чисел, где $\xi_1 \wedge \xi_2 := \min(\xi_1, \xi_2)$, $\xi_1 \vee \xi_2 := \max(\xi_1, \xi_2)$;