Модел. и анализ информ. систем. Т. **21**, № **4** (2014) 5–12

УДК 517.9

Транслятор с возможностью статического анализа безопасности информационных потоков для простого языка программирования

Антошина Е. Ю., Баракова А. Н., Никитин Е. С., Чалый Д. Ю.[1]

Ярославский государственный университет им. П. Г. Демидова
150000 Россия, г. Ярославль, ул. Советская, 14

e-mail: kantoshina@gmail.com, crow_ala@mail.ru, nik.zhenya@gmail.com, chaly@uniyar.ac.ru

получена 28 апреля 2014

Ключевые слова: языковая безопасность, статический анализ, информационный поток

Рассматриваются программы, написанные на while-языке, с переменными двух типов безопасности: секретными и открытыми. Статический анализ безопасности информационных потоков программ идентифицирует небезопасные информационные потоки, через которые могут произойти утечки информации. При таком анализе по правилам, предложенным в [6], определяются типы конфиденциальности выражений, операторов и композиции операторов.

На основе этих правил был разработан алгоритм статического анализа безопасности программ, который заключается в попытке их типизирования. Если в результате программа типизируема, то она является безопасной с точки зрения информационных потоков; если же программе нельзя присвоить тип безопасности, то в ней содержатся небезопасные информационные потоки, и, следовательно, она является небезопасной.

С помощью средств генерации лексических и синтаксических анализаторов flex и bison [5] был разработан транслятор программ, написанных на while-языке, в код машины MMIX [2].

1. Введение

Современные стандартные методики обеспечения безопасности не дают гарантию того, что сквозное поведение (end-to-end behavior) вычислительной системы удовлетворяет такой важной политике безопасности, как конфиденциальность. Сквозная политика конфиденциальности (end-to-end condentiality policy) устанавливает, что атакующий не может узнать никакой информации при помощи наблюдения за изменением значений открытых переменных; эта политика регулирует информационный поток. Информационным потоком от источника к приемнику называется преобразование информации в приемнике, зависящее от информации источника [1].

[1]Работа выполнена при поддержке РФФИ, проект N 14-01-31539.

Обычные механизмы безопасности, такие как управление доступом и шифрование, не касаются напрямую соблюдения безопасности информационного потока. Недавно был разработан новый перспективный подход [6]: использование средств языка программирования для определения и предписания политики информационных потоков. Одним из главных механизмов для реализации этих целей является статический программный анализ, подробно рассматриваемый в данной работе на примере программ, написанных на простом языке программирования.

2. Семантика безопасности программ

В работе рассматриваются программы, написанные на модельном языке, называемом while-языком и являющемся вариантом языка, представленного в [4]. Это простой язык программирования, содержащий операторы присваивания, оператор пропуска skip, не выполняющий никаких действий, условный оператор и оператор цикла while. В контексте нашей задачи рассматриваются переменные двух типов безопасности: секретные и открытые переменные.

В процессе работы программы между переменными различных типов безопасности возникают информационные потоки. Безопасные информационные потоки, через которые не могут произойти утечки информации, обладают свойством невмешательства. Оно означает, что никакие открытые переменные не изменяются в зависимости от значения секретных переменных. Если все информационные потоки программы обладают свойством невмешательства, то программа является безопасной. По существу это означает, что в безопасных программах недопустимы информационные потоки от секретных переменных к открытым.

$$[\text{E1–2}] \quad \vdash exp : high \quad \frac{h \notin Vars(exp)}{\vdash exp : low}$$

$$[\text{C1–3}] \quad [pc] \vdash skip \quad [pc] \vdash h := exp \quad \frac{\vdash exp : low}{[low] \vdash l := exp}$$

$$[\text{C4–5}] \quad \frac{[pc] \vdash C_1 \quad [pc] \vdash C_2}{[pc] \vdash C_1; C_2} \quad \frac{\vdash exp : pc \qquad [pc] \vdash C}{[pc] \vdash while\ exp\ do\ C}$$

$$[\text{C6–7}] \quad \frac{\vdash exp : pc \qquad [pc] \vdash C_1 \qquad [pc] \vdash C_2}{[pc] \vdash if\ exp\ then\ C_1\ else\ C_2} \quad \frac{[high] \vdash C}{[low] \vdash C}$$

Рис. 1. Семантические правила задания типов безопасности

Для идентификации небезопасных информационных потоков и программ используется статический анализ. В нем у каждой переменной, у каждого выражения, у операторов и у программы в целом определяется тип безопасности. Использующи-

еся статическим анализом правила задания типов безопасности были предложены в [6] и представлены на рис. 1.

Программа 1.

```
h:=1;
l:=2;
if(h>0) then
    h:=h+l
else skip;
```

Программа 2.

```
h:=1;
l:=h+1;
if(h>0) then
    h:=h+l
else skip;
```

Программа 3.

```
l:=1;
h:=1;
while (h>0) {
    l:=l+1;
    h:=h-2;
}
```

Рис. 2. Примеры безопасных и небезопасных программ на while-языке (через `l` обозначаются открытые переменные, а через `h` — секретные)

Рассмотрим программы, представленные на рис. 2, с точки зрения безопасности информационных потоков. Первая программа безопасна, так как в ней формирование значений открытых переменных происходит вне зависимости от значений секретных переменных. Во второй программе видна прямая зависимость значения открытой переменной от значения секретной переменной: присваивание открытой переменной l значения $h{+}1$ содержит информационный поток от секретной переменной к открытой, что говорит о несоблюдении политики невмешательства, поэтому программа 2 небезопасна. Рассмотрим программу 3. В ней нет прямых информационных потоков от секретных переменных к открытым, но если рассмотреть условие и тело цикла, то можно увидеть, что в зависимости от значения секретной переменной меняется значение открытой переменной, поэтому присваивание $l := l + 1$ в зависимости от значения переменной h также является нарушением политики невмешательства, поэтому программа 3 не является безопасной с точки зрения информационных потоков.

3. Алгоритм типизации программы

Представленные правила типизации операторов позволили разработать алгоритм типизации программы в целом. Правила [E1–2] используются для типизации выражения, правила [C1], [C2–3], [C5] и [C6] – это правила типизации операторов пропуска, присваивания, цикла и условного оператора соответственно, правило [C4] – это правило типизации композиции операторов, а правило [C7] – это правило понижения уровня конфиденциальности, которое используется для типизации операторов и их композиций.

Данный алгоритм описывает последовательность действий для типизации программы. В нем рассматриваются все возможные случаи сочетаний типов выражений и операторов.

1. Если программой является операция присваивания $p := exp$:

    1) Определить тип безопасности переменной p.

2) Определить тип безопасности выражения exp:
   - $high$ – если в нем есть переменные типа $high$;
   - $[high, low]$ – если в нем нет переменных типа $high$, так как семантические правила из п. 2. позволяют осуществить вывод обоих типов.

3) Типизировать программу $p := exp$:
   - $[high, low]$ – если p имеет тип $high$;
   - $[low]$ – если p имеет тип low и exp имеет тип $high, low$;
   - $[\times]$ – не типизируем, если p имеет тип low, а exp имеет тип $high$.

2. Если программой является операция пропуска $skip$, то ей присваивается тип безопасности $[high, low]$.

3. Если программой является оператор цикла вида $while\ exp\ do\ C$:

   1) Определить тип безопасности выражения exp:
      - $high$ – если в нем есть переменные типа $high$;
      - $high, low$ – если в нем нет переменных типа $high$.

   2) Определить тип безопасности тела цикла C в соответствии с данным алгоритмом.

   3) Типизировать программу $while\ exp\ do\ C$:
      - $[high, low]$ – если exp имеет тип $high$ или $high, low$, а C имеет тип $[high, low]$;
      - $[low]$ – если exp имеет тип $high, low$, а C имеет тип $[low]$;
      - $[\times]$ – не типизируем, если exp имеет тип $high$, а C имеет тип low, или если C не типизируема.

4. Если программой является условный оператор вида $if\ exp\ then\ C_1\ else\ C_2$:

   1) Определить тип безопасности выражения exp:
      - $high$ – если в нем есть переменные типа $high$;
      - $high, low$ – если в нем нет переменных типа $high$.

   2) Типизировать отдельно C_1 и C_2 в соответствии с данным алгоритмом.

   3) Типизировать программу $if\ exp\ then\ C_1\ else\ C_2$:
      - $[high, low]$ – если exp имеет тип $high$ или $high, low$, и C_1 и C_2 имеют тип $[high, low]$;
      - $[low]$ – если exp имеет тип $high, low$, и хотя бы одна из C_1 или C_2 имеет тип $[low]$;
      - $[\times]$ – не типизируема, если exp имеет тип $high$ и хотя бы одна из C_1 или C_2 имеет тип $[low]$, или если C_1 или C_2 не типизируема.

5. Если программа является композицией операторов $\{C1; C2\}$:

   1) Взять за $C1$ все операторы, кроме последнего, а за $C2$ – последний оператор программы.

2) Типизировать отдельно подпрограммы $C1$ и $C2$ в соответствии с данным алгоритмом.

3) Типизировать композицию $\{C1; C2\}$:

   - $[high, low]$ – если и $C1$, и $C2$ имеют тип $[high]$ или $[high, low]$, или одна из подпрограмм типа $[high]$, а вторая $[high, low]$;
   - $[low]$ – если $C1$ или $C2$ имеет тип $[low]$, а вторая типизирована;
   - $[\times]$ – не типизируема, если хотя бы одна из подпрограмм $C1$ или $C2$ не типизируема.

Если программа была типизирована в результате работы алгоритма, то она является безопасной. Если программа не типизируема, то в ней существуют информационные потоки, не обладающие свойством невмешательства, и, следовательно, она не безопасна.

4. Алгоритм работы транслятора

Для реализации предложенного в п. 3. алгоритма был разработан транслятор while-языка. От оригинального языка [4] его отличает введение типов безопасности для объявляемых переменных. Транслятор осуществляет преобразование программы на while-языке в код машины MMIX [2] и производит ее статический анализ с целью обнаружения небезопасных информационных потоков.

Задача трансляции является хорошо изученной областью информатики (см. [3]) и ее решение заключается в построении двух взаимодействующих компонентов — лексического и синтаксического анализатора. Современные программные средства позволяют автоматизировать процесс генерации кода транслятора, однако требуют определения лексических и синтаксических конструкций языка программирования. Нами был разработан набор регулярных выражений, описывающих токены нашего языка программирования (например, константы, идентификаторы и т.п.), а также LALR(1)-грамматика [3], задающая синтаксические конструкции.

Реализация транслятора была выполнена при помощи программных средств flex и bison [5]. Программа flex на основе шаблонов токенов генерирует код лексического анализатора на языке C, который тесно интегрируется со сгенерированным кодом синтаксического анализатора, созданным программой bison. Входные данные программы bison похожи на формализм атрибутных грамматик [3], в котором продукции может соответствовать исполняемый код. Этот код выполняется, если при построении дерева разбора применяется данная продукция.

4.1. Анализ безопасности программ

Анализ безопасности происходит при синтаксическом анализе программы. Правила для анализа безопасности были реализованы для каждой продукции грамматики. В разработанной грамматике есть два типа безопасности переменных: высокий (high) и низкий (low). По умолчанию (если не указан тип безопасности) переменным присваивается тип безопасности low.

```
// Типы безопасности:
// 0 – нетипизируемо, 1 – low, 2 – high, 3 – [high, low]
statement:
  SKIP { $$.security = 3; }                          // [high, low]
| ID ASSIGN rexpression {
    if($1.security == 2) $$.security = 3;            // [high, low]
    if($1.security == 1) {
        if($3.security == 2) $$.security = 0;        // нетипизируемо
        else $$.security = 1;                        // [low]
    }
};
```

Рис. 3. Фрагмент кода, реализующий анализ безопасности оператора skip и оператора присваивания

Пример, который демонстрирует принцип реализации алгоритма, показан на рис. 3. Нотация \$1, \$2 и т.д. в коде на языке C соответствует символам в правой части продукции в порядке их следования, а переменная \$\$ — нетерминалу *statement* из головы продукции. Каждое такое обозначение соответствует множеству вычисляемых атрибутов. Целью вычислений является атрибут *security* у оператора в целом.

По алгоритму из п. 3. тип безопасности операции skip — [high, low]. В коде этому типу соответствует константа 3 (0 – выражение не типизируемо, 1 – low, 2 – high). Тип безопасности оператора присваивания также вычисляется согласно алгоритму. Если тип безопасности переменной с именем, соответствующим ID, равен $high$, то все выражение может быть типизировано и как $high$, и как low; если тип безопасности переменной low, то возможны два варианта: если нетерминал $rexpression$ типизируется как $high$, то все выражение присваивания не типизируемо, в противном случае выражение присваивания типизируемо как low.

4.2. Генерация кода на языке машины MMIX

В качестве целевого языка был выбран язык ассемблера MMIXAL, который выполняется на 64-битной RISC-архитектуре MMIX [2], которая схожа с архитектурой большинства современных систем.

Для хранения и обработки данных архитектура MMIX имеет 2^{64} ячеек памяти, 256 регистров общего назначения и 32 специальных регистра. Вся память машины MMIX делится на две части, которые называются пользовательским пространством (пространство программы) и пространством ядра (пространство операционной системы). Пользовательское пространство далее делится на четыре сегмента по 2^{61} байтов каждый: текстовый сегмент, сегмент данных, сегмент пула и сегмент стека.

Для решения задачи хранения данных с разными атрибутами безопасности сегмент данных разбивается на два сегмента. Первый сегмент будет служить для хранения открытых данных, второй сегмент — для хранения секретных данных. При помощи такого подхода мы изолируем область памяти с высоким уровнем безопас-

ности от области памяти с низким уровнем безопасности. Физически обеспечить безопасность области памяти с высоким уровнем безопасности можно, например, при помощи шифрования.

При трансляции кода нашего языка программирования на язык MMIXAL происходит замена каждой команды на соответствующую ей последовательность команд MMIXAL. Вычисления организуются при помощи стеков, которые организуются в каждом сегменте памяти с дном в начале сегмента. Наша реализация языка допускает вложенные области видимости, что реализуется динамически: память под глобальные переменные выделяется внизу стека, а вход в каждую новую область видимости создает новый элемент стека, содержащий выделенную память под локальные переменные, выход из области видимости удаляет такой элемент из стека.

Для обработки данных используются регистры, которые необходимо рационально использовать. Архитектура MMIX имеет достаточное количество регистров общего назначения, что даёт возможность организации стека для обработки данных. В процессе создания транслятора был разработан следующий алгоритм:

1. В начале работы программы дно стека помещается в первый регистр.

2. В процессе выполнения последовательности операций, приводящих к получению данных, данные кладутся на вершину стека.

3. Унарные операции производятся над первым элементом на вершине стека.

4. Бинарные операции производятся над первым и вторым элементом на вершине стека.

5. При утрате необходимости в данных на вершине стека данные выталкиваются из стека.

В итоге на основе фрагментов кода MMIXAL для отдельных операторов нашего языка программирования собирается код для всей программы в целом.

5. Заключение

В статье рассмотрена задача анализа безопасности информационных потоков для простого языка программирования, приводится алгоритм, позволяющий обнаруживать утечки информации из секретных переменных в открытые. На основе проведенных теоретических исследований был разработан транслятор для этого языка программирования, который доступен по адресу [7].

Список литературы

1. *Девянин П. Н.* Модели безопасности компьютерных систем: Учебное пособие для студентов высших учебных заведений. М.: Академия, 2005. 144 с. [*Devyanin P. N.* Modeli bezopasnosti kompjuternyx sistem: Uchebnoe posobie dlja studentov vysshix uchebnyx zavedenii. M.: Akademija, 2005. 144 p. (in Russian)].

2. *Кнут Д.* Искусство программирования. MMIX — RISC-компьютер для нового тысячелетия. Т. 1, вып. 1. М.: Вильямс. 160 с. (English ed.: *Knuth D. E.* The Art of Computer Programming. MMIX — A RISC Computer for the New Millenium. Vol. 1, Fascile 1. Addison-Wesley Professional, 2005. 144 p.)

3. *Grune D., Jacobs C.J.H.* Parsing Techniques. A Practical Guide. 2nd ed. Springer, 2008. 664 p.

4. *Hoare C.A.R.* An Axiomatic Basis for Computer Programming // Communications of the ACM. 1969. Vol. 12, Issue 10. P. 576–580.

5. *Levine J.* flex & bison. O'Reilly Media. 2009. 292 p.

6. *Sabelfeld A., Myers A.C.* Language-Based Information-Flow Security // IEEE Journal on Selected Areas in Communications. 2003. Vol. 21. P. 5–19.

7. *SWHILE — while-language translator with security types* // WWW: https://bitbucket.org/kafti/swhile. Дата доступа: 22.04.2014.

A Translator with a Security Static Analysis Feature of an Information Flow for a Simple Programming Language

Antoshina E. Ju., Barakova A. N., Nikitin E. S., Chalyy D. Ju.

P.G. Demidov Yaroslavl State University, Sovetskaya str., 14, Yaroslavl, 150000, Russia

Keywords: language security, static analysis, information flow

We consider while-language programs with variables of two security types: low and high. Security static analysis of information flows of such programs identifies insecure information flows which can cause leaks. Semantic rules of such an analysis which was proposed in [6] assign security types for expressions, operators and compositions of operators. We use these rules to propose an algorithm of security static analysis to discover a security type of the program under consideration. If such a type can be assigned, information flows of the program are secure; otherwise, it contains insecure information flows. We have used flex and bison [5] tools to implement a translator for a while-language into the MMIX computer [2] instruction sequence.

Сведения об авторах:

Антошина Екатерина Юрьевна,
Ярославский государственный университет им. П.Г. Демидова,
магистрант

Баракова Александра Николаевна,
Ярославский государственный университет им. П.Г. Демидова,
магистрант

Никитин Евгений Сергеевич,
Ярославский государственный университет им. П.Г. Демидова,
студент

Чалый Дмитрий Юрьевич,
Ярославский государственный университет им. П.Г. Демидова,
доцент