НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

Динамическая верификация ядра операционной системы для анализа выполнения высокоуровневых требований защиты информации

Аспирант 2-го года обучения (напр. 05.13.11):
Ефремов Денис (defremov@hse.ru)

Научный руководитель:
проф., д-р физ.-мат. наук Петренко А.К.

Верификация

Верификация - проверка соответствия программного обеспечения предъявляемым к нему требованиям;

Дедуктивная верификация – представление корректности программы как набора математических утверждений, называемых условиями верификации, выполнение которых проверяется автоматическими или интерактивными доказателями теорем;

Спецификация - набор требований и параметров, которым удовлетворяет некоторый объект (представлена в виде мат. модели, тестовых наборов, формальной спецификации)

Требования стандартов

- Orange Book: division A (verified protection) (A1, Beyond A1)
- Common Criteria (EAL7)
- DO-178C/DO-333 "Formal Methods Supplement to DO-178C and DO-278A”
- IEC 61508 (SIL4)
- ФСТЭК России ГОСТ Р ИСО/МЭК 15408 «Требованиях безопасности информации к операционным системам» профили защиты операционных систем общего назначения (типа «А»)

Высокоуровневые требования защиты информации в виде модели безопасности

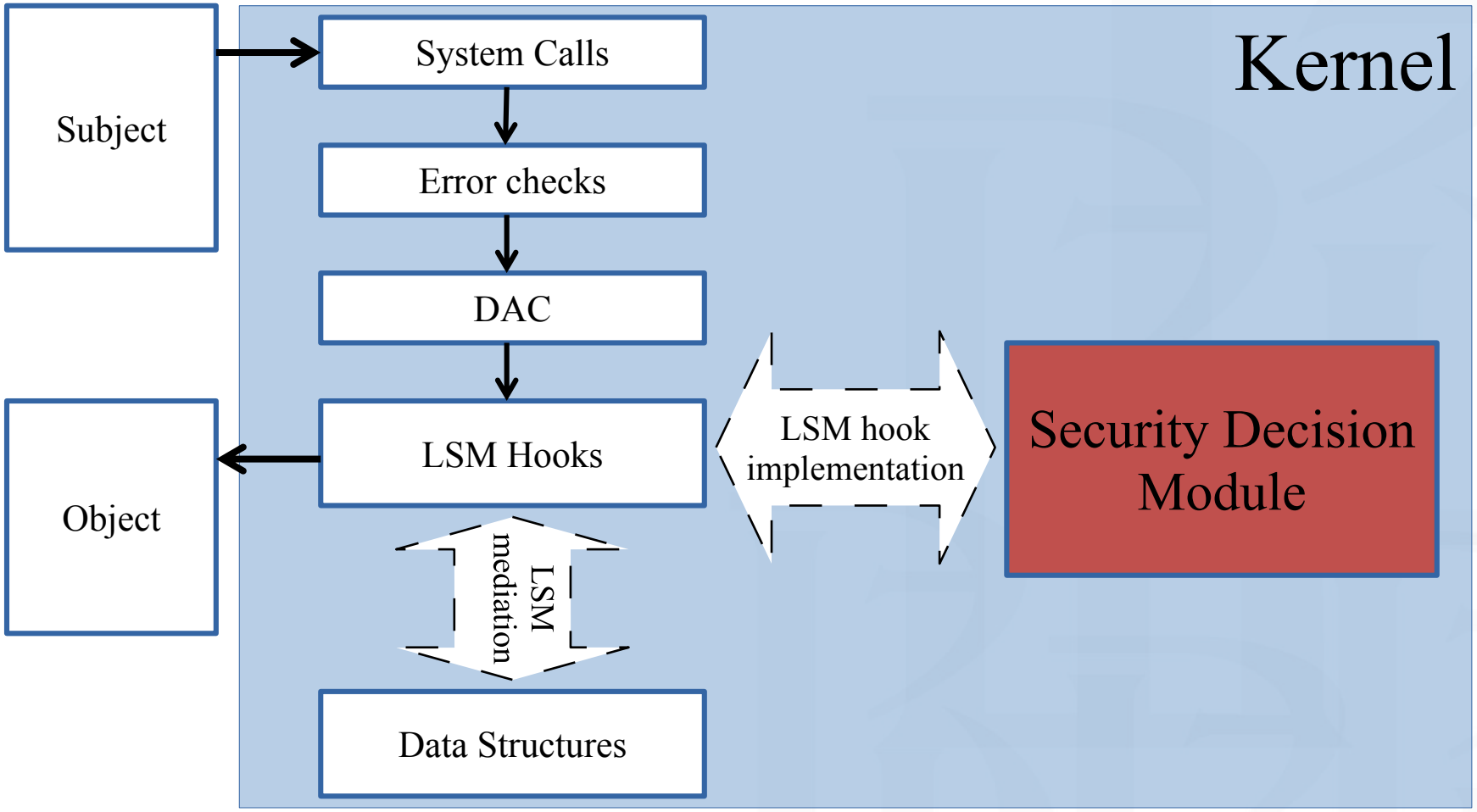
- Модель безопасности формализована в нотации Event-B
  - Организация модели по слоям
  - Глобальное состояние
  - Инварианты, описывающие глобальное состояние
  - Операции (переходы)
    - Предусловия (guard)
    - Действия (actions), обновляющие глобальное состояние
- Высокоуровневые требований безопасности (инварианты) доказаны на модели

Пример события модели Event-B

```
event create_user
  any user // parameter
      profile // parameter
      session // parameter
  where
    @grd1 user∈UserAccounts\CurrentUserAccounts
    @grd2 profile∈ℙ1(CurrentEntities)
    @grd3 session∈CurrentSessions
  then
    @act1 CurrentUserAccounts ≔ CurrentUserAccounts ∪ {user}
    @act2 UserProfiles(user) ≔ profile
end
```

Высокоуровневые требования защиты информации в виде модели безопасности

Модуль безопасности ядра Linux (LSM), реализующий требования модели



Вопрос соотношение модели с реализацией (1)

- Модель – высокоуровневая
  - Оперирует сущностями высокого уровня (файл, пользователь)
- Модель – абстрактная
  - Исходит из определенных представлений о том, как функционирует «обобщенная» ОС
  - Не содержит в себе никаких привязок к оборудованию
- Модель описывает поведение системы «в целом»
  - Ядро, его драйверы и программы

Вопрос соотношения модели с реализацией (2)

- Реализация модели сосредоточена в модуле безопасности ядра
- Модуль безопасности реализован на основе LSM интерфейса ядра
- Полагается на корректность реализации ядром интерфейса LSM
- Оперирует низкоуровневыми сущностями (идентификаторы, файловые дескрипторы, индексные дескрипторы и т.д.)
- В какой-то мере зависит от функционирования оборудования
- Написан на языке Си (ошибки, специфичные для языка Си)

Проверка реализации: дедуктивная верификация модуля безопасности ядра Linux (LSM) (1)

Если выполнен ряд предположений:

- Компилятор и линковщик работают корректным образом
- В программном обеспечении, использующемся при верификации, не произошло ошибок
- Компьютер функционирует таким образом, как мы думаем об этом (rowhammer)
- Пользователь компьютера, если он есть, специально не «пакостит»
- Выполнены предположения о входных данных программы, о начальном состоянии
- …

Проверка реализации: дедуктивная верификация модуля безопасности ядра Linux (LSM) (2)

Гарантии того, что программное обеспечение функционирует в точном соответствии с требованиями, к нему предъявляемыми, на всех входных данных, начальных состояниях, при любом поведении окружения * **

* В предположении что все предположения выполнены

** В соответствии с моделями, использующимися в верификационном ПО

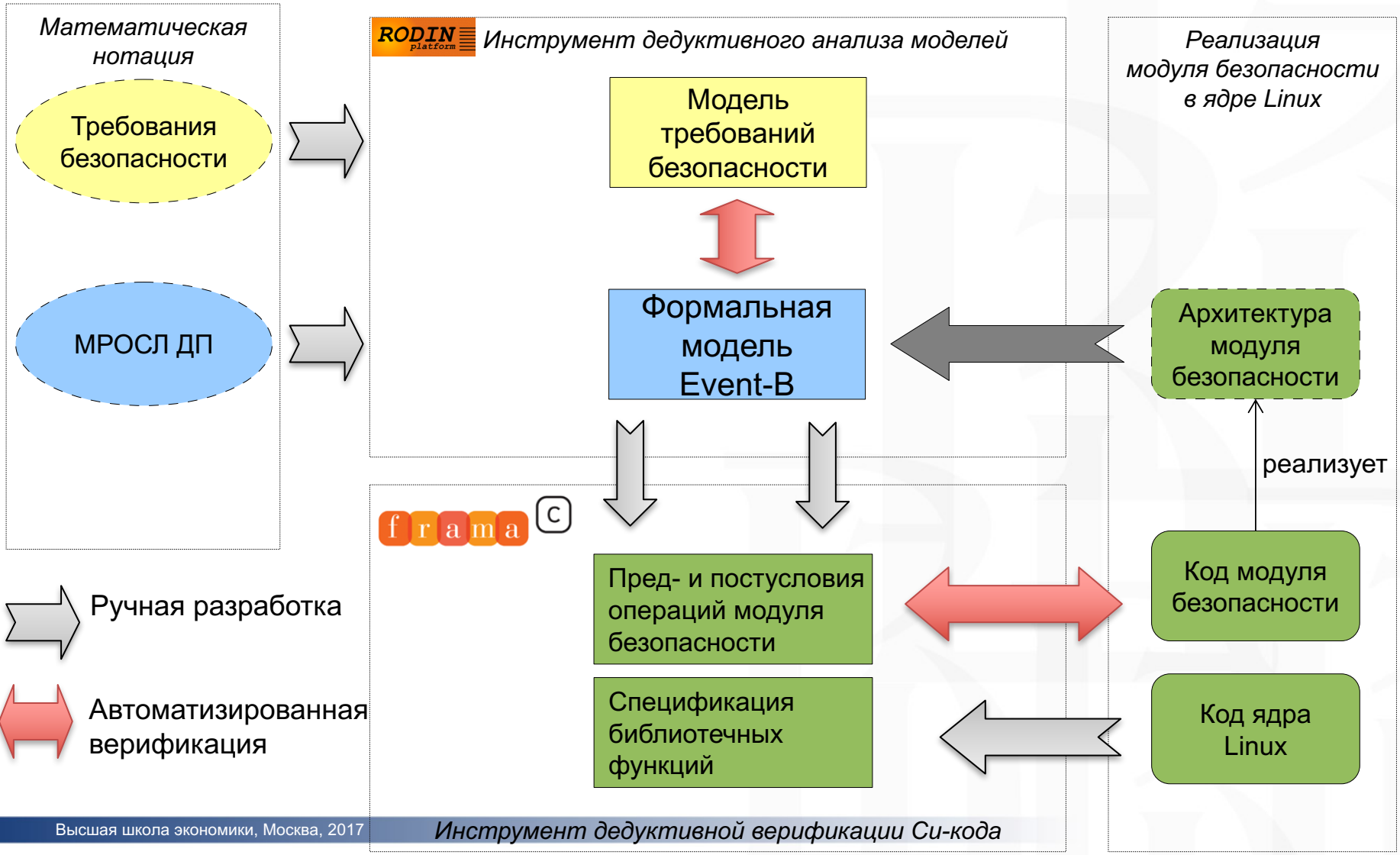
Пример спецификации функции бинарного поиска

```
/*@ requires \valid(a + (0..n-1));
    requires \forall integer i, j; 0 <= i < j < n ==> a[i] <= a[j];
    assigns \nothing;
    ensures 0 <= \result <= n;
    ensures \forall integer k; 0 <= k < \result ==> a[k] < val;
    ensures \forall integer k; \result <= k < n ==> val <= a[k];
*/
unsigned lower_bound(const int *a, unsigned n, int val) {
   ...
   /*@ loop invariant 0 <= left <= right <= n;
       loop invariant \forall integer i; 0 <= i < left ==> a[i] < val;
       loop invariant \forall integer i; right <= i < n ==> val <= a[i];
       loop variant right - left;
   */
   while (left < right) {
      middle = left + (right - left) / 2;
      if (a[middle] < val) {
         left = middle + 1;
      } else right = middle;
   }
```

Проверка реализации: дедуктивная верификация модуля безопасности ядра Linux (LSM) (3)

- Доказана корректность больше сотни функций модуля безопасности
- Сформулированы и доказаны некоторые модельные высокоуровневые требования безопасности
- Разработаны спецификации и доказана корректность ряда библиотечных функций ядра, на использование которых опирается модуль безопасности
  - https://github.com/evdenis/verker
- Описан контекст вызова (предусловия) части LSM интерфейса
- Произведено исследование возможности частичной трансляции спецификаций в проверки времени выполнения (assert)
- Разработаны инструменты для автоматизации работы со спецификациями и кодом
- Показана невозможность доказательства всех требований модели безопасности на исходном коде модуля безопасности с помощью подхода дедуктивной верификации

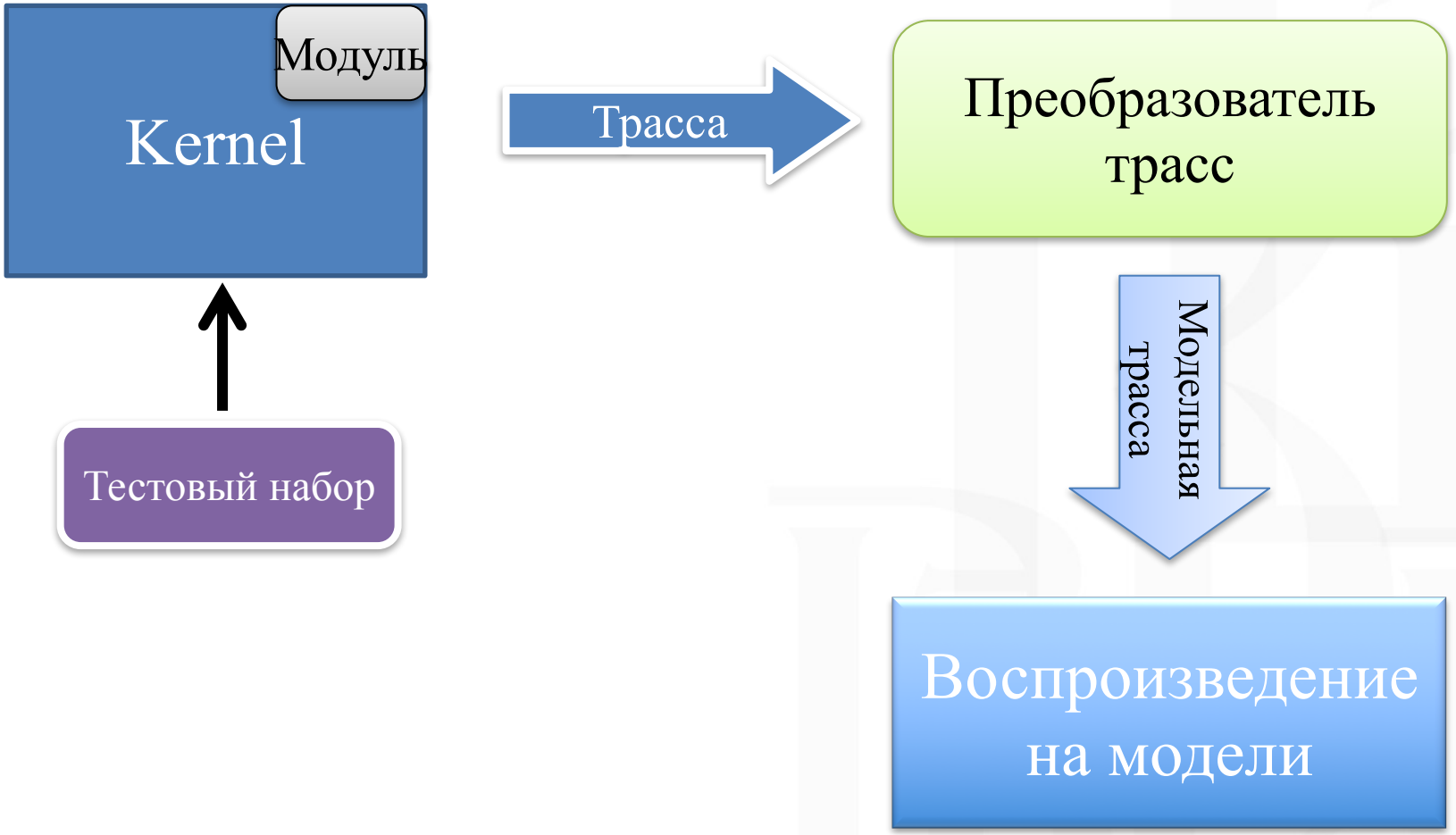
Общая схема верификации



Динамическая верификация ядра ОС на соответствие модели

- Выбор отслеживаемых событий в модели
- Выбор точек в ядре Linux, соответствующих модельным событиям
- Сбор трасс в динамике
- Проверка трасс на модели

Динамическая верификация ядра ОС на соответствие модели



Выбор точек в ядре Linux, соответствующих модельным событиям

Ядро – обработчик внешних сигналов от приложений, оборудования

Обработчики:

- Системных вызовов
- Прерываний
- Исключений
- LSM интерфейс

Сбор трасс в динамике

- Инструмент SystemTap для мониторинга событий в ядре
- Точек сбора событий ~300
- ~ 30 обработчиков для разных событий внутри ядра с записью в трассы
- Модификация модели под реалии ядра
- Воспроизведение трассы на модели после сбора полных трасс на ядре

Проверка модельных трасс на модели (1)

- Нет существующего решения для воспроизведения трассы событий на Event-B
- Есть инструмент моделирования событий на модели ProB
- Подбор модельных событий для симуляции воспроизведения на основе алгоритма унификации языка Prolog (fail)

Проверка модельных трасс на модели (2)

- Трансляция модели на Event-B в исполняемую
  - Не все конструкции языка моделирования возможно перевести в код
  - Существующие трансляторы на Си, Java не работают на существующей модели
  - Язык трансляции Haskell
- Воспроизведение трассы в сгенерированной программе
- Возможность интеграции в дистрибутив «бинарной модели»

Тестовый набор и мониторинг покрытия

- Стандартные тестовые наборы на подсистемы ядра
- Тестовый набор от разработчиков модуля безопасности
- Мониторинг покрытия по коду – GCOV
- Инструмент Fault Injection для расширения тестового покрытия - KEDR
- Фаззинг syzkaller
- Concolic инсполнение - S2E

Преимущества и недостатки метода

- Адаптируемость к изменениям в ядре, в модели, в реализации модуля безопасности
- Масштабируемость
- Отсутствие ложно-положительных срабатываний
- Возможность осуществления постоянной real-time проверки
- Результат зависит от тестового набора
- Накладные расходы на мониторинг ~30% на данный момент

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

Спасибо за внимание!