ИСП РАН

Анализ программ: эффективность и безопасность

Арутюн Аветисян
arut@ispras.ru

Вызов – обеспечение качества ПО

- Качество современных программных систем определяется тремя взаимосвязанными компонентами:
  - эффективность
  - ***безопасность***
  - переносимость
- Обеспечение требуемого качества ПО связано с использованием ручного труда высококвалифицированных специалистов и является искусством (существенное увеличение стоимости и сроков разработки ПО)

Безопасность

В настоящее время уровень информационной безопасности в значительной мере определяется наличием **ошибок в ПО и их эксплуатируемостью**

Ошибки* в исполняемом коде могут приводить к:

- Потере стабильности работы программы
- Уязвимостям защиты (security vulnerability) – потенциальная возможность обойти средства разграничения прав доступа программы или ОС, в которой программа выполняется, создание эксплойта

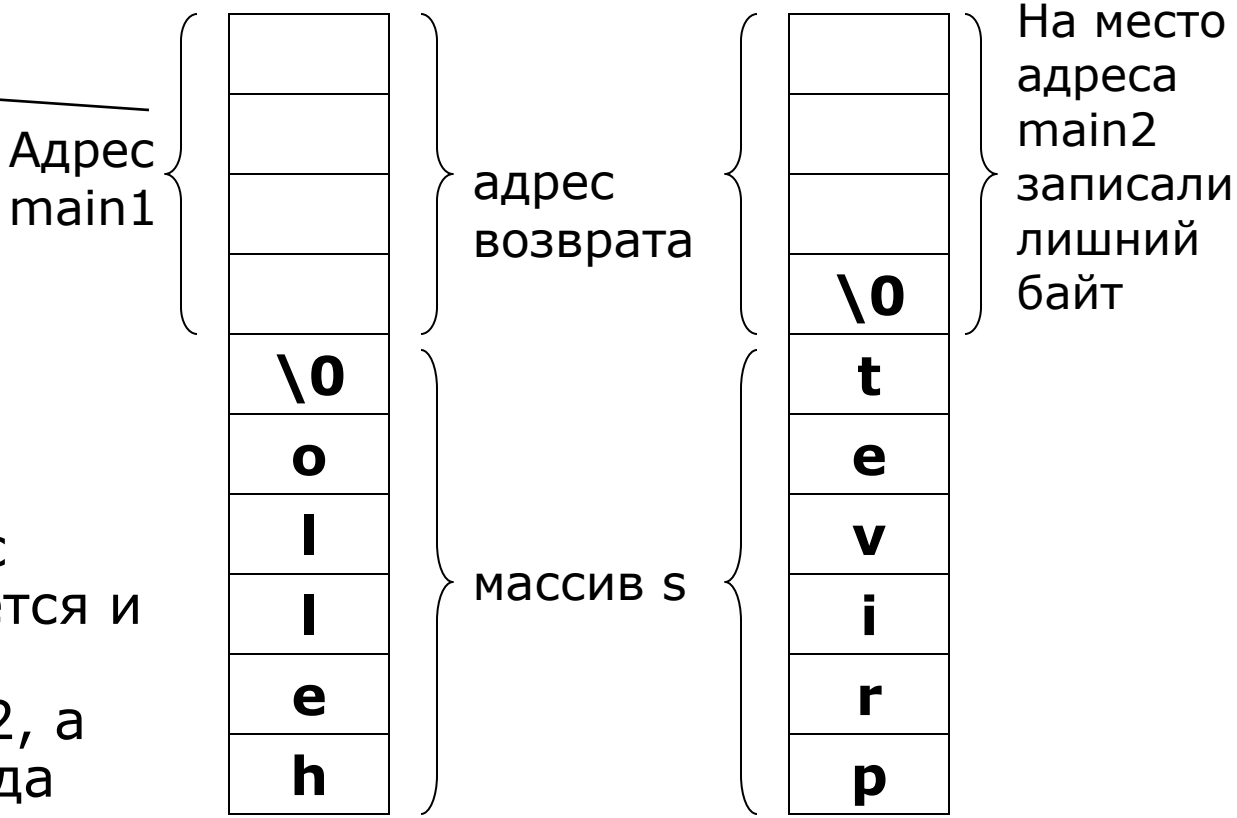
*Границы между ошибками программиста, закладками, НДВ размыты

Ошибки. Примеры.

- Heartbleed (OpenSSL) – чтение памяти на сервере или клиенте. Конец 2011- апрель 2014. На момент публикации уязвимости было подвержено около 0.5 млн сайтов (CVE-2014-0160)
- Клиент OpenSSH версии от 5.4 до 7.1, утечка приватного ключа клиента, возможна атака MITM при первом подключении к новой системе (CVE-2016-0777). Трудно сказать ошибка или закладка.
- Ядро Linux от 3.8 до 4.5 - локальный пользователь может получить права суперпользователя (CVE-2016-0728)
- FreeBSD версии 9.3, 10.1 и 10.2 , отказ в обслуживании, повышение привилегий, раскрытие данных (CVE-2016-1881,CVE-2016-1880, CVE-2016-1879, CVE-2016-1882, CVE-2015-5677)
- Apple IOS 6.0-9.2 и OSX 10.0-10.11.2, повышение привилегий, выполнение произвольного кода, отказ в обслуживании (CVE-2016-1722)

Пример (переполнение буфера)

```
f(char * p)
{     char s[6];
      strcpy(s,p);
}
main1 ()
{       f("hello");
}
main2 ()
{       f("privet");
}
```



В случае main2 адрес возврата перезапишется и управление будет передано не на main2, а на другой участок кода

Пример (ошибки непроверенного ввода)

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    char buf[80], cmd[100];
    fgets(buf, sizeof(buf), stdin);
    snprintf(cmd,sizeof(cmd),"ls -l %s", buf);
    system(cmd);
    return 0;
}
```

Например: `ls -l myfile ; rm -rf /`

Тенденции развития (I)

- Эскалация размеров и сложности

Система	Год	Размер (10^6 LOC)
Дистрибутив Debian Sid	2016	>1166
Android 5	2014	>47
Android 6	2015	>39
Tizen 3	2015	>134
Linux Kernel 4.4	2016	>20

- Широкое использование Свободного ПО
  (одна ошибка тиражируется во многих продуктах, например, уязвимость в OpenSSL)
- Возрастание сложности среды разработки и сборки
  (например, компилятор может добавить уязвимость*, которой нет в исходном коде)

```
*memset (password, '\0', len);
 free (password);
```

Тенденции развития (II)

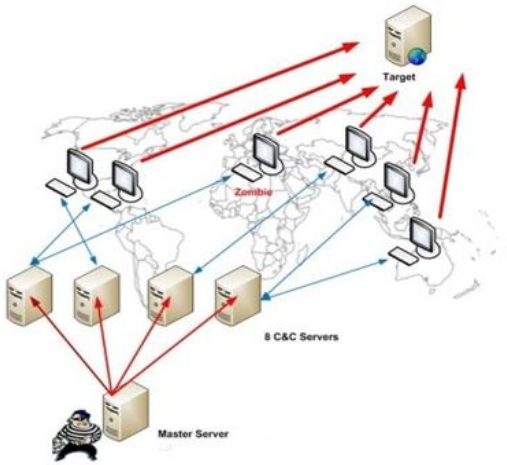
- Сервис-ориентированные центры обработки данных («облачные» инфраструктуры)
- Массовое внедрение мобильных платформ
- Массовое внедрение встроенных систем в физические объекты и их объединение в единую сеть – «Интернет вещей»

Традиционные подходы защиты по периметру недостаточны

Ошибки – существенный рост ущерба

Аварии на критических обьектах

Перехват управления



Бот-неты

Уязвимости

Кража паролей

Кража информации о кредитных картах

Ошибки. Статистика

- Debian Linux:
  - 1892 критических ошибки в текущем релизе
  - исправлены 229
  - ~ 140 программ содержат не исправленные CVE с 2010 по 2016 год
- Кластер из 100 узлов, тестирование заняло 28 дней машинного времени* (проанализировано 37 тыс. программ из состава Debian Linux)

*от авторов инструмента ***Mayhem***, на Debian (David Brumley, Thanassis Avgerinos) forallsecure.com/debian

ИСП РАН

Подходы к выявлению ошибок в ПО

Традиционные: экспертиза (опыт и знания людей), тестирование – недостаточны

Одним из перспективных направлений исследований является развитие методов анализа программ (*исходного и бинарного кода*) и разработка соответствующих инструментов:

- Статический анализ
  - Поиск возможных дефектов в коде по некоторым шаблонам
- Динамический анализ
  - фаззинг, символьное выполнение, слайсинг, профилирование, …
- Комбинированный анализ
  - Статический/динамический анализ с использованием (неполных) формальных моделей

Дефект vs. Ошибка

Несоблюдение свойств модели некоторого уровня

- вычисления
  - деление на ноль, защита памяти, …
- язык программирования
  - выход за пределы буфера, …
- архитектура приложения
  - состояние гонок, …
- логика приложения
  - политика безопасности – неправомерный доступ к данным

Статический анализ исходного кода (I)

Активные исследования и разработки по созданию инструментов статического анализа (без выполнения кода) исходного кода, обеспечивающих автоматическое обнаружение дефектов в больших объемах кода (десятки миллионов строк) ведутся с середины 2000-х гг.

Разработан ряд стандартов (что искать?): *CWE, CWE/SANS Top 25, CERT, OWASP, DISA STIG, MISRA*

Статический анализ исходного кода (II)

Требования:

- Необходимость анализа систем в целом (межпроцедурный анализ, анализ указателей, анализ циклов, модель памяти)
- Анализ проекта размера Android и Tizen должен длиться 4-6 часов (ночная сборка)
- Поиск ошибок разного уровня критичности (от неверного форматирования, до переполнения буфера)
- Поддержка популярных языков программирования
- Высокий уровень истинных срабатываний

Преимущества:

- Автоматический анализ многих путей исполнения одновременно
- Обнаружение дефектов, проявляющихся только на редких путях исполнения, или на необычных входных данных (которые могут быть установлены злоумышленником в процессе атаки)
- Возможность анализа на неполном наборе исходных файлов

Виды статического анализа

Уровень работы анализа	Подходы	Инструменты
Лексический анализ	Поиск вызова заданных функций	ITS4, RATS, Flawfinder
Синтаксический анализ	Поиск шаблонов по AST	Lint, CppCheck, PVS-Studio
Семантический анализ	Анализ потока данных	*HP Fortify*, FindBugs, PC-Lint, CodeSonar, Parasoft C/C++test
	Символьное выполнение	Clang Static Analyzer, Coverity SAVE, Klockwork Insight, **Svace (ИСП РАН)**

Svace – автоматическое обнаружение дефектов (I)

ИСП РАН

- Базируется на **модели** анализируемого кода (абстрактные ячейки памяти, их атрибуты и значения атрибутов, контексты инструкций), которая создается из промежуточного представления, генерируемого компилятором LLVM
- Выполняется статический анализ модели:
  - ✓ итерационный статический внутрипроцедурный анализ (анализ указателей, анализ интервалов, ...)
  - ✓ межпроцедурный анализ, управляемый графом вызовов
  - ✓ абстрактная интерпретация (символьное выполнение)
- Возможность анализа неполных программ (модулей, библиотек)
- Расширяемость набора подсистем поиска дефектов
- В настоящее время поддерживаются языки Си, Си++, Java, C#, находятся в разработке JavaScript и Python

Svace – автоматическое обнаружение дефектов (II)

ИСП РАН

- Находит более 100 различных видов дефектов
  (переполнение буфера, разыменование нулевого указателя, проблемы синхронизации, нефильтрованный ввод, проблемы работы с памятью и др.)
- Нет ограничений на размер анализируемой программы:
  линейная масштабируемость (без потери качества анализа) – полный анализ Андроида (около 5 миллионов строк кода) за 3 часа
- Качество анализа и производительность сопоставимы с инструментами Coverity SAVE и Klockwork Insight
- Пользовательский интерфейс поддерживает распределенную работу с историей результатов анализа, как из командной строки, так и среды разработки Eclipse, в том числе, через web

Svace – автоматическое обнаружение дефектов (III)

ИСП РАН

Тип предупреждения	Истинные срабатывания Svace
Переполнение буфера	60%
Работа с динамически выделенной памятью	50%
Разыменование NULL	70%
Испорченный ввод	70%
Неинициализированные данные	60%
Несоотвестие типов возвращаемых значений	60%
Состояние гонки	90%
Передача по значению	100%
Другие (более 30)	50%

Анализ бинарного кода

ИСП РАН

Актуальность:

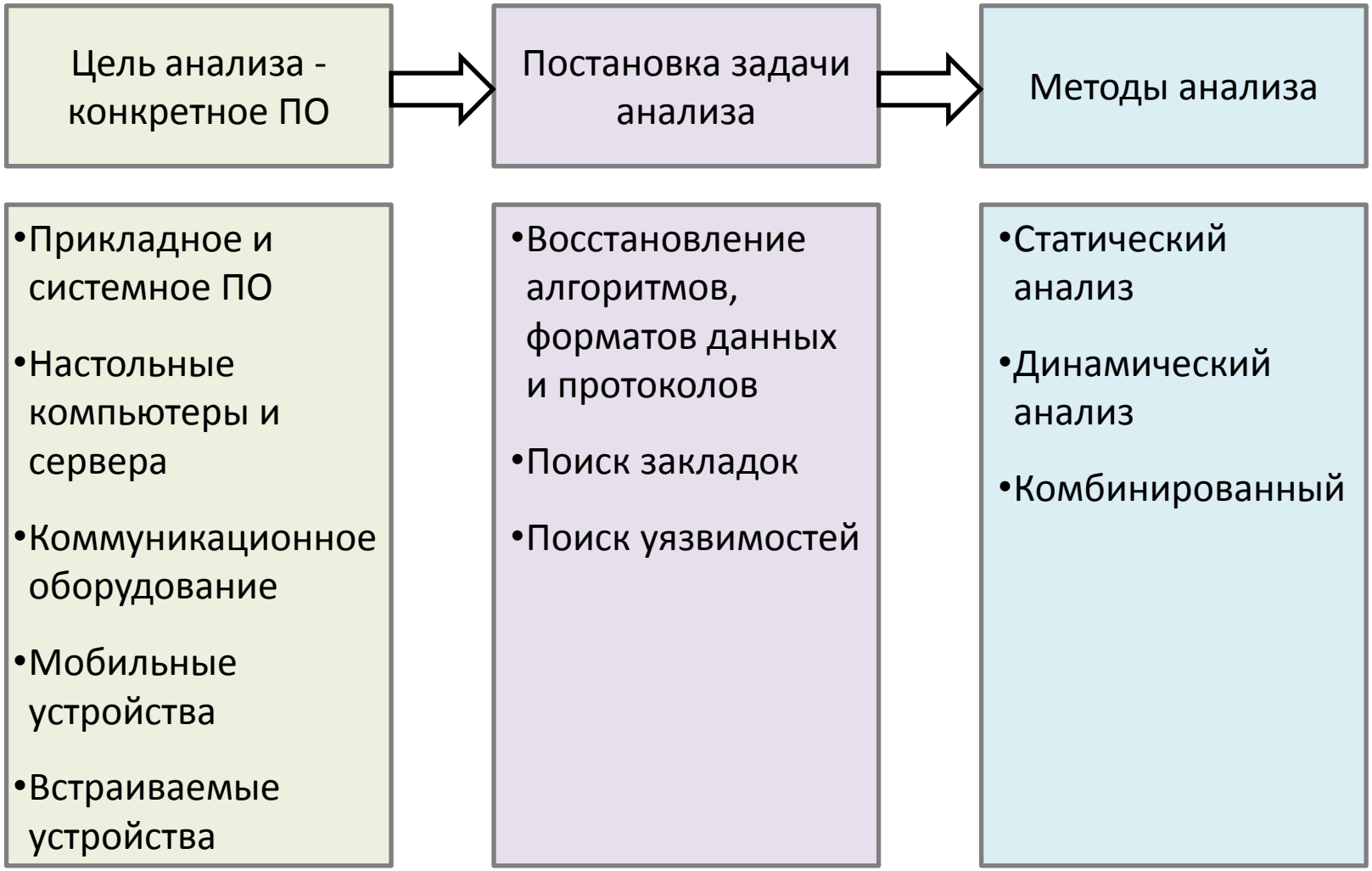
- Используется ПО, имеющееся только в бинарном коде, и/или «недостоверная» среда сборки
- Агрессивная оптимизация может вносить в код дефекты

Средства:

- Статический анализ (*трудно применять к программам, снабженным средствами защиты от анализа*)
- Динамический или комбинированный анализ

ИСП РАН

Цели и задачи анализа бинарного кода



Динамический анализ бинарного кода – Avalanche

ИСП РАН

- Avalanche – технология автоматического динамического анализа (C/C++, Java), основанная на обходе возможных трасс выполнения и поиске ошибок на этих трассах:
  - ✓ Инструментация кода для получения трасс выполнения в виде набора ограничений (C/C++ - Valgrind, Java – BCEL)
  - ✓ Разрешение ограничений с помощью солвера (используется солвер на основе свободного ПО STP)
  - ✓ Построение новых наборов входных данных
  - ✓ Поиск ошибок, в том числе критических:
    - NPD, деление на ноль найдены в mono, llvm, libjpeg, swftools, xmllint и др.;
    - состояния гонки – в Browser и Mms (Android)
- Достоинство – генерация входных данных, на которых проявляется дефект
- Недостаток – ресурсоемкость

Trex: среда комбинированного анализа бинарного кода (I)

ИСП РАН

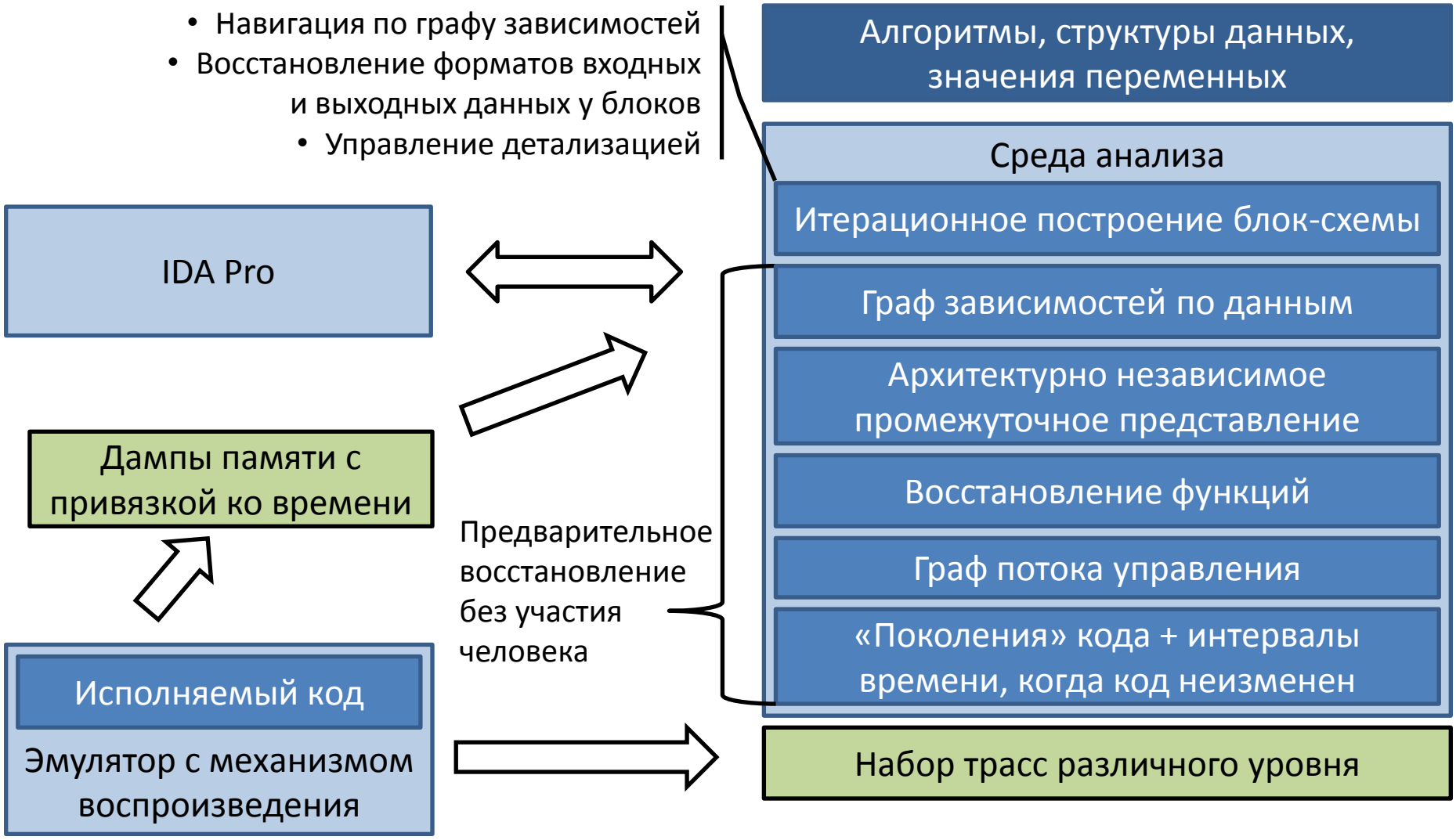
- Базируется на **модели** анализируемого бинарного кода, получаемой на основе трасс и состоящей из модели кода в платформо-независимом представлении и модели областей памяти (регистров, стека, областей статических и глобальных данных) – регистровая RISC-машина
- Обеспечивает:
  - ✓ сбор и систематический анализ трасс (объединение трасс, динамический слайсинг и др.)
  - ✓ интеграцию с результатами статического анализа и др.
- Позволяет восстанавливать:
  - ✓ граф потока управления интересующих функций
  - ✓ структуры входных и выходных параметров
  - ✓ протоколы обмена данными и др.

Trex: среда комбинированного анализа бинарного кода (II)

ИСП РАН

Автоматизированный анализ в диалоговом режиме

- Навигация по графу зависимостей
- Восстановление форматов входных и выходных данных у блоков
- Управление детализацией



Trex: среда комбинированного анализа бинарного кода (III)

- Восстановлен алгоритм проверки лицензии SmartPlant License Manager 2010
- Восстановлен алгоритм распаковки кода и формат файла, в котором этот код содержится
- Выявлены механизмы внедрения вредоносного кода в операционную систему (вирус/руткит Trojan.Win32.Tdss.ajfl)

Пример выделения инструкций требуемого алгоритма:

	Трасса, содержащая код анализируемой программы в пользовательском режиме			Число инструкций после обработки
	Размер, МБ	Число шагов	Число инструкций в листинге	
VB v6.0	42	575 392	26 620	356
VB .NET	35	484 248	62 726	143

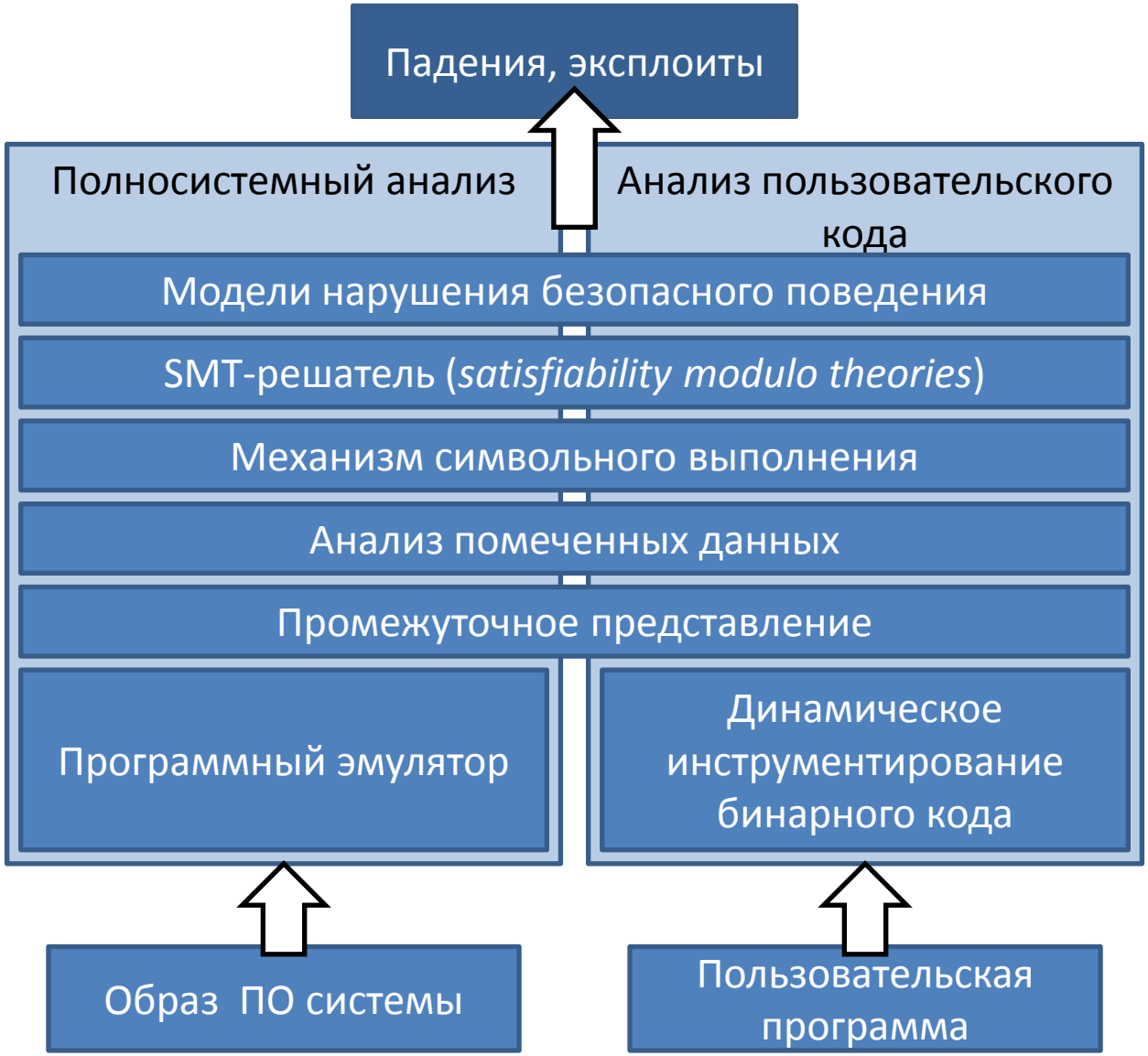
Главные цели анализа ПО
(*технологии двойного назначения*):

- Защита:
  - найти и устранить на этапах **разработки** и внедрения максимальное количество ошибок в исполняемом коде
  - предотвратить эксплуатацию существующих ошибок или смягчить последствия их эксплуатации
- Нападение:
  - Поиск уязвимостей и генерация эксплойта

Перспективное направление (I)

ИСП РАН

Автоматический поиск уязвимостей



Перспективное направление (II)

ИСП РАН

Смешанные техники

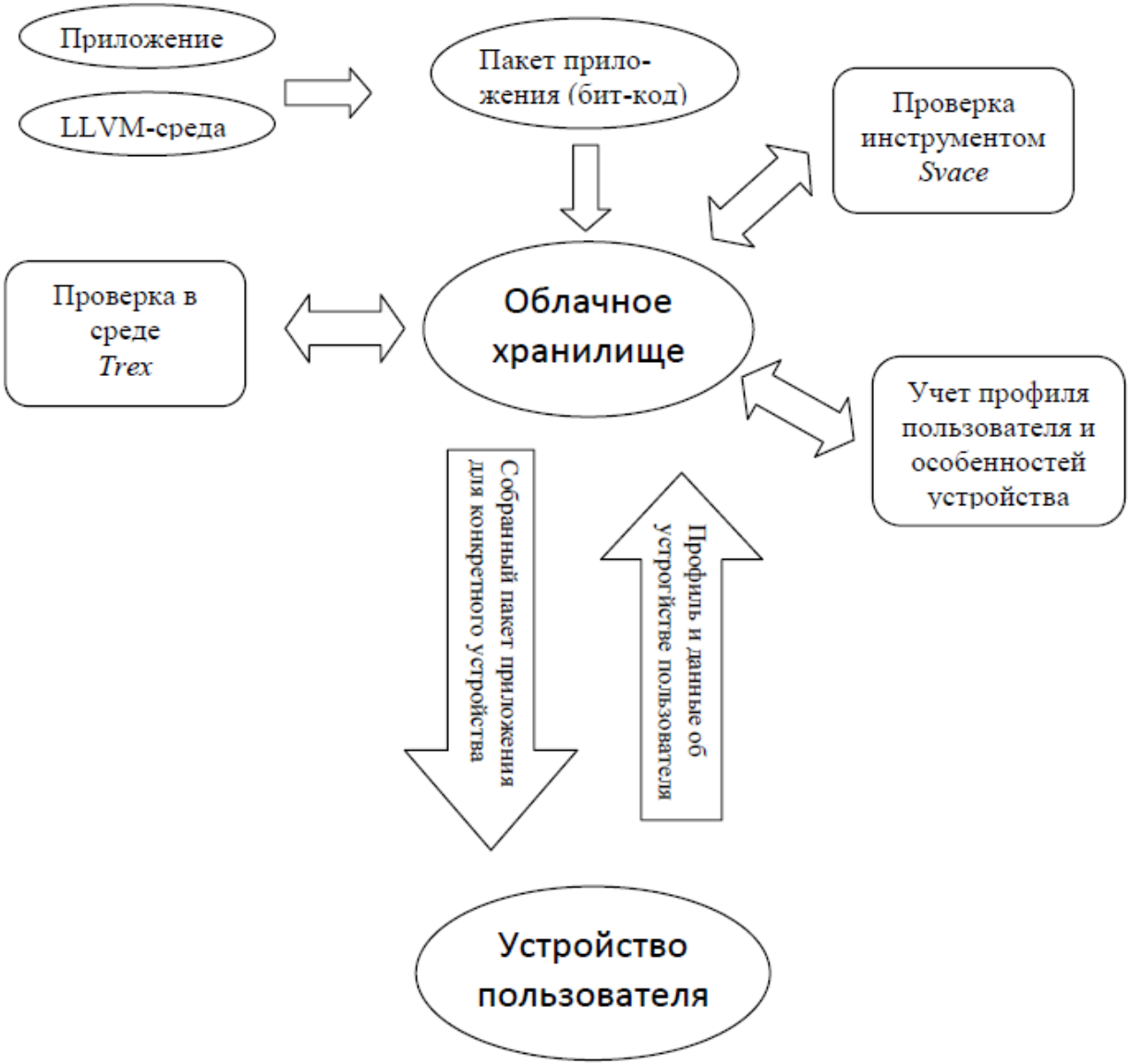
- Статический анализ предоставляет список ошибок ПО и условия их возникновения
  - Исходный код
  - Бинарный код
- Динамический анализ подбирает данные для исполнения пути приводящего к ошибке
  - Фаззинг
  - Символьное выполнение

Переносимость C/C++-приложений с сохранением эффективности

- Двухэтапная компиляция на основе LLVM:
  - ✓ распространение программы в биткоде LLVM
  - ✓ машинно-независимые оптимизации на стороне разработчика
  - ✓ машинно-зависимые оптимизации и оптимизации по профилю:
    в «облаке» (индивидуально для устройства)
    или статически/динамически на устройстве
  - ✓ возможность дополнительного статического анализа в облаке (выявление дефектов, обфускация, …)

- В настоящее время аналогичные работы ведутся в Apple

«Облачное хранилище» приложений нового поколения



ИСП РАН

Спасибо