Окончательная версия текста

**Вступление**

Добрый день, позвольте представиться: меня зовут Тимошенко Александр. Мои коллеги: Кострицына Анастасия и Цатурьян Константин. Мы являемся студентами 307 кафедры Московского Авиационного Института. Сегодня мы хотели бы предоставить вашему вниманию наш совместный проект, который был выполнен под руководством преподавателя нашей кафедры: Максимова Алексея Николаевича. Тема нашего сегодняшнего повествования будет разработка прототипа системы управления автономным роботом на базе Kaspersky OS.

**(Саша)Слайд 2. Актуальность, цель работы.**

В наше время всё более широкое применение начинают приобретать автономные роботы, так как они позволяют ощутимо облегчить жизнь человека. Например, можно уже сейчас увидеть на дорогах роботов-доставщиков, производящих доставку еды и не только. Использование роботов в различных сферах деятельности имеет преимущества, такие как: повышение качества производимой продукции или услуг, точность, сокращение расходов. Кроме уже реализованных решений существует большое количество ещё не автоматизированных задач , которые можно делегировать роботам.

Целью нашей работы является разработка автономной системы управления роботами для их навигации в закрытых помещениях. Разрабатываемая система должна обладать достаточной гибкостью для настройки под конкретную задачу и обеспечивать хорошую защиту от взлома.

**(Саша)Слайд 3. Описание системы в целом. КТС.**

Для достижения поставленной цели спроектирована информационная система, состоящая из сервера, камер видеонаблюдения, подключенных через Ethernet и мобильных роботов, подключенных к серверу по Wi-Fi. Сервер обрабатывает видеопоток, поступающий с камер, находящихся на стенах помещения, распознает положение роботов и направляет им соответствующие команды.

В ходе разработки мы ориентировались на роботов на платформе Raspberry PI. Связь между сервером и роботами осуществляется по протоколу TCP/IP. Робот принимает команду, декодирует её и выполняет.

Потенциально в данной системе можно использовать MQTT, что позволит возможно вести лог команд и сделать их более комфортными для восприятия человеком. Сервер работает на базе операционной системы Linux, однако его можно запустить и на KOS, что позволит повысить безопасность. В любом случае роботы находятся под управлением операционной системы KasperskyOS. Её мы выбрали, так как это отечественная кибериммунная система, то есть обладающая встроенной защищенностью от подавляющего большинства видов кибератак. Продукты, под управлением Kaspersky OS, способны выполнять свои функции даже в агрессивной среде. Кибериммунный подход в основе KasperskyOS не позволяет злоумышленнику влиять на работу системы, даже если ему удается взломать какой-то из ее компонентов, не обладающих кибериммунитетом.

**(Костя)Слайд 4. Решение проблемы должной безопасности. Или почему именно KOS?**

Данная операционная система вместила в себя максимум тех разработок в сфере безопасности, которые были доступны на текущий момент. Она основана как бы на “трех слонах” – архитектуре MILS, архитектуре FLASK и других дополнениях от команды Kaspersky. Основные принципы описаны на слайде. Благодаря этому она является достаточно защищенной от известных уязвимостей и хорошо подходит для решений в сфере IoT.

Архитектура MILS основана на концепциях разделения и контроля потока информации. Механизмы разделения, поддерживающие как недоверенные, так и заслуживающие доверия компоненты; гарантируют, что решение полной безопасности не поддается обходу, может быть оценено, всегда вызывается и защищено от несанкционированного доступа (это принципы называются NEAT, см. приложение 1).

Архитектура FLASK – обеспечивает гибкую поддержку политик безопасности (правил доступа). Расшифровывается как Flux Advanced Security Kernel. Она позволяет нам более гибко настраивать политики доступа и безопасности в целом для конкретных задач.

**(Костя)Слайд 5. VFS и GPIO**

Основной частью нашего проекта является решение проблемы преобразования сигналов. Компонент VFS (Virtual File System) содержит в себе реализацию сетевого стека, поэтому POSIX-вызовы направляются на него, затем VFS вызывает сетевой драйвер. С помощью встроенных ограничений политик безопасности мы можем настроить управление вызовами. Так как двигатели робота будут подключены через GPIO (General-Purpose Input/Output) у Raspberry PI, то далее, команды направленные сервером через TCP/IP, декодируются, происходит вызов соответствующей функции управления двигателями. Через подключенный драйвер GPIO мы посылаем импульсы на двигатели, происходит движение робота.

**(Костя)Слайд 6. Диаграмма последовательностей**

На данной диаграмме представлена последовательность взаимодействия робота с сервером. Когда робот включается, он пробует подключиться по заданной сети к серверу. Затем производится аутентификации и установка подключения. Когда канал передачи готов, сервер обрабатывает снимки с камер, находит данного робота, отправляет ему команду. Затем робот выполняет команду, тем временем сервер продолжает обрабатывать снимки, отправляет новую команду и так далее.

**(Настя)Слайд 7. Диаграмма классов.**

На данном слайде представлена диаграмма классов, показывающая программное устройство робота. На диаграмме проиллюстрированы следующие связи – от интерфейса ICommandReceiver и IEngine наследуется по 2 соответствующих класса. А эти интерфейсы в свою очередь являются неотъемлемой частью робота, но инициализируются отдельно от объекта класса Robot, то есть тип связи здесь – агрегация.

Каждый робот содержит в себе Command Receiver и некоторое количество моторов, которые приводят его в движение. В текущей реализации их типы и количество можно легко изменять. Также робот содержит в себе радиус ходовых колес, что поможет для получения обратной связи о пройденном пути.

Для более удобной отладки мы создали по шаблону проектирования Strategy два типа двигателей: Console Engine подразумевает то, что работа мотора будет отображена в виде вывода в консоль; GPIO Engine подразумевает отправку команды через GPIO на реальный мотор робота.

Аналогично для удобства отладки были реализованы классы-наследники от интерфейса ICommandReceiver, таким образом, мы можем переключаться между ручным вводом команд и принятием команд от сервера по TCP/IP. При вводе с консоли все понятно, а когда инициализируется обмен данными через TCP, то в конструктор класса TCPCommandReceiver передается статический локальный адрес сервера. Затем созданный объект получения команд передается в конструктор класса Robot.

Итак, на этом все и мы можем подвести итоги.

**(Настя)Слайд 8.1. Выводы по имеющемуся.**

Итак, в результате работы представлен рабочий прототип системы, который демонстрирует возможность разработки защищенных управляющих приложений, использующих FLASK архитектуру для обеспечения контроля выполнения политики безопасности. На основе достигнутого нами результата можно уже собрать такую защищенную систему и приспособить её под конкретные задачи. Однако это ещё не все.

**(Настя)Слайд 8.2. Перспективный взгляд на проект.**

В будущем мы планируем и далее развивать систему. А именно конечной целью мы подразумеваем создание абсолютно гибкой системы. Она позволит не только применять её для управления реальными роботами, но и использовать её модификацию для обучения студентов или специалистов в IoT сфере.

Конкретно это будет происходить так: мы планируем подключить существующий прототип к игровому движку, например Unreal Engine 4, в нем у нас будет имитация камеры и самого робота. Модель камеры будет передавать данные на сервер, сервер будет их обрабатывать, посылать на робота команды, робот (а именно симуляция Raspberry PI под QEMU) будет отрабатывать абсолютно так же, как бы это делал реальный робот. При отправке сигнала на двигатели, сигнал пойдёт на их модели, которые в свою очередь будут передавать данные среде Unreal Engine 4, как и куда двигается робот. Тем самым мы сможем создавать различные виртуальные окружения и тестировать в них поведения робота и систему в целом.

**Заключение**

На этом все. Спасибо за внимание. Мы будем рады ответить на появившиеся вопросы.

**Приложение 1**

**NEAT**

N (non-bypassable) – невозможно обойти: компонент не может использовать другой канал связи, включая механизмы более низкого уровня, для обхода монитора безопасности.

E (evaluable) – поддающийся оценке: любой доверенный компонент может быть оценен до уровня гарантии, требуемого для этого компонента. Это означает, что компоненты являются модульными, хорошо спроектированными, хорошо определенными, хорошо реализованными, небольшими, малозатратными и т.д.

A (always-invoked) – всегда вызываемый: каждая попытка доступа/сообщение проверяется соответствующими мониторами безопасности (т.е. монитор безопасности не будет просто проверять первый доступ, а затем пропускать все последующие обращения/сообщения).

T (tamperproof) – защита от несанкционированного доступа: система контролирует права "изменять" код, конфигурацию и данные монитора безопасности; предотвращение несанкционированных изменений.