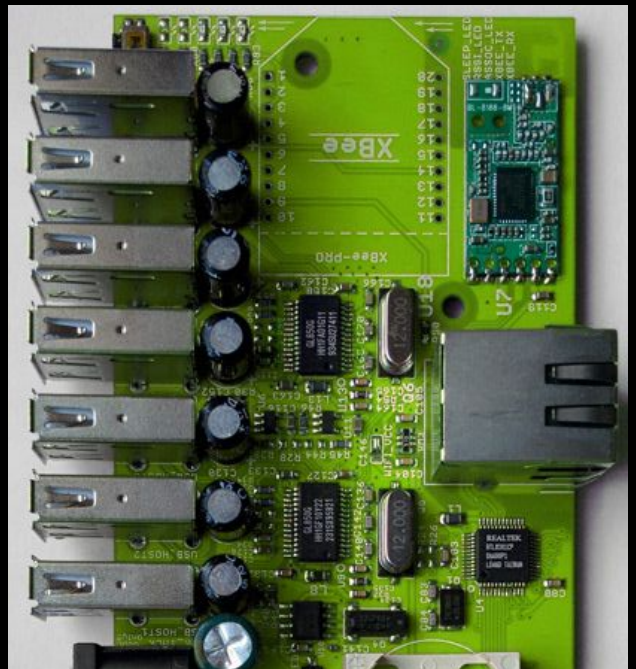
Обратная разработка бинарных файлов с помощью Kaitai Struct

Михаил Якшин

Whitebox Labs, Kaitai Project

ZERONIGHTS

Давайте знакомиться?



В Whitebox Labs мы разрабатываем контроллеры для экосистем:

- Аквариумы
- Террариумы
- Гидропоника
- Контроль производства вина, пива, сыра
- Любые другие замкнутые экосистемы for fun and profit

Чем управляет контроллер?

Датчики

уровня воды, протечки, температуры, качества воды (pH, растворенный кислород, проводимость, ORP, соленость…), потока, влажности, освещенности, камеры, уровня аккумуляторов, ...

Актуаторы

помпы, дозаторы, источники света, диммеры, охлаждение / нагрев, автоматические кормушки, скиммеры, реакторы, реле (произвольная нагрузка), управление электропитанием, ...

Проблема №1: vendor lock-in

- Что это значит?
- Контроллеры общаются с периферией (датчики и актуаторы) по своим, закрытым и проприетарным протоколам.
- Купив однажды контроллер фирмы X, впредь пользователь обречен покупать все железо только фирмы X.

Как ее решаем мы?

- **Производим** собственный контроллер
- **Не производим** собственные датчики и актуаторы
- Reverse engineering протоколов взаимодействия существующих устройств сторонних производителей
- Реализация протокола в нашем контроллере

Что нам нравится и не нравится

Нравится

- Честный “clean room” AKA “black box” reversing
- Делать разбор протоколов быстро
- Иметь гарантированно совпадающие реализации на JavaScript, Python и Ruby

Не нравится

- Делать отдельно:
  - разбор протокола
  - документацию
  - реализацию
- Писать одно и то же по многу раз на разных языках программирования

Что такое Kaitai Struct?

Декларативный язык описания бинарных форматов

1. Описываем формат
2. Проверяем с помощью средств визуализации (ksv)
3. Компилируем в библиотеку на целевом языке (ksc)
4. Используем полученный API

Декларативное или императивное?

Императивное

```
code = in.read_char();
len = in.read_int();
buf = in.read_str(0x40);
```

Декларативное

```
typedef struct {
    char code;
    int len;
    char buf[0x40];
} SomeRecord;
```

Декларативное или императивное?

Императивное

Как прочитать или записать формат

Привязано к конкретному языку

Как правило, реализуется вручную ⇒ зачастую рассадник ошибок

Декларативное

Что хранится в структуре

Не привязано к конкретному языку

Нужен какой-то способ претворять в жизнь:

- интерпретатор
- компилятор

Что можно парсить с помощью KS?

- Коммуникационные протоколы
- Упакованные embedded firmware
- Контейнерные форматы файлов
- Исполняемые файлы (executables, objects, байт-код, ...)
- Файлы контента (базы данных, таблицы, тексты, графика, текстуры, ...)
- Файловые системы

Что не стоит парсить с помощью KS?

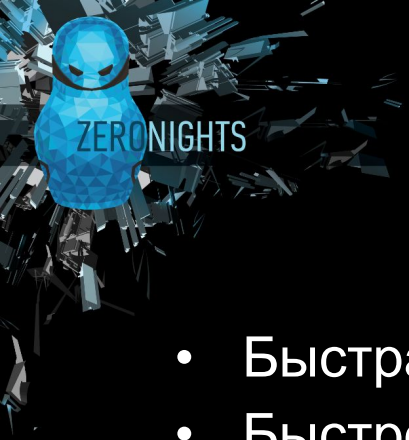
Форматы, не разработанные для машинной обработки:

- Там, где есть двусмысленность
- Там, где машина парсинга должна иметь сложную модель состояний
- Там, где нужно уметь заглядывать вперед по потоку*
- LL, LR, LALR, SLR, PEG, регулярные языки и т.п. — не к нам

Целевые языки

KS умеет компилироваться в:

- C++/STL
- C#
- Java
- JavaScript
- Perl
- PHP
- Python
- Ruby



Чем еще хорош Kaitai Struct?

- Быстрая разработка
- Быстро работающий результат (т.к. компилятор)
- Отлаживаем на медленных, но удобных скриптовых языках,
  в production — на быстром C++
- Генерируемый код вполне читаем
- Free/open source: компилятор - GPL, runtime - MIT/Apache
- .ksy = YAML ⇒ легко писать свои инструменты
- Можно откомпилировать .ksy в наглядную схему формата с помощью GraphViz

Планы на ближайшие 3 часа

- Установить инструментарий и подготовиться к работе
- Рассмотреть несколько задач
  - Исходные данные доступны на http://kaitai.io/workshop/
  - Послушать предысторию
  - Посмотреть, как задача решается с помощью KS
  - Попробовать KS в действии
- Пообедать ;)

… инсталляция …

http://kaitai.io/

Use case #1: Загадочный utmp

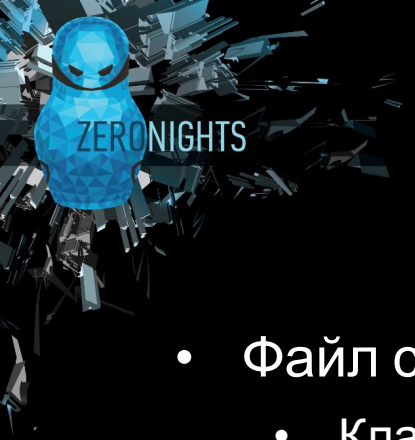
http://kaitai.io/workshop/1

Задача: дано

- Файл с IoT-устройства с неизвестной ОС (предположительно, *NIX-подобной)
- Известно, что файл находился в /var/run/utmp
- Доступа к самому устройству нет
- Никакой другой информации об устройстве нет

Задача: нужно

- Научиться читать содержимое этого файла
- Выводить список тех, кто логинился на устройство, и когда его включали/выключали
- Т.е. фактически написать работающий аналог who(1), last(1) и т.д.



Что мы знаем про utmp?

- Файл с бинарной структурой
  - Классический security-by-obscurity: якобы это должно останавливать злоумышленников, которые хотят замести следы
- Единого стандарта нет:
  - Есть более-менее устойчивый стандарт Linux / glibc
  - Есть стандарт(ы) BSD
  - Есть стандарт(ы) SysV
  - Зачастую структура зависит от архитектуры платформы (32 vs 64)
- man utmp(5)?

… демонстрация ...

Use case #2: Транспортное табло

http://kaitai.io/workshop/2

A long time ago in a galaxy far, far away

Транспортное табло

- Открытый WiFi с подозрительно намекающим названием
- Роутинга в Internet в нем нет
- Зато есть какие-то пакеты в локальной сети
- Что же в них?

… демонстрация ...

Use case #3: Кибер-археология

http://kaitai.io/workshop/3

Исходные данные

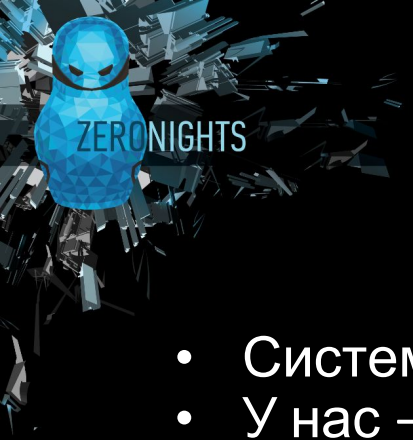
- От начала до конца — use case из реальной жизни
- База данных в российской библиотеке
- Разработка: середина 1970-х - середина 1990-х
- Файлы БД есть
- Исходники программ утеряны

Платформы

- изначально — ЕС-1022 (~ IBM System/360)
- затем — ЕС-1036 (~ доработанная System/360)
- Искра-226 (~Wang 2200)
- IBM PC / DOS

Задача

- Восстановить содержимое БД
  - По словам библиотекарей — внутри БД десятки (если не сотни) тысяч уникальных записей о публикациях, которые вводились за период 1970-1990-х
  - В идеале — в виде какой-нибудь современной SQL-based СУБД
    - Видимо, устроит вывод в что-нибудь типа JSON или INSERT-операторов SQL



Собираем сведения

- Система работала до середины 1990-х
- У нас — самая последняя из существующих версий БД
- Пройдя путь от ЕС, последние версии работали все-таки под IBM PC (x86)
  - IBM System/360 - big-endian?
  - x86 - little-endian?
  - Кодировка - “КОИ8” или “ГОСТ альтернативная” (866)?
- Вспоминают, что в формате БД использовалась какая-то сериализация, похожая на “ISO 2…”

Первые прикидки

- Самый большой файл — 26 мегабайт
  - 16 bit? 32 bit? Явно не 64, даже в 1990-х.
- 26 мегабайт / 10 000 записей ~ 2600 байт на запись
- 26 мегабайт / 100 000 записей ~ 260 байт на запись

Предметная область

Предположительно (+ по воспоминаниям и по результатам общения с библиотекарями), в БД есть информация:

- о публикациях (ГОСТ 7.1-84)
- о людях (авторах публикаций)
- об источниках (журналы / монографии, в которых статьи публиковались)
- об организациях (в которых работали авторы)

Что можно ожидать найти в БД?

- Собственно данные
  - Фиксированная длина записи
  - Записи с произвольной длиной (строки)
- Индексы
- Журнал транзакций

… демонстрация ...

Use case #4: Контейнерный формат

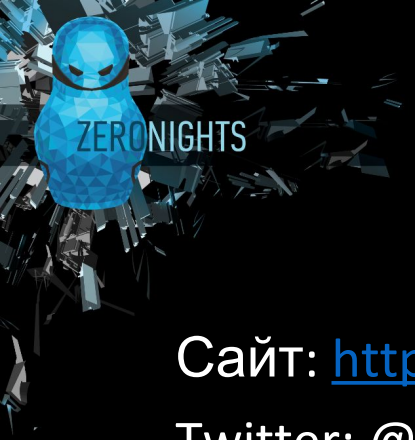
http://kaitai.io/workshop/4

… демонстрация ...

Use case #5: Упакованный firmware

http://kaitai.io/workshop/4

… демонстрация ...



Спасибо за внимание!

Сайт: http://kaitai.io/

Twitter: @kaitai_io

GitHub: https://github.com/kaitai-io/kaitai_struct

E-mail: greycat.na.kor@gmail.com

Gitter: http://gitter.com/kaitai_struct/Lobby