Об одном подходе к анализу строк в языке C

для поиска переполнения буфера

Дудина И. А., Малышев Н. Е.

22 ноября 2018 г.

Институт системного программирования им. В. П. Иванникова РАН

Московский государственный университет им. М. В. Ломоносова

Поиск переполнения с помощью статического анализа

Статический анализ — анализ кода программы без её запуска (в частности, для обнаружения дефектов).

Переполнение буфера — обращение к буферу по индексу, выходящему за границы размера буфера.

Задача поиска переполнения алгоритмически неразрешима, поэтому любой инструмент выдаёт ложные предупреждения и/или пропускает некоторые ошибки.

Переполнения в строках языка C

Строка — последовательность символов, заканчивающаяся нуль-терминатором.

'i'	's'	'p'	'r'	'a'	's'	' '	'o'	'p'	'e'	'n'	'\0'

Строки — важный частный случай буферов в языке C. Переполнения при работе со строками чаще всего возникают при работе с библиотечными функциями обработки строк.

```
strcpy    strncpy    strlcpy
strcat    strncat    wcscpy
strlen    strnlen    ...
```

Анализ строк языка C

Мы предложили подход для поиска переполнений буфера в массивах с известным в момент компиляции размером, чувствительный к контексту и путям выполнения. Подход использует метод статического символьного исполнения, масштабируется до анализа миллионов строк кода и показывает высокий уровень истинных срабатываний.

Идея подхода заключается в анализе длины буферов и индексов доступа к ним с помощью отслеживания целочисленных операций и условий их выполнения.

В работе показывается, как можно расширить предложенный подход для поиска переполнений в строках, дополнительно анализируя длину строки.

Символьное исполнение с объединением состояний

Пусть:

- V — множество переменных программы,
- S — множество символьных переменных анализа (например: $\alpha,\ \beta,\ \gamma,\ ...$),
- *SE* — множество символьных выражений (например: $\alpha + \beta,\ \gamma >> 4,\ ...$),
- *Cond* — множество условий над символьными выражениями (например: $\alpha + \delta \neq \beta,\ \gamma \leqslant 7,\ ...$).

Абстрактное состояние в точке q:

- $\sigma : V \to SE$ — символьное хранилище,
- $\pi \in Cond$ — предикат достижимости точки,
- $VS : SE \to Summary$ — информация о целочисленных значениях на различных путях программы, используемая для обнаружения переполнения.

Поиск переполнения буфера в рамках одной функции

```
1  #define S 10
2  int buf[S];
3  int foo (int a, int b) {
4    if (a >= S-1) {
5      // ...
6    }
7    if (b)
8      a++;
9    return buf[a]; //BOF here
10 }
```

Ошибочный путь:

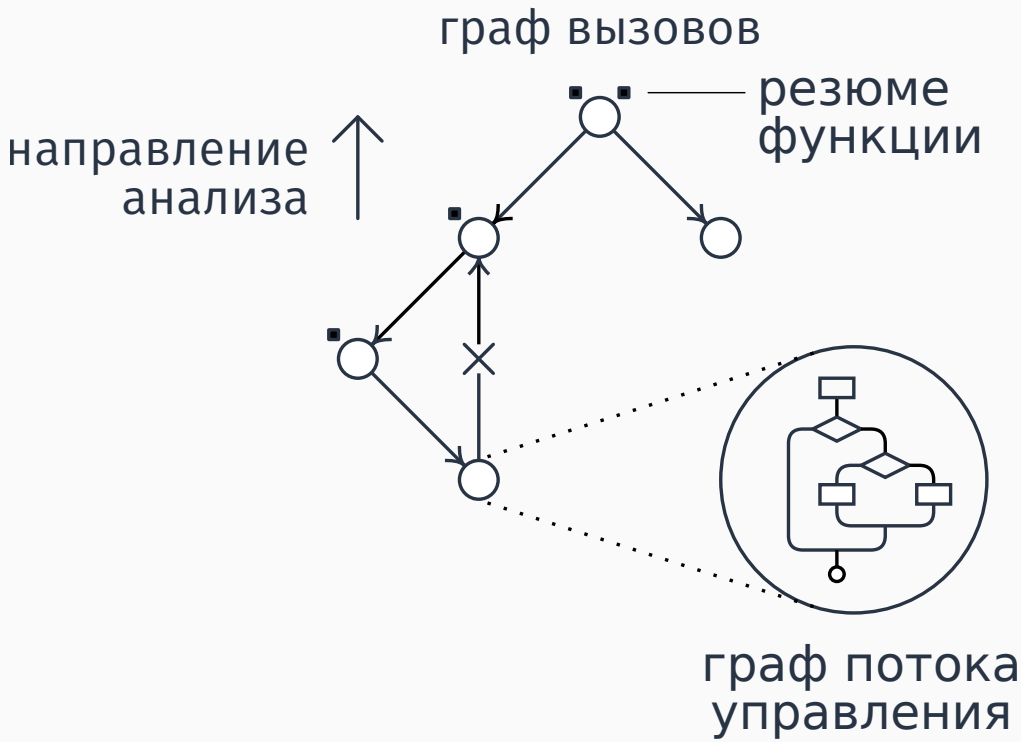
- выполним хотя бы для одного набора входных данных,
- для любого набора входных данных этот путь либо невыполним, либо содержит ошибку.

3-4-5-7-8-9 ошибочный!

- Условие достижимости
  $\pi(p_5) = a_3 \geq S - 1, \qquad \pi(p_7) = \top.$
- Необходимо для точки p_{access} построить достаточное условие того, что управление пришло в p_{access} по ошибочному пути.
  $ErrCond(p_9) = (a_9 \geq S) \wedge ((a_9 = a_3 + 1) \wedge (b \neq 0)) \wedge (a_3 \geq S - 1)$
- Если $ErrCond(p_{access}) \wedge \pi(p_{access})$ совместно, то ошибочный путь существует.
  Модель $a_3 = 9, a_9 = 10, b = 42$ определяет ошибочный путь.

Межпроцедурный анализ

1. Построение графа вызовов.
2. Разрыв циклов.
3. Анализ функций в обратном топологическом порядке.
4. В результате анализа функции строится т. н. «резюме», которое используется в точках вызова этой функции.



Выбор абстракции

Обычных анализов размера массива и величины индекса в операциях доступа к буферу в случае строк недостаточно $\Rightarrow$ необходимо выбрать абстракцию для содержимого строки.

Чем более подробная абстракция используется, тем точнее результаты анализа, но в то же время сильнее растёт размер абстрактного состояния и требуется больше ресурсов на его вычисление.

Слишком общая абстракция не позволяет вычислить адекватное приближение свойств программы.

В данной работе в качестве абстракции содержимого строки выбрана её длина.

Расширение абстрактного состояния для работы со строками

Пусть P — множество переменных (указателей), которые могут хранить адрес строки (с точки зрения языка C).

$Slen : P \to SE$ — отображение, хранящее длины строк в виде символьных выражений.

Передаточные функции обновляются с учётом нового отображения:

`strncpy(dst, src, n)`

$$\text{STRNCPY}\ \frac{\begin{array}{c} Slen_{in} \vdash \texttt{src} \to l_s \quad Slen_{in} \vdash \texttt{dst} \to l_d \quad \sigma \vdash \texttt{n} \to v_n \\ \pi' = (v_n > l_s \ \wedge\ l_r = l_s) \vee \\ \vee (v_n \leqslant l_s \ \wedge\ v_n \leqslant l_d \ \wedge\ l_r = l_d) \vee \\ \vee (v_n \leqslant l_s \ \wedge\ v_n > l_d \ \wedge\ l_r \geqslant v_n) \end{array}}{Slen_{out} = Slen_{in}\{\texttt{dst} \mapsto l_r\} \quad \pi_{out} = \pi_{in} \wedge \pi'}$$

Расширение отображения VS при работе со строками

```
1 void foo (int c, int n) {
2     char s[100] = "", dst[10]= "";
3     if (c)
4         strcpy(s, "very␣very␣long␣string");
5     if (n > 15)
6         //...
7     strncat(dst, s, n);
8 }
```

$$\dfrac{\dfrac{c \neq 0 \qquad \dfrac{21 > 10}{strlen_4(\text{"very..."}) \geqslant 10}}{strlen_7(s) \geqslant 10} \qquad \dfrac{n > 15 \qquad 15 > 10}{n \geqslant 10}}{\texttt{переполнение в функции strncat}}$$

Модель для переполнения: $n = 16, \quad c = 42.$

Ошибочный путь: 2-3-4-5-6-7

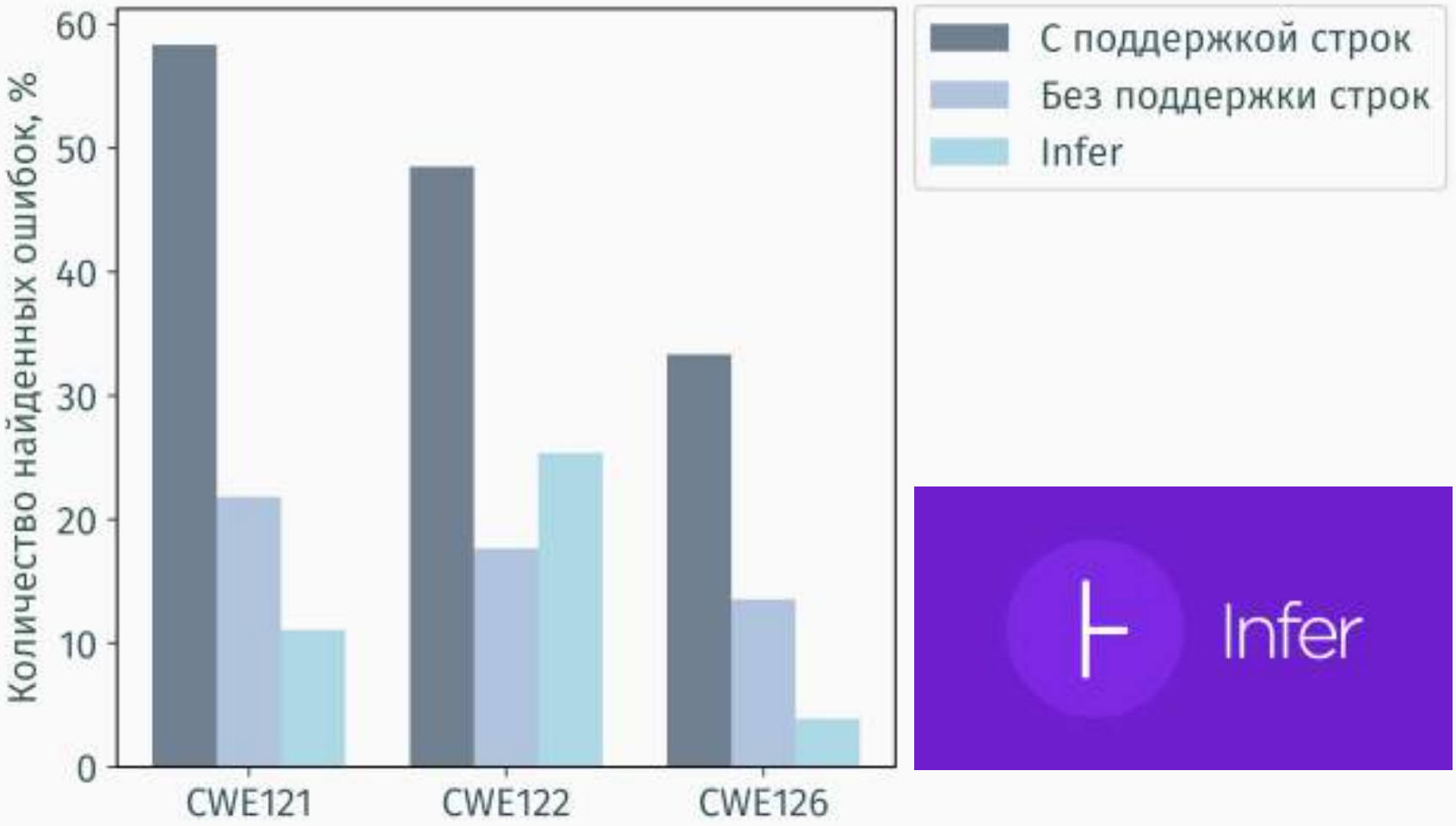
Реализация и тестирование

Описанные подходы были реализованы в рамках инфраструктуры статического анализа Svace.

Для тестирования был использован набор синтетических тестов Juliet Test Suite C/C++, созданный в NSA's Center for Assured Software.

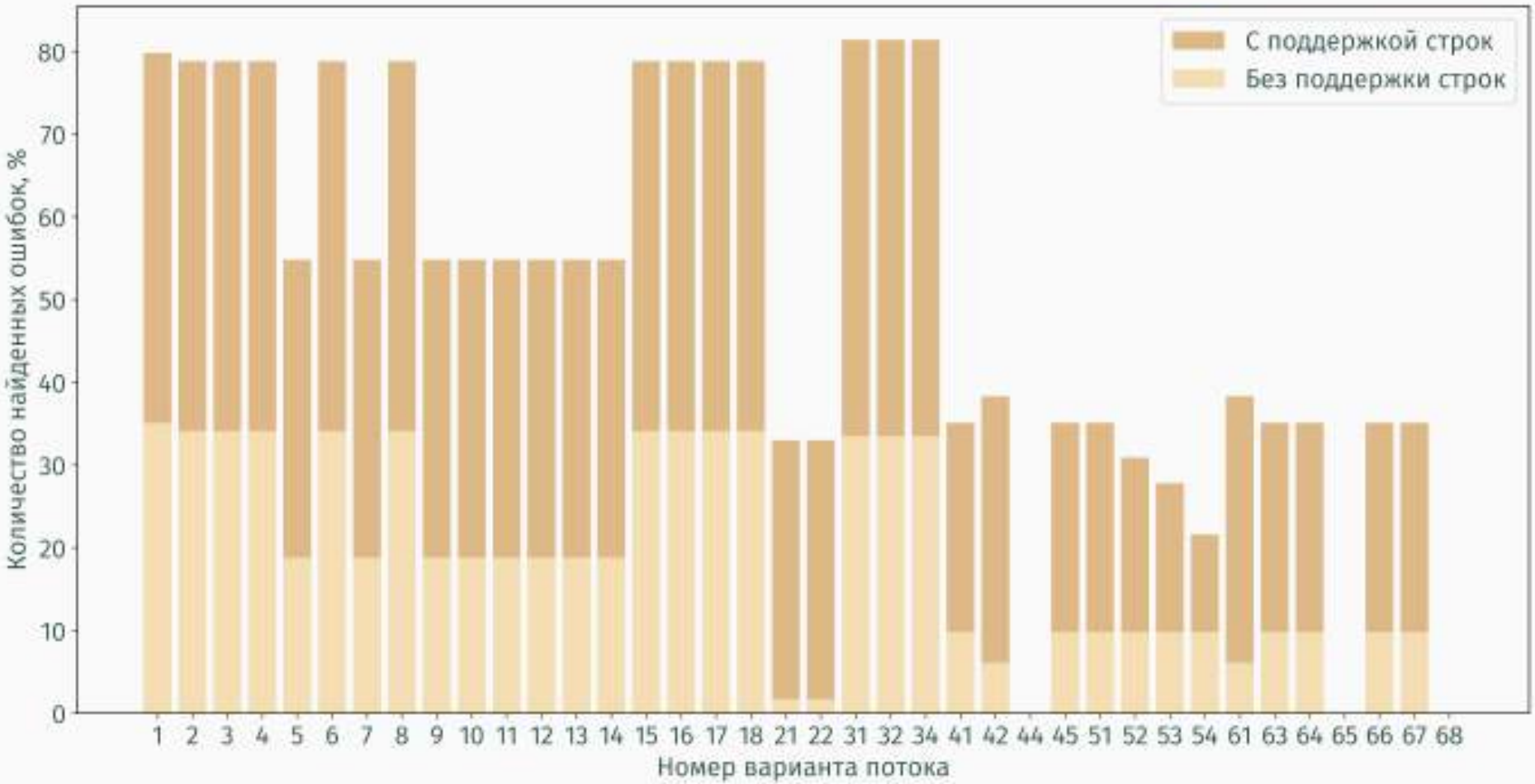
Для сравнения был использован статический анализатор Infer, разрабатываемый компанией Facebook.

Результаты: сравнение по CWE



Juliet Test Suite C/C++ v1.3

Результаты: сравнение по вариантам потока



Результаты: сравнение по типам дефектов

	Число функций	Svace		Infer
		строки		
		+	−	
Memcpy	1 294	63,1	33,9	23,8
Memmove	1 258	64,5	34,9	24,5
Loop	1 222	22,9	16,5	23,7
Cpy, Cat, Ncat	1 532	65,7	0,0	0,0
Ncpy	652	56,7	15,0	17,8
Other	1 350	20,1	18,4	6,7

Сравнение результатов различных версий Svace и анализатора Infer на различных типах дефектов

Результаты на Android 5.0.2

Включение поддержки строк не замедлило анализ даже на большом проекте.

Было выдано шесть истинных предупреждений, связанных с переполнением буфера при вызове обёртки над функцией strcpy.

Пропало несколько ложных срабатываний других детекторов, использующих движок символьного исполнения, вследствие уточнения предиката π условиями на длины строк.

Выводы и дальнейшая работа

Отслеживание длин строк может улучшить качество анализа, не ухудшив показателей масштабируемости инструмента.

Наличие подобного анализа может быть полезно не только для детектора переполнения буфера.

Аналогичный подход может быть применён для отслеживания размеров коллекций, например в языках C++, Java.

Для более точного анализа необходим анализ содержимого строк. В контексте символьного исполнения интерес представляют активно развивающиеся решатели с поддержкой теории строк (z3str).