
НИУ ВШЭ, МИЭМ, ИБ, 4 курс

30 мая 2023 г.

# Гибридный Фаззинг Фреймворка PyTorch

Теодор Арсений Ларионов-Тришкин

tlarionovtrishkin@edu.hse.ru

# Цели и Задачи

1. Исследование существующих подходов гибридного фаззинга
2. Оценка поверхности атаки фреймворка PyTorch
3. Проведение фаззинг-тестирования, исправление найденных ошибок во фреймворке PyTorch
4. Реализация предложенных улучшений для инструмента Sydr-Fuzz
5. Экспериментальная оценка предложенных улучшения для Sydr-Fuzz

# Обзор подходов к поиску ошибок в ПО методами динамического анализа

Глава – “Software Security Analysis Techniques”, pp. 7-22

# Что Такое Фаззинг?

```
void crash(char* buf) {
    if (buf[0] == 'F') {
        if (buf[1] == 'U') {
            if (buf[2] == 'Z') {
                if (buf[3] == 'Z') {
                    *(int*)NULL = 0x1337;
                }
            }
        }
    }
}
```

# Символьная Интерпретация

```
void bug(uint32_t val) {
    if (val * 0xa9a57b == 0x1337beef) {
        crash();
    }
}
```

# Символьная Интерпретация

```
void bug(int SYM_VAR) {
    if (SYM_VAR * 0xa9a57b == 0x1337beef) {
        crash();
    }
}
```

```
SYM_VAR = 0x1337beef / 0xa9a57b
SYM_VAR = 0x1d
```

# Поиск ошибок в PyTorch

Глава – “PyTorch Fuzzing”, pp. 23-34

# Мотивация

1. PyTorch является фундаментальным блоком во многих AI/ML приложениях.
2. Фреймворк разрабатывается с невероятной скоростью. Это приводит к тому, что многие ошибки не успевают найти.
3. PyTorch не фаззится разработчиками.

# Определение Поверхности Атаки

Ручной Анализ

+ Позволяет найти наиболее интересные цели

- Время затратный

Автоматический Анализ

+ Позволяет найти интересные точки входа

+ Применим на больших кодовых базах

- Не “понимает” код

# Определение Поверхности Атаки

Найденные цели для тестирования:

1. Загрузка предобученных моделей
2. Протокол удаленного взаимодействия (RPC)

# Подготовка к Фаззингу

1. Разработка фаззинг-оберток
2. Создание изначального тестового корпуса
3. Упаковка цели в докер-образ

Глава – “PyTorch Fuzzing”, p. 28

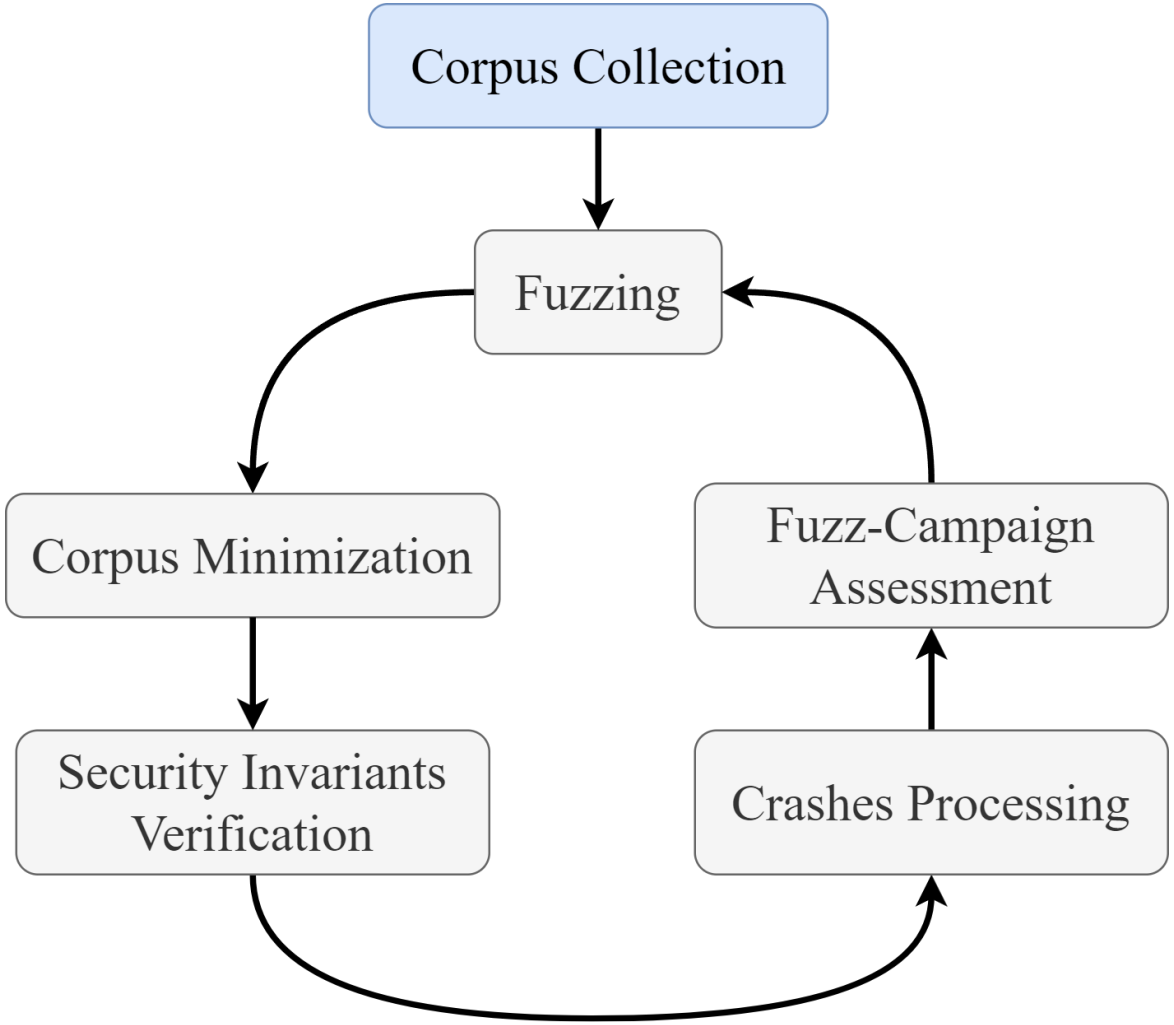
# Подготовка к Фаззингу

```
extern "C" int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size) {
  std::stringstream ss;
  std::copy((char *)data, (char *)data + size,
            std::ostreambuf_iterator<char>(ss));

  try {
    auto m = torch::jit::load(ss);
  } catch (const c10::Error &e) {
    return 0;
  } catch (const torch::jit::ErrorReport &e) {
    return 0;
  } catch (const std::runtime_error &e) {
    return 0;
  }

  return 0;
}
```

# Пайплайн Динамического Анализа



# Улучшения инструмента для гибридного фаззинга Sydr-Fuzz

Глава – “Hybrid Fuzzer Improvements”, pp. 35-39

# Оптимизация механизма верификации срабатываний предикатов безопасности

```
char data[64];
uint index = read_int();
if (index < 74)
    data[index] = 0x37;
```

Глава – “Enhancing Security Invariants Checking Mechanism”, p. 37

# Стратегия запуска Sydr в режиме моделирования памяти

```
uint16_t crc_ibm_table[256] = {
    0x0000, 0xc0c1, 0xc181, 0x0140, ...
};

uint16_t crc_ibm_byte(uint16_t crc, const uint8_t c)
{
    uint8_t sym_idx = (crc ^ c) & 0xFF;
    return crc_ibm_table[sym_idx] ^ (crc >> 8);
}

if (crc_ibm_byte(0, buf[0]) == 0x1337) {
    error();
}
```

# Результаты

Глава – “Results”, pp. 40-47

# Найденные ошибки в PyTorch

**17 Ошибок** было найдено, и **5 пулл-реквестов** с исправлениями отправлено:

1. #94300: Add size check before calling stack_.at(dict_pos) in unpickler.cpp
2. #94298: Add stack emptiness checks inside interpreter.cpp
3. #94297: Add size check before calling .back() in rpc/script_call.cpp
4. #94295: Add exception handlers for stoll in schema_type_parser.cpp
5. #91401: Add out-of-bounds checks inside irparser.cpp and unpickler.cpp

Глава – “Results”, p. 40

# Замеры производительности оптимизированного механизма верификации срабатываний предикатов безопасности

Производительность механизма для верификации срабатываний предикатов безопасности была улучшена на ~99%.

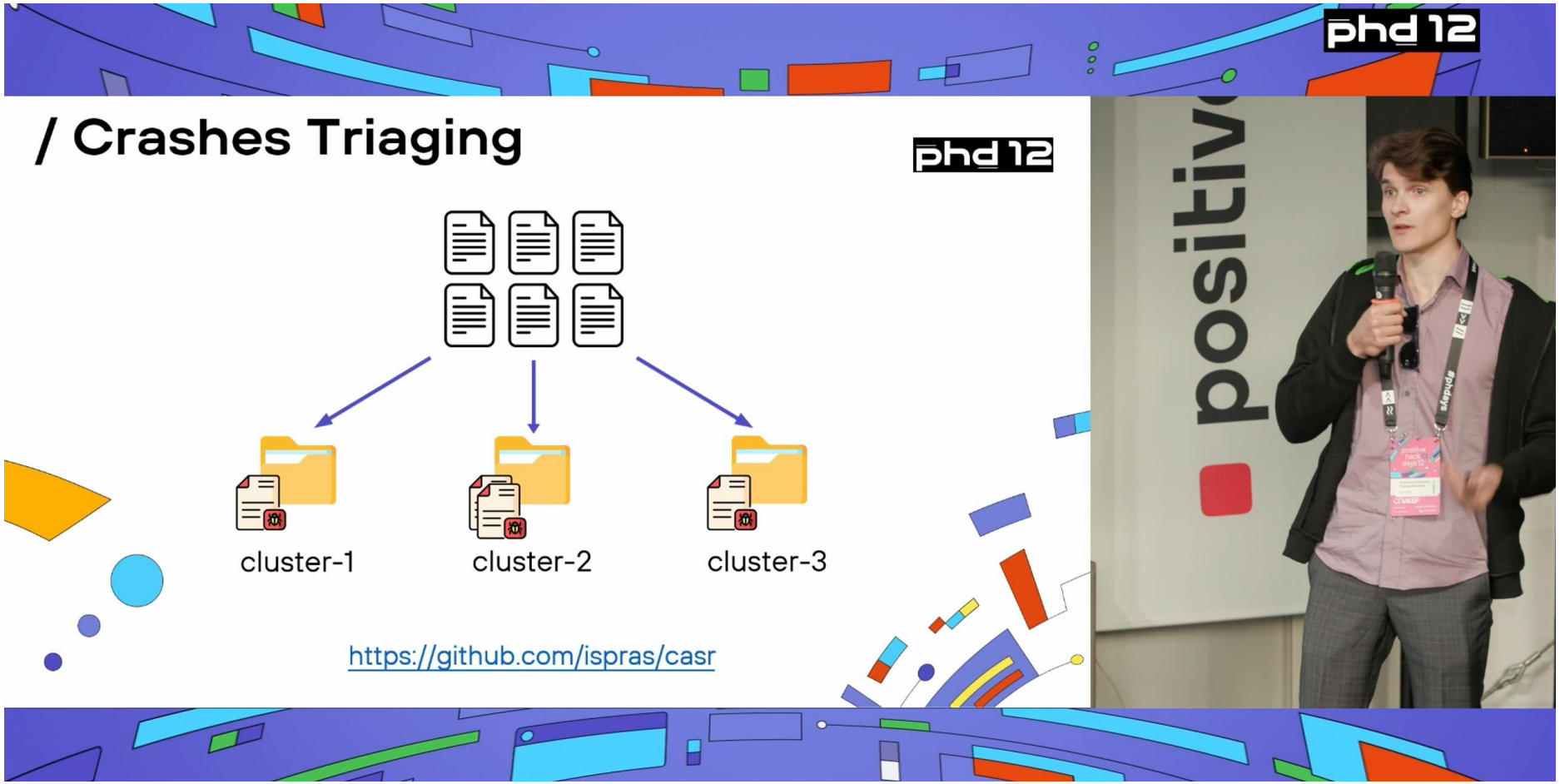
Глава – “Results”, p. 45

# Экспериментальные результаты для предложенной стратегии

Стратегия запуска Sydr в режиме моделирования памяти позволила добиться наиболее высокого покрытия кода, показав результат на ~2% лучше, чем Sydr без данной стратегии.

Глава – “Results”, p. 44

# Выступление на конференции Positive Hack Days (PHD) 2023




Ищем баги в PyTorch с помощью непрерывного фаззинга

# Спасибо за внимание!

Вопросы?