Аннотирование ELF файлов

Михаил Кашкаров

Задача

1. Проверить, что все приложения/библиотеки собраны с определенными политиками:
   - `stack-protector`, `stack_clash`, `FORTIFY`, `GLIBCXX_ASSERTION`, `...`
   - ABI-значимые опции: `short enum`, `wchar_t size`, ...

2. Найти объекты, в которых эти требования не выполняются

Задача

Варианты:

1. Передать глобальные опции всем проектам/библиотекам для сборки
    - Не подходит решение для большого количества разных пакетов с разными системами сборок

Задача

Варианты:

1. Передать глобальные опции всем проектам/библиотекам для сборки
    - Не подходит решение для большого количества разных пакетов с разными системами сборок
2. Предоставить тулу для анализа бинарных объектов для проверки
    - Не все опции записываются в результирующий файл
        - Например, -D_FORTIFY_SOURCE
    - Опции могут перезаписываться атрибутами функций
        - __attribute__((optimize))

Задача

Варианты:

1. Необходим способ сохранять оптимизации и всю нужную информацию
    - Должна применяться не только ко всему исходному файлу, но и к отдельным функциям

2. Должна быть компактной и хранится внутри бинарников

Задача

- В RedHat для этого сделали плагин для gcc Annobin (Annotate Binaries), который уже используется в Fedora.

- Расскажем, как использовали эту технологию, как встроили её у себя в компилятор/линкер/загрузчик и расширили применимость для последующего анализа

Toolchain watermarking

Задача добавить маркеры в ELF объекты для дальнейшего их чтения и обработки.

Основные цели:

1. Проверить все объекты на ABI совместимость
2. Проверить, что объекты собраны с требуемыми флагами
3. Получить требования объектов для запуска

Toolchain watermarking

Задача добавить маркеры в ELF объекты для дальнейшего их чтения и обработки.

Основные цели:

1. Определите, все ли объекты реализуют один и тот же ABI (например, они используют один и тот же формат long double / c ++ std :: string / std :: list).

Это также должно включать отрицательные свойства, чтобы, например, если библиотека не использует тип wchar_t, ее можно было бы слинковать с приложением, которое использует любой размер wchar_t.

Toolchain watermarking

Задача добавить маркеры в ELF объекты для дальнейшего их чтения и обработки.

Основные цели:

1. Определите, все ли объекты реализуют один и тот же ABI (например, они используют один и тот же формат long double / c ++ std :: string / std :: list).

В идеале мы хотим найти ответ на следующие вопросы:

- Какой (специфичный для архитектуры) вариант ABI используется в объекте X и совместим ли он с объектом Y?

Toolchain watermarking

Задача добавить маркеры в ELF объекты для дальнейшего их чтения и обработки.

Основные цели:

1. Определите, все ли объекты реализуют один и тот же ABI

В идеале мы хотим найти ответ на следующие вопросы:

- Какой (специфичный для архитектуры) вариант ABI используется в объекте X и совместим ли он с объектом Y?
- Каковы размеры основных типов, используемых в объекте X? (Для тех типов, которые явно не включены в ABI, например enum и wchar_t). Если объект не использует конкретный тип, это также должно быть обнаружено.

Toolchain watermarking

Задача добавить маркеры в ELF объекты для дальнейшего их чтения и обработки.

Основные цели:

2. Определите, был ли объект скомпилирован в соответствии с применимыми политиками безопасности (например, -fstack-protector-strong).

Это также включает возможность проверки того, какие инструменты использовались для создания объекта, чтобы, например, можно было определить, был ли объект скомпилирован с помощью устаревшей версии компилятора.

Toolchain watermarking

Задача добавить маркеры в ELF объекты для дальнейшего их чтения и обработки.

Основные цели:

2. Определите, был ли объект скомпилирован в соответствии с применимыми политиками безопасности (например, -fstack-protector-strong).

Вопросы, на которые мы хотим здесь ответить, включают:

- Каждая ли функция в объекте X была скомпилирована с опцией Y?
- Каждая ли функция в объекте X была скомпилирована с версией Y компилятора (или более новой)?
- Был ли объект X связан с включенной опцией Y (например, relro)?

Toolchain watermarking

Задача добавить маркеры в ELF объекты для дальнейшего их чтения и обработки.

Основные цели:

3. Определите требования времени выполнения объекта (например, версию оборудования, которая им нужна, или объем стека, который им требуется).

Toolchain watermarking

- Добавление аннотаций должно быть простым и быстрым для чтения/обработки:

Загрузчик исполняемых файлов - сильно оптимизированная программа и обработка новых аннотаций должна быть тоже быстрой и надежной

Реализация

- Реализация с использованием ELF Notes

Реализация

- Реализация с использованием ELF Notes

- Один небольшой набор аннотаций будет храниться в выделенном разделе и будет содержать только информацию, необходимую загрузчику.

Реализация

- Реализация с использованием ELF Notes

- Один небольшой набор аннотаций будет храниться в выделенном разделе и будет содержать только информацию, необходимую загрузчику.
- Второй раздел, не предназначенный для размещения, будет содержать более подробные заметки, которые можно анализировать с помощью отдельных статических инструментов.

Реализация

- Реализация с использованием ELF Notes

- Один небольшой набор аннотаций будет храниться в выделенном разделе и будет содержать только информацию, необходимую загрузчику.
- Второй раздел, не предназначенный для размещения, будет содержать более подробные заметки, которые можно анализировать с помощью отдельных статических инструментов.

- Необходимая информация будет собрана компилятором

Реализация

Аннотации делятся на 2 типа:

- Статические (незагружаемые)
  У секции отсутствует флаг **SHF_ALLOC**, аннотации не загружаются

- Динамические (загружаемые)
  У секции выставлен флаг **SHF_ALLOC**, аннотации загружает динамический линковщик в память

Статические аннотации (незагружаемые)

Информация сохраняется в новом разделе файла в формате ELF note:.

- Как и дебаг информация не загружается в память и может хранится отдельно
- Хранят диапазоны памяти, в котором применяется атрибут/свойство.
- Имя начинается со строки «GA», которая является аббревиатурой от атрибута GNU.
- Аббревиатура используется в целях экономии места.
- Хранятся в строковом виде, чтобы инструменты, которые не знают об этих аннотациях, все равно могли анализировать их.

Статические аннотации (незагружаемые)

```
GA*foo\001\0\002\0      Attribute 'foo' with numeric value 0x200010 (assuming a little endian
target).

GA*bar\0\0              Attribute 'bar' with numeric value 0

GA$fred\0hello\0        Attribute 'fred' with string value "hello"

GA*\004\377\377\0       Attribute stack size with numeric value 0xffff

GA*\002\001\0           -fstack-protector has been enabled.

GA*\002\004\0           -fstack-protector-explicit has been enabled.

GA$\001\002p1\0         Supports spec version 2, notes generated by plugin version 1.

GA$\005gcc v7.0\0       Attribute build tool "gcc v7.0".
```

Статические аннотации (незагружаемые)

Для одного и того же атрибута может существовать несколько аннотаций (с разными значениями и разными диапазонами адресов)

Динамические аннотации (загружаемые)

- Информация сохраняется в новом разделе файла в формате ELF note.
- Используются для проверки объектов в рантайме во время запуска
- Используют спецификацию Linux gABI (стандартизировано)
- Содержат массив свойств объекта:
    - Каждый элемент массива представляет одно свойство программы с типом, размером данных и данными.

Динамические аннотации (загружаемые)

- Содержат массив свойств объекта

- Настраивается и расширяется в зависимости от задач

- Это свойство содержит различные биты на основе флагов компиляции:
  - Линковщик должен выбрать максимальное значение среди всех объектов и скопировать это в вывод.

Динамические аннотации (загружаемые)

Типы свойств:
GNU_COMPILER_FLAGS:

- Битовая маска для быстрой проверки на этапе загрузки объекта
- Содержит опции компиляции, в которых мы хотим убедиться, что они применились
- Например, что все скомпилировали с санитайзерами

Динамические аннотации (загружаемые)

Типы свойств:
GNU_PROPERTY_STACK_SIZE

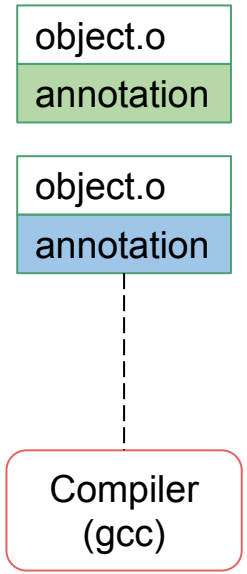
- Загрузчик выполнения должен поднять предел стека до значения, указанного в этом свойстве.

Интеграция в тулчейн

Для полной интеграции в сборку поддержка аннотаций есть в:

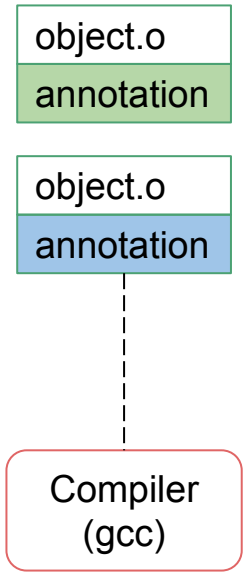
1. Компиляторе (Генерация аннотаций во время компиляции)
2. Линкере (Объединение и сравнение на этапе линковки)
3. Загрузчике (Сравнение на этапе запуска приложения/библиотеки)

Интеграция в тулчейн: компилятор (1/3)



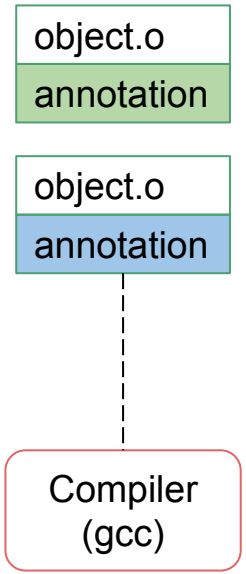
- Компилятор генерирует аннотации на регионы исходного кода
- Регионы определяются по адресам функций
- Включается опцией компилятора (`g++ -fannobin`)
- Генерируемые аннотации так же настраиваются дополнительными опциями

Интеграция в тулчейн: компилятор (1/3)



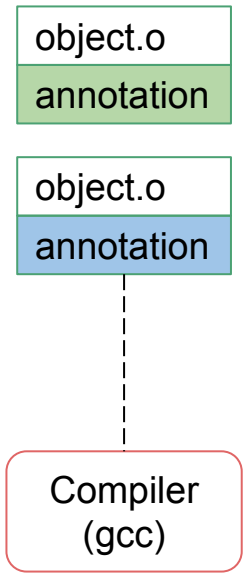

```
$ g++ test.cc ... -fdump-passes -fannobin

          ...

          rtl-shorten                                    :   ON

          rtl-nothrow                                    :   ON

          rtl-dwarf2                                     :   ON

        ==> rtl-annobin <==                              :   ON

          rtl-final                                      :   ON

         rtl-dfinish                                     :   ON

      *clean_state
```

Интеграция в тулчейн: компилятор (1/3)




```
<assemble annobin notes before function>

assemble_start_function(...)
function_body(...)
assemble_end_function(..)

<assemble annobin notes after function>
```

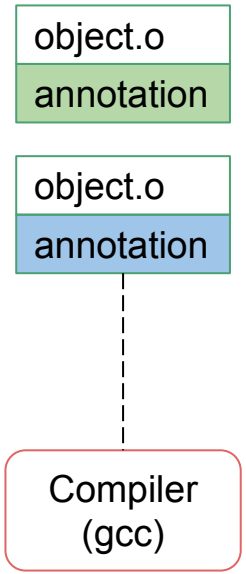
Интеграция в тулчейн: компилятор (1/3)



Аннотации привязаны либо к юниту трансляции, либо к определенной функции:

```
// Global file annotations: [...], range = full translation unit
...
// foo() annotations: [...], range = foo [start, end] address
int foo() {}
```

Интеграция в тулчейн: компилятор (1/3)

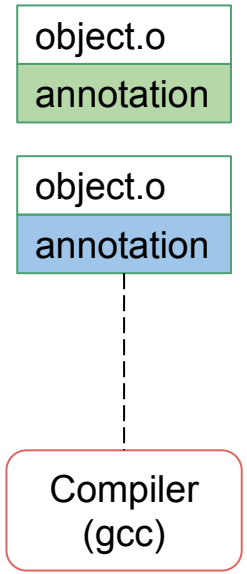


- Пример аннотации для функции:

```
__attribute__((no_sanitize_address))
void foo() {}
```

- $ g++ test.cc -fsanitze=address -fannobin

Интеграция в тулчейн: компилятор (1/3)



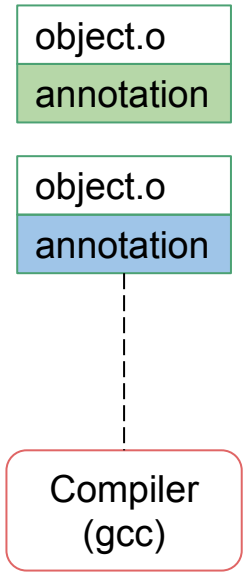
- Пример аннотации для функции:

```
__attribute__((no_sanitize_address))
void foo() {}
```

- Генерируется в ASM вставка ELF секции:

```
.pushsection .gnu.build.attributes, "", %note
    .balign 4
...
    .quad .annobin__Z3foov.start
    .quad .annobin__Z3foov.end
.popsection
```

Интеграция в тулчейн: компилятор (1/3)



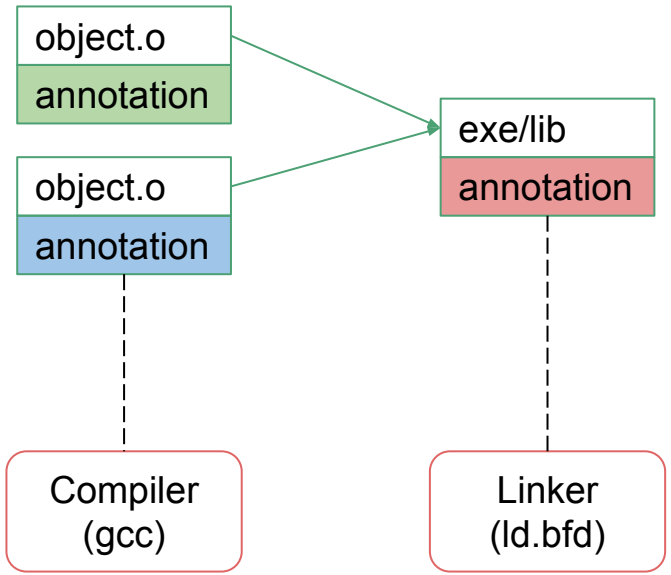
- Пример аннотации для функции:

```
__attribute__((no_sanitize_address))
void foo() {}
```

- Аннотации можно прочитать в объектном файле (например, readelf):

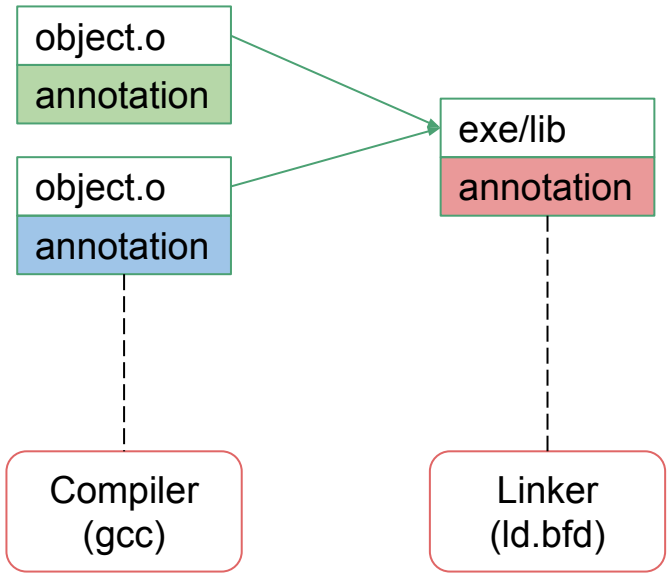
```
$ readelf test.o -NW
...
  GA*<asan>on    0x00000000       OPEN
    Applies to region from 0x4012ea to 0x4012ea (.test.cc)
...
  GA*<asan>off   0x00000000       OPEN
    Applies to region from 0x4011d0 to 0x4011d0 (.test.cc.Z3foov)
```

Интеграция в тулчейн: линкер (2/3)



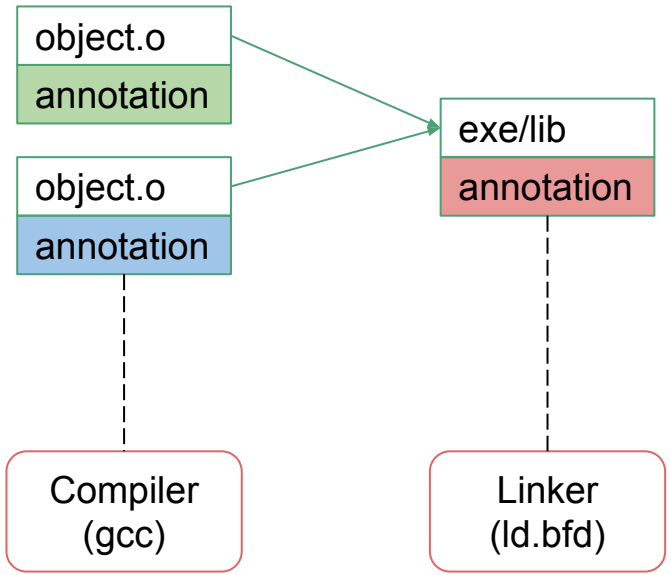
- Производит объединение аннотаций

Интеграция в тулчейн: линкер (2/3)



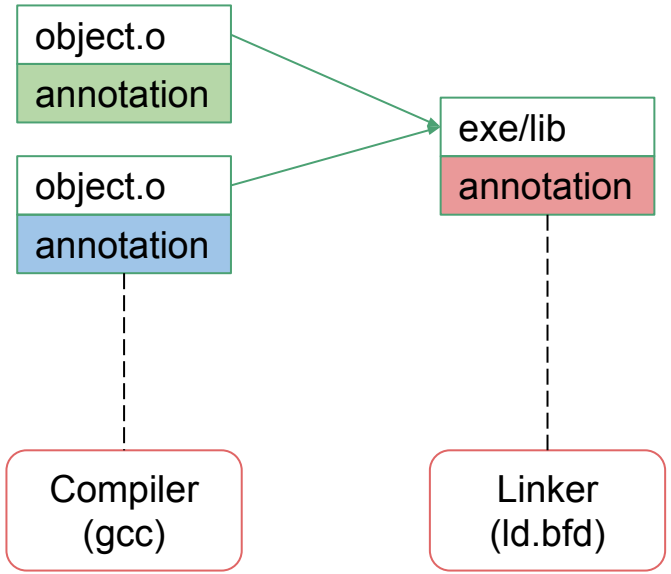
- Производит объединение аннотаций
- Добавляет аннотации, связанные с линковкой

Интеграция в тулчейн: линкер (2/3)



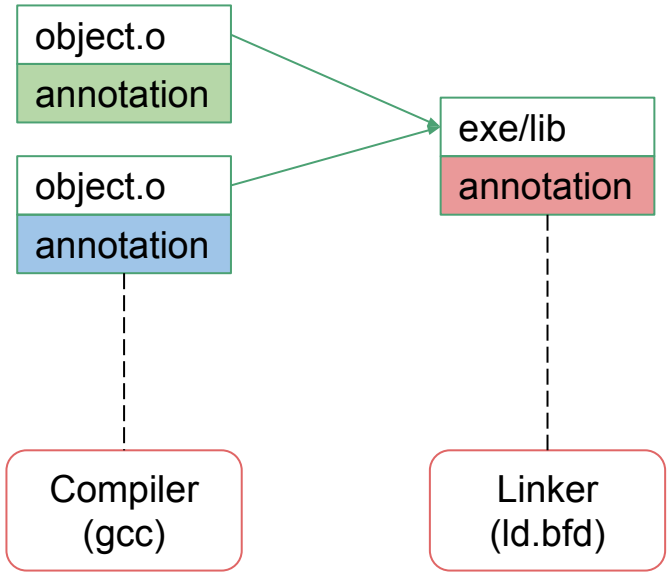
- Производит объединение аннотаций
- Добавляет аннотации, связанные с линковкой
- Производит проверку совместимости аннотаций

Интеграция в тулчейн: линкер (2/3)



- Производит объединение аннотаций
- Добавляет аннотации, связанные с линковкой
- Производит проверку совместимости аннотаций
- В зависимости от режима:
  - strict: выкидываем ошибку линковки
  - warning: предупреждаем о различиях

Интеграция в тулчейн: линкер (2/3)



- Производит объединение аннотаций
- Добавляет аннотации, связанные с линковкой
- Производит проверку совместимости аннотаций
- В зависимости от режима:
  - strict: выкидываем ошибку линковки
  - warning: предупреждаем о различиях
- Пример strict режима:

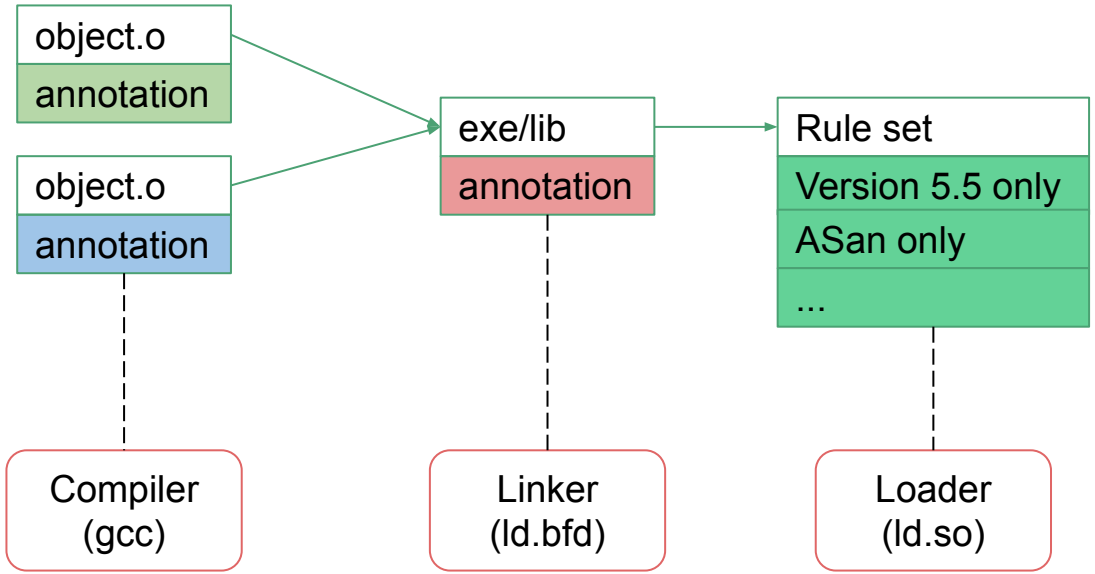
```
$ gcc -fannobin -c sanitized.c -fsanitize=address
$ gcc -fannobin -c unsanitized.c
$ gcc -fannobin sanitized.o unsanitized.o -fsanitize=address

/bin/ld: sanitized.o: Error: Linking sanitized with
                              unsanitized objects, abort
```

Интеграция в тулчейн: линкер (2/3)

- Пример линковки объекта без отсутствия требуемых опций:

```
$ gcc -fplugin=annobin -c s.c -fsanitize=address

$ gcc -fannobin -c u.c

$ gcc -fannobin s.o u.o -fsanitize=address

/bin/ld: s.o: Error: Linking sanitized with unsanitized objects, abort
```

Интеграция в тулчейн: загрузчик (3/3)



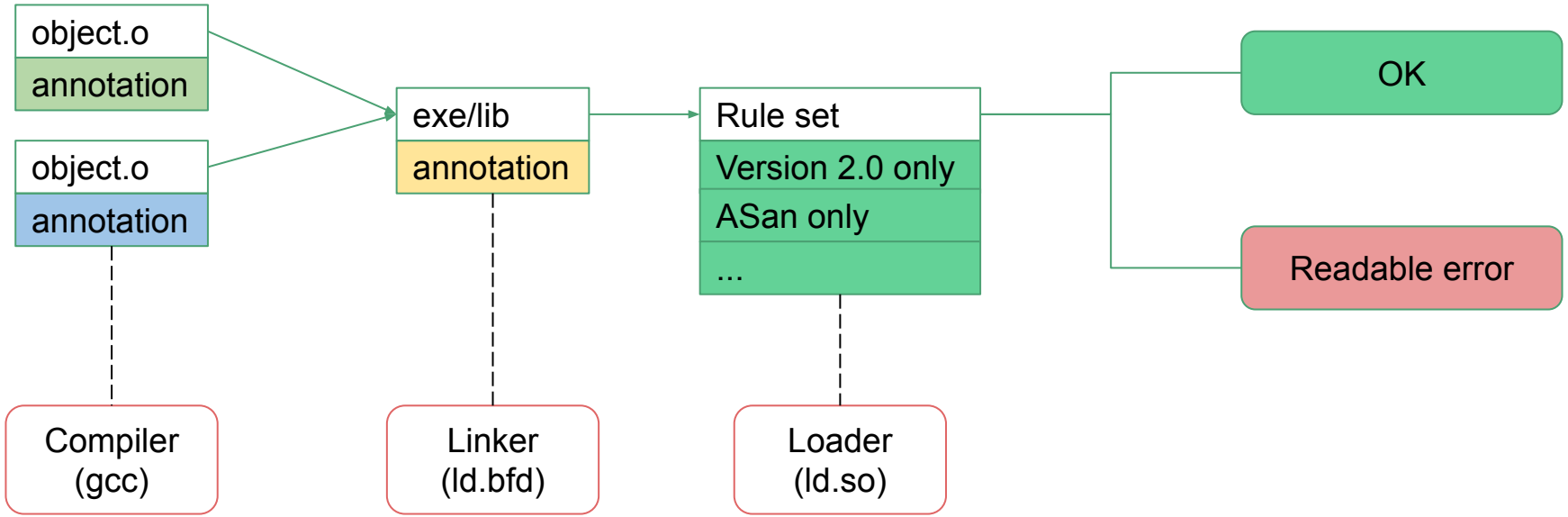
- Содержит правила проверки аннотаций
- Если аннотация помечена компилятором как strict - завершаем запуск приложения/библиотеки с описанием ошибки.

Интеграция в тулчейн: загрузчик (3/3)

```
// As-is:

$ ./a.out:

relocation error:

./a.out:

symbol _ZNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEC1EPKcRKS3_,

version GLIBCXX_3.4.21 not defined in file libstdc++.so.6 with link time reference
```

```
// With annobin, strict annotations detected

$ ./a.out

run-time error: You are trying to execute Tizen 4.0 application on Tizen 5.5 enviroment
```

Интеграция в тулчейн



Результаты

Собрав все приложения в проекте с аннотациями мы получаем информацию о:

- Все ли объекты используют одинаковый ABI:
  - Информацию о совместимости ABI в объекте X и совместимость с Y
  - Размеры базовых типов в объекте X

Результаты

Собрав все приложения в проекте с аннотациями мы получаем информацию о:

- Все ли объекты используют одинаковый ABI:
  - Информацию о совместимости ABI в объекте X и совместимость с Y
  - Размеры базовых типов в объекте X
- Скомпилирован ли объект X в соответствии требованиям:
  - Какие функции в объекте X были скомпилированы/слинкованы с опцией Y?

Результаты

Собрав все приложения в проекте с аннотациями мы получаем информацию о:

- Все ли объекты используют одинаковый ABI:
  - Информацию о совместимости ABI в объекте X и совместимость с Y
  - Размеры базовых типов в объекте X
- Скомпилирован ли объект X в соответствии требованиям:
  - Какие функции в объекте X были скомпилированы/слинкованы с опцией Y?
- Требования запуска объектов:
  - Какие HW ресурсы необходимы для запуска объекта X (arch/stack size ...)

Результаты

Собрав все приложения в проекте с аннотациями мы получаем информацию о:

- Все ли объекты используют одинаковый ABI:
  - Информацию о совместимости ABI в объекте X и совместимость с Y
  - Размеры базовых типов в объекте X
- Скомпилирован ли объект X в соответствии требованиям:
  - Какие функции в объекте X были скомпилированы/слинкованы с опцией Y?
- Требования запуска объектов:
  - Какие HW ресурсы необходимы для запуска объекта X (arch/stack size ...)
- Читабельные ошибки в случае несовместимости

Результаты

Практические применения, которые уже используются:

- С помощью вспомогательных тулов проводится анализ на совместимость объектов и прохождение требований, например:
  - Все ли функции скомпилированы с `-fstack-protector-strong/FORTIFY/short enums etc.`
  - Список исходных файлов/функций, где эти требования не выполняются

Результаты

Практические применения, которые уже используются:

- С помощью вспомогательных тулов проводится анализ на совместимость объектов и прохождение требований, например:
  - Все ли функции скомпилированы с `-fstack-protector-strong/FORTIFY/short enums etc.`
  - Список исходных файлов/функций, где эти требования не выполняются

- С аннотациями на хранение определенных оптимизаций можем:
  - Получить список функций, скомпилированных без санитайзеров
  - Определить неиспользуемые символы в пределах всего проекта по всем приложениям

Размер файлов

- Динамические аннотации (загружаемые):
  - 32 байта на каждый объектный файл
  - Библиотеки/исполняемые файлы включают объединение аннотаций

Размер файлов

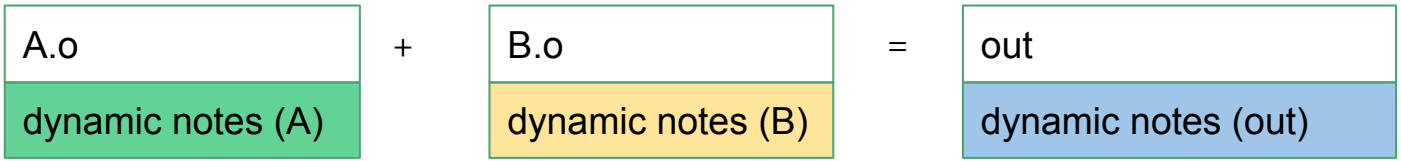
- Динамические аннотации (загружаемые):
  - 32 байта на каждый объектный файл
  - Библиотеки/исполняемые файлы включают объединение аннотаций

$ gcc A.o B.o -o out

Размер файлов

- Динамические аннотации (загружаемые):
  - 32 байта на каждый объектный файл
  - Библиотеки/исполняемые файлы включают объединение аннотаций

$ gcc A.o B.o -o out



min stack size: 4096

min stack size: 8192

min stack size: 8192

Размер файлов

- Статические аннотации (незагружаемые)
  - Зависит от размера исходного кода, от 1 до 50 KB на реальных проектах (gcc/clang/chromium)
  - Библиотеки/исполняемые файлы включают сумму всех статических аннотаций, поэтому по аналогии с дебаг информацией окончательных размер измеряется уже в Mb

Размер файлов

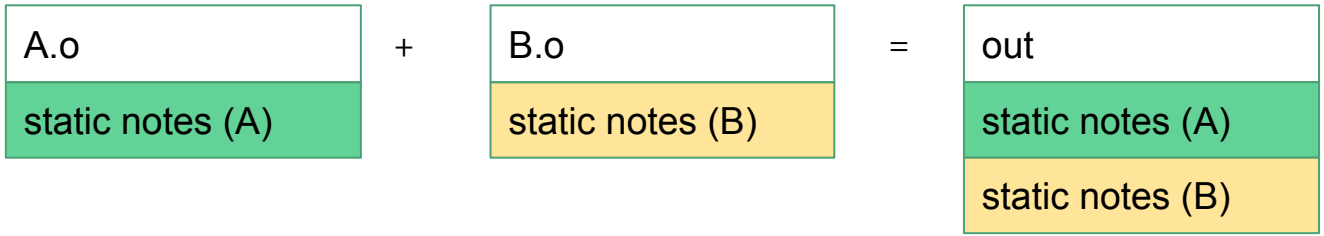
- Статические аннотации (незагружаемые)
  - Зависит от размера исходного кода, от 1 до 50 KB на реальных проектах (gcc/clang/chromium)
  - Библиотеки/исполняемые файлы включают сумму всех статических аннотаций, поэтому по аналогии с дебаг информацией окончательных размер измеряется уже в Mb

$ gcc A.o B.o -o out

Размер файлов

- Статические аннотации (незагружаемые)
  - Зависит от размера исходного кода, от 1 до 50 KB на реальных проектах (gcc/clang/chromium)
  - Библиотеки/исполняемые файлы включают сумму всех статических аннотаций, поэтому по аналогии с дебаг информацией окончательных размер измеряется уже в Mb

$ gcc A.o B.o -o out



Спасибо за внимание!

Links:

- Watermark specification:

  https://fedoraproject.org/wiki/Toolchain/Watermark

- GNU Tools Cauldron 2019 presentation:

  https://gcc.gnu.org/wiki/cauldron2019#cauldron2019talks.Annobin_A_Tale_of_A_GCC_Plugin_s_Lows_Highs

- Annobin git repository:

  git://sourceware.org/git/annobin.git