AURA: программная платформа высокоскоростного анализа сетевого трафика для задач информационной безопасности

Денис Гамаюнов, Дмитрий Казачкин

лаборатория вычислительных комплексов ВМК МГУ имени М. В. Ломоносова

AURA?

AUtomata for **R**ecognition & **A**nalysys

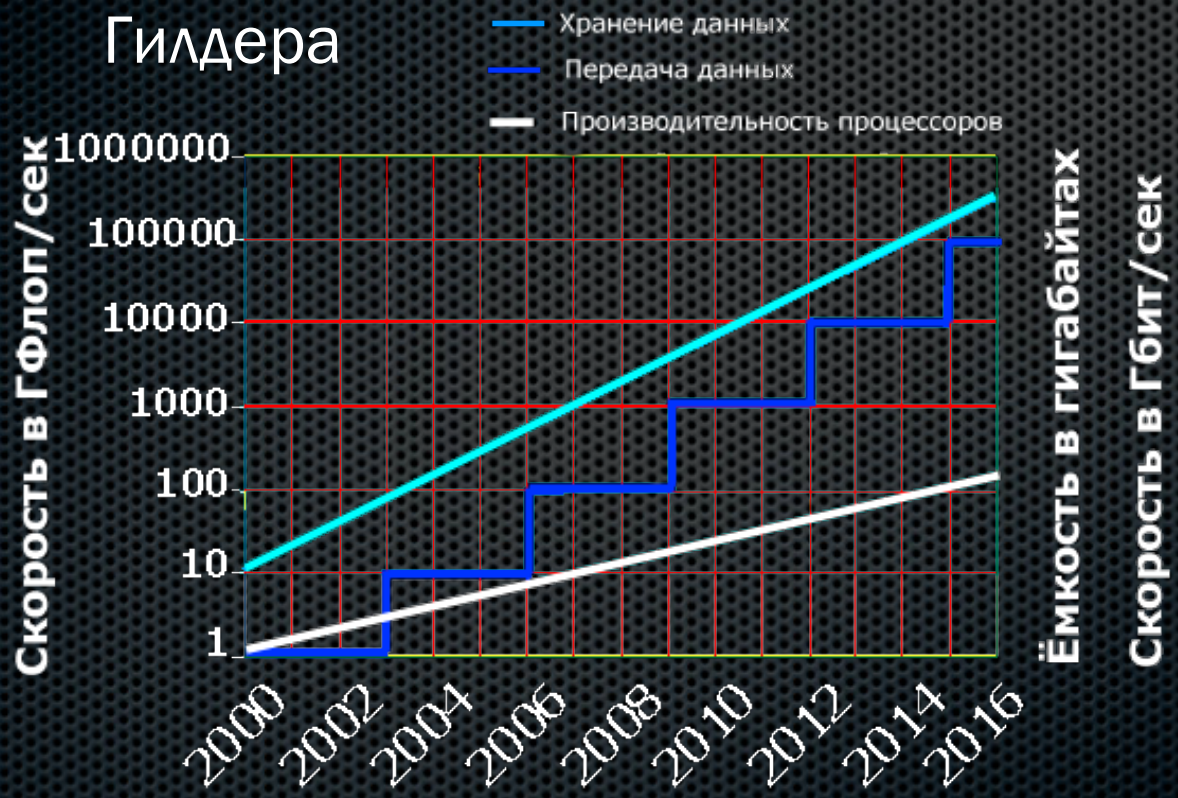
- Специализированный язык программирования на основе альтернирующих автоматов
- Байткод LLVM (Low-Level Virtual Machine)
- Параллельная система выполнения
- Библиотека алгоритмов

План доклада

- Постановка проблемы
- Типовые задачи анализа трафика
- Описание языка
- Примеры
- Результаты экспериментов

Анализ трафика – задача реального времени

- Закон Мура vs закон Гилдера



1GigE:

- минимальный пакет 64 байта
- ~700нс на обработку (худший случай)

10GigE?

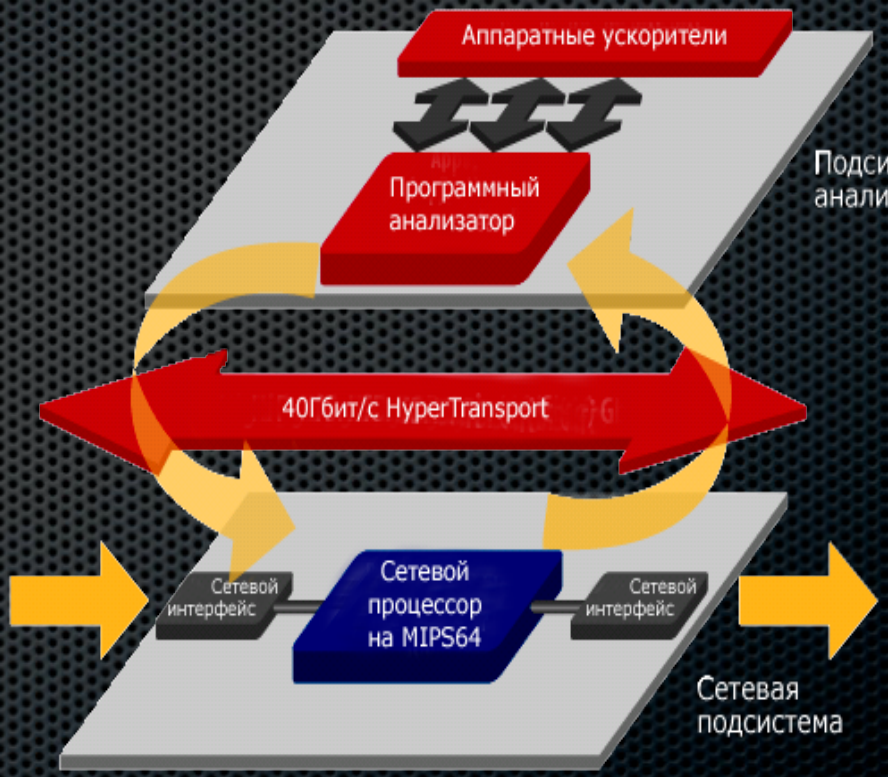
Типовые задачи анализа

- Межсетевые экраны
- Системы обнаружения и предотвращения атак
- Системы контентной фильтрации
- Антиспам
- WAF
- DLP
- …

- Фильтрация потоков
  - Radix tree
  - Bloom filters
- Поиск шаблонов
  - Boyer-Moore
  - Aho-Corasick
  - Regular expressions
- Реконструкция потоков
- Реконструкция данных уровня приложений

Программно-аппаратный анализатор трафика: основные проблемы

- Захват
  - pcap
  - PF_RING (Linux)
  - Сетевые процессоры
- Фильтрация
  - BPF
  - Встроенные в ОС МСЭ (iptables, pf, и т.д.)
  - Фильтрация на сетевых картах
  - Сетевые процессоры
- Анализ
  - Программный (поиск шаблонов, реконструкция потоков, и т.д.)
  - Аппаратный (сетевые процессоры)
- Инъекция
  - Варианты размещения
    - Разрыв канала
    - Изменение маршрутизации
  - MITM
    - Контрольные суммы
    - TCP Seqnum



Пример: платформа Bivio 7000

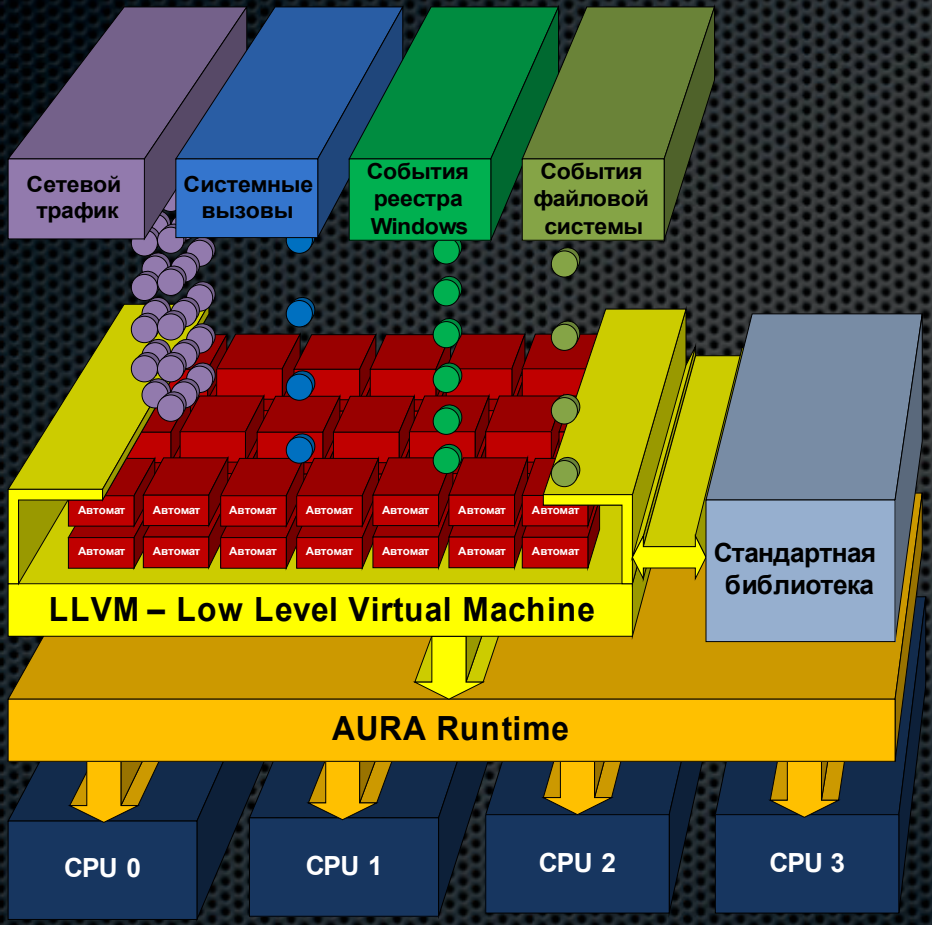
Мягкое vs жёсткое реальное время

- Обнаружение без фильтрации
  - Задержка может накапливаться
  - Soft realtime
- Обнаружение с фильтрацией, MITM
  - Накопление задержки недопустимо
  - Soft/hard realtime

Ключевые факторы

- ☑ Ресурсы аппаратуры общие для всех этапов
- ☑ Ошибки каждого этапа увеличивают ошибки всех последующих этапов – ошибка накапливается
- ☑ Этап анализа наиболее сложный вычислительно

Программная платформа AURA



- Язык AURA
- Система прогона (runtime)
- Поставщики событий
  - Сетевой трафик: IP, ICMP, TCP, TCP stream и т.д.
  - Системные вызовы: Linux 2.4, 2.6, Windows 2000/XP+
  - Специальные: события журналов ОС, реестра, файловой системы, процессов
- Стандартная библиотека
  - Алгоритмы анализа событий
  - Динамические структуры данных
  - IPC
- Framework
  - Инфраструктура разработки анализаторов

Язык AURA

STATL
C++

AURA

- «Программа» - автомат специального вида
- Компиляция в байткод LLVM
- Компиляция в native код при загрузке, либо JIT
- Автоматный IPC – обмен событиями и общие структуры данных

$$Scenario: (S, P_S, T, P_T, s_0, I, g, q)$$

- S – множество состояний;
- P_S – множество логических предикатов T – множество переходов;
- состояний;
- P_T – множество логических предикатов переходов;
- s_0 – начальное состояние;
- I – множество экземпляров автомата;
- g – глобальное окружение;
- q – глобальная очередь таймера.
- Множество экземпляров зависит от истории выполнения.

Язык AURA

- Каждый сценарий может включать в себя:
  - Множество состояний
  - Множество переходов:
    - Поглощающий (consuming)
    - Не поглощающий (nonconsuming)
    - Свертка (unwinding)
  - Предикат перехода (логическое выражение над входным событием и состоянием окружения)
  - События:
    - События от поставщиков событий (трафик, узловые события, сообщения)
    - Таймеры
  - Вызов внешних функций (стандартная библиотека, пользовательские библиотеки)

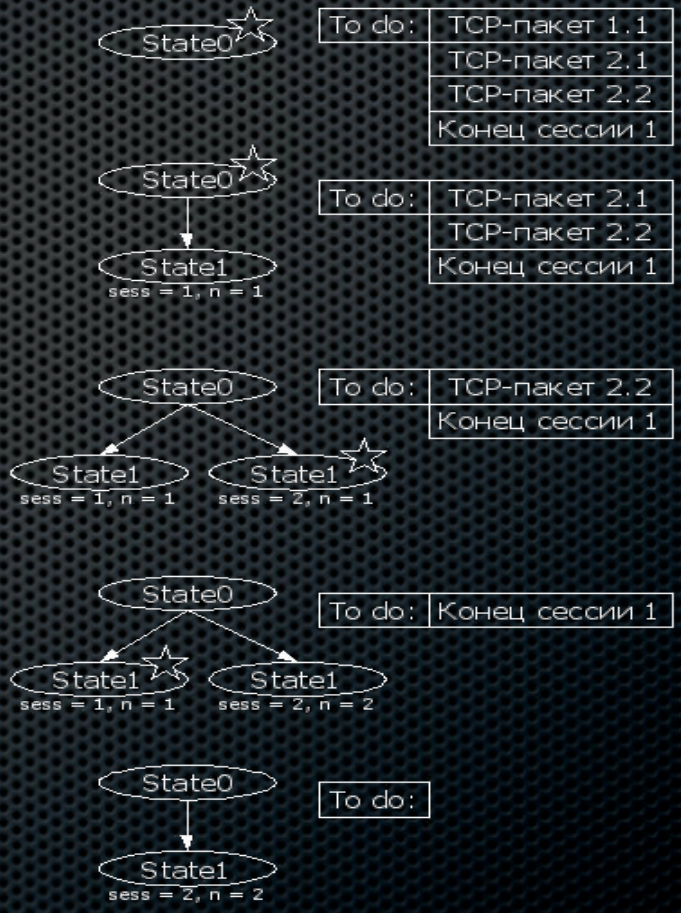
Язык AURA: пример

```
scenario TCPGather ( NetTCPStreamEvent streamEv,
        NetTCPStreamCloseEvent closeEv ) {
       u_int_32       sess;
       initial state state0 {}
       state state1 {}

       nonconsuming transition state0 -> state1
         event NetTCPStreamEvent ( true )
         {sess = streamEv.tcpStreamId;}

       consuming transition state1 -> state1
         event NetTCPStreamEvent ( streamEv.tcpStreamId
              == sess ) { StopEventProcessing ();}

       unwinding transition state1 -> state1
         event NetTCPStreamCloseEvent
                  ( closeEv.tcpStreamId == sess &&
                    closeEv.tcpStreamDirection ==
                    IDS_TCP_FORWARD )  {} };
```

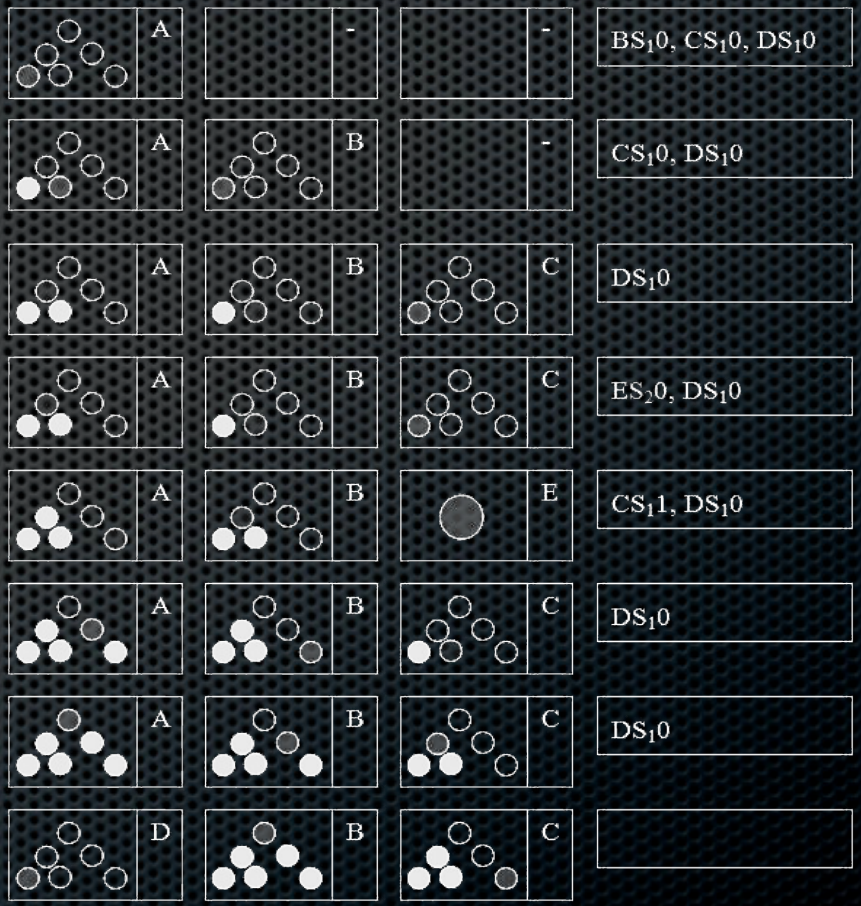


LLVM

- OpenSource проект: http://llvm.org/
- Полноценный оптимизирующий компилятор
- RISC-подобное «виртуальное» представление машинного кода («байткод»)
- Трансляция из LLVM в ассемблер или машинный код архитектур X86, X86-64, PowerPC 32/64, ARM, Thumb, IA-64, Alpha, SPARC, MIPS и CellSPU
- Just-In-Time компиляция для X86, X86-64, PowerPC 32/64.
- Результирующий код быстрее полученного с помощью GCC -O3

Параллельная система прогона (runtime)

- Асинхронная очередь событий
- Единица планирования — пара (событие, дерево экземпляров автомата)
- Приоретизация событий и автоматов



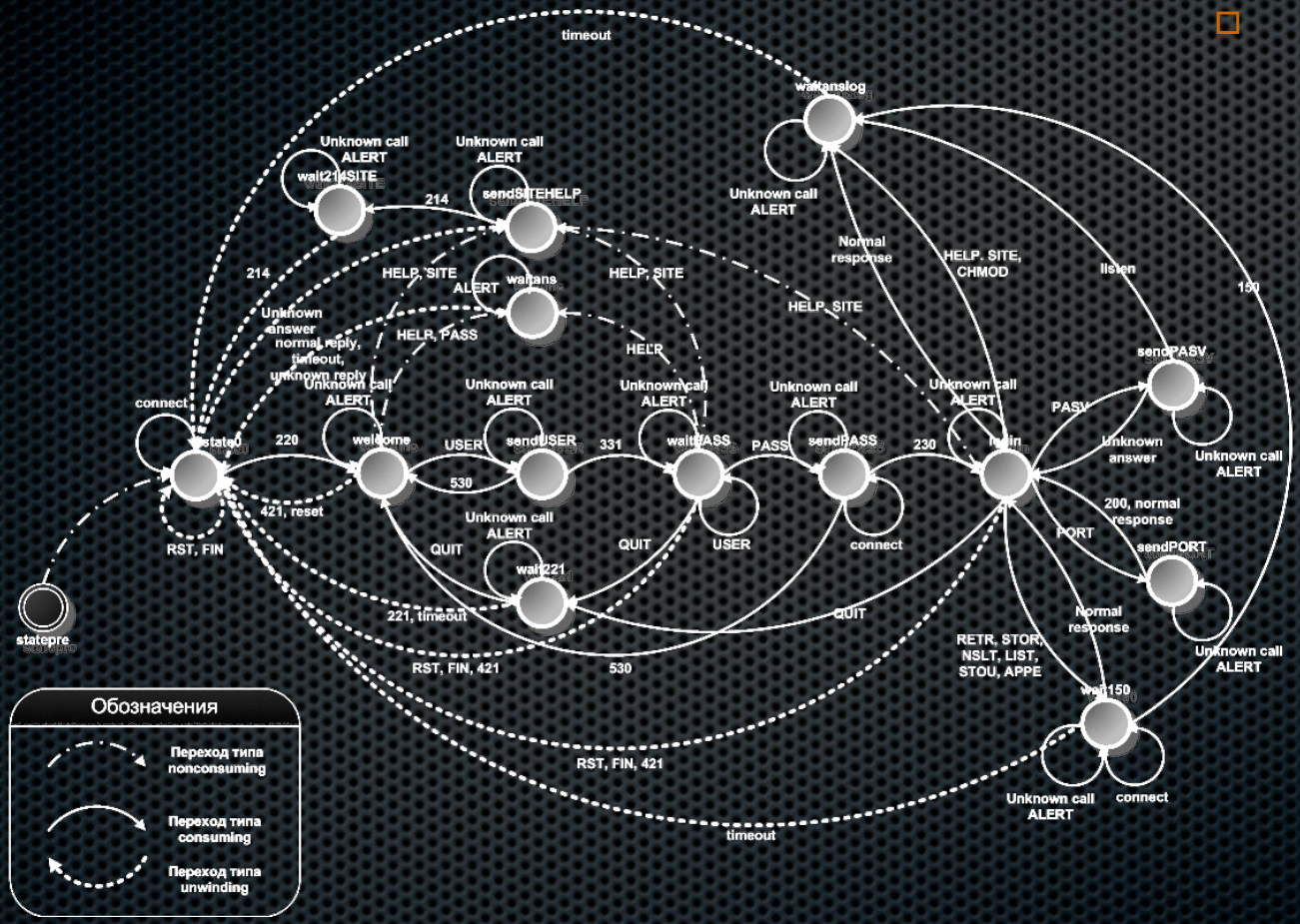
Поддерживаемые операционные системы

- Linux userland
  - События сетевого трафика, реконструкция TCP
  - События процессов (запуск, завершение, fork(), clone() и т.д.)
  - События файловой системы
- Windows userland
  - События сетевого трафика, реконструкция TCP
  - События процессов
  - События реестра
  - События файловой системы
- FreeBSD kernelspace
  - События сетевого трафика, реконструкция TCP
  - Фильтрация событий

Примеры решаемых задач

- Сигнатурный анализ трафика
  - Преобразование из формата Snort в предикатные деревья на языке AURA
- Анализ узловых событий - обнаружение bindshell
  - Поймать accept()
  - Поймать fork(), построить дерево потомков
  - Поймать bind/listen в потомке
  - Поймать connect/accept извне к потомку.

Примеры: модели нормального поведения протоколов и сервисов



- Модель сервера FTP
  - Отслеживает состояние сервера по вводимым командам FTP
  - Обнаруживает «запрещённые» системные вызовы
  - Построена на основе спецификации протокола и исходного кода proftpd

Эксперименты

- Производительность системы прогона на одном ядре: ~15 млн. событий в секунду
- Анализ и фильтрация трафика в качестве FreeBSD netgraph node: ~20Мб/сек*
- Обнаружение участков полиморфных NOP-зон в трафике: ~4Мб/сек на одном ядре (вычислительно сложный анализ)
- **Планируется**:
  - эксперименты на Sun Niagara
  - использование PF_RING в ОС Linux

* 3ГГц Intel Xeon, ~7000 сигнатур, без префильтрации — все пакеты прошли через анализатор

Спасибо за внимание

- Контактная информация:
  - Денис Гамаюнов: gamajun@lvk.cs.msu.su
  - Дмитрий Казачкин: zok@lvk.cs.msu.su
  - Тел. +7 (495) 939 46 71
  - Москва, 119899 Ленинские горы вл. 1/52, факультет ВМК МГУ имени М. В. Ломоносова, к. 764