ИСП РАН

Разработка обфусцирующего компилятора на базе LLVM

Курмангалеев Шамиль
kursh@ispras.ru

ИСП РАН

Обфусцирующий компилятор

- Применяется во время развертывания
- Защита от обратной инженерии
- Простановка водяных знаков, уникальный код программы для каждого клиента
- Защита от эксплуатации уязвимостей
  - Неработоспособность эксплоита на сборках для разных клиентов

ИСП РАН

Критерии эффективности методов обфускации

- Маскирующее преобразование должно затрагивать и поток управления, и поток данных запутываемой программы
- Стойкость преобразования должна основываться на алгоритмически сложных задачах, например, требовать от атакующего применения анализа указателей, для точного восстановления потоков данных защищенной программы
- При разработке преобразования нужно учитывать особенности работы средств анализа, например для автоматических декомпиляторов следует насытить граф потока управления несводимыми участками

ИСП РАН

Подход к реализации

- Многие алгоритмы обфускации требуют наличия информации характерной для компиляторов
- Встраивание защиты во время компиляции позволяет увеличить ее стойкость и скорость разработки защиты
- Во время компиляции мы обладаем максимальной информацией о программе
- Автоматическая поддержка нескольких целевых архитектур

Требуется комиляторная инфраструктура

ИСП РАН

LLVM

Компиляторная инфраструктура с открытыми исходными кодами

- **Модульная и расширяемая архитектура**
- **Является статическим компилятором, а так же имеет возможность JIT'тить биткод**

Поддерживает несколько фронтэндов

- **C, C++, Objective-C (Clang, GCC/dragonegg)**
- **Ruby (Rubinius, MacRuby)**

Поддерживает множество целевых архитектур

- **ARM, Alpha, Intel x86, Microblaze, MIPS, PowerPC, SPARC, …**

Промежуточное представление играет центральную роль в процессе компиляции (LLVM IR)

- **Все оптимизации реализованы как компиляторные проходы преобразования "LLVM IR to LLVM IR"**
- **Анализ кода, может быть реализован как отдельный проход, а его результаты могут разделять несколько проходов трансформирующих код**
- **Все машинно-зависимые оптимизации происходят в отдельном бэкэнде для каждой машины**

ИСП РАН

Методы усложнения кода

- **Основные**
  - **Перемещение локальных переменных в глобальную область видимости**
  - **Приведение графа потока управления к плоскому виду**
  - **Переплетение нескольких функций в одну**
  - **Сокрытие вызовов функций**
  - **Создание несводимых участков в графе потока управления**
- **Вспомогательные**
  - **Шифрование константных строк, используемых программой**
  - **Вставка в код фиктивных циклов, из 1 итерации (do-while)**
  - **Размножение тел функций**
  - **Разбиение целочисленных констант**

Некоторые из указанных методов используют непрозрачные предикаты

Обеспечение детерминированности преобразований

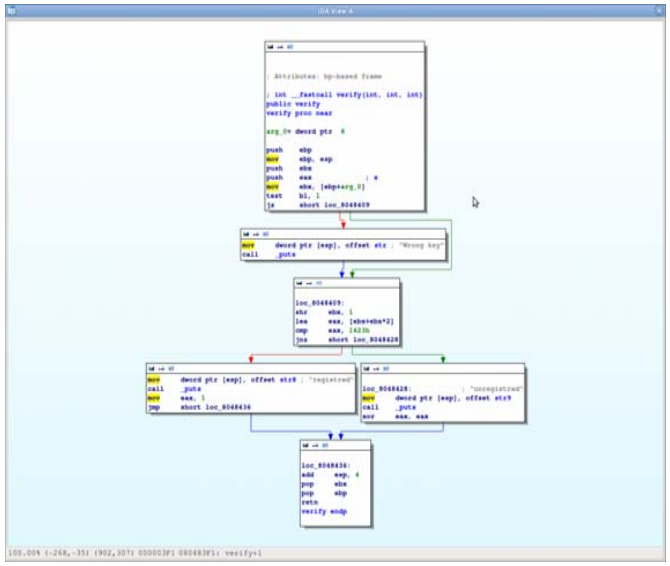
ИСП РАН

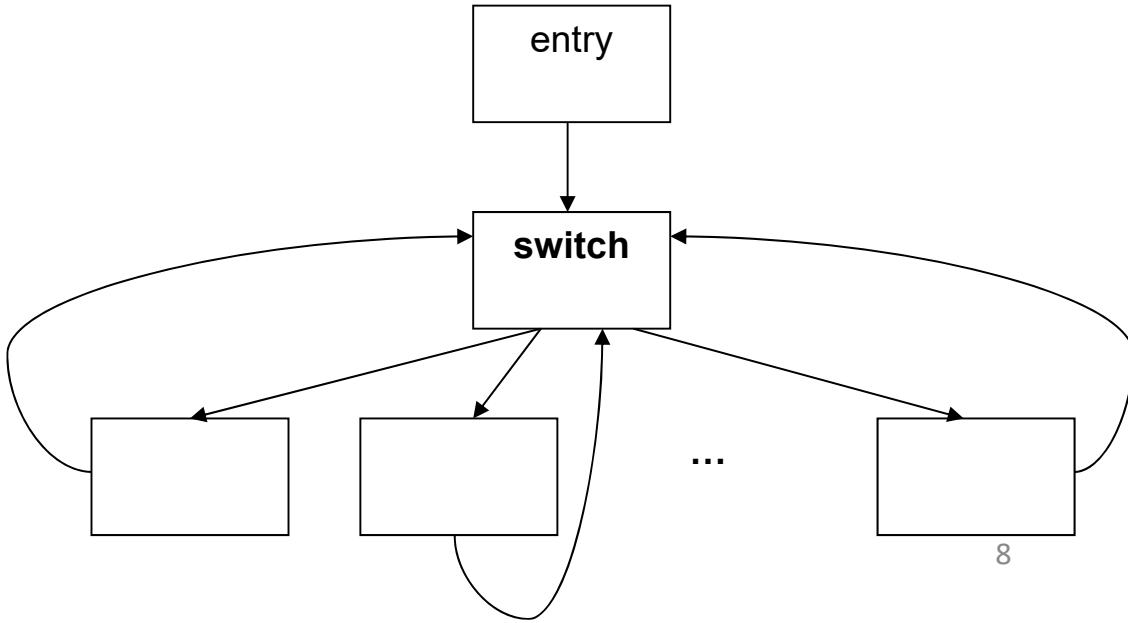
- В запутывающих преобразованиях используется линейный конгруэнтный генератор случайных чисел.
- Ключ пользователя выступает в роли «затравки» для генератора, накладывая на программу уникальный для каждого пользователя характер изменения программы

ИСП РАН

Маскирующее преобразование «диспетчер»

- В начало функции вставляем блок «диспетчер» - аналог switch в языке С
- Усредняем размер базовых блоков
- Создаем несколько копий каждого базового блока
- В конец каждого блока дописываем переменную содержащую номер следующего блока
- Переменная диспетчеризации сцепляется посредством **xor** с «живыми» переменными в каждом блоке





ИСП РАН

Формирование непрозрачных предикатов

Поддерживается три типа непрозрачных предикатов, например:

1. Истинность диофантова уравнения:
   $x^2 - n * y^2 = 1$ – истинность/ложность задается на этапе компиляции
2. Предикат, основанный на модульной арифметике:
   $(x^3 - x) \bmod 3 = 0$, это выражение **всегда истинно**
3. Целочисленное уравнение: $7 * y^2 - 1 = x^2$, это выражение **всегда ложно**

Для вариантов 2 и 3 значения переменных x и y случайным образом выбираются среди глобальных целочисленных переменных

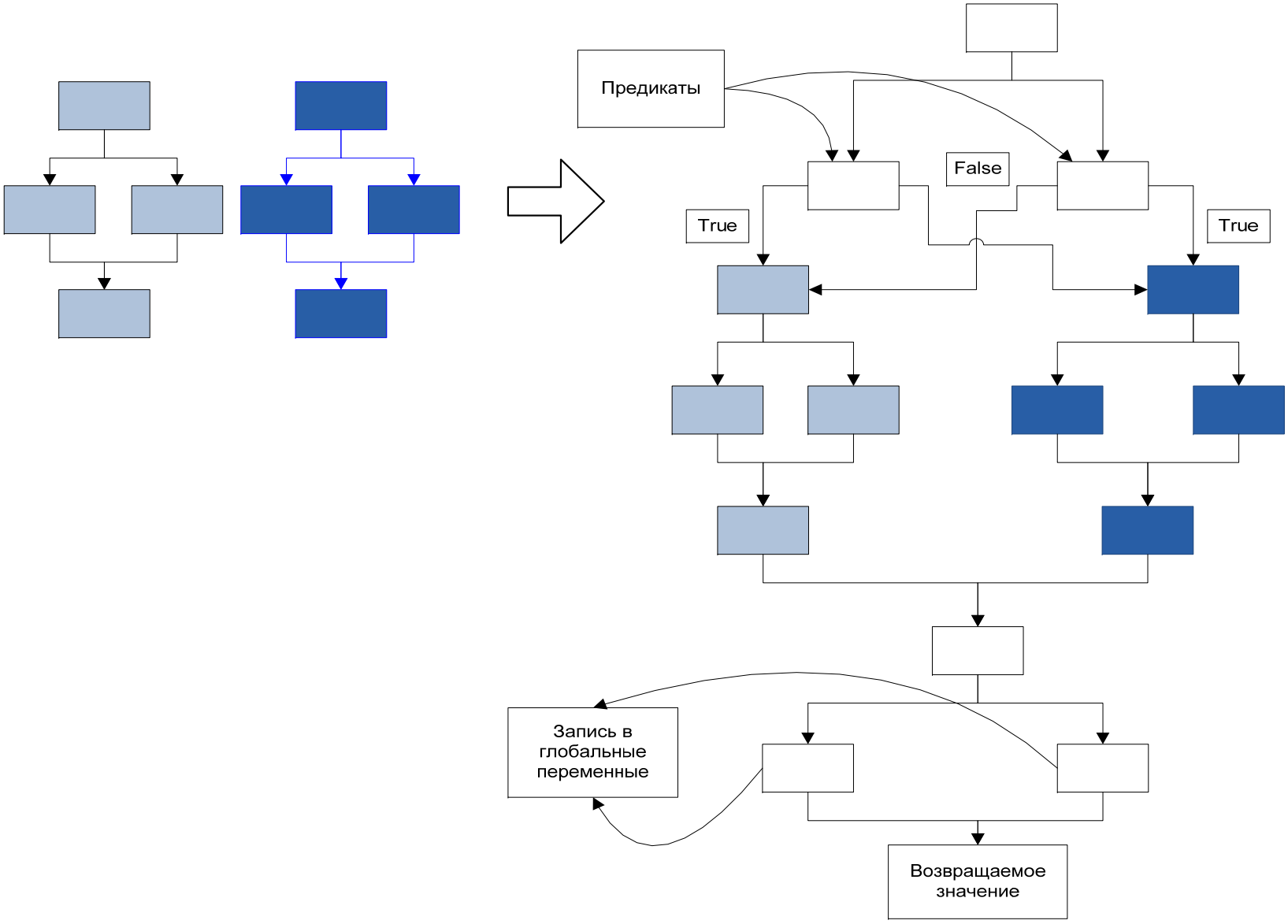
ИСП РАН

Маскировка вызовов внешних функций

- Создаем функцию-переходник, вызывающую несколько различных функций:
  - switch (r1 % (p * q))
    - Case fn: call fn();
- Половина функций в переходнике с похожими сигнатурами, половина с различными
- Аргументы для функции шифруются с помощью операции **xor,** и дешифруются в переходнике перед вызовом функции

ИСП РАН

Переплетение функций



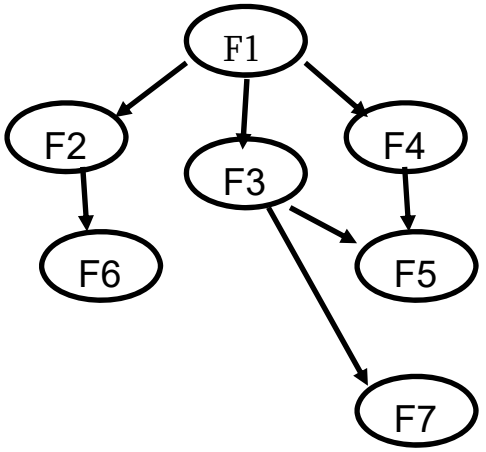
ИСП РАН

Увеличение сложности анализа потока данных в программе

Идея преобразования состоит в следующем:

- Локальные переменные перенести в глобальные
- В разных функциях получать ссылки на эти переменные
- Изменять эти переменные используя ссылки

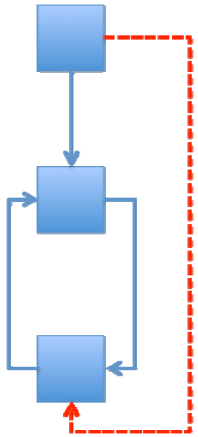
Анализируя граф вызовов для каждой функции можно вычислить множество переменных, модификация которых не нарушит работоспособности программы



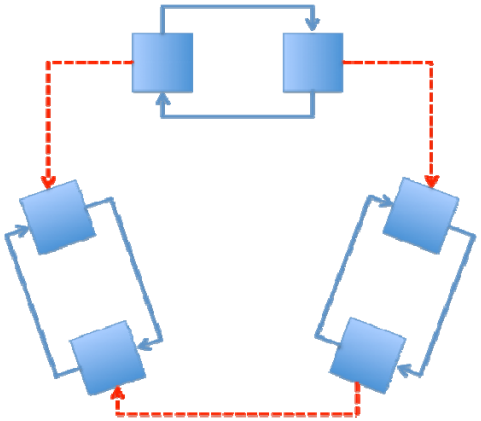
ИСП РАН

Приведение графа потока управления к несводимому

- Преобразования над циклами



1



2

- Запутывание всего графа: из множества базовых блоков функции выбирается N количество блоков и между ними случайно добавляются ребра с использованием непрозрачного предиката

ИСП РАН

Оценка качества

- Тестирование производилось на пакете OpenSSL
- Замедление для одного запутывающего преобразования составляет 1,2-5,5 раз
- Размер программы увеличивается в 1,05-2,5 раз
- Для примерной оценки сложности анализа, был проведен эксперимент. К программе Sqlite были применены следующие преобразования:
  - переплетения функций
  - перевода переменных в глобальную область видимости
  - преобразования диспетчеризации
  - сокрытия вызовов функций

Оценка качества

Размер кода приложения увеличился с 2.9 МБ до 15 Мб.

Потребление памяти дизассемблером Ida Pro возросло в ~10 раз.

Время анализа возросло примерно в 10 раз, затем произошло исключение в библиотеках дизассемблера

- Также было произведено исследование с помощью инструмента комбинированного анализа TrEx, полученные результаты свидетельствуют о том, что обеспечиваемый уровень защиты сравним с уровнем, обеспечиваемым коммерческими разработками

ИСП РАН

Спасибо за внимание

? Вопросы ?