ИСП РАН

Унифицированная среда безопасной разработки программного обеспечения

ПАДАРЯН В.А.
vartan@ispras.ru

ТБ Форум 2024
Актуальные вопросы защиты информации
14 февраля

Система центров компетенций кибербезопасности

Сложности внедрения ГОСТ Р 56939

- Реорганизация процессов разработки
  - Новые активности, надо больше сотрудников
  - Обучение сотрудников
  - Методики анализа ПО не вообще, а в частности – для требуемых уровней доверия и модели угроз
- Инструменты анализа кода
  - Что выбрать, чтоб не заплатить дважды?
    Соответствие техническим требованиям, а не советам маркетологов
  - Проприетарные инструменты vs. Open source, что на самом деле дешевле?
  - Отечественная аппаратура, где взять инструменты?
  - Интеграция инструментов и результатов анализа ПО
    ASOC? Решение для кого - руководства? ИБ? разработчиков?
- Подготовка к сертификации
  - Анализ всего кода ПО
  - Сбор материалов для испытательной лаборатории

Унифицированная среда разработки безопасного ПО

Регулятор

Квалификационные тесты для оценки ключевых характеристик инструментов

Разработчики СЗИ

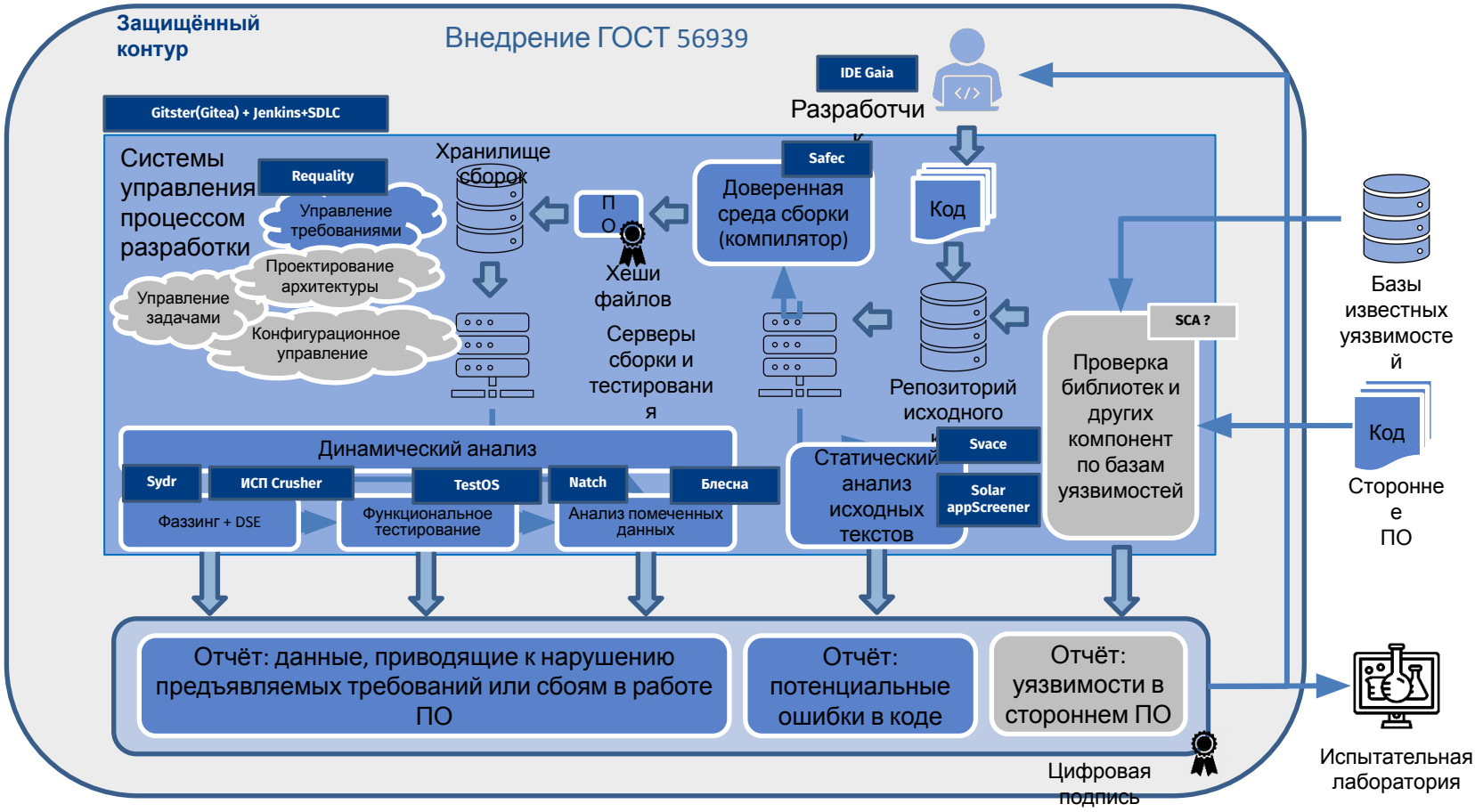
Методические рекомендации по внедрению

Технологии / инструменты анализа кода ПО

IT-сервисы, CI/CD: Jenkins, Gitea, IDE Eclipse Theia

Инфраструктура, аттестованная АС: серверы и СЗИ

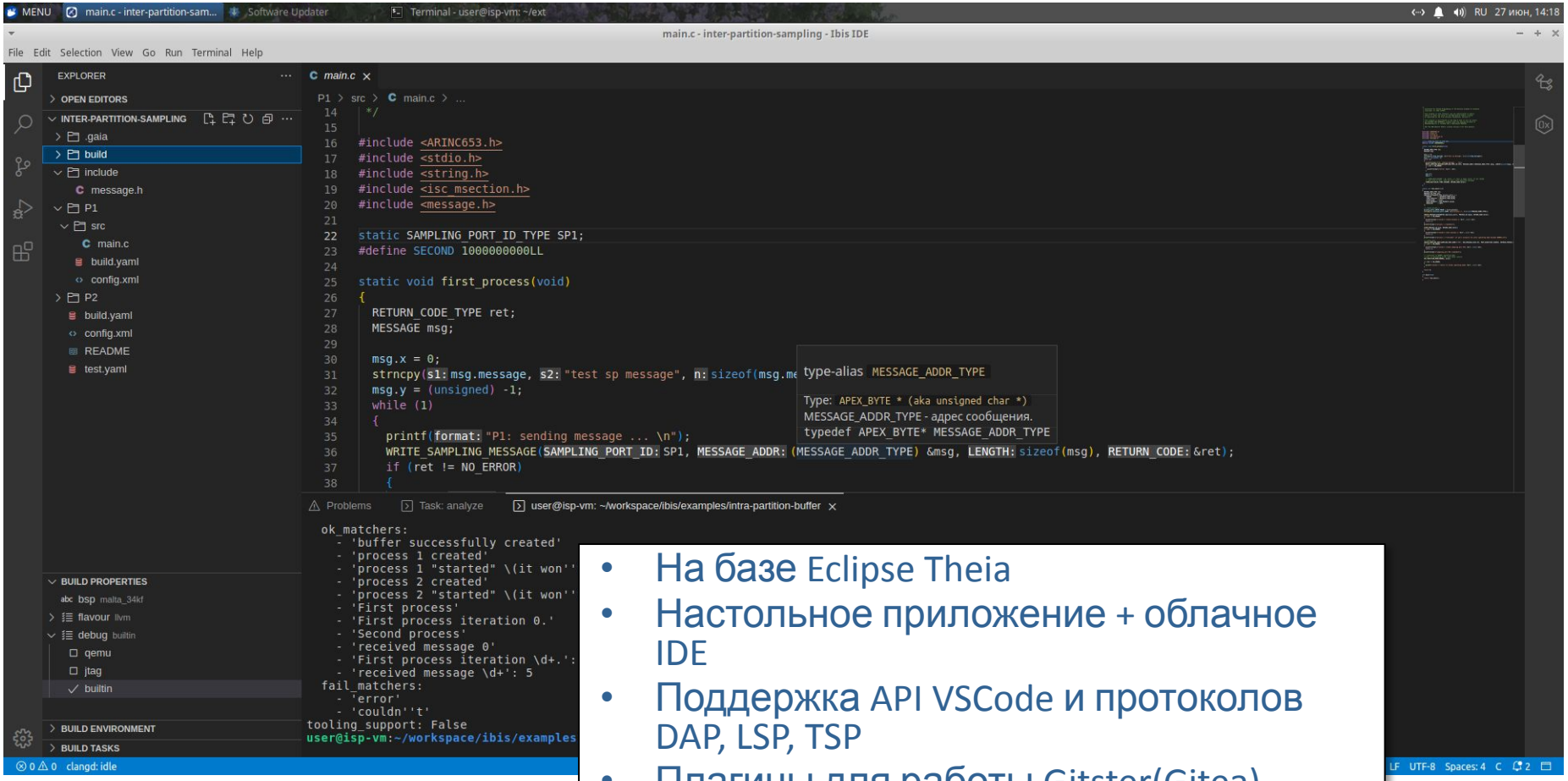
Унифицированная среда разработки безопасного ПО: технологии



Унифицированная среда разработки безопасного ПО: состав работ

- Доступный SDL для отечественных разработчиков
  - Отвечающий действующим и перспективным требованиям Регулятора
- Состав технологий разработки и анализа кода
  - IDE и CD/CD
  - Безопасный компилятор C/C++
  - Инструменты кросс-платформенной разработки (эмуляторы и отладчики)
  - Инструменты разработки и выполнения модульных тестов
  - Инструменты статического анализа (C/C++, Java, Python, …)
  - Инструменты фаззинг-тестирования и автоматизированного анализа поверхности атаки
  - Инструменты выявления утечек информации по памяти на основе полносистемного динамического анализа помеченных данных
- Поддерживаемые платформы: ОС Linux x86-64 и ARM
- ***Исследование и портирование технологий анализа кода на отечественные процессоры***
  - Байкал, Эльбрус, RISC-V
- ***Наборы квалификационных тестов для инструментов статического и динамического анализа***
  - Реализация требований национальных стандартов по оценке технологий анализа кода

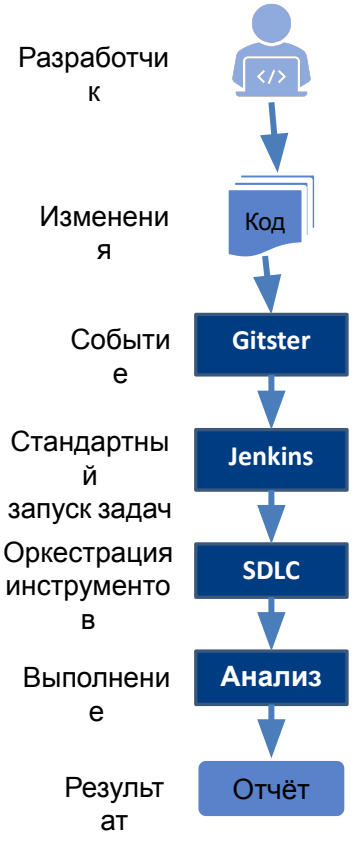
IDE Gaia



- На базе Eclipse Theia
- Настольное приложение + облачное IDE
- Поддержка API VSCode и протоколов DAP, LSP, TSP
- Плагины для работы Gitster(Gitea), SARIF, специфичные для инструментов анализа ПО (в разработке)

Непрерывная интеграция Gitster(Gitea) + Jenkins + SDLC

- SDLC - оркестратор инструментов анализа
  - Использует описания сценариев анализа
  - Сценарий может храниться в Git-е, локальной директории, S3 хранилище
  - Формат сценария - toml
- Запуск множества сценариев одной командой
- Параметризация сценариев
- Хранение всех параметров запуска, промежуточных результатов, возможность импорта и экспорта, повторных запусков
- Управление вычислительными ресурсами, распределение задач анализа по доступным вычислительным узлам, очереди задач
- Обработка результатов работы инструментов, загрузка в Svacer, разбор результатов фаззинга, сбор покрытия и его показ
- CLI и Web UI, gRPC и GraphQL API
- Возможность ссылаться на артефакты выполненных сценариев
  (т.е повторное использование результатов анализа)



ИСП РАН

Система национальных стандартов разработки безопасного ПО

Рекомендации по внедрению стандартов разработки для различных видов ПО
- Встраиваемое ПО, IoT
- **Доверенная загрузка** (BIOS)
- **Операционные системы**
- Гипервизоры
- Контейнеры
- **Антивирусы**
- **Межсетевые экраны, СОВ**
- Среды функционирования интерпретируемых языков
- СУБД
- Офисные приложения
- Мобильные приложения
- Аудио и видеосвязь, мессенджеры
- Web-приложения

...

ГОСТ Р 56939-2016 Разработка безопасного ПО. Общие требования

Разработка безопасного ПО. Термины и определения

ГОСТ Р 58412-2019 Разработка безопасного ПО. Угрозы безопасности информации при разработке программного обеспечения

Разработка безопасного ПО. Руководство по разработке безопасного программного обеспечения.

Группа технологических стандартов

TODO: Тестирование на проникно-вение

Композиционный анализ. Общие требования

РГ7

Сборочная среда

ГОСТ Р 71206-2024 Безопасный компилятор С/С++. Общие требования

TODO: CI/CD, безопасность сборочной среды

ГОСТ Р 71207-2024 Статический анализ исходных текстов. Общие требования

ГОСТ Р 59453.1-2021 Формальная модель управления доступом

ГОСТ Р 59453.2-2021 Верификация формальной модели управления доступом

Динамический анализ

Функциональное тестирование

Фаззинг-тестирование

Выявление утечек чувствительной информации

TODO: Моделирова-ние угроз

TODO: Экспертиза безопасности исходного и бинарного кода

TODO: Документирование

TODO: Реагирование на инциденты, выпуск обновлений

Разработка безопасного ПО. Руководство по оценке безопасности разработки программного обеспечения

Методология разработки доверенных систем Конструктивная информационная безопасность

Общие положения

Методология разработки

Шаблоны проектирования

Защита информации. Доверенная среда исполнения. Общие требования

РГ9

Действующий или принятый стандарт

Разрабатываемый/пересматриваемый стандарт

Разработка приостановлена

РГx Рабочая группа №x

Статический анализ программного обеспечения. Общие требования (1/2)

- Термины, определения и сокращения
  Разъяснение понятий из областей:
  - Языки программирования
  - Компиляторные технологии, анализ и оптимизация кода в частности
  - Технологий программирования
- Требования к внедрению и порядку выполнения статического анализа
  - Руководство по порядку внедрения, применимое в компаниях с небольшим штатом сотрудников
  - «Выжимка» лучших практик
- Классификация критических ошибок, находимых статическими анализаторами
  - Что обязательно надо искать и потом исправлять
  - Разделение типов для компилируемых и интерпретируемых языков
  - Система построения тестов для оценки ключевых характеристик статического анализатора: ***поточные и модельные варианты ошибки***
- Требования к методам анализа, реализованным в статическом анализаторе
  - Ключевые характеристики
    - **Доля ложноотрицательных срабатываний – не более 50%**
    - **Доля ложноположительных срабатываний – не более 50%**
    - **Скорость анализа – все ПО не более чем за 48 часов**
  - Разделение требований для компилируемых и интерпретируемых языков

Статический анализ программного обеспечения. Общие требования (2/2)

- Требования к инструментам статического анализа
  - Применение в промышленном окружении процессов разработки
  - Выдаваемые результаты: диагностика, формат, типизация ошибок
- Требования к специалистам, проводящим статический анализ
- Требования к методике проверки требований к статическому анализатору
  - Проверка требований на системе тестов. Синтетические тесты и реальные программы различного размера.
- Приложения
  - Пример применения классификации критических ошибок, выявляемых статическими анализаторами, для ошибки переполнения буфера в языке С
  - Пример построения квалификационных тестов для ошибки переполнения буфера с использованием поточных вариантов в языке С

Квалификационные тесты
Можно ли доверять Джульетте?

Pro

- Фактический стандарт для оценки стат. анализаторов
- Тестов (очень) много

Тестовый набор	Число тестов
Juliet 1.3 Java	28881
Juliet 1.3 C/C++	64099
C# Vulnerability Test Suite	32003
PHP Vulnerability Test Suite	42212

- Тесты покрывают разнообразие ситуаций
  - Поток управления, разные управляющие операторы
  - Разные подтипы ошибки
- Тесты для основных ЯП
  - C/C++, Java, C#, PHP
- Есть разметка внесенных ошибок

Contra

- Основной разработчик – NSA
- Неточности в разметке внесенных ошибок
- Некоторые ошибки машинально переносились между разными языками
  - Java → C#
- Тесты синтетические, небольшого размера – невозможно оценить масштабирование алгоритмов анализа

Тесты для статического анализа

- Поддерживаемые языки
  - C/C++, Java, C#, Python
- Разметка внесенных ошибок – отдельный json-файл
- Оцениваемый результат анализа в формате SARIF
- Состав тестов
  - Juliet 1.3

Java

# модулей	1	2	3	4	5
# тестов	17953	8009	729	929	1261

C

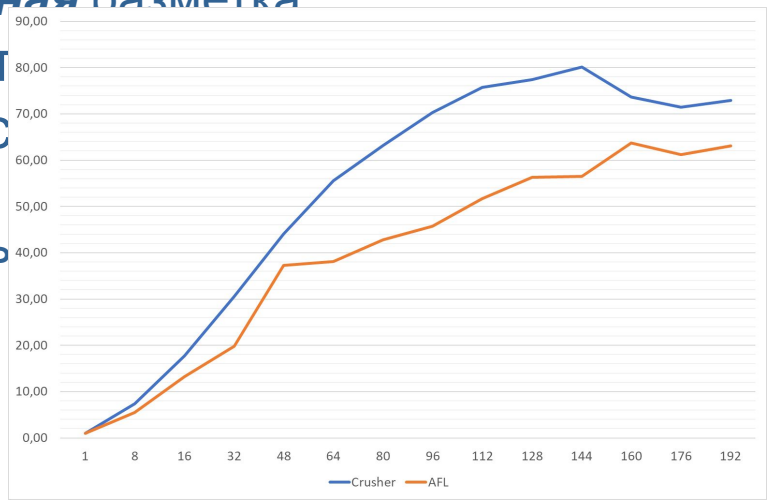
# модулей	1	2	3	4	5
# тестов	40626	15474	1239	3977	2807

  - Реальные программы ми
    - PDF-парсер, мессенджер, веб-браузер, драйвер устройства
    - Типы ошибок: переполнение буфера, форматная строка, ошибки работы с динамической памятью, утечки данных
      CWE-121, CWE-122, CWE-134, CWE-415, CWE-416, CWE-200
- Регламент пополнения

Идентификатор CWE	Общее количество тестов (N)	Svace			Cppcheck		
		Количество найденных истинных ошибок (T)	Доля найденных истинных ошибок TP(%)	Доля пропущенных истинных ошибок FN(%)	Количество найденных истинных ошибок (T)	Доля найденных истинных ошибок TP(%)	Доля пропущенных истинных ошибок FN(%)
CWE121	3100	2218	71.55	28.45	19	0.61	99.39
CWE122	3870	2747	70.98	29.02	511	13.20	86.80
CWE124	1168	481	41.18	58.82	30	2.57	97.43
CWE126	870	310	35.63	64.37	19	2.18	97.82
CWE127	1168	478	40.92	59.08	30	2.57	97.43
CWE476	54	54	100.00	0.00	54	100.00	0.00

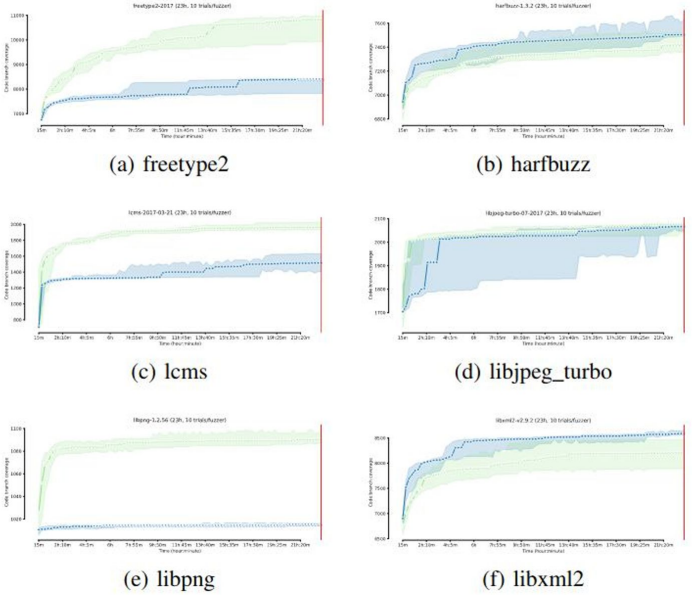
Фаззинг – что можно оценивать?

- Доля уникальных аварийных завершений *U = C_N / T_N*
  - *C_N* – число групп аварийных завершений, которые получены в результате группировки аварийных завершений, обусловленных одной и той же ошибкой
  - *T_N* – общее число зафиксированных аварийных завершений тестируемой программы
  - Сравнение результатов группировки по F1-мере классификационной (основанной на подсчете пар) метрики на одинаковом для разных фаззеров наборе аварийных завершений, для которого была проведена ***ручная*** разметка
- Длительность разогрева фаззинг
- Поддерживаемый размер корпус образцов
  - Влияние характера входных данны
- Масштабируемость
  - Влияние конфигурации сервера



Фаззинг – что нужно оценивать?

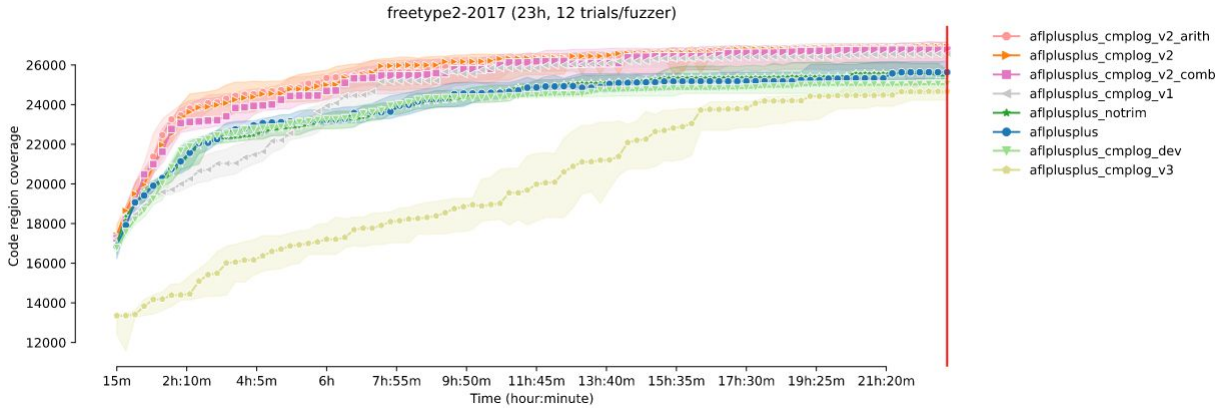
- FuzzBench by Google – сравнительная оценка фаззеров
- Критерии – рост покрытия, число ошибок
- Фаззинг в один поток в докере
- Многократное повторение эксперимента для уточнения статистики
- Регулярно фиксируется достигнутое множество входных данных, по нему отдельно подсчитывается покрытие



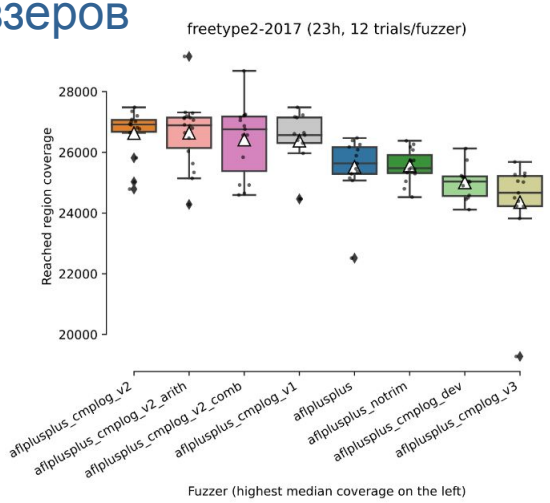
Sydr-Fuzz vs. AFL++

Сравнение фаззеров: FuzzBench от Google

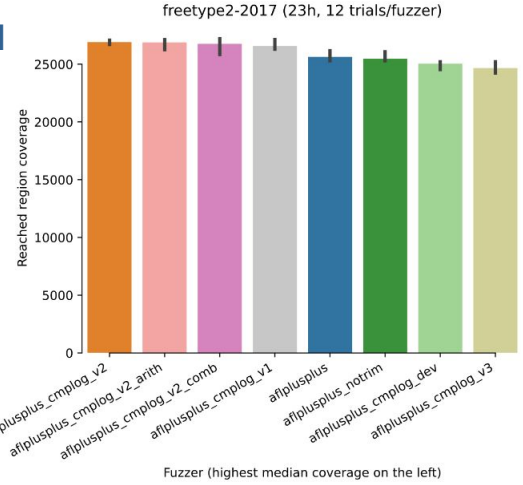
- График роста покрытия фаззеров



- Диаграмма распределения финального покрытия фаззеров



- Диаграмма для ранжирования фаззеров по размеру достигнутого покрытия



Портирование фаззеров на отечественные процессорные архитектуры

		Байкал М (AArch64)	RISC-V 64	Эльбрус
Фаззеры	libFuzzer	✔	✔	Работает без обратной связи
	AFL++	✔	✔	Работает без обратной связи
	Atheris (Python)	✔	✔	✔
	Sydr/Crusher	□	□	□
Анализ крешей	CASR	✔	✔	□
Базовое СПО	Lcov	✔	✔	✔
	DynamoRIO	✔	□	□
	QEMU	✔	✔	□
	TRITON	✔	□	□
	Gdb	✔	✔	✔
	Capstone	✔	✔	□
ЯП/компиляторы	Gcc	✔	✔	Lcc, ASAN, MSAN, совместим с Gcc 7.3.0
	llvm	✔	✔	Экспериментальная версия, 9.0.1
	Python3	✔	✔	✔
	Rust	✔	✔	□

Направления развития Унифицированной Среды

- Применение ИИ в технологиях и инструментах жизненного цикла безопасного ПО
  - Фильтрация ложных срабатываний статического анализа средствами ИИ
  - Построение тестовых наборов для инструментов статического анализа средствами ИИ
  - Повышение качества анализа интерпретируемых языков программирования
  - Анализ поверхности атаки ПО средствами ИИ в целях определения приоритетных фаззинг-целей
- Расширение номенклатуры инструментов
- Развитие наборов квалификационных тестов
  - В первую очередь – для интерпретируемых языков программирования