УДК 004.4′422

ВНУТРЕННЕЕ ПРЕДСТАВЛЕНИЕ ПРОТОТИПА УТИЛИТЫ ДЛЯ ВОССТАНОВЛЕНИЯ КОДА

Израилов Константин Евгеньевич
аспирант Санкт-Петербургского государственного университета телекоммуникаций имени проф. М.А. Бонч-Бруевича, г. Санкт-Петербург.

АННОТАЦИЯ

Статья посвящена внутреннему представлению прототипа утилиты, предназначенной для восстановления алгоритмов бинарного кода. Описываются назначение и форма обособленных представлений, составляющих вместе единое внутреннее. Также приводятся внутренние данные утилиты в каждом из представлений для реально работающего примера.

Ключевые слов: уязвимости кода; восстановление алгоритмов; утилита; внутреннее представление.

THE INTERNAL REPRESENTATION OF A PROTOTYPE UTILITY FOR THE RECOVERY CODE

Izrailov K.E.
Postgraduate student of the Bonch-Bruevich Saint-Petersburg State University of Telecommunications, Saint Petersburg

ABSTRACT

The article is devoted to the internal representation of a utility's prototype that designed to recover the algorithm of binary code. The purpose and form of separate representations, which together constitute a single internal, is in the article. Also, there is given the internal data of utility in each representations for the actual working example.

Keywords: code vulnerability; recovery algorithms; utility; internal representation.

Введение

Задача восстановления алгоритмов работы бинарного кода была востребованной уже с первых дней использования программного обеспечения в области информационной безопасности, однако на данный момент удовлетворительного решения она так и не имеет. И хотя абсолютно полностью выполнить данную задачу

невозможно даже теоретически, тем не менее уже частичное ее решение позволило бы подвергнуть анализу восстановленные алгоритмы на предмет поиска в нем уязвимостей. Такой анализ может быть необходим в случае отсутствия открытого исходного кода и применения бинарного в критических с точки зрения безопасности областях.

Как правило, современный глубокий поиск уязвимостей производится вручную, используя ассемблерное представление кода и занимает ощутимое время. В противовес − анализ описания алгоритмов в более понятном специалисту виде (например, таком, как исходный код) происходит значительно быстрее. Поэтому любая автоматизация механизма восстановления увеличила бы общую эффективность (количество найденных уязвимостей относительно времени поиска) нахождения уязвимостей.

Утилиты такого типа (далее − Утилита), автоматизирующие получение описания алгоритмов из ассемблера, работают по общему механизму и схожи с классическим компилятором процедурных языков (таких как *C* и *Pascal*). А одним из залогов успешности последнего (впрочем, как и ряда другого программного обеспечения) является правильная организация собственного внутреннего представления.

Далее будут рассмотрены представления, использовавшиеся в созданном прототипе (далее Прототип) такой Утилиты. Прототип принимает на вход ассемблерный текст программы для процессора *PowerPC* и создает его алгоритмическое представление в виде, подобном программе на языке *C*. Описание метода поиска уязвимостей в коде телекоммуникационных устройств, в котором может быть использована Утилита, подробно изложено в статье автора [1].

В поддержку обусловленности использования именно таких представлений приведем факт работоспособности Прототипа, хотя и лишь на простых примерах ассемблерного кода.

Внутренние представления

Назначением внутреннего представления является такая форма используемых данных, которая удобна для их хранения и обработки в программе. В случае Утилиты, данные должны

быть адаптированы для хранения входной программы (как в виде начального ассемблера, так и конечного описания алгоритма) со следующих позиций:

– деление программы на функции и потоки управления в них;

– используемые переменные (регистры, входные и выходные аргументы функций, временные объекты и т.п.), их значения и время жизни;

– другая специальная информация, которая может быть использована (например, для поиска уязвимостей в восстановленном алгоритме).

Представления, использованные в Прототипе, приведены далее как перечисление их названий и характеристик в виде разделов статьи. Все представления используются для анализа программой с последующей генерацией иных: начальное представление программы в виде ассемблера преобразуется в набор промежуточных, а затем и в конечное – структурированное, оптимизированное и удобное для понимания специалистом.

Представление TextAssembler

Назначение

Хранение программы в виде, соответствующем входному ассемблеру и подходящем для лексического и синтаксического анализа.

Форма

Текст (на языке ассемблера *PowerPC*).

Пример

Ассемблер программы, код которой на языке С приведен в комментариях к инструкциям, содержит следующий текст.

```
max2(){
    0x00000001:  max2:     // { x(r3), y(r4), t(r5)
    0x00000002:  cmpw r3, r4 //   if(x > y)
    0x00000003:  ble label_1    //   {
    0x00000004:  mr r6, r3       //     m = x;
    0x00000005:  b label_2       //   }
```

```
    0x00000006:  label_1:          //   else {
    0x00000007:  mr r6, r4        //     m  = y;
    0x00000008:  label_2:          //   }
    0x00000009:  cmpw r6, r5  //   if(m > z)
    0x0000000A:  ble label_3    //   {
    0x0000000B:  mr r7, r6        //     n = m;
    0x0000000C:  b label_4        //   }
    0x0000000D:  label_3:          //   else {
    0x0000000E:  mr r7, r5        //     n = z;
    0x0000000F:  label_4:          //   }
    0x00000010:  mr r3, r7        //   return n;
    0x00000011:  blr                  // }
}
```

Здесь и далее примеры различных представлений будут относится к данному ассемблеру *PowerPC*, представляющему собой функцию нахождения максимального из двух чисел (далее − Функция). Помимо ассемблера, код в примере содержит заголовок, начало и конец функции в С-стиле, позволяя перенести фазу деления кода по функциям на предварительную фазу (с выполнением достаточно хорошо справляется продукт *IDA Pro*).

Представление AbstractSyntax

Назначение

Предоставление теста ассемблера, близкого человеку, в более формализованном и структурированном виде, доступном для анализа программами. В данном представлении деления на функции не является явным.

Форма

Дерево с узлами, хранящими информацию о токенах и правилах входного языка.

Пример

Дерево представлено в виде списка строк, каждая из которых содержит описание узла элемента дерева, пробельные отступы означают глубину вложенности элемента, а более ранний в тексте элемент с меньшим отступом − родительский узел дерева.

```
IrList()
  IrIdent('max3')
    IrList()
      IrLabel('max2'), name='max2'
      IrBranch('ble')
        IrCond('ble'), kind='?false'
          IrOperation('ble'), kind='.'
            IrReg('cr'), id=32
            IrInteger(''), value=1
        IrLabel('label_1'), name='label_1'
      IrOperation('mr'), kind='='
        IrReg('r5'), id=5
        IrReg('r3'), id=3
      IrBranch('b')
        IrEmpty()
        IrLabel('label_2'), name='label_2'
      IrLabel('label_1'), name='label_1'
      IrOperation('mr'), kind='='
        IrReg('r5'), id=5
        IrReg('r4'), id=4
      IrLabel('label_2'), name='label_2'
      IrOperation('mr'), kind='='
        IrReg('r3'), id=3
        IrReg('r5'), id=5
      IrBranch('blr')
        IrEmpty()
        IrReg('lr'), id=41
```

Как хорошо видно на примере, представлением является структурированное описание основных элементов кода программы (имен функций, регистров, констант, операторов, переходов и т.п.) при отсутствии так называемого «лексического мусора» (пробелы, запятые, неиспользуемые адреса и проч.).

Представление BasicBlock

Назначение

Хранение программы в виде, который позволит разделить задачи анализа инструкций вычисления значений и инструкций перехода.

Форма

Графы базовых блоков функций таких, что каждый блок состоит из последовательности инструкций, имеющую только один вход и один выход. Таким образом, внутри базового блока не может быть инструкций передачи управления (кроме последней). В данном и всех последующих представлениях деление на функции осуществлено.

Пример

Граф для Функции представлен на рис. 1.

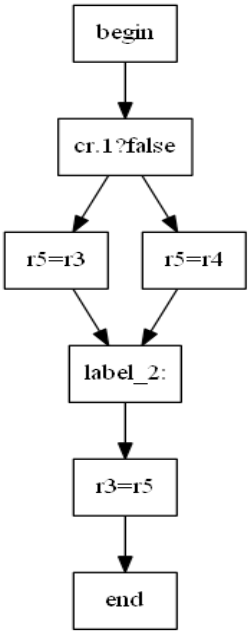

Рисунок 1 – Граф для Функции (представление BasicBlock)

Как хорошо видно на рисунке, представлением Функции является граф, имеющий начало и конец (соответствующий началу и концу самой Функции) и описывающий последовательно выполняемые списки инструкций с переходами между ними.

Представление BasicBlockFramed

Назначение

Более формализованная форма BasicBlock, лучше подходя-

щая для написания сложных алгоритмов обработки графа. Имеет дополнительные узлы − Frame, которые восстанавливают структуру программы даже после ее оптимизации.

Форма

Граф, аналогичный BasicBlock, но имеющий дополнительные узлы − Frame, сохраняющих стандартную для Прототипа топологию представления. Например, узел условного перехода за счет Frame всегда имеет две выходящие связи, даже если одна из них в исходной программе указывала на пустые инструкции.

Пример

Граф для Функции имеет следующие изображение (см. рис. 2).

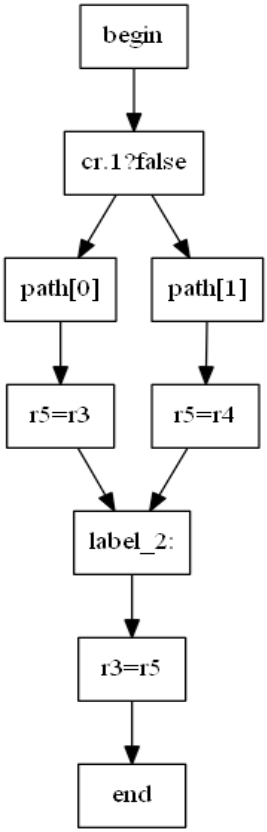

Рисунок 2 – Граф для Функции (представление BasicBlockFramed)

Представление на рис. 2 хотя и подобно BasicBlock, но имеет следующее отличие. Узел условного перехода «cr.1?false» имеет 2 узла-ребенка (то есть, с входящими связями из данного), определяющих две ветки выполнения согласно результату проверки условия: if(cr.1?false == true) then goto «path[0]»; else goto «path[1]»; endif

Представление RegisterValue

Назначение

Хранение информации о значениях регистров и временах их жизни.

В данном контексте, под временем жизни регистра понимается набор инструкций, в рамках которых значение в данном регистре используется для вычисления других значение, хранимых в регистрах с таким же временем жизни. Такое представление применяется для оптимизации восстановленного алгоритма, а именно – замены набора регистров, хранящих одно и тоже числовое значение, на переменную Функции.

Форма

Граф, топологически аналогичен BasicBlockFramed, но его узлы хранят информацию о значениях регистров и временах их жизни. Значение регистра может состоять из набора других значений, например, в точке схождения веток условного перехода – метке ассемблерного кода. Также, в узлах хранится вспомогательная информация, такая как первое и последнее использования значения регистра.

Пример

Граф для Функции представлен на рис. 3.

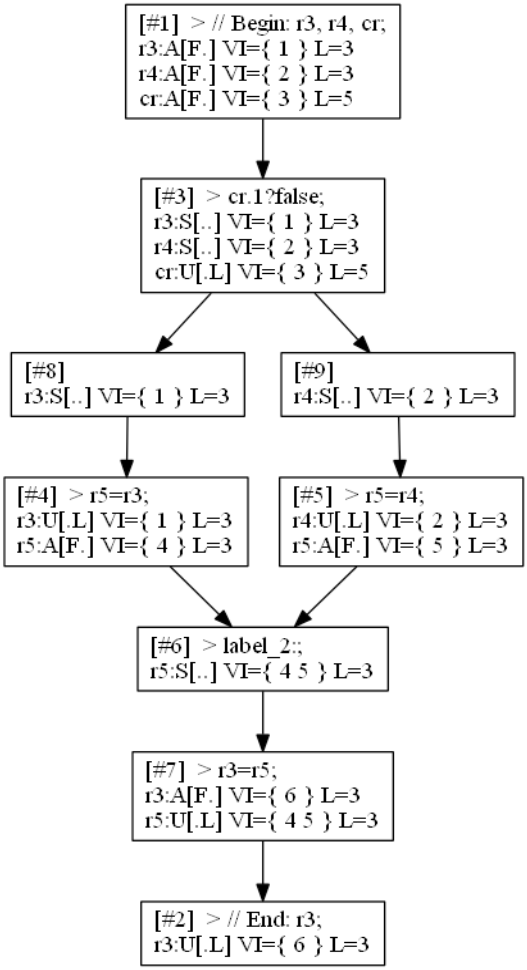

Рисунок 3 – Граф для Функции (представление RegisterValue)

Представление топологически схоже с BasicBlockFramed, однако граф содержит следующую дополнительную информацию:

– r3, r4, r5 – используемые регистры общего назначения;

– cr – регистр, хранящий результаты сравнения значений и используемый для выполнения условных переходов;

– U[], A[], S[] – типы операции над регистром (Usage – использование, Assign – присваивание, Storage – хранение значения);

– [F.], [.L] – битовое поле флагов использования значения регистра (First – первое, Last – последнее использование);

– VI={} – информация о значении регистра в виде набора идентификаторов других значений;

– {1, 2, 3, 4, 5, 6}, {4 5} – наборы уникальных идентификаторов значений регистров;

– L = 3, L = 5 – информация о времени жизни регистра с ее уникальным идентификатором.

Представление ValueInfo

Назначение

Хранение информации о связи между вычисляемыми значениями. Такая информация используется для распределения значений, хранящихся на множестве регистров начального ассемблера, на более компактное множество переменных конечного восстановленного алгоритма. Также представление применяется для оптимизации восстановленного алгоритма, а именно упрощения математических выражений.

Форма

Набор графов, узлами которого являются идентификаторы значений (из представления RegisterValue), а направленными ребрами – связи со значениями, получаемыми из них.

Пример

Граф для Функции имеет следующие изображение (см. рис. 4).

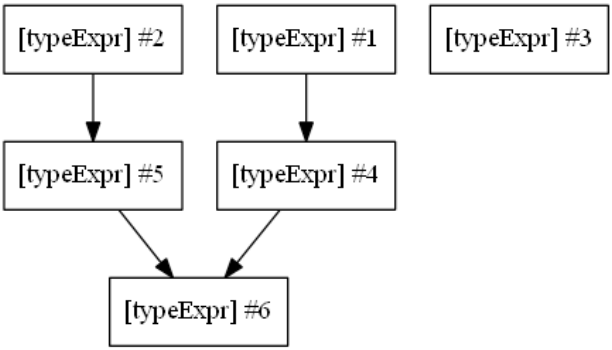

Рисунок 4 – Граф для Функции (представление ValueInfo)

Представление описывает логику вычисления значений и не имеет лишней информации о регистрах, инструкциях и переходах между ними.

Представление TextRestoredAlgorithm

Назначение

Хранение восстановленного алгоритма после структурирования, оптимизации и других действий Прототипа.

Форма

Текст (на С-подобном языке)

Пример

```
max3(arg_0, arg_1, arg_2){
  if(arg_2.1?false){
    loc_0 = arg_1;
  }else{
    loc_0 = arg_0;
  }
  return (loc_0);
}
```

Восстановленный подобным образом алгоритм довольно хорошо описывает логику работы исходного ассемблера тестовой программы. Отметим, что поскольку Прототип находится в процессе разработки, то переменная «arg_2» не является аргументом функции «max3()», а должна инициализироваться результатом сравнения переменных «arg_0» и «arg_1».

Заключение

Приведенные описания представлений и пример работы Прототипа полностью подтверждают право на жизнь выбранных способов хранения и обработки данных. А данная статья в целом может стать отправной точкой как для развития темы восстановления кода (или более модного аналога – реверс-инжиниринга), так и для реализации соответствующих утилит автоматизации.

Список литературы

1. Буйневич М.В., Израилов К.Е. Метод алгоритмизации машинного кода телекоммуникационных устройств. // Телекоммуникации.– 2012.– № 12.– С. 2-6.

УДК 004.051

МЕТОДЫ ПОВЫШЕНИЯ БЫСТРОДЕЙСТВИЯ ПРОГРАММ

Скворцов Юрий Владимирович
аспирант Санкт-Петербургского государственного университета телекоммуникаций имени проф. М.А. Бонч-Бруевича, г. Санкт-Петербург

АННОТАЦИЯ

Представлена история развития центральных процессоров, причины по которым их быстродействие не может повышаться как раньше. Рассмотрены возможности наращивания производительности программ, используя специфические процессорные инструкции, а также даны границы их применимости.

Ключевые слова: процессор; программирование; быстродействие; интринсики.

METHODS TO IMPROVE SOFTWARE PERFORMANCE

Skvortsov Y.V.
Postgraduate student of the Bonch-Bruevich Saint-Petersburg State University of Telecommunications, Saint Petersburg

ABSTRACT

Presents the brief history about CPUs development and the reasons why its performance cannot grow further as it used to grow before. Considered the possibilities to increase software performance using specific processor's instructions and given their scope.

Keywords: processor; programming; performance; intrinsic.

В настоящее время может показаться, что вопрос быстродействия практически исчерпал себя: современные процессоры работают потрясающе быстро, позволяя решать задачи, ещё десятилетие назад требующие на выполнение нескольких часов, за считанные минуты. Но и ожидания пользователей компьютеров также значительно изменились за