kaspersky

Исследование безопасности: CODESYS Runtime – фреймворк для управления ПЛК

Александр Ночвай

18.09.2019 Version 1.0

Оглавление

Исследование безопасности технологий, которые используются разработчиками систем автоматизации и имеют потенциал применения на промышленных объектах по всему миру, является одним из приоритетных направлений работы Центра реагирования на инциденты информационной безопасности промышленных инфраструктур «Лаборатории Касперского» (Kaspersky ICS CERT).

В этой статье мы продолжаем рассказ об исследовании популярных OEM-технологий, входящих в состав продуктов большого количества производителей. Уязвимости в таких технологиях с большой вероятностью сказываются на безопасности многих, если не всех продуктов, в которых они используются. Иногда это могут быть сотни продуктов, используемых на производстве и в критической инфраструктуре. Именно так обстоит дело с CODESYS Runtime® – фреймворком для разработки и выполнения программ автоматизированного управления технологическим процессом от компании CODESYS. Права на торговую марку принадлежат компании 3S-Smart Software Solutions, входящей в состав CODESYS Group.

Согласно официальной информации от компании-разработчика, CODESYS Runtime уже адаптирован для более чем 350 устройств от различных вендоров, которые используются в энергетики, промышленном производстве, системах интернета вещей и промышленного интернета вещей и т.д. Отметим также, что в реальности эта цифра значительно больше, так как устройства множества компаний, которые используют фреймворк CODESYS Runtime для своих ПЛК, не находятся в официальном списке. Количество таких устройств постоянно растет – в 2016 году их было всего 140, – и мы не будем удивлены, если оно продолжит расти и в будущем.

Фрагменты технической информации были удалены в связи с требованием CODESYS Group. Более подробную информацию можно запросить по адресу security@codesys.com.

Фреймворк

В современном мире использование готового программного кода в своем продукте – скорее правило, чем исключение. Это позволяет разработчикам нового продукта не заниматься «изобретением велосипеда» и сокращает время на разработку.

Степень влияния стороннего кода на использующий его программный продукт и степень влияния на безопасность системы, в которой этот продукт используется, может быть различной.

Сторонний код часто используется для реализации конкретной функции или набора функций, таких как, отрисовка изображения, отображение пользовательского интерфейса, печать на принтере или сохранение данных в БД. Мы уже проводили исследования безопасности и обнаруживали уязвимости в различном стороннем коде. Например, в 2017 году – для части программно-аппаратного комплекса контроля соблюдения лицензионных соглашений и защиты приложения от «взлома» SafeNet Sentinel, а в 2018 – для библиотеки OPC UA от OPC Foundation.

С фреймворком CODESYS ситуация другая: разработчик ПЛК на основе CODESYS адаптирует фреймворк для работы на своих аппаратных комплексах и разрабатывает при необходимости дополнительные модули, используя предоставляемые CODESYS сервисные функции. Конечный пользователь ПЛК (инженер) разрабатывает при помощи среды разработки CODESYS код программы автоматизации технологического процесса. И поток выполнения разработанных дополнительных модулей и программы автоматизации технологического процесса контролируется на ПЛК адаптированным под него вариантом CODESYS Runtime.

Контроль фреймворком потока программы означает, что использование фреймворка накладывает ограничения на уровне архитектуры – на этапе проектирования продукта. Иначе говоря, фреймворк представляет собой уже созданный сложный механизм, а пользовательский код должен стать винтиком в этом механизме.

С точки зрения обеспечения безопасности при использовании фреймворка разработчику необходимо ответить на следующие вопросы:

- Что внутри этого фреймворка?
- Как он работает?
- Как мне сделать свое ПО безопасным, если уязвимость не в моем коде, а во фреймворке?

Данная статья посвящена исследованию безопасности CODESYS Runtime. В ней мы дадим ответы на первые два вопроса: что происходит внутри этого фреймворка и как он работает. Также мы продемонстрируем один из способов обнаружения уязвимостей без возможности анализа исходного кода.

Описание объекта исследования: CODESYS Runtime

Прежде чем говорить о результатах исследования безопасности объекта, необходимо разобраться в том, что он из себя представляет. Техническое описание CODESYS Runtime, приведенное в этой главе, было сформировано нами в ходе его изучения.

Связка продуктов компании CODESYS Group

Компания CODESYS Group разрабатывает два основных продукта:

1. Среда разработки – CODESYS Development System
2. Среда исполнения – CODESYS Runtime

Оба продукта работают в связке. CODESYS Development System представляет собой IDE для разработки программ управления устройствами, на которых запущен CODESYS Runtime. В среду включено множество инструментов для упрощения процесса разработки и тестирования.

В контексте нашего исследования важно, что CODESYS Development System – это кастомизируемая среда разработки. На её основе созданы IDE SoMachine от компании Schneider Electric, TwinCAT от компании Beckhoff Automation, IdraWorks от компании Bosch, Wagilo Pro от компании WAGO, одноименные IDE CODESYS от компаний Owen, STW Technic, prolog-plc и другие IDE.

Для программирования контроллера с помощью IDE CODESYS Development System, на нем должен быть запущен CODESYS Runtime. Для корректного запуска CODESYS Runtime на конкретном устройстве его необходимо адаптировать под выбранную операционную систему и железо. Согласно информации на официальном сайте CODESYS, сами разработчики CODESYS адаптировали CODESYS Runtime только для 15 устройств. Дистрибьютерами же CODESYS Runtime был адаптирован для более чем 350 устройств.

Среди адаптированных CODESYS Runtime есть версии:

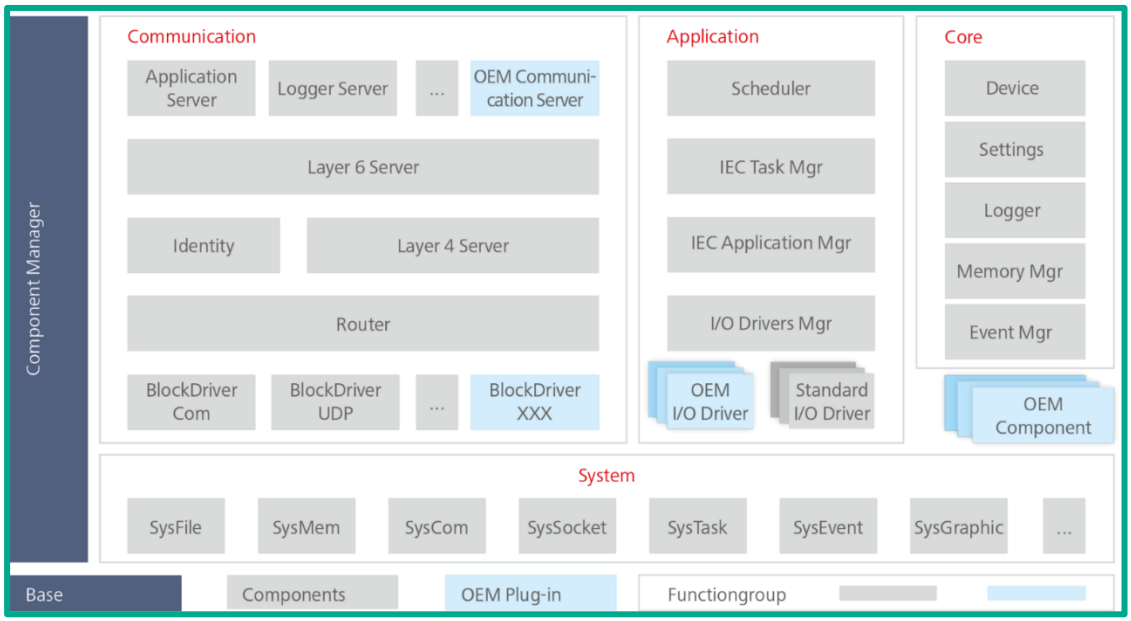
- Для одноплатных компьютеров с операционной системой Linux, такие как Raspberry Pi, UniPi, BeagleBone;
- Для Soft PLC на Windows и Linux;
- Для ПЛК компаний ASEM S.p.A, exceet electronics AG, Hitachi Europe GmbH, Hans Turck GmbH & Co. KG, elrest Automationssysteme GmbH, Janz Tec AG, Kendrion Kuhnke Automation GmbH, Beijer Electronics, ifm electronic gmbh, Nidec Control Techniques Limited, Advantech Europe B.V, WAGO Kontakttechnik GmbH & Co. KG, KEB Automation KG, Berghof Automation GmbH и многих других.

Архитектура

Компоненты

CODESYS Runtime основан на компонентно-ориентированной архитектуре, то есть представляет собой набор компонентов – модулей, на которые разделена каждая логическая или функциональная часть CODESYS Runtime.

Каждый компонент ответственен за свою задачу и область действия – например, отвечает за логирование, за сетевое взаимодействие, за взаимодействие по серийному кабелю, за распределенную нагрузку на ядрах, за отладку программы и т.д.



Компонентно-ориентированная архитектура CODESYS Runtime
Источник: https://www.codesys.com/fileadmin/data/Downloads/Broschueren/CODESYS-Runtime-en.pdf

Среди всех компонентов CODESYS Runtime можно выделить основные:

1. **Component Manager, или CM** – компонент для запуска и инициализации всех остальных компонентов в системе;
2. **System Components** – группа компонентов для описания взаимодействий с операционной системой и с железной составляющей. Компоненты из этой группы отвечают за взаимодействие с физическими портами, с файловой системой, с работой по динамическому и статическому выделению памяти и т.д.
3. **Communication Components** – группа компонентов для взаимодействия с внешним миром, например по сети или через последовательный кабель;
4. **Application components** – компоненты управления программой ПЛК;
5. **Core components** – компоненты для управления ПЛК и его состоянием.

У разработчика есть несколько способов расширить CODESYS Runtime:

1. Заменить существующие компоненты;
2. Написать собственные компоненты;
3. Написать собственные компоненты, которые будут расширять функциональность уже существующих компонентов.

Структура компонентов

Компоненты CODESYS – это динамические библиотеки (аналог *.dll в ОС Windows и *.so в ОС Linux). Все компоненты загружаются компонентом **Component Manager**.

CODESYS Runtime может иметь статическую и динамическую сборку.

Если CODESYS Runtime имеет статическую сборку, то программный код компонентов содержится в самом исполняемом файле.

Если же CODESYS Runtime имеет динамическую сборку, то в конфигурационном файле указывается список загружаемых компонентов, а файлы компонентов находятся отдельно от исполняемого файла.

Структура самого файла компонента не являлась объектом данного исследования, однако можно с уверенностью сказать, что в этой структуре содержится:

- Программный код компонента;
- Название компонента, имя автора, версия и описание компонента;
- Различные контрольные суммы и магические значения.

Структура интерфейсов взаимодействия

Более интересным, чем структура файла компонента, для исследования является то, каким образом CODESYS Runtime взаимодействует с внешними компонентами, и как внутренние компоненты взаимодействуют друг с другом.

Каждый компонент обязан содержать следующие функции: функцию инициализации, функцию экспорта, функцию импорта, функцию обработки событий, функции удаления и создания своей сущности. Также компонент обязан иметь свой уникальный числовой идентификатор.

Функции удаления и создания своей сущности – опциональны. Они оказались пустыми для большинства компонентов. Поэтому мы не станем здесь их рассматривать. Остальные же функции будут рассмотрены далее.

Реализация функции инициализации

Функция инициализации – аналог entry point для PE и ELF файлов, с той лишь разницей, что непосредственную работу компонента она не начинает. Вызов этой функции осуществляется компонентом Component Manager.

```
01 Удалено по требованию вендора
02: {
03:   Удалено по требованию вендора
04:   Удалено по требованию вендора
05:   Удалено по требованию вендора
06:   Удалено по требованию вендора
07:   Удалено по требованию вендора
08:   Удалено по требованию вендора
09:   Удалено по требованию вендора
[...]
17:   Удалено по требованию вендора
18: }
```

Декомпилированный код функции инициализации компонента CmpBlkDrvUdp из группы Communication

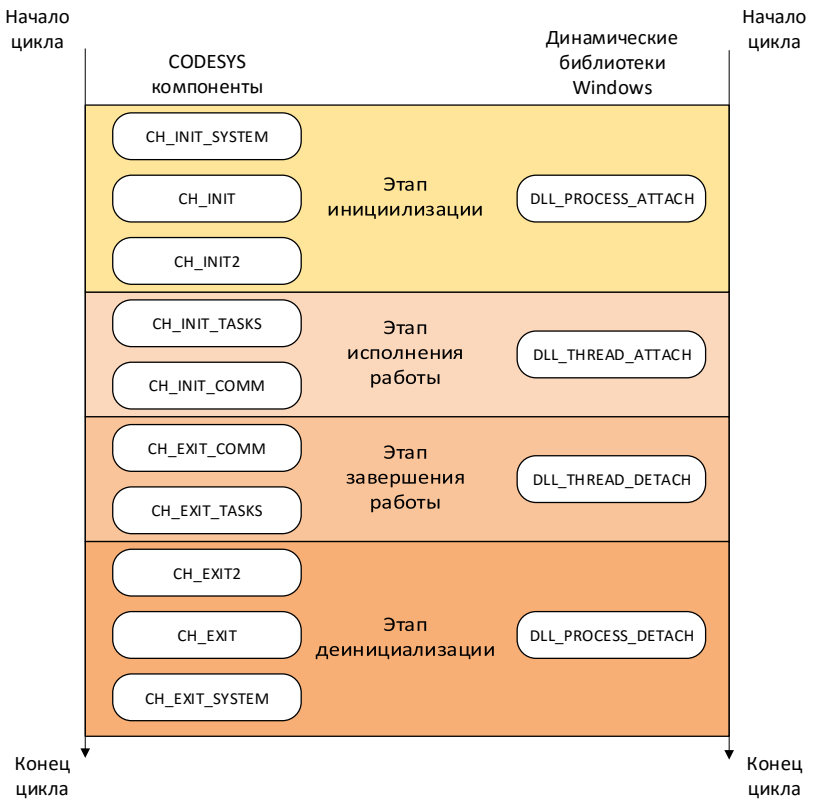
Функция **ModuleCmpBlkDrvUdp_entry** – функция инициализации. Аргумент для этой функции структура INIT_STRUCT. Обычно эта функция вызывается с использованием **Component Manager** для заполнения структуры. Функция инициализации заполняет поля в структуре, среди которых – все вышеперечисленные функции и идентификатор компонента, который для компонента CmpBlkDrvUdp равен 7.

Реализация функции обработки событий

Следующая интересная функция – это функция обработки событий. Для компонента CmpBlkDrvUdp этой функцией является **ModuleCmpBlkDrvUdp_hook.** Эта функция определяет по полученному идентификатору события, что от неё требует выполнить **Component Manager**.

Идентификаторы основных событий:

- CH_INIT_SYSTEM – идентификатор 1. В случае, если компонент относится к группе системных компонентов (System Components), то ему необходимо инициализироваться;
- CH_INIT – идентификатор 2. Компоненты должны инициализировать все локальные переменные;
- CH_INIT2 – идентификатор 3. Компонент должен инициализироваться;
- CH_INIT_TASKS – идентификатор 5. Компонент может запустить свои потоки выполнения;
- CH_INIT_COMM – идентификатор 6. Компонент может начать коммуникацию;
- CH_EXIT_COMM – идентификатор 10. Компоненту необходимо завершить все каналы коммуникаций;
- CH_EXIT_TASKS – идентификатор 11. Компоненту необходимо остановить и завершить все созданные им потоки выполнения;
- CH_EXIT2 – идентификатор 13. Компоненту необходимо сохранить все данные перед вызовом CH_EXIT;
- CH_EXIT – идентификатор 14. Компоненту необходимо освободить память;
- CH_EXIT_SYSTEM – идентификатор 15. В случае если компонент относится к группе системных компонентов (System Components), то ему необходимо освободить память;
- CH_COMM_CYCLE – идентификатор 20. Вызывается каждый цикл. Используется для созданных потоков выполнения.



Сравнение жизненного цикла работы компонента CODESYS с циклом работы динамически подключаемой библиотеки Windows (Dynamic Link Library)

События для обработки компонентом, как и события вызова DLL Windows, созданы «зеркальными» парами: [CH_INIT_SYSTEM – CH_EXIT_SYSTEM], [CH_INIT – CH_EXIT] и т.д.

```
001: Удалено по требованию вендора
002: {
003:   Удалено по требованию вендора
004:   Удалено по требованию вендора
005:   Удалено по требованию вендора
006:   Удалено по требованию вендора
007:   Удалено по требованию вендора
008:
009:   Удалено по требованию вендора
010:    {
011:      Удалено по требованию вендора : // Initialization of the component's variables
012:        Удалено по требованию вендора
013:        Удалено по требованию вендора
014:        Удалено по требованию вендора
015:        Удалено по требованию вендора
016:        Удалено по требованию вендора
017:        Удалено по требованию вендора
018:        Удалено по требованию вендора
019:        Удалено по требованию вендора
020:        Удалено по требованию вендора
021:        Удалено по требованию вендора
022:        Удалено по требованию вендора
023:        Удалено по требованию вендора
024:        Удалено по требованию вендора
025:      Удалено по требованию вендора : // Initialization of the component. Setting values from the
configuration file
026:        Удалено по требованию вендора
027:        Удалено по требованию вендора
```

```
028:        Удалено по требованию вендора
029:        Удалено по требованию вендора
030:          Удалено по требованию вендора
031:        Удалено по требованию вендора
032:        Удалено по требованию вендора
033:        Удалено по требованию вендора
034:        {
035:          iУдалено по требованию вендора
036:            Удалено по требованию вендора
037:          Удалено по требованию вендора
038:            Удалено по требованию вендора
039:          Удалено по требованию вендора
040:        }
041:        Удалено по требованию вендора
042:        {
043:          Удалено по требованию вендора
044:          Удалено по требованию вендора
045:          {
046:            Удалено по требованию вендора
047:            Удалено по требованию вендора
048:              Удалено по требованию вендора
049:            Удалено по требованию вендора
050:            Удалено по требованию вендора
051:            Удалено по требованию вендора
052:          }
053:        }
054:        Удалено по требованию вендора
055:      Удалено по требованию вендора  // Creating a communication thread and starting communication
056:        Удалено по требованию вендора
057:        Удалено по требованию вендора
058:        {
059:          Удалено по требованию вендора
060:            Удалено по требованию вендора
061:          Удалено по требованию вендора
062:        }
063:        Удалено по требованию вендора
064:      Удалено по требованию вендора : // Completing communication
065:        Удалено по требованию вендора
066:        Удалено по требованию вендора
067:        {
068:          Удалено по требованию вендора
069:          Удалено по требованию вендора
070:          Удалено по требованию вендора
071:        }
072:        Удалено по требованию вендора
073:      Удалено по требованию вендора:  // Saving data and releasing it
074:        Удалено по требованию вендора
075:        {
076:          Удалено по требованию вендора
077:          {
078:            Удалено по требованию вендора
079:              Удалено по требованию вендора
080:            Удалено по требованию вендора
081:          }
082:          Удалено по требованию вендора
083:          Удалено по требованию вендора
084:        }
085:        Удалено по требованию вендора
086:        {
087:          Удалено по требованию вендора
088:          Удалено по требованию вендора
089:        }
090:        Удалено по требованию вендора
091:      Удалено по требованию вендора  // Updating the communication socket
092:        Удалено по требованию вендора
093:        {
094:          Удалено по требованию вендора
095:          Удалено по требованию вендора
096:          {
097:            Удалено по требованию вендора
098:            Удалено по требованию вендора
099:          }
```

```
100:          Удалено по требованию вендора
101:        }
102:        Удалено по требованию вендора
103:      Удалено по требованию вендора :
104:        Удалено по требованию вендора
105:    }
106: }
```

Декомпилированный код функции ModuleCmpBlkDrvUdp_hook, в котором продемонстрирована обработка событий компонентом CmpBlkDrvUdp

На примере функции **ModuleCmpBlkDrvUdp_hook** видно следующее:

- Компоненты могут пренебрегать предусмотренными правилами работы с событиями и объединять обработку нескольких событий в одном обработчике, как это сделано, например, в рассматриваемой функции: при обработке события **CH_INIT_COMM** происходит одновременно инициализация потока выполнения и его непосредственный запуск, хотя для последней задачи предназначено событие **CH_INIT_TASKS**;
- Компоненты не обязательно должны обрабатывать все события. В том числе, компонентам не обязательно обрабатывать оба «симметричных» события, если одно событие из «зеркальной» пары уже было обработано – например, компонент **CmpBlkDrvUdp** не обрабатывает событие **CH_EXIT,** хотя им было обработано событие **CH_INIT**.

Реализация функций импорта и экспорта

Функция экспорта и функция импорта используют механизм, который в совокупности дает эффект работы экспортируемых и импортируемых функций в динамических библиотеках ОС Windows и Linux. Главная разница функций экспорта и импорта с библиотеками ОС Windows и Linux в том, что во время работы этих функций происходит регистрация экспортируемых функций и инициализация указателей на импортируемые функции.

```
01: Удалено по требованию вендора
02: {
03:     Удалено по требованию вендора
04:     Удалено по требованию вендора
05:     Удалено по требованию вендора
06:     Удалено по требованию вендора
07: }
08:
09: Удалено по требованию вендора
10: Удалено по требованию вендора
11: Удалено по требованию вендора
12: Удалено по требованию вендора
13:
14: Удалено по требованию вендора
15: {
16:   Удалено по требованию вендора
17:
18:    Удалено по требованию вендора
19:
20:   Удалено по требованию вендора
21: }
22:
```

Псевдо-код функции экспорта компонента CmpRasPi (есть исключительно в CODESYS Runtime for Raspberry PI)

Функция **CMRegisterAPI** использует в качестве первого аргумента указатель на заполненный массив, содержащий сведения об экспортируемых функциях, а в качестве последнего – идентификатор компонента. Для примера экспортируемой функции: на

строке 10 есть заполненная структура exported_function с указателем на функцию sub_84e5bc0, именем функции “raspiyuv”, хешем 0xF81Fd05 и версией 0x3050400.

Таким образом компонент CmpRasPi предоставляет API для взаимодействия с модулем камеры на устройстве Rapsberry PI для всех остальных компонентов CODESYS и прикладных программ. Пример использования этого API продемонстрирован в примере проекта Camera.project для CODESYS Control for Raspberry Pi.

```
1: PROGRAM PLC_PRG
2: VAR
3:   xTakePicture: BOOL;
4: END_VAR
5:
6: IF xTakePicture THEN
7:   Raspberry_Pi_Camera.Still('-o Picture.jpg');
8:   xTakePicture := FALSE;
9: END_IF
```

Код программы проекта Camera.project

Функция импорта в компонентах пытается найти функции, экспортируемые другими компонентами, и запомнить их местоположение.

```
01: Удалено по требованию вендора
02: {
03:   Удалено по требованию вендора
04:   Удалено по требованию вендора
05:   Удалено по требованию вендора
06:   Удалено по требованию вендора
07:   Удалено по требованию вендора
08:   Удалено по требованию вендора
09:   Удалено по требованию вендора
10:   Удалено по требованию вендора
11: [...]
12:
```

Функция импорта модуля CmpApp

Функция CMGetAPI2 ищет функцию, которая была зарегистрирована другим компонентом. Первый аргумент – это имя функции, второй – значение, куда сохранить полученный указатель на функцию, третий – это ожидаемый хеш функции, если передается, и последний – ожидаемая версия.

Перед этим все эти функции были зарегистрированы компонентом SysTarget.

```
01: Удалено по требованию вендора
02: Удалено по требованию вендора
03: Удалено по требованию вендора
04: Удалено по требованию вендора
05: Удалено по требованию вендора
06: Удалено по требованию вендора
07: Удалено по требованию вендора
08: Удалено по требованию вендора
09: Удалено по требованию вендора
10: Удалено по требованию вендора
11: Удалено по требованию вендора
12: Удалено по требованию вендора
13: Удалено по требованию вендора
14: Удалено по требованию вендора
15: Удалено по требованию вендора
16: Удалено по требованию вендора
```

Фрагмент массива с экспортируемыми функциями

Механизм импорта и экспорта функций предоставляет разработчикам основную функциональность для создания собственных компонентов или расширения возможностей существующих.

Настройка компонентов

Механизм настройки компонентов демонстрирует, как работает часть архитектуры CODESYS Runtime. Поэтому он будет рассмотрен здесь как заключительная часть главы про архитектуру CODEYS Runtime.

Пользователю CODESYS Runtime предоставляется возможность управления компонентами через конфигурационный ini-файл. Конфигурационный ini-файл – это обычный текстовый файл, в котором находятся ключи и параметры для настройки компонентов.

```
[...]
28: [CmpWebServer]
29: ConnectionType=0
30:
31: [CmpOpenSSL]
[...]
```

Фрагмент файла конфигурации для CODESYS Control for Raspberry Pi

Component Manager инициализирует все компоненты. В первую очередь он инициализирует системные компоненты, такие как CmpMemPool, CmpLog, CmpSettings и SysFile и т.д.

```
********* CoDeSysControl DEMO VERSION - runs 2 hours*********
[...]
============================================================================
1526222855: Cmp=CM, Class=1, Error=0, Info=4, pszInfo= CODESYS Control V3
1526222855: Cmp=CM, Class=1, Error=0, Info=5, pszInfo= Copyright (c) 3S - Smart Software Solutions GmbH
1526222855: Cmp=CM, Class=1, Error=0, Info=6, pszInfo= <version>3.5.12.0</version> <builddate>Dec 18
2017</builddate>
============================================================================
1526222855: Cmp=CM, Class=1, Error=0, Info=10, pszInfo= System: <cmp>CM</cmp>, <id>0x00000001</id>
<ver>3.5.12.0</ver>
1526222855: Cmp=CM, Class=1, Error=0, Info=10, pszInfo= System: <cmp>CmpMemPool</cmp>, <id>0x0000001e</id>
<ver>3.5.12.0</ver>
1526222855: Cmp=CM, Class=1, Error=0, Info=10, pszInfo= System: <cmp>CmpLog</cmp>, <id>0x00000013</id>
<ver>3.5.12.0</ver>
1526222855: Cmp=CM, Class=1, Error=0, Info=10, pszInfo= System: <cmp>CmpSettings</cmp>,
<id>0x0000001a</id> <ver>3.5.12.0</ver>
1526222855: Cmp=CM, Class=1, Error=0, Info=10, pszInfo= System: <cmp>SysFile</cmp>, <id>0x00000104</id>
<ver>3.5.12.0</ver>
1526222855: Cmp=CM, Class=1, Error=0, Info=10, pszInfo= System: <cmp>SysTimer</cmp>, <id>0x00000116</id>
<ver>3.5.12.0</ver>
1526222855: Cmp=CM, Class=1, Error=0, Info=10, pszInfo= System: <cmp>SysTimeRtc</cmp>, <id>0x00000127</id>
<ver>3.5.12.0</ver>
[...]
```

Фрагмент лога запуска исполняемого файла CODESYS Control for Raspberry Pi

Среди системных компонентов есть компонент CmpSettings. Он интересен нам тем, что функция экспорта этого компонента регистрирует API, которыми пользуются все остальные компоненты для получения параметров из файла конфигурации.

```
01: Удалено по требованию вендора
02: Удалено по требованию вендора
03: Удалено по требованию вендора
04: Удалено по требованию вендора
05: Удалено по требованию вендора
06: Удалено по требованию вендора
07: Удалено по требованию вендора
08: Удалено по требованию вендора
09: Удалено по требованию вендора
10: Удалено по требованию вендора
11: Удалено по требованию вендора
12: Удалено по требованию вендора
13: Удалено по требованию вендора
```

```
14: Удалено по требованию вендора
15: Удалено по требованию вендора
```

Фрагмент массива экспортируемых функций компонента CmpSettings

Функции **SettgGetIntValue** и **SettgGetStringValue** используются большинством компонентов для определения параметров своей работы. По перекрестным ссылкам вызовов этих функций можно определить, какие компоненты могут быть настроены через файл конфигурации, и какие ключи необходимо указать в файле конфигурации.

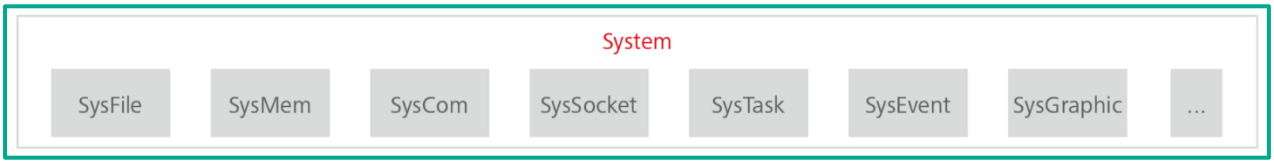
Используя метод поиска вызовов функции SettgGetIntValue, по перекрестным ссылкам можно найти ключ DemoTimeUnlimited для настройки компонента ComponentManager:

```
01: Удалено по требованию вендора
02: {
03:   Удалено по требованию вендора
04:
05:   Удалено по требованию вендора
06:   Удалено по требованию вендора
07:   Удалено по требованию вендора
08:   Удалено по требованию вендора
09:   Удалено по требованию вендора
10:   Удалено по требованию вендора
11:   Удалено по требованию вендора
12:     Удалено по требованию вендора
13:   Удалено по требованию вендора
14: }
```

Ключи настройки компонента ComponentManager

Адаптация

Возможность адаптировать CODESYS Runtime под любое железо и операционную систему – несомненно, главная его особенность. На разработчиках продукта с использованием фреймворка лежит ответственность по адаптации системных компонентов CODESYS Runtime под нужды и требования конкретного применения, включая тип технологического процесса. Адаптированный фреймворк CODESYS Runtime должен уметь взаимодействовать с аппаратными интерфейсами, Ethernet, освобождать и выделять память, работать с таймером, событиями, межпотоковым взаимодействием и т.д.



Системные компоненты из компонентно-ориентированной архитектуры CODESYS Runtime

Адаптация системных компонентов происходит путем экспортирования функций, которые требуют остальные компоненты (этот путь описан в предыдущей главе).

Проведя анализ нескольких вариантов CODESYS Runtime, мы выяснили, что системных компонентов всего 25.

Список основных системных компонентов:

```
SysTimer, SysTimeRtc, SysTime, SysTask, SysTarget, SysSocket, SysShm, SysSemProcess,
SysSemCount, SysSem, SysReadWriteLock, SysProcess, SysOut, SysMutex, SysMsgQ,
SysModule, SysMem, SysInternalLib, SysFile, SysExcept, SysEvent, SysEthernet, SysDir,
SysCpuHandling, SysCom
```

После обеспечения работоспособности системных компонентов разработчик должен создать собственные модули CODESYS Runtime для ПЛК со специфичной функциональностью.

Реализация

В декабре 2016 вышла первая версия CODESYS Control for Raspberry Pi. В июне 2018 года – версия для Linux (CODESYS Control for Linux SL). Помимо этого, еще есть эмулятор CODESYS Control для Windows, который входит в пакет программ CODESYS Development System. Все эти реализации однотипны с реализацией CODESYS Control for Raspberry Pi и имеют схожие или полностью идентичные элементы реализации.

В этой главе мы рассказываем о реализации CODESYS Runtime на примерах CODESYS Control for Raspberry Pi и CODESYS Control for Linux SL.

Файл – установщик

CODESYS Development System передает на устройство Raspberry Pi установщик CODESYS Control, используя SSH client. Сам установщик представляет собой .deb-файл (Debian binary package).

```
# dpkg -c codesyscontrol_arm_raspberry_V3.5.12.0.deb
drwxr-xr-x root/root         0 2017-12-18 09:22 ./
drwxr-xr-x root/root         0 2017-12-18 09:22 ./var/
drwxr-xr-x root/root         0 2017-12-18 09:22 ./var/opt/
drwxr-xr-x root/root         0 2017-12-18 09:22 ./var/opt/codesys/
drwxr-xr-x root/root         0 2017-12-18 09:22 ./var/opt/codesys/backup/
drwxr-xr-x root/root         0 2017-12-18 09:22 ./var/opt/codesys/cmact_licenses/
-rwxr-xr-x root/root      2640 2017-12-18 09:22 ./var/opt/codesys/bacstacd.ini
-rw-r--r-- root/root     20736 2017-12-18 09:22 ./var/opt/codesys/3SLicense.wbb
drwxr-xr-x root/root         0 2017-12-18 09:22 ./var/opt/codesys/restore/
drwxr-xr-x root/root         0 2017-10-09 14:16 ./etc/
-rw-r--r-- root/root       216 2017-10-09 14:16 ./etc/CODESYSControl_User.cfg
drwxr-xr-x root/root         0 2017-12-18 09:22 ./etc/init.d/
-rw-r--r-- root/root      3355 2017-12-18 09:22 ./etc/init.d/codesyscontrol
-rw-r--r-- root/root       158 2017-10-09 14:16 ./etc/3S.dat
-rw-r--r-- root/root       943 2017-10-09 14:16 ./etc/CODESYSControl.cfg
drwxr-xr-x root/root         0 2017-12-18 09:22 ./opt/
drwxr-xr-x root/root         0 2017-12-18 09:22 ./opt/codesys/
drwxr-xr-x root/root         0 2017-12-18 09:22 ./opt/codesys/bin/
-rwxr-xr-x root/root   7330296 2017-12-18 09:22 ./opt/codesys/bin/codesyscontrol.bin
drwxr-xr-x root/root         0 2017-12-18 09:22 ./opt/codesys/scripts/
```

Содержимое .deb-файла

Основными элементами .deb-файла являются файл конфигурации и исполняемый файл.

Файл конфигурации

Файл конфигурации содержит огромное количество различных параметров конфигурации для CODESYS Control. По содержимому этого файла можно сделать следующие выводы:

- CODESYS Control for Raspberry Pi способен работать в качестве веб-сервера;
- CODESYS Control for Raspberry Pi использует OpenSSL;
- Есть параметры логирования для компонента CmpLog;
- Параметры компонента CmpSettings могут ссылаться на другие файлы;
- Для компонента SysProcess есть ключ Command.%d, значение параметра которого совпадает с именем системной утилиты shutdown в ОС Linux.

```
01: # cat etc/CODESYSControl_User.cfg
02: [SysCom]
03: ;Linux.Devicefile=/dev/ttyS
04:
05: [CmpBlkDrvCom]
06: ;Com.0.Name=MyCom
07: ;Com.0.Baudrate=115200
08: ;Com.0.Port=3
09: ;Com.0.EnableAutoAddressing=1
10:
11: [SysProcess]
12: Command.0=shutdown
13:
14: [CmpApp]
15: Bootproject.RetainMismatch.Init=1
16:
17: root@raspberrypi:/home/pi/tt #

01: # cat etc/CODESYSControl.cfg
02: [SysFile]
03: FilePath.1=/etc/, 3S.dat
04: PlcLogicPrefix=0
05:
06: [CmpLog]
07: Logger.0.Name=/tmp/codesyscontrol.log
08: Logger.0.Filter=0x0000000F
09: Logger.0.Enable=1
10: Logger.0.MaxEntries=1000
11: Logger.0.MaxFileSize=1000000
12: Logger.0.MaxFiles=1
13: Logger.0.Backend.0.ClassId=0x00000104 ;writes logger messages in a file
14: Logger.0.Type=0x314 ;Set the timestamp to RTC
15:
16: [CmpSettings]
17: FileReference.0=SysFileMap.cfg, SysFileMap
18: FileReference.1=/etc/CODESYSControl_User.cfg
19:
20: [SysExcept]
21: Linux.DisableFpuOverflowException=1
22: Linux.DisableFpuUnderflowException=1
23: Linux.DisableFpuInvalidOperationException=1
24:
25: [CmpBACnet]
26: IniFile=bacstacd.ini
27:
28: [CmpWebServer]
29: ConnectionType=0
30:
31: [CmpOpenSSL]
32: WebServer.Cert=server.cer
33: WebServer.PrivateKey=server.key
34: WebServer.CipherList=HIGH
35:
36: [SysMem]
37: Linux.Memlock=0
38:
39: [CmpCodeMeter]
40: InitLicenseFile.0=3SLicense.wbb
41:
42: [SysEthernet]
43: Linux.ProtocolFilter=3
44:
45: [CmpSchedule]
46: ProcessorLoad.Enable=1
47: ProcessorLoad.Maximum=95
48: ProcessorLoad.Interval=5000
```

Содержимое файла конфигурации CODESYS Control For Raspberry Pi v3.5.14.10

Исполняемый файл

Параметры защиты файла

Первичный анализ исполняемых файлов, как правило, включает в себя проверку опций компиляции на предмет установки параметров защиты. Инструмент checksec показывает эти значения параметров защиты для исполняемых файлов.

```
# ./checksec.sh/checksec -o csv -f codesyscontrol_armv6l_raspberry.bin
No RELRO,No Canary found,NX disabled,No PIE,RPATH,No RUNPATH,No SYMTABLES,No
Fortify,0,23,codesyscontrol_armv6l_raspberry.bin

# ./checksec.sh/checksec -o csv -f codesyscontrol_armv7l_raspberry.bin
No RELRO,No Canary found,NX disabled,No PIE,RPATH,No RUNPATH,No SYMTABLES,No
Foritfy,0,23,codesyscontrol_armv7l_raspberry.bin
```

Результат обработки утилитой checksec исполняемых файлов CODESYS Control for Raspberry Pi v3.5.14.10

По результатам работы утилиты видно, что исполняемые файлы CODESYS Control for Raspberry Pi v3.5.14.10 были скомпилированы без дополнительной защиты, которая могла бы усложнить эксплуатацию бинарных уязвимостей.

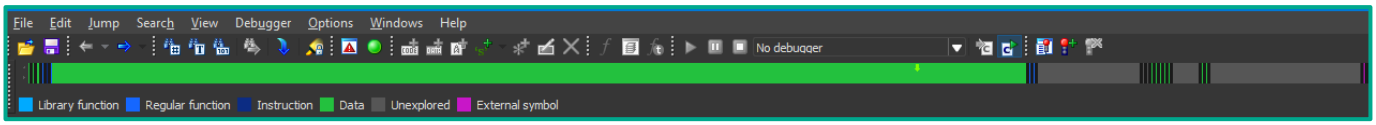
Чуть лучше ситуация с компиляцией файла CODESYS Control for Linux SL, потому что в этом файле содержится установленная опция PIE.

```
# ./checksec.sh/checksec -o csv -f codesyscontrol.bin
Partial RELRO,No Canary found,NX disabled,PIE enabled,No RPATH,No RUNPATH,No
SYMTABLES,No Fortify,0,23,codesyscontrol.bin
```

Результат обработки утилитой checksec исполняемых файлов CODESYS Control for Linux SL v3.5.14.10

Состояние исполняемого файла

Статический анализ исполняемого файла с помощью инструмента IDA Pro показывает, что этот файл на 99% состоит из данных (зеленый цвет), а не из машинного кода:



Состояние запокованного исполняемого файла CODESYS Runtime For Raspberry Pi

Обычно такое состояние характерно для исполняемых файлов, машинный код которых был запакован. Тем не менее, для всех исполняемых файлов обязательным атрибутом является наличие точки входа (Entry point). Для исполняемого файла CODESYS Runtime for Raspberry Pi точкой входа является функция start. С этой функции можно начать анализ.

```
Удалено по требованию вендора
Удалено по требованию вендора
Удалено по требованию вендора
Удалено по требованию вендора
Удалено по требованию вендора
Удалено по требованию вендора
Удалено по требованию вендора
Удалено по требованию вендора
Удалено по требованию вендора
```

```
Удалено по требованию вендора
Удалено по требованию вендора
Удалено по требованию вендора
Удалено по требованию вендора
Удалено по требованию вендора
Удалено по требованию вендора
Удалено по требованию вендора
Удалено по требованию вендора
Удалено по требованию вендора
Удалено по требованию вендора
Удалено по требованию вендора
Удалено по требованию вендора
Удалено по требованию вендора
```

Ассемблированный код функции start

Код функции **start** нормально распознается, и IDA Pro подсказывает, что функция **sub_86a0840**, она же функция **main**, также содержит валидный программный код.

```
1: Удалено по требованию вендора
2: {
3:   Удалено по требованию вендора
4:   Удалено по требованию вендора
5:   Удалено по требованию вендора
6:   Удалено по требованию вендора
7:   Удалено по требованию вендора
8: }
```

Декомпилированный псевдокод функции sub_86a0840

Функция **main** записывает в глобальные переменные количество аргументов командной строки и указатель на их значения (строки 3:4). Затем она вызывает функцию **mprotect** (строка 5), которая изменяет параметры доступа к области памяти. Первый аргумент – указатель на функцию **start** или же начало сегмента .text. Второй аргумент – размер памяти, параметр доступа к которой будет изменен. Размер памяти также должен указывать на конец сегмента. Последний аргумент — это параметры доступа к памяти, на которые меняются исходные параметры. Его значение равно 7, то есть суммарному значению параметров PROT_READ | PROT_WRITE | PROT_EXEC.

Иначе говоря, строка 5 готовит область памяти к распаковке и возможности выполнения программного кода в этом участке памяти. Затем вызывается следующая функция (строка 6) и в качестве аргумента передается указатель на участок памяти, которым является переменная **dword_86A0460,** – это указатель на оригинальную функцию main после распаковки.

Таким образом, для дальнейшего исследования файла необходимо его распаковать.

Запущенный процесс

CODESYS Runtime for Raspberry Pi и for Linux выполняет трассировку своего процесса, то есть CODESYS Runtime for Raspberry Pi и For Linux занимается отладкой самого себя. Данный механизм используется в двух случаях: для перехвата системных вызовов (syscalls) и для реализации простейшей защиты от отладки: к запущенному процессу CODESYS Runtime нельзя присоединиться, используя сторонние инструменты отладки, такие как gdb, IDA Pro, radare или же strace.

Трассировка

```
01: # strace -f ./codesyscontrol.bin
02: execve("./codesyscontrol.bin", ["./codesyscontrol.bin"], [/* 18 vars */]) = 0
03: brk(NULL)                               = 0x90ce000
04: uname({sysname="Linux", nodename="raspberrypi", ...}) = 0
```

```
05: [...]
06: mprotect(0x8050000, 6495192, PROT_READ|PROT_WRITE|PROT_EXEC) = 0
07: cacheflush(0x8050000, 0x8681bd8, 0, 0x8681bd8, 0x8681828) = 0
08: open("/home/pi/", O_RDONLY)                 = 3
09: rt_sigaction(SIGTERM, {sa_handler=0x8050a40, sa_mask=[], sa_flags=SA_RESTORER,
sa_restorer=0x76d436b0}, NULL, 8) = 0
10: rt_sigaction(SIGINT, {sa_handler=0x8050a40, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x76d436b0},
NULL, 8) = 0
11: rt_sigaction(SIGPIPE, {sa_handler=SIG_IGN, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x76d436b0},
NULL, 8) = 0
12: rt_sigaction(SIGABRT, {sa_handler=SIG_IGN, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x76d436b0},
NULL, 8) = 0
13: clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x76faabf8) = 4290
14: strace: Process 4290 attached
15: [pid  4290] set_robust_list(0x76faac00, 12) = 0
16: [pid  4290] getppid( <unfinished ...>
17: [pid  4289] ptrace(PTRACE_CONT, 4290, NULL, SIG_0 <unfinished ...>
18: [pid  4290] <... getppid resumed> )      = 4289
19: [pid  4290] getsid(4289)                 = 3663
20: [pid  4290] ptrace(PTRACE_TRACEME)       = -1 EPERM (Operation not permitted)
21: [pid  4290] getpid()                     = 4290
22: [pid  4290] kill(4290, SIGKILL)          = ?
23: [pid  4289] <... ptrace resumed> )       = -1 ESRCH (No such process)
24: [pid  4289] wait4(-1,  <unfinished ...>
25: [pid  4290] +++ killed by SIGKILL +++
26: <... wait4 resumed> [{WIFSIGNALED(s) && WTERMSIG(s) == SIGKILL}], 0, NULL) = 4290
27: --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_KILLED, si_pid=4290, si_uid=0, si_status=SIGKILL,
si_utime=0, si_stime=1} ---
28: ptrace(PTRACE_CONT, 4290, NULL, SIG_0)  = -1 ESRCH (No such process)
29: getpid()                                 = 4289
30: kill(4289, SIGKILL)                      = ?
31: +++ killed by SIGKILL +++
32: Killed
33:
```

Запуск утилиты strace с ключом –f на исполняемый файл CODESYS Control For Raspberry Pi v3.5.14.00

Из лога запуска утилиты strace с ключом -f видно, что CODESYS Runtime меняет параметры доступа к памяти (строка 06), что, как известно из предыдущего пункта, нужно для распаковки программного кода.

Далее syscall clone создает дочерний процесс (строка 13). Родительский процесс имеет идентификатор 4289. Созданному дочернему процессу присвоен идентификатор 4290. Из-за наличия ключа -f strace пытается трассировать дочерние процессы, поэтому на строке 14 происходит оповещение о подключении к созданному дочернему процессу.

Далее родительский процесс пытается возобновить работу остановленного дочернего процесса вызовом функции ptrace с аргументом PTRACE_CONT (строка 17). Дочерний процесс в это время выполняет ptrace с аргументом PTRACE_TRACEME (строка 20), обозначив этим, что этот процесс должен быть трассирован родительским процессом.

Однако, результат выполнения функции указывает на то, что процесс не может быть трассирован родительским процессом. Из-за этого дочерний процесс завершает свою работу (строки 21:22). После этого родительский процесс получает ответ от функции ptrace (строка 23) и узнает, что дочернего процесса уже нет в системе. Далее родительский процесс пытается еще раз обратиться к дочернему процессу (строка 28) и, снова не обнаружив его, завершает свою работу (строки 29:30).

На этом утилита strace завершает свою работу.

Отладка

Аналогичная ситуация возникнет, если попробовать запустить исполняемый файл под отладчиком gdb.

```
01: # gdb ./codesyscontrol.bin
02: GNU gdb (Raspbian 7.12-6) 7.12.0.20161007-git
03: Copyright (C) 2016 Free Software Foundation, Inc.
04: License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
05: This is free software: you are free to change and redistribute it.
06: There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
07: and "show warranty" for details.
08: This GDB was configured as "arm-linux-gnueabihf".
09: Type "show configuration" for configuration details.
10: For bug reporting instructions, please see:
11: <http://www.gnu.org/software/gdb/bugs/>.
12: Find the GDB manual and other documentation resources online at:
13: <http://www.gnu.org/software/gdb/documentation/>.
14: For help, type "help".
15: Type "apropos word" to search for commands related to "word"...
16: Reading symbols from ./codesyscontrol.bin...(no debugging symbols found)...done.
17: (gdb) set follow-fork-mode child
18: (gdb) run
19: Starting program: /home/pi/ggasss/codesyscontrol.bin
20: [Thread debugging using libthread_db enabled]
21: Using host libthread_db library "/lib/arm-linux-gnueabihf/libthread_db.so.1".
22: [New process 5379]
23: [Thread debugging using libthread_db enabled]
24: Using host libthread_db library "/lib/arm-linux-gnueabihf/libthread_db.so.1".
25:
26: Program terminated with signal SIGKILL, Killed.
27: The program no longer exists.
```

Запуск отладчика gdb на исполняемый файл CODESYS Control For Raspberry Pi v3.5.14.00

Для отладки дочернего процесса для gdb необходимо выставить соответствующий режим set follow-fork-mode child (строка 17). После этого запустить CODESYS Runtime (строка 18). Далее создается дочерний процесс (строка 22), и спустя некоторое время программа самозавершается (строки 26:27).

Таким образом, для исследования исполняемого файла после распаковки его нужно привести в состояние, способное к отладке.

Необходимо сказать, что существует возможность сэмулировать успешно созданный процесс трассировки файла, указав в качестве переменной окружения LD_PRELOAD библиотеку, которая будет содержать функции fork, ptrace, getppid и getsid. Однако на этом этапе это не будет иметь особой эффективности.

Потоки

CODESYS Runtime – многопоточное приложение. Помимо того, что запущенный процесс клонируется и выполняет трассировку дочернего процесса, дочерний процесс порождает огромное количество потоков. Системные утилиты Linux ps и htop получают список потоков, созданных процессом.

```
01: # ps aux | grep -i codesyscontrol
02: root      5404 10.0  0.7  11184  7448 pts/0    S    04:25   0:02 ./codesyscontrol.bin
03: root      5405  5.6  1.3  14852 13172 pts/0    SLl  04:25   0:01 ./codesyscontrol.bin
04: root      5419  0.0  0.0   4372   540 pts/0    S+   04:25   0:00 grep --color=auto -i codesyscontrol
05:
06: # htop -p 5405
07:
08:   PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
09:  5405 root       20   0 14852 13404  2684 S  4.7  1.4  0:54.62 │  └─ ./codesyscontrol.bin
10:  5416 root       20   0 14852 13404  2684 S  0.0  1.4  0:00.32 │     ├─ BlkDrvTcp
```

```
11:  5415 root      20   0 14852 13404  2684 S  0.0  1.4  0:00.37 |     ├─ BlkDrvUdp
12:  5414 root      20   0 14852 13404  2684 S  0.0  1.4  0:00.00 |     ├─ GwCommDrvTcp
13:  5413 root      20   0 14852 13404  2684 S  0.7  1.4  0:01.07 |     ├─ OPCUAServer
14:  5412 root      20   0 14852 13404  2684 S  0.7  1.4  0:00.19 |     ├─ WebServerCloseC
15:  5411 root     -70   0 14852 13404  2684 S  0.0  1.4  0:00.00 |     ├─ CAAEventTask
16:  5410 root     -95   0 14852 13404  2684 S  2.7  1.4  0:29.29 |     ├─ Schedule
17:  5409 root     -69   0 14852 13404  2684 S  0.0  1.4  0:00.00 |     ├─ SchedException
18:  5408 root      20   0 14852 13404  2684 S  1.3  1.4  0:11.77 |     └─ SchedProcessorL
```

Получение списка потоков, порожденных процессом исполняемого файла CODESYS Control for Raspberry Pi v3.5.14.00

По результату выполнения утилиты ps и фильтрации вывода утилитой grep видно, что дочерний процесс имеет идентификатор 5405 (строка 03).

Результат выполнения команды htop для процесса 5405 (строка 06) выводит потоки, созданные дочерним процессом (строки 09:18).

Из имен потоков частично узнаются имена компонентов группы коммуникации (компонент BlkDrvTcp, компонент BlkDrvUdp), имя индустриального протокола OPC UA.

Сетевые коммуникации

По результатам работы утилиты netstat, CODESYS Runtime слушает следующие порты:

```
1: # netstat -ntupl | grep -i codesys
2: tcp        0      0 0.0.0.0:11740           0.0.0.0:*               LISTEN      5405/./codesyscontr
3: tcp        0      0 0.0.0.0:1217            0.0.0.0:*               LISTEN      5405/./codesyscontr
4: tcp        0      0 127.0.0.1:4840          0.0.0.0:*               LISTEN      5405/./codesyscontr
5: tcp        0      0 192.168.0.92:4840       0.0.0.0:*               LISTEN      5405/./codesyscontr
6: udp        0      0 192.168.0.255:1740      0.0.0.0:*                           5405/./codesyscontr
7: udp        0      0 192.168.0.92:1740       0.0.0.0:*                           5405/./codesyscontr
```

Список прослушивающих портов, открытых процессом исполняемого файла CODESYS Control for Raspberry Pi v3.5.14.00

CODESYS Runtime слушает как TCP, так и UDP порты. TCP порт 11740 (строка 2) используется для TCP-связи между CODESYS Runtime и CODESYS Development System.

UDP порт 1740 (строка 7) используется для этой же цели, только общение осуществляется по UDP-протоколу.

Также CODESYS Runtime слушает UDP порт 1740 на широковещательном адресе (строка 6). Широковещательные адреса на стороне клиентов обычно слушаются для возможности обнаружения этих клиентов серверами, т.е. в качестве discovery service. TCP порт 4840 (строки 4:5) используется в качестве discovery service OPC UA.

Информация в открытых источниках

Поиск информации в открытых источниках – неотъемлемая часть исследовательской работы. Нами было найдено:

- *Удалено по требованию вендора*. Содержит технический обзор архитектуры CODESYS Control.
- *Удалено по требованию вендора*. Содержит много полезной информации для статического анализа, например, аргументы и назначения функций.
- Руководство по адаптации CODESYS Control (*Удалено по требованию вендора*). Описывает базовый подход по портированию CODESYS Runtime на устройство без ОС.

К сожалению, вся обнаруженная документации датируется концом 2015 года. Однако её изучение было необходимо: несмотря на устаревшие версии документов, они дали множество подсказок на вопросы, которые возникали во время исследования протокола общения между CODESYS Development System и CODESYS Runtime.

Исследование стека протоколов CODESYS PDU

Эта глава посвящена исследованию стека протоколов CODESYS PDU (Packet Data Unit). Этот стек протоколов используется для коммуникации между узлами сети CODESYS, в том числе CODESYS Development System и CODESYS Runtime.

Стек протоколов CODESYS PDU базируется на модели ISO/OSI. Как и в модели ISO/OSI, каждый слой в протоколе CODESYS PDU отвечает за свою область деятельности. Для полного понимания работы протокола CODESYS PDU необходимо детально разобрать каждый из его слоев.

Примечание: ввиду того, что исследование стека протоколов CODESYS PDU проходило методом «черного ящика», большинство названий полей и уровней основаны на их назначении. Из-за этого используемые названия в дальнейшем описании могут расходиться с теми, что используются в публичных документациях или были даны другими исследователями.

Так, например, в одном из найденных документов для различных уровней протокола CODESYS PDU используются названия:

- Для первого уровня: “datagram layer”, “Layer 2” или “block driver” (далее используется уровень Block Driver (Block Driver Layer))
- Для второго уровня: “network layer”, “Layer 3” или “router” (далее используется уровень датаграм (datagram layer))
- Для третьего уровня: “protocol layer”, “Layer 4” или “channel management” (далее используется канальный уровень (channel layer))
- Для четвертого уровня: “application layer”, “layer 7” или “application services” (далее используется уровень служб (services layer))

Базовое описание протокола

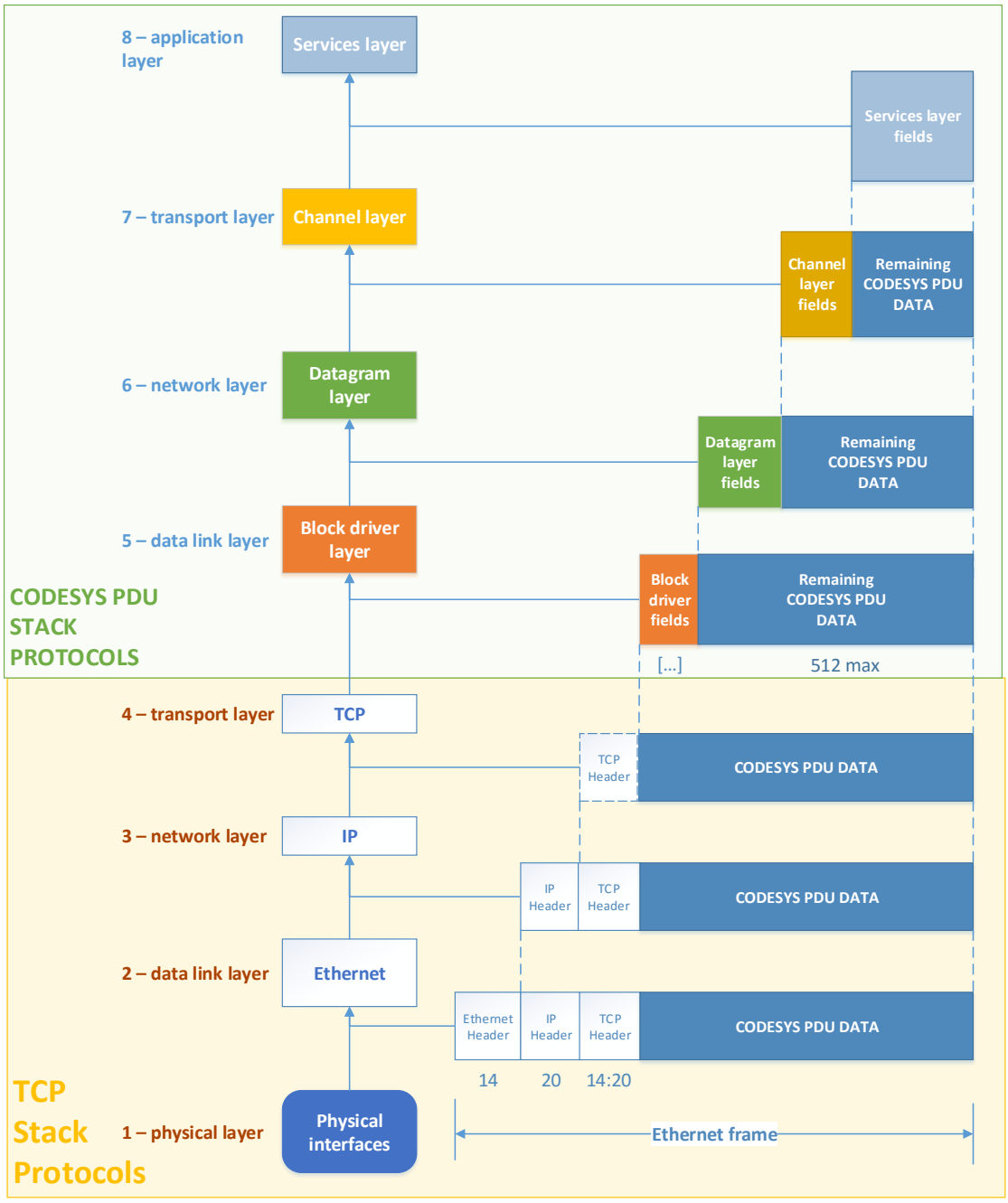
CODESYS PDU (Packet Data Unit) – это стек протоколов, состоящий из четырех различных уровней:

- уровень Block Driver (Block Driver Layer);
- уровень датаграм (datagram layer);
- канальный уровень (channel layer);
- уровень служб (services layer);

Порядок байт в этом стеке протоколов – little endian, но при необходимости может быть изменен на big endian. Принцип работы – синхронный или асинхронный в зависимости от уровня протокола.

Использование протокола CODESYS PDU не ограничивается сетевой коммуникацией. Он используется также для коммуникации по USB, Can–шине и последовательным портам. При этом, среда исполнения CODESYS Runtime всегда используют возможности ОС, под которую она была адаптирована. Таким образом, конечный информационный пакет будет содержать сформированные данные CODESYS Runtime и данные, сформированными драйверами ОС под определенный физический интерфейс.

Так, например, конечный CODESYS PDU пакет, отправленный через сетевой интерфейс по TCP, будет содержать сразу два стека протокола – TCP и CODESYS PDU.

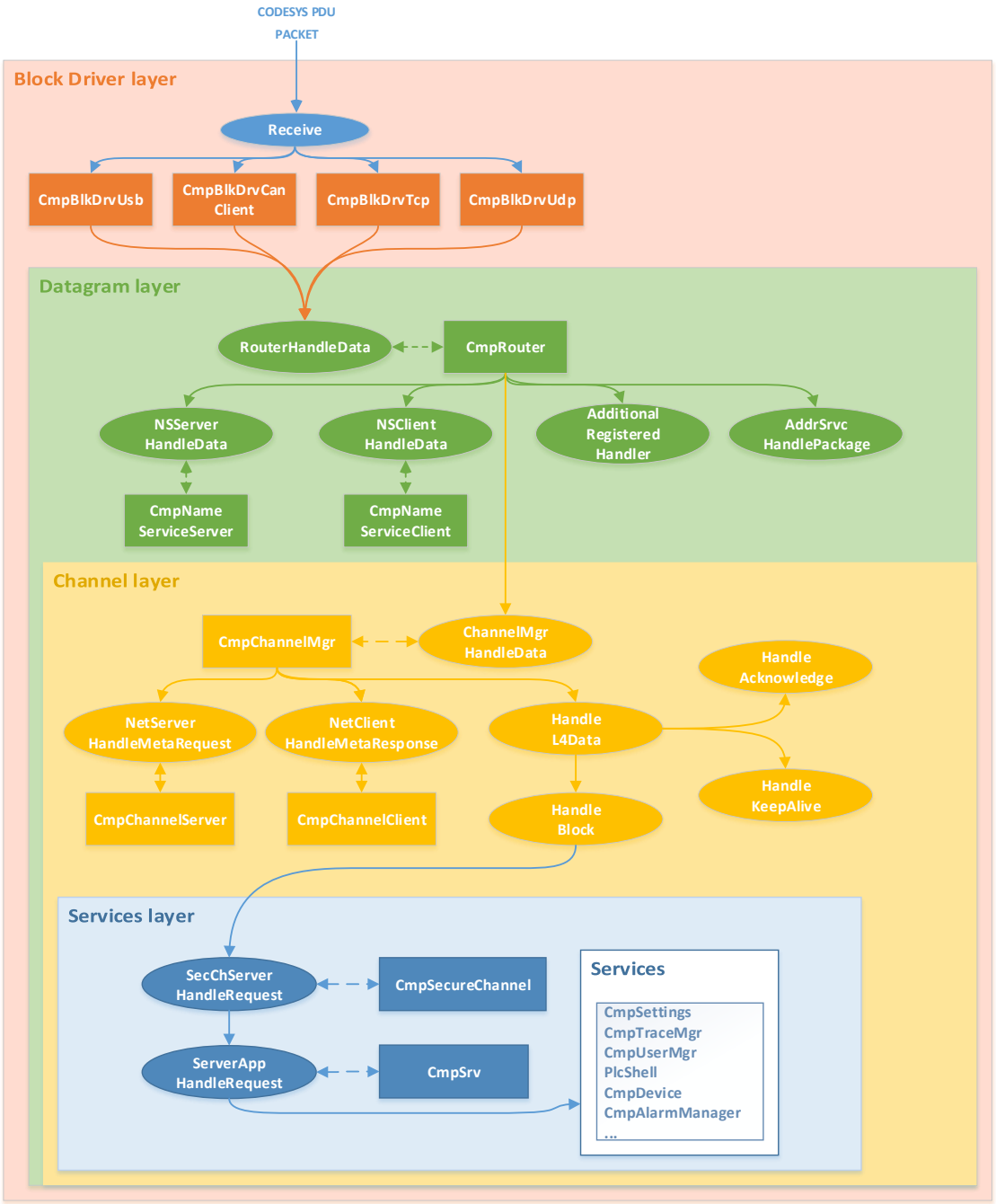


Пример использования стеков протоколов TCP и CODESYS PDU в одном пакете

Способность общения по физическим интерфейсам реализуется компонентами группы Communication – Block Drivers. Помимо этого, любой разработчик может разработать свой Block Driver и использовать протокол CODESYS PDU в нем.

Данный протокол основан на модели ISO/OSI. Из этой модели CODESYS PDU полностью исключил физический уровень, а сессионный уровень и уровень представлений были объединены с уровнем приложений. Обработку каждого из уровней берет на себя один компонент или несколько компонентов из одной группы.

Ниже – схематичное представление того, как компоненты участвуют в разборе входящего пакета, сформированного по протоколу CODESYS PDU.



Схематичное представление разбора компонентами пакета CODESYS PDU

Совокупная работа этих компонентов определяет мощность стека протокола CODESYS PDU.

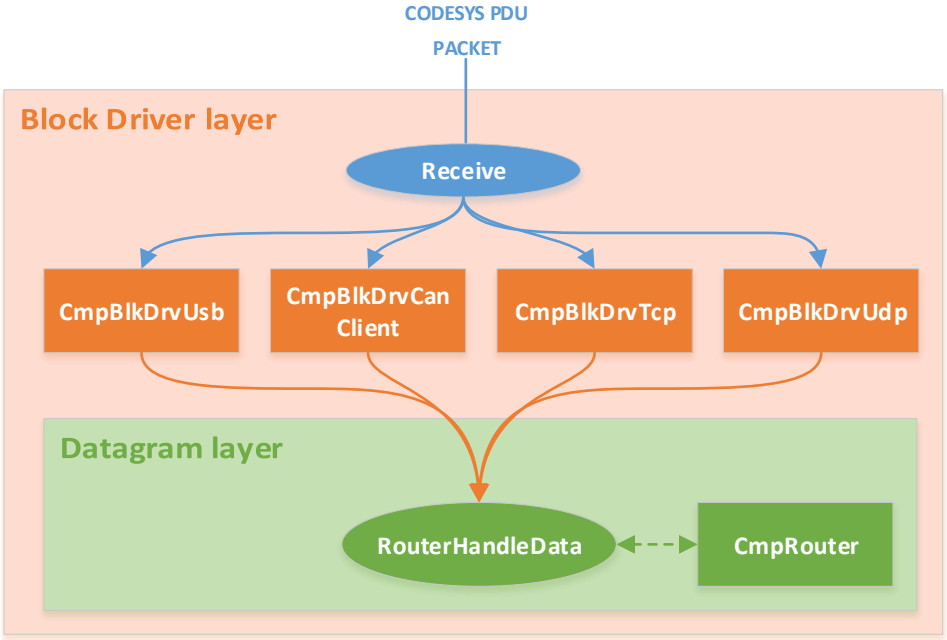
Далее мы рассмотрим каждый из уровней в стеке протокола CODESYS PDU.

Разбор стека протоколов

Уровень Block driver (Block driver layer)

Вся полезная работа CODESYS Runtime – это совокупная работа его компонентов. Компоненты могут расширять возможности друг друга. В том числе, это работает для компонентов, которые разбирают полученный пакет, сформированный по протоколу PDU.

Основная задача компонентов из группы Block Drivers – создание возможности коммуникации через физический или программный интерфейс. Любой компонент Block Drivers является «входной точкой» для получения информационного пакета и точкой его отправки. Поэтому эти компоненты перед отправкой пакета могут добавить дополнительные поля в протоколе.

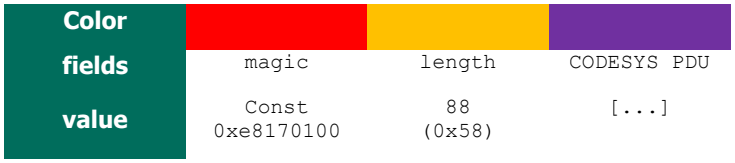


Схематичное представление разбора компонентами пакета CODESYS PDU на уровне Block Driver (Block Driver Layer)

Так работает, например, компонент Block Driver **CmpBlkDrvTcp**. Этот компонент реализует коммуникацию по TCP-протоколу. В каждом сообщении **CmpBlkDrvTcp** добавляет два поля, каждый из которых является 4-байтовым числом:

```
> Transmission Control Protocol, Src Port: 49171, Dst Port: 11740, Seq: 93, Ack
> CoDeSys V3 Protocol
```

```
0000  08 00 27 a5 f2 66 08 00  27 90 85 bf 08 00 45 00   ··'··f·· '·····E·
0010  00 80 00 cc 40 00 80 06  77 e2 c0 a8 00 21 c0 a8   ····@··· w····!··
0020  00 58 c0 13 2d dc 99 0c  36 6a 9d 4a 13 5e 50 18   ·X··-··· 6j·J·^P·
0030  3f e1 2c fc 00 00 00 01  17 e8 58 00 00 00 c5 6b   ?·,····· ··X····k
0040  40 40 00 53 2d dc c0 a8  00 58 2d df c0 a8 00 21   @@·S-··· ·X-····!
0050  80 00 00 00 00 00 01 81  10 00 01 00 00 00 00 00   ········ ········
0060  00 00 24 00 00 00 db c5  6d 9e 55 cd 10 00 01 00   ··$····· m·U·····
0070  01 00 00 00 00 00 10 00  00 00 00 00 00 00 01 8c   ········ ········
0080  80 00 06 10 00 00 00 00  00 00 00 00 00 00        ········ ······
```



Пример использования двух дополнительных полей на уровне Block Driver

- **magic** – магическое число. Константное число **0xe8170100** вставляется и проверяется компонентом **CmpBlkDrvTcp** каждый раз, когда компонент получает сетевой пакет.
- **length** – суммарное количество байт в пакете, включая размеры полей **magic** и **length** (оба поля размером по 4 байта).

Ниже представлена трассировка вызова функции **Receive()**, которая принадлежит компоненту **CmpBlkDrvTcp**. Функция **Receive()** обрабатывает поля **magic** и **length**.

```
Call trace:
       Удалено по требованию вендора
       Удалено по требованию вендора
       Удалено по требованию вендора
       Удалено по требованию вендора
       Удалено по требованию вендора

Pseudocode:
001: Удалено по требованию вендора
002: {
[...]
061:              Удалено по требованию вендора
[...]
068:              Удалено по требованию вендора
069:                              Удалено по требованию вендора
070:                              Удалено по требованию вендора
071:                              Удалено по требованию вендора
072:                              Удалено по требованию вендора
073:                              Удалено по требованию вендора
[...]
094:              Удалено по требованию вендора
095:              Удалено по требованию вендора
096:              {
099:                Удалено по требованию вендора
100:                {
[...]
101:                Удалено по требованию вендора
102:                {
[...]
144:                Удалено по требованию вендора
[...]
148:                Удалено по требованию вендора
149:                              Удалено по требованию вендора
150:                              Удалено по требованию вендора
151:                              Удалено по требованию вендора
152:                              Удалено по требованию вендора
153:                              Удалено по требованию вендора
[...]
179:                    Удалено по требованию вендора
180:                    Удалено по требованию вендора
[...]

196:       Удалено по требованию вендора
[...]
206: }
```

Декомпилированный псевдокод функции Receive компонента CmpBlkDrvTcp

Получение всех данных из сети для дальнейшей обработки компонентом **CmpBlkDrvTcp** происходит в два шага:

1. На первом шаге компонент достает первые 8 байт (строка 068) из полученных по сети данных через функцию **SysSockRecv,** которая была экспортирована системным компонентом **SysSocket.** Максимальное количество байт, которые можно получить, передается в третьем аргументе функции **SysSockRecv**. Далее первые 4 байта сравниваются с магической константой (строка 099). Вторые 4 байта сравниваются с числом 520. Число 520 было получено путем сложения максимально возможного размера пакета, сформированного по протоколу CODESYS PDU (512 байт), и суммарного размера полей **magic** и **length** (8 байт).
2. На втором шаге извлекаются оставшиеся данные. Ожидается, что размер данных будет равен разнице значения поля **length** и суммарного размера полей **magic** и **length** (строка 144).

Далее компонент **CmpBlkDrvTcp** передает управление функции **RouterHandleData** (строка 196), которая зарегистрирована компонентом **CmpRouter**, – на уровень датаграм (datagram layer).

Стоит учесть, что в случае коммуникации по UDP протоколу поля **magic** и **length** будут отсутствовать.

```
> User Datagram Protocol, Src Port: 1743, Dst Port: 1743
> CoDeSys V3 Protocol
```

```
0000  ff ff ff ff ff ff 08 00  27 90 85 bf 08 00 45 00
0010  00 34 00 9e 00 00 80 11  b7 aa c0 a8 00 21 c0 a8
0020  00 ff 06 cf 06 cf 00 20  2f ac c5 74 40 03 00 30
0030  26 6e 03 21 80 00 00 00  00 00 02 c2 00 04 8d f5
0040  00 00
```

Color	
fields	CODESYS PDU
value	[...]

Пример отсутствия дополнительных полей на уровне Block Driver

Функция **UdpReceiveBlock()** компонента **CmpBlkDrvUdp** аналогична функции **Receive()** компонента **CmpBlkDrvTcp**.

```
Call trace:
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора

Pseudocode:
001: Удалено по требованию вендора
002: {
019:
[...]
047:            Удалено по требованию вендора
[...]
055:      Удалено по требованию вендора
056:      {
[...]
064:            Удалено по требованию вендора
[...]
071:      }
072:      Удалено по требованию вендора
073:      {
[...]
080:          Удалено по требованию вендора
```

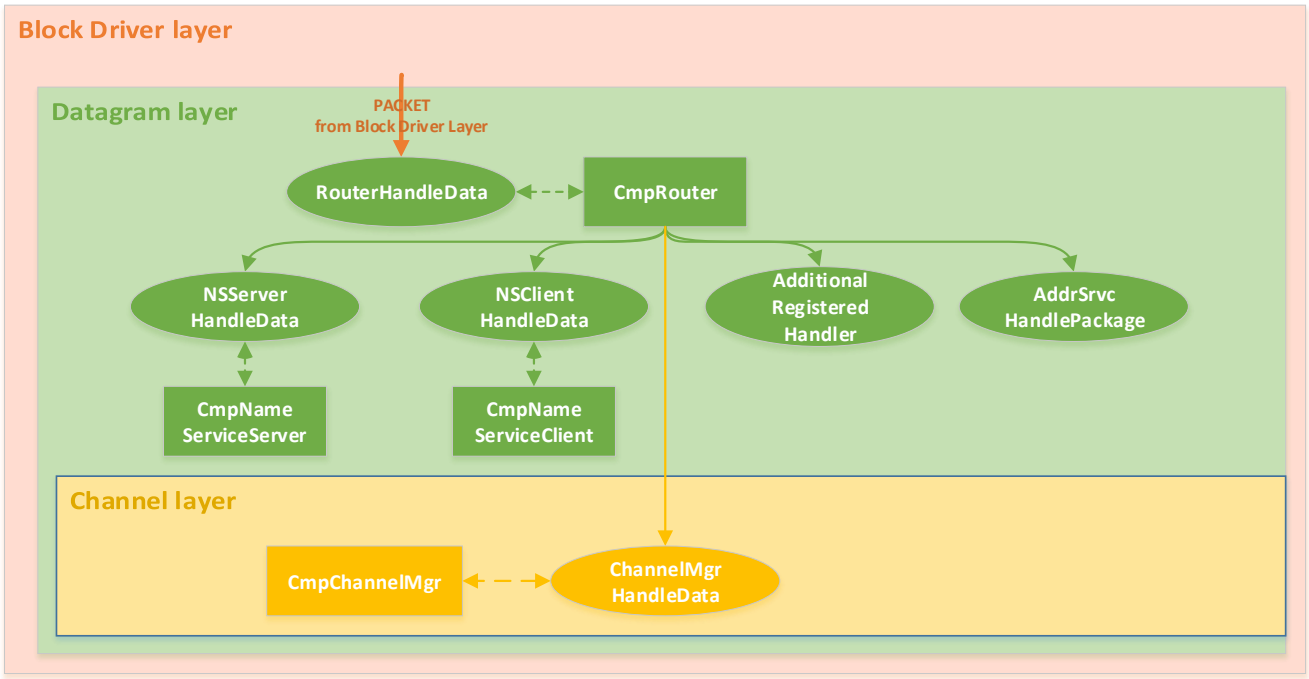
```
081:          Удалено по требованию вендора
083:      }
084:    }
[...]
111:      Удалено по требованию вендора
[...]
```

Декомпилированный псевдокод функции UdpReceiveBlock компонента CmpBlkDrvUdp

Функция **UdpReceiveBlock()** не выполняет каких-либо проверок полученных данных. Помимо этого, у компонента **CmpBlkDrvUdp** есть другая особенность. Она заключается в том, что функция **UdpReceiveBlock()** слушает широковещательные сообщения (строка 047). Если данных не оказалось, то компонент пытается считать отправленные конкретно ему данные (строка 64). Если в одном из случаев данные были, то компонент **CmpBlDrvUdp** вызывает функцию **RouterHandleData** для дальнейшей обработки (строка 111).

Уровень датаграм (Datagram layer)

Уровень датаграм – следующий уровень в стеке протоколов CODESYS PDU. Основное назначение этого уровня – маршрутизация пакетов, обнаружение узлов в сети CODESYS и передача данных на следующий уровень. Основным компонентом на этом уровне является **CmpRouter**. Вспомогательными – компоненты **CmpNameServiceClient** и **CmpNameServiceServer**.



Схематичное представление разбора компонентами пакета CODESYS PDU на уровне датаграм (Datagram layer)

Компоненты группы Block Drivers обязаны вызвать функцию **RouterHandleData,** которая действует на уровне датаграм. В аргументах вызова функции компоненты передают полученные данные.

```
> Transmission Control Protocol, Src Port: 49171, Dst Port: 11740, Seq: 93, Ack
> CoDeSys V3 Protocol
```

```
0000  08 00 27 a5 f2 66 08 00  27 90 85 bf 08 00 45 00   ··'··f·· '·····E·
0010  00 80 00 cc 40 00 80 06  77 e2 c0 a8 00 21 c0 a8   ····@··· w····!··
0020  00 58 c0 13 2d dc 99 0c  36 6a 9d 4a 13 5e 50 18   ·X··-··· 6j·J·^P·
0030  3f e1 2c fc 00 00 00 01  17 e8 58 00 00 00 c5 6b   ?·,····· ··X····k
0040  40 40 00 53 2d dc c0 a8  00 58 2d df c0 a8 00 21   @@·S-··· ·X-····!
0050  80 00 00 00 00 00 01 81  10 00 01 00 00 00 00 00   ········ ········
0060  00 00 24 00 00 00 db c5  6d 9e 55 cd 10 00 01 00   ··$····· m·U·····
0070  01 00 00 00 00 00 10 00  00 00 00 00 00 00 01 8c   ········ ········
0080  80 00 06 10 00 00 00 00  00 00 00 00 00 00 00      ········ ·····
```

Color								
fields	Blk_driver_fields (CmpBlkDrvTcp)	magic	hop_info_byte		packet_info			
			hop_count (5 bit)	header_length (3 bit)	priority (2 bit)	signal (1 bit)	type_address (1 bit)	length_data_block (4 bit)
value	[...]	197 (0xc5)	0xd	0x3	0x1 (NORMAL)	0x0 (NO_SIGNAL)	0x0 (DIRECT)	0x0

Color								
fields	service_id	message_id	lengths		sender		Receiver	
			receiver_length	sender_length	port	address	port	Address
value	0x40	0x00	0x5	0x3	11740 (2ddc)	192.168.0.88 (c0a80058)	11743 (2ddf)	192.168.0.33 (c0a80021) 800000

Color		
Fields	Padding	Remaining data
value	0x0000	[...]

Используемые поля на уровне датаграм (datagram layer)

С точки зрения трафика эта функция обрабатывает следующие поля и данные:

- **magic** – магическое число пакета, сформированного по протоколу CODESYS PDU. Размер этого поля – один байт, он вставляется компонентом **CmpRouter**.
- **hop_info** – битовая структура, которая состоит из двух полей: 5-битового поля **hop_count** и 3-битового поля **header_length**:

  a. Поле **hop_count** отвечает за возможное количество передач полученного по сети пакета. Каждый раз, когда один узел сети CODESYS получает пакет и перенаправляет его другому узлу сети CODESYS, он декрементирует значение поля **hop_count**. Если узел получил пакет и не является его конечным получателем, при этом значение поля **hop_count** равно 0, то узел сбросит этот пакет. По сути, это поле защищает сеть, в которой есть узлы CODESYS, от бесконечной пересылки пакета;

  b. Поле **header_length** указывает на количество байт до следующего поля с размером данных (**lengths)**. Ожидается, что при сложении значения поля **header_length** с его позицией в пакете получится позиция на поле **lengths.**

- **packet_info** – параметры пакета. Это поле также является битовой структурой:

  a. Первые два бита – это поле **priority**. Оно обозначает приоритетность обрабатываемого пакета. Для обозначения приоритетности используются следующие числовые значения: 0 – низкая (low), 1 – обычная (normal), 2 – высокая (high), 3 – срочная (emergency);
  b. Следующий бит **signal** используется компонентом **CmpRouter** в качестве возвращаемого статуса обработки пакета, в котором можно указать на ошибки;
  c. Поле **type_address** указывает на тип передаваемого адреса. Это поле нужно для понимания компонентом **CmpRouter** содержимого полей **sender** и **receiver**. Есть два значения для поля **type_address**: 0 – полный адрес, 1 – относительный адрес;
  d. Последнее поле **length_data_block** указывает на максимальный размер данных, которые может принять получатель.

- **service_id** – идентификатор службы. Указывает на то, какая именно служба должна обработать полученные данные. CODESYS Runtime содержит и определяет следующие службы:

  a. Служба с идентификатором **1** для запроса и **2** для ответа – служба адресов (address service). Эта служба используется для обнаружения «живых» узлов в сети и для построения самой информационной сети из узлов. Узел в этой сети представляет собой участника с запущенным CODESYS Runtime или CODESYS Development System.
  b. Служба с идентификатором **3** для запроса и **4** для ответа – служба имен (name services). Эта служба используется для получения информации об узле.
  c. Служба с идентификатором **64 (0x40)** как для запроса, так и для ответа – служба канала (channel services). Эта служба используется для обращения к серверу и к менеджеру канала связи.

- **message_id** – идентификатор сообщения. Это значение указывается отправителем и используется для идентификации сообщения. Обычно в качестве идентификатора сообщения **CmpRouter** передает 4-битное значение текущего времени. Таким образом достигается уникальность идентификатора сообщения.
- **lengths** – размеры полей **receiver и sender**. Поле **lengths** – это битовая структура, в которой старшие 4 бита содержат значение, соответствующее половине количества байт в поле **receiver**, а младшие 4 бита – половине количества байт в поле **sender**. То есть, количество байт в поле **receiver** и в поле **sender** будет в два раза больше указанных в поле lengths. Например, для рассматриваемого пакета значение старших 4 битов поля **lengths** (0x53) равно 5. Это значит, что итоговое количество байт для поля **receiver** будет 10.
- **sender** – адрес, которому предназначено сообщение.
- **receiver** – адрес, которому необходимо отправить ответ на сообщение.
- Поле **padding** добавляется в конец пакета. Не является обязательным.

Поля **sender** и **receiver** имеют свой формат данных, который зависит от используемого компонента Block Driver. Например, **CmpBlkDrvTcp** ожидает в этих полях полный сетевой адрес узла и номер сетевого порта. То есть, байты поля receiver (**2ddcc0a80058)** на самом деле содержат порт 11740 (**2ddc**) и адрес получателя 192.168.0.88 (**c0a80058**).

CmpBlkDrvUdp использует другой формат. Вместо полного адреса он использует относительный адрес, а вместо двух байт для значения порта – один. Этот байт порта указывает на индекс порта. **CmpBlkDrvUdp** определяет четыре индекса портов: 0, 1, 2, 3. Каждый из индексов соответствуют одному UDP порту: 0 – 1740, 1 – 1741, 2 – 1742, 3 – 1743. Относительный сетевой адрес – это последний байт в числовом формате физического адреса.

```
> User Datagram Protocol, Src Port: 1743, Dst Port: 1740
> CoDeSys V3 Protocol
```

```
0000  08 00 27 a5 f2 66 08 00  27 90 85 bf 08 00 45 00   ··'··f·· '·····E·
0010  00 64 15 df 40 00 80 11  62 e0 c0 a8 00 21 c0 a8   ·d··@··· b····!··
0020  00 58 06 cf 06 cc 00 50  53 60 c5 6b 40 40 00 31   ·X·····P S`·k@@·1
0030  00 58 03 21 80 00 00 00  00 00 01 81 24 00 01 00   ·X·!···· ····$···
0040  00 00 00 00 00 00 24 00  00 00 13 fd 2b 3a 55 cd   ······$· ····+:U·
0050  10 00 01 00 01 00 00 00  00 00 10 00 00 00 00 00   ········ ········
0060  00 00 01 8c 80 00 06 10  00 00 05 00 00 00 00 0d   ········ ········
0070  05 03                                             ··
```

Color								
fields	lengths		Sender		receiver		padding (optional)	Remaining data
	receiver_ length	sender_ length	port_ index	relative address	port_ index	relative_ address		
value	0x1	0x3	0	88 (0x58)	3	33 (0x21) 800000	0x0000	[...]

Пример содержимого полей Sender и Receiver при использовании компонента CmpBlkDrvUdp

Таким образом, значение байт поля **sender** (**0058**) будут содержать порт 1740 (значение поля **port_index** равно **0x0**) и адрес 192.168.0.88 (**0x58** – последний байт адреса, первые три байта извлекаются из адреса интерфейса). Получатель будет ожидать ответ на порт 1743 (значение поля **port_index** равно **0x3**) и на адрес 192.168.0.33 (**0x21**).

Назначение и формат всех оставшихся данных в пакете зависит от службы (**service_id**), для которой этот пакет предназначен. Если этот идентификатор равен **1**, **2**, **3** или **4**, то обработчиком остается компонент **CmpRouter** или его вспомогательные компоненты **CmpNameServiceClient** и **CmpNameServiceServer**. Если идентификатор равен **64 (0x40)**, то все оставшиеся данные передаются компоненту **CmpChannelManager**.

Основываясь на поле **sender**, компонент **CmpRouter** определяет, нужно ли переслать пакет другому узлу или же пакет предназначен ему. В первом случае компонент **CmpRouter** сначала декрементирует значение **hop_count** в пакете, затем отправляет его как есть на узел, указанный в поле **sender**. Во втором случае компонент **CmpRouter** обрабатывает пакет и возвращает результат на адрес, указанный в поле **receiver**. Обработкой пакета, который был предназначен узлу, занимается функция **HandleLocally**.

```
Call trace:
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора

Pseudocode:
001: Удалено по требованию вендора
002: {
[...]
044:    Удалено по требованию вендора
045:    Удалено по требованию вендора
046:    {
[...]
051:      Удалено по требованию вендора
[...]
060:    }
061:    Удалено по требованию вендора
062:    {
063:      Удалено по требованию вендора
064:      {
[...]
069:        Удалено по требованию вендора
[...]
078:      }
079:      Удалено по требованию вендора
080:      {
[...]
086:        Удалено по требованию вендора
[...]
096:        Удалено по требованию вендора
097:      }
098:    }
099:    Удалено по требованию вендора
100:    {
101:      Удалено по требованию вендора
102:        Удалено по требованию вендора
103:      Удалено по требованию вендора
111:      [...]
119:      Удалено по требованию вендора
120:    }
121:    Удалено по требованию вендора
122:    {
123:      Удалено по требованию вендора
124:      {
125:        Удалено по требованию вендора
126:        Удалено по требованию вендора
127:          Удалено по требованию вендора
[...]
142:      }
143:    }
144:    Удалено по требованию вендора
145: }
```

Декомпилированный псевдокод функции HandleLocally компонента CmpRouter

Функция **HandleLocally** по значению поля **service_id** определяет, какой именно обработчик необходимо вызвать:

- Для значения, равного **1** или **2**, – функция **AddrSrvcHandlePackage** (строка 103). Это обработчик службы адресов (address service).
- Для значения, равного **3,** – функция **NSServerHandleData** (строка 51). Это обработчик службы имен (Name Service), который обрабатывает входящие запросы, то есть работает в качестве сервера.

- Для значения, равного **4**, – функция **NSClientHandleData** (строка 69). Это обработчик службы имен (Name Service), который обрабатывает результаты выполненных запросов, то есть работает в качестве клиента.
- Для значения, равного **0x40**, – функция **ChannelMgrHandleData** (строка 86). Это обработчик службы канала (Channel Service).

Если для полученного **service_id** не было найдено подходящего обработчика, то происходит его поиск среди дополнительных обработчиков, зарегистрированных функцией **RouterRegisterProtocolHandler**. При наличии такового, выполняется его вызов (строки 125:127).

```
01: Удалено по требованию вендора
02: {
[...]
14:   Удалено по требованию вендора
15:   {
16:     Удалено по требованию вендора
17:     {
18:       Удалено по требованию вендора
19:       {
20:         Удалено по требованию вендора
21:       }
22:       Удалено по требованию вендора
23:       {
24:         Удалено по требованию вендора
25:         Удалено по требованию вендора
26:         Удалено по требованию вендора
[...]
39:   Удалено по требованию вендора
40: }
```

Декомпилированный псевдокод функции RouterRegisterProtocolHandler

Функция **RouterRegisterProtocolHandler** в качестве аргументов (строка 1) принимает идентификатор сервиса (**service_id**) и обработчик (**handler**). Эта функция не позволяет зарегистрировать обработчик (строка 16) для следующих идентификаторов: **1**, **2**, **3**, **4** и **64 (0x40)**. В случае если обработчик для данного идентификатора не установлен (строка 18), то обработчик будет добавлен в глобальный словарь обработчиков (**s_protocolHandlers**), где ключ для обработчика – первый аргумент **service_id** (строка 24).

Далее будут рассмотрены обработчики системных служб CODESYS Runtime, а именно обработчики службы адресов (address service) и обработчики службы имен (name service)

Обработчик службы каналов (channel service) рассмотрен в главе «Канальный уровень».

Служба адресов (address service)

Обработчик службы адресов определяет по идентификатору **service_id** две возможные команды: команду «запрос» с идентификатором **service_id** 1 и команду «ответ» с идентификатором **service_id** 2. Обработчиком является функция **AddrSrvcHandlePackage.**

```
Call trace:
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
```

```
        Удалено по требованию вендора

Pseudocode
01: Удалено по требованию вендора
02: {
[...]
23:   Удалено по требованию вендора
24:   {
25:     Удалено по требованию вендора
26:       Удалено по требованию вендора
27:   }
28:   Удалено по требованию вендора
29:   {
30:     Удалено по требованию вендора
31:     Удалено по требованию вендора
32:     {
33:       Удалено по требованию вендора
34:       {
[...]
75:               Удалено по требованию вендора
76:         }
77:       }
78:     }
[...]
98: }
```

Декомпилированный псевдокод функции AddrSrvcHandlePackage

Команда «запрос» с идентификатором 1 (строка 23) отправляется узлом сети CODESYS для оповещения остальных узлов о своем существовании. Этот запрос можно постоянно наблюдать в трафике по широковещательному адресу: он отправляется с одинаковой периодичностью всеми узлами сети CODESYS на широковещательный адрес.

```
9227 971.844759   192.168.0.92   192.168.0.255   CODESYSV3   60 1740 → 1740 Len=8
9228 971.844930   192.168.0.92   192.168.0.255   CODESYSV3   60 1740 → 1741 Len=8
9229 971.844931   192.168.0.92   192.168.0.255   CODESYSV3   60 1740 → 1742 Len=8
9230 971.845001   192.168.0.92   192.168.0.255   CODESYSV3   60 1740 → 1743 Len=8
9268 978.456825   192.168.0.88   192.168.0.255   CODESYSV3   60 1740 → 1740 Len=12
9269 978.456868   192.168.0.88   192.168.0.255   CODESYSV3   60 1740 → 1741 Len=12
9270 978.456868   192.168.0.88   192.168.0.255   CODESYSV3   60 1740 → 1742 Len=12
9271 978.456868   192.168.0.88   192.168.0.255   CODESYSV3   60 1740 → 1743 Len=12
9579 1012.630681  192.168.0.33   192.168.0.255   CODESYSV3   50 1743 → 1740 Len=8
9580 1012.630796  192.168.0.33   192.168.0.255   CODESYSV3   50 1743 → 1741 Len=8
9581 1012.630859  192.168.0.33   192.168.0.255   CODESYSV3   50 1743 → 1742 Len=8
9582 1012.630942  192.168.0.33   192.168.0.255   CODESYSV3   50 1743 → 1743 Len=8
```

Широковещательные оповещения узлов сети CODESYS

Если среди узлов сети CODESYS есть родительский узел, то он узнает о существовании узла, который отправил запрос. Этот же запрос может отправить родительский узел. Если такой запрос получат дочерние узлы, то они оповестят его о своем существовании. Дополнительных полей для этого запроса не используется.

Команда «ответ» с идентификатором 2 (строка 28) обычно отправляется родительским узлом. Эта команда используется для построения информационной сети CODESYS.

Она использует следующие поля:

Fields	size (byte)	Type
version_major	1	Uint8_t
version_minor	1	Uint8_t
Unused	1	Uint8_t
address_len	1	Uint8_t
address	30	Uint8_t[]
subnet_id	4	Uin32_t
subnet_params	1	Uint8_t
parent_subnet_params	1	Uint8_t
Unused или reserved	2	Uint16_t

- **version_major** – поле, которое используется для обозначения версии информационной сети CODESYS, которая будет сформирована. Для протокола CODESYS PDU значение **version_major** всегда равно 1 (стока 75 декомпилированного псевдокода функции **AddrSrvcHandlePackage**).
- **version_minor** – поле, указывающее на используемую версию команды. Она определяет дополнительные поля в команде и в ответе. Например, при её положительном значении в пакете будет использоваться поле **parent_subnet_params**.
- **address_len** – указывает на количество байт поля **parent_address**, которое нужно обработать. Итоговое количество байт умножается на два.
- **address** – адрес родительского узла.
- **subnet_id** – идентификатор сформированной подсети.
- **subnet_params и parent_subnet_params** – параметры текущей подсети и подсети родительского узла.

Узел сети CODESYS, получив такое сообщение и при отсутствии заранее указанного родительского узла, установит в качестве родительского узла источник этого сообщения.

Служба имен (name service)

Обработку сообщений, которые предназначены для службы имен (name services), выполняют вспомогательные компоненты **CmpNameServiceClient** и **CmpNameServiceServer**. Сообщения службы имен (name services) также условно разделяются на команду «запрос» и команду «ответ». Входящие «запросы» от других узлов обрабатывается компонентом **CmpNameServiceServer**. Запросы, которые были отправлены узлом CODESYS Runtime, обрабатываются как «ответы» компонентом **CmpNameServiceClient**.

Общий заголовок сообщений службы имен (**name_service_header**) использует следующие поля:

Fields	size (byte)	Type
Subcmd	2	Uint16_t
Version	2	Uint16_t
Message_id	4	Uint32_t
Message_data	n	Uint8_t[n]

- **subcmd** – идентификатор команды, которую необходимо выполнить службе имен.
- **version** – номер версии команды. Это поле определяет наличие дополнительных полей в поле **message_data**, характерных для указанной версии команды.
- **message_id** – идентификатор сообщения. Это значение возвращается в ответе. Для создания значения этого поля используется числовой идентификатор текущего времени.
- **message_data** – поля команды, формат которых определяет сама команда (**subcmd)**.

Компонент **CmpNameServiceServer** экспортирует функцию **NSServerHandleData.** Функция **NSServerHandleData** выступает в качестве обработчика запросов к службе имен (name service) от других узлов.

```
Call trace:
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора

Pseudocode:
01: Удалено по требованию вендора
02: {
[...]
09:   Удалено по требованию вендора
10:     Удалено по требованию вендора
11:   Удалено по требованию вендора
12:     Удалено по требованию вендора
13:   Удалено по требованию вендора
14:   Удалено по требованию вендора
15:   {
[...]
18:     Удалено по требованию вендора
19:   }
20:   Удалено по требованию вендора
21:   {
[...]
24:     Удалено по требованию вендора
25:   }
26:   Удалено по требованию вендора
27:   {
28:     Удалено по требованию вендора
29:   }
30:   Удалено по требованию вендора
31: }
```

Декомпилированный псевдокод функции NSServerHandleData компонента CmpNameServiceServer

Обработчик **NsServerHandleData** определяет два возможных идентификатора поля **subcmd** и для каждого вызывает соответствующий вспомогательный обработчик:

- Для идентификатора **0xc202** (строка 20) – обработчик **HandleResolveAddrReq** (строка 18). Это запрос для получения информации об узле. Ответ на этот запрос использует значение идентификатора **service_id 4** и будет обработан функцией **NsClientHandleData**.
- Для идентификатора **0xc201** (строка 14) – обработчик **HandleResolveNameReq** (строка 18). Этот запрос аналогичен запросу с идентификатором **0xc202**, с той лишь разницей, что в теле запроса передается имя узла. Если переданное имя узла не совпадает с именем узла, который получил запрос, то узел игнорирует запрос. Ответ на этот запрос использует значение идентификатора **service_id 4** и будет обработан функцией **NsClientHandleData**.

Компонент **CmpNameServiceClient** экспортирует функцию **NSClientHandleData** которая выступает в качестве обработчика полученных от узлов ответов на запросы службы имен (name services). Ответ на запрос использует общий заголовок сообщения (**name_service_header)**. В зависимости от указанного значения в поле **version**, ответ может различаться. Для поля **version,** равного **0x103**, ответ будет содержать следующие поля:

Fields	size (byte)	Type
max_channels	2	Uint16_t
byte_order	1	Uint8_t
Unknown	1	Uint8_t
node_name_length	2	Uint16_t
device_name_length	2	Uint16_t
vendor_name_length	2	Uint16_t
target_type	4	Uint32_t
target_id	4	Uint32_t
target_version	4	Uint32_t
Node_name	node_name_length	Uint8_t[node_name_length]
device_name	device_name_length	Uint8_t[device_name_length]
vendor_name	vendor_name_length	Uint8_t[vendor_name_length]

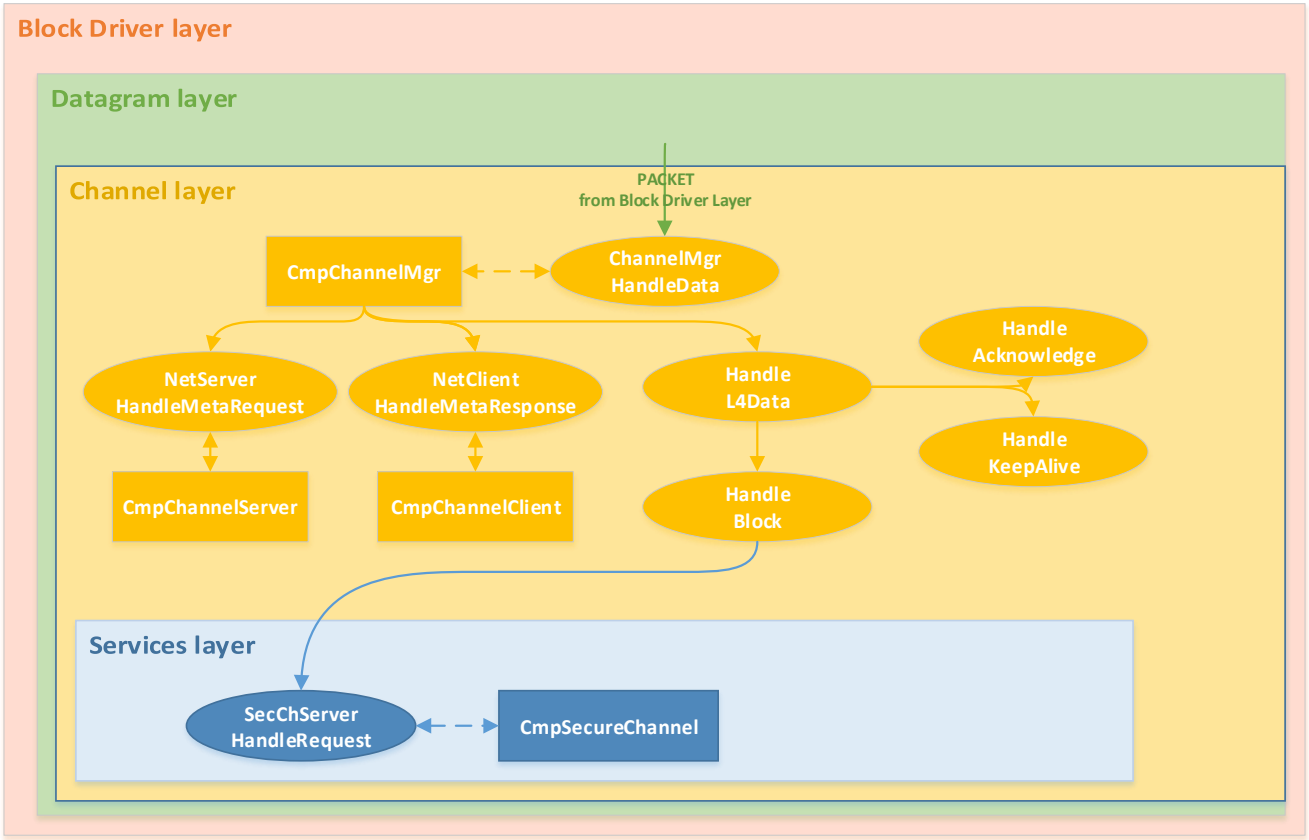
- **max_channels** – значение поля указывает на количество одновременно поддерживаемых каналов связи. Это количество регулируется настройками компонента **CmpChannelMgr**. Сам же канал связи используется на канальном уровне (Chanel layer) стека протоколов CODESYS PDU.
- **byte_order** – указывает на используемый порядок байт в протоколе. Как было сказано ранее, по умолчанию CODESYS PDU использует в качестве порядка байт little – endian, однако порядок байт может быть изменен. Значение поля **byte_order**, равное 1, указывает на использование little-endian.
- ***Unknown*** – назначение поле в ходе исследования распознать не удалось.
- **node_name_length** – размер поля **node_name**.
- **device_name_length** – размер поля **device_name**.
- **vendor_name_length** – размер поля **vendor_name**.
- **target_type** – тип устройства.
- **target_id** – идентификатор устройства.

- **target_version** – версия устройства.
- **node_name** – сетевое имя устройства.
- **device_name** – имя устройства.
- **vendor_name** – имя организации, которая разработала устройство или внедрила в устройство CODESYS Runtime.

Если в поле **version** запроса было указано значение **0x400**, то ответ будет содержать следующие поля: адрес родительского узла, номер лицензии, тип компонента Block Driver.

Канальный уровень (Channel layer)

Канальный уровень – следующий уровень в стеке протоколов CODESYS PDU.



Схематичное представление разбора компонентами пакета CODESYS PDU на канальном уровне (Channel Layer)

Канал представляет собой механизм коммуникации между узлами сети CODESYS, который гарантирует синхронность общения, проверку на целостность передаваемых данных, уведомление о доставке сообщения и передачу больших объемов данных.

Основным компонентом на этом уровне является компонент **CmpChannelMgr** (Component Channel Manager). Этот компонент является менеджером канала связи. Он следит за синхронизацией общения между узлами и целостностью получаемых данных или передает управление на сервер каналов (компонент **CmpChannelServer)** или клиенту каналов связи (компонент **CmpChannelClient)**.

Компонент **CmpChannelServer** является сервером каналов. Он ответственен за:

- создание буфера накопления принятых и отправляемых сообщений;
- создание и закрытие каналов связи;
- выдача информации о канале связи;
- завершение каналов, время жизни которых истекло или к которым давно не было обращения.

Компонент **CmpChannelClient** является клиентом каналов. Он формирует необходимые запросы и занимается обработкой ответов от сервера каналов.

Компонент **CmpChannelMgr** экспортирует функцию **ChannelMgrHandleData**, к которой обращается компонент **CmpRouter** в случае если на уровне датаграм значение поля **service_id** равно 0x40.

```
Call trace:
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора

Pseudocode:
001: Удалено по требованию вендора
[...]
079:     Удалено по требованию вендора
080:     {
[...]
086:       Удалено по требованию вендора
[...]
096:       Удалено по требованию вендора
097:     }
[...]
```

Местоположения передачи управления функции ChannelMgrHandleData

Для канального уровня используется общий заголовок (**channel_common_header**). Он содержит следующие поля:

Fields	size (byte)	Type
package_type	1	Uint8_t
Flags	1	Uint8_t
packet_data	n	Uint8_t[n]

- Поле **package_type** определяет тип пакета. В случае если в значение этого поля установлен старший бит, то пакет является командой для сервера канала. При отсутствии старшего бита пакет предназначен для менеджера канала.
- Поле **flags** имеет различное назначение, которое зависит от поля **package_type.**
- Поле **packet_data** содержит оставшиеся данные пакета и определяется полем **packet_type.**

Экспортируемая компонентом **CmpChannelMgr** функция **ChannelMgrHandleData** является обработчиком на канальном уровне (Channel layer). Работу этой функции можно условно разделить на три категории: работу с сервером каналов, работу с клиентом каналов и работу с менеджером каналов.

```
Call trace:
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора

Pseudocode:
01: Удалено по требованию вендора
02: {
[...]
20:         Удалено по требованию вендора
21:         {
22:           Удалено по требованию вендора
23:           {
[...]
27:             Удалено по требованию вендора
28:           }
29:         }
30:         Удалено по требованию вендора
31:         {
[...]
35:           Удалено по требованию вендора
36:         }
37:       }
 [...]
43:         Удалено по требованию вендора
44:       }
45:       Удалено по требованию вендора
[...]
48: }
```

Фрагмент декомпилированного псевдокода функции ChannelMgrHandleData компонента CmpChannelMgr

Работа с сервером каналов происходит в обработчике **NetServerHandleMetaRequest.**

Ответ от сервера канала обрабатывается клиентом канала в обработчике **NetClientHandleMetaResponse** (строка 35) и (строка 27).

Работа с менеджером каналов происходит в функции **HandleL4Data** (строка 43).

Команды для сервера канала связи

Первой мы рассмотрим группу команд, которая работает непосредственно с сервером канала связи. Эта группа содержит команды для открытия, закрытия канала и получения информации о сервере канала связи.

```
Call trace:
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора

Pseucode:
01: Удалено по требованию вендора
02: {
[...]
27:             Удалено по требованию вендора
```

```
[...]
35:         Удалено по требованию вендора
36:       }
[...]
48: }
```

Фрагмент декомпилированного псевдокода функции ChannelMgrHandleData

Обработчиком команд для сервера канала связи является функция **NetServerHandleMetaRequest,** а команд для клиента – **NetClientHandleMetaResponse**. Первая функция обрабатывает входящие запросы, то есть реализует серверную часть. Вторая функция обрабатывает ответы на запросы, то есть реализует клиентскую часть. Далее будут рассмотрены обе эти функции.

```
Call trace:
       Удалено по требованию вендора
       Удалено по требованию вендора
       Удалено по требованию вендора
       Удалено по требованию вендора
       Удалено по требованию вендора
       Удалено по требованию вендора
       Удалено по требованию вендора
       Удалено по требованию вендора
       Удалено по требованию вендора

Pseudocode:
01: Удалено по требованию вендора
02: {
03:   Удалено по требованию вендора
04:
05:   Удалено по требованию вендора
06:     Удалено по требованию вендора
07:   Удалено по требованию вендора
08:   Удалено по требованию вендора
09:   {
10:     Удалено по требованию вендора
11:       Удалено по требованию вендора
12:       Удалено по требованию вендора
13:     Удалено по требованию вендора
14:       Удалено по требованию вендора
15:       Удалено по требованию вендора
16:     Удалено по требованию вендора
17:       Удалено по требованию вендора
18:       Удалено по требованию вендора
19:   }
20:   Удалено по требованию вендора
21: }
```

Фрагмент декомпилированного псевдокода функции NetServerHandleMetaRequest

Функция **NetServerHandleMetaRequest** (строка 01) определяет три возможных идентификатора команды (**command_id)** для обработки:

- **0xC2** (строка 16) для команды **GET_INFO.** Обработчиком команды выступает функция **HandleInfoReq** (строка 17).
- **0xC3** (строка 10) для команды **OPEN_CHANNEL**. Обработчиком команды выступает функция **HandleOpenChannelReq** (строка 11).
- **0xC4** для команды **CLOSE_CHANNEL**. Обработчиком команды выступает функция **HandleCloseChannelReq** (строка 14).

```
Call trace:
       Удалено по требованию вендора
       Удалено по требованию вендора
       Удалено по требованию вендора
       Удалено по требованию вендора
       Удалено по требованию вендора
       Удалено по требованию вендора
```

```
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора

Pseudocode:
23: Удалено по требованию вендора
24: {
25:   Удалено по требованию вендора
26:
27:   Удалено по требованию вендора
28:     Удалено по требованию вендора
29:   Удалено по требованию вендора
30:   Удалено по требованию вендора
31:   {
32:     Удалено по требованию вендора
33:   }
34:   Удалено по требованию вендора
35:   {
36:     Удалено по требованию вендора
37:   }
38:   Удалено по требованию вендора
39: }
```

Фрагмент декомпилированного псевдокода функции NetClientHandleMetaResponse

При этом клиентская функция **NetClientHandleMetaResponse** (строка 23) определяет только две возможные команды для клиента:

- **0xC3** (строка 30) на команду **OPEN_CHANNEL**. Обработчиком команды выступает функция **HandleOpenChannelResp** (стока 32).
- **0xC4** (строка 34) на команду **CLOSE_CHANNEL**. Обработчиком команды выступает функция **HandleCloseChannelResp** (строка 36).

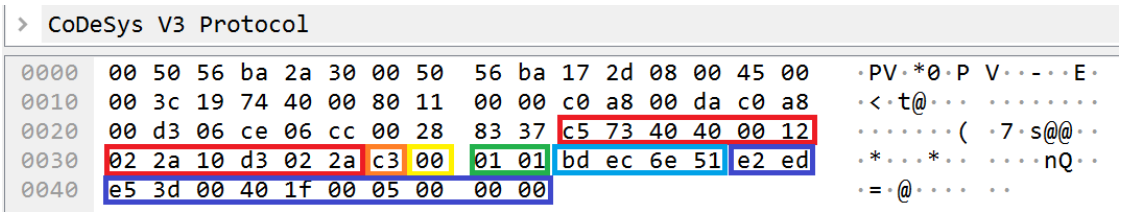
Сообщение канального уровня, которое предназначено для сервера и клиента канала, имеет свой заголовок (**channel_header**). В заголовке **channel_header** используются следующие поля:

Fields	size (byte)	Type
command_id	1	Uint8_t
Flags	1	Uint8_t
version	2	Uint16_t
checksum	4	Uint32_t
command_data	n	Uint8_t[n]

- Поле **command_id** определяет идентификатор команды и обозначает, является ли сообщение ответом на запрос или самим запросом. Выставленный 7-й бит указывает на то, что сообщение является ответом. Остальные первые 6 бит обозначают идентификатор команды. Существуют следующие идентификаторы команд для сервера канала:
  - **0xc2 (GET_INFO)** – информационная команда для получения количества единовременно поддерживаемых каналов на узле;
  - **0xc3 (GET_CHANNEL)** – запрос на создание канала связи между узлами;
  - **0xc4 (CLOSE_CHANNEL)** – запрос на закрытие канала связи между узлами.

- Во время исследования не было обнаружено использования значения, установленного в поле **flags.**
- Поле **version** указывает на идентификатор версии команды. В зависимости от этого поля могут быть использованы дополнительные поля в теле сообщения команды.
- Оставшиеся данные команды определяются командой (**command_id).**

Например, при запросе открытия канала связи серверная функция **NetServerHandleMetaRequest** обрабатывает следующие поля и данные:

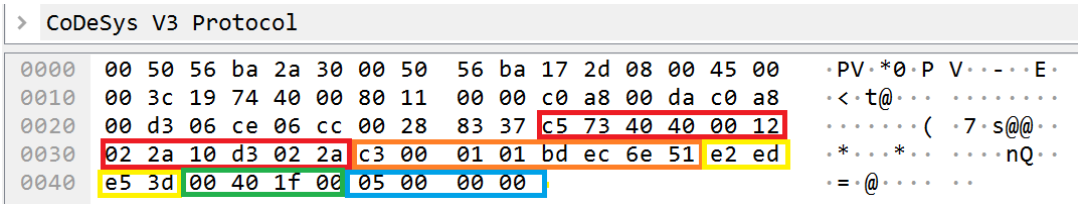


Color						
fields	datagram_ layer_ fields	command_id	Flags	version	checksum	command_data
value	[...]	195 (0xc3) GET_CHANNEL	0	0x101	0x516eecbd	[...]

Используемые поля на канальном уровне (Channel layer)

- **command_id** с идентификатором **0xc3** означает, что сообщение является запросом на открытие канала коммуникации **(GET_CHANNEL).**
- Поле **flags** будет проигнорировано при обработке команды **GET_CHANNEL.**
- Поле **version** определяет наличие дополнительных полей в сообщении. Для текущего значения будет использовано два дополнительных поля.
- **checksum** – контрольная сумма пакета. В качестве контрольной суммы используется алгоритм CRC32.
- **command_data** – поле рассмотрено ниже.

Для запроса команды **GET_CHANNEL** используются следующие поля и данные:



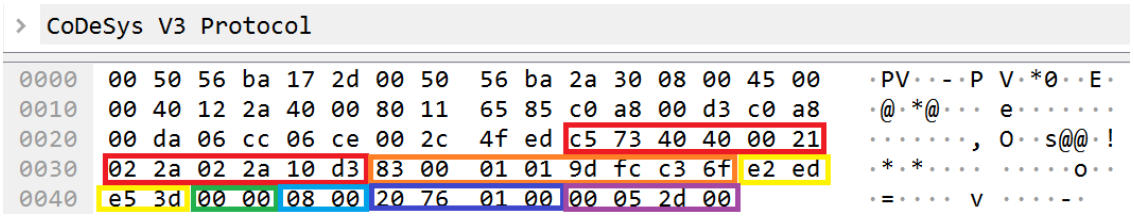
Color					
fields	Datagram_ layer_ fields	channel_header	message_id	receiver_buffer_size	Unknown
value	[...]	[...]	0x3de5ede2	0x1f4000	0x5

Используемые поля запроса с командой GET_CHANNEL

- Поле **datagram_layer_fields** – поля уровня датаграм (datagram layer).
- Поле **channel_header** – заголовок команды серверу канала.

- **Message_id** – идентификатор сообщения. Обычно в качестве значения этого поля используется 4-битное представление текущего времени.
- **Receiver_buffer_size** – максимально возможное количество данных, которое может накапливать получатель в канале связи.

В ответе на этот запрос поле **command_id** заголовка **channel_header** установит 7-й бит. Поля **command_data** в ответе на запрос будут следующими:



Color							
fields	Datagram_ layer_ fields	channel_header	message_id	Reason	channel_id	Reveiver_ buffer_ size	Unknown
value	[...]	[...]	0xe2ede53d	0x00 (OK)	0x08	0x20760100	0x52d00

Используемые поля ответа на команду GET_CHANNEL

- Поле **datagram_layer_fields** – поля уровня датаграм (datagram layer).
- Поле **channel_header** – заголовок команды клиенту канала.
- **Message_id** – возвращаемый идентификатор сообщения. Это значение эквивалентно значению, которое было получено в запросе.
- **Reason** – статус обработки команды.
- **Channel_id** – идентификатор открытого канала связи.
- **Receiver_buffer_size** – максимально возможное количество данных, которое может накапливать получатель в канале связи.

Другие команды используют свой набор полей в поле **command_data**. Исключением является команда **GET_INFO.** Для получения результата этой команды достаточно отправить заполненный **channel_header** с указанным в нем идентификатором команды **GET_INFO**. В ответе будет содержаться одно поле:

Fields	size (byte)	Type
Max_channels	4	Uint32_t

- Поле **Max_channels** содержит максимальное количество единовременно поддерживаемых каналов связи.

Запрос на закрытие канала (**CLOSE_CHANNEL**) использует следующие поля в **command_data**:

Fields	size (byte)	Type
channel_id	2	Uint16_t
Reason	2	Uint16_t

- **channel_id** – идентификатор канала, который необходимо закрыть.
- **reason** – причина закрытия канала.

Узел, получивший такой запрос, не возвращает какой-либо ответ.

Команды для менеджера канала связи

Вторая рассматриваемая группа команд работает непосредственно с менеджером канала связи. Для вызова любой команды из этой группы необходим открытый канал связи. Эта группа содержит команды для передачи данных на следующий уровень, уведомления об их получении и поддержки созданного канала связи.

Команды для менеджера канала связи используют общий заголовок (**channel_manager_header)** со следующими полями**:**

Fields	size (byte)	Type
package_type	1	Uint8_t
Flags	1	Uint8_t
packet_data	N	Uint8_t[n]

- **packet_type** – идентификатор типа пакета. Определяются следующие типы пакетов и их назначения:
  - **BLK** – передача данных для следующего уровня в стеке протоколов;
  - **ACK** – уведомление о получении данных;
  - **KEEPALIVE** – поддержка жизни канала связи.
- **Flags** – дополнительные параметры или указатели, которые специфичны для типа пакета (**packet_type**).
- **packet_data** – специфичные данные для типа пакета (**packet_type**).

Функция **HandleL4Data** обрабатывает все команды рассматриваемой группы.

```
Call trace:
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора

Pseudocode:
01: Удалено по требованию вендора
02: {
[...]
17:
20:    Удалено по требованию вендора
21:    Удалено по требованию вендора
22:    Удалено по требованию вендора
23:    Удалено по требованию вендора
24:    {
25:      Удалено по требованию вендора // ACK
26:        Удалено по требованию вендора
27:        Удалено по требованию вендора
28:          Удалено по требованию вендора
29:        Удалено по требованию вендора
30:        Удалено по требованию вендора
```

```
31:      Удалено по требованию вендора // KEEPALIVE
32:        Удалено по требованию вендора
33:        Удалено по требованию вендора
34:      Удалено по требованию вендора // BLK
35:        Удалено по требованию вендора
36:        Удалено по требованию вендора
37:          Удалено по требованию вендора
38:        Удалено по требованию вендора
39:        Удалено по требованию вендора
[...]
43:    }
[...]
62:    Удалено по требованию вендора
63:    {
[...]
69:      Удалено по требованию вендора
72:      Удалено по требованию вендора
73:    }
74:    Удалено по требованию вендора
75:    {
81:      Удалено по требованию вендора
82:    }
[...]
94: } }
```

Фрагмент декомпилированного псевдокода функции HandleL4Data

Функция **HandleL4Data** (строка 01) определяет три возможных идентификатора **packet_type** для обработки **BLK (0x1)**, **ACK (0x2)**, **KEEPALIVE (0x3)**.

Каждый тип пакета имеет специфичное для этого типа тело данных. Так, типа пакета **BLK** использует следующие поля в теле сообщения:

> CoDeSys V3 Protocol

```
0000  00 50 56 ba 2a 30 00 50  56 ba 17 2d 08 00 45 00   ·PV·*0·P V··-··E·
0010  00 98 2c 5c 40 00 80 11  00 00 c0 a8 00 da c0 a8   ··,\@··· ········
0020  00 d3 06 ce 06 cc 00 84  83 93 c5 73 40 40 00 12   ········ ···s@@··
0030  02 2a 10 d3 02 2a 01 81  20 00 02 00 00 00 01 00   ·*···*·· ·· ·····
0040  00 00 5c 00 00 00 51 40  52 8d 55 cd 10 00 01 00   ··\···Q@ R·U·····
0050  02 00 11 00 00 00 48 00  00 00 00 00 00 00 22 84   ·····H· ·····"·
0060  80 00 01 00 00 00 23 84  80 00 15 14 4e 11 81 01   ·····#· ····N···
0070  b4 00 10 0e 41 64 6d 69  6e 69 73 74 72 61 74 6f   ····Admi nistrato
0080  72 00 11 a0 80 00 ce 01  29 3b 20 5f 36 12 18 42   r····· ); _6··B
0090  46 58 f9 75 70 68 4c 54  68 75 77 3f 70 68 76 44   FX·uphLT huw?phvD
00a0  72 2a 87 55 62 52                                  r*·UbR
```

Color									
fields	Datagram_ layer_ fields	packet_type	flags	channel_id	Blk_id	Ack_id	Remaining_ data_ size	Checksum	Remaining_ data
value	[...]	0x01 (BLK)	0x81	32 (0x20)	0x02	0x1	0x5c	0x89	[...]

Используемые поля для пакета типа BLK на канальном уровне (channel layer)

- **Packet_type** – тип пакета **BLK (0x1),** который означает передачу данных.
- **Flags** – флаги пакета для типа пакета **BLK**. Указанное значение 0x81 означает, что:
  - Узел, который получил этот пакет, выступает в качестве сервера (старший бит), данные запросы являются первыми в передачи (младший бит);

  - Если младший бит не установлен, то сообщение содержит продолжение данных последнего пакета. Младший бит данного пакет указывает, что этот пакет первый раз передает данные следующему уровню.

- **Channel_id** – идентификатор открытого канала, по которому осуществляется передача данных.
- **Blk_id** – идентификатор текущего BLK-сообщения. Этот идентификатор инкрементируется каждый раз стороной, которая инициировала начало общения по каналу.
- **Ack_id** – идентификатор последнего ACK-сообщения. Этот идентификатор меняется каждый раз отвечающей стороной. После получения последнего пакета для передачи данных службе на уровне приложений, отвечающая сторона меняет значение этого идентификатора на значение **Blk_id.**
- **Remaining_data_size** – размер ожидаемых данных, который содержит поле **remaining_data**.
- **Checksum** – контрольная сумма данных, содержащихся в поле **remaining_data.** Для подсчета контрольной суммы используется алгоритм CRC32.

Если пакет типа **BLK** содержал данные, размер которых не превышает максимального размера пакета CODESYS PDU (512 байт), то ответ будет содержать измененное значение поля **flags,** в котором старший бит не будет выставлен, т.к. получатель теперь является клиентом. Значение поля **ack_id** будет изменено на значение поля **blk_id**.

Несмотря на то, что максимальный размер пакета CODESYS PDU 512 байт, по самому протоколу CODESYS PDU возможно передавать данные очень больших размеров. Это работает путем накопления входящих данных на получающей стороне. Получающая сторона понимает, что данные необходимо накапливать, за счет значений в поле **flags.** Понимание того, что пакет содержит последние для команды данные, осуществляется по полям **checksum** и **remaining_data_size.**

Тип сообщения **ACK** используется для уведомления отправляющей стороны о том, что часть данных была получена и ожидается следующая часть данных. Это сообщение использует следующие поля:

Fields	size (byte)	Type
Channel_id	2	Uint16_t
Blk_id	4	Uint32_t

- **Channel_id** – идентификатор канала, по которому был получен пакет типа BLK.
- **Blk_id** – значение поля **Blk_id** из пакета типа BLK, который был получен принимающей стороной.

Тип сообщения **KEEPALIVE** используется для поддержки жизни открытого канала связи. В случае, если менеджер каналов не получает сообщений, то в скором времени он закроет канал. Время ожидания сообщений перед закрытием канала регулируется параметрами компонента.

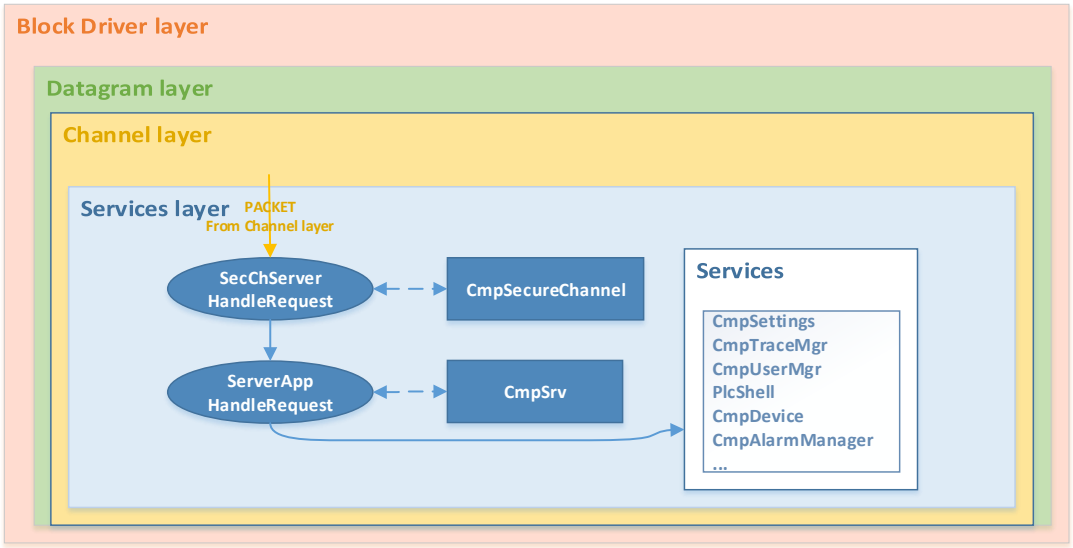
Тип сообщения **KEEPALIVE** использует одно поле:

Fields	size (byte)	Type
Channel_id	2	Uint16_t

- **Channel_id** – идентификатор канала, время которого необходимо продлить.

Уровень служб (services layer)

Следующий уровень в стеке протоколов CODESYS PDU – уровень служб (services layer).



Схематичное представление разбора компонентами пакета CODESYS PDU на уровне служб (Services Layer)

Уровень служб – это условно объединенный уровень из нескольких уровней модели ISO/OSI: сессии, представлений и приложений. Основная задача этого уровня – вызов запрашиваемой службы и передача параметров для её работы. Дополнительные задачи уровня служб — это кодирование, декодирование, шифрование и расшифрование передаваемых на этом уровне данных. Другая дополнительная задача – поддержка сессионности на устройстве.

В последних реализациях CODESYS Runtime поддерживает шифрование данных на уровне служб. Компонент **CmpSecureChannel** шифрует и расшифровывает данные на этом уровне. Это происходит в экспортируемой им функции **SecChServerHandleRequest.** Если данные были успешно расшифрованы или если они не были зашифрованы изначально, то они передаются функции **ServerAppHandleRequest**, которую экспортировал компонент **CmpSrv**.

В случае отсутствия компонента **CmpSecureChannel**, компонент **CmpChannelServer** передает управление функции **ServerAppHandleRequest** самостоятельно.

Сам же формат заголовка сообщения (**protocol_header**), как для зашифрованного, так и для незашифрованного сообщения, следующий:

Fields	size (byte)	Type
protocol_id	2	Uint16_t
header_size	2	Uint16_t
cmd_group	2	Uint16_t
subcmd	2	Uint16_t
session_id	4	Uint32_t
content_size	4	Uint32_t
additional_data	4	Uint32_t
protocol_data	content_size	Uint8_t[content_size]

- **protocol_id** – идентификатор используемого протокола. Этот идентификатор указывает на то, какой обработчик протокола изменил данные или какому протоколу необходимо передать данные службам. Существуют два системных идентификатора протокола:
    - **HeaderTagProtocol** с идентификатором **0xcd55.** Этот идентификатор протокола указывает на то, что данные поля **protocol_data** содержат теги (tags).
    - **SecureProtocol** с идентификатором **0x7557** – протокол для защищенной передачи данных. Этот идентификатор указывает на то, что данные поля **protocol_data** подлежат расшифровке.
- **Header_size** – размер заголовка **protocol_header**. Значение этого поля не содержит размеры предыдущих полей и текущего поля
- **service_group** – идентификатор вызываемой службы. Если в идентификаторе установлен старший бит, то сообщение является ответом от службы. По идентификатору службы в качестве службы определяются следующие компоненты:
    - **CmpAlarmManager – 0x18;**
    - **CmpApp – 0x2;**
    - **CmpAppBP – 0x12;**
    - **CmpAppForce – 0x13;**
    - **CmpCodeMeter – 0x1d;**
    - **CmpCoreDump – 0x1f;**
    - **CmpDevice – 0x1;**
    - **CmpFileTransfer – 0x8;**
    - **CmpIecVarAccess – 0x9;**
    - **CmpIoMgr – 0xb;**
    - **CmpLog – 0x5;**
    - **CmpMonitor – 0x1b;**
    - **CmpOpenSSL – 0x22;**
    - **CmpSettings – 0x6;**
    - **CmpTraceMgr – 0xf;**
    - **CmpTraceMgr – 0xf;**
    - **CmpUserMgr – 0xc;**
    - **CmpVisuServer – 0x4;**
    - **PlcShell – 0x11;**
    - **SysEthernet – 0x7.**
- **service_id** – идентификатор команды. Этот идентификатор определяет, что именно служба должна сделать.
- **session_id** – идентификатор сессии. Содержит значение полученной или пустой сессии. Это значение проверяется обработчиками протоколов и большинством команд, которые требуют повышенных привилегий пользователя.
- **content_size** – размер данных в поле **protocol_data.**
- **additional_data** – поле для дополнительных данных.
- **protocol_data** – данные, сформированные по использованному протоколу (**protocol_id**).

Если сообщение было зашифровано по протоколу **SecureProtocol**, то почти все поля заголовка **protocol_header** будут содержать нулевые байты. Исключением будут поля **header_size и content_size,** которые работают в обычном режиме, и поле **protocol_data**, которое содержит зашифрованный заголовок **protocol_header**. После

расшифровки поля **protocol_data** расшифрованный заголовок **protocol_header** будет обработан обработчиком протокола **HeaderTagProcol**.

Если сообщение не было зашифровано и был использован протокол **HeaderTagProcol,** то поле **protocol_data** будет содержать теги (**tags**).

Пользователь может зарегистрировать свой обработчик для **protocol_id** с помощью функции **ServerRegisterProtocolHandler,** которая экспортирована компонентом **CmpSrv**:

```
01: Удалено по требованию вендора
02: {
[...]
06:
07:   Удалено по требованию вендора
08:   Удалено по требованию вендора
09:     Удалено по требованию вендора
10:   Удалено по требованию вендора
11:   {
12:     Удалено по требованию вендора
13:       Удалено по требованию вендора
14:   }
15:   Удалено по требованию вендора
16:   {
17:     Удалено по требованию вендора
18:     {
19:       Удалено по требованию вендора
20:       Удалено по требованию вендора
21:     }
22:   }
23:   Удалено по требованию вендора
24:     Удалено по требованию вендора
25:   Удалено по требованию вендора
26:   Удалено по требованию вендора
27:   Удалено по требованию вендора
28:     ++Удалено по требованию вендора
29:   Удалено по требованию вендора
30: }
```

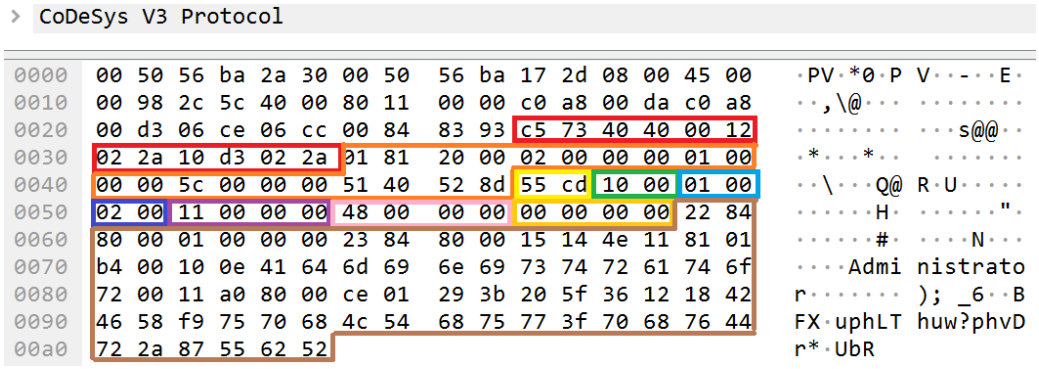
Декомпилированный псевдокод функции ServerRegisterProtocolHandler

Функция **ServerRegisterProtocolHandler** довольно простая, и её алгоритм заключается в следующем:

1. На строках с 10 по 12 функция сравнивает каждый из зарегистрированных обработчиков для поля **protocol_id** с обработчиком, который планируется зарегистрировать. В случае обнаружения среди зарегистрированных такого же обработчика, возвращает соответствующий статус (строка 13).
2. Далее она пытается найти не занятую ячейку для регистрации обработчика (строки 15:21).
3. Регистрирует новый обработчик в свободной ячейке (строка 25:26).

Для поля **service_id** один компонент может одновременно зарегистрировать только один обработчик. Для регистрации своего обработчика для **service_id** необходимо вызвать функцию **ServerRegisterServiceHandler.** Ее алгоритм аналогичен алгоритму функции **ServerRegisterProtocolHandler**, поэтому мы не будем её рассматривать.

Например, для не зашифрованного запроса прохождения аутентификации функция **ServerAppHandleRequest** обрабатывает пакет следующим образом:



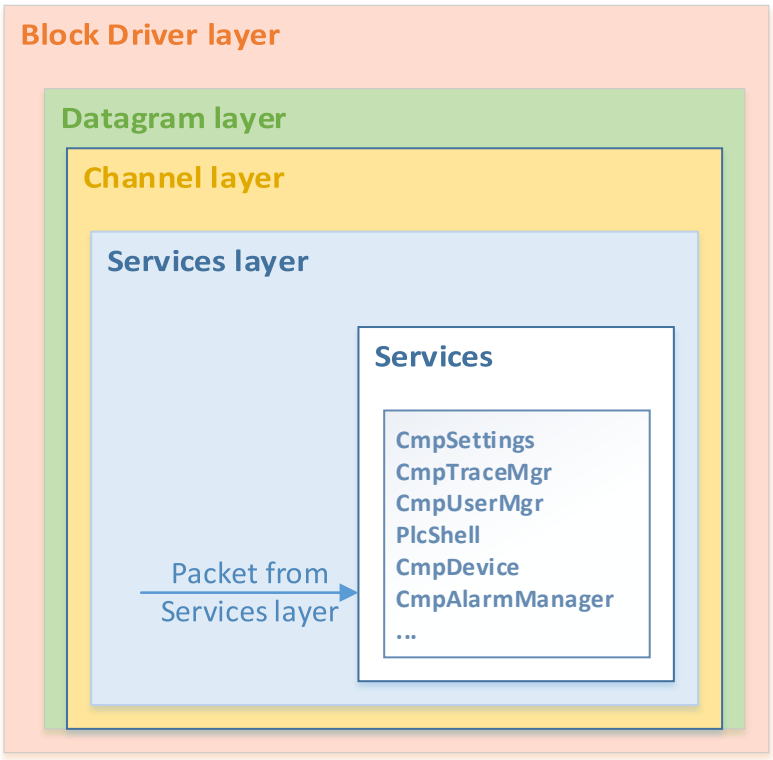
Color						
Fields	datagram_layer_fields	channel_layer	protocol_id	header_size	service_group	service_id
Value	[...]	[...]	0xcd55	0x10	0x1 (**CmpDevice**)	0x2 (**AUTH**)
Color						
Fields	Session_id	protocol_data_size	Additional_data			protocol_data
Value	0x11	0x48	0x0			[...]

Пример содержимого полей на уровне служб (services layer)

- **Protocol_id** – содержит идентификатор **0xcd55**. Это значит, что был использован протокол **HeaderTagProtocol:** данные поля **protocol_data** не зашифрованы, а поля **service_group** и **service_id** содержат значения запрашиваемой службы.
- **Header_size** – содержит значение 0x10. Это значит, что размер используемого заголовка (**protocol_header**) 16 (0x10) байт.
- **Service_group** – указывает на идентификатор службы, которая была зарегистрирована компонентом **CmpDevice.**
- **Service_id** – указывает на идентификатор запрашиваемой команды для службы. Значение 2 указывает на то, что зарегистрированной службе компонентом **CmpDevice** необходимо выполнить команду **AUTH.**
- **Protocol_data_size** – указывает**,** что размер данных поля **protocol_data** 72 (0x48) байта.
- Поле **additional_data** не было использовано.

Теги (tags)

Последний рассматриваемый слой в стеке протоколов CODESYS PDU – это теги (tags).



Схематичное представление разбора компонентами пакета CODESYS PDU на уровне служб

Теги – это интерфейс передачи параметров для служб. Служба на стороне клиента знает, каким образом необходимо сформировать данные, чтобы служба на стороне сервера корректно получила параметры из этих данных.

Типы тегов

Теги передаются в поле **protocol_data** заголовка **protocol_header.** Они бывают двух видов: тег данных и родительский тег.

Оба вида тегов имеют одинаковую структуру, но используют разные размеры для первых двух элементов структуры полей:

Fields	Parent tag		Data tag	
	size (byte)	Type	Size (byte)	Type
tag_id	2	Uint16_t	1	Uint8_t
tag_size	2	Uint16_t	1	Uint8_t
additional_data	0:10	Uint8_t[0:9]	0:10	Uint8_t[0:9]
tag_data	Tag_size	Uint8_t[tag_size]	Tag_size	Uint8_t[tag_size]

- **tag_id** – идентификатор тега. Идентификаторы родительского тега и тега данных различают по значению старшего бита. Если значение старшего бита выставлено, то тег является родительским, и все оставшиеся данные характерны для родительского тега. В противном случае тег является тегом данных, и его данные содержат конечные параметры для службы.
- **tag_size** – размер данных. Это поле определяет количество данных в поле **tag_data**. Помимо этого, значение старшего бита поля **tag_size** определяет существование дополнительного поля **additional_data:** если значение старшего бита установлено, то поле **additional_data** существует.
- **additional_data** – дополнительное поле. Имеет динамический размер, который не может быть больше 10 байт. Окончание этого поля определяется нулевым байтом.
- **tag_data** – данные родительского тега или данные тега данных.

Данные, извлеченные службой из тега данных, преобразуются в данные определенного типа. Например, тег, содержащий 4 байта в поле **tag_data**, может быть приведен службой к одному из числовых типов. Сам тип переменной в структуре тега не передается. Однако, множество раз было замечено, что службы CODESYS Runtime передают группу тегов, из которых один тег может содержать значение, второй – идентификатор типа, третий – размер. Такие связки тегов обычно объединены под одним родительским тегом.

Родительский тег используется для компоновки нескольких тегов в один логический элемент – пример такой компоновки приведен выше. Другим примером может быть компоновка тегов с именем и паролем пользователя, которые используются для аутентификации. Тег данных, содержащий имя пользователя, и тег данных, содержащий пароль, могут быть укомплектованы в один родительский тег.

Обработка тегов

Рассмотрим, как конкретный обработчик службы извлекает параметры из полученных тегов на примере пакета с тегами для аутентификации пользователя. В этом случае будут разобраны теги:

```
> CoDeSys V3 Protocol
0000  00 50 56 ba 2a 30 00 50  56 ba 17 2d 08 00 45 00   ·PV·*0·P V··-··E·
0010  00 98 2c 5c 40 00 80 11  00 00 c0 a8 00 da c0 a8   ··,\@··· ········
0020  00 d3 06 ce 06 cc 00 84  83 93 c5 73 40 40 00 12   ········ ···s@@··
0030  02 2a 10 d3 02 2a 01 81  20 00 02 00 00 00 01 00   ·*···*·· ········
0040  00 00 5c 00 00 00 51 40  52 8d 55 cd 10 00 01 00   ··\···Q@ R·U·····
0050  02 00 11 00 00 00 48 00  00 00 00 00 00 00 22 84   ·····H· ·····"·
0060  80 00 01 00 00 00 23 84  80 00 15 14 4e 11 81 01   ·····#· ····N···
0070  b4 00 10 0e 41 64 6d 69  6e 69 73 74 72 61 74 6f   ····Admi nistrato
0080  72 00 11 a0 80 00 ce 01  29 3b 20 5f 36 12 18 42   r······ ); _6··B
0090  46 58 f9 75 70 68 4c 54  68 75 77 3f 70 68 76 44   FX·uphLT huw?phvD
00a0  72 2a 87 55 62 52                                  r*·UbR
```

Color								
fields	datagram_ layer_ fields	channel_ layer_ fields	Services_ layer_ fields	Data_tag_1	Data_tag_2	Parent_tag_1	Data_tag_3	Data_tag_4

Пример разбора содержимого поля protocol_data на теги в запросе прохождения аутентификации

Внутри заголовка **services_layer_fields** есть поле **protocol_data**. Если разбить это поле на теги, то в нем находятся 4 тега данных и 1 родительский тег.

Иерархия тегов будет следующей:

- Data_tag_1
- Data_tag_2
- Parent_tag_1
    - Data_tag_3
    - Data_tag_4

Теги данных Data_tag_1 и Data_tag_2 находятся на одном уровне с родительским тегом Parent_tag_1.

Data_tag_3 и Data_tag_4 находятся внутри родительского тега.

Для взаимодействия с тегами используется множество API-функций, среди которых есть функции для взаимодействия с входящими тегами с префиксом BTagReader (Binary Tag Reader) и для взаимодействия с исходящими тегами с префиксом BtagWriter (Binary Tag Writer).

Основные API-функции для взаимодействия с тегами:

Для входящих тегов

1. **BTagReaderInit** – инициализация структуры чтения полученных данных. Структура чтения данных хранит в себе следующие элементы: сами данные, текущую позицию в данных, возможную конечную позицию данных и размер данных. В большинстве случаев, все функции для взаимодействия со входящими тегами используют для работы только указатель на текущую позицию в данных – так происходит «перемещение» по тегам в чистых данных.
2. **BTagReaderGetTagId** – получение идентификатора тега.
3. **BTagReaderGetContent** – получение данных тега.
4. **BTagReaderMoveNext** – переход к следующему тегу.
5. **BTagReaderSkipContent** – перемещение к концу данных текущего тега.

Для исходящих тегов

1. **BTagWriterInit** – инициализация структуры записи исходящих данных. Структура записи данных хранит в себе следующие элементы: изначально полученные данные, указатель на конец данных и текущий размер данных. Все функции для взаимодействия с исходящими тегами меняют все элементы структуры.
2. **BTagWriterStartTag** – открытие нового тега для исходящих данных. Открытие тега должно сопровождаться вызовом функции закрытия тега – **BTagWriterEndTag**.
3. **BTagWriterAppendBlob** – добавление данных для созданного тега.
4. **BTagWriterEndTag** – закрытие созданного тега данных.
5. **BTagWriterFinish** – завершение записи тегов в качестве исходящих данных. По сути, эта функция проверяет, что все добавленные теги имеют валидную структуру и были закрыты функцией **BTagWriterEndTag**.

В пакете есть заголовок **services_layer_fields**, в котором значение поля **service_group** равно 1. Этот идентификатор зарегистрирован компонентом **CmpDevice**.

Сам компонент во время своей инициализации регистрирует функцию **DeviceServiceHandler** в качестве службы, устанавливая ей идентификатор, равный 1:

```
Call trace:
       Удалено по требованию вендора

Pseudocode:
1: Удалено по требованию вендора
2: {
3:   Удалено по требованию вендора
4: }
```

Декомпилированный псевдокод функции DeviceSrvInitComm компонента CmpSrv

В этом же пакете команда, указанная в поле идентификатора команды (**service_id),** равна 2. Функция **DeviceServiceHandler** определяет 9 команд, среди которых есть команда с идентификатором 2 (строка 254):

```
001: Удалено по требованию вендора
002: {
[...]
133:   Удалено по требованию вендора
134:   {
135:     Удалено по требованию вендора
[...]
254:     Удалено по требованию вендора
[...]
399:     Удалено по требованию вендора
[...]
411:     Удалено по требованию вендора
[...]
473:     Удалено по требованию вендора
[...]
548:     Удалено по требованию вендора
[...]
611:     Удалено по требованию вендора
[...]
624:     Удалено по требованию вендора
[...]
695:     Удалено по требованию вендора
[...]
763: }
```

Фрагмент декомпилированного псевдокода функции DeviceServiceHandler

Функция **DeviceServiceHandler** условно выполняет команду с идентификатором 2 в три этапа:

1. Извлекает параметры и устанавливает локальные значения;
2. Исполняет команды с локальными значениями и полученными параметрами;
3. Возвращает результат исполнения команды.

```
001: Удалено по требованию вендора
002: {
[...]
129:   Удалено по требованию вендора
130:   Удалено по требованию вендора
131:   Удалено по требованию вендора
132:   Удалено по требованию вендора
133:   Удалено по требованию вендора
134:   {
[...]
254:     Удалено по требованию вендора
[...]
266:       Удалено по требованию вендора
267:       Удалено по требованию вендора
268:       {
269:         Удалено по требованию вендора
```

```
270:          Удалено по требованию вендора
271:          {
272:            Удалено по требованию вендора
273:              Удалено по требованию вендора
274:              Удалено по требованию вендора
275:            Удалено по требованию вендора
276:              Удалено по требованию вендора
277:              Удалено по требованию вендора
278:              {
279:                Удалено по требованию вендора
280:                Удалено по требованию вендора
281:                {
282:                  Удалено по требованию вендора
283:                }
284:                Удалено по требованию вендора
285:                {
286:                  Удалено по требованию вендора
287:                  Удалено по требованию вендора
288:                }
289:                Удалено по требованию вендора
290:                {
291:                  Удалено по требованию вендора
292:                }
293:                Удалено по требованию вендора
294:                Удалено по требованию вендора
295:                Удалено по требованию вендора
296:              }
297:              Удалено по требованию вендора
298:            Удалено по требованию вендора
299:              Удалено по требованию вендора
300:              Удалено по требованию вендора
301:            Удалено по требованию вендора
302:              Удалено по требованию вендора
303:              Удалено по требованию вендора
304:          }
305:          Удалено по требованию вендора
306:          Удалено по требованию вендора
307:          Удалено по требованию вендора
308:        }
[...]
315:        Удалено по требованию вендора
316:        {
317:          Удалено по требованию вендора
318:          Удалено по требованию вендора
[...]
327:        }
330:        Удалено по требованию вендора
331:        Удалено по требованию вендора
332:        {
333:          Удалено по требованию вендора
334:        }
335:        Удалено по требованию вендора
336:        {
337:          Удалено по требованию вендора
338:          Удалено по требованию вендора
339:          {
[...]
341:            Удалено по требованию вендора
342:            Удалено по требованию вендора
343:            Удалено по требованию вендора
344:            Удалено по требованию вендора
345:            Удалено по требованию вендора
346:            Удалено по требованию вендора
347:            Удалено по требованию вендора
348:            Удалено по требованию вендора
349:          }
[...]
357:          Удалено по требованию вендора
[...]
365:          Удалено по требованию вендора
366:        }
367:        Удалено по требованию вендора
```

```
[...]
376:        Удалено по требованию вендора
377:        Удалено по требованию вендора
378:        Удалено по требованию вендора
379:        Удалено по требованию вендора
380:        Удалено по требованию вендора
[...]
389:        Удалено по требованию вендора
390:        Удалено по требованию вендора
391:        Удалено по требованию вендора
392:        Удалено по требованию вендора
393:        Удалено по требованию вендора
394:        Удалено по требованию вендора
395:        Удалено по требованию вендора
396:        Удалено по требованию вендора
397:        Удалено по требованию вендора
398:        Удалено по требованию вендора
[...]
761:    }
762:    Удалено по требованию вендора
763: }
```

Фрагмент декомпилированного псевдокода функции DeviceServiceHandler

Алгоритм работы команды с идентификатором 2 для каждого этапа следующий:

Этап извлечения параметров

1. На строке 131 происходит инициализация структуры для записи исходящих данных (**writer**), которая будет использоваться на этапе возвращения результата. На строке 132 – инициализация структуры для чтения входящих данных (**reader**), которая используется на текущем этапе.
2. Происходит попытка распознать теги во входящих данных (строка 266). В случае успеха из первого тега извлекается идентификатор тега (269).
3. В зависимости от полученного на предыдущем шаге идентификатора тега происходит запись данных тега в соответствующие переменные.
   Для идентификатора 0x23 (строка 272) происходит запись в переменную **pulChallenge** (строка 273), для идентификатора 0x22 (строка 298) – в переменную **pulCrypeType** (строка 299).
   Если идентификатор тега равен 0x81 (строка 275), то тег является родительским и в нем происходит поиск тегов данных (строка 279) с идентификаторами 16 (строка 280). Данные из найденных тегов записываются в переменную **user_name** (строка 282).
   Из найденного тега с идентификатором 17 (строка 284) данные записываются в переменную **encrypted_password** (строка 286).
4. На строке 315 происходит проверка того, что переменные, необходимые для исполнения команды, заполнены данными из тегов.

Этап исполнения команды

1. Полученные переменные **encrypted_password**, **pulCrypeType** и **pulChallenge** используются в функции расшифровки пароля **UserMgrDecryptPassword** (строка 318). Расшифрованный пароль будет записан в переменную **decrypted_password**.
2. Переменная **user_name** будет использована в функции проверки существования пользователей **FindUser**. Эта функция ищет запись о пользователях в базе данных.
3. В случае нахождения записи о пользователе с именем из переменной **user_name**, происходит проверка соответствия пароля пользователя

с расшифрованным паролем, который был сохранен в переменную **decrypted_password**.

4. Если расшифрованный пароль совпадает с паролем из базы данных, то для текущего пользователя происходит проверка прав на объект “Device” (строка 357).
5. При наличии прав у пользователя на объект “Device”, сгенерированный идентификатор сессии (переменная **ulSessionId**) присваивается текущему пользователю (строка 365) и используемому каналу связи (строка 367).

Этап возвращения результата

1. Для исходящих данных открывается тег с идентификатором 0x82 (строка 376). Этот тег является родительским, и внутри него последовательно открываются и закрываются теги с идентификаторами 0x20 (открытие на строке 377 и закрытие на строке 379), 0x24 (открытие на строке 389 и закрытие на строке 391) и 0x21 (открытие на строке 392 и закрытие на строке 394).
2. В теги данных записываются следующие значения: с идентификатором 0x20 – значение статуса выполнения команды (строка 378); с идентификатором 0x24 – значение настроек устройства (строка 390); с идентификатором 0x21 – значение сгенерированной сессии (строка 393).
3. На строке 395 закрывается родительский тег с идентификатором 0x82 и завершается запись исходящих данных (строка 396).

В случае отсутствия записи о пользователе, нехватки прав, несовпадения паролей или неправильных данных, на этапе возвращения результата формируется соответствующий ответ.

На запрос прохождения аутентификации CODESYS Runtime при верных данных аутентификации пользователя возвращается следующий пакет:

```
> CoDeSys V3 Protocol
```

```
0000  00 50 56 ba 17 2d 00 50  56 ba 2a 30 08 00 45 00   ·PV··-·P V·*0··E·
0010  00 68 1e 03 40 00 80 11  59 84 c0 a8 00 d3 c0 a8   ·h··@··· Y·······
0020  00 da 06 cc 06 ce 00 54  17 b6 c5 73 40 40 00 21   ·······T ···s@@·!
0030  02 2a 02 2a 10 d3 01 01  1c 00 02 00 00 00 02 00   ·*·*···· ········
0040  00 00 2c 00 00 00 70 63  90 23 55 cd 10 00 81 00   ··,···pc ·#U·····
0050  02 00 11 00 00 00 18 00  00 00 00 00 00 00 82 01   ········ ········
0060  94 00 20 02 00 00 24 84  80 00 0f 00 00 00 21 84   ·· ···$· ·····!·
0070  80 00 12 f9 3e 9f                                 ····>·
```

Color							
fields	datagram_ layer_ fields	channel_ layer_ fields	Services_ layer_ fields	Parent_tag	Tag with status	Tag with settings	Tag with **session_id**

Пример разбора на теги содержимого поля protocol_data в ответе на запрос прохождения аутентификации

Таким образом, мы можем однозначно определить, в каких тегах с какими идентификаторами будут переданы определенные параметры для служб. Основные параметры в этом ответе будут находиться в теге с идентификатором 0x21. Этот тег будет содержать идентификатор сессии, который потом будет использован в качестве значения поля **session_id** на уровне служб (services layer).

Обнаруженные уязвимости и возможные атаки

После того, как мы создали условия для проведения статического и динамического анализа и разобрали луковицу из слоев протокола, мы смогли заняться поиском уязвимостей.

В этой главе будет рассмотрено несколько обнаруженных логических уязвимостей, эксплуатация которых приводит к захвату устройства, на котором установлено ПО CODESYS Runtime.

По результатам нашего исследования мы также планируем опубликовать статью, посвященную автоматизированному поиску сетевых бинарных уязвимостей в условиях отсутствия исходного кода.

Описание стенда

Для демонстрации эксплуатации уязвимостей мы будем использовать стенд, состоящий из следующих сетевых узлов:

1. Компьютеры атакующего с сетевыми адресами 192.168.0.2 и 192.168.0.30. Узлы атакующего обозначены как Attacker №1 и Attacker №2;
2. Устройство Raspberry Pi с запущенным CODESYS Control For Raspberry PI, на котором установлен сетевой адрес 192.168.0.91. Далее для этого узла будет использоваться сокращенное имя Client №1;
3. Устройство Raspberry Pi с запущенным CODESYS Control For Raspberry PI, на котором установлен сетевой адрес 192.168.0.92. Далее для этого узла будет использоваться сокращенное имя Client №2;
4. Компьютер с установленным CODESYS Development System с сетевым адресом 192.168.0.39. Для этого узла будет использоваться сокращенное имя IDE.

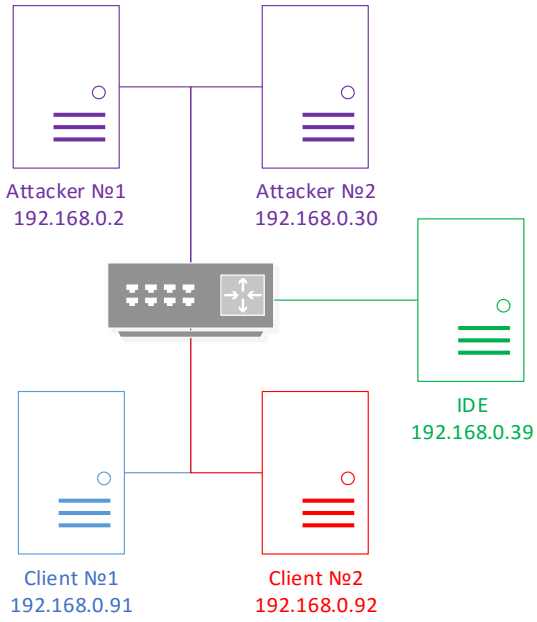


Схема стенда

Атаки на уровне датаграм (datagram layer)

Протокол CODESYS PDU основан на модели ISO/OSI. Исходя из этого, можно предположить, что вместе с идеей модели ISO/OSI и её стеком протоколов, протокол CODESYS PDU унаследовал слабости этой модели и соответствующие этим слабостям угрозы безопасности.

Подмена IP-адреса (IP-spoofing)

Для каждого из протокола в модели ISO/OSI есть свой перечень угроз. Подмена IP-адреса (IP-spoofing) – атака модели ISO/OSI на сетевом уровне (network layer), которая заключается в подмене адреса источника сообщения (IP SRC) с целью сокрытия адреса отправителя. Жертва, получив такой пакет, обработает запрос и вернет ответ на указанный в поле адрес источника (IP SRC).

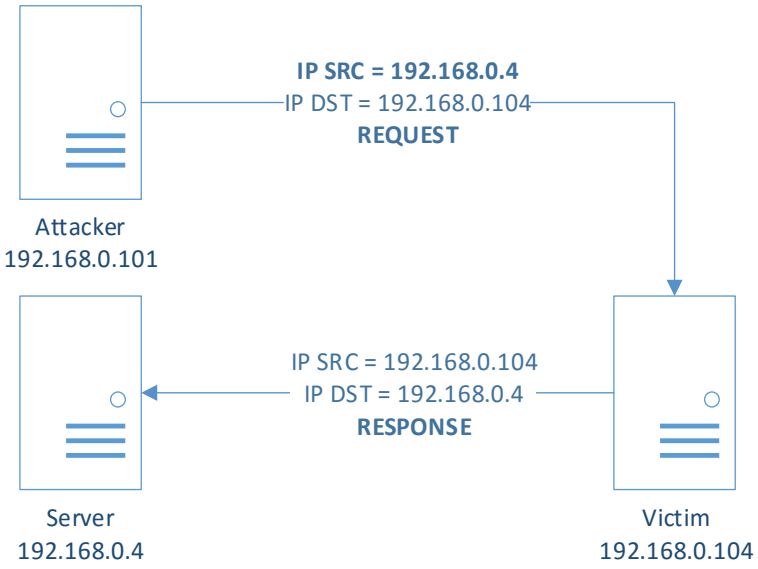


Схема атаки подмены IP-адреса (IP-spoofing) для модели ISO/OSI

Сокрытие адреса отправителя используется для обмана систем безопасности и усложнения раскрытия атаки.

Атаки, аналогичные IP-spoofing, можно провести по протоколу CODESYS PDU.

Будут рассмотрены две атаки на протокол CODESYS PDU:

1. Атака с целью сокрытия адреса источника сообщения;
2. Атака с целью перехвата управления в существующем канале общения между узлами сети CODESYS.

Атака с целью сокрытия адреса источника сообщения

Компонент **CmpRouter** обрабатывает поля на уровне датаграм (datagram layer). В заголовке уровня датаграм (datagram layer) находятся поля адресации, параметризации пакета, идентификатор службы канального уровня и другие.

Среди полей адресации есть поле **receiver**, которое указывает, на какой адрес послать ответ на запрос.

```
> Frame 8: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interfa
> Ethernet II, Src: PcsCompu_90:85:bf (08:00:27:90:85:bf), Dst: Raspberr_92:3a:
> Internet Protocol Version 4, Src: 192.168.0.39, Dst: 192.168.0.92
> User Datagram Protocol, Src Port: 1742, Dst Port: 1740
> CoDeSys V3 Protocol
```

```
0000  b8 27 eb 92 3a ff 08 00  27 90 85 bf 08 00 45 00   ·'··:··· '·····E·
0010  00 3c 02 03 00 00 80 11  b6 da c0 a8 00 27 c0 a8   ·<····· ·····'··
0020  00 5c 06 ce 06 cc 00 28  a1 f8 c5 73 40 40 00 11   ·\·····( ···s@@··
0030  00 5c 02 27 00 00 c3 00  01 01 df db 21 ab a5 ed   ·\·'···· ····!···
0040  37 39 00 40 1f 00 04 00  00 00                     79·@···· ··
```

Color				
fields	Sender		**Receiver**	
	Port index	Address	Port index	Address
value	0 (1740)	0x5c (192.168.0.92)	2 (1742)	0x27 (192.168.0.39)

Местоположение поля receiver в заголовке уровня датаграм (datagram layer)

Как видно на примере потока данных выше, в поле **receiver** содержится значение последнего байта числового представления адреса узла IDE (0x27) и значение индекса порта, равное 2. Оба эти значения совпадают с числовым значением адреса источника сообщения (поле src, равное 192.168.0.39) и со значением источника порта, с которого был отправлен запрос (поле src port, равно 1742).

Изменив значение в поле **receiver**, злоумышленник может реализовать классическую схему атаки по подмене IP-адреса (IP-spoofing):

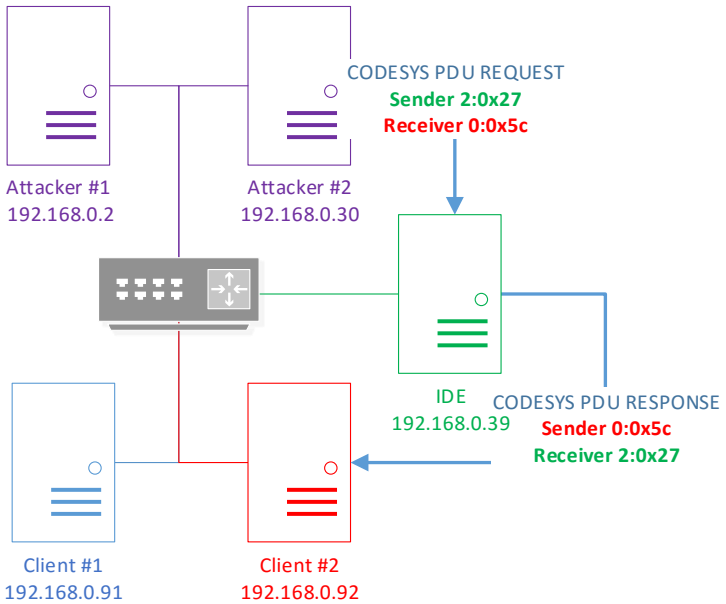


Схема классического варианта атаки подмены IP-адреса (IP-spoofing) по протоколу CODESYS PDU

Существует еще одна архитектурная уязвимость протокола CODESYS PDU, приводящая к реализации расширенного варианта атаки подмены IP-адреса. Она заключается в одной из обязанностей компонента **CmpRouter** – маршрутизации.

```
> Frame 8: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interfac
> Ethernet II, Src: PcsCompu_90:85:bf (08:00:27:90:85:bf), Dst: Raspberr_92:3a:f
> Internet Protocol Version 4, Src: 192.168.0.39, Dst: 192.168.0.92
> User Datagram Protocol, Src Port: 1742, Dst Port: 1740
> CoDeSys V3 Protocol
```

```
0000  b8 27 eb 92 3a ff 08 00  27 90 85 bf 08 00 45 00   ·'··:··· '·····E·
0010  00 3c 02 03 00 00 80 11  b6 da c0 a8 00 27 c0 a8   ·<······ ·····'··
0020  00 5c 06 ce 06 cc 00 28  a1 f8 c5 73 40 40 00 11   ·\·····( ···s@@··
0030  00 5c 02 27 00 00 c3 00  01 01 df db 21 ab a5 ed   ·\·'···· ····!···
0040  37 39 00 40 1f 00 04 00  00 00                     79·@···· ··
```

Color				
fields	Sender		Receiver	
	Port index	Address	Port index	Address
value	0 (1740)	0x5c (192.168.0.92)	2 (1742)	0x27 (192.168.0.39)

Местоположение поля sender в заголовке уровня датаграм (datagram layer)

Поле **sender** указывает на адрес узла, для которого предназначен пакет. Компонент **CmpRouter** перенаправляет полученный пакет CODESYS PDU на другой узел в сети, соответствующий указанному в поле **sender**, если значение поля **sender** не совпадает с адресом узла, получившего пакет.

Манипулируя полем **sender**, злоумышленник может модифицировать атаку по подмене IP-адреса, дополнив её промежуточным узлом. Промежуточный узел будет выступать в качестве прокси для перенаправления вредоносного пакета на другие узлы в сети CODESYS.

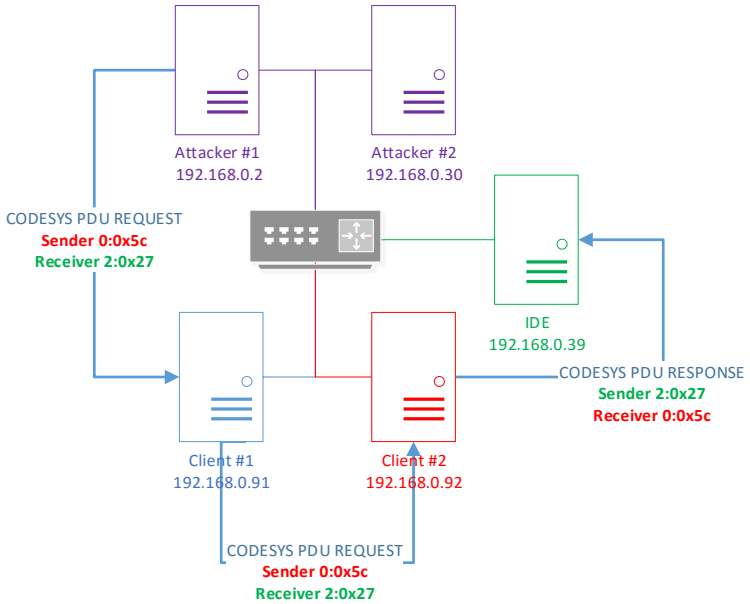


Схема модифицированного варианта атаки подмены IP-адреса (IP-spoofing) по протоколу CODESYS PDU

Последним штрихом в атаке по подмене IP-адреса по протоколу CODESYS PDU будет скрытое получение ответа на перенаправленный запрос. Скрытое получение ответа

можно осуществить, указав в сообщении в качестве получателя ответа широковещательный адрес.

```
> Frame 1444: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
> Ethernet II, Src: D-LinkIn_ad:fb:c0 (9c:d6:43:ad:fb:c0), Dst: Raspberr_92:3a:ff (b8:27:eb:92:3a:ff)
> Internet Protocol Version 4, Src: 192.168.0.30, Dst: 192.168.0.92
> User Datagram Protocol, Src Port: 1740, Dst Port: 1740
> CoDeSys V3 Protocol
```

```
0000  b8 27 eb 92 3a ff 9c d6  43 ad fb c0 08 00 45 00   ·'··:··· C·····E·
0010  00 3c f6 09 00 00 80 11  00 00 c0 a8 00 1e c0 a8   ·<······ ········
0020  00 5c 06 cc 06 cc 00 28  82 04 c5 73 40 40 00 11   ·\·····( ···s@@··
0030  00 5c 00 ff 00 00 c3 00  01 01 bf 19 68 af 00 00   ·\····· ····h···
0040  00 00 00 40 1f 00 04 00  00 00                     ···@···· ··
```

Color				
fields	Sender		**Receiver**	
	Port index	Address	Port index	Address
value	0 (1740)	0x5c (192.168.0.92)	0 (1740)	0xff (192.168.0.255)

Пример пакета, в котором широковещательный адрес указан в качестве значения для поля receiver в заголовке уровня датаграм (datagram layer)

Получив запрос, в котором в качестве получателя ответа в поле **receiver** указан последний байт широковещательного адреса (0xff), узел вернет ответ на широковещательный адрес.

```
1444 235.811028   192.168.0.30    192.168.0.92     CODESY…   74 1740 → 1740 Len=32
1445 235.813098   192.168.0.92    192.168.0.255    CODESY…   78 1740 → 1740 Len=36
1446 235.816995   192.168.0.30    192.168.0.92     CODESY…  110 Device.GetTargetIdent
```

```
> Frame 1445: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface 0
> Ethernet II, Src: Raspberr_92:3a:ff (b8:27:eb:92:3a:ff), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
> Internet Protocol Version 4, Src: 192.168.0.92, Dst: 192.168.0.255
> User Datagram Protocol, Src Port: 1740, Dst Port: 1740
> CoDeSys V3 Protocol
```

```
0000  ff ff ff ff ff ff b8 27  eb 92 3a ff 08 00 45 00   ·······' ··:···E·
0010  00 40 5d ce 40 00 40 11  5a 33 c0 a8 00 5c c0 a8   ·@]·@·@· Z3···\··
0020  00 ff 06 cc 06 cc 00 2c  1b f7 c5 73 40 40 00 11   ·······, ···s@@··
0030  00 ff 00 5c 00 00 83 00  01 01 2f da 45 dd 00 00   ···\···· ··/·E···
0040  00 00 00 00 08 00 20 76  01 00 00 04 29 08         ····· v ····)·
```

Пример отправки узлом ответа на широковещательный адрес

Злоумышленник может скрыть адрес своего узла, получая с широковещательного узла ответы от узлов жертвы.

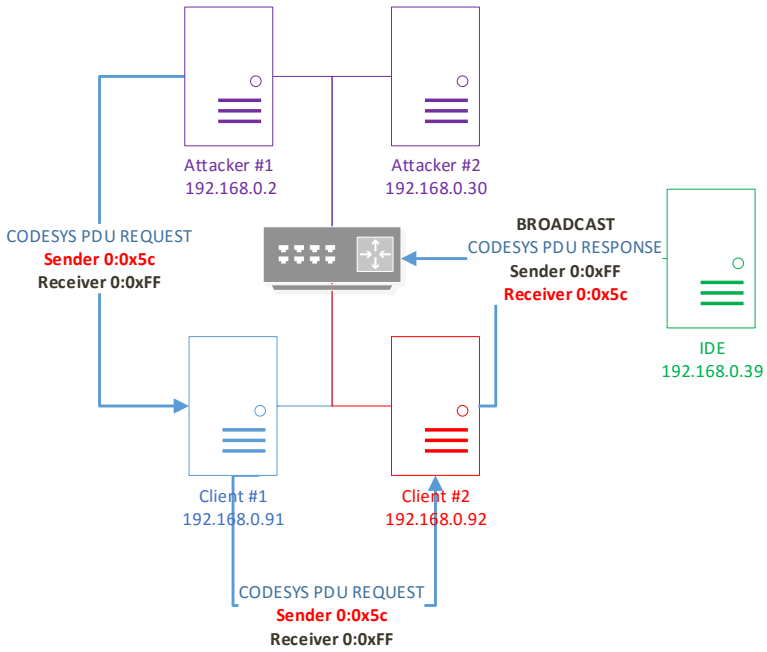


Схема атаки подмены IP-адреса (IP-spoofing) по протоколу CODESYS PDU со скрытым получением ответа на запрос

Таким образом, автоматизировав эксплуатацию обнаруженных уязвимостей и реализовав описанную схему атаки, злоумышленник мог усложнить аналитикам проведение расследования атаки.

Атака с целью с целью перехвата управления в существующем канале общения между узлами сети CODESYS

Ввиду того, что компонент **CmpRouter** выполняет свою работу, основываясь только на данных в пакете CODESYS PDU, злоумышленник может внедриться в существующую коммуникацию между узлами. Однако из-за того, что каждый уровень протокола CODESYS PDU зависим от предыдущего уровня, для перехвата управления на самом последнем уровне (уровне служб) необходимо перехватить управление на всех предыдущих уровнях.

Для взаимодействия на канальном уровне (channel layer) участникам сети необходимо установить канал связи. Для вызова большинства служб на уровне служб (services layer) одному узлу необходимо пройти аутентификацию на другом узле и получить идентификатор сессии. Значение идентификатора канала передается в заголовке на канальном уровне. Значение идентификатора сессии передается в заголовке уровня служб.

Таким образом, перехват управления в протоколе CODESYS PDU можно разбить на несколько задач:

1. Получение адресов участников общения;
2. Получение идентификатора канала;
3. Получение идентификатора BLK и ACK;
4. Получение идентификатора сессии.

Первая задача может быть решена штатными возможностями протокола CODESYS PDU. Третья решается перебором идентификаторов.

Чтобы решить вторую и четвертую задачи необходимо было обнаружить и проэксплуатировать несколько обнаруженных уязвимостей.

Если бы злоумышленник обнаружил найденные нами уязвимости и автоматизировал выполнение атаки, то он смог бы реализовать следующую схему атаки:

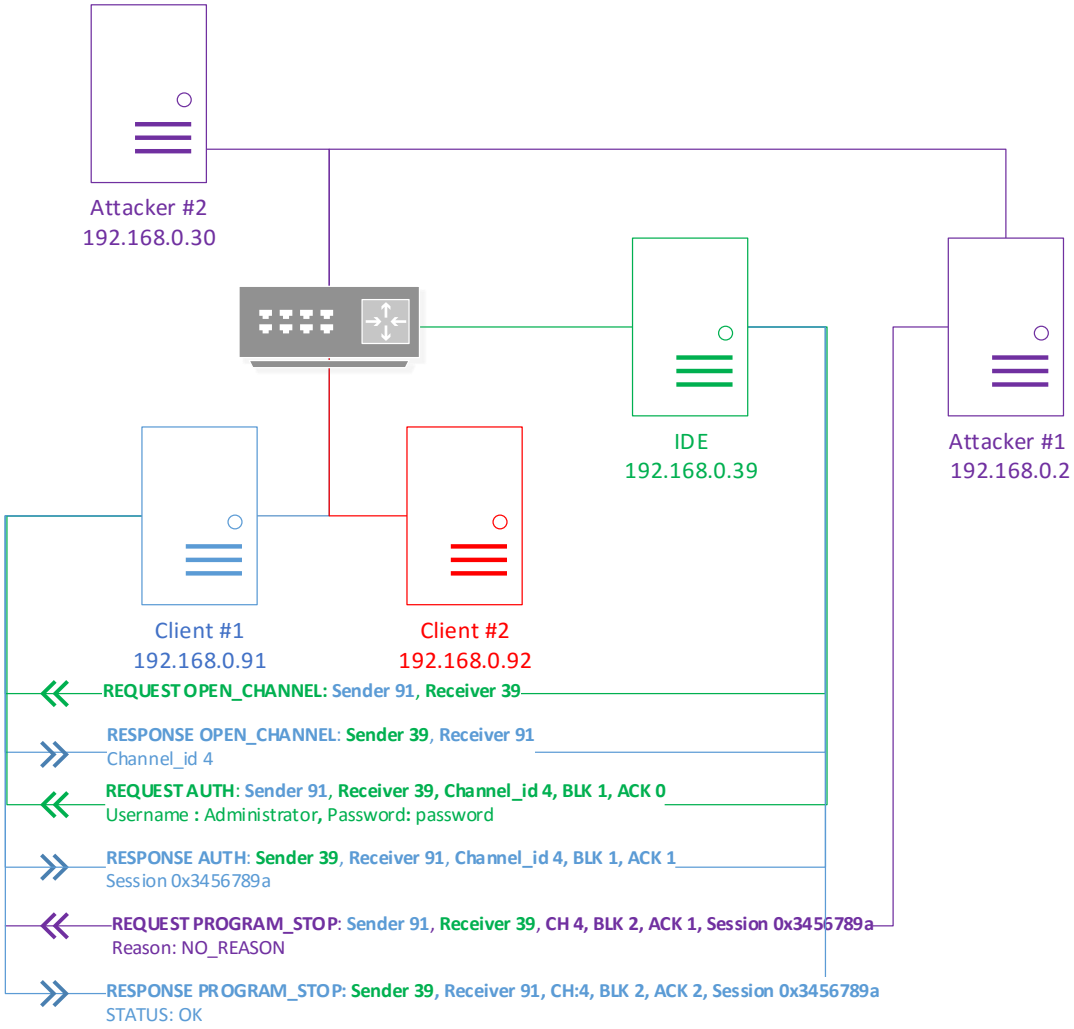


Схема атаки с целью перехвата управления

Алгоритм атаки может быть следующим:

1. **Запрос OPEN_CHANNEL**:
   Узел IDE запрашивает открытие канала на узле Client #1.
2. **Ответ OPEN_CHANNEL**:
   Узел Client #1 открывает канал связи узлу IDE и возвращает ему идентификатор канала (channel_id), равный 4.
3. **Запрос AUTH**:
   Узел IDE аутентифицируется на узле Client #1, передавая ему пароль (Password) и имя пользователя (Username).
4. **Ответ AUTH**:
   Узел Client #1 ищет в базе данных запись о полученном пользователе. После нахождения записи и сравнении пароля Client #1 возвращает узлу IDE идентификатор сессии (Session), равный 0x3456789a.

5. **Запрос PROGRAM_STOP**:
   Злоумышленник от узла Attacker #1 отправляет на узел Client #1 сообщение с командой PROGRAM_STOP. В этом пакете должен быть указан инкрементированный идентификатор BLK из последнего сообщения узла IDE и идентификатор ACK, аналогичный идентификатору в последнем сообщении узла IDE. Идентификаторы сессии (session_id) и канала (channel_id) совпадают с идентификаторами, используемыми в коммуникации между узлами.
6. **Ответ PROGRAM_STOP**:
   Узел Client #1 обрабатывает полученный от узла Attacker #1 запрос с командой PROGRAM_STOP, как если бы этот запрос пришел от узла IDE, потому что полученный идентификатор канала связи (channel_id) совпадает с существующим идентификатором канала связи между узлами IDE и Client #1, полученные идентификаторы BLK и ACK являются правильными для используемого канала, а идентификатор сессии существует. В результате узел Client #1 вернет положительный ответ об остановке программы.

Установка произвольного родительского узла

Перехват сетевого трафика является одной из угроз для канального уровня модели ISO/OSI (data link layer). Атаку, которая реализует угрозу перехвата сетевого трафика, называют «человек посередине» (Man in the Middle, MitM).

В протоколе ARP, который работает на канальном уровне ISO/OSI, одна из реализаций атак «человек посередине» – ARP-poising. Эта атака заключается в изменении ARP-таблицы на компьютере жертвы путем отправки специально сформированных ARP-ответов на сетевые узлы жертв. После изменений ARP-таблицы, весь исходящий трафик и входящий трафик жертвы будет проходить через сетевой адрес атакующего.

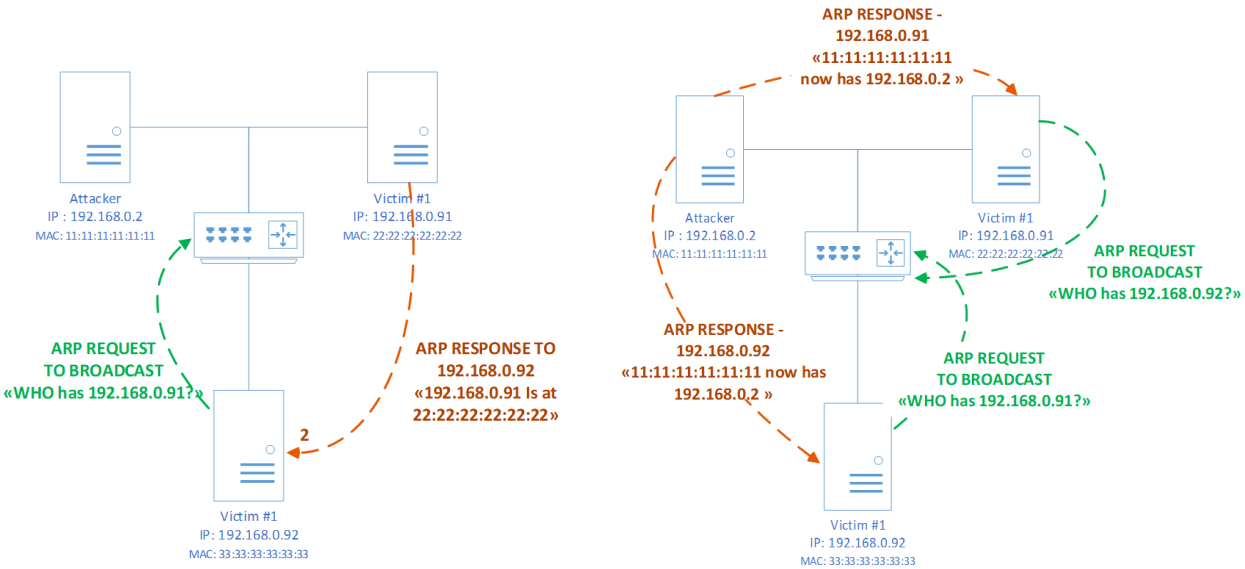


Схема атаки ARP-poisoning по протоколу ARP. Шаг 1 – изменение ARP-таблицы

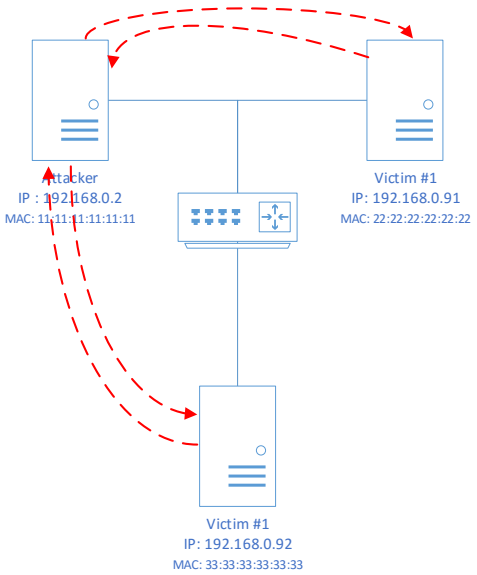


Схема атаки ARP-poisoning по протоколу ARP. Шаг 2 – измененный маршрут сетевого потока

В CODESYS Runtime отсутствует ARP-таблица, однако присутствует механизм смены маршрута сети CODESYS, за который ответственен компонент **CmpRouter**.

Адрес любого узла сети CODESYS состоит из адресов всех предыдущих родительских узлов и своего адреса в качестве конечного. Узел сети CODESYS формирует и запоминает свой полный адрес после получения серии запросов на службу адресов (address service). После формирования адреса узел будет отправлять все исходящие пакеты на источник запроса, считая его родительским узлом.

```
    1 0.000000   192.168.0.39   192.168.0.255   CODESYSV3   62 1742 → 1743 Len=20
    2 0.003809   192.168.0.92   192.168.0.39    CODESYSV3   297 1740 → 1742 Len=255
    3 20.131552  192.168.0.30   192.168.0.255   CODESYSV3   119 1740 → 1740 Len=77
    4 60.022305  192.168.0.39   192.168.0.255   CODESYSV3   62 1742 → 1743 Len=20
    5 60.026594  192.168.0.92   192.168.0.30    CODESYSV3   297 1740 → 1740 Len=255
```

```
> Frame 1: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface 0
> Ethernet II, Src: PcsCompu_90:85:bf (08:00:27:90:85:bf), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
> Internet Protocol Version 4, Src: 192.168.0.39, Dst: 192.168.0.255
> User Datagram Protocol, Src Port: 1742, Dst Port: 1743
> CoDeSys V3 Protocol
```

```
0000  ff ff ff ff ff ff 08 00  27 90 85 bf 08 00 45 00   ........ '.....E.
0010  00 30 00 e3 00 00 80 11  b7 63 c0 a8 00 27 c0 a8   .0...... .c...'..
0020  00 ff 06 ce 06 cf 00 1c  5e 66 c5 74 40 03 00 10   ........ ^f.t@...
0030  63 d3 02 27 00 00 02 c2  00 04 a2 f3 00 00        c..'.... ......
```

Color				
Network address	192.168.0.255 (Broadcast)	192.168.0.39	192.168.0.92	192.168.0.30
Host		IDE	Client #1	Attacker #2

Реализация атаки по установке произвольного родительского узла

На рисунке выше показаны 5 пакетов и содержимое последнего пакета. Ниже дано описание каждого из пакетов и последовательность действий узлов сети CODESYS на нашем стенде:

1. Узел IDE отправляет пакет №1 на **широковещательный адрес** (broadcast). Этот пакет содержит запрос на получение информации об узлах в сети. Сам запрос предназначен для сервиса имен (name service).
2. Узел Client #1 получает пакет №1 с **широковещательного адреса**. Обработав запрос, узел извлекает из поля **receiver** адрес узла, на который необходимо отправить ответ. В значении поля **receiver** указан адрес узла IDE, поэтому узел Client #1 отвечает пакетом №2 узлу IDE. Этот пакет содержит информацию об узле Client #1.
3. Attacker #2 отправляет пакет №3 на **широковещательный адрес**. Этот пакет содержит запрос для сервиса адресов (address service) на изменение родительского адреса (service_id 2). Пакет №3 получает узел Client #1. После обработки пакета №3 узел Client #1 изменяет маршрут для своего сетевого интерфейса и устанавливает адрес Attacker #2 в качестве родительского узла.
4. Узел IDE отправляет на **широковещательный адрес** запрос (пакет №4) на получение информации об узлах в сети. Пакет №4 идентичен пакету №1.
5. Узел Client #1 получает пакет №4 с **широковещательного адреса**. Обработав этот запрос, узел формирует ответ и отправляет его в пакете №5. Этот пакет отправляется уже не на указанный в поле **receiver** адрес узла IDE, а на адрес родительского узла, которым стал узел Attacker #2.

Несмотря на то, что узел Client #1 получил оба запроса на получение информации с одного и того же адреса, а ответы отправил на разные адреса, содержимое этих ответов одинаково. Не изменились в них и значения полей **sender** и **receiver**. То есть, узел с измененной маршрутизацией и установленным родительским узлом ожидает от своего родительского узла доставку отправленного пакета, поэтому узел и не изменяет значения полей **receiver** и **sender**.

Таким образом злоумышленник может изменить маршрут сетевого трафика CODESYS без каких-либо привилегий на стороне узла с запущенным CODESYS Runtime. Сделав свою машину родительским узлом, злоумышленник может внедриться в уже существующий или в будущий поток, реализовав атаку типа «человек посередине».

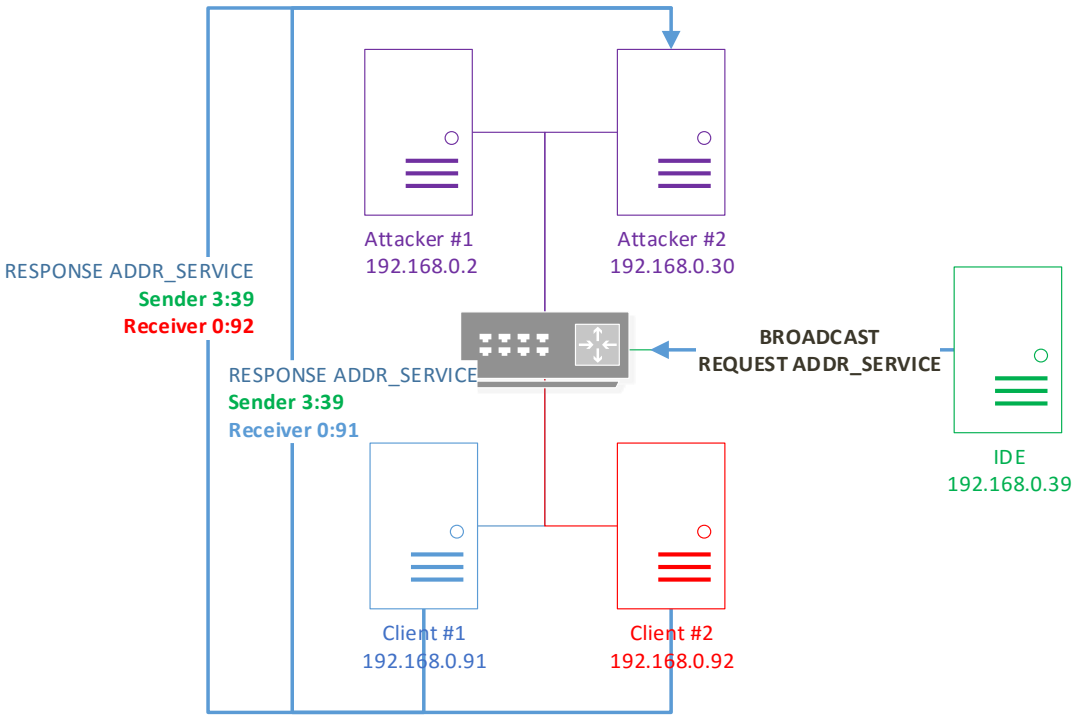


Схема измененного маршрута сетевого потока после установки произвольного родительского узла

Уязвимость в канальном уровне (Channel layer). Предсказуемость идентификатора канала

Изначально мы предполагали, что значение идентификатора созданного канала связи всегда инкрементировалось на четыре от значения последнего идентификатора канала связи. На это указывал выводимый программой лог:

```
1528045861: Cmp=CmpChannelServer, Class=16, Error=0, Info=0, pszInfo= Channel <channelid>4</channelid> connected
1528045863: Cmp=CmpChannelServer, Class=16, Error=0, Info=0, pszInfo= Channel <channelid>4</channelid> closed by request, <error>0</error>
1528045864: Cmp=CmpChannelServer, Class=16, Error=0, Info=0, pszInfo= Channel <channelid>8</channelid> connected
1528045864: Cmp=CmpChannelServer, Class=16, Error=0, Info=0, pszInfo= Channel <channelid>8</channelid> closed by request, <error>0</error>
1528045865: Cmp=CmpChannelServer, Class=16, Error=0, Info=0, pszInfo= Channel <channelid>12</channelid> connected
1528045865: Cmp=CmpChannelServer, Class=16, Error=0, Info=0, pszInfo= Channel <channelid>12</channelid> closed by request, <error>0</error>
1528045865: Cmp=CmpChannelServer, Class=16, Error=0, Info=0, pszInfo= Channel <channelid>16</channelid> connected
1528045865: Cmp=CmpChannelServer, Class=16, Error=0, Info=0, pszInfo= Channel <channelid>16</channelid> closed by request, <error>0</error>
1528045866: Cmp=CmpChannelServer, Class=16, Error=0, Info=0, pszInfo= Channel <channelid>20</channelid> connected
1528045866: Cmp=CmpChannelServer, Class=16, Error=0, Info=0, pszInfo= Channel <channelid>20</channelid> closed by request, <error>0</error>
1528045866: Cmp=CmpChannelServer, Class=16, Error=0, Info=0, pszInfo= Channel <channelid>24</channelid> connected
1528045867: Cmp=CmpChannelServer, Class=16, Error=0, Info=0, pszInfo= Channel <channelid>24</channelid> closed by request, <error>0</error>
1528045867: Cmp=CmpChannelServer, Class=16, Error=0, Info=0, pszInfo= Channel <channelid>28</channelid> connected
1528045867: Cmp=CmpChannelServer, Class=16, Error=0, Info=0, pszInfo= Channel <channelid>28</channelid> closed by request, <error>0</error>
1528045868: Cmp=CmpChannelServer, Class=16, Error=0, Info=0, pszInfo= Channel <channelid>32</channelid> connected
1528045868: Cmp=CmpChannelServer, Class=16, Error=0, Info=0, pszInfo= Channel <channelid>32</channelid> closed by request, <error>0</error>
1528045868: Cmp=CmpChannelServer, Class=16, Error=0, Info=0, pszInfo= Channel <channelid>36</channelid> connected
```

Фрагмент лога ПО CODESYS Runtime

Для подтверждения этой гипотезы мы исследовали функцию **HandleOpenChannelReq**, которая выступает в качестве обработчика на канальном уровне для команды открытия каналов (**OPEN_CHANNEL)**. Сама функция принадлежит компоненту **CmpChannelServer.**

```
Call trace:
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора

Pseudocode:
001: Удалено по требованию вендора
002: {
[...]
064:   Удалено по требованию вендора
065:   {
[...]
086:   Удалено по требованию вендора
087:   {
[...]
094:       Удалено по требованию вендора
095:       Удалено по требованию вендора
096:         Удалено по требованию вендора
097:       Удалено по требованию вендора
[...]
128:       Удалено по требованию вендора
129:       Удалено по требованию вендора
130:       Удалено по требованию вендора
131:       Удалено по требованию вендора
132:       Удалено по требованию вендора
133:       Удалено по требованию вендора
134:       Удалено по требованию вендора
135:     }
[...]
148:   Удалено по требованию вендора
149:   Удалено по требованию вендора
150:   Удалено по требованию вендора
151:   Удалено по требованию вендора
152:   Удалено по требованию вендора
153:   Удалено по требованию вендора
154:   {
155:     Удалено по требованию вендора
156:     Удалено по требованию вендора
157:   }
158: }
```

Декомпилированный псевдокод функции HandleOpenChannelReq компонента CmpChannelServer

Исследовав код функции **HandleOpenChannelReq**, мы обнаружили, что каждый новый идентификатор канала действительно зависит от значения предыдущего идентификатора канала связи, однако он инкрементируется не на фиксированное значение, равное четырем, а на количество единовременно поддерживаемых каналов связи (строка 94).

Функцией обработки событий компонента **CmpChannelServer** устанавливается количество одновременно поддерживаемых каналов в глобальную переменную **s_iMaxServerChannels**.

```
001: Удалено по требованию вендора
002: {
[...]
008:
009:   Удалено по требованию вендора
010:   {
011:     Удалено по требованию вендора
[...]
```

```
023:          Удалено по требованию вендора
024:          Удалено по требованию вендора
025:          Удалено по требованию вендора
026:            Удалено по требованию вендора
027:          Удалено по требованию вендора
028:          Удалено по требованию вендора
029:          {
030:            Удалено по требованию вендора
031:            Удалено по требованию вендора
032:            {
033:              Удалено по требованию вендора
034:              Удалено по требованию вендора
035:            }
036:          }
037:          Удалено по требованию вендора
038:          Удалено по требованию вендора
039:          Удалено по требованию вендора
040:          Удалено по требованию вендора
041:          Удалено по требованию вендора
042:          Удалено по требованию вендора
[...]
124:    Удалено по требованию вендора
125: }
```

Дизассемблированный код функции CmpChannelServer_hook

Из строки 24 псевдокода функции **CmpChannelServer_hook** видно, что значение глобальной переменной **s_iMaxServerChannels** регулируется параметрами конфигурационного файла. В случае отсутствия секции с именем CmpChannelServer и параметра с именем MaxChannels, в переменной **s_iMaxServerChannels** устанавливается значение по умолчанию, которое равно четырем.

Злоумышленник может получить значение переменной **s_iMaxServerChannels**, обратившись с командой **GET_INFO** к менеджеру канала связи на канальном уровне (channel layer) или же с любой командой для службы имен (name service) на уровне датаграм (datagram layer). Для выполнения этих команд не требуются привилегии.

```
Call trace:
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора
        Удалено по требованию вендора

Pseudocode:
01: Удалено по требованию вендора
02: {
03:   Удалено по требованию вендора
04:
05:   Удалено по требованию вендора
06:   Удалено по требованию вендора
07:   Удалено по требованию вендора
08:   Удалено по требованию вендора
09:   Удалено по требованию вендора
10: }
```

Дизассемблированный псевдокод функции HandleInfoReq

Функция **HandleInfoReq** обрабатывает команду **GET_INFO** для менеджера канала связи на канальном уровне. Глобальная переменная **s_iMaxServerChannels** будет записана в последние два байта ответа (строка 08).

В зависимости от используемых настроек в файле конфигурации CODESYS Runtime может не обрабатывать информационные запросы на уровне датаграм (datagram layer) и на канальном уровне (channel layer). Но злоумышленник всегда может отправить несколько парных запросов на открытие и закрытие канала. Разница между двумя подряд полученными идентификаторами каналов будет однозначно идентифицировать количество одновременно поддерживаемых каналов на узле CODESYS Runtime.

Уязвимости уровня служб (services layer)

Уязвимости в системе аутентификации

В этой главе будут рассмотрены две обнаруженные уязвимости в системе аутентификации: в генерации идентификатора сессии и в шифровании пароля. Эксплуатация первой уязвимости приводит к предугадыванию идентификатора сессии. Эксплуатация второй уязвимости приводит к расшифровке перехваченного пароля.

Уязвимость в предсказуемости генерации идентификатора сессии

Для выполнения большинства служб узлу необходимо пройти аутентификацию. Обработку запроса на прохождение аутентификации выполняет функция **DeviceServiceHandler**, которая зарегистрирована компонентом **CmpDevice** в качестве службы с идентификатором 1. Функция **DeviceServiceHandler** рассматривалась в качестве примера обработки тегов в главе «Обработка тегов».

При обработке функцией **DeviceServiceHandler** входящего запроса с командой, идентификатор которой равен 2, функция **DeviceServiceHandler** передает управление в функцию **ServerGenerateSessionId** (будет рассмотрена далее).

```
001: Удалено по требованию вендора
002: {
[...]
129:   Удалено по требованию вендора
130:   Удалено по требованию вендора
131:   Удалено по требованию вендора
132:   Удалено по требованию вендора
133:   Удалено по требованию вендора
134:   {
[...]
254:     Удалено по требованию вендора
[...]
262:       Удалено по требованию вендора
263:       Удалено по требованию вендора
[...]
365:         Удалено по требованию вендора
366:       }
367:       Удалено по требованию вендора
[...]
389:       Удалено по требованию вендора
390:       Удалено по требованию вендора
391:       Удалено по требованию вендора
392:       Удалено по требованию вендора
393:       Удалено по требованию вендора
394:       Удалено по требованию вендора
395:       Удалено по требованию вендора
396:       Удалено по требованию вендора
397:       Удалено по требованию вендора
398:       Удалено по требованию вендора
[...]
761:   }
762:   Удалено по требованию вендора
763: }
```

Фрагмент декомпилированного псевдокода функции DeviceServiceHandler компонента CmpDevice

В ответе на успешное выполнение запроса на прохождение аутентификации содержится тег данных с идентификатором 0x21 (строка 392). В данных этого тега находится сгенерированный идентификатор созданной сессии (строка 393). Сам же идентификатор сессии создается внутри функции **HandleLoginSessionId** еще до проверки полученных данных аутентификации (строка 263). Сгенерированный идентификатор возвращается в переменной **ulSessionId**.

```
01: Удалено по требованию вендора
02: {
[...]
09:   Удалено по требованию вендора
10:   Удалено по требованию вендора
11:   Удалено по требованию вендора
12:   {
13:     Удалено по требованию вендора
14:     Удалено по требованию вендора
15:   }
16:   Удалено по требованию вендора
17:   {
18:     *Удалено по требованию вендора
19:     Удалено по требованию вендора
20:   }
21:   Удалено по требованию вендора
22:   {
23:     *Удалено по требованию вендора
24:     Удалено по требованию вендора
25:   }
26:   Удалено по требованию вендора
27:   Удалено по требованию вендора
28: }
```

Фрагмент декомпилированного псевдокода функции HandleLoginSessionId

Внутри функции **HandleLoginSessionId** вызывается функция **ServerGenerateSessionId** (строка 13), которая генерирует идентификатор сессии. Вызов этой функции происходит при условии, что полученный идентификатор сессии равен одному из чисел: 0x0, 0x11 или 0x815. Дополнительным условием создания идентификатора сессии является то, что для полученного идентификатора канала не должно уже существовать идентификатора сессии (строка 10).

```
01: Удалено по требованию вендора
02: {
03:   Удалено по требованию вендора
04:
05:   Удалено по требованию вендора
06:   Удалено по требованию вендора
07: }
08:
09: Удалено по требованию вендора
10: {
11:   Удалено по требованию вендора
[...]
15:   Удалено по требованию вендора
16:     Удалено по требованию вендора
17:   Удалено по требованию вендора
18:   Удалено по требованию вендора
19:   *Удалено по требованию вендора
20:   *Удалено по требованию вендора
21:   Удалено по требованию вендора
22: }
```

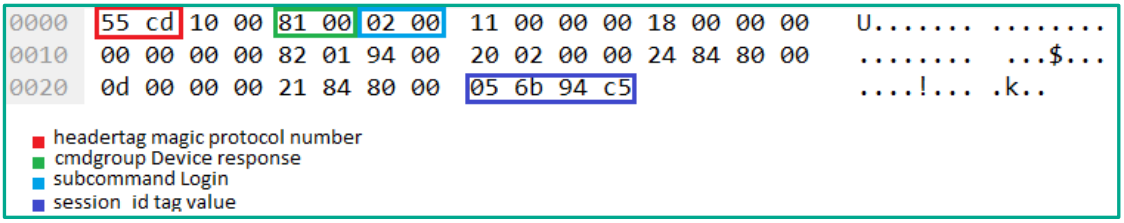
Фрагменты декомпилированного псевдокода функции SysTimeGetMs компонента SysTimer и декомпилированного псевдокода функции ServerGenerateSessionId компонента CmpSrv

Функция **ServerGenerateSessionId** экспортируется компонентом **CmpSrv**. Её алгоритм описан ниже:

1. Проверить наличие указателя аргумента (строки 15 и 16);
2. Получить текущее время через вызов функции **SysTimeGetMs** (строка 17);
3. Инициализировать генератор псевдослучайных чисел. В качестве значения seed для генератора выставить полученное на предыдущем шаге значение (строка 18);
4. Сложить полученное значение времени со случайным числом и записать его в значение по указателю аргумента (строка 19);
5. Установить старший бит в значение по указателю аргумента (строка 20).

Данный алгоритм генерации использует генератор псевдослучайный чисел. Это значит, что сгенерированное случайно число зависит от установленного значения seed и может быть восстановлено. Злоумышленник может перебирать идентификаторы сессий, декрементируя значения seed и воссоздавая сам идентификатор сессии для измененного seed. Такой подбор может быть осуществлен за адекватный промежуток времени, даже несмотря на то, что атака будет происходит удаленно.

Вторая слабость этого алгоритма заключается в использовании функции **SysTimeGetMs.** Эта функция экспортируется системным компонентом **SysTimer**. Ввиду того, что для корректного запуска CODESYS Runtime разработчику необходимо реализовать все системные компоненты, представленная реализация функции **SysTimeGetMs** может отличаться от реализации этой же функции в другом CODESYS Runtime. С точки зрения безопасности такая реализация функции, которая генерирует сессионный идентификатор, накладывает дополнительную ответственность на разработчика, который будет адаптировать системные компоненты, в том числе компонент **SysTimer**.



Color				
fields	protocol_id	service_group	service_id	Session id
Value	0xcd55 (HeaderTagProtocol)	0x81 (CmpDevice, Response)	0x2 (Login)	0xc5946b05

Пример фрагмента пакета, содержащего сгенерированный идентификатор сессии (session_id)

Например, в аналогичном приведенному в главе «Обработка тегов» примере ответа на запрос на аутентификацию, фрагмент ответа содержит тег данных с параметром идентификатора сессии, а для генерации значения идентификатора сессии, равного 0xc5946b05, был использован seed со значением 356267299.

Уязвимости в шифровании пароля

В зарегистрированной компонентом **CmpDevice** функции **DeviceServiceHandler** в качестве службы было обнаружено несколько уязвимостей в механизме шифрования пароля. Эксплуатация этих уязвимостей приводит к расшифровке перехваченного пароля.

Обработку запроса на прохождение аутентификации выполняет функция **DeviceServiceHandler**, которая зарегистрирована компонентом **CmpDevice** в качестве службы с идентификатором 1. Функция **DeviceServiceHandler** рассматривалась в качестве примера обработки тегов в главе «Обработка тегов».

Для прохождения аутентификации CODESYS Development System необходимо отправить запрос для службы **DeviceServiceHandler** с командой, идентификатор которой равен 2. В отправленных данных для прохождения аутентификации будет передан зашифрованный пароль, который во время обработки службой будет расшифрован.

```
001: Удалено по требованию вендора
002: {
[...]
130:   Удалено по требованию вендора
[...]
133:   Удалено по требованию вендора
134:   {
[...]
254:     Удалено по требованию вендора
[...]
266:       Удалено по требованию вендора
267:       Удалено по требованию вендора
268:       {
269:         Удалено по требованию вендора
270:         Удалено по требованию вендора
271:         {
272:           Удалено по требованию вендора
273:             Удалено по требованию вендора
274:             Удалено по требованию вендора
275:           Удалено по требованию вендора
276:             Удалено по требованию вендора
277:             Удалено по требованию вендора
278:             {
279:               Удалено по требованию вендора
280:               Удалено по требованию вендора
281:               {
282:                 Удалено по требованию вендора
283:               }
284:               Удалено по требованию вендора
285:               {
286:                 Удалено по требованию вендора
287:                 Удалено по требованию вендора
288:               }
289:               Удалено по требованию вендора
290:               {
291:                 Удалено по требованию вендора
292:               }
293:               Удалено по требованию вендора
294:               Удалено по требованию вендора
295:               Удалено по требованию вендора
296:             }
297:             Удалено по требованию вендора
298:           Удалено по требованию вендора
299:             Удалено по требованию вендора
300:             Удалено по требованию вендора
301:           Удалено по требованию вендора
302:             Удалено по требованию вендора
303:             Удалено по требованию вендора
304:         }
305:         Удалено по требованию вендора
306:         Удалено по требованию вендора
307:         Удалено по требованию вендора
308:       }
[...]
315:       Удалено по требованию вендора
316:       {
317:         Удалено по требованию вендора
318:         Удалено по требованию вендора
```

```
[...]
327:       }
[...]
761:   }
762:   Удалено по требованию вендора
763: }
```

Фрагмент декомпилированного кода функции DeviceServiceHandler компонента CmpDevice

Расшифровка полученного пароля осуществляется в функции **UserMgrDecryptPassword** (строка 318). В качестве аргументов эта функция использует следующие значения:

1. **encrypted_password** – значение зашифрованного пароля, который извлекается из тега данных с идентификатором 17 (строка 286);
2. **pulCrypeType** – идентификатор алгоритма шифрования, которым был зашифрован переданный пароль. Значение идентификатора извлекается из тега данных с идентификатором 0x22 (строка 299);
3. **pulChallenge** – значение случайного числа (nonce), которое участвует в шифровании пароля. Значение случайного числа извлекается из тега данных с идентификатором 0x23.

```
01: Удалено по требованию вендора
02: {
[...]
16:   Удалено по требованию вендора
17:     Удалено по требованию вендора
18:   Удалено по требованию вендора
19:   Удалено по требованию вендора
20:   Удалено по требованию вендора
21:   Удалено по требованию вендора
22:   {
23:     Удалено по требованию вендора
24:     {
25:       Удалено по требованию вендора
26:       Удалено по требованию вендора
27:       Удалено по требованию вендора
28:       Удалено по требованию вендора
29:       Удалено по требованию вендора
30:       Удалено по требованию вендора
31:       {
32:         Удалено по требованию вендора
33:         Удалено по требованию вендора
34:         Удалено по требованию вендора
35:           Удалено по требованию вендора
36:         Удалено по требованию вендора
37:           Удалено по требованию вендора
38:       }
39:       Удалено по требованию вендора
40:     }
[...]
```

Фрагмент декомпилированного псевдокода функции UserMgrDecryptPassword

Функция **UserMgrDecryptPassword** выполняет следующие действия:

1. Сравнивает значения аргумента **pulCrypeType** с 1 (строка 16). Если эти значения разные, то функция дальше не отрабатывается. Иначе говоря, CODESYS Runtime предоставляет только один алгоритм шифрования для передаваемого пароля, а значение **pulCrypeType** необходимо передавать каждый раз во время аутентификации, несмотря на отсутствие альтернатив в выборе алгоритма шифрования пароля.
2. Записывает фиксированный ключ в локальную переменную **key** (строка 18).

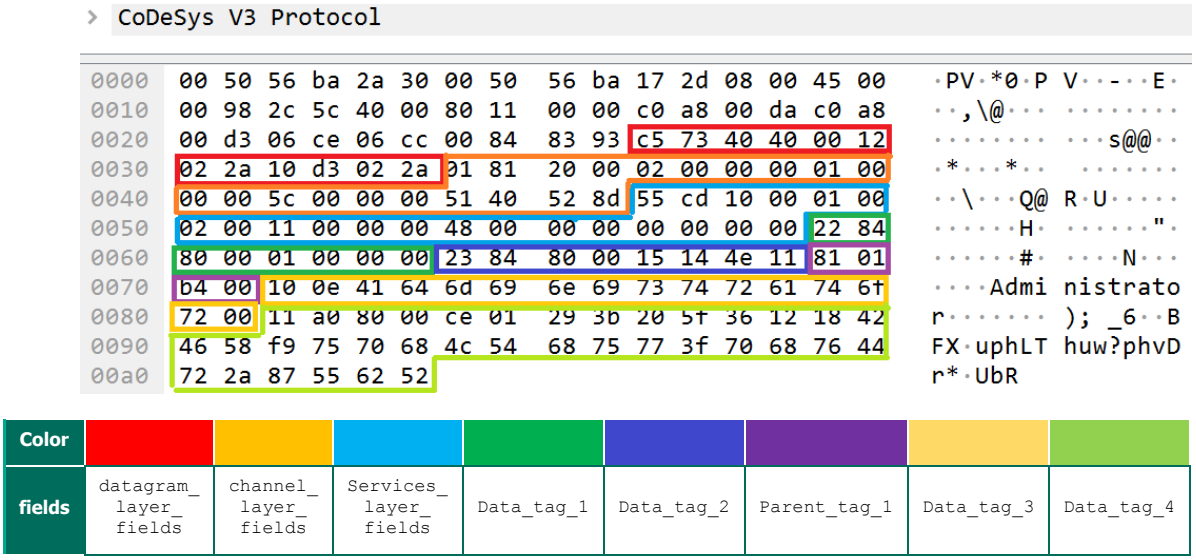
3. Из 4-байтового значения аргумента **pulChallenge** записывает только младший байт в 4-байтовый массив **aChallenge** (строка 25). Для остальных трех байт массива устанавливаются нулевые значения (строки 26-28).
4. Далее функция использует три индекса: **index** – индекс пароля, **index_key** – индекс ключа и **challenge_index** – индекс случайного числа. Индекс пароля определяет окончание расшифровки зашифрованного пароля. Когда индекс пароля будет равен размеру зашифрованного пароля, функция **UserMgrDecryptPassword** завершится и вернет управление функции **DeviceServiceHandler** (строка 32). Остальные два индекса (индекс случайного числа и индекс ключа) будут обнуляться всякий раз, когда размер их переменных (**aChallenge** и **key**) будет равен значению индекса (строка 34 для переменной **key** и строка 36 для переменной **aChallenge**).
5. Далее посимвольно расшифровывается пароль. Каждый расшифрованный символ получается из сложения одного байта, взятого из массива **ulChallenge** по индексу случайного числа (**challenge_index)**, и одного байта, взятого из переменной **key** по индексу ключа (**index_key)**. Для полученного числа выполняется операция XOR с байтом, взятым из аргумента пароля **encrypted_key** по индексу зашифрованного пароля (строка 32).

При успешном выполнении функции **UserMgrDecryptPassword** будет расшифрован полученный пароль, который будет записан в аргумент **decrypted_password**.

Из описанного алгоритма функции можно выделить сразу три уязвимости:

1. **Использование слабого значения случайного числа.** Несмотря на то, что зарегистрированная компонентом **CmpDevice** функция **DeviceServiceHandler** для команды прохождения аутентификации извлекает из полученных тегов данных 4-байтовое числовое значение, для расшифровки полученного пароля используется только один байт из четырех. Ввиду того, что алгоритм шифрования симметричен, в шифровании пароля участвует также один байт.
   Использование случайного однобайтового значения для защиты пароля не повышает защищенность передаваемого пароля, т.к. это значение может быть подобрано за очень короткое время.
2. **Использование произвольного значения в качестве случайного числа.** Из рассмотренного алгоритма функции **DeviceServiceHandler** видно, что команда аутентификации службы компонента **CmpDevice** в качестве случайного числа (**aChallenge**) для расшифровки пароля использует полученный числовой параметр от проходящего аутентификацию узла.
   Другими словами, несмотря на то что при получении запроса на прохождение аутентификации сторона, на которой проходят аутентификацию, отправляет случайное число, которое должно быть использовано в шифровании пароля, другая сторона, которая проходит аутентификацию, может зашифровать пароль со своим числом и передать это число для расшифровки.
   Это означает, что злоумышленник, единожды перехватив данные аутентификации, может повторно отправлять их без изменений для прохождения аутентификации.
3. **Использование фиксированного ключа.** Рассмотренный алгоритм расшифровки использует фиксированный ключ для шифрования и расшифровки пароля. Это означает, что злоумышленник, получивший зашифрованные данные аутентификации, всегда может расшифровать перехваченный пароль.

На примере рассмотренного в главе «Обработка тегов» пакета с запросом на прохождение аутентификацию мы покажем, как можно частично расшифровать передаваемый пароль.



Пакет с запросом на прохождение аутентификации

Зашифрованный пароль передается в теге данных с идентификатором 0x11, который обозначен как **Data_tag_4.** Если извлечь данные зашифрованного пароля из тега данных и для каждого байта данных выполнить операцию XOR с байтами ключа по таким же индексам, то можно частично восстановить пароль.

```
1: encrypted_password =
"\xce\x01\x29\x3b\x20\x5f\x36\x12\x18\x42\x46\x58\xf9\x75\x70\x68\x4c\x54\x68\x75\x77\x3f\x70\x68\x76\x44\
x72\x2a\x87\x55\x62\x52"
2: KEY = "zeDR96EfU#27vuph7Thub?phaDr*rUbR"
3: for c, s in enumerate(encrypted_password):
4:     print chr(ord(KEY[c]) ^ ord(encrypted_password[c])),
5:
6: d m i  i s t M a t o
```

Скрипт на языке программирования Python для частичной расшифровки пароля

Частично восстановленный пароль содержит символы d, m, I, I, s, t, a, t, o (строка 6). Эти символы входят в строку “Administrator”, которая является паролем по умолчанию для пользователя Administrator.

Уязвимость кода приложения

Одной из задач, которую выполняет CODESYS Runtime, является загрузка, управление и исполнения приложений. CODESYS Development System компилирует приложение для CODESYS Runtime и загружает его по протоколу CODESYS PDU. Загружаемое приложение представляет собой поток бинарных данных.

Во время нашего исследования мы не фокусировались на исследовании структуры скомпилированного приложения. (Отметим, что при поддержке Организации по управлению военно-морских исследований США (U.S. Office of Naval Research) была проведена исследовательская работа по анализу содержимого скомпилированного приложения для CODESYS Runtime. Результатом этой работы стала разработка фреймворка ICSREF.)

Мы же, изучив содержимое отправляемых пакетов от CODESYS Development System на CODESYS Runtime во время загрузки приложения, обнаружили места в бинарном потоке, куда можно внедрить **произвольный машинный код** – shellcode.

Лазейками стали две функции: функция инициализации глобальных переменных (в упомянутой работе названа **Global INIT**) и стартовая функция программы (в упомянутой работе названа **PLC_PRG**).

```
Header:
1: PROGRAM PLC_PRG
2: VAR
3:        magic: DWORD:= 16#DEADBEEF;
4: END_VAR

Body:
5: magic := magic + 16#BEEF;
```

Исходный код программы PLC_PRG

Чтобы подтвердить возможность внедрения **произвольного машинного кода с целью его исполнения в бинарный поток приложения**, мы, используя CODESYS Development System, скомпилировали программу и удаленно загрузили ее на устройство Raspberry PI с запущенным CODESYS Runtime. В блоке объявления переменных была объявлена переменная **magic** со значением **0xDEADBEEF** (строка 3). В блоке тела программы указано, что значение переменной **magic** необходимо постоянно складывать со значением **0xBEEF** и записывать получившийся результат в переменную **magic** (строка 5).

```
00c0  00 0a 0b 00 00 ea 6c 40  99 e5 08 50 a0 e3 04 50
00d0  85 e0 05 50 d9 e7 0a 00  55 e3 05 00 00 1a 01 40
00e0  a0 e3 0c 40 ca e5 6c 40  99 e5 01 40 84 e2 6c 40
00f0  89 e5 ff ff ff ea 30 42  bd e8 08 d0 8d e2 00 84
0100  bd e8 00 00 00 60 a0 01  c0 00 21 06 03 00 28 15
0110  01 00 22 b4 80 00 30 00  00 00 00 44 2d e9 0d a0
0120  a0 e1 30 00 2d e9 18 b0  9f e5 00 40 9b e5 0c 50
0130  9f e5 05 40 84 e0 00 40  8b e5 30 00 bd e8 00 84
0140  bd e8 ef be 00 00 70 38  00 00 a0 01 f0 08 21 06
0150  03 00 58 15 01 00 22 e4  88 00 60 04 00 00 00 44
0160  2d e9 0d a0 a0 e1 20 d0  4d e2 71 00 2d e9 00 40
0170  a0 e3 10 40 ca e5 00 40  a0 e3 20 40 0a e5 00 40
0180  a0 e3 1c 40 0a e5 00 40  a0 e3 18 40 0a e5 1f 40
0190  a0 e3 14 40 4a e5 1c 40  a0 e3 13 40 4a e5 1f 40
01a0  a0 e3 12 40 4a e5 1e 40  a0 e3 11 40 4a e5 1f 40
01b0  a0 e3 10 40 4a e5 1e 40  a0 e3 0f 40 4a e5 1f 40
01c0  a0 e3 0e 40 4a e5 1f 40  a0 e3 0d 40 4a e5 1e 40
01d0  a0 e3 0c 40 4a e5 1f 40  a0 e3 0b 40 4a e5 1e 40
01e0  a0 e3 0a 40 4a e5 1f 40  a0 e3 09 40 4a e5 0c 40
01f0  9a e5 00 50 94 e5 08 50  0a e5 10 d0 4d e2 08 40
0200  9a e5 00 40 8d e5 a8 43  9f e5 04 40 8d e5 9c b3
0210  9f e5 00 40 9b e5 0d 00  a0 e1 04 a0 2d e5 88 a3
0220  9f e5 04 a0 2d e5
```

```
00c0  00 00 60 4b 00 00 40 5f  00 00 00 00 00 60 a0 01
00d0  c8 00 21 06 03 00 e0 89  01 00 22 bc 80 00 38 00
00e0  00 00 00 44 2d e9 0d a0  a0 e1 08 d0 4d e2 10 08
00f0  2d e9 00 40 a0 e3 09 40  ca e5 00 40 a0 e3 08 40
0100  0a e5 00 40 a0 e3 04 40  4a e5 10 08 bd e8 08 d0
0110  8d e2 00 84 bd e8 00 00  00 60 a0 01 c8 00 21 06
0120  03 00 18 8a 01 00 22 bc  80 00 38 00 00 00 00 44
0130  2d e9 0d a0 a0 e1 08 d0  4d e2 10 08 2d e9 00 40
0140  a0 e3 09 40 ca e5 00 40  a0 e3 08 40 0a e5 00 40
0150  a0 e3 04 40 4a e5 10 08  bd e8 08 d0 8d e2 00 84
0160  bd e8 00 00 00 60 a0 01  d8 00 21 06 03 00 50 8a
0170  01 00 22 cc 80 00 48 00  00 00 00 44 2d e9 0d a0
0180  a0 e1 08 d0 4d e2 10 08  2d e9 00 40 a0 e3 09 40
0190  ca e5 00 40 a0 e3 08 40  0a e5 00 40 a0 e3 04 40
01a0  4a e5 14 40 9f e5 0c b0  9f e5 00 40 8b e5 10 08
01b0  bd e8 08 d0 8d e2 00 84  bd e8 70 38 00 00 ef be
01c0  ad de a0 01 d8 00 21 06  03 00 98 8a 01 00 22 cc
01d0  80 00 48 00 00 00 00 44  2d e9 0d a0 a0 e1 08 d0
01e0  4d e2 10 08 2d e9 00 40  a0 e3 09 40 ca e5 00 40
01f0  a0 e3 08 40 0a e5 00 40  a0 e3 04 40 4a e5 00 40
0200  a0 e3 0c b0 9f e5 00 40  8b e5 10 08 bd e8 08 d0
0210  8d e2 00 84 bd e8 7c 38  00 00 00 00 00 60 a0 01
0220  98 05 21 06 03 00
```

Фрагмент трафика при загрузке приложения с упоминаниями использования байт EF BE

После загрузки скомпилированной программы на устройство Raspberry Pi в трафике был произведен поиск байт EF BE. Эти байты содержатся в числах **0xDEADBEEF** и **0xBEEF** при порядке байт в числах от младшего к старшему (little-endian). Всего было обнаружено два упоминания этих байт в трафике (выделено красным). Необходимо сказать, что в одном пакете рядом с искомыми байтами EF BE были обнаружены байты AD DE (выделено синим).

Далее весь загружаемый поток бинарных данных был проанализирован на наличие машинных инструкций процессора ARM, на котором работает Raspberry Pi. По поиску тех же байт EF BE была обнаружена инструкция с обращением к числу **0xDEADBEEF** и инструкция с обращением к числу **0xBEEF**.

Рассмотрим инструкции первого обращения.

```
01: 00 00 00 60       ANDVS           R0, R0, R0
02: A0 01 D8 00       SBCEQS          R0, R8, R0,LSR#3
03: 21 06 03 00       ANDEQ           R0, R3, R1,LSR#12
04: 50 8A 01 00       ANDEQ           R8, R1, R0,ASR R10
05: 22 CC 80 00       ADDEQ           R12, R0, R2,LSR#24
06: 48 00 00 00       ANDEQ           R0, R0, R8,ASR#32
07: 00 44 2D E9       STMFD           SP!, {R10,LR}
08: 0D A0 A0 E1       MOV             R10, SP
09: 08 D0 4D E2       SUB             SP, SP, #8
10: 10 08 2D E9       STMFD           SP!, {R4,R11}
11: 00 40 A0 E3       MOV             R4, #0
12: 09 40 CA E5       STRB            R4, [R10,#9]
13: 00 40 A0 E3       MOV             R4, #0
14: 08 40 0A E5       STR             R4, [R10,#-8]
15: 00 40 A0 E3       MOV             R4, #0
16: 04 40 4A E5       STRB            R4, [R10,#-4]
17: 14 40 9F E5       LDR             R4, =0xDEADBEEF
18: 0C B0 9F E5       LDR             R11, =0x3870
19: 00 40 8B E5       STR             R4, [R11]
20: 10 08 BD E8       LDMFD           SP!, {R4,R11}
21: 08 D0 8D E2       ADD             SP, SP, #8
22: 00 84 BD E8       LDMFD           SP!, {R10,PC}
```

Обнаруженные ассемблерные инструкции для процессора ARM с обращением к числу 0xDEADBEEF

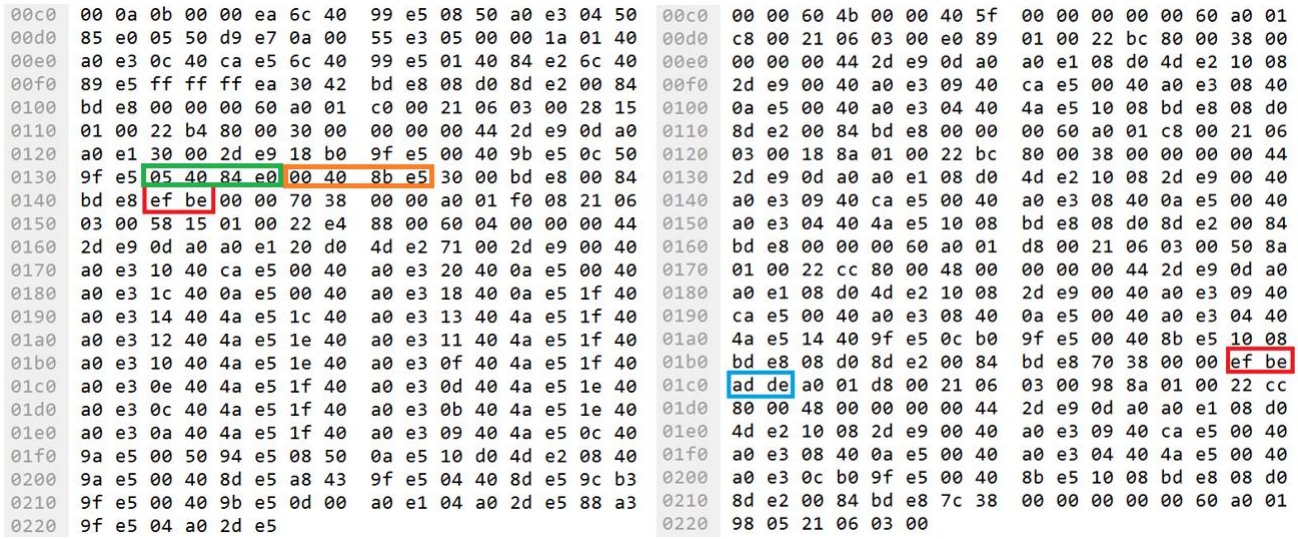
На строке 17 в регистр R4 записывается константа **0xDEADBEEF**. На строке 18 в регистр R11 записывается константа **0x3870.** На строке 19 значение регистра R4 (**0xDEADBEEF**) записывается по адресу значения регистра R11 (**0x3870**).

```
01: 00 00 00 60       ANDVS           R0, R0, R0
02: A0 01 C0 00       SBCEQ           R0, R0, R0,LSR#3
03: 21 06 03 00       ANDEQ           R0, R3, R1,LSR#12
04: 28 15 01 00       ANDEQ           R1, R1, R8,LSR#10
05: 22 B4 80 00       ADDEQ           R11, R0, R2,LSR#8
06: 30 00 00 00       ANDEQ           R0, R0, R0,LSR R0
07: 00 44 2D E9       STMFD           SP!, {R10,LR}
08: 0D A0 A0 E1       MOV             R10, SP
09: 30 00 2D E9       STMFD           SP!, {R4,R5}
10: 18 B0 9F E5       LDR             R11, =0x3870
11: 00 40 9B E5       LDR             R4, [R11]
12: 0C 50 9F E5       LDR             R5, =0xBEEF
13: 05 40 84 E0       ADD             R4, R4, R5
14: 00 40 8B E5       STR             R4, [R11]
15: 30 00 BD E8       LDMFD           SP!, {R4,R5}
16: 00 84 BD E8       LDMFD           SP!, {R10,PC}
```

Обнаруженные ассемблерные инструкции для процессоар ARM с обращением к числу 0xBEEF

В отличие от машинного кода первого обращения, машинный код второго обращения действует от обратно. Сначала происходит запись константы **0x3870** в регистр R11 (строка 10). Далее в регистр R4 записывается содержимое ячейки памяти по адресу константы **0x3870** (строка 11). После происходит запись константы **0xBEEF** в регистр R5 (строка 12). Значение регистра R5 (**0xBEEF**) суммируется со значением по адресу **0x3870** (строка 13). Результат суммы сохраненного значения из памяти и константы **0xBEEF** записывается в регистр R4, значение которого потом записывается в ячейку памяти по адресу **0x3870**.

Исходя из рассмотренных обращений можно предположить, что по адресу **0x3870** находится адрес объявленной глобальной переменной **magic**. Значение **0xDEADBEEF** записывается в переменную **magic** в рассмотренном машинном коде первого найденного обращения. Во втором найденном обращении в переменную **magic** прибавляется число **0xBEEF**. Результат сложения снова записывается по адресу глобальной переменной **magic** (**0x3870**). Оба фрагмента машинного кода соответствуют исходному коду программы **PLC_PRG**.



```
00c0  00 0a 0b 00 00 ea 6c 40  99 e5 08 50 a0 e3 04 50
00d0  85 e0 05 50 d9 e7 0a 00  55 e3 05 00 00 1a 01 40
00e0  a0 e3 0c 40 ca e5 6c 40  99 e5 01 40 84 e2 6c 40
00f0  89 e5 ff ff ff ea 30 42  bd e8 08 d0 8d e2 00 84
0100  bd e8 00 00 00 60 a0 01  c0 00 21 06 03 00 28 15
0110  01 00 22 b4 80 00 30 00  00 00 00 44 2d e9 0d a0
0120  a0 e1 30 00 2d e9 18 b0  9f e5 00 40 9b e5 0c 50
0130  9f e5 05 40 84 e0 00 40  8b e5 30 00 bd e8 00 84
0140  bd e8 ef be 00 00 70 38  00 00 a0 01 f0 08 21 06
0150  03 00 58 15 01 00 22 e4  88 00 60 04 00 00 00 44
0160  2d e9 0d a0 a0 e1 20 d0  4d e2 71 00 2d e9 00 40
0170  a0 e3 10 40 ca e5 00 40  a0 e3 20 40 0a e5 00 40
0180  a0 e3 1c 40 0a e5 00 40  a0 e3 18 40 0a e5 1f 40
0190  a0 e3 14 40 4a e5 1c 40  a0 e3 13 40 4a e5 1f 40
01a0  a0 e3 12 40 4a e5 1e 40  a0 e3 11 40 4a e5 1f 40
01b0  a0 e3 10 40 4a e5 1e 40  a0 e3 0f 40 4a e5 1f 40
01c0  a0 e3 0e 40 4a e5 1f 40  a0 e3 0d 40 4a e5 1e 40
01d0  a0 e3 0c 40 4a e5 1f 40  a0 e3 0b 40 4a e5 1e 40
01e0  a0 e3 0a 40 4a e5 1f 40  a0 e3 09 40 4a e5 0c 40
01f0  9a e5 00 50 94 e5 08 50  0a e5 10 d0 4d e2 08 40
0200  9a e5 00 40 8d e5 a8 43  9f e5 04 40 8d e5 9c b3
0210  9f e5 00 40 9b e5 0d 00  a0 e1 04 a0 2d e5 88 a3
0220  9f e5 04 a0 2d e5

00c0  00 00 60 4b 00 00 40 5f  00 00 00 00 00 60 a0 01
00d0  c8 00 21 06 03 00 e0 89  01 00 22 bc 80 00 38 00
00e0  00 00 00 44 2d e9 0d a0  a0 e1 08 d0 4d e2 10 08
00f0  2d e9 00 40 a0 e3 09 40  ca e5 00 40 a0 e3 08 40
0100  0a e5 00 40 a0 e3 04 40  4a e5 10 08 bd e8 08 d0
0110  8d e2 00 84 bd e8 00 00  00 60 a0 01 c8 00 21 06
0120  03 00 18 8a 01 00 22 bc  80 00 38 00 00 00 00 44
0130  2d e9 0d a0 a0 e1 08 d0  4d e2 10 08 2d e9 00 40
0140  a0 e3 09 40 ca e5 00 40  a0 e3 08 40 0a e5 00 40
0150  a0 e3 04 40 4a e5 10 08  bd e8 08 d0 8d e2 00 84
0160  bd e8 00 00 00 60 a0 01  d8 00 21 06 03 00 50 8a
0170  01 00 22 cc 80 00 48 00  00 00 00 44 2d e9 0d a0
0180  a0 e1 08 d0 4d e2 10 08  2d e9 00 40 a0 e3 09 40
0190  ca e5 00 40 a0 e3 08 40  0a e5 00 40 a0 e3 04 40
01a0  4a e5 14 40 9f e5 0c b0  9f e5 00 40 8b e5 10 08
01b0  bd e8 08 d0 8d e2 00 84  bd e8 70 38 00 00 ef be
01c0  ad de a0 01 d8 00 21 06  03 00 98 8a 01 00 22 cc
01d0  80 00 48 00 00 00 00 44  2d e9 0d a0 a0 e1 08 d0
01e0  4d e2 10 08 2d e9 00 40  a0 e3 09 40 ca e5 00 40
01f0  a0 e3 08 40 0a e5 00 40  a0 e3 04 40 4a e5 00 40
0200  a0 e3 0c b0 9f e5 00 40  8b e5 10 08 bd e8 08 d0
0210  8d e2 00 84 bd e8 7c 38  00 00 00 00 00 60 a0 01
0220  98 05 21 06 03 00
```

Фрагмент трафика при загрузке приложения с ассемблерными инструкциями

Машинные инструкции процессора ARM также передаются по протоколу CODESYS PDU. В примере второго обращения инструкции **ADD R4, R4, R5** (строка 13) соответствуют байты 05 40 84 E0 (выделены в пакете зеленым), следующей инструкции **STR R4, [R11]** (строка 14) соответствуют байты 00 40 8B E5 (выделены оранжевым).

Таким образом, злоумышленник может вставить свой машинный код и выполнить его на целевом устройстве. Необходимо сказать, что системные демоны CODESYS Runtime на устройстве Raspberry Pi запускаются в качестве фонового процесса (daemon) от имени пользователя root, который имеет наивысший уровень привилегий в ОС Linux. Поэтому исполняемый произвольный машинный код будет также работать с наивысшими правами в системе. Эмулятор CODESYS Runtime на ОС Windows также запускается с наивысшими правами в системе – SYSTEM.

Итоги

CODESYS Runtime представляет собой сложный и мощный инструмент для разработки программы для ПЛК и управления ПЛК. При этом в CODESYS Runtime реализован эффективный архитектурный подход, который позволяет расширять возможности самого CODESYS Runtime. Протокол общения между средой разработки CODESYS Development System и средой исполнения CODESYS Runtime является многоуровневым, динамичным, а главное – закрытым.

В результате использования закрытых протоколов в конечном счете страдают производители ПЛК, потому что они ограничены в знаниях об используемом ими программного обеспечения. Из-за этого они вынуждены вслепую доверить начальную защиту ПЛК разработчикам фреймворка.

Компания CODESYS сделала множество шагов для повышения безопасности своих продуктов. Однако, как показало проведенное нами исследование безопасности CODESYS Runtime, этих шагов оказалось недостаточно.

В результате исследования безопасности CODESYS Runtime мы обнаружили 15 уязвимостей, о которых сообщили производителю ПО. О 5 из обнаруженных уязвимостей вендору уже было известно (являлись дубликатами), 2 были признаны группой безопасности CODESYS архитектурными особенностями, а остальные 8 были исправлены. Исправленные уязвимости были оценены по шкале CVSS от 5.4 балла до 9.

Примечательно, что одну уязвимость, ранее отмеченную как «архитектурная особенность», компания CODESYS позже все же исправила.

Наше исследование проходило методом «черного ящика», то есть изначально у нас не было никакой информации о CODESYS Runtime, мы получали её из открытых источников информации и в ходе технического исследования.

После изучения передаваемых данных по протоколу CODESYS и сопоставления программного кода CODESYS Runtime с обработкой получаемых данных по сети, нами было обнаружено четыре уязвимости в механизме аутентификации – в уровне служб (в стеке протокола CODESYS это последний уровень из четырех). Этим уязвимостям были присвоены идентификаторы KLCERT-18-037 (CVE-2018-20025) и KLCERT-19-031 (CVE-2019-9013). Их эксплуатация в системе аутентификации приводит к расшифровке передаваемого пароля, позволяет реализовать атаку повторного использования зашифрованных данных аутентификации без их изменения и предугадать идентификатор сессии.

Разработчики CODESYS взяли за основу своего протокола стек протоколов TCP/IP. В результате CODESYS унаследовал некоторые проблемы TCP/IP: на уровне датаграм (второй из четырех уровней) и на канальном уровне (третий из четырех уровней) нами были обнаружены уязвимости, про возможность которых в стеке протоколов TCP/IP было известно еще в 1989 году (см. Security Problems in the TCP/IP Protocol Suite).

На уровне датаграм в стеке протоколов CODESYS мы обнаружили возможность проведения атаки, идентичной IP-spoofing. Этой уязвимости был присвоен идентификатор KLCERT-18-036 (CVE-2018-20026). Автоматизировав эксплуатацию этой уязвимости, злоумышленник мог бы долго скрывать свои действия в сети, манипулируя устройствами с запущенным CODESYS Runtime и заставляя их отправлять вредоносные пакеты друг другу.

Мы также обнаружили возможность проведения атаки на уровень датаграм, которая похожа на хорошо известную атаку ARP spoofing: механизм маршрутизации сети CODESYS позволяет выстроить информационную сеть с топологией дерево из узлов CODESYS Runtime. Отсутствие необходимости прохождения аутентификации для изменения родительского узла приводит к возможности проведения атаки типа «человек посередине». Так, злоумышленник, используя мощности протокола на уровне датаграм, может сообщить узлу CODESYS Runtime, что стал его новым родителем, и в дальнейшем этот узел будет пересылать весь исходящий трафик через нового родителя.

Последний зверь в зоопарке классических уязвимостей – отсутствие песочницы для загружаемой программы. В ходе исследования протокола обнаружилось, что некоторые фрагменты загружаемой программы представляют собой машинные инструкции. Гипотеза о том, что вместо этих инструкций можно внедрить произвольный код (шеллкод) подтвердилась. Этой уязвимости был присвоен идентификатор KLCERT-18-035 (CVE-2018-10612).

Из-за того, что фоновый процесс (daemon) CODESYS Runtime в ОС Linux и сервис CODESYS Runtime в ОС Windows работают с наивысшими привилегиями в системе (root и SYSTEM соответственно), произвольный код будет также выполняться с наивысшими привилегиями. Таким образом, злоумышленнику уже не надо будет производить дополнительные манипуляции с системой или эксплуатировать уязвимости для получения максимальных привилегий.

Как говорит многолетний опыт специалистов по информационной безопасности, подход «security by obscurity» – не лучшая стратегия защиты информации. Это в полной мере касается недокументированных, закрытых протоколов сетевой коммуникации. Рано или поздно протокол будет исследован, а уязвимости в нём найдены. К сожалению, во многих случаях злоумышленники сделают это раньше добропорядочных исследователей. Как минимум, потому что обычно они гораздо лучше мотивированы.

Мы считаем, что все описанные в статье уязвимости, и, возможно, какие-то ещё, могли бы быть обнаружены сообществом специалистов и энтузиастов информационной безопасности ещё на ранних этапах проектирования протокола, его разработки и использования. Если бы спецификация протокола существовала в открытом для потенциальных пользователей доступе, то уязвимости могли бы быть обнаружены при ее обсуждении и анализе и не затронули бы продукты сотен разработчиков, установленные на десятках и сотнях тысяч промышленных объектов.

Сейчас эта работа требует существенно больших затрат и гораздо более специфических знаний, которые, к сожалению, могут оказаться труднодоступными для многих из разработчиков, использующих CODESYS в своих решениях. Возможно, выбранный подход к разработке протокола оказался не самым оптимальным в этом смысле.

Группа безопасности CODESYS Group оперативно и ответственно отреагировала на информацию об обнаруженных уязвимостях.

Мы искренне благодарим CODESYS Group за сотрудничество.

```
pi@raspberrypi:~ $ ./opt/codesys/bin/codesyscontrol.bin -vvvvvvv
CODESYS Control V3.5.12.0 for ARM - build Dec 18 2017
type:4102 id:0x00000010 name:CODESYS Control for Raspberry Pi SL vendor: 3S - Smart
Software Solutions GmbH
buildinformation: <none>

 __________
< ... bye >
 ----------
        \    ^__^
         \   (--)\_______
             (__)\       )\/\
                 ||----w |
                 ||     ||
```

Центр реагирования на инциденты информационной безопасности промышленных инфраструктур «Лаборатории Касперского» (Kaspersky ICS CERT) — глобальный проект «Лаборатории Касперского», нацеленный на координацию действий производителей систем автоматизации, владельцев и операторов промышленных объектов, исследователей информационной безопасности при решении задач защиты промышленных предприятий и объектов критически важных инфраструктур.

Kaspersky ICS CERT ics-cert@kaspersky.com