ИНСТИТУТ ИНТЕЛЛЕКТУАЛЬНЫХ КИБЕРНЕТИЧЕСКИХ СИСТЕМ

Кафедра
«Криптология и кибербезопасность»

ОТЧЕТ
о проектной работе по дисциплине
«Информационная безопасность автоматизированных систем управления технологическим процессом»

«Обзор спецификации OPC Unified Architecture»

Исполнители:
студенты гр. Б17-505: Иванова Н.Д.

Николаева Е.А.

Шайбель А.Р.

Шипилов А.В.

Преподаватель: Финошин М.А.

Москва – 2020

СОДЕРЖАНИЕ

Описание протокола

1 Введение

Технология OPC была разработана и впервые запущена в 1996 году OPC Foundation. Цель создания технологии заключалась в том, чтобы объединить в себе все существующие на тот момент протоколы, обеспечивающие работу SCADA-систем. До создания протокола производители продуктов SCADA были вынуждены задействовать сотни драйверов для корректной работы оборудования. Однако с появлением и внедрением OPC-серверов такая необходимость отпала.

OPC-сервер – программное обеспечение, реализованное в соответствии со спецификациями OPC и обеспечивающее преобразование протоколов обмена к стандартам спецификации OPC [1].

После успешного старта данная технология начала бурно развиваться, и по сей день используется на многих предприятиях. Однако с течением времени возникла необходимость пересмотреть работу «классических» стандартов.

Будучи реализована на технологии Microsoft DCOM, предыдущая версия протокола обладала рядом присущих этой технологии существенных ограничений: доступность только на операционных системах семейства Microsoft Windows, связь с технологией DCOM, исходные коды которой являются закрытыми, что не позволяет решать вопросы надежности ПО, а также выявлять и устранять возникающие программные отказы, бывают проблемы конфигурирования, связанные с DCOM, неточные сообщения DCOM о прерываниях связи, неприспособленность DCOM для обмена данными через интернет, неприспособленность DCOM для обеспечения информационной безопасности [2]. Чтобы уйти от ограничений технологии DCOM и решить некоторые другие обнаруженные за время использования протокола OPC проблемы, к примеру использование интернета также сделало некоторые системы более уязвимыми к хакерским атакам, возникла необходимость улучшить методы шифрования и защиты данных, OPC

Foundation разработал и выпустил новую версию протокола, который не был бы привязан к DCOM. По сути, новый протокол сохранил в себе все преимущества «классической» OPC технологии, но был избавлен от её недостатков. Этой технологией стал протокол обмена OPC UA.

OPC Unified Architecture (Унифицированная архитектура OPC) — спецификация, определяющая передачу данных в промышленных сетях и взаимодействие устройств в них. Разработана промышленным консорциумом OPC Foundation и значительно отличается от его предшествующих спецификаций.

OPC UA обеспечивает надежную и безопасную коммуникацию, противодействие вирусным атакам, гарантирует идентичность информации клиента и сервера. Благодаря своим новым свойствам и продуманной архитектуре протокол OPC UA становится основой коммуникаций между компонентами систем промышленного интернета вещей и умных городов.

OPC-UA — это платформо-независимый стандарт, основанный на кроссплатформенных мерах безопасности. Безопасность OPC-UA основана на инфраструктуре открытых ключей (PKI) с использованием цифровых сертификатов промышленного стандарта x.509 и адресов аутентификации, авторизации, шифрования и целостности данных. Спецификация предусматривает различные независимые от платформы и основанные на стандартах комбинации для межпроцессного взаимодействия клиент / сервер. OPC-UA поддерживает два формата сообщений и два транспортных протокола для межпроцессного взаимодействия.

Основные преимущества OPC-UA [3]:

– независимость от платформы: COM и DCOM переводятся в статус устаревших, Microsoft не уделяет внимания COM / DCOM как средству межпроцессного взаимодействия и вместо этого продвигает сервис-ориентированный подход и кроссплатформенные веб-сервисы. Отказ от зависимости COM / DCOM освобождает клиентские и серверные реализации OPC-UA от платформы Microsoft;

– встроенные платформы: серверы OPC-UA могут быть написаны для встроенных платформ - устройство может иметь собственный встроенный сервер OPC-UA. Для масштабируемости полная функциональность спецификации OPC-UA разбита на блоки в дискретные профили: встроенный сервер OPC-UA должен реализовывать только те профили, которые требуются от него клиентской средой;
– повышенная безопасность: классический OPC не имеет внутренней безопасности; это делегировано на уровень COM/DCOM. OPC-UA имеет комплексную модель безопасности, построенную на основе инфраструктуры открытого ключа, чтобы обеспечить безопасный канал связи клиент/сервер и средства для авторизации и аутентификации пользователей;
– улучшенное моделирование: стандарт OPC-UA предоставляет обширный словарь для моделирования устройств и процессов, находящихся под контролем, включая возможность вводить компоненты (позволяя клиентам устанавливать семантическую информацию) и выражать отношения между компонентами (позволяя клиентам легче просматривать между связанными компонентами);
– публикация данных на уровне предприятия: серверы OPC-UA могут публиковать данные через стандартные веб-службы SOAP с высоким уровнем безопасности. Используя этот метод, разные серверы OPC-UA могут безопасно обмениваться информацией о состоянии друг с другом через брандмауэры. Использование широко распространенного стандарта, такого как веб-службы SOAP через HTTP, также позволяет клиентам, не относящимся к OPC-UA, использовать выходные данные, опубликованные сервером OPC-UA.

Безопасность OPC UA состоит из аутентификации и авторизации, шифрования и обеспечения целостности данных при помощи сигнатур. OPC Foundation ориентировалась на спецификации Web Service Security. Для веб-служб используются WS Secure Conversation и следовательно они совместимы с .NET и другими реализациями SOAP. Для двоичного

протокола соблюдаются алгоритмы WS Secure Conversation и также конвертируются в двоичный эквивалент. Также присутствует смешанная версия, где код двоичен, но транспортным уровнем является SOAP. Это компрометирует эффективность двоичного кодирования и удобной для межсетевых экранов передачи. Двоичное кодирование всегда требует UA Secure Conversation. При аутентификации используются исключительно сертификаты x509. Выбор схемы сертификации, используемой приложением, возлагается на разработчиков приложений.

В OPC UA используется понятие объекта, под которым понимается физический или абстрактный элемент системы. Примерами объектов могут быть физические устройства, включающие их системы и подсистемы. Датчик температуры, к примеру, может быть представлен как объект, который включает в себя значение температуры, набор параметров сигнализаций и границы их срабатывания. Объект, по аналогии с объектно-ориентированным программированием, определяется как экземпляр класса, а класс рассматривается как тип данных. Объекты включают в себя переменные, события и методы .

OPC UA использует несколько различных форматов данных, основными из которых являются бинарные структуры и XML документы. Формат данных может быть определен поставщиком OPC сервера или стандартом. Для работы с произвольными форматами клиент может запросить у сервера информацию об описании этого формата. Во многих случаях используется автоматическое распознавание формата данных во время их передачи.

Организация OPC UA предоставляет промышленный стандарт OPC UA (Open Platform Communication Unified Architecture) для взаимодействия и горизонтальной и вертикальной интеграции информации от датчиков/исполнительных механизмов/машин в ERP (Enterprise Resource Planning). OPC UA предоставляет необходимую инфраструктуру для

взаимодействия в масштабе всего предприятия, от машины к машине, от машины к предприятию и всего, что между ними (рисунок 1) [4].

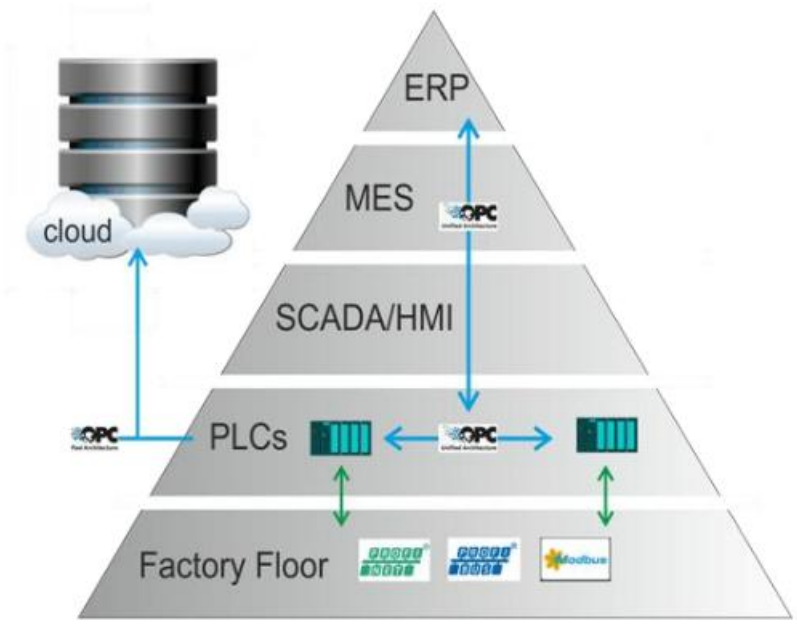


Рисунок 1 — Вертикальная и горизонтальная интеграция с унифицированной архитектурой OPC в автоматизации производства

2 Архитектура OPC UA

Архитектура системы OPC UA (рисунок 2) моделирует клиентов и серверов OPC UA как взаимодействующих партнеров [2,5].

Рисунок 2 — Архитектура системы OPC UA

Каждая система может содержать несколько клиентов и серверов. Каждый клиент может одновременно взаимодействовать с одним или несколькими серверов, и каждый сервер может одновременно взаимодействовать с одним или несколькими клиентами.

Приложение может объединять компоненты сервера и клиента, чтобы обеспечить взаимодействие с другими серверами и клиентами. Клиенты OPC UA

Архитектура клиента OPC UA моделирует клиентскую конечную точку взаимодействия клиент/сервер. Рисунок 3 иллюстрирует основные элементы типичного клиента OPC UA и их взаимосвязь.

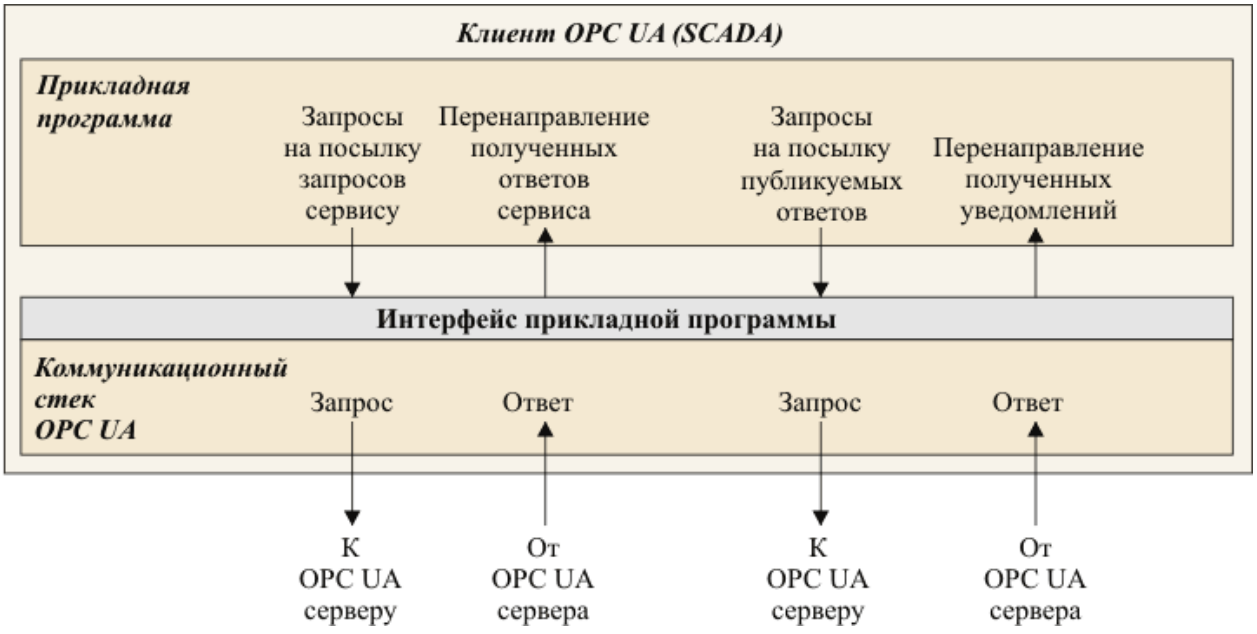


Рисунок 3 — Архитектура клиента OPC UA

Клиентское приложение — это код, реализующий функцию клиента. Он использует OPC UA Клиентский API для отправки и получения запросов и ответов службы OPC UA серверу OPC UA.

Архитектура сервера OPC UA моделирует конечную точку сервера взаимодействия клиент/сервер (рисунок 4).

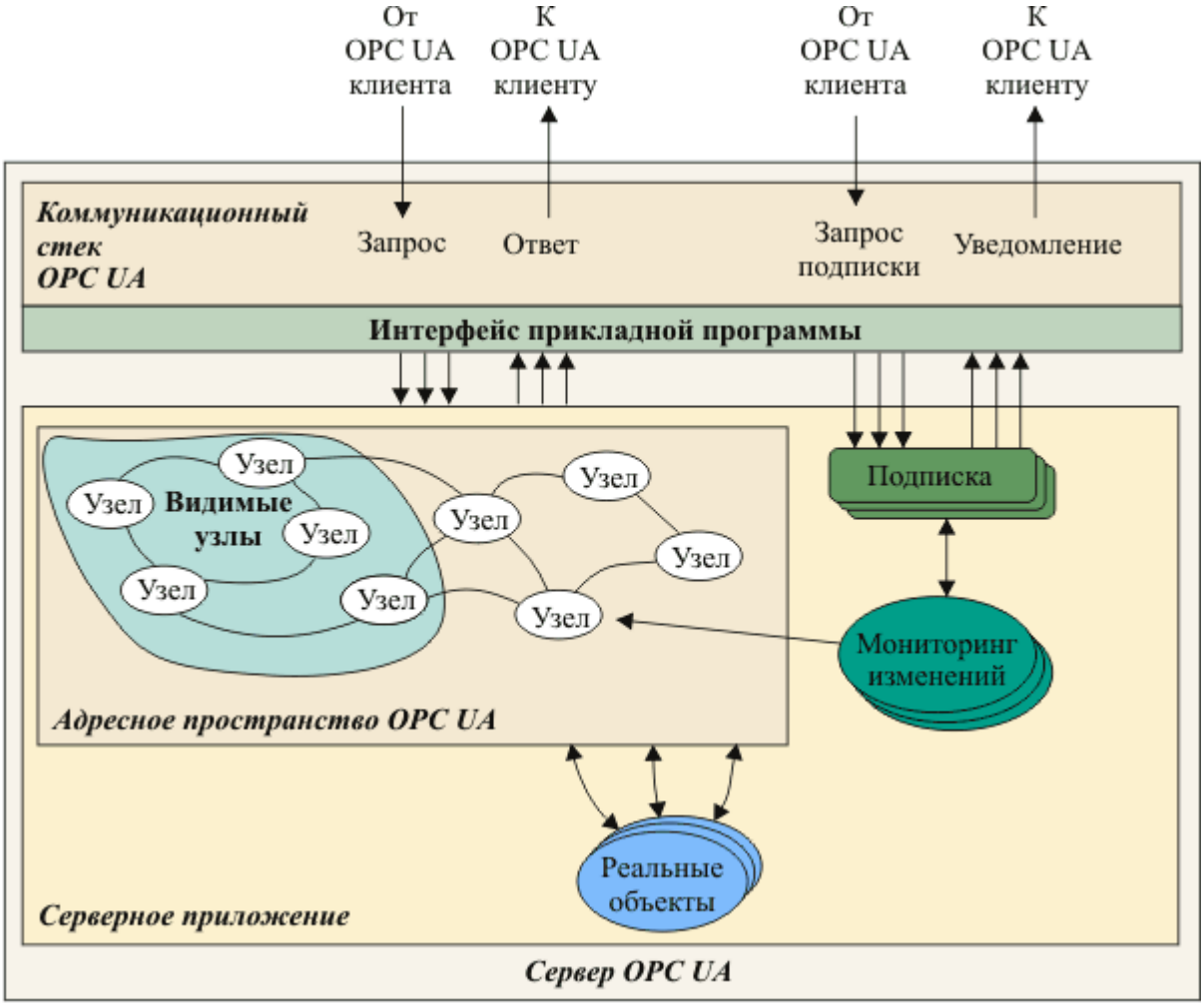


Рисунок 4 — Архитектура сервера OPC UA

Реальные объекты — это физические или программные объекты, доступные для приложения сервера OPC UA (например, физические устройства и счетчики диагностики).

Приложение OPC UA Server — это код, реализующий функцию сервера. Он использует API сервера OPC UA для отправки и получения сообщений OPC UA от клиентов OPC UA.

Взаимодействие сервера с сервером — это взаимодействие, при котором один сервер действует как клиент второго сервера (рисунок 5).

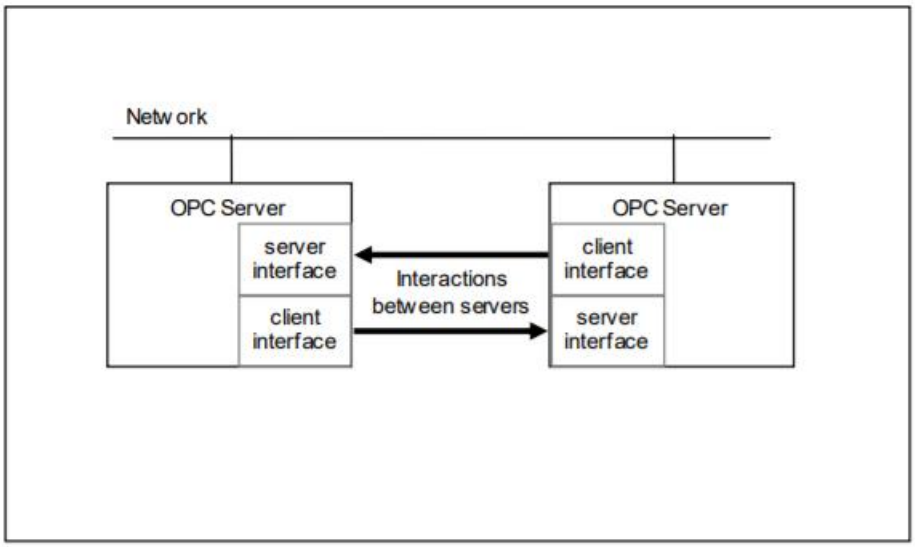


Рисунок 5 — Одноранговые взаимодействия между серверами

Рисунок 6 расширяет предыдущий пример и иллюстрирует соединение серверов OPC UA вместе для организации вертикального доступа к данным на предприятии.

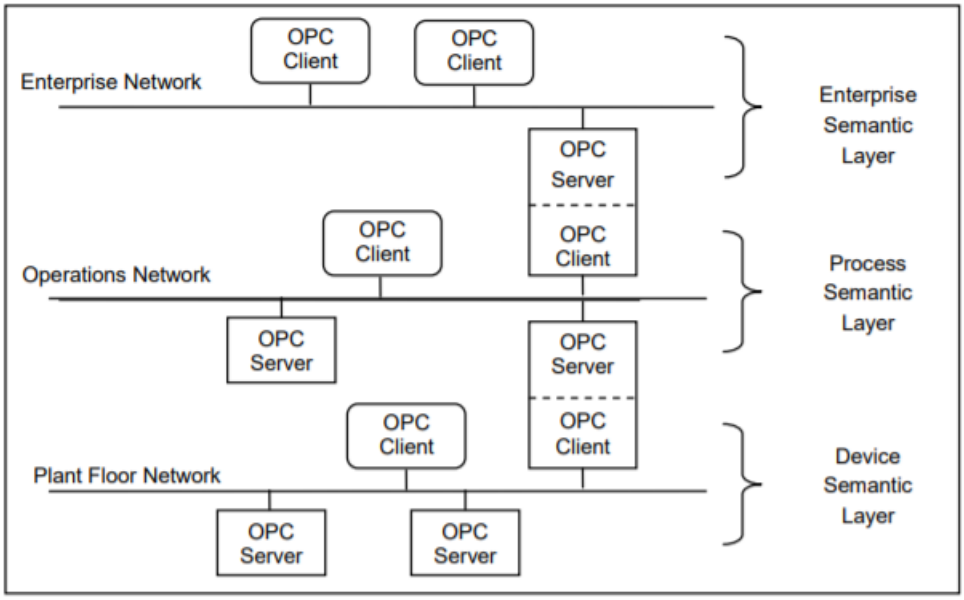


Рисунок 6 — Пример связанного сервера

Соединение клиент-сервер в OPC UA состоит из трех вложенных уровней [6]:

- raw connection (незащищенное соединение);
- secure channel (защищенный канал);
- session (сеанс).

Незащищенное соединение создается путем открытия TCP-соединения с соответствующим именем хоста и порта и начальным подтверждением связи. На этапе рукопожатия устанавливаются основные настройки соединения.

Защищенные каналы создаются поверх незащищенного TCP-соединения. SecureChannel устанавливается с помощью пары сообщений запроса и ответа OpenSecureChannel. Несмотря на то, что SecureChannel является обязательным, шифрование все же может быть отключено. SecurityMode SecureChannel может иметь значение None, Sign или SignAndEncrypt.

При включенной подписи или шифровании сообщения OpenSecureChannel шифруются с использованием асимметричного алгоритма шифрования (криптография с открытым ключом). В рамках сообщений OpenSecureChannel клиент и сервер согласовывают значение разделяемого секрета по изначально незащищенному каналу. Для защиты последующих сообщений используется симметричное шифрование с. разделяемым секретом, как более быстрое.

Сеансы создаются поверх защищенного канала. Это гарантирует, что пользователи могут аутентифицироваться без отправки своих учетных данных, таких как имя пользователя и пароль, в открытом виде. В настоящее время определены механизмы аутентификации: анонимный вход, имя пользователя/пароль, сертификаты Kerberos и x509. Последнее требует, чтобы сообщение с запросом сопровождалось подписью, чтобы доказать, что отправитель обладает закрытым ключом, с помощью которого был создан сертификат.

3 Чтение данных с устройства

Спецификация OPC-UA описывает две службы, с помощью которых клиенты OPC-UA получают доступ к данным устройства через серверы OPC-UA [3]:

- служба чтения (опрос): эта служба обеспечивает простое «разовое» чтение - по сути, опрос;
- служба подписки (публикация подписки): эта служба позволяет клиенту OPC-UA подписываться на обновления данных сервера OPC-UA и запрашивать получение уведомлений об обновлениях с указанной периодичностью

Подписка: клиенты подписываются на получение уведомлений об изменении интересующих их данных. Подписки действуют в течение одной клиентской сессии и выполняются с периодичностью (указываемой клиентом). В каждом цикле подписка пытается отправить уведомления клиенту с помощью следующего механизма: клиенты отправляют запросы публикации серверу, сервер ставит их в очередь. На сервере каждый цикл подписки выводит из очереди запрос на публикацию, упаковывает его со всеми неотправленными уведомлениями об отслеживаемых элементах и отправляет обратно клиенту. Если есть уведомления для публикации, но нет запросов публикации в очереди, сервер переходит в состояние ожидания, в котором исходящие запросы публикации упаковываются и немедленно возвращаются.

Keep Alive: если изменений в данных не происходит, уведомления не создаются, поэтому подписка не отвечает на запросы публикации клиента. Подписки увеличивают счетчик после каждого «инертного» цикла подписки (в течение которого не происходит изменений данных), и как только этот счетчик достигает порога активности подписки (значения, указанного клиентом), подписка удаляет запрос публикации из очереди, упаковывает ответ на сообщение и возвращает его клиенту.

Как описано выше, существует два протокола. Прикладной программист распознает их только через различие в URL opc.tcp://server для двоичного протокола и http://server/ для веб-служб. Иначе говоря, OPC UA работает полностью прозрачно для API.

Двоичный протокол [7]:

- лучшая производительность, минимальные накладные расходы;
- потребляет минимум ресурсов (не требуются парсер XML, SOAP и HTTP, что важно для встраиваемых устройств);
- наилучшая возможная совместимость (двоичный код определён явно и допускает меньшую степень свободы в процессе исполнения в отличие от XML);
- всего один порт TCP (4840) используется для коммуникации и легко может быть туннелирован или пропущен через межсетевой экран.

Веб-службы (SOAP):

- лучшая поддержка из доступных инструментов. Легко может быть использован, например, из окружения Java или .Net;
- применимый с межсетевыми экранами. Порт 80 (http) и 443 (https) обычно будут использоваться без дополнительных настроек.

4 Форматы сообщений

OPC-UA поддерживает два формата сообщений: двоичный UA и XML. Формат определяет, как кодируются данные сообщения. Отправитель сообщения должен закодировать данные в соответствующий формат для передачи, а получатель должен иметь возможность декодировать содержимое передачи для восстановления исходных данных [3].

UA Binary: этот формат сообщения кодирует данные, сериализованные в байтовый массив. UA Binary предлагает сниженные вычислительные затраты с точки зрения кодирования и декодирования, но может быть интерпретирован только клиентами, совместимыми с OPC-UA. UA Binary с большей вероятностью будет использоваться в коммуникациях на уровне устройства, где вычислительная мощность ограничена, а производительность имеет высокий приоритет.

XML: XML-документы — повсеместный метод обмена данными высокого уровня. Сообщения в кодировке XML могут интерпретироваться клиентами OPC-UA, а также универсальными клиентами с использованием контракта схемы XML (общие клиенты — это клиенты, не обладающие внутренним знанием OPC-UA). Сериализация и десериализация данных в формате XML является более дорогостоящим с точки зрения вычислений, чем двоичный формат UA, поэтому кодирование XML с большей вероятностью будет использоваться в корпоративной части спектра связи.

4.1 Структура протокольного сообщения

Рассмотрим пример двоичного диалога OPC UA, записанный и отображаемый с помощью инструмента Wireshark, показанный в: numref: ‘ua-wirehark` (рисунок 7) [6].

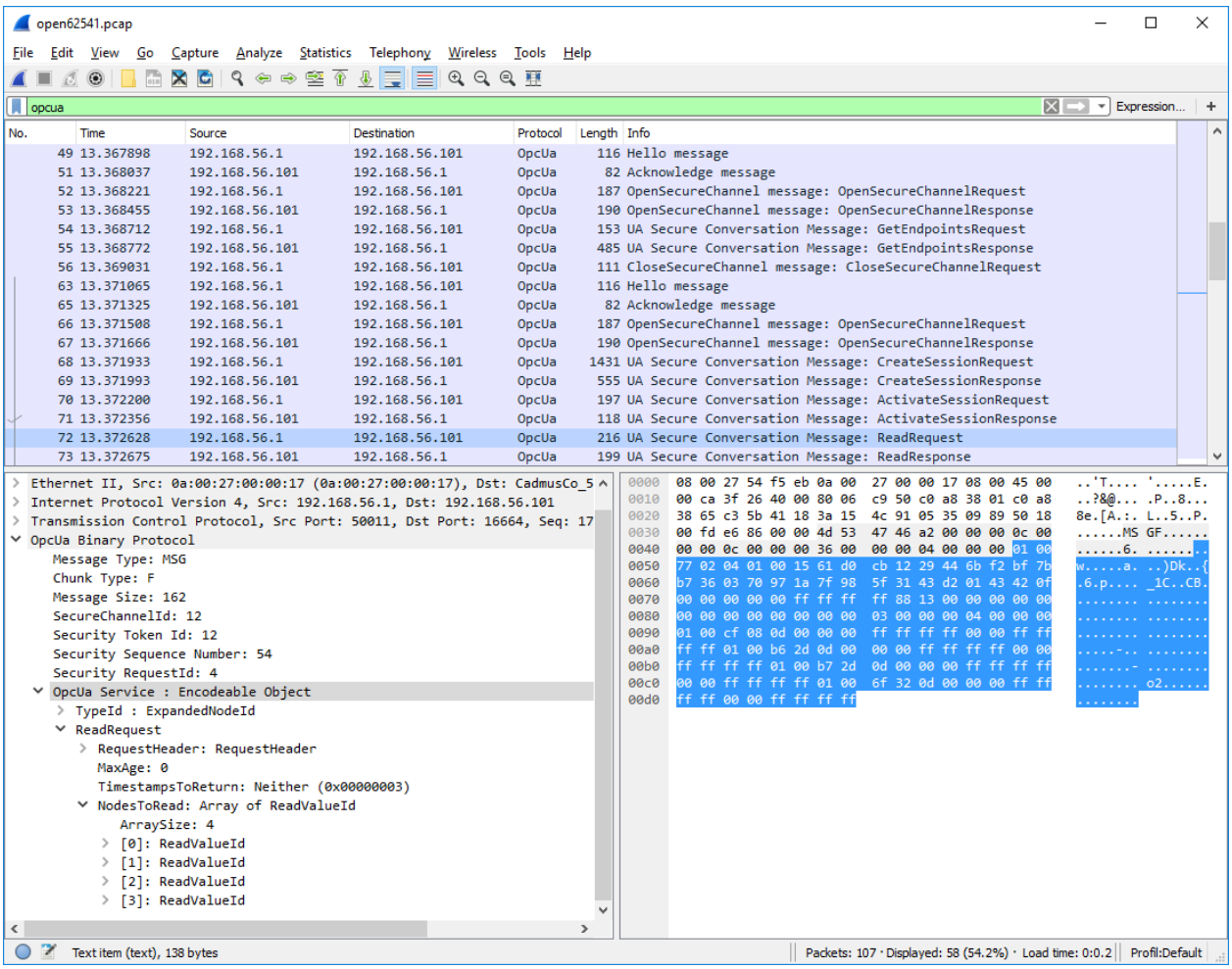


Рисунок 7 — Пример двоичного диалога OPC UA

На рисунках 8, 9 показана структура сообщения [8]:

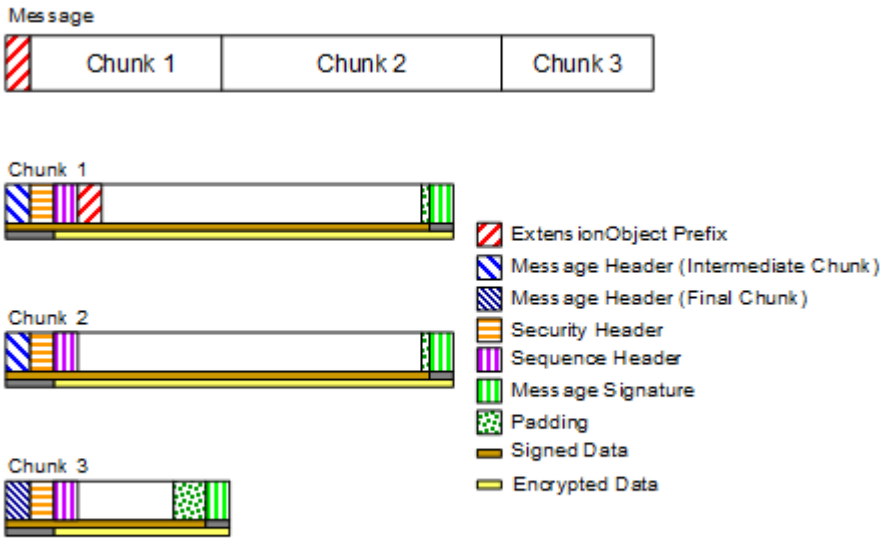


Рисунок 8 — Структура фрагментированного сообщения, передаваемого по протоколу OPC UA

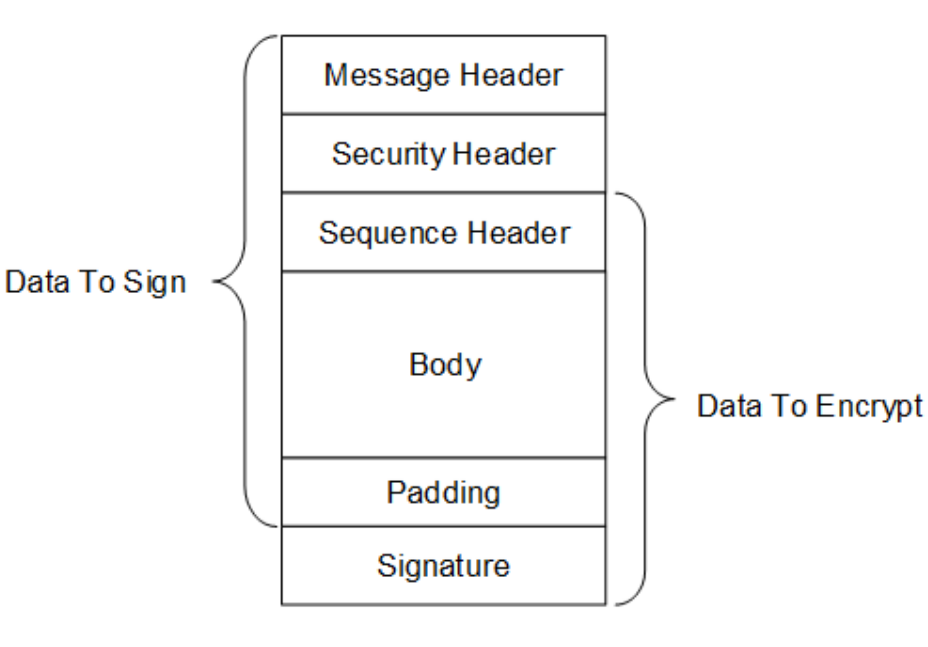


Рисунок 9 — Структура фрагмента сообщения, передаваемого по протоколу OPC UA

4.2 Заголовок сообщения

Каждое сообщение протокола соединения OPC UA имеет заголовок с полями, определенными в таблице 1.

Таблица 1 — Заголовок сообщения протокола соединения OPC UA

Название	Тип	Описание
MessageType	Byte [3]	Трехбайтовый код ASCII, определяющий тип сообщения. В настоящее время определены следующие значения: HEL (Hello Message), ACK (Acknowledge Message), ERR (Error Message), RHE (Reverse Hello Message). Уровень SecureChannel определяет дополнительные значения, которые должен принимать уровень протокола соединения OPC

		UA.
Reserved	Byte [1]	Игнорируется. Должен быть установлен на коды ASCII для «F», если MessageType является одним из значений, поддерживаемых протоколом соединения OPC UA.
MessageSize	UInt32	Длина сообщения в байтах. Это значение включает 8 байтов заголовка сообщения.

Заголовок сообщения протокола соединения OPC UA идентичен первым 8 байтам заголовка сообщения безопасного разговора OPC UA. Это позволяет уровню протокола соединения OPC UA извлекать сообщения SecureChannel из входящего потока, даже если он не понимает их содержания. Уровень протокола соединения OPC UA должен проверить MessageType и убедиться, что MessageSize меньше согласованного ReceiveBufferSize перед передачей любого сообщения на уровень SecureChannel.

Сообщение Hello имеет дополнительные поля, показанные в таблице 2.

Таблица 2 — Приветственное сообщение протокола соединения OPC UA

Название	Тип	Описание
ProtocolVersion	UInt32	Последняя версия протокола UACP, поддерживаемая сервером. Если клиент принимает соединение, он несет ответственность за то, чтобы отправляемые сообщения соответствовали данной версии протокола. Клиент должен поддерживать и все более поздние версии.
ReceiveBufferSize	UInt32	Наибольший MessageChunk, который может получить отправитель. Это значение не должно превышать то, которое было запрошено клиентом в

		сообщении Hello.
SendBufferSize	UInt32	Наибольший MessageChunk, который отправит отправитель. Это значение не должно превышать то, которое было запрошено клиентом в сообщении Hello.
MaxMessageSize	UInt32	Максимальная длина сообщения запроса. Клиент должен вернуть сообщение об ошибке Bad_RequestTooLarge, если длина отправленного сообщения превышает это значение. Нулевое значение говорит о том, что у сервера нет ограничений на длину сообщения.
MaxChunkCount	UInt32	Максимальное количество фрагментов в сообщении запроса. Клиент должен вернуть сообщение об ошибке Bad_RequestTooLarge, если длина отправленного сообщения превышает это значение. Нулевое значение говорит о том, что у сервера нет ограничений на длину сообщения.

Если у сервера недостаточно ресурсов для создания нового SecureChannel, он должен немедленно вернуть сообщение об ошибке Bad_TcpNotEnoughResources и корректно закрыть сокет. Клиент не должен перегружать серверы, которые возвращают эту ошибку, немедленно пытаясь создать новый SecureChannel.

Сообщение подтверждения имеет дополнительные поля, показанные в таблице 3.

Таблица 3 — Сообщение с подтверждением протокола соединения OPC UA

Название	Тип	Описание
ProtocolVersion	UInt32	Последняя версия протокола UACP,

		поддерживаемая сервером. Если клиент принимает соединение, он несет ответственность за то, чтобы отправляемые сообщения соответствовали данной версии протокола.
ReceiveBufferSize	UInt32	Наибольший MessageChunk, который может получить отправитель. Это значение не должно превышать то, которое было запрошено клиентом в сообщении Hello.
SendBufferSize	UInt32	Наибольший MessageChunk, который отправит отправитель. Это значение не должно превышать то, которое было запрошено клиентом в сообщении Hello.
MaxMessageSize	UInt32	Максимальная длина сообщения запроса. Клиент должен вернуть сообщение об ошибке Bad_RequestTooLarge, если длина отправленного сообщения превышает это значение. Нулевое значение говорит о том, что у сервера нет ограничений на длину сообщения.
MaxChunkCount	UInt32	Максимальное количество фрагментов в сообщении запроса. Клиент должен вернуть сообщение об ошибке Bad_RequestTooLarge, если длина отправленного сообщения превышает это значение. Нулевое значение говорит о том, что у сервера нет ограничений на длину сообщения.

Сообщение об ошибке содержит дополнительные поля, показанные в таблице 4.

Таблица 4 — Сообщение об ошибке протокола соединения OPC UA

Название	Тип	Описание
Error	UInt32	Числовой код ошибки.
Reason	String	Более подробное описание ошибки. Эта строка не должна быть более 4 096 байтов. Клиент должен игнорировать строки, которые длиннее этого.

Сокет всегда корректно закрывается клиентом после получения сообщения об ошибке.

Сообщение ReverseHello содержит дополнительные поля, показанные в таблице 5.

Таблица 5 — Сообщение протокола соединения OPC UA ReverseHello

Название	Тип	Описание
ServerUri	String	ApplicationUri сервера, который отправил сообщение. Закодированное значение должно быть менее 4 096 байтов. Клиент должен вернуть Bad_TcpEndpointUrlInvalid ошибку и закрыть соединение, если длина превышает 4 096 или если он не распознает Сервер, определенный в URI.
EndpointUrl	String	URL конечной точки, которую Клиент использует при создании SecureChannel. Это значение должно быть передано обратно серверу в сообщении Hello. Закодированное значение должно быть менее 4 096 байтов. Клиенты должны вернуть Bad_TcpEndpointUrlInvalid ошибку и закрыть

		соединение, если длина превышает 4 096 или если они не могут распознать ресурс, определенный в URL. Это значение является уникальным идентификатором для сервера, который Клиент может использовать для поиска информации о конфигурации. Это должен быть один из URL-адресов, возвращенных сервисом GetEndpoints.

Для протокола на основе подключений (например, TCP), сообщение ReverseHello позволяет серверам за брандмауэрами без открытых портов подключаться к клиенту и запрашивать у него установку SecureChannel с использованием сокета, созданного сервером.

Для протоколов на основе сообщений сообщение ReverseHello позволяет серверам сообщать о своем присутствии клиенту. В этом сценарии EndpointUrl указывает адрес протокола сервера и любые токены, необходимые для доступа к нему.

Структура клиент-сервер

Для разработки структуры клиент-сервера для исследуемого протокола OPC UA использовался язык программирования Python.

Ниже представлен код серверной программы server.py.

```
import
time
        from opcua import ua, Server


        if __name__ == "__main__":
            server = Server()

            server.set_endpoint("opc.tcp://0.0.0.0:4840/freeopcua/server/")
            ## opc.tcp -- протокол обмена с другими сетевыми устройствами
            ## 0.0.0.0 --IP клиента, с которого разрешается прием пакетов
            ## 4840 -- порт, на который должны поступать запросы от OPC UA-клиентов
            ## server.set_endpoint() -- создает точку для подключения клиентов

            ## регистрация адресного пространства имен в структуре сервера

            ns = server.register_namespace("Concepts")

            ## создание объекта Object и измерительной величины Variable
            objects = server.get_objects_node()
            myobj = objects.add_object(ns, "Object")
            myvar = myobj.add_variable(ns, "Variable", 0.0)

            ## разрешение доступа на запись
            myvar.set_writable()

            server.start()

            while True:
                value = myvar.get_value()
                print("{0:.1f} units".format(value))
                time.sleep(1)

            server.stop()
```

Демонстрация работы сервера представлена на рисунке 10.

```
(env) cheraten@cheraten-VirtualBox:~/python-virtual-environments/src$ python server.py
cryptography is not installed, use of crypto disabled
cryptography is not installed, use of crypto disabled
Endpoints other than open requested but private key and certificate are not set.
Listening on 0.0.0.0:4840
0.0 units
0.0 units
0.0 units
0.0 units
0.0 units
0.0 units
0.0 units
0.0 units
0.0 units
0.0 units
0.1 units
0.2 units
0.3 units
0.4 units
0.5 units
0.6 units
0.7 units
0.8 units
0.9 units
1.0 units
```

Рисунок 10 — Демонстрация работы сервера

Ниже представлен код клиентской программы client.py.

```
import
time
        from opcua import Client


        if __name__ == "__main__":

            ## указание сетевого ресурса, к которому клиент должен подключиться

            client = Client("opc.tcp://localhost:4840/freeopcua/server/")
            client.connect()

            ## по идентификатору подключаемся к нужному нам узлу
            var = client.get_node("ns=2;i=2")

            count = 0
            while True:
                time.sleep(1)
                count += 0.1
                print("{:8.1f} units".format(count))

                ## запись значения в переменную var
                var.set_value(count)
```

```
client.disconnect()
```

Демонстрация работы клиента представлена на рисунке 11.

```
(env) cheraten@cheraten-VirtualBox:~/python-virtual-environments/src$ python client.py
cryptography is not installed, use of crypto disabled
cryptography is not installed, use of crypto disabled
     0.1 units
     0.2 units
     0.3 units
     0.4 units
     0.5 units
     0.6 units
     0.7 units
     0.8 units
     0.9 units
     1.0 units
```

Рисунок 11 — Демонстрация работы клиента

Сообщения клиента и сервера OPC UA были проанализированы с помощью программного средства Wireshark.

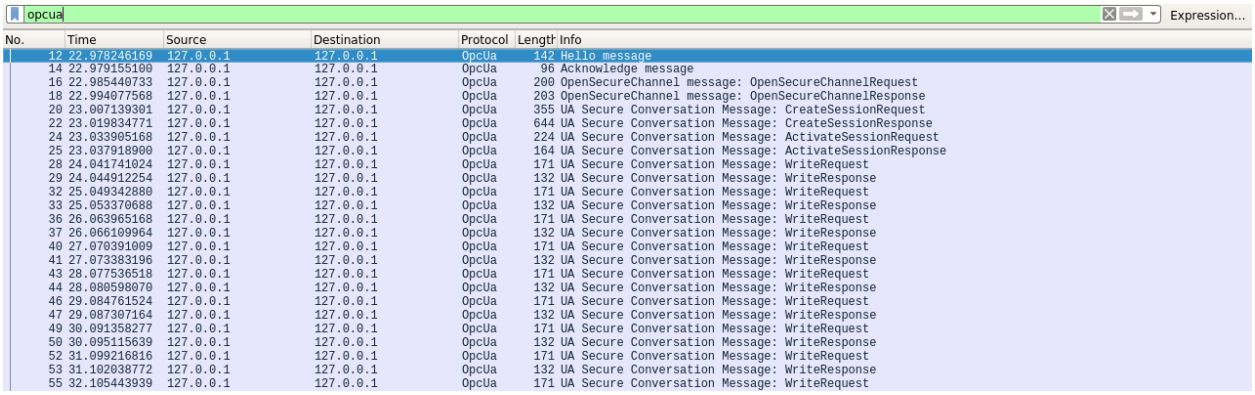


Рисунок 12 — Вывод сообщений OPC UA

Было выбрано 28 сообщение: первый WriteRequest, отправленный от клиента серверу. В NodeId содержатся параметры идентификатора узла, в Value — значение записываемой переменной, равное 0.1 (рисунок 12).

Рисунок 13 — Детали сообщения первого WriteRequest

В следующем WriteRequest (сообщение 32) значение Value уже равно 0.2 (рисунок 13).

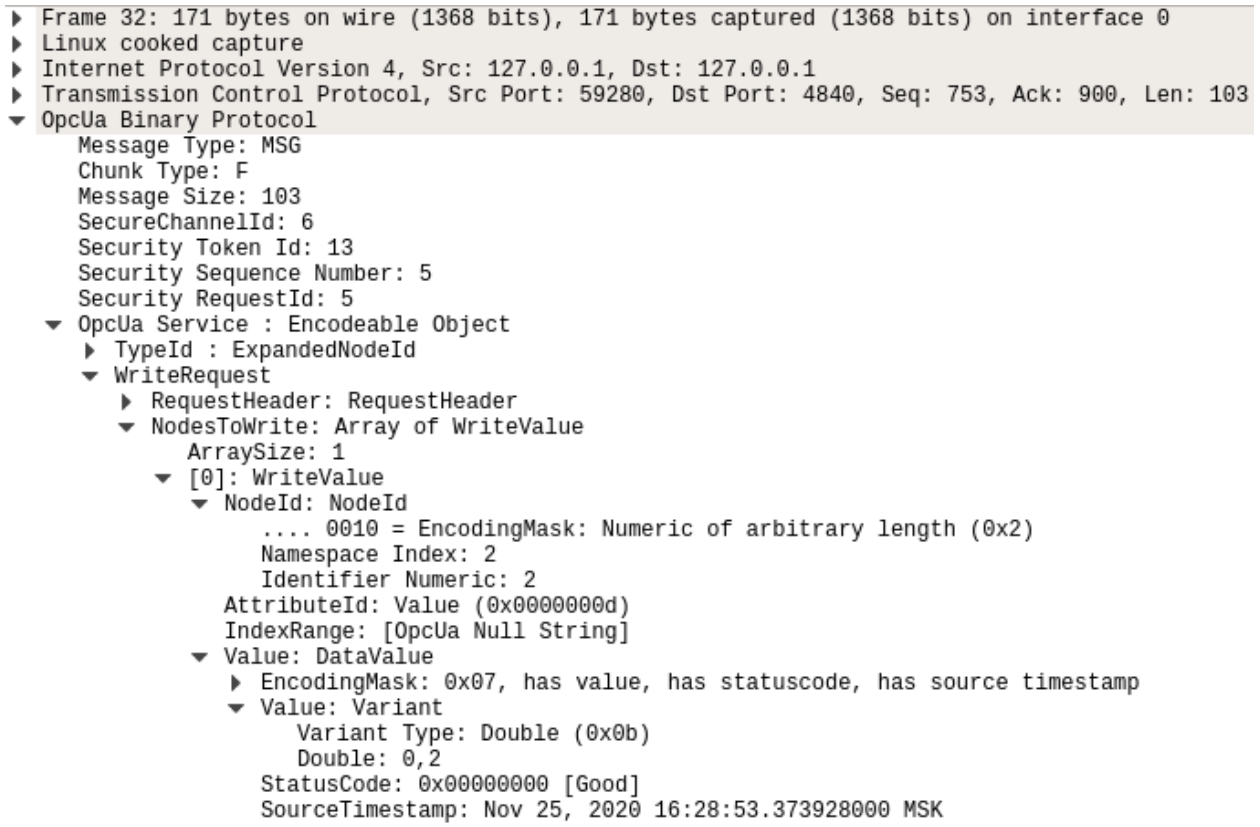


Рисунок 14 — Детали сообщения второго WriteRequest

Настройка Suricata

Установка Suricata:

Для Ubuntu:

1. sudo apt-get update && sudo apt-get upgrade -y
2. sudo apt-get install libpcre3 libpcre3-dbg libpcre3-dev build-essential \
   libpcap-dev libnet1-dev libyaml-0-2 libyaml-dev pkg-config zlib1g zlib1g-dev \
   libcap-ng-dev libcap-ng0 make libmagic-dev libjansson-dev \
   libnss3-dev libgeoip-dev liblua5.1-dev libhiredis-dev libevent-dev liblz4-dev \
   m4 autoconf autogen
4. git clone https://github.com/OISF/suricata.git
5. cd suricata
6. git clone https://github.com/OISF/libhtp
7. sudo ./autogen.sh
8. sudo ./configure --prefix=/usr --sysconfdir=/etc --localstatedir=/var
9. sudo make && sudo make install && sudo make install-conf
10. sudo ifconfig lo mtu 1522
11. sudo cp suricata.yaml /etc/suricata
12. sudo suricata-update -D /etc/suricata

Для Mac OS X:

1. brew install suricata

Изменение пути к файлу с правилами в конфигурационном файле:

1. sudo gedit /etc/suricata/suricata.yaml
2. rule-files:
   - <opcua.rules>

(Под <opcua.rules> понимается путь к файлу с правилами)

```
##
## Configure Suricata to load Suricata-Update managed rules.
##

default-rule-path: /etc/suricata/rules

rule-files:
  - suricata.rules
```

Конфигурация loopback-интерфейса:

```
##
## Step 3: Configure common capture settings
##
## See "Advanced Capture Options" below for more options, including Netmap
## and PF_RING.
##

# Linux high speed capture support
af-packet:
  - interface: lo
```

Проверка работы правил:

1. Запуск Wireshark:

   sudo Wireshark
2. Редактирование правила:

   sudo gedit /etc/suricata/rules/suricata.rules
3. Запуск Suricata:

   sudo suricata -c /etc/suricata/suricata.yaml -i lo --set capture.disable-offloading=false
4. Запуск сервера:

   cd python-virtual-environments

   source env/bin/activate

   cd src

   sudo python server.py
5. Запуск клиента:

   cd python-virtual-environments

   source env/bin/activate

   cd src

   sudo python client.py
6. Просмотр логов для проверки, отработало ли правило:

   sudo cat /var/log/suricata/fast.log

Правила Suricata

1. simple_hel

```
cheraten@cheraten-VirtualBox:~$ sudo cat /var/log/suricata/fast.log
12/08/2020-12:19:19.591257  [**] [1:0:0] HELLO opcua packet (from client) [**] [
Classification: (null)] [Priority: 3] {TCP} 127.0.0.1:56866 -> 127.0.0.1:4840
```

Детектирует HELLO-сообщение.

alert tcp 127.0.0.1 any -> 127.0.0.1 4840 (msg: "HELLO opcua packet (from client)"; content: "HEL"; startswith;)

2. simple_ack

```
12/08/2020-12:21:40.340762  [**] [1:0:0] ACK opcua packet (from server) [**] [Classificati
on: (null)] [Priority: 3] {TCP} 127.0.0.1:4840 -> 127.0.0.1:56868
```

Детектирует Acknowledgment-сообщение.

alert tcp 127.0.0.1 4840 -> 127.0.0.1 any (msg: "ACK opcua packet (from server)"; content: "ACK"; startswith;)

3. simple_opn

```
12/08/2020-12:25:14.385826  [**] [1:0:0] OPN opcua packet [**] [Classification: (null)] [P
riority: 3] {TCP} 127.0.0.1:56872 -> 127.0.0.1:4840
12/08/2020-12:25:14.388759  [**] [1:0:0] OPN opcua packet [**] [Classification: (null)] [P
riority: 3] {TCP} 127.0.0.1:4840 -> 127.0.0.1:56872
```

Детектирует сообщения открытия защищенного канала.

alert tcp 127.0.0.1 any <> 127.0.0.1 4840 (msg: "OPN opcua packet"; content: "OPN"; startswith;)

4. create_session_request

```
12/08/2020-12:27:41.806586  [**] [1:0:0] CreateSessionRequest by opcua client [**] [Classi
fication: (null)] [Priority: 3] {TCP} 127.0.0.1:56876 -> 127.0.0.1:4840
```

Детектирует сообщение CreateSessionRequest.

alert tcp 127.0.0.1 any -> 127.0.0.1 4840 (msg: "CreateSessionRequest by opcua client"; content: "|cd 01|"; offset: 26;depth: 2; rawbytes;)

5. ns_i

```
12/08/2020-12:29:38.987844  [**] [1:0:0] connect to ns=2,i=2 [**] [Classification: (null)]
 [Priority: 3] {TCP} 127.0.0.1:56878 -> 127.0.0.1:4840
12/08/2020-12:30:05.221414  [**] [1:0:0] connect to ns=2,i=2 [**] [Classification: (null)]
 [Priority: 3] {TCP} 127.0.0.1:56878 -> 127.0.0.1:4840
```

Детектирует подключение клиента к ns=2;i=2 сервера.

Правило детектирует 2 соседних поля отдельно:

alert tcp 127.0.0.1 any -> 127.0.0.1 4840 (msg: "connect to ns=2,i=2"; content: "|02 00|"; content: "|02 00 00 00|"; distance: 0; within: 4; rawbytes;)

Правило детектирует 2 соседних поля как одно и работает более быстро:

alert tcp 127.0.0.1 any -> 127.0.0.1 4840 (msg: "connect to ns=2,i=2"; content: "|02 00 02 00 00 00|"; offset: 67; depth: 8; rawbytes;)

6. 100bytes_msg

```
12/08/2020-12:31:11.133958  [**] [1:0:0] client WriteRequest length is more than 100 [**]
[Classification: (null)] [Priority: 3] {TCP} 127.0.0.1:56880 -> 127.0.0.1:4840
12/08/2020-12:31:12.141050  [**] [1:0:0] client WriteRequest length is more than 100 [**]
[Classification: (null)] [Priority: 3] {TCP} 127.0.0.1:56880 -> 127.0.0.1:4840
```

Детектирует WriteRequest больше 100б.

alert tcp 127.0.0.1 any -> 127.0.0.1 4840 (msg: "client WriteRequest length is more than 100"; dsize:>100; content: "|a1 02|"; offset: 26; depth: 2; rawbytes;)

7. verybig_msg

```
12/08/2020-12:33:21.488696  [**] [1:0:0] extremely big message (>500)! this is CreateSessi
onResponse [**] [Classification: (null)] [Priority: 3] {TCP} 127.0.0.1:4840 -> 127.0.0.1:5
6884
```

Детектирует CreateSessionResponse больше 500б.

alert tcp 127.0.0.1 4840 -> 127.0.0.1 any (msg: "extremely big message (>500)! this is CreateSessionResponse"; dsize:>500; content: "|d0 01|"; offset: 26; depth: 2; rawbytes;)

8. check_certificate

```
12/08/2020-12:35:13.209730  [**] [1:0:0] the subscriber does not have a certificate for op
ening a secure channel !! [**] [Classification: (null)] [Priority: 3] {TCP} 127.0.0.1:5688
6 -> 127.0.0.1:4840
12/08/2020-12:35:21.264320  [**] [1:0:0] the subscriber does not have a certificate for op
ening a secure channel !! [**] [Classification: (null)] [Priority: 3] {TCP} 127.0.0.1:4840
 -> 127.0.0.1:56886
```

Детектирует отсутствие сертификата при открытии защищенного канала.

alert tcp 127.0.0.1 4840 <> 127.0.0.1 any (msg: "the subscriber does not have a certificate for opening a secure channel !!"; content: "OPN"; content: "|ff ff ff ff|"; distance: 60; within: 4; rawbytes;)

9. check_securechannel_forOPNmsg

```
12/08/2020-12:37:24.160348  [**] [1:0:0] secure channel not open for OPN msg !! [**] [Clas
sification: (null)] [Priority: 3] {TCP} 127.0.0.1:56890 -> 127.0.0.1:4840
```

Детектирует отсутствие защищенного канала при открытии защищенного канала. Такое сообщение должно быть только одно (запрос от клиента на открытие).

alert tcp 127.0.0.1 4840 <> 127.0.0.1 any (msg: "secure channel not open for OPN msg !!"; content: "OPN"; content: "|00 00 00 00|"; distance: 5; within: 4; rawbytes;)

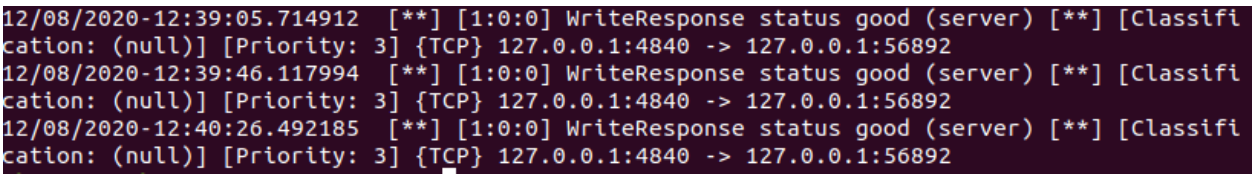
10.alert_securechannel_forMSGmsg

Предупреждает о пакетах типа MSG с отсутствием защищенного канала.

alert tcp 127.0.0.1 4840 <> 127.0.0.1 any (msg: "secure channel not open for MSG msg !!"; content: "MSG"; content: "|00 00 00 00|"; distance: 5; within: 4; rawbytes;)

Так как таких пакетов при нормальной работе сервера быть не должно, правильным результатом работы правила посчиталось отсутствие предупреждений о срабатывании.

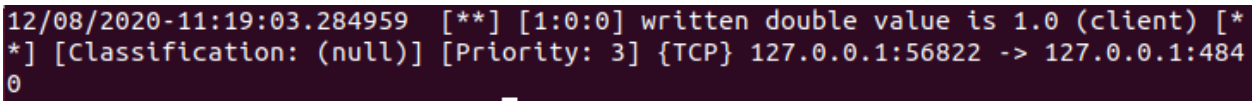
11.check_status_writeresponse



Детектирует status good WriteResponse от сервера.

alert tcp 127.0.0.1 4840 -> 127.0.0.1 any (msg: "WriteResponse status good (server)"; content: "|a4 02|"; content: "|00 00 00 00|"; distance: 12; within: 4; rawbytes;)

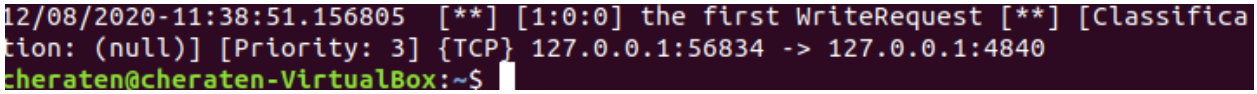
12.detect_value1.0



Детектирует сообщение передачи от клиента переменной на запись серверу со значением 1.0.

alert tcp 127.0.0.1 any -> 127.0.0.1 4840 (msg: "written double value is 1.0 (client)"; content: "|a1 02|"; content: "|ff ff ff ff ff ff ef 3f|"; distance: 55; within: 8; rawbytes;)

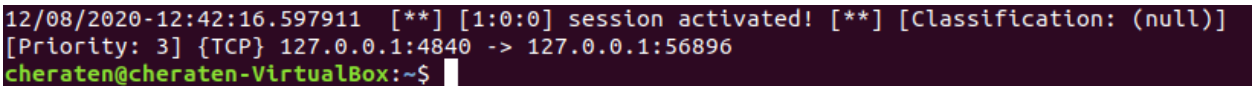
13.detect_firstwriterequest



Детектирует первый WriteRequest.

alert tcp 127.0.0.1 any -> 127.0.0.1 4840 (msg: "the first WriteRequest"; content: "|04 00 00 00|"; content: "|a1 02|"; distance: 2; within:2; rawbytes;)

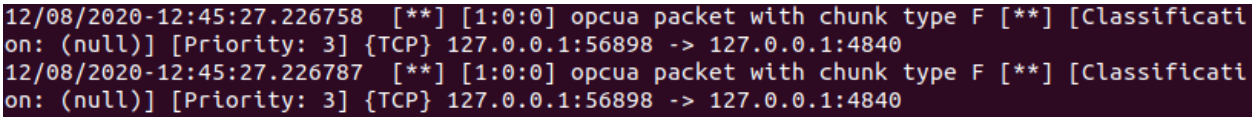
14.check_activatesessionstatus



Детектирует сообщение ActivateSessionResponse со статусом good.

alert tcp 127.0.0.1 4840 -> 127.0.0.1 any (msg: "session activated!"; content: "|d6 01|"; content: "|00 00 00 00|"; distance: 12; within: 4; rawbytes;)

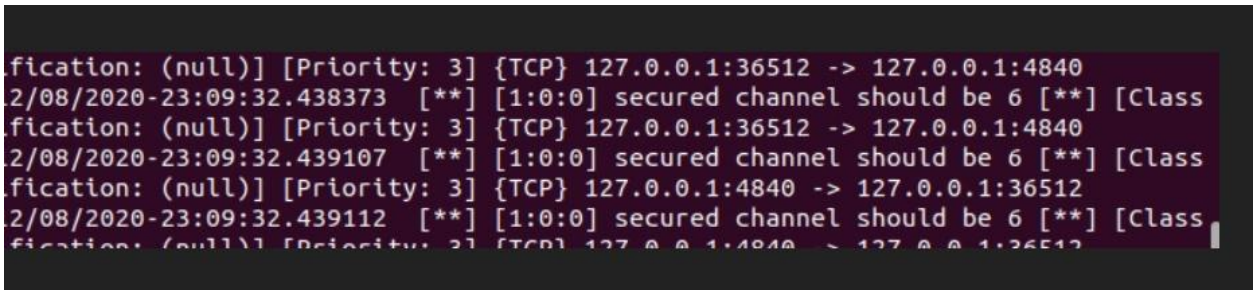
15.check_byteF



Проверка на наличие зарезервированного байта F.

alert tcp 127.0.0.1 any <> 127.0.0.1 4840 (msg: "opcua packet with chunk type F"; content: "F"; offset: 3; depth: 1;)

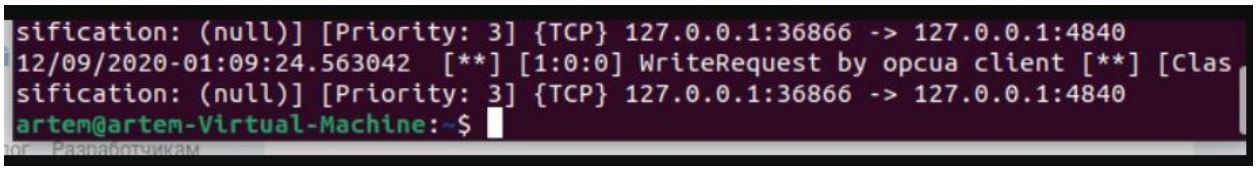
16.check_6securechannel



Детектит сообщения по 6-му защищенному каналу.

alert tcp 127.0.0.1 any <> 127.0.0.1 4840 (msg: "secure channel 6"; content: "|06 00 00 00|"; offset: 8; depth: 4; rawbytes;)

17. detect_writerequest



Детектирует WriteRequest (все сообщения должны быть от клиента).

alert tcp 127.0.0.1 any <> 127.0.0.1 4840 (msg: "WriteRequest by opcua client"; content: "|a1 02|"; offset: 26; depth: 2; rawbytes;)

18.detect_14request_6securechannel

```
12/09/2020-10:14:49.240736  [**] [1:0:0] channel 6 and id 14 [**] [Classification: (null)] [Priority: 3] {TCP} 127
.0.0.1:34446 -> 127.0.0.1:4840
12/09/2020-10:14:56.297168  [**] [1:0:0] channel 6 and id 14 [**] [Classification: (null)] [Priority: 3] {TCP} 127
.0.0.1:4840 -> 127.0.0.1:34446
```

Детектит сообщения 14-го запроса, проходящие по защищенному каналу № 6. Таких сообщений может быть только 2 (от клиента и от сервера)!

alert tcp 127.0.0.1 any <> 127.0.0.1 4840 (msg: "channel 6 and id 14"; content: "|06 00 00 00|"; content: "|0e 00 00 00|"; distance: 8; within: 4; rawbytes;)

19.detect_13token

```
12/09/2020-10:56:00.188534  [**] [1:0:0] message with token_id = 13 [**] [Classification: (null)] [Priority: 3] {T
CP} 127.0.0.1:34482 -> 127.0.0.1:4840
12/09/2020-10:56:00.188569  [**] [1:0:0] message with token_id = 13 [**] [Classification: (null)] [Priority: 3] {T
CP} 127.0.0.1:34482 -> 127.0.0.1:4840
12/09/2020-10:56:00.191264  [**] [1:0:0] message with token_id = 13 [**] [Classification: (null)] [Priority: 3] {T
CP} 127.0.0.1:4840 -> 127.0.0.1:34482
12/09/2020-10:56:00.191282  [**] [1:0:0] message with token_id = 13 [**] [Classification: (null)] [Priority: 3] {T
CP} 127.0.0.1:4840 -> 127.0.0.1:34482
```

Детектирует сообщения с id токена доступа = 13.

alert tcp 127.0.0.1 any <> 127.0.0.1 4840 (msg: "message with token_id = 13"; content: "|0d 00 00 00|"; offset: 9; depth: 4; rawbytes;)

20.detect_createsession_endpointURI_opcuaserver

```
12/09/2020-10:59:10.867498  [**] [1:0:0] CreateSession to opc.tcp://localhost:4840/freeopcua/server/ [**] [Classif
ication: (null)] [Priority: 3] {TCP} 127.0.0.1:34484 -> 127.0.0.1:4840
```

Детектирует запрос на открытие сессии к точке с URI = opc.tcp://localhost:4840/freeopcua/server/

alert tcp 127.0.0.1 any <> 127.0.0.1 4840 (msg: "CreateSession to opc.tcp://localhost:4840/freeopcua/server/"; content: "opc.tcp://localhost:4840/freeopcua/server/"; offset: 162; depth: 42;)

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. СМИС Эксперт [Электронный ресурс] / — Режим доступа: https://www.smis-expert.com/programmnoe-obespechenie/protokol-opc-ua-kommunikatsionnyj-standart-novogo-pokoleniya.html, свободный (дата обращения: 24.10.2020).
2. Энциклопедия АСУ ТП [Электронный ресурс] / — Режим доступа: https://www.bookasutp.ru/Chapter9_2_4.aspx, свободный (дата обращения: 24.10.2020).
3. CERN Accelerating science [Электронный ресурс] / — Режим доступа: https://readthedocs.web.cern.ch/display/ICKB/OPC-UA+Summary, свободный (дата обращения: 24.10.2020).
4. Trends in industrial communication and OPC UA [Текст] / Drahoš, Peter & Kucera, Erik & Haffner, Oto & Klimo, Ivan — CYBERI, 2018.
5. OPC Unified Architecture Part 1: Overview and Concepts [Текст] / OPC Foundation — February 5, 2009.
6. Open62541 [Электронный ресурс] / — Режим доступа: https://open62541.org/doc/current/protocol.html, свободный (дата обращения: 24.10.2020).
7. Secure Communication in Industrial Automation by Applying OPC UA [Текст] / Leitner, Stefan-Helmut & Mahnke, Wolfgang & Schierholz, Ragnar —2020.
8. OPC UA Online Reference [Электронный ресурс] / — Режим доступа: https://reference.opcfoundation.org/v104/Core/docs/Part6/7.1.2/, свободный (дата обращения: 24.10.2020).