Безопасная сервис-ориентированная архитектура

на примере приложения голосового управления умным домом

Wire Snark, 2017

Об авторе

- Разработчик демонов и сервисов в GNU/Linux и Android
- Исследователь безопасности Java-приложений (Андроид, серверные приложения)
- Основатель Paranoid Security
- Со-основатель
  DEF CON Нижний Новгород defcon-nn.ru

wsnark@tuta.io

@wsnark

@wiresnark

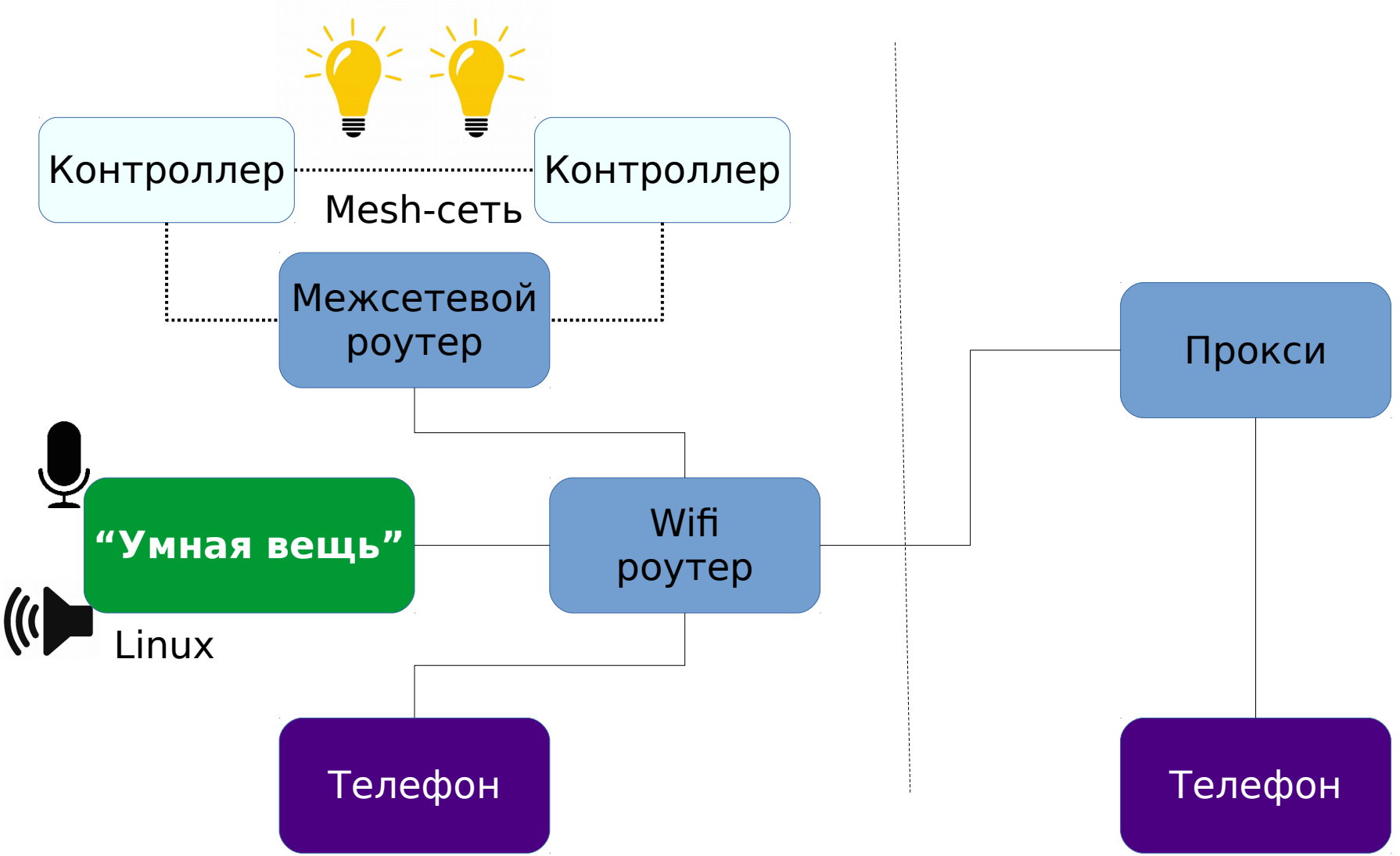
GnuPG fingerprint
4497 F125 194A 47AD 0E1B
05A3 E12E D410 7595 4ED2

[m]@wsnark:matrix.org

“Интернет вещей”

- Стартапы: игры в “умный дом” без обеспечения важнейших свойств
  - Устройства перестают работать при проблемах с доступом к сети
  - Проблема обновления прошивки
  - Отсутствие безопасности и приватности
- Корпорации: умный дом как способ владения пользователями
  - Все данные и мета-данные попадают на корпоративные сервера
- Twitter: @internetofshit

Типовая схема умного дома



“Контроллер”

- ~50Mhz ARMv4 CPU
- 32Kb+ RAM
- 1Mb+ flash storage
- Low-energy mesh network
- RTOS

“Умная вещь”

- 0.5GHz ARMv7 CPU
- 100Mb+ RAM
- 1GB+ flash storage
- Wifi, BLE
- Embedded Linux (OpenEmbedded, Yocto)
- Серверо-подобная:
  - без GUI, предоставляет сервисы
- Клиенто-подобная:
  - без администрирования, минимальные конфиги

Голосовое управление умным домом: требования

- Работа онлайн – через Alexa Voice Services (AVS) с потенциальной возможностью добавить поддержку других подобных сервисов
- Ограниченная работа оффлайн – локальное распознавание речи
- Безопасность

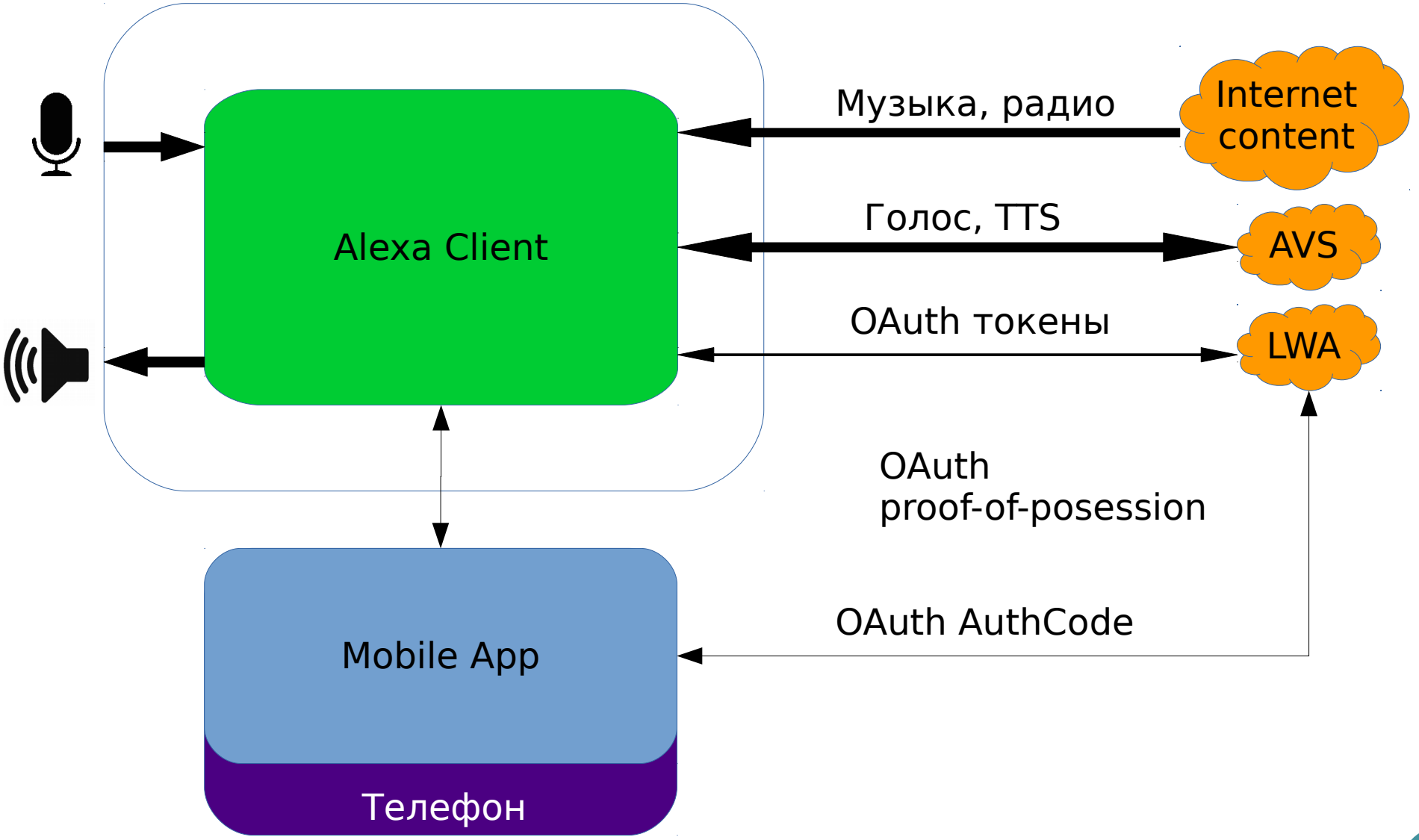
Что такое Алекса?

- “Умный” голосовой помощник от Amazon
- Языки: английский и немецкий
- Предоставляет информацию о погоде, пробках, новости
- Отвечает на вопросы, предоставляет информацию, в т.ч. из Википедии
- Проигрывает музыку, радио
- Позволяет задавать таймеры, будильники, списки дел и покупок

Что такое Алекса?

- Амазон Эхо
- Приложения для Android и iOS
- Сторонние голосовые команды - Alexa Skills Kit, Alexa Smart Home
- Сторонние клиенты - Alexa Voice Services

Приложение глосового управления – Amazon sample



Alexa Voice Services (AVS)

- Сервис с сохранением состояния клиента
- Транспорт – только HTTP/2 с ALPN (Application-Layer Protocol Negotiation – расширение TLS)
- Взаимодействие через события (event на клиенте) и директивы (directive, от сервиса)
- Авторизация через Login With Amazon OAuth 2.0

Протокол HTTP/2

- Бинарный протокол, развитие Google SPDY
- Определен в RFC 7540, май 2015
- Сложный
- Поддержка в HTTP-клиентах не очень хорошая
- Поддержка нескольких потоков внутри одного TCP-соединения
- Server Push

Транспорт в AVS

- Каждый запрос на распознавание обрабатывается в отдельном HTTP/2-потоке (stream)
- Директивы, инициированные AVS, пересылаются в downchannel stream – потоке, который всегда должен быть открыт
- Начальное состояние клиента и любое изменение состояния необходимо отправлять на сервер
- Коммуникация в составных (multipart) HTTP/2-сообщениях
- Запросы в формате JSON, звук – в бинарном формате, разбитом на части (отправка голосовых данных в реальном времени)

Интерфейсы AVS

- SpeechRecognizer – распознавание запроса
- SpeechSynthesizer – проигрывание ответа
- Alerts – напоминания, таймеры и т.п.
- AudioPlayer – проигрывание музыки, радио
- Speaker – управление громкостью
- PlaybackController
- Settings
- System

Нетривиальные особенности AVS

- Директива StopCapture: поддержка “remote speech endpointing”, т.е. детектирование окончания речи самим AVS
- Директива ExpectSpeech: уточняющий вопрос от AVS пользователю, диалог
- Директивы Play, AdjustVolume, SetMute: от внешнего (по отношению к клиенту) управления – например, из мобильного приложения.
- Директива Play: запрос на проигрывание потока с произвольного URL

Авторизация через Oauth 2.0

- https://oauth.net RFC 6749, 6750, 6819 + куча расширений. Очень большой и сложный.
- Легко реализовать небезопасным образом: https://sakurity.com/oauth

Security Development Lifecycle (SDL)

1) Обучение
2) Определение требований
   - Требования безопасности и приватности
3) Проектирование
   - Модель угроз, анализ поверхности атаки, требования безопасного проектирования
4) Реализация
5) Верификация
6) Выпуск
7) Реагирование на ициденты

Требования безопасности и приватности

- Конфиденциальность
- Целостность
- Доступность

- Аутентификация
- Авторизация
- Неотказуемость

Модель угроз

- Определяем ценные данные и функции системы (assets)
  - Доступ к микрофону и динамикам
  - Доступ к управлению устройствами умного дома
  - Токены доступа к AVS
  - Настройки, системные параметры
  - Логи
  - Мета-данные (факт и длительность передачи данных, использования ресурса или устройства и т.д.)
  - Операционная система

Модель угроз

- Определяем точки входа и выхода, поверхность атаки
  - Интерфейс к AVS (включая сторонние аудио-сервисы)
  - Интерфейс к LWA
  - Интерфейс к мобильному приложению
  - Интерфейс к управлению умным домом
  - Голосовой вход (подсистема записи и распознавания), звуковой выход (подсистема разбора, декодирования и проигрывания)
- Определяем потенциальных атакующих (threat agents)
  - Вредоносное ПО (ненацеленное)
  - Корпораций (конкуренты, инвесторы)
  - Преступники (воры, вымогатели, операторы ботнетов и продавцы персональных данных)
  - Профессионалы (спецслужбы, частные детективы, коллекторы)
  - Индивиды (соседи, знакомые, сотрудники, скрипт-кидс)

Модель угроз

- Исходя из предполагаемых возможностей атакующих, выбираем допущения и границы доверия (assumptions, trust boundaries). Клиенту не следует доверять
  - сторонним серверам с контентом
  - серверам AVS и LWA
  - мобильному приложению
  - голосовому вводу
  - устройствам из умного дома (включая управляющие)

Модель угроз

- Классифицируем все вероятные угрозы по модели STRIDE
  - Spoofing: имитация, подделка чего-либо
  - Tampering: вмешательство, изменение
  - Repudiation: отрицание операции
  - Information disclosure: раскрытие информации
  - Denial of service: отказ в обслуживании
  - Elevation of privilege: повышение привилегий

Принципы безопасного проектирования

- Минимизация поверхности атаки
- Безопасность по умолчанию
- Минимальные привилегии
- Разделение обязанностей
- Эшелонированная защита (defence in depth)
- Безопасная обработка ошибок
- KSS – Keep Security Simple

Изоляция недоверенного кода

- Минимизация привилегий
  +
- Разделение обязанностей
  +
- Минимизация поверхности атаки
  =
- Изоляция в виде обработчиков с одним входом и выходом

http://cr.yp.to/qmail/qmailsec-20071101.pdf

Компонентный дизайн

- auth-manager
- alexa-client
- iot-controller

Платформа

- Yocto Linux / OpenEmbedded
- 500MHz NXP/Freescale i.MX6UL
- 512Gb RAM
- 4Gb eMMC

Локальное распознавание голоса

- Коммерческие проекты, например Sensory THF – работают практически “из коробки”, поддержка, тренировка акустических моделей
- Открытые – CMU Sphinx (poketsphinx) https://github.com/cmusphinx

Выбор средств разработки

- Языки: C, C++, Python, Node.js, Go, Rust, Java
- Межпроцессное взаимодействие: D-Bus, Unix Pipes, Unix Sockets, MQTT
- Системные библиотеки: Qt, GLib2
- Медиа-фреймворк: GStreamer, VLC

Межпроцессное взаимодействие через D-Bus

- Адресация и роутинг
- Активация сервисов
- Граница безопасности (можно задать политики безопасности)
- Возможна передача файловых дескрипторов **(!)**
- Бинарный протокол
- Требует биндингов, нетривиально в использовании. Для C – GDbus, C++ - QtDbus

Критерии выбора языка

- Обязательные
  - HTTP/2 client with ALPN
  - D-Bus bindings
- Желательные
  - Популярность
  - История использования во встраиваемых системах
  - Производительность
  - Быстрый старт
- Приятные
  - Легкий в использовании и изучении
  - Низкое потребление ресурсов

Результаты анализа

- C: небезопасный; дорогостоящая разработка
- C++: очень сложный, также проблемы с безопасностью и дорогостоящая разработка
- Rust: нет готового HTTP/2-клиента
- Python: HTTP/2-клиент в альфа-версии, низкая производительность
- Node.js: долгий старт и низкая производительность
- JVM-based: нет биндингов D-Bus, долгий старт
- **Go** – лучший кандидат под данные критерии!

Медиа-фреймворк

- VLC
  - тяжело собрать под Yocto Linux: требует JVM с AWT, OpenJDK для Yocto без него. Пришлось использовать Oracle JDK
  - Непопулярен как фреймворк – скорее плеер
- GStreamer
  - в системе “из коробки”, поддерживается SoC-вендором
  - стандарт де-факто
  - очень гибкий для PoC: gst-launch-1.0 позволяет делать почти всё!
  - сложный API, проблемы с биндингами (Glib2)
  - недоверенный C-код

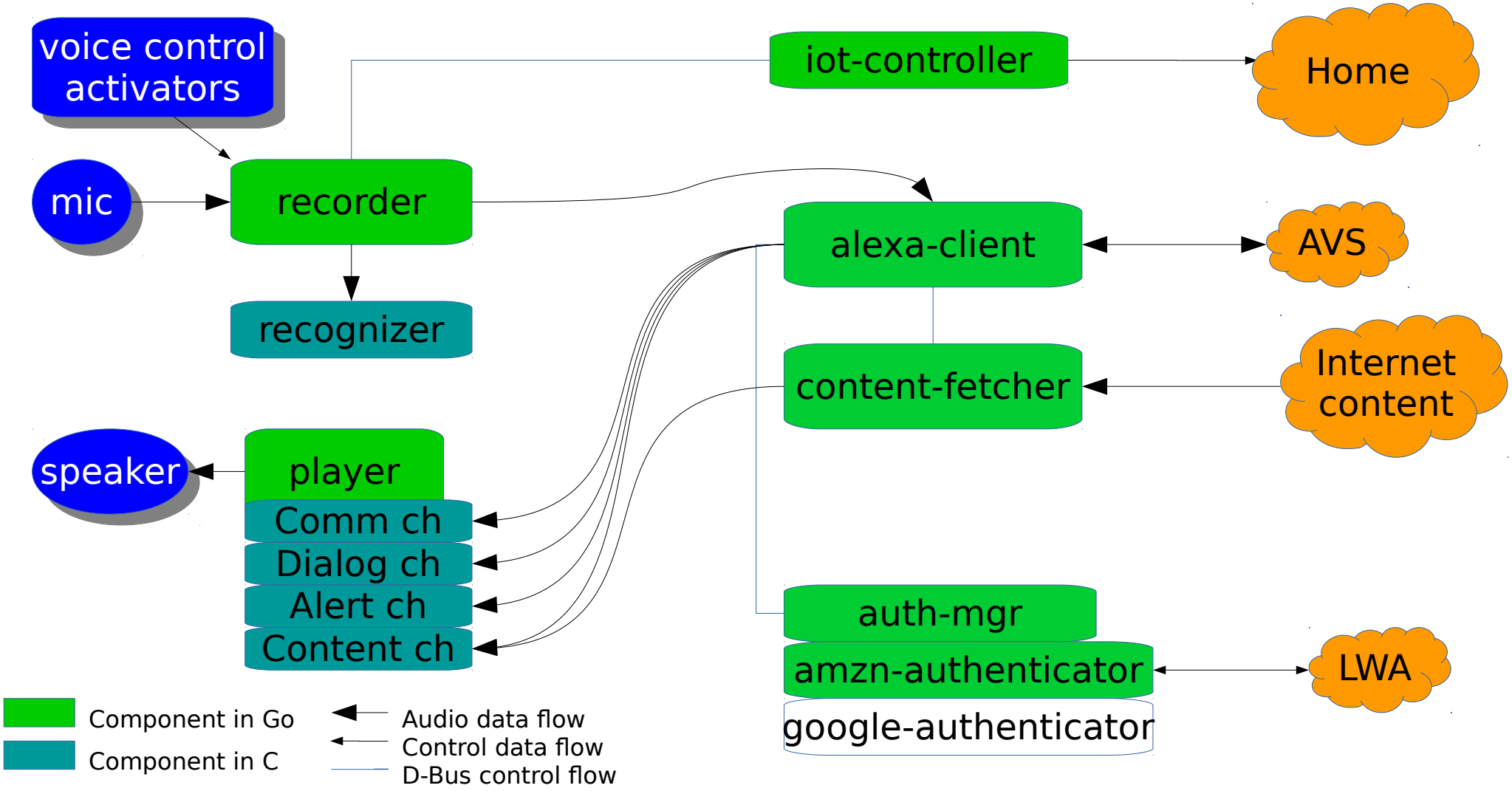
Подводные камни AVS

- Неспецифицированные области
- Поведение не по спецификации
  - Нужна качественная обработка ошибок на стороне клиента
- Проблемы Golang net/http – слишком высокоуровневый API
- Сложности со снижением задержек

Компонентный дизайн

- Вдохновлен OKWS https://www.usenix.org/legacy/event/usenix04/tech/general/full_papers/krohn/krohn.pdf
- MIT Security 6.858. Разделение обязанностей https://www.youtube.com/watch?v=dNl22h1kW1k
- Основные идеи и методы реализации:
  - Разделение обязанностей и доверия
  - Независимую функциональность следует разнести по различным сервисам
  - Каждый сервис запускается отдельным юзером
  - Недоверенный код запускается в изолированных контейнерах
  - Для межпроцессного взаимодействия с изолированным кодом применяется передача файловых дескрипторов

Компонентный дизайн



Компонент alexa-client

```
interface com.lightpad.AlexaClient {
  // Recognize initiates dialog between AVS and the user
  // dialogToken is a unique and secure (not forgeable)
  // token representing this dialog - to reference it later.
  // voicesrc is file descriptor pointing to voice data
  // source (normally a pipe)

func Recognize(dialogToken string, voicesrc fd) void

}
```

Атаки на alexa-client

- Из сети:
  - захваченный (subverted) AVS
  - захваченный TLS (защита: certificate pinning)
  - вредоносный аудио-поток (защита: выделенный процесс content-fetcher для работы с недоверенными аудио-потоками из интернета; выделенный процесс player для декодирования).
- Из локальных процессов:
  - захваченный recorder
  - захваченный player (высокий риск, поскольку декодирует недоверенные аудио-потоки)
  - захваченный authmgr
  - Неавторизованные процессы вызывают alex-client API (защита: D-Bus policy, позволяющая использовать этот API только нескольким юзерам/процессам)
  - Другие процессы захватывают D-Bus-имя alexa-client (защита: D-Bus policy, запрещающая это)
- Из локальных файлов:
  - Конфигурационные файлы

Поверхность атаки в самом alexa-client

- Соединение с AVS
- Парсер ответов AVS
- Компоненты Go Runtime (TLS, HTTP, D-Bus)
- Нет границы безопасности между AVS и alexa-client, так что компрометация AVS ~ компрометации клиента

Последствия от захвата alexa-client

- Раскрытие пользовательских запросов к AVS, предоставление произвольных ответов, контроль громкости
  - Нет защиты
- Получение доступа к пользовательскому микрофону через директиву RequestSpeech или бесконечная отправка данных (неотсылка директивы StopCapture)
  - DialogToken
  - Таймаут

Изоляция недоверенного кода

- Недоверенный код
  - player (включая независимые аудио-каналы)
  - recognizer
- Технологии изоляции
  - seccomp: ограничение доступа к системным вызовам ядра
  - Linux namespaces & cgroups: разделение системных ресурсов (процессов, файловых систем, сети и т.д.) между группами процессов
  - AppArmor, SELinux: детальное разграничение доступа к файловой системе, другим процессам и ресурсам

Изоляция недоверенного кода

- Практические готовые решения для изоляции (sandboxing)
  - Chroot (небезопасный)
  - Systemd-nspawn (https://www.freedesktop.org/software/systemd/man/systemd-nspawn.html)
  - Firejail (https://firejail.wordpress.com/, 12 CVE в 2016-2017)
  - LXC (Linux Containers)
  - Docker

Заключение

- В IoT нужны инженерные подходы
- Модель угроз помогает построить безопасную архитектуру
- Используйте безопасные языки Go, Rust, Python в Embedded Linux!
- Изолируйте код, работающий с недоверенными данными (например, парсеры)

Вопросы?

Ссылки

- OWASP https://www.owasp.org
  - Top10, Top10 Mobile, Cheat Sheets, Code Review Guide, Testing Guide, Secure by Design и другие
- OAuth & OpenID Connect
  https://oauth.net/2/
  https://openid.net/connect/
  https://sakurity.com/oauth (критика)
- Lucida (former Sirius): open-source alternative to Alexa http://lucida.ai/
- CMU Sphinx: open-source speech recognition
  https://github.com/cmusphinx

Ссылки

- DJB about qmail, trusted code base minimization vs minimal privilege http://cr.yp.to/qmail/qmailsec-20071101.pdf
- Курс MIT по безопасности систем: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-858-computer-systems-security-fall-2014/
- OKWS, secure web server https://www.usenix.org/legacy/event/usenix04/tech/general/full_papers/krohn/krohn.pdf

Ссылки

- Firejail
  - https://firejail.wordpress.com
  - CVE’s https://firejail.wordpress.com/download-2/cve-status/