МФТИ ИСП РАН

Исследование применения формальных моделей для тестирования технологии eBPF в ядре Linux

Максимов Даниил Андреевич
Научный руководитель: Хорошилов Алексей Владимирович
Научный консультант: Буздалóв Денис Викторович

Про eBPF

- Технология, которая позволяет выполнять пользовательскую логику в коде ядра Linux

Про eBPF

- Технология, которая позволяет выполнять пользовательскую логику в коде ядра Linux
- Используется:
  - изоляция недоверенных процессов (например, в Docker или Chrome)
  - фильтрация сетевых пакетов (например, в AWS)
  - отслеживание потребления заряда приложениями в Android
  - и многое другое…

Про eBPF

- Технология, которая позволяет выполнять пользовательскую логику в коде ядра Linux
- Используется:
  - изоляция недоверенных процессов (например, в Docker или Chrome)
  - фильтрация сетевых пакетов (например, в AWS)
  - отслеживание потребления заряда приложениями в Android
  - и многое другое…
- Но не любую логику можно исполнить! Существует eBPF Verifier

Про eBPF

- Технология, которая позволяет выполнять пользовательскую логику в коде ядра Linux
- Используется:
  - изоляция недоверенных процессов (например, в Docker или Chrome)
  - фильтрация сетевых пакетов (например, в AWS)
  - отслеживание потребления заряда приложениями в Android
  - и многое другое…
- Но не любую логику можно исполнить! Существует eBPF Verifier
- В нем достаточно часто находят ошибки

Про eBPF

- Технология, которая позволяет выполнять пользовательскую логику в коде ядра Linux
- Используется:
  - изоляция недоверенных процессов (например, в Docker или Chrome)
  - фильтрация сетевых пакетов (например, в AWS)
  - отслеживание потребления заряда приложениями в Android
  - и многое другое…
- Но не любую логику можно исполнить! Существует eBPF Verifier
- В нем достаточно часто находят ошибки
- Хотим повысить его качество

Почему сложно тестировать eBPF?

- eBPF похож на виртуальную машину

МФТИ ИСП РАН

Почему сложно тестировать eBPF?

- eBPF похож на виртуальную машину
- Входные данные верификатора - целая программа

Почему сложно тестировать eBPF?

- eBPF похож на виртуальную машину
- Входные данные верификатора - целая программа
- Верификатор накладывает серьёзные ограничения, например:
  - Должна останавливаться
  - У регистров строгая типизированность
  - Не всегда доступны все инструкции
  - это только часть…

Почему сложно тестировать eBPF?

- eBPF похож на виртуальную машину
- Входные данные верификатора - целая программа
- Верификатор накладывает серьёзные ограничения, например:
  - Должна останавливаться
  - У регистров строгая типизированность
  - Не всегда доступны все инструкции
  - это только часть…
- Если просто генерировать случайные программы, то мы с большой вероятностью не попадём в ограничения

Текущее состояние eBPF: теория

Производятся попытки специфицирования eBPF и верификации имеющихся алгоритмов компиляции/проверки eBPF программ:

Simple and Precise Static Analysis of Untrusted Linux Kernel Extensions

Elazar Gershuni
Tel Aviv University, Israel and VMware Research, USA
elazarg@gmail.com

Nadav Amit
VMware Research, USA
namit@vmware.com

Arie Gurfinkel
University of Waterloo, Canada
arie.gurfinkel@uwaterloo.ca

Nina Narodytska
VMware Research, USA
nnarodytska@vmware.com

Jorge A. Navas
SRI International, USA
jorge.navas@sri.com

Noam Rinetzky
Tel Aviv University, Israel
maon@cs.tau.ac.il

Leonid Ryzhyk
VMware Research, USA
lryzhyk@vmware.com

Mooly Sagiv
Tel Aviv University, Israel
msagiv@cs.tau.ac.il

Verifying the Verifier: eBPF Range Analysis Verification

Harishankar Vishwanathan(✉), Matan Shachnai, Srinivas Narayana, and Santosh Nagarakatte

Rutgers University, New Brunswick, USA
{harishankar.vishwanathan,m.shachnai, srinivas.narayana,santosh.nagarakatte}@rutgers.edu

CAV Artifact Evaluation ★ Available

Specification and verification in the field: Applying formal methods to BPF just-in-time compilers in the Linux kernel

Luke Nelson, Jacob Van Geffen, Emina Torlak, and Xi Wang
University of Washington

Текущее состояние eBPF: практика

- В ядре Linux присутствуют unit-тесты для eBPF

Текущее состояние eBPF: практика

- В ядре Linux присутствуют unit-тесты для eBPF
- Разработаны и активно используются различные fuzzer’ы
  - Syzkaller
  - BVF
  - Buzzer

Текущее состояние eBPF: практика

- В ядре Linux присутствуют unit-тесты для eBPF
- Разработаны и активно используются различные fuzzer’ы
  - Syzkaller
  - BVF
  - Buzzer
- Они опираются на *знание реализации*

Текущее состояние методов тестирования

- У крупных серьёзных систем существуют спецификации

Текущее состояние методов тестирования

- У крупных серьёзных систем существуют спецификации
- Тестирование может отталкиваться не от реализации, а от этих формальных требований

Текущее состояние методов тестирования

- У крупных серьёзных систем существуют спецификации
- Тестирование может отталкиваться не от реализации, а от этих формальных требований
- Idris2 - это язык программирования, позволяющий описать модель системы и одновременно задать ограничения к ней

Текущее состояние методов тестирования

- У крупных серьёзных систем существуют спецификации
- Тестирование может отталкиваться не от реализации, а от этих формальных требований
- Idris2 - это язык программирования, позволяющий описать модель системы и одновременно задать ограничения к ней
- DepTyCheck - библиотека на языке Idris2, с помощью которой можно генерировать структуры с достаточно сложными ограничениями

Постановка задачи

1. Выбрать подмножество языка eBPF (для создания модели)
2. Выбрать тестируемое свойство модели
3. Построить модель этого подмножества с учётом семантических ограничений eBPF на языке Idris2
4. Провести тестирование eBPF с помощью библиотеки DepTyCheck

Ход работы: выбор подмножества eBPF

- **eBPF не обладает официальной полной спецификацией**

Ход работы: выбор подмножества eBPF

- **eBPF не обладает официальной полной спецификацией**
- Но есть eBPFPL!

$$
\begin{array}{rcl}
cmd & ::= & w := E \mid w :=_{sz} *p \mid *p :=_{sz} x \\
 & \mid & \texttt{assume}(B) \mid w := \texttt{shared}\, K \\
E & ::= & K \mid x \mid x{+}y \mid x{-}y \\
B & ::= & x = y \mid x \neq y \mid x \leq y
\end{array}
$$

Figure 2. Primitive commands. K denotes a numeral.

Ход работы: выбор подмножества eBPF

- **eBPF не обладает официальной полной спецификацией**
- Но есть eBPFPL!

$$
\begin{array}{rcl}
cmd & ::= & w := E \mid w :=_{sz} *p \mid *p :=_{sz} x \\
 & \mid & \texttt{assume}(B) \mid w := \texttt{shared}\, K \\
E & ::= & K \mid x \mid x{+}y \mid x{-}y \\
B & ::= & x = y \mid x \neq y \mid x \leq y
\end{array}
$$

Figure 2. Primitive commands. K denotes a numeral.

- Было рассмотрено подмножество этого языка:
  - Любые корректные операции add, sub, mul, mov с регистрами и константами
  - Разрешены только jmp-инструкции вперёд

Ход работы: выбор тестового оракула

Чтобы не переусложнять модель, было принято решение рассмотреть 2 таких оракула:

Ход работы: выбор тестового оракула

Чтобы не переусложнять модель, было принято решение рассмотреть 2 таких оракула:

1. Любая семантически корректная eBPF программа из выделенного подмножества должна быть принята eBPF Verifier’ом

Ход работы: выбор тестового оракула

Чтобы не переусложнять модель, было принято решение рассмотреть 2 таких оракула:

1. Любая семантически корректная eBPF программа из выделенного подмножества должна быть принята eBPF Verifier’ом
2. Любая eBPF программа из выделенного подмножества с семантически корректными инструкциями, но некорректным завершением, должна быть отвергнута eBPF Verifier’ом

Ход работы: построение модели и генератора

- Модель описывает программу в целом, не уточняя семантически незначимые детали

Ход работы: построение модели и генератора

- Модель описывает программу в целом, не уточняя семантически незначимые детали
- Результат генерации - eBPF программа, записанная в виде структуры на языке C

Ход работы: построение модели и генератора

- Модель описывает программу в целом, не уточняя семантически незначимые детали
- Результат генерации - eBPF программа, записанная в виде структуры на языке C
- Генерация происходит в 2 этапа:
  - Генерируется объект, соответствующий модели
  - Генерируются незначимые для модели детали (такие как конкретные константы)

Ход работы: построение модели и генератора

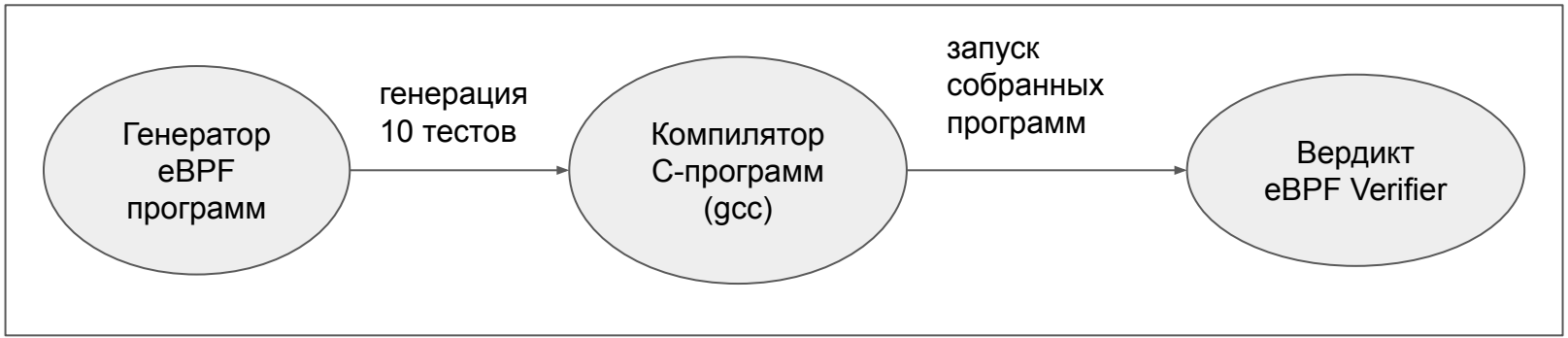
```
r0 = r1
r2 = 1337
r3 = 1580
if (r3 & r2) then
  r0 = 1
  exit
else
  r0 = 0
  exit
```

```
prog : FullProgram 8 True
prog =
  ModifyOp (MovReg 0 1) $
  ModifyOp (MovImm 2) $
  ModifyOp (MovImm 3) $
  IfThenElse (CondSrReg Set 3 2) (
    ModifyOp (MovImm 0) $
    Exit
  ) (
    ModifyOp (MovImm 0) $
    Exit
  )
```

```
struct bpf_insn autogen_insns[] = {
    BPF_MOV64_REG(BPF_REG_0, BPF_REG_1),
    BPF_MOV_IMM(BPF_REG_2, 1337ul),
    BPF_MOV_IMM(BPF_REG_3, 1580ul),
    BPF_JMP_REG(BPF_SET, BPF_REG_3, BPF_REG_2, 2),
    BPF_MOV_IMM(BPF_REG_0, 1),
    BPF_EXIT_INSN(),
    BPF_MOV_IMM(BPF_REG_0, 0),
    BPF_EXIT_INSN(),
};
```

Ход работы: тестирование

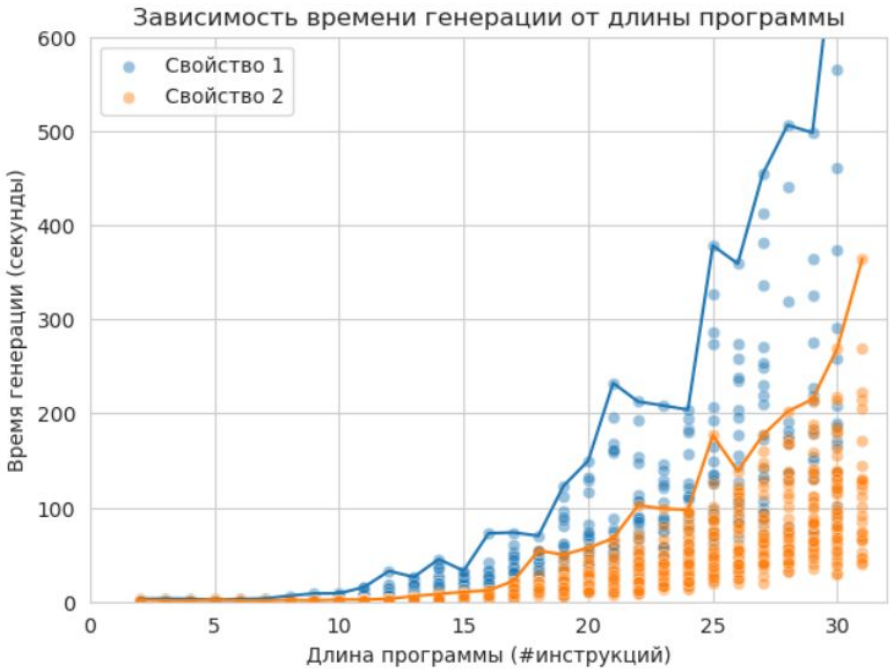
- Тестирование выполнялось 10ч
- Параметры машины:
  - Процессор AMD Ryzen 5 1600
  - Установлена Ubuntu с ядром Linux версии 6.5.0-28-generic
- Конвейер тестирования (1 итерация):



Результаты

1. Построена модель подмножества языка eBPFPL
2. Проведено тестирование верификатора eBPF программ
   a. За 10ч сгенерировано 196 программ
   b. Из них 46 - относятся к первому свойству (полная корректность)
   c. Остальные 150 - ко второму свойству (наличие нетерминируемого пути)
   d. Ошибок в работе eBPF Verifier’a не найдено
3. Вывод: выбранный метод подходит для тестирования eBPF

Результаты


Зависимость времени генерации от длины программы
Свойство 1
Свойство 2
Время генерации (секунды)
Длина программы (#инструкций)
0
100
200
300
400
500
600
0
5
10
15
20
25
30


Зависимость времени генерации от длины программы
Макс. время 1
Макс. время 2
Ср. время 1
Ср. время 2
Время генерации (секунды)
Длина программы (#инструкций)
0
100
200
300
400
500
600
0
5
10
15
20
25
30

Что дальше?

1. Полноценно поддержать всю модель eBPFPL
2. Построить архитектуру для тестирования
   a. Включает в себя модификацию кода ядра
   b. Текущий конвейер тестирования не полностью автоматизирован
3. Провести масштабное тестирование с использованием всех созданных технологий

Дополнительные слайды