Статический анализ бинарных модулей z/OS

Леденева Е. Ю., студентка кафедры системного программирования СПбГУ, cthfvrl@gmail.com

Аннотация

В данной работе производится сравнение некоторых статических анализаторов исполняемых файлов архитектуры x86, а также рассматривается разработка аналогичного анализатора для бинарных модулей z/OS на основе наиболее подходящего из рассмотренных решений.

Введение

Статический анализ кода — метод получения разнообразной информации о программе без её исполнения.

Статический анализ используется для поиска уязвимостей в коде, верификации программного обеспечения, анализа поведения программ при отсутствии исчерпывающей документации, обратного инжиниринга. Анализ может производиться как над исходным кодом, так и над бинарными файлами. Потребность анализировать бинарный код возникает, например, при отсутствии доступа к исходному коду.

Работа статического анализатора обычно основывается на построении графа потока управления — графа, вершины которого содержат блоки инструкций, а рёбра представляют возможные потоки управления. Затем этот граф используется для выполнения анализа или используется пользователем для понимания структуры исходного бинарного модуля. Предварительно производится дизассемблирование бинарного кода.

Существуют различные продукты, в том числе с открытым исходным кодом, реализующие статический анализ, каждый из которых имеет как достоинства, так и недостатки. Поэтому необходимо проанализировать известные продукты, выбрать наиболее подходящий и расширить его функциональность для наших нужд. Так, большинство инструментов поддерживают только архитектуру процессора x86, однако позволяют встраивать в себя реализацию и иных архитектур, что и планируется сделать в рамках данной работы.

Трудность задачи

Анализ бинарных модулей вообще и модулей z/Architecture в частности имеет ряд проблем:

- Существование динамических переходов: до запуска программы в общем случае неизвестно, куда будут осуществляться переходы, так как их адреса могут храниться в регистрах, вычисляться по ходу исполнения и таким образом указывать на любые точки как внутри, так и за пределами анализируемого модуля. Это существенно затрудняет построение адекватного графа потока управления;

- Большое количество инструкций в архитектуре и сложность их семантики;

- Инструкции меняют состояние системы, что влияет на дальнейшее поведение. Так, в z/Architecture некоторые команды меняют поля PSW[1], в частности, Condition Code, от содержимого которого может зависеть дальнейшее исполнение. Например, возможны изменения последовательности исполнения команд, поэтому анализатор должен учитывать эти побочные эффекты;

- Отсутствие разделения на сегменты кода и данных в z/Architecture.

- Наличие инструкции `EX`, имеющей динамическую семантику [1].

Перечисленные проблемы делают задачу построения графа потока управления очень сложной, поэтому она сужается до построения приближения графа.

Обзор существующих решений

В данном разделе представлен обзор двух платформ бинарного анализа — BAP и Jakstab, а также обоснование выбора одной из них как основы для реализации анализатора модулей z/OS.

[1] Program status word, слово состояния программы.

BAP

BAP (Binary Analysis Platform) — фреймворк бинарного анализа с открытым кодом, разработанный в университете Карнеги-Меллон [2, 3]. На данный момент BAP поддерживает архитектуры процессоров ARM, x86 и другие. Платформа написана на OCaml, предоставляется интерфейс для C, Python и Rust. Помимо самого фреймворка существует набор уже готовых инструментов, плагинов и библиотек.

Результаты анализа, произведённые BAP, выводятся в форме графа потока управления, ассемблерного кода, набора инструкций промежуточного языка и других формах.

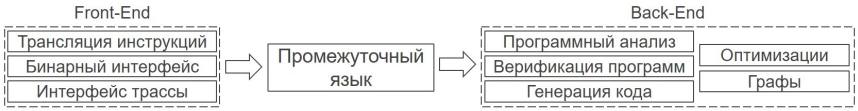

Рис. 1: Архитектура бинарного анализа и компоненты BAP [3]

BAP состоит из двух частей (рис. 1): front-end и back-end:

- Front-end переводит бинарный код в программу на промежуточном языке BIL[2]. Для этого используется дизассемблирование алгоритмом линейной развёртки (linear sweep). Семантика BIL полностью отражает семантику инструкций ассемблера.

- Back-end занимается непосредственно анализом полученного кода на промежуточном языке — строит граф потока управления, производит оптимизации и верификацию.

Таким образом, для встраивания некоторой архитектуры в BAP, необходимо реализовать для неё только front-end — преобразование бинарного модуля в BIL.

Недостаток BAP заключается в том, что при построении графа потока управления он игнорирует косвенные переходы, что делает его анализ далёким от полноты. Для улучшения анализа предлагается использовать плагин IDA Pro, который использует алгоритм рекурсивного обхода для дизассемблирования [4].

Однако IDA Pro не имеет открытого исходного кода, поэтому встраивание в неё поддержки z/Architecture невозможно. К тому же, для

[2]BAP Instruction Language.

разрешения косвенных переходов в этом инструменте используются эвристики, основанные на обнаружении в коде некоторых известных шаблонов, например, конструкции switch (путём поиска таблиц переходов). Такой подход не даёт желаемых результатов.

Jakstab

Jakstab (Java Toolkit for Static Analysis of Binaries) — платформа для статического анализа бинарных модулей, разработанная Йоханнесом Киндером [5].

Jakstab реализован на Java и представляет собой фреймворк для построения графа потока управления. Jakstab поддерживает только архитектуру x86, однако можно добавить желаемые архитектуры путём конфигурирования дизассемблера — одной из компонент платформы.

В результате работы Jakstab генерируются:

- Ассемблерный код;
- Граф потока управления;
- Автомат потока управления[3].

Jakstab не разделяет работу на последовательные стадии дизассемблирования и построения графа потока управления, а производит реконструкцию потока управления, "на лету"дизассемблируя достижимые инструкции (рис. 2). Этим Jakstab существенно отличается от подходов BAP, IDA Pro и других подобных инструментов.

Jakstab имеет свой промежуточный язык в стиле RTL[4], в который он переводит дизассемблированные инструкции, используя библиотеку семантик.

Дизассемблирование является итеративным: реконструкция выполняется не за один просмотр входного модуля, а за несколько итераций дизассемблера, на каждой из которых извлекается новая информация о возможных переходах. При построении графа потока управления анализируется и поток данных: именно это позволяет строить предположения о содержимом регистров и разрешать зависящие от них динамические переходы.

---

[3] Граф, вершины которого помечены логическими состояниями, а рёбра — инструкциями. Логическое состояние — это набор, состоящий из счётчика команд, значений регистров, содержимого памяти и динамически выделенных объектов.

[4] Register transfer language.

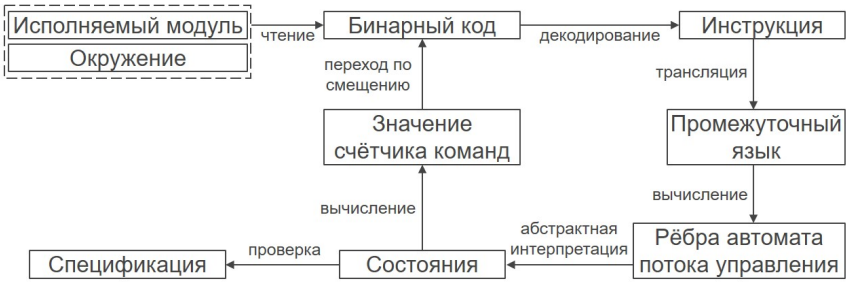

Рис. 2: Единая архитектура дизассемблирования и анализа Jakstab [5]

Сравнение

Подходы к анализу рассмотренных платформ принципиально различаются.

В BAP никак не рассматривается динамическая сторона программ — содержимое регистров и переходы по ним. Однако, в реальных примерах поток управления очень часто зависит от регистров, поэтому исключение этого аспекта из анализа даёт слишком плохие результаты.

Jakstab тоже не разрешает абсолютно все динамические переходы, но одновременный анализ и кода, и данных позволяет делать полезные предположения о потоке управления и получать существенно более точную информацию.

Таким образом, по результатам проведённого исследования, принято решение использовать Jakstab как основу для реализации статического анализа бинарных модулей z/OS.

Имеющиеся результаты

В качестве основы для реализации статического анализа использованы результаты работы Василия Мироновича [6], которому удалось встроить обработку бинарных модулей z/OS в платформу Jakstab, а проведённое мною сравнение существующих подходов выявило явное преимущество платформы Jakstab над другими разработками.

Разработанный прототип поддерживает ограниченное количество инструкций HLASM[5] и принимает файлы двух форматов — объект-

[5]High-Level Assembler — ассемблер фирмы IBM для z/Architecture.

ные файлы и исполняемые модули.

В настоящий момент ведётся разбаротка графического интерфейса[6], а также работа по развитию модуля поддержки z/Architecture[7].

Реализация графического интерфейса

В ходе работы был реализован графический интерфейс, как более удобная оболочка для имеющегося интерфейса командной строки Jakstab.

Графический интерфейс — отдельное приложение, написанное на Java и принимающее путь к Jakstab в качестве одного из параметров. Пользователь задаёт путь к исполняемому модулю или объектному файлу, который надо проанализировать. Пути к результатам работы — графу и автомату потока управления и ассемблерному коду — можно как заполнить автоматически по исходному пути, так и задать явно. Поддерживается вывод графов в формате .png или в отдельную вкладку приложения. Также вывести на экран можно дизассемблированный код.

На рис. 3 представлен пример работы программы. Выведен построенный Jakstab'ом (со встроенным модулем поддержки z/Architecture) граф потока управления. Можно также заметить, что в данном примере есть динамический переход — `BCTR R4,R6` — и возможные адреса перехода вычислены верно. На рис. 4 представлен результат дизассемблирования того же модуля.

В качестве дальнейшей работы планируется добавление поддержки некоторых флагов Jakstab, доступных в интерфейсе командной строки, — в частности, флагов для выбора анализов, проводимых при реконструкции потока управления.

Развитие статического анализатора

Ведётся работа по улучшению работы имеющегося прототипа модуля поддержки z/Architecture.

На данный момент была добавлена более адекватная обработка входных форматов — с распознаванием неподдерживаемых файлов и выводом соответствующего сообщения об ошибке.

[6] Графический интерфейс на GitHub: https://github.com/cthfvrl/jakstab-gui
[7] Код на GitHub: https://github.com/cthfvrl/jakstab

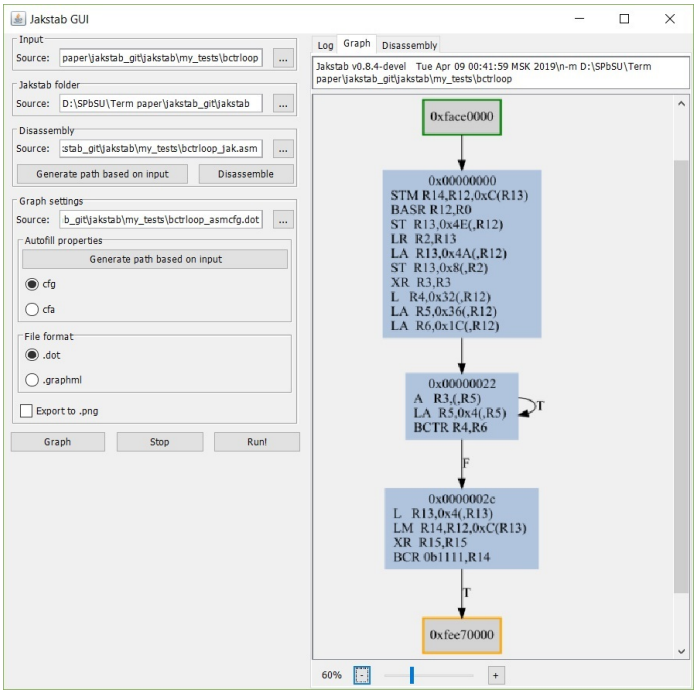

Рис. 3: Граф потока управления

Была замечена и исправлена ошибка в обработке инструкций перехода по регистру. Исправлены некоторые сообщения об ошибках.

Одной из главных проблем остаётся инструкция `EX` — широко используемая на практике, но имеющая сложную динамическую семантику. Она исполняет инструкцию по динамически определяемому адресу — что можно было бы имплементировать аналогично динамическим переходам. Однако `EX` также меняет код исполняемой инструкции, что представляет основную сложность.

Такое поведение можно сравнить с самоизменяющимся кодом, разбор которого обычно не рассматривается при статическом анализе. Jakstab предполагает встраивание и такого вида анализа, и это является основной перспективой дальнейшей работы.

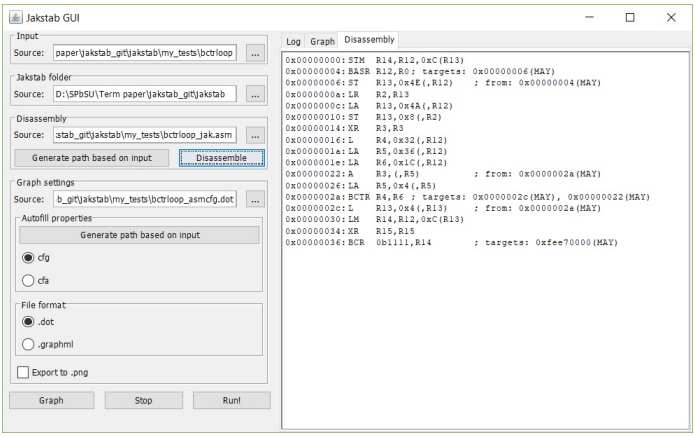

Рис. 4: Дизассемблирование

Заключение

В данной работе было произведено детальное сравнение известных решений статического анализа для архитектуры x86 и сделан выбор в пользу Jakstab. Разработан графический интерфейс для Jakstab для более удобного запуска анализа и вывода его результатов. Произведена частичная доработка взятого за основу прототипа, однако на этом работа не заканчивается.

Литература

[1] IBM Corporation. z/Architecture Principles of Operation, Tenth Edition (September, 2012).

[2] BAP: Binary Analysis Platform.
— https://github.com/BinaryAnalysisPlatform/bap/

[3] David Brumley and Ivan Jager and Thanassis Avgerinos and Edward J. Schwartz. BAP: A Binary Analysis Platform. // Springer, Berlin, Heidelberg, 2011. — Vol. 6806 of Lecture Notes in Computer Science.

[4] IDA. — https://www.hex-rays.com/products/ida/

[5] Dipl.-Inf. Johannes Kinder. Static Analysis of x86 Executables. // Technische Universität Darmstadt, 2010.

[6] Миронович, В.С. Статический анализ бинарных модулей среды z/OS с помощью Java toolkit for static analysis of binaries (Jakstab). // Материалы 7-й всероссийской научной конференции по проблемам информатики «СПИСОК-2017», 2017. — С. 38–43. http://spisok.math.spbu.ru/2017/txt/SPISOK-2017.pdf