ИСП РАН

Операционные платформы. Современные вызовы.

Арутюн Аветисян
Институт системного программирования РАН
arut@ispras.ru
23 мая, 2017

История развития ОС

Четыре этапа:

- Пакетная обработка заданий (1950–1960)
- Интерактивные системы (1960–1975)
- Десктопные системы (1975–2005)
- Облачно-мобильные платформы (2005–по н.в.)

Исторический опыт

Сотрудники ИСП РАН участвовали в ряде крупных проектов СССР таких, как:

- Программное обеспечение для знаменитой машины БЭСМ-6 (1968)
  - ✓первая операционная система Д-68 (1968)
  - ✓система реального времени НД-70 (1971)
- Архитектура и программное обеспечение распределенной неоднородной системы АС-6 (1979),
  - ✓ эксплуатация в Центрах управления полетами >10 лет
- Архитектура, программное обеспечение и система автоматизации проектирования ЭВМ суперкомпьютера «Электроника СС Бис»(1987)

Характерные особенности современного ПО

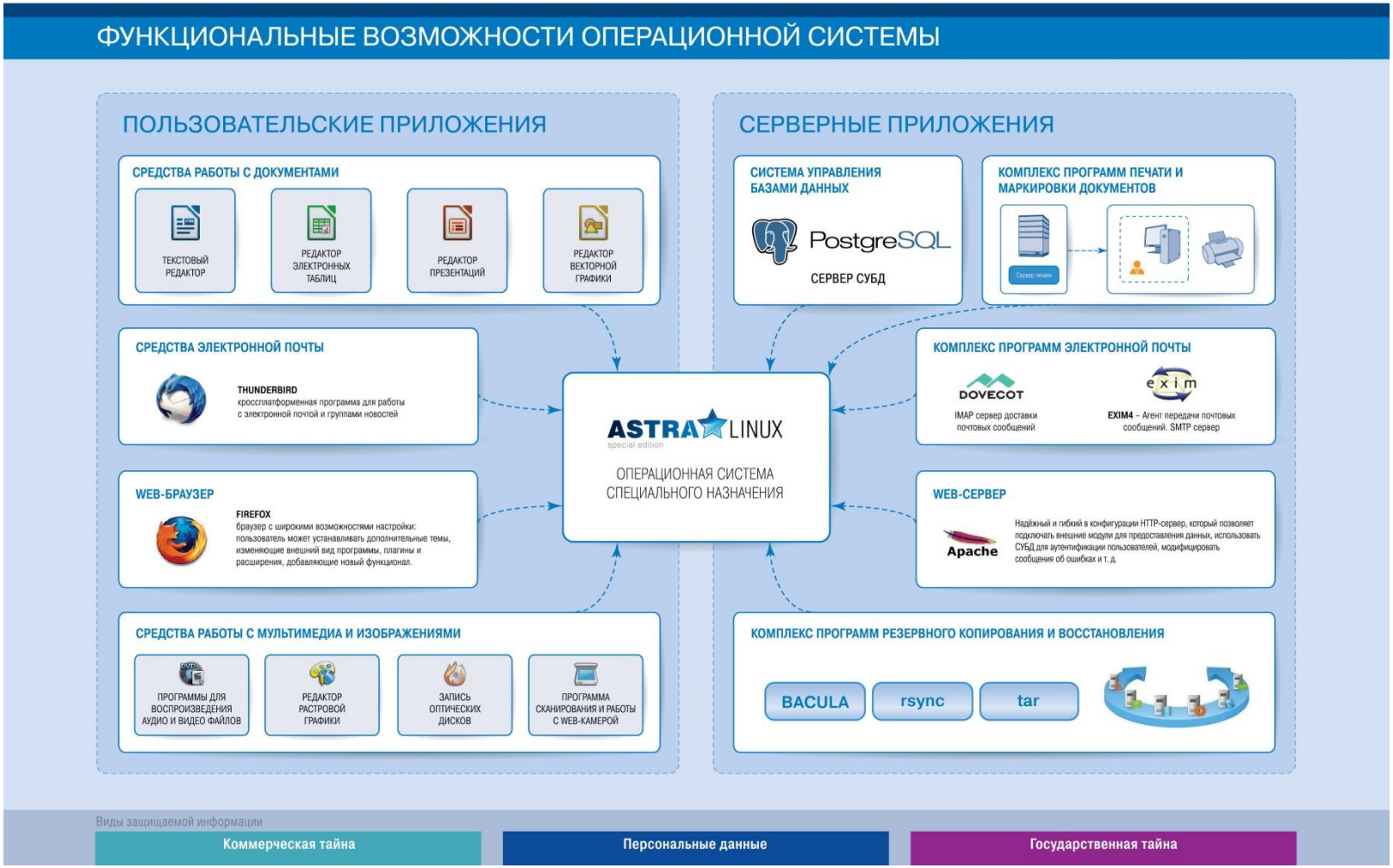
- Эскалация размеров и сложности
- Сложность среды разработки и сборки (*например, компилятор может добавить уязвимость, которой нет в исходном коде*)
- Повсеместное использование параллельных и распределенных вычислений
- Облачные вычисления, массовое внедрение мобильных платформ, «Интернет вещей» (отсутствие локально изолированных систем)
- Массовое внедрение ИТ во все сферы жизнедеятельности – переход к так называемой «цифровой экономике»

термин «кризис программного обеспечения» веден Фридрихом Л. Бауэром (Friedrich L. Bauer) на Конференции НАТО «Инженерия программного обеспечения» в 1968

ИСП РАН

ASTRA LINUX special edition

Научно-производственное объединение РусБИТех Открытое акционерное общество



1. Количество исходных .deb пакетов - 2228
2. Количество бинарных .deb пакетов после компиляции - 6894
3. Суммарное количество строк кода во всем дистрибутиве ~150 млн. (в ядре ~17 млн.)

Вызов – обеспечение качества ПО на долгосрочной основе

ИСП РАН

- Качество – конкурентоспособность современных операционных платформ – определяется тремя взаимосвязанными компонентами:
  - эффективность – производительность, масштабируемость, переносимость
    (гетерогенность, параллелизм, динамические языки, map-reduce, …)
  - продуктивность
    (средства разработки, API, удобное использование непрофисиональными программистами и др.)
  - ***безопасность***
    (принципиальное наличие ошибок в ПО, границы между ошибками программиста, закладками, НДВ размыты)

Эффективность и продуктивность

- Работы по оптимизациям в рамках: одного компилятора, набора инструментов, среды разработки, новых языков
- Развитие средств разработки:
  - ✓ **iOS**: Objective-C/LLVM – Swift/LLVM (2014)
  - ✓ **Android**: Java/Dalvik (2010), Java/Android Runtime (ART) (2014), Jack (2016)
  - ✓ **Tizen**: C++, JavaScript/WebKit & V8 (2012), C# / Roslyn (2016)
  - ✓ **Десктопные ОС**: Оптимизации link time /GCC, LLVM, Just-in-Time оптимизации / LLVM, WebKit, ...
- Крупные компании независимо вкладываются в продуктивность разработки

Эффективность – языки C/C++

Все работы для промышленных компиляторов (GCC, LLVM), которые в настоящий момент:

- ✓ поддерживают генерацию кода для различных платформ (x86, ARM, …)
- ✓ содержат миллионы строк кода
- ✓ включают сотни различных оптимизаций (эвристик)

➢ Разработка методов оптимизации

- ✓ ускорение в среднем 1-5% для набора программ (SPEC CPU, тесты заказчика), до 10% для отдельных программ
- ✓ возможность оптимизации сверхбольших программ
- ✓ более 100 патчей включены в основную ветвь разработки GCC и LLVM

➢ Автоматическая настройка компилятора (TACT)

- ✓ на основе генетических алгоритмов
- ✓ ускорение в среднем 10% (6 тестов SPEC), до 30% на отдельных приложениях

Эффективность – динамические языки

- Необходимость оптимизаций во время выполнения (динамические типы и объекты)
- Все работы для промышленных компиляторов SFX, V8
- Разработка методов оптимизации для SFX JavaScript
  - ✓ Ускорение на стандартных тестовых пакетах (SunSpider, V8, Kraken, BrowserMark) 4-10% (до 30% на отдельных тестах)
  - ✓ Более 10 патчей включены в основную ветвь SFX
- Оффлайн-оптимизация до выполнения (AOTC) для SFX, V8

Ускорение запуска JavaScript-приложений и затруднение анализа их исходного кода. Обеспечено:

- ✓ Ускорение 2-4% (до 15%) на тех же тестовых пакетах
- ✓ Сокрытие исходных текстов JavaScript

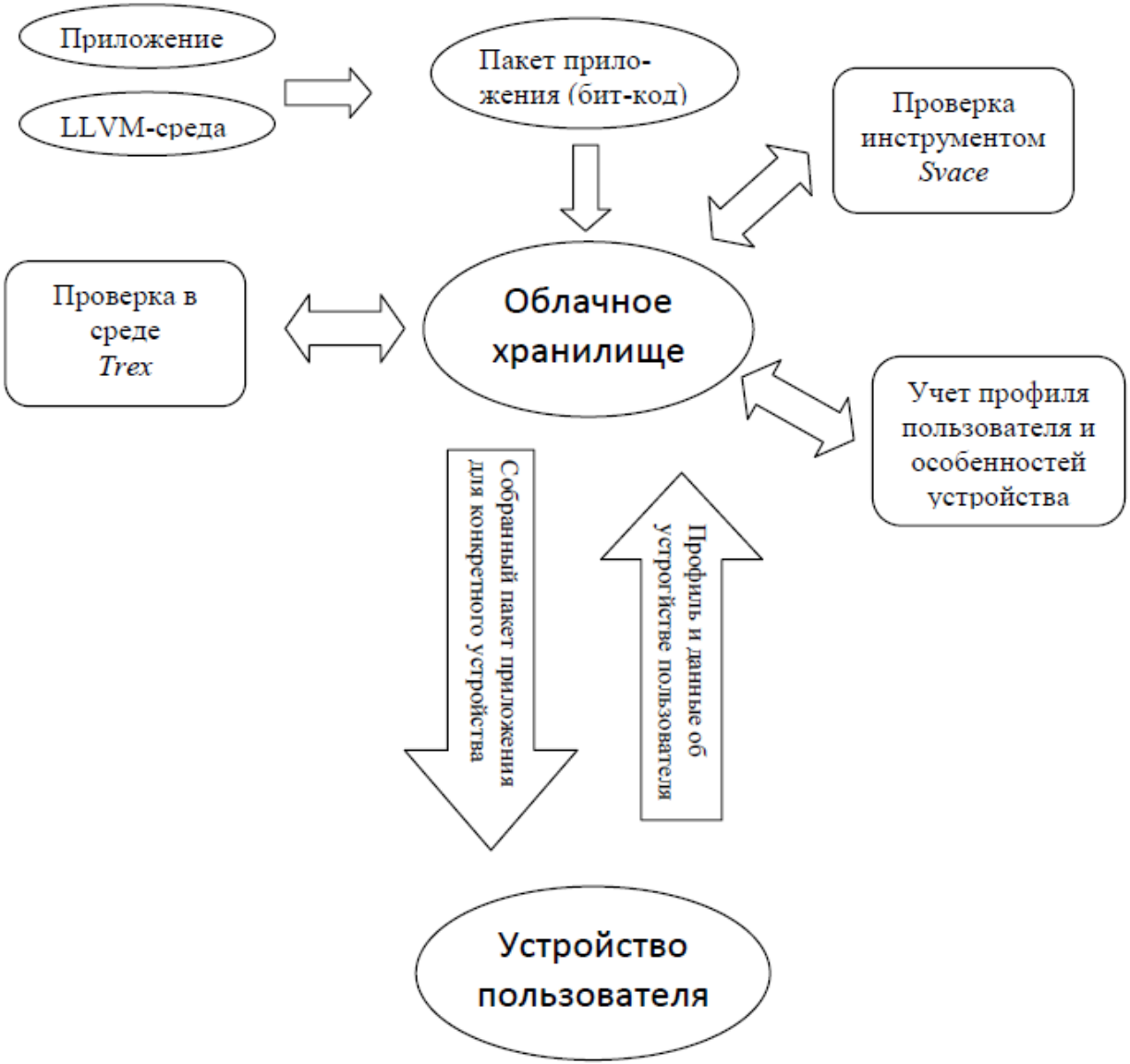
Продуктивность: переносимость C/C++-приложений с сохранением эффективности

ИСП РАН

- 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, …)

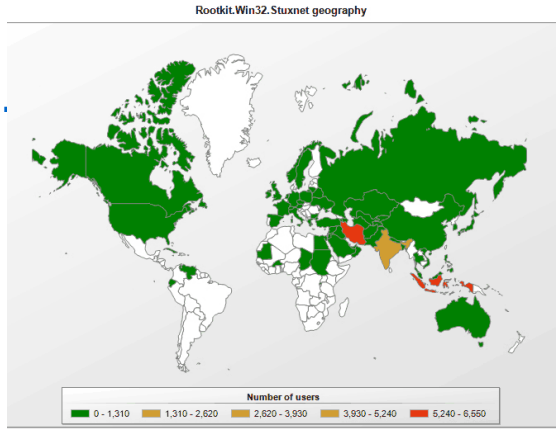
- В настоящее время аналогичные работы ведутся в Apple

«Облачное хранилище» приложений нового поколения

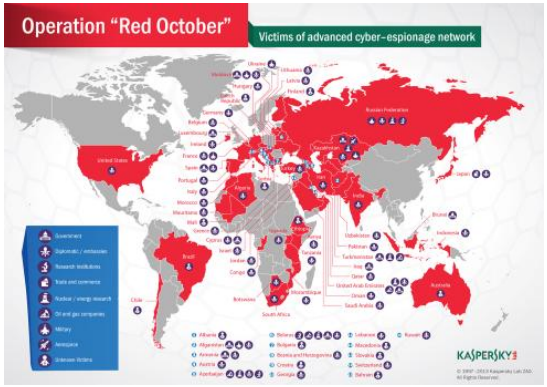


Безопасность

- «Эпидемии» вредоносного ПО
  - MyDoom, Conficker, Stuxnet, Duqu, Flame, …
  - Red October действовал более 5 лет, взламывая защищенные компьютеры
  - Возможности криминального рынка и спецслужб сопоставимы



- Классические методы ИБ недостаточны
  - Защита по периметру
  - Организационные мероприятия



- Причина - наличие **дефектов*** **в ПО**
  - Дефекты работы с динамической памятью
  - Организация многопоточной работы
  - Критические уязвимости …

***Границы между ошибками программиста, закладками, НДВ размыты**

Дефекты (I)

- 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)

Дефекты (II)

- Кластер из 100 узлов, тестирование заняло 28 дней машинного времени* (проанализировано 37 тыс. программ из состава Debian Linux)

*от авторов инструмента ***Mayhem***, David Brumley, Thanassis Avgerinos

ИСП РАН

Дефекты (III)

- Долгое время остаются необнаруженными

Уязвимость	ПО	Время существования, лет
CVE-2014-0196	Linux kernel	4.5
CVE-2014-3153	Linux kernel	4.5
CVE-2014-0160 (Heartbleed)	OpenSSL	2

- Современные дистрибутивы содержат приложения, уязвимости в которых не закрыты до сих пор. Например, Debian Linux:
  - ✓ ~ 140 программ содержали не исправленные CVE с 2010 по 2016 год
- С помощью инструментов анализа ИСП РАН были обнаружены:
  - ✓ 42 уязвимых приложения в дистрибутиве Arch Linux
  - ✓ Около 100 уникальных ошибок в дистрибутиве Astra Linux 1.5

Необходимы* методы анализа программ (*исходного и бинарного кода*) и соответствующие инструменты, позволяющие:

- найти и устранить на этапах **разработки** и внедрения максимальное количество ошибок в исполняемом коде
- предотвратить эксплуатацию существующих ошибок или смягчить последствия их эксплуатации

***Уровнем используемых инструментов анализа определяется уровень информационной безопасности эксплуатирующихся программно-аппаратных систем**

Анализ* ПО с целью выявления дефектов

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

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

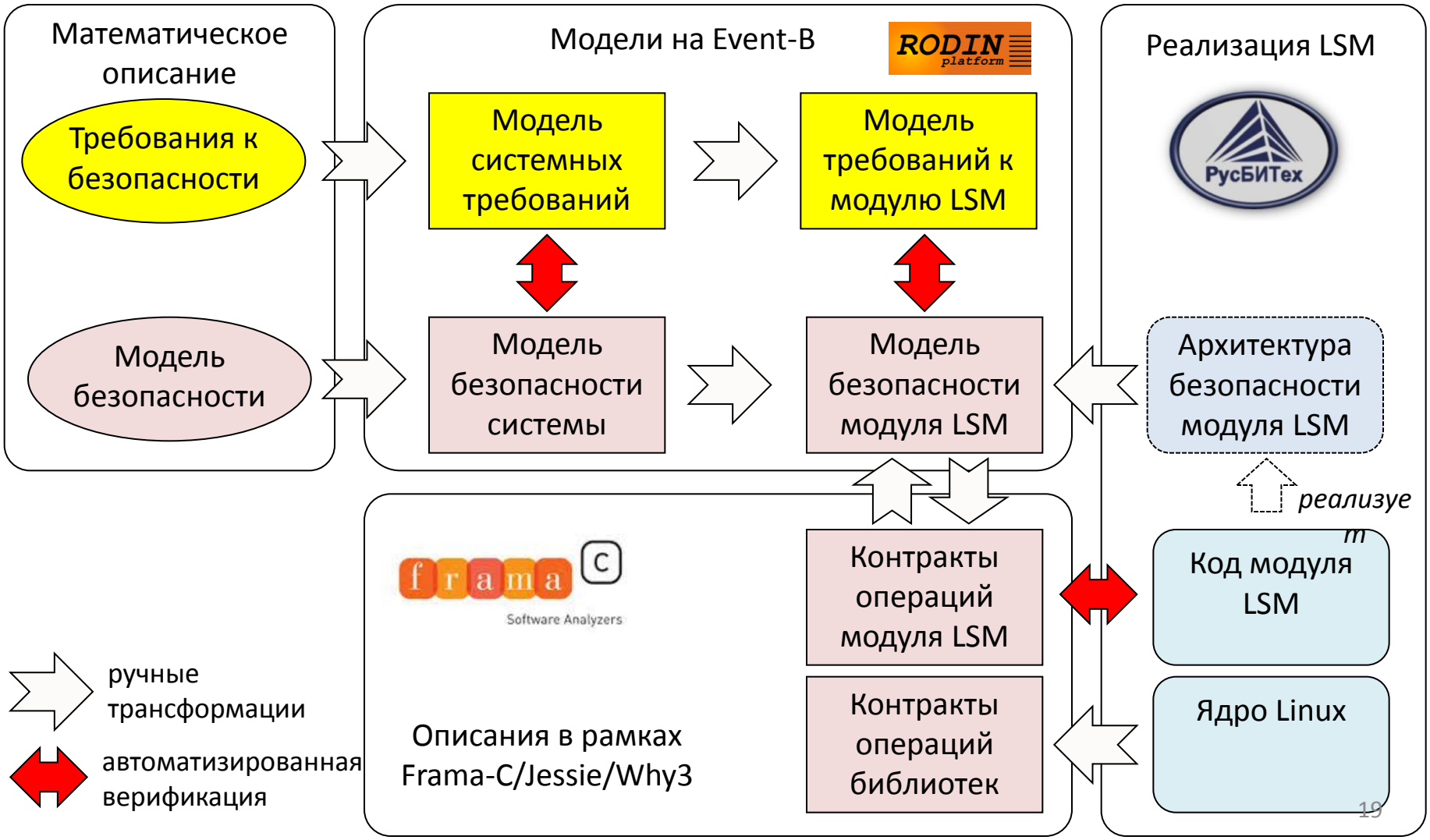
Дедуктивный анализ (I)

- Возникновение метода 1969
  - R. Floyd, C.A.R. Hoare
- Появление инструментов ~1993-2005
  - SunRise, ESC/Java, Frama-C, LOOP, Boogie/VCC, Rodin
- Промышленные применения ~2003-2014
  - Системы управления АЭС (Франция, UK)
  - Автономные транспортные системы (Франция)
  - Авионика (UK, Airbus, NASA)
  - Ядра операционных систем и гипервизоров seL4, PikeOS, Hyper-V
  - Компоненты микропроцессоров (проектная модель) Intel, Verisoft

seL4: ~ 10000 строк кода
~200000 строк модели
~ 12 чел 4 года

Дедуктивный анализ (II)

Верификация модели безопасности AstraLinux
РусБИТех – ИКСИ – ИСП РАН



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 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 Prevent, Klocwork Insight, **Svace (ИСП РАН)**

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*) – IDA Pro
- Динамический или комбинированный анализ

Актуальные направления исследований

ИСП РАН

- Восстановление алгоритмов и протоколов из бинарного кода
  - *Компиляторное промежуточное представление*
  - *Последовательное повышение уровня представления*
  - **Bitblaze, BAP, GrammaTech CodeSurfer/x86, Tral**
- Автоматический поиск ошибок в бинарном коде
  - *Смешанное конкретное/символьное выполнение*
  - *Анализ помеченных данных*
  - *SMT-решатели*
  - **Bitblaze, Mayhem, S2E, McVeto, Avalanche-Anxiety**
- Развитие возможностей виртуальных машин
  - *Аппаратная виртуализация*
  - *Детерминированное воспроизведение, обратная отладка*
  - **Доработанные версии QEMU**

Tral: среда комбинированного анализа бинарного кода (I)

ИСП РАН

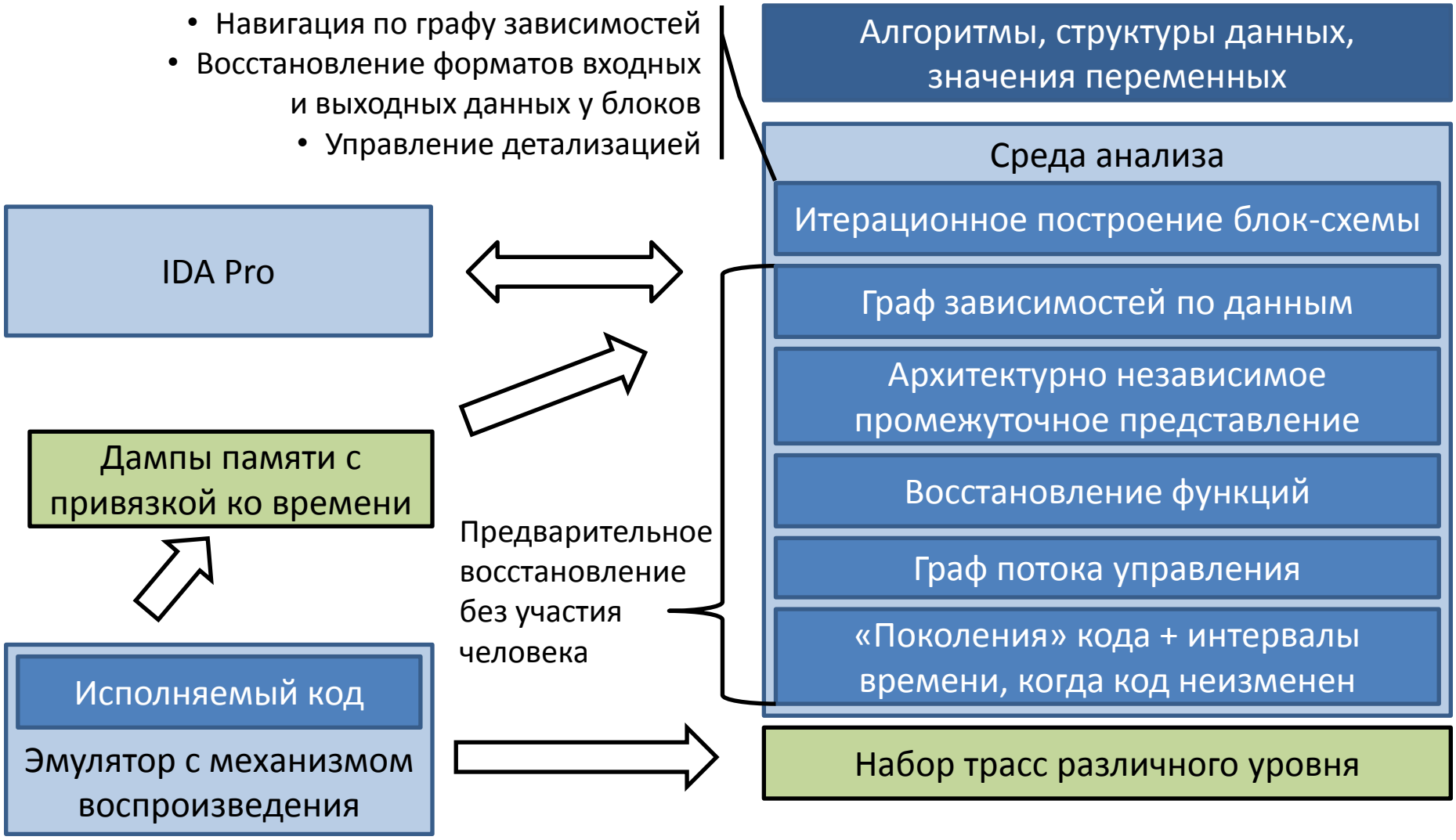
- Базируется на **модели** анализируемого бинарного кода, получаемой на основе трасс и состоящей из модели кода в платформо-независимом представлении и модели областей памяти (регистров, стека, областей статических и глобальных данных) – регистровая RISC-машина
- Обеспечивает:
  - ✓ сбор и систематический 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.

Tral: 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



Tral: 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)

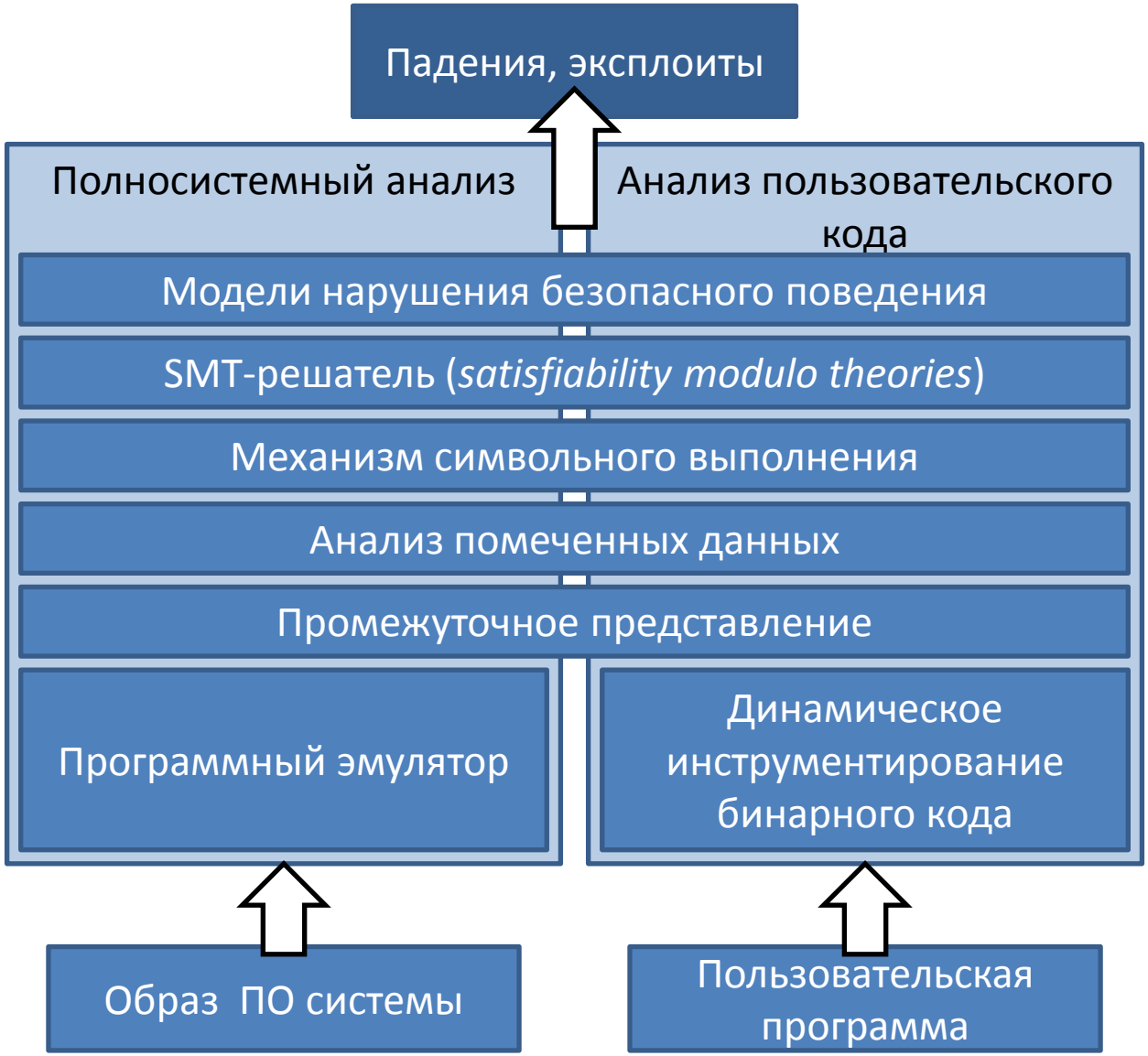
Пример выделения инструкций требуемого алгоритма:

	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

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

Этап эксплуатации ПО

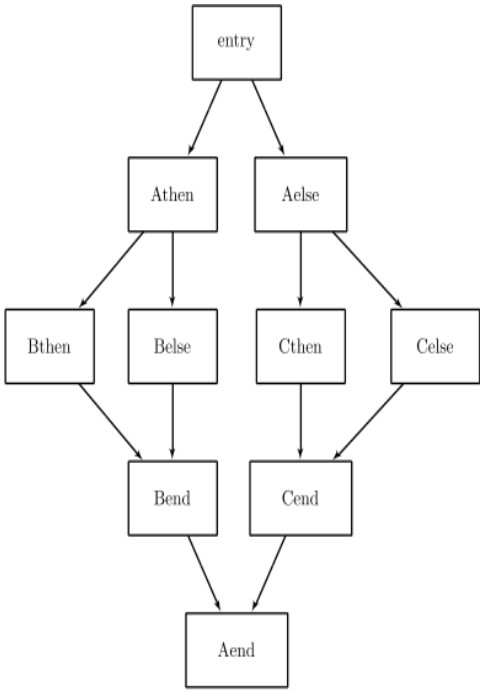
- Песочницы
- HoneyPot
- Журналирование действий пользователя
- Обфускация (запутывание)
- Гомоморфное шифрование
- …

Обфускация

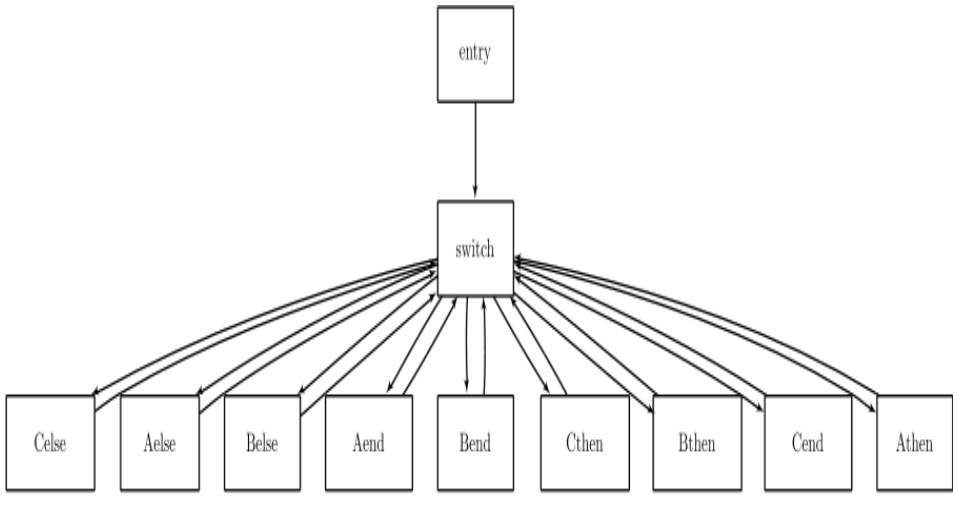
- ✓ усложнение понимания алгоритмов и структур данных
- ✓ затруднение генерации эксплойтов на основе анализа патчей
- ✓ расстановка водяных знаков
- ✓ предотвращение эксплуатации известной уязвимости

ИСП Обфускатор – усложнение понимания алгоритмов и структур данных

- Реализован в LLVM
- Замедление для одного запутывающего преобразования составляет 1,2-5,5 раз
- Размер программы увеличивается в 1,05-2,5 раз



CFG for 'main' function



flattened CFG for 'main' function

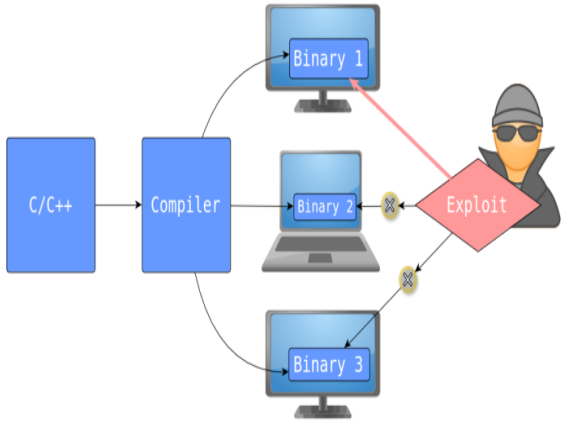
ИСП Обфускатор – противодействие эксплуатации уязвимостей

Реализован в GCC

- Статическая – уникальные сборки

Замедление: ~1-2%

Затруднение эксплуатации. Пример - переполнение буфера: изменяются адреса и длины буферов и порядок локальных переменных.

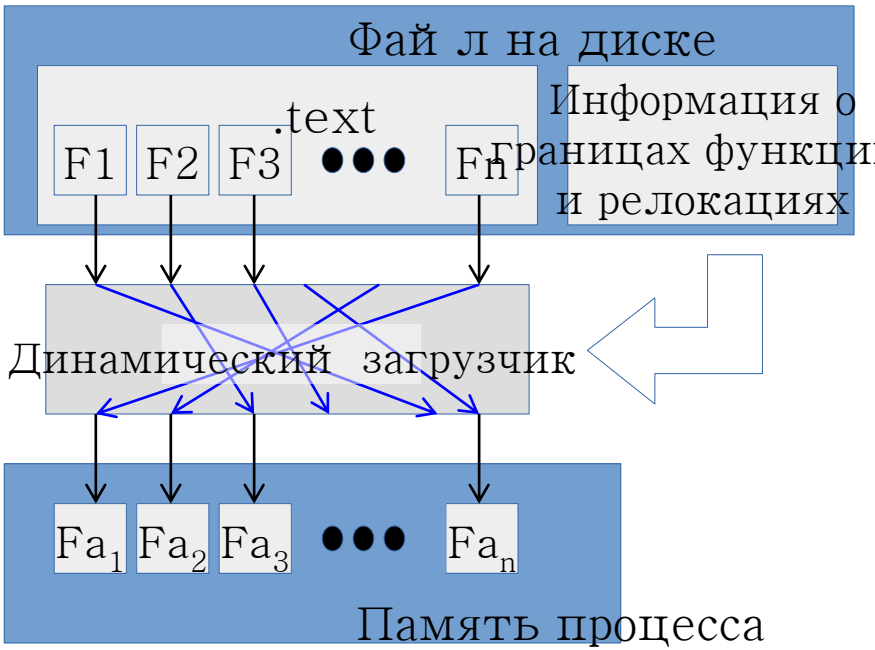


- Динамическая, во время работы

Гранулярность рандомизации до функций

Замедление: ~1%

Затруднение эксплуатации: широкий круг уязвимостей, поскольку меняются адреса участков кода, который можно использовать в эксплоитах. Препятствует ROP атакам, усиливая стандартную ASLR.

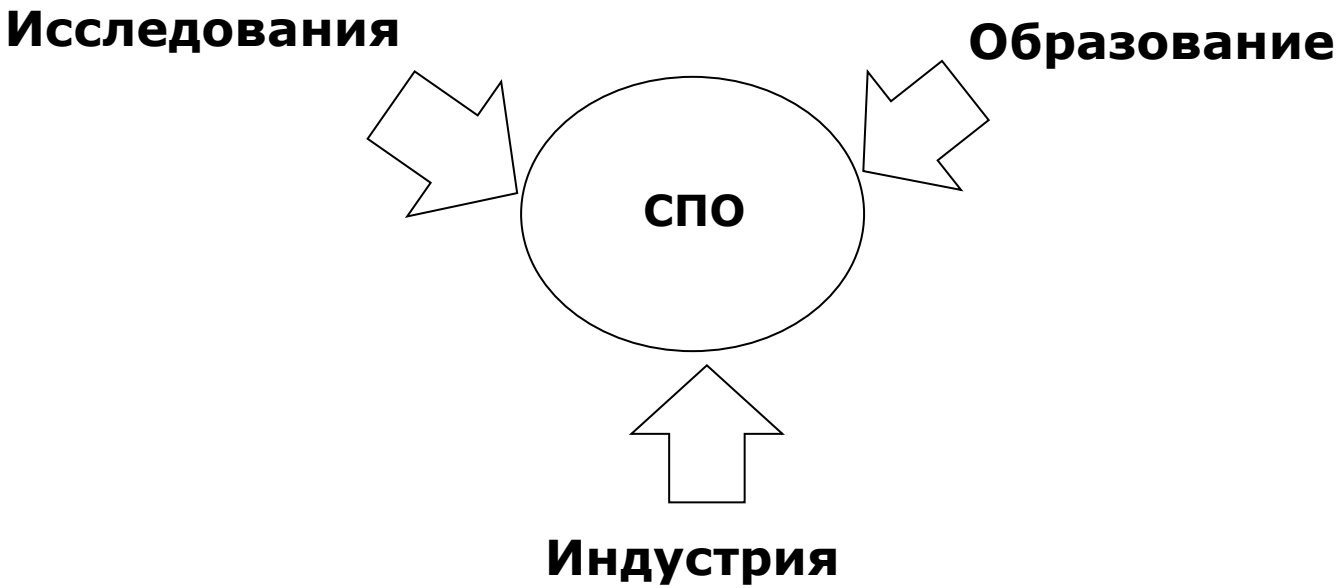


- Сложность систем, быстро меняющиеся требования и новые вызовы требуют постоянных **инноваций:**
  - разработка и внедрение новых технологий
  - адаптация и развитие существующих под специфику прикладных областей

- Никакая отдельно взятая компания или даже государство не в состоянии поддерживать необходимые темпы развития без существенных рисков **(нехватка кадров)**

*Свободное программное обеспечение (СПО) – один из способов решения **кадровой** проблемы и потенциальная возможность обеспечить **технологическую независимость** и снизить риски как на уровне компаний, так и государства*

Преимущества СПО (I) - единый «язык» для исследований, образования и индустрии



- Позволяет решать проблему разрыва между образованием и индустрией
- Обеспечивает механизмы ускоренного внедрения инноваций
- Позволяет эффективно управлять рисками и разделять ресурсы

Преимущества СПО (II) – более высокий уровень безопасности

- Критическая масса разработчиков и независимых компаний – любой дефект (уязвимость) становится достоянием общественности и фиксируется

- Следствие: число ошибок в промышленном OpenSource меньше, чем в проприетарном ПО того же класса

Замечание. Импортозамещение может освободить рынок для отечественных компаний с низкой культурой разработки и в госучреждения может пойти поток проприетарного российского ПО с большим числом ошибок

Недостатки-риски использования СПО в контексте обеспечения безопасности

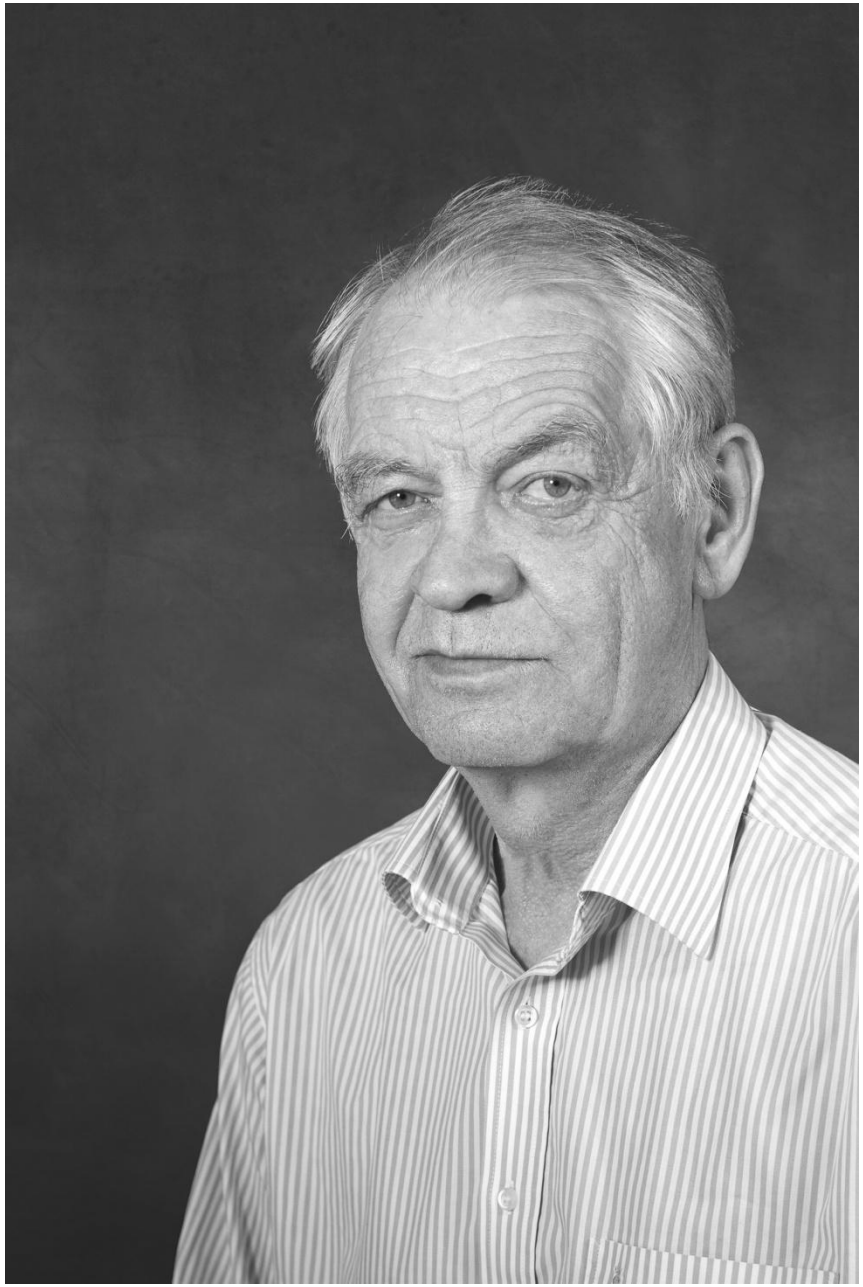
ИСП РАН

- Инструментов промышленного уровня на основе СПО по многим критическим аспектам анализа ПО практически нет
- Существует *множество* исследовательских проектов и их количество постоянно растет
- Высок риск использования незрелых технологий, которые невозможно поддерживать и развивать
- Необходимы кадры высшей квалификации – не только мониторинг, выбор, адаптация и внедрение, но и долгосрочное развитие с учетом перспективных трендов

Необходимы институциональные механизмы, обеспечивающие эффективное участие РФ в международной кооперации в модели СПО, с целью долгосрочного трансфера знаний и технологий в РФ, удержания кадров, а также создания благоприятных условий экспорта инноваций

- *Необходима комплексная программа исследований и разработок, нацеленная на создание технологий и кадрового потенциала мирового уровня по всем основным аспектам анализа ПО в области обеспечения безопасности*

- *Необходимы институциональные механизмы, обеспечивающие эффективное участие РФ в международной кооперации в модели СПО, с целью долгосрочного трансфера знаний и технологий в РФ, удержания кадров, а также создания благоприятных условий экспорта инноваций*



“ISPRAS Open” 2017

Посвящается памяти
Виктора Петровича Иванникова

-Ноябрь 30 – Дек 1, 2017

ИСП РАН

Спасибо!

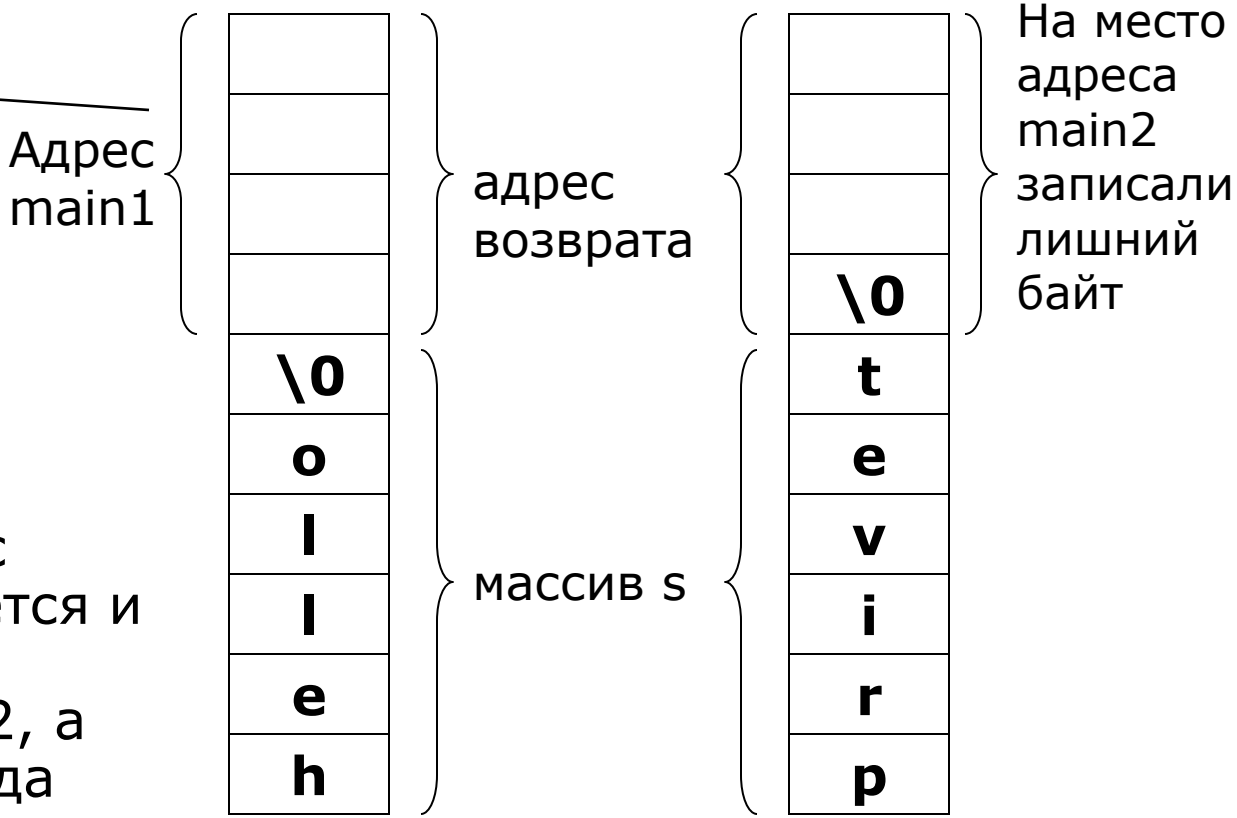
Экосистема* ИСП РАН

Важный компонент созданной экосистемы – широкое использование свободного программного обеспечения (СПО), рассматриваемый как:

- ***Инструмент***, предоставляющий легитимный свободный доступ ко всему многообразию современных технологий, включая готовые к использованию программные продукты, технологии и открытые стандарты
- *Мощный* ***образовательный ресурс*** – среда и инфраструктура международных СПО-проектов *могут и должны* использоваться для подготовки специалистов высокой квалификации
- Возможность взаимодействия с глобальным рынком продуктов и услуг, что позволяет перейти с аутсорсинга на ***инновационное развитие***

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)

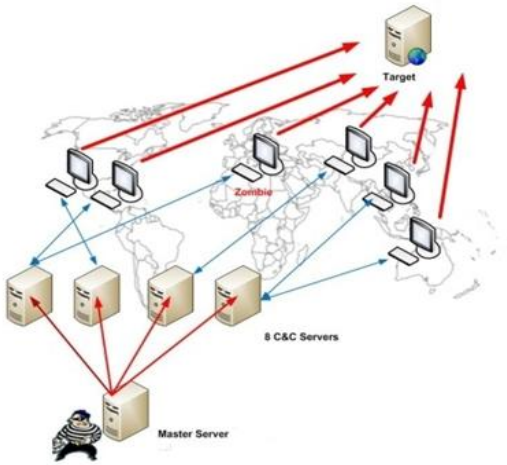
```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    char buf[80], cmd[100];
    fgets(buf, sizeof(buf), stdin);
    snprintf(cmd,sizeof(cmd),"ls -l %s", buf);
    system(cmd);
    return 0;
}
```

Например: `ls -l myfile ; rm -rf /`

Ошибки – существенный рост ущерба

Аварии на критических обьектах

Перехват управления



Бот-неты

Уязвимости

Кража паролей

Кража информации о кредитных картах