ИСП РАН

УПРАВЛЯЕМОЕ ПОСТРОЕНИЕ ПРЕДИКАТА ПУТИ ПРИ СИМВОЛЬНОЙ ИНТЕРПРЕТАЦИИ ТРАСС БИНАРНОГО КОДА

Алексей Вишняков
vishnya@ispras.ru

Москва, 29 мая 2017 г.

Символьное выполнение

- Символьное выполнение – процесс интерпретации программы, где конкретным значениям переменных сопоставляются символьные переменные, которые могут принимать произвольные значения

- Преобразования данных в программе описываются в виде формул над символьными переменными и константами

- Каждое условное ветвление, условие выполнения которого зависит от символьных переменных, порождает уравнение, описывающее прохождение потока управления по определенной ветке

Предикат пути

- Символьные переменные ставятся в соответствие входным данным программы
- Исследуется конкретный путь выполнения в программе
- Предикат пути – система уравнений, построенная в результате символьной интерпретации этого пути
- Предикат пути подается на вход SMT-решателю
- Решением системы уравнений является набор входных данных, обеспечивающий прохождение потока управления по исследуемому пути

Предикат пути

```
int a, b, c;
scanf(“%d %d %d”, &a, &b, &c);
int r = a;
if (a < 0) {
    r = -r;
}
if (b + c < 0) {
    r -= b + c;
} else {
    r += b + c;
}
printf(“|a| + |b + c| = %d\n”, r);
```

r → R, R = A

П = A >= 0

П = (B + C < 0) ∧ (A >= 0)

R = A – (B + C)

Входные данные:

a = 1, b = 2, c = -3

П – предикат пути

Сопоставление символьным переменным:

a → A, b → B, c → C

Трасса машинных инструкций

- В полносистемном эмуляторе запускается программа, на вход которой подается буфер фиксированного размера
- Снимается трасса выполнения, которая содержит:
  - Выполненные инструкции
  - Значения регистров
  - Информацию о прерываниях и взаимодействии с периферийными устройствами
- Шаг трассы соответствует одной выполненной инструкции

Построение предиката пути

- Буфер с входными данными и начальный шаг трассы может задаваться явно аналитиком или определяться автоматизированно по описанию функции, считывающей входные данные
- Для каждого байта буфера с входными данными создается символьная переменная
- Байты входного буфера помечаются на начальном шаге и производится распространение пометок по трассе
  - Отбираются инструкции, обрабатывающие помеченные данные

Построение предиката пути

- Отобранные инструкции транслируются в промежуточное представление
- Промежуточное представление интерпретируется для выявления выполненных условных переходов в промежуточном представлении инструкции
- Производится символьная интерпретация промежуточного представления
- В результате символьной интерпретации всех отобранных инструкций будет построен предикат пути

Актуальность

- Описанному методу построения предиката пути свойственны проблемы
- Недостаточная помеченность – это ситуация, когда значение не помечено, несмотря на то что оно зависит от входных данных
  - Происходит из-за конкретизации символьных адресов значениями из трассы
  - Входные данные, полученные в результате решения предиката пути, не проведут программу по тому же самому пути выполнения, для которого строился предикат пути
- Избыточная помеченность
  - В процессе построения предиката пути возникает неминуемый рост количества помеченных инструкций, не все из которых оказывают существенное влияние на ход анализа (например, функция malloc)

Постановка задачи

Необходимо разработать и реализовать метод управляемого построения предиката пути для борьбы с проблемами недостаточной помеченности и избыточной помеченности.

Метод должен позволять:

- Добавлять ограничения на предикат пути для борьбы с недостаточной помеченностью
- Пропускать неинтересные для анализа функции для борьбы с избыточной помеченностью

Ограничения и список неинтересных для анализа функций должны задаваться на основе априорных знаний о программе.

Обзор известных решений (недостаточная помеченность)

- Два основных подхода к обработке символьных адресов:
  - Конкретизация адреса (SAGE)
  - Работа с адресом как с полностью символьным
- Индексная модель памяти (Mayhem)
  - Определяются границы, в которых лежат значения символьного индекса, при помощи бинарного поиска
  - Загрузка по символьному индексу представляется в виде бинарного дерева поиска
    - $IST(E) = ite(i < addr(E_{right}), IST(E_{left}), IST(E_{right}))$
  - Листья объединяются по принципу линейной зависимости значения (v_i) от индекса (i_i): $v_i = \alpha i_i + \beta$
- S^2E
  - Определяется интервал, которому могут принадлежать значения символьного указателя (бинарный поиск)
  - Создается n состояний, каждое из которых имеет одно фиксированное значение указателя, удовлетворяющее текущему предикату пути

Предлагаемые решения

- В данной работе используется конкретизация символьных адресов
- Предлагается альтернативное решение проблемы символьных адресов путем добавления ограничений на символьные переменные
- Для борьбы с избыточной помеченностью предлагается пропускать анализ функций, не оказывающих существенного влияния на построение предиката пути

Формальное описание функций и их параметров

- Используются модели функций, которые содержат следующие атрибуты:
  - Имя функции
  - Имя модуля, в котором располагается функция
  - Список описаний входных и выходных параметров функции
    - Имя параметра
    - Элемент адресного пространства (регистр, стековый параметр, буфер памяти), содержащий его значение или адрес нуль-терминированной строки

Преодоление недостаточной помеченности

- Предикат пути дополняется недостающими ограничениями на символьные переменные:
  - Ограничения на шаге трассы
  - Ограничения на инструкции по определенному адресу
  - Ограничения на параметр модели функции
- Программа faad из 32-х битного дистрибутива Debian 8.3.0
  - Функция vsscanf проверяет наличие в буфере-источнике терминальных символов, при обнаружении которых копирование заканчивается
  - Были добавлены необходимые ограничения (отсутствие пробелов) на параметр буфера-источника str функции vsscanf
    - $\forall i \in [0, \texttt{\#str}] : \texttt{v(\&str + i, 1) != 0x20}$, где `#str` — длина строки, а `&str` — ее адрес

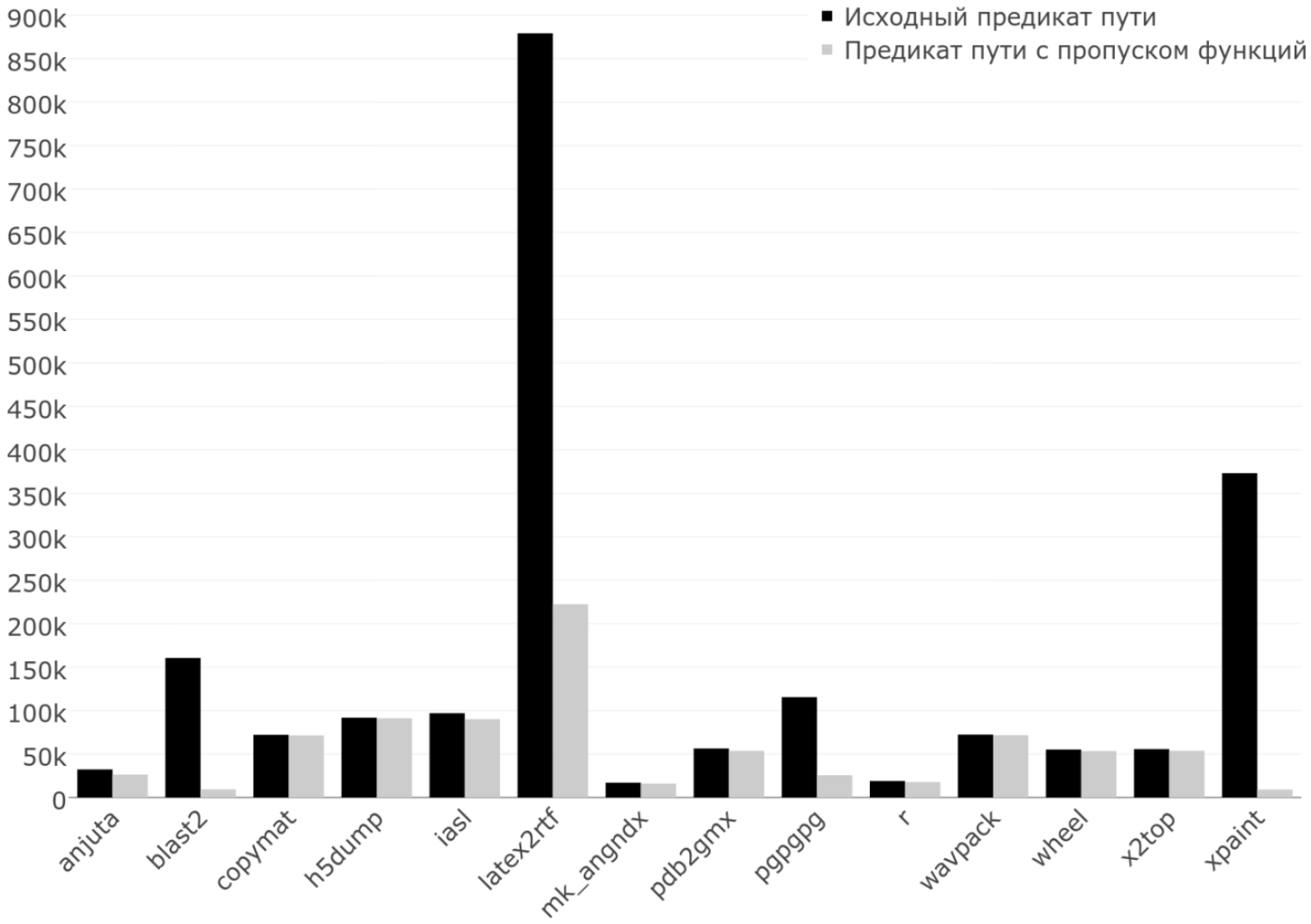
Преодоление избыточной помеченности

- Функции работы с кучей часто используются в программах, но их работа не оказывает влияния на построение предиката пути
  - Распространена ситуация, когда параметры этих функций оказываются помеченными
- При анализе трасс выполнения программы на буфере фиксированного размера часто не требуется учитывать символьную длину буфера
  - Обработка функций вычисления длины строки приводит к тому, что вычисляемая длина исходного буфера становится символьной
- Указывается модель функции и параметры, с которых необходимо снять пометки в процессе символьной интерпретации

Результаты

- Измерения производились на 32 программах из 32-х битного дистрибутива Debian 6.0.10
- Во время построения предиката пути пометки снимались с параметров функций работы с кучей free, malloc и calloc
- Количество отобранных инструкций уменьшилось для 14 программ

Результаты


Исходный предикат пути
Предикат пути с пропуском функций
900k
850k
800k
750k
700k
650k
600k
550k
500k
450k
400k
350k
300k
250k
200k
150k
100k
50k
0
anjuta
blast2
copymat
h5dump
iasl
latex2rtf
mk_angndx
pdb2gmx
pgpgpg
r
wavpack
wheel
x2top
xpaint

Результаты

Программа	Число инструкций без пропуска функций	Число инструкций с пропуском функций	Число пропущенных инструкций	Доля пропущенных инструкций
anjuta	32349	26392	5957	0.18
blast2	160382	9368	151014	0.94
copymat	72124	71494	630	0.01
h5dump	91806	91049	757	0.01
iasl	97037	89973	7064	0.07
latex2rtf	879249	222309	656940	0.75
mk_angndx	17092	15877	1215	0.07
pdb2gmx	56286	53753	2533	0.05
pgpgpg	115539	25622	89917	0.78
r	18870	17766	1104	0.06
wavpack	72306	71672	634	0.01
wheel	55032	53500	1532	0.03
x2top	55716	53674	2042	0.04
xpaint	373192	9219	363973	0.98
Среднее				**0.28**

Спасибо за внимание