Поиск ошибок целочисленного переполнения методами динамической символьной интерпретации

Кобрин Илай Александрович

27 мая 2022 г.

МГУ им. М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра системного программирования

Актуальность

Программное обеспечение стремительно развивается, принося с собой множество программных ошибок, поэтому необходимо его тестировать.

Программист самостоятельно может тестировать ПО только на ограниченном наборе тестов, поэтому необходимо разрабатывать методы автоматического тестирования.

Одной из самых распространенных ошибок в программах является ошибка целочисленного переполнения.

Символьная интерпретация

- **Символьная интерпретация** — метод автоматического тестирования программ, при котором происходит интерпретация программы, где конкретным значениям переменных, зависящих от входных данных, сопоставляются символьные переменные, принимающие произвольные значения.
- **Предикат пути** — система из уравнений и неравенств над символьными переменными и константными значениями, решение которой обеспечивает прохождение потока управления по исследуемому пути.
- **Предикат безопасности** — дополнительные условия на предикат пути, которые позволяют обнаружить ошибку в программе.

Постановка задачи

Необходимо разработать и реализовать метод поиска ошибок целочисленного переполнения в бинарном коде с помощью динамической символьной интерпретации, который

- Строит предикат безопасности для ошибок целочисленного переполнения;
- Находит источники и стоки ошибок;
- Находит ошибки для различных размеров целочисленных типов
- Определяет знаковость результата арифметической операции, приводящей к переполнению, когда это возможно;
- Генерирует входные данные, приводящие к проявлению ошибки;
- В определенных случаях подбирает такие входные данные, чтобы последствия ошибки переполнения вероятнее приводили к аварийному завершению.

Известные решения

Разработанный метод был реализован на базе инструмента Sydr, использующего Triton и DynamoRIO.

Примеры других известных решений:

- KLEE — поиск ошибок одновременно с расширением покрытия
- Mayhem — автоматическая генерация эксплойтов
- Google Sanitizers — обнаружение ошибок в программе во время ее работы
- SAVIOR — фреймворк для гибридного тестирования, нацеленный на поиск ошибок
- ParmeSan — инструмент для фаззинга с обратной связью по санитайзерам
- IntScope — инструмент для поиска ошибок целочисленного переполнения при помощи символьной интерпретациии

Схема проверки предиката безопасности

- Составляем предикат безопасности на равенство флагов `CF`/`OF` единице
- Применяем слайсинг к предикату пути (удаление избыточных ограничений)
- Конъюнкция предиката безопасности и слайса предиката пути
- Проверка выполнимости конъюнкции с помощью математического решателя
- Если выполнима — печатаем предупреждение и сохраняем файл для воспроизведения ошибки

Понятие источника и стока ошибки

- Из-за слишком частых арифметических инструкций проверять на переполнение каждую слишком расточительно, а также может привести к большому количеству ложных срабатываний.
- **Источник** — арифметическая инструкция, в которой потенциально может произойти переполнение значения
- Виды **стоков**:
  - Условные переходы
  - Разыменование указателя
  - Аргументы функций

Определение знаковости

- На уровне ассемблера не всегда возможно определить, является ли значение стока знаковым или беззнаковым
- Для обнаружения знаковости используем следующий алгоритм:
  - Идем в обратном порядке по всем условным переходам, в которых были использованы переменные, которые есть в стоке
  - Из найденных условных переходов узнаем знаковость
- Также можем определить знаковость, если соответствующие входные данные были обработаны функцией `strto*l`

Например, инструкция условного перехода `JL` говорит о знаковости стока.

Пример Integer Overflow to Buffer Overflow

Входные данные: +00000000002

Ответ: +01073741825

Разрядность: 32 бита

```
1  int main() {
2      int size;
3      fscanf(stdin, "%d", &size);
4      if (size <= 0) return 1;
5      size_t i;
6      int *p = malloc(size * sizeof(int));
7      if (p == NULL) return 1;
8      for (i = 0; i < (size_t)size; i++) {
9          p[i] = 0;
10     }
11     printf("%d\n", p[0]);
12     free(p);
13 }
```

Juliet Dynamic

CWE	P=N	Текстовые ошибки			Верификация		
		TPR	TNR	ACC	TPR	TNR	ACC
Integer Overflow	2580	99.92%	90.89%	95.41%	98.10%	90.89%	94.50%
Integer Underflow	1922	99.90%	91%	95.45%	97.45%	91%	94.22%
Int Overflow to BOF	188	100%	100%	100%	100%	100%	100%

FreeImage и xlnt

С помощью разработанного метода было найдено целочисленное переполнение в проекте **FreeImage** при вызове функции `fseek`:

```
unsigned off_head, off_setup, off_image, i;
...
fseek(ifp, off_setup + 792, SEEK_SET);
```

Также ошибка целочисленного переполнения была найдена в проекте **xlnt**:

```
in_->seekg(static_cast<std::ptrdiff_t>(sector_data_start()
 + sector_size() * static_cast<std::size_t>(id)));
std::vector<byte> sector(sector_size(), 0);
```

unbound

В проекте **unbound** также была найдена ошибка целочисленного переполнения в приведенном фрагменте в 9 строке:

```
1  int sign = 0;
2  uint32_t i = 0;
3  uint32_t seconds = 0;
4
5  for(*endptr = nptr; **endptr; (*endptr)++) {
6      switch (**endptr) {
7      ...
8      case '9':
9          i *= 10;
10     ...
11 }
```

Заключение

- Был разработан и реализован метод поиска ошибок целочисленного переполнения в бинарных файлах методами динамической символьной интерпретации
- Метод протестирован на наборе тестов Juliet и показал высокие результаты
- Найдены новые ошибки в проектах с открытым исходным кодом FreeImage, xlnt и unbound

Публикации по теме курсовой работы

- Vishnyakov A., Logunova V., Kobrin E., Kuts D., Parygina D., Fedotov A. Symbolic Security Predicates: Hunt Program Weaknesses. 2021 Ivannikov ISPRAS Open Conference (ISPRAS), IEEE, 2021, pp. 76-85. DOI: 10.1109/ISPRAS53967.2021.00016
- Vishnyakov A., Fedotov A., Kuts D., Novikov A., Parygina D., Kobrin E., Logunova V., Belecky P., Kurmangaleev Sh. Sydr: Cutting Edge Dynamic Symbolic Execution. 2020 Ivannikov ISPRAS Open Conference (ISPRAS), IEEE, 2020, pp. 46-54. DOI: 10.1109/ISPRAS51486.2020.00014

Спасибо за внимание!

Juliet Dynamic

Для проверки эффективности работы предикатов была сделана тестовая система на основе набора тестов Juliet.

- Собираем и запускаем тесты, собираем результаты TP, TN, FP, FN на основе вывода Sydr
- Проверяем на основе сгенерированного файла корректность результата с помощью санитайзеров
- Выводим отдельно результаты на основе вывода Sydr и результаты, верифицированные санитайзерами

Ссылки на результаты

- Juliet Dynamic: https://github.com/ispras/juliet-dynamic
- FreeImage: https://sourceforge.net/p/freeimage/bugs/347/
- xlnt: https://github.com/tfussell/xlnt/issues/616
- unbound: https://github.com/NLnetLabs/unbound/issues/637