Символьные предикаты безопасности для обнаружения ошибок в бинарном коде

Кобрин Илай Александрович

27 апреля 2023 г.

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

Научный руководитель: к.ф.-м.н. Батузов Кирилл Андреевич
Научный консультант: к.ф.-м.н. Вишняков Алексей Вадимович

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.

Динамическая символьная интерпретация позволяет строить математическую модель программы и находить ошибки путем решения соответствующих систем уравнений и неравенств.

Динамическая символьная интерпретация

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

Постановка задачи

Необходимо разработать и реализовать метод построения и проверки символьных предикатов безопасности для обнаружения ошибок в бинарном коде с помощью динамической символьной интерпретации, который должен

- Находить ошибки целочисленного переполнения, выхода за границы массива, деления на ноль и разыменования нулевого указателя;
- Для ошибок целочисленного переполнения находить источники ошибок и их стоки;
- Генерировать входные данные, которые приведут к проявлению ошибки.

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

- Интерпретируем программу, анализируем каждую инструкцию на пути выполнения
- Составляем предикат безопасности для обнаружения конкретной ошибки
- Составляем конъюнкцию предиката безопасности и предиката пути
- Проверяем выполнимость конъюнкции с помощью математического решателя
- Если выполнима — печатаем предупреждение и сохраняем файл для воспроизведения ошибки
- Для ошибок разыменования нулевого указателя и деления на ноль составляем уравнения на равенство адреса и делителя нулю

Выход за границу массива

- Строим предикат безопасности для инструкций, разыменовывающих адрес.
- Находим границы с помощью теневой кучи, теневого стека или с помощью эвристического подхода.
- Составляем предикат безопасности в виде неравенства на то, может ли адрес выходить за границы.
- Составляем дополнительные условия, чтобы попытаться перезаписать адрес возврата из функции или разыменовать отрицательный адрес.

Целочисленное переполнение

- Строим предикат безопасности для арифметических инструкций, представляющий из себя равенство флагов `CF/OF` единице.
- Так как таких инструкций в программах очень много, то проверяем предикат только если было найдено опасное место, использующее переполненное значение (сток ошибки).
- Стоками ошибки являются: условный переход, разыменование адреса, аргументы опасных функций (`malloc`, `memcpy` и т.п.), аргументы остальных функций.
- Знаковость переполнения определяем по ранее встретившимся инструкциям условных переходов (например, инструкция `JL` будет обозначать знаковый тип).
- Для функций копирования и функций выделения памяти составляем предикат безопасности так, чтобы ошибка переполнения вероятнее привела к дальнейшему выходу за границы массива.

Оценка точности на наборе тестов Juliet

Для оценки точности символьных предикатов безопасности была сделана тестовая система на основе набора тестов Juliet.

- Собираем и запускаем тесты, измеряем истинно положительные, истинно отрицательные, ложно положительные и ложно отрицательные срабатывания на основе текстового вывода
- Верифицируем результат на основе срабатываний санитайзеров при запуске на сгенерированном файле
- Считаем отдельно точность (ACC) на основе текстового вывода и на основе верификации санитайзерами

Оценка точности на наборе тестов Juliet

CWE	Кол-во	Текстовые ошибки			Верификация		
		TPR	TNR	ACC	TPR	TNR	ACC
Stack BOF	376	100%	100%	100%	100%	100%	100%
Heap BOF	752	100%	100%	100%	100%	100%	100%
Buffer Underwrite	376	100%	100%	100%	100%	100%	100%
Buffer Overread	376	100%	100%	100%	100%	100%	100%
Buffer Underread	376	100%	100%	100%	100%	100%	100%
Integer Overflow	5160	99.92%	90.89%	95.41%	99.92%	90.89%	95.41%
Integer Underflow	3844	99.90%	91%	95.45%	99.90%	91%	95.45%
Unexpected Sign Ext	1504	100%	100%	100%	100%	100%	100%
Signed to Unsigned	1504	100%	100%	100%	100%	100%	100%
Divide by Zero	1128	66.67%	100%	83.33%	66.67%	100%	83.33%
Int Overflow to BOF	376	100%	100%	100%	100%	100%	100%
ИТОГО	15772	97.57%	94.83%	96.20%	97.57%	94.83%	96.20%

P — положительные тесты, N — отрицательные тесты

TP — истинно положительные срабатывания, FP — ложно положительные срабатывания

TN — истинно отрицательные срабатывания, FN — ложно отрицательные срабатывания

$$TPR = \frac{TP}{P} = \frac{TP}{TP+FN},\ TNR = \frac{TN}{N} = \frac{TN}{TN+FP}$$

$$ACC = \frac{TP+TN}{TP+TN+FP+FN}$$

Результаты практического применения

Были найдены ошибки в следующих проектах с открытым исходным кодом:

- FreeImage (целочисленное переполнение)
- rizin (целочисленное переполнение, ведущее к выходу за границы массива)
- xlnt (целочисленное переполнение и выход за границу массива)
- unbound (целочисленное переполнение)
- hdp (деление на ноль)
- miniz (целочисленное переполнение)

Заключение

Был разработан и реализован метод построения и проверки символьных предикатов безопасности для обнаружения ошибок в бинарном коде.

- Метод позволяет находить ошибки разыменования нулевого указателя, выхода за границы массива, целочисленного переполнения и деления на ноль.
- Метод показал высокую точность в 96.20% на наборе тестов Juliet.
- Найдены ошибки в проектах с открытым исходным кодом: FreeImage, rizin, xlnt, unbound, hdp и miniz.

Публикации

Результаты были опубликованы:

- 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
- Вишняков А.В., Кобрин И.А., Федотов А.Н. Поиск ошибок в бинарном коде методами динамической символьной интерпретации. Труды Института системного программирования РАН, том 34, вып. 2, 2022, стр. 25-42. DOI: 10.15514/ISPRAS-2022-34(2)-3

Публикации по теме работы:

- 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
- Vishnyakov A., Kuts D., Logunova V., Parygina D., Kobrin E., Savidov G., Fedotov A. Sydr-Fuzz: Continuous Hybrid Fuzzing and Dynamic Analysis for Security Development Lifecycle. 2022 Ivannikov ISPRAS Open Conference (ISPRAS), IEEE, 2022, pp. 111-123. DOI: 10.1109/ISPRAS57371.2022.10076861

Спасибо за внимание!

Теневой стек и теневая куча

- Теневой стек хранит информацию о фреймах для текущего стека вызовов
  - Обновляется при вызове функций и возврате из них
  - Позволяет получить верхнюю границу буфера на стеке как верхнюю границу фрейма, в котором был выделен буфер
- Теневая куча хранит информацию о текущих границах буферов, выделенных на куче
  - Обновляется при вызове функций аллоцирования и освобождения памяти
  - Позволяет получать точные границы буферов на куче

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 }
```

FreeImage

В проекте FreeImage были найдены ошибки целочисленного переполнения в следующих местах:

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

```
int doff;
...
fseek(ifp, doff + base, SEEK_SET);
...
```

FreeImage

Неявное преобразование типа `long` к `int`:

```
int parse_tiff(int base);
...
parse_tiff(thumb_offset + 12);
```

Целочисленное переполнение в условном переходе:

```
if (*len *
    tagtype_dataunit_bytes[
        (*type <= LIBRAW_EXIFTAG_TYPE_IFD8) ? *type : 0]
        > 4)
{
    fseek(ifp, get4() + base, SEEK_SET);
}
```

Беззнаковое целочисленное переполнение при вычислении ширины:

```
width = raw_width - left_margin - (get4() & 7);
```

rizin

В проекте rizin была найдена ошибка целочисленного переполнения, приводящая к выходу за границы массива:

```
symbols_size = (symbols_count + 1) * 2 *
                    sizeof(struct symbol_t);

if (symbols_size < 1) {
    ht_pp_free(hash);
    return NULL;
}
if (!(symbols = calloc(1, symbols_size))) {
    ht_pp_free(hash);
    return NULL;
}
```

rizin

Выход за границы массива вследствие переполнения:

```
for (i = 0; i < bin->nsymtab && i < symbols_count; i++) {
...
    symbols[j].last = 0;
    if (inSymtab(hash, symbols[j].name,
                 symbols[j].addr)) {
        RZ_FREE(symbols[j].name);
    } else {
        j++;
    }
...
}
...
symbols[j].last = true;
```

xlnt

Ошибки целочисленного переполнения при умножении и сложении:

```
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);
```

xlnt

Найденная ошибка выхода за границы массива:

```
sector_chain
compound_document::follow_chain(sector_id start,
                                const sector_chain &table)
{
    auto chain = sector_chain();
    auto current = start;
    while (current >= 0)
    {
        chain.push_back(current);
        current =
            table[static_cast<std::size_t>(current)];
    }
    return chain;
}
```

unbound

Ошибка целочисленного переполнения в unbound:

```
int sign = 0;
uint32_t i = 0;
uint32_t seconds = 0;

for(*endptr = nptr; **endptr; (*endptr)++) {
    switch (**endptr) {
    ...
    case '9':
        i *= 10;
    ...
}
```

hdp

Ошибка целочисленного деления на ноль в проекте hdp:

```
int32 buf_size;
/* we are bounded above by VDATA_BUFFER_MAX */
buf_size = MIN(total_bytes, VDATA_BUFFER_MAX);
// make sure there is at least room
// for one record in our buffer
chunk = buf_size / hsize + 1;
```

miniz

Ошибки целочисленного переполнения в miniz (зависимость PyTorch):

```
if (cdir_size < pZip->m_total_files *
                MZ_ZIP_CENTRAL_DIR_HEADER_SIZE)
    return mz_zip_set_error(pZip,
                MZ_ZIP_INVALID_HEADER_OR_CORRUPTED);
```

Ссылки на результаты

- Juliet Dynamic: https://github.com/ispras/juliet-dynamic
- FreeImage: https://sourceforge.net/p/freeimage/bugs/347/
- rizin: https://github.com/rizinorg/rizin/issues/2935
- xlnt: https://github.com/tfussell/xlnt/issues/616, https://github.com/tfussell/xlnt/issues/626
- unbound: https://github.com/NLnetLabs/unbound/issues/637
- hdp: https://bugs.launchpad.net/ubuntu/+source/libhdf4/+bug/1915417
- miniz: https://github.com/richgel999/miniz/pull/238