ИСП РАН

Динамическая символьная интерпретация для процессорной архитектуры Байкал-М (AArch64)

Влада Логунова

22 июня 2023 г.

OS DAY 2023, Москва

Направление исследований

- Автоматическое тестирование ПО для поиска ошибок при обработке внешних данных
- Разработка средств динамического анализа для создания безопасного ПО (согласно ГОСТ Р 56939-2016)

CVE-2018-4124

Динамический анализ бинарного кода:

- Гибридный фаззинг = фаззинг + символьная интерпретация
- Фаззинг: информация о покрытии кода (greybox)
- *Символьная интерпретация*: моделирование семантики инструкций для символьного потока данных (whitebox)

Создание доверенных отечественных экосистем

- Актуализация задач обеспечения технологической независимости программно-аппаратного комплекса
- Возможность доверенной загрузки российских дистрибутивов Linux для архитектуры Байкал-М
- Внедрение стандартов жизненного цикла безопасной разработки

Символьная интерпретация

Построение системы уравнений:

- интерпретация семантики вычислений с *символьными данными* (т.е. входными данными и их производными) в виде формул (AST)
- описание условий ветвления, зависящих от символьных данных
- уравнения системы, отражающие *выбранные* направления *символьных* ветвлений на пути, составляют *предикат пути*

Получение новых входных данных:

- последовательное инвертирование символьных переходов, т.е. ограничений, входящих в предикат пути, с помощью SMT-решателя (например, Z3 или Bitwuzla)
- *предикаты безопасности* – наборы дополнительных ограничений, выполнение которых говорит о потенциальном наличии ошибки
- обмен с фаззером при гибридном фаззинге

Динамическая символьная интерпретация

- Concolic = concrete + symbolic
- Предикат пути на основе трассы инструкций реального запуска
- Достаточная скорость для повышения качества входных данных в процессе гибридного фаззинга
- Недостаток: ограничение длины входных данных

Sydr

- Динамическая бинарная инструментация (DBI): DynamoRIO
- Работа с AST-представлением: Triton
- SMT-решатель: Bitwuzla
- Извлечение контекста выполнения и применение техник символьной интерпретации:
  - InstructionEvent
  - Анализ символьных адресов и jump-таблиц
  - Слайсинг предиката пути
  - Предикаты безопасности
  - Моделирование семантики функций libc

Архитектура ARM64 или AArch64

- ARM: Advanced RISC Machine (в оригинале Acorn RISC Machine)
- В 2011 г. в версии ARMv8-A появилось 64-битное расширение AArch64 (ARMv7 и ниже только 32/16 бит)
- ARM используется в подавляющем большинстве мобильных устройств, а также в процессорах Apple M1, Байкал-М
- Fujitsu Fugaku – первый ARM-суперкомпьтер, ставший лидером Top500 (2020, 2021)

Архитектурные компоненты, требуемые при портировании

- Набор регистров (Wn/Xn и пр.)
- Набор инструкций
- Вычисление адресных выражений
- Механизмы передачи управления, соглашение о вызовах
- Условное выполнение инструкций, учёт флагов
- Выравнивание pc, sp

Технические детали реализации

- Triton поддерживает 176/402 спецификаций базового набора инструкций ARM DDI 0487 (сделано 4 PR с исправлениями)
- Учёт внутренних сдвигов в адресах и арифметике:
  `ldr x0, [x1, w2, uxtx #3]` $\longrightarrow$ `x0 := x1 + ext(w2)*8`
- Обновление теневого стека для инструкций `bl/blr` и `ret`
- Изменение инструкций, к которым привязаны предикаты безопасности

Анализ косвенных переходов (switch)

- Два варианта содержимого памяти: прямой адрес перехода либо сдвиг (относительно базового адреса)
- Эвристики как для x86 не работают :(
- Новая эвристика: у сдвига есть свой базовый адрес
- Размеры значения сдвига для косвенного перехода 1, 2, 4 байта
- Учёт внутреннего сдвига lsl, т.е. умножение значения оффсета
- Уточнение размеров jump-таблицы по инструкции `cmp`

Вычисление адреса перехода на уровне инструкций

Прямой переход:

```
...
cmp  x8, 0x2f
b.hi 4006e8 <main+0x2c>
[...]
adrp x9, 420000 <__libc_...>
add  x9, x9, 0x28
add  x8, x9, x8, lsl 3
sub  x8, x8, 0x100
ldr  x8, [x8]
blr  x8
...
```

Переход с оффсетом:

```
...
cmp  w8, 0x8
b.hi 400758 <main+0x4a>

adrp x9, 400000 <_init-0x520>
add  x9, x9, 0x818
adr  x10, 4006f0 <main+0x3c>
ldrb w11, [x9, x8]
add  x10, x10, x11, lsl 2
br  x10
...
```

Результаты сравнения Sydr с SymQEMU

Приложение	Sydr				SymQEMU			
	Покрытие	**Уник.**	**Вх.файлы**	**Время**	**Покрытие**	**Уник.**	**Вх.файлы**	**Время**
freetype2	**98.93%**	227	1645	20m	97.99%	120	1287	20m
jsoncpp	**100%**	176	907	11,4s	75.35%	0	193	4,2s
lcms	**99.78%**	5	103	9,7s	99.64%	3	422	16m35s
libpng	**97.60%**	76	317	20m	97.47%	72	734	20m
libxml2	96.94%	12	971	20m	**99.85%**	249	1736	20m
openthread	**99.96%**	12	285	1m26s	99.89%	5	486	2m51s
re2	**93.37%**	219	47	5,6s	89.64%	140	81	36,1s
woff2	**100%**	163	239	20m	93.52%	0	408	20m

Покрытие – доля инструмента относительно объединения покрытия, достигнутого всеми инструментами. Уник. – количество строк покрытия, которые были достигнуты только для данного инструмента Timeout: 20 минут общий; 10 секунд на каждый запрос к SMT-решателю

Future work

- Сравнение инструментов гибридного фаззинга для AArch64
- Поиск багов в открытых приложениях в рамках проекта OSS-Sydr-Fuzz

Контакты: vlada@ispras.ru

Информация о Sydr: sydr-fuzz.github.io

Спасибо за внимание