ИСП РАН

Магистерская диссертация на тему

Исследование и разработка методов динамического анализа для определения входных данных влияющих на выполнение условных переходов

Дьячков Л.А.

Руководитель: к.ф-м.н, с.н.с. Курмангалеев Ш.Ф.

5 Апреля 2019

ИСП РАН

Введение

Фаззинг

Фаззинг-тестирование – активно развивающийся метод для поиска ошибок в программном обеспечении

Введение

Фаззинг

Фаззинг-тестирование – активно развивающийся метод для поиска ошибок в программном обеспечении

Проблема

Может быть затруднено нахождение входных данных, позволяющих “пройти” условный переход

Введение

Фаззинг

Фаззинг-тестирование – активно развивающийся метод для поиска ошибок в программном обеспечении

Проблема

Может быть затруднено нахождение входных данных, позволяющих “пройти” условный переход

Предлагаемое решение

Использовать методы динамического анализа для определения какие байты из входного файла влияют на конкретный условный переход

Постановка задачи

Цель работы

- Разработка метода динамического анализа, позволяющего определять байты входного файла, влияющего на выполнение инструкции условного перехода.
- Программная реализация метода, работающая на операционной системе Linux с архитектурой процессора x86-64

Постановка задачи

Цель работы

- Разработка метода динамического анализа, позволяющего определять байты входного файла, влияющего на выполнение инструкции условного перехода.
- Программная реализация метода, работающая на операционной системе Linux с архитектурой процессора x86-64

Подзадачи

- Изучение существующих технологий динамического символьного выполнение и динамического анализа потока данных.
- Сравнение технологий на тестовом наборе
- Обзор возможных подходов, реализация прототипов, разработка метода
- Программная реализация на основе выбранной технологии

Динамическое символьное выполнение

Определение

Метод динамического анализа, заключающийся в том в том, что во время выполнения программы некоторым конкретным значениям ставятся в соответствие символьные переменные. Затем, для каждой выполняемой инструкции, генерируются формулы для SMT решателя.

online символьное выполнение

Модуль символьного выполнения используется для того, чтобы генерировать конкретные данные для посещения новых узлов cfg программы.

Offline символьное выполнение

Модуль символьного выполнения используется для анализа конкретной трассы выполнения.

Динамическое символьное выполнение

Определение

Метод динамического анализа, заключающийся в том в том, что во время выполнения программы некоторым конкретным значениям ставятся в соответствие символьные переменные. Затем, для каждой выполняемой инструкции, генерируются формулы для SMT решателя.

online символьное выполнение

Модуль символьного выполнения используется для того, чтобы генерировать конкретные данные для посещения новых узлов cfg программы.

Offline символьное выполнение

Модуль символьного выполнения используется для анализа конкретной трассы выполнения.✓

Динамический анализ помеченных данных

Определение

Динамический анализ помеченных данных (Dynamic Taint Analysis), также известный как динамический анализ потока данных (Dynamic Flow tracking) – это техника анализа програм, позволяющая определить какие состояния программы зависят от входных данных.

Принцип работы

- *Определение источников помеченных данных*. Обычно метками снабжаются данные, получаемые из недоверенного источника (файл, stdin, сеть).
- *Распространение пометок (Taint propogation)*. Для каждой инструкции необходимо принять решение как распротраняются пометки в зависимости от её операндов и факта их помеченности
- *Применение политик безопасности*. Например отслеживание попадания помеченных данных в аргументы “опасных” функций, или факта помеченности счетчика инструкций.

Исследованные технологии

Динамическое символьное выполнение

- Triton
- Angr
- Manticore

Динамический анализ помеченных данных

- Triton
- Taintgrind
- libdft64
- moflow (bap gentrace)

Triton

Фреймворк, поддерживающий архитектуры x86 и x86-64, содержит модули DSE (offline) и анализа помеченных данных, использует *Intel Pin* для динамической бинарной инструментации.

Достоинства

- Поддерживает offline режим динамического символьного выполнения (Нет необходимости предварительно снимать трасу, при помощи pin это делается “на лету”)
- Поддерживает **"ONLY_TAINTED"** режим, позволяющий генерировать smt формулы только для помеченных инструкций

Недостатки

- Работает крайне медленно (примерно на два порядка медленее других инструментов на базе pin)
- Нет поддержки символьных файлов/пометки данных на основе системных вызовов
- Грубый модуль пометки данных, низкая гранулярность

Angr

Платфмормонезависимый фреймворк динамического символьного выполнения, использующий трансляцию инструкций в VEX с последующей эмуляцией.

Достоинства

- Поддержка множества архитектур
- Хороший онлайн движок, отлично работающий поиск состояний на небольших примерах.
- Есть множество полезных примитивов (таких как символьные файлы) из коробки

Недостатки

- Нет оффлайн режима (плагины, которые должны его поддерживать не работают)
- На настоящих (например binutils) программах онлайн поиск не может дойти до интересных состояний

Manticore

Аналог Angr, поддерживающий также смарт-контракты. использует эмуляцию инструкций.

Достоинства

- Поддерживает offline режим

Недостатки

- Медленно работает (около 50 секунд эмуляции для того, чтобы дойти до main в программе с glibc)
- Нет поддержки SSE инструкций
- Нет поддержки некоторых системных вызовов при эмуляции

Taintgrind

Плагин для valgrind, реализующий динамической анализ потока данных.

Достоинства

- Поддержка множества архитектур (все что поддерживает valgrind)
- Возможность помечать данные из конкретных файлов
- Возможность статической инструментации исходного кода для пометки данных

Недостатки

- Низкий уровень гранулярности пометок

libdft64

Инструмент для динамического анализа помеченных данных, работающий на основе *Intel Pin*. Реализация libdft (поддерживает только x86) из проекта vuzzer64 работающая с 64 разрядными исполняемыми файлами.

Достоинства

- Возможность помечать данные из конкретных файлов
- Гранулярность меток на уровне байта

Недостатки

- Игнорируется регистр флагов
- Поддерживаются не все инструкции

moflow (gentrace)

Инструмент для динамического анализа помеченных данных, работающий на основе *Intel Pin*. Часть фреймворка moflow, изначально являющийся частью bap.

Достоинства

- Возможность помечать данные из конкретных файлов
- Гранулярность меток на уровне байта

Недостатки

- Отсутсвие учета семантики инструкций в механизме распространения пометок
- Использование специальной метки *MIXED_TAINT* для случая, когда адрес зависит от нескольких пометок.
- Механизм распространения пометок работает на уровне старших регистров

Метод на основе символьного выполнения

Каждому байту входного файла ставится в соответствие символьная переменная. Для каждого предиката пути выполняется следующий алгоритм

Input: Предикат пути, представленный в виде AST для SMT формулы
Output: Список символьных переменных
Function `GetLeafs(`V`):`

```
Found ← ∅;
for child ∈ V do
    if IsLeaf(child) then
        Found ← Found ∪ {child};
    else
        Found ← Found ∪ GetLeafs(child);
return Found;
```

Algorithm 1: Метод на основе символьного выполнения

Поскольку переменные взаимнооднозначно соответствуют адресам – задача решена.

Метод на основе символьного выполнения

Достоинства

- Простой и естественный алгоритм
- Высокая точность работы

Недостатки

- Низкая производительность (Построение формул на каждую инструкцию)
- Ни один из расмотренных DSE инструментов не позволяет реализовать метод, работающий на несинтетических примерах без ощутимых доработок самого инструмента

Метод на основе символьного выполнения

Достоинства

- Простой и естественный алгоритм
- Высокая точность работы

Метод на основе символьного выполнения

Достоинства

- Простой и естественный алгоритм
- Высокая точность работы

Недостатки

- Низкая производительность (Построение формул на каждую инструкцию)
- Ни один из расмотренных DSE инструментов не позволяет реализовать метод, работающий на несинтетических примерах без ощутимых доработок самого инструмента

Метод на основе анализа помеченных данных

- Каждому байту входного файла ставится в соответсвие метка (тег).
- Для всех последующих инструкций в трассе выполнение выполняется алгоритм распространения пометок
- Для каждой инструкции условного извлекаются теги, которыми помечен регистр флагов. Байты, соответствующие этим тегам – искомые.

Метод на основе анализа помеченных данных

Достоинства

- Высокая скорость работы
- Простая реализация

Метод на основе анализа помеченных данных

Достоинства

- Высокая скорость работы
- Простая реализация

Недостатки

- Точность ниже чем у символьного выполнения.

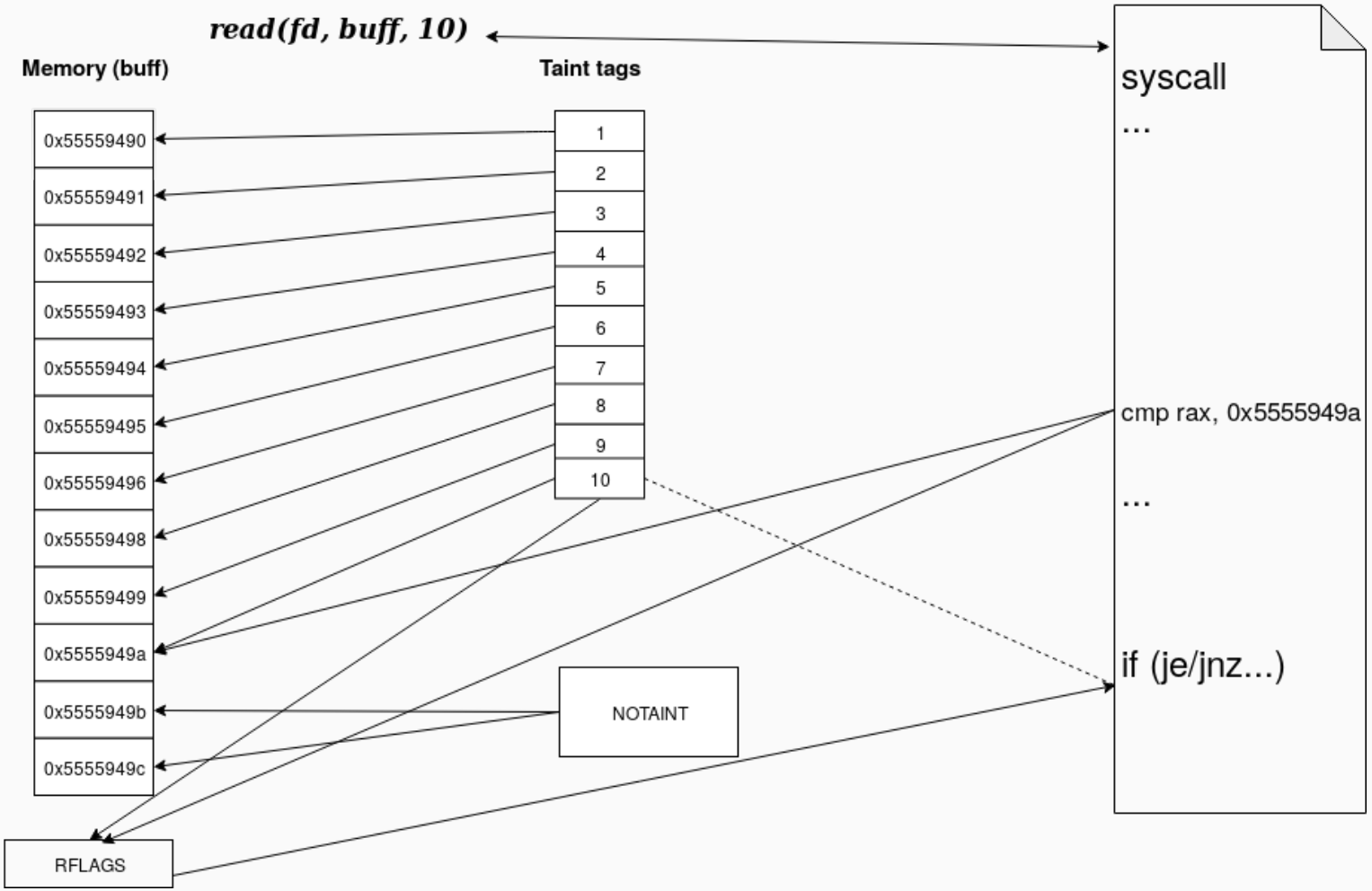
Реализация

Метод на основе динамического символьного выполнения был реализован на фреймворке **Angr**. Для промышленного применения не подходит по озвученным выше причинам.

Реализация

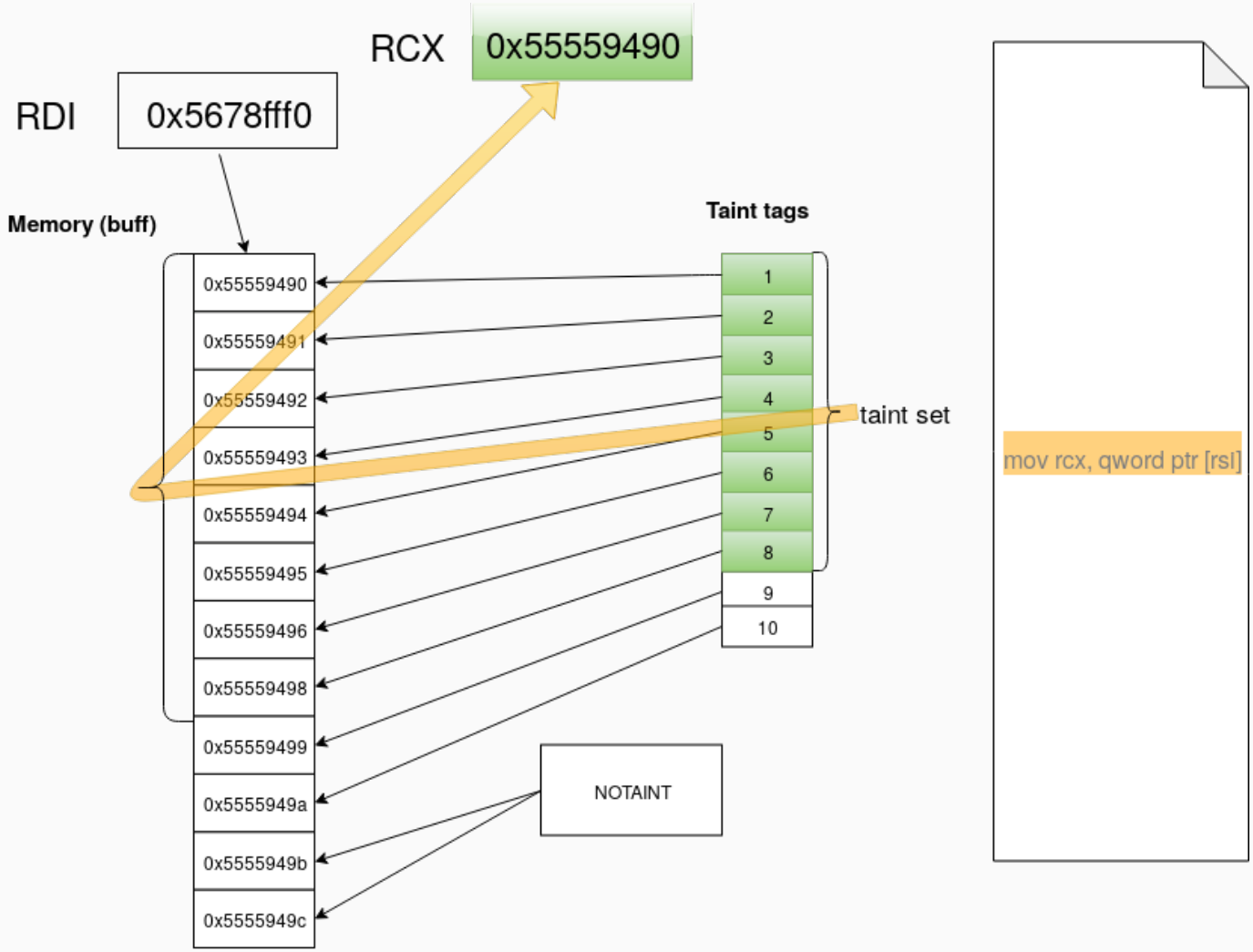
Метод на основе динамического символьного выполнения был реализован на фреймворке **Angr**. Для промышленного применения не подходит по озвученным выше причинам.
Для реализации метода на основе анализа помеченных данных был выбран Moflow.

Как работает Moflow



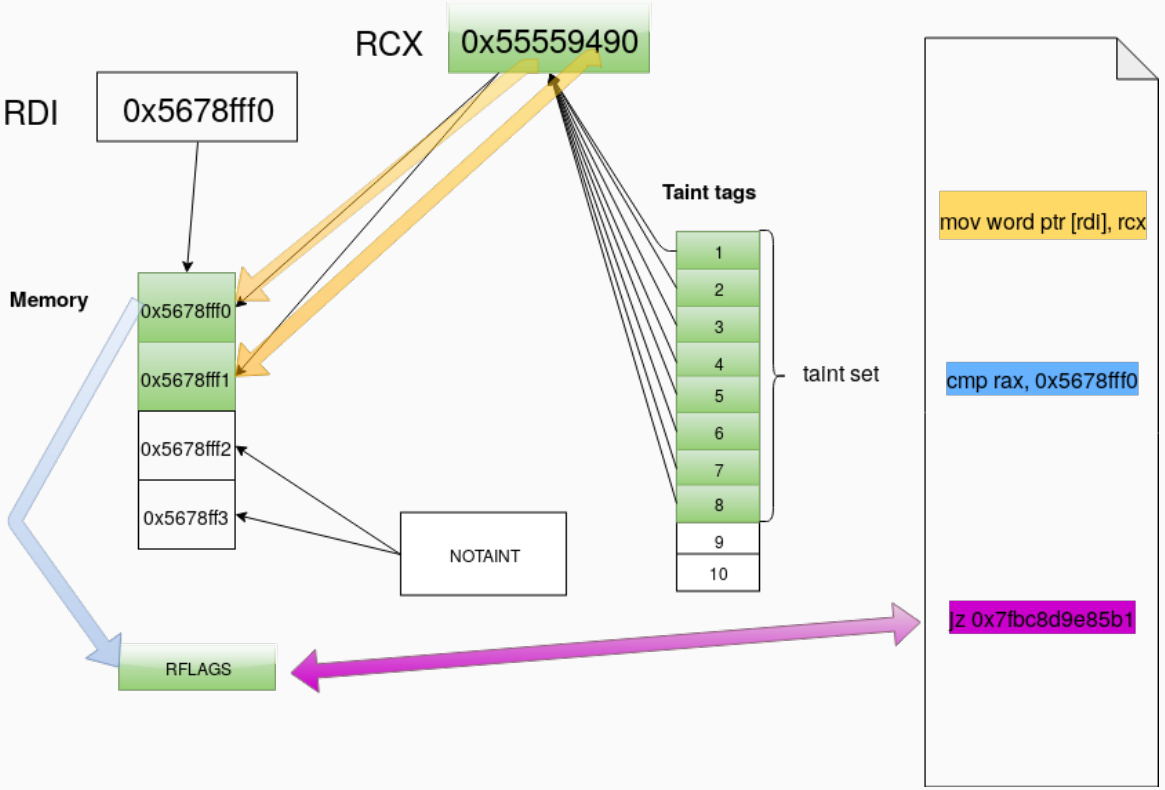
Доработки Moflow

RCX помечен 8 тегами. Moflow бы пометил бы **RCX** как **MIXED_TAINT**, для реализации множественного помечивания нужна структура **Множество пометок**.



Доработки Moflow

0x5678fff0 и **0x5678fff1** помечаются 8 тэгами. Правильно было бы представить **RCX** как 8 отдельных байт, которые могут быть помечены независимо, тогда **0x5678fff0** и **0x5678fff1** будут помечены 2 различными тегами.



Множество пометок – структура для поддержки множественных пометок, от неё требуется поддержка следующих операций

- Добавление **пометки в множество**.
- Объединение с другим **Множеством пометок**
- Вывод содержимого **Множества пометок**

Множество пометок – структура для поддержки множественных пометок, от неё требуется поддержка следующих операций

- Добавление **пометки в множество**.
- Объединение с другим **Множеством пометок**
- Вывод содержимого **Множества пометок**

Было разработано несколько подходов к реализации структуры.

Время работы в секундах для различных реализаций

	vanilla	vector	bitset6000	roaring	bitset64 tree	bitset256 tree	bitset512 tree
cmark	3.5s	4.6s	3.7s	4.2s	3.8s	3.7s	3.7s
file	20.8s	47s	60s	70s	45.5s	46.3s	47.5s
libjpeg	14.5s	1762s	49s	378s	307s	108s	80s
libyaml	16.5s	22.5s	25s	26s	23.5s	23.5s	23.5s

- Поскольку не менее 70% операций объединения проводится над пустыми множествами и множествами из 1 элемента, во всех случаях множество пометок было реализовано как пара из **uint32_t** и указателя на более сложную структуру (который в этих 70% оказывается нулевым).
- Обычные **std::set** и **std::unordered_set** оказались заметно медленее приведенных реализаций, и в таблицу не склюены.
- Roaring (https://github.com/RoaringBitmap/CRoaring) – библиотека сжатых битовых векторов.
- Bitset – реализация на основе **std::bitset**, 6000 битов достаточно для всех примеров из тествого набора. Размер битового множества должен быть известенна этапе компиляции.
- **bitset64 tree** и **bitset256 tree** реализации, в которых **Множество пометок** представлено как а **std::map** с целочисленными ключами и **std::bitset** размеров 64 и 256 в качестве значений.