Национальный исследовательский ядерный университет
«МИФИ»

Отчет
по лабораторной работе
«Добавление модуля анализа промышленного протокола в ПО Suricata»

Выполнили: Голуб Светлана
Герасимов Фёдор
Грубач Арсений
Земский Максим

Группа: Б17-505

Москва 2020г.

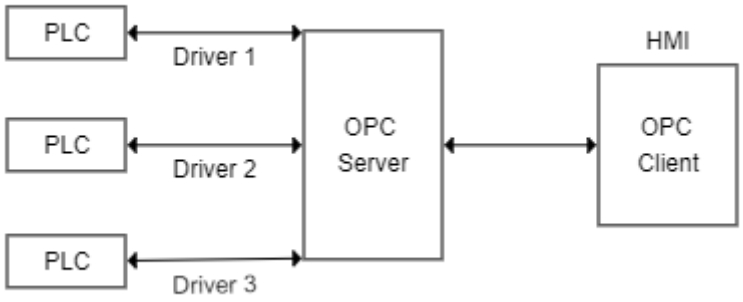
СОДЕРЖАНИЕ

OPC UA

1. Описание протокола.

Предшественником OPC UA была спецификация OPC.

OPC (OLE for Process Control) — это семейство протоколов и технологий, предоставляющих универсальный механизм сбора данных из различных источников и передачу этих данных любой клиентской программе вне зависимости от типа используемого оборудования.



Однако данная технология была доступна только в операционных системах Microsoft Windows и отсутствовала возможность обеспечить безопасность передачи данных. Это стало причиной разработки Унифицированной архитектуры OPC.

OPC Unified Architecture (OPC UA) - спецификация, являющаяся мультиплатформенным стандартом, с реализованной безопасностью. Также система является масштабируемой и поддерживает многопоточность.

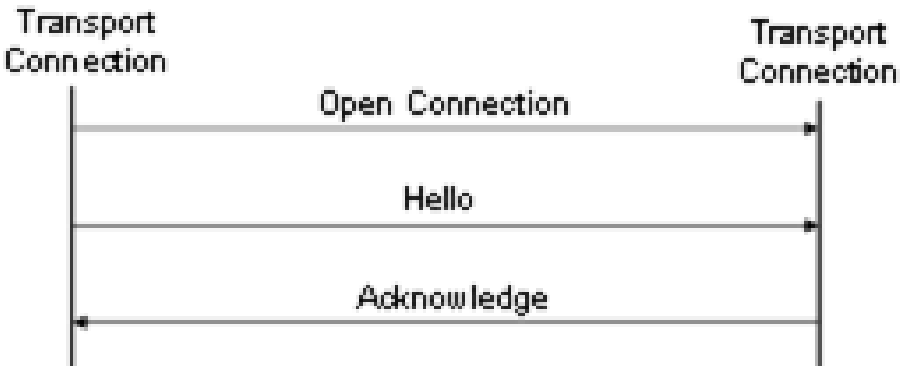
1.1. Кодировки

Протокол OPC UA поддерживает передачу данных в трёх кодировках: OPC UA Binary, OPC UA XML и OPC UA JSON. Кодирование и декодирование формата XML занимает много времени, поэтому поддерживается представление информации в виде бинарного файла. Для того, чтобы позволить приложениям OPC UA взаимодействовать с приложениями в сети Интернет, было разработано кодирование данных в формате JSON.

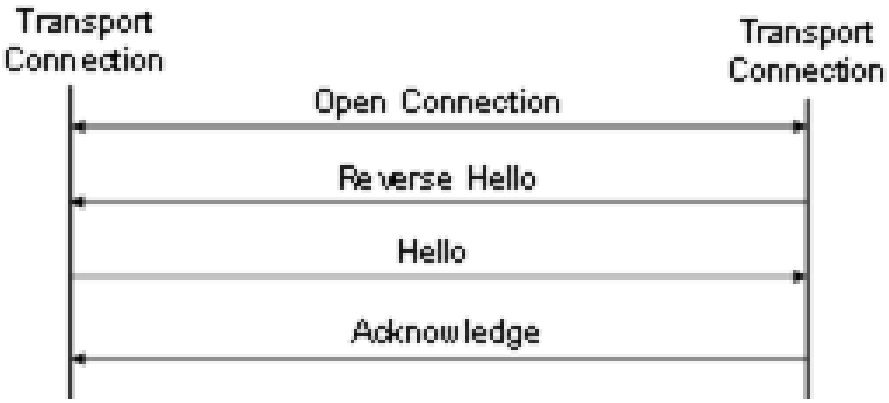
1.2. Взаимодействие между клиентом и сервером.

Для начала взаимодействия между клиентом и сервером необходимо установить соединение:

1. Соединение, инициированное клиентом



2. Соединение, инициированное сервером



Форматы сообщений, необходимых для инициации соединения - OPC UA Connection Protocol:

1. **Hello Message** – маркер начала передачи данных от клиента.

Поле	Тип данных	Описание
ProtocolVersion	UInt32	Последняя версия протокола UACP, поддерживаемая клиентом.
ReceiveBufferSize	UInt32	Самый большой MessageChunk, который может получить отправитель.
SendBufferSize	UInt32	Самый большой MessageChunk, который может отправит отправитель.
MaxMessageSize	UInt32	Максимальный размер любого ответного сообщения.
MaxChunkCount	UInt32	Максимальное количество фрагментов в любом ответном сообщении.
EndpointUrl	String	URL-адрес конечной точки, к которой клиент хотел подключиться.

2. **ReverseHello Message** – маркер начала передачи данных от сервера.

Поле	Тип данных	Описание
ServerUri	String	ApplicationUri сервера, отправившего сообщение. Закодированное значение должно быть меньше 4096 ов.
EndpointUrl	String	URL-адрес конечной точки, которую Клиент использует при установке SecureChannel. Это значение должно быть передано обратно серверу в “Hello Message”.

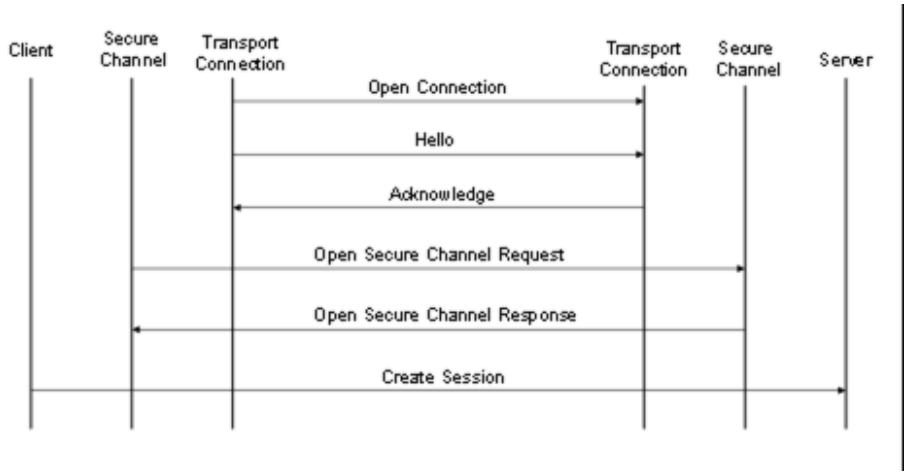
3. **Acknowlege Message** – сообщение, которое сервер посылает клиенту в ответ на сообщение Hello.

Поле	Тип данных	Описание
ProtocolVersion	UInt32	Последняя версия протокола UACP, поддерживаемая сервером.
ReceiveBufferSize	UInt32	Самый большой MessageChunk, который может получить отправитель. Это значение не должно быть больше того, что клиент запросил в Hello Message.
SendBufferSize	UInt32	Самый большой MessageChunk, который может отправит отправитель. Это значение не должно быть больше того, что клиент запросил в Hello Message.
MaxMessageSize	UInt32	Максимальный размер любого сообщения запроса. Размер сообщения рассчитывается с использованием незашифрованного тела сообщения.
MaxChunkCount	UInt32	Максимальное количество фрагментов в любом сообщении запроса.

1.3. Установка защищённого соединения

После установки соединения клиент посылает сообщение OPEN, которое обозначает открытие канала передачи данных с предложенным клиентом методом шифрования.

В ответ сервер отправляет сообщение OPEN, которое содержит уникальный ID канала передачи данных, а также показывает, что он согласен на предложенный метод шифрования (или его отсутствие).



OpenSecureChannel Request - запрос на открытие защищённого канала:

Поле	Тип данных	Описание
requestHeader	RequestHeader	Общие параметры запроса
clientCertificate	ApplicationInstanceCertificate	Сертификат, который идентифицирует клиента
requestType	Enum SecurityToken RequestType	- ISSUE_0 создает новый *SecurityToken* для нового *SecureChannel* . - RENEW_1 создает новый *SecurityToken* для существующего *SecureChannel*
secureChannelId	BaseDataType	Идентификатор *SecureChannel,* которому должен принадлежать новый токен.
securityMode	Enum MessageSecurityMode	Тип защиты, применяемой к сообщениям.
securityPolicyUri	String	URI для *SecurityPolicy*
clientNonce	ByteString	Случайное число, которое не должно использоваться ни в каком другом запросе.
requestedLifetim	Duration	Запрошенное время жизни в миллисекундах для нового *SecurityToken*

OpenSecureChannel Response – ответ на запрос:

Поле	Тип данных	Описание
responseHeader	ResponseHeader	Общие параметры ответа
securityToken	ChannelSecurityToken	Описывает новый *SecurityToken,* выданный *сервером*
channelId	BaseDataType	Уникальный идентификатор *SecureChannel*

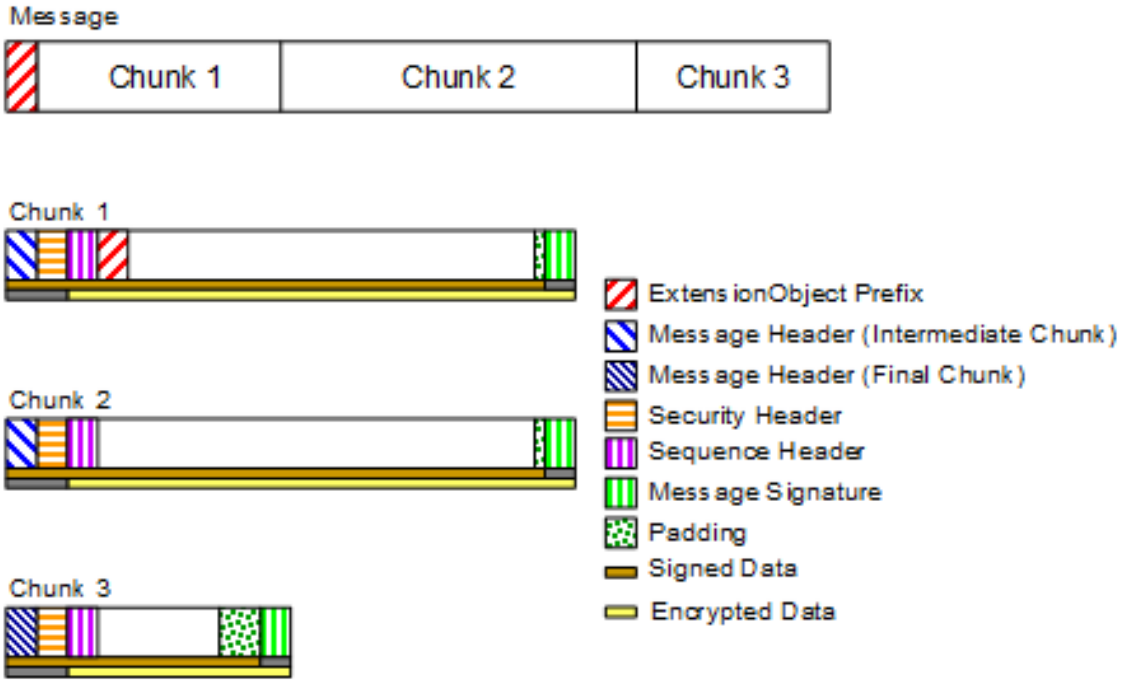
tokenId	ByteString	Уникальный идентификатор для конкретного *SecurityToken*
createdAt	UtcTime	Время создания *SecurityToken*
revisedLifetime	Duration	Время жизни *SecurityToken* в миллисекундах
serverNonce	ByteString	Случайное число, которое не должно использоваться ни в каком другом запросе

Протокол поддерживает три режима безопасности: None, SignOnly и SignAndEncrypt.

OPC UA приложения используют сертификаты X.509 v3 для хранения открытых ключей, необходимых для асимметричной криптографии.

1.4. Обмен сообщениями

После открытия защищенного канала передачи данных клиент и сервер начинают обмениваться сообщениями MESSAGE (MSG). В силу того, что протоколы транспортного уровня имеют ограниченный размер сегмента, все сообщения разбиваются на фрагменты, размер которых не превышает SendBufferSize, принятый клиентом и сервером при установке соединения.



Каждый фрагмент подписывается и данные, которые идут после заголовка безопасности шифруются выбранным на этапе открытия защищённого канала алгоритмом шифрования (если он предусмотрен).

Фрагменты сообщения имеют следующие заголовки:

Message Header описывает тип сообщения.

Поле	Тип ных	Описание
MessageType	Byte	Трехбайтовый код ASCII, определяющий тип сообщения: · MSG - Сообщение, защищенное ключами, связанными с каналом. · OPN - OpenSecureChannel. · CLO - CloseSecureChannel.
IsFinal	Byte	Указывает, является ли фрагмент последним в сообщении. · C - Промежуточный фрагмент · F - Последний фрагмент · A - Последний фрагмент (используется, когда произошла ошибка и сообщение было прервано). Это поле имеет значение только для MessageType из «MSG». Это поле всегда равно «F» для других типов сообщений.
MessageSize	UInt32	Длина фрагмента в байтах
SecureChannel	UInt32	Уникальный идентификатор SecureChannel, присвоенный Сервером. Если Сервер получает SecureChannel, который он не распознает, он должен вернуть соответствующую ошибку транспортного уровня.

Security Header

- **Asymmetric algorithm Security header –** заголовок безопасности в случае асимметричного шифрования.

Поле	Тип данных	Описание
SecurityPolicyUriLength	Int32	Длина SecurityPolicyUri в байтах. Это значение не должно превышать 255 байтов. Если URI не указан, это значение может быть 0 или -1
SecurityPolicyUri	Байт	URI политики безопасности, используемой для защиты сообщения
SenderCertificateLength	Int32	Длина SenderCertificate в байтах Если сертификат не указан, это значение может быть 0 или -1.
SenderCertificate	Байт	X.509 v3 сертификата присваивается отправляющим приложением экземпляра

ReceiverCertificateThumbprint Length	Int32	Длина ReceiverCertificateThumbprint в байтах. В зашифрованном виде длина этого поля составляет 20 байт.
ReceiverCertificateThumbprint	Байт	Индивидуальный код X.509 v3. Это указывает, какой открытый ключ был использован для шифрования фрагмента. Это поле должно быть пустым, если сообщение не зашифровано.

- **Symmetric algorithm Security header –** заголовок безопасности в случае симметричного шифрования.

Поле	Тип данных	Описание
TokenID	UInt32	Уникальный идентификатор SecureChannel SecurityToken, используемый для защиты сообщения.

3. **Sequence header –** заголовок гарантирует, что первый зашифрованный фрагмент каждого *сообщения,* отправляемого по каналу, будет начинаться с разных данных.

Имя	Тип данных	Описание
SequenceNumber	UInt32	Монотонно увеличивающийся порядковый номер, присвоенный отправителем каждому фрагменту, отправляемому по защищенному каналу.
RequestId	UInt32	Идентификатор, присвоенный клиентом сообщению запроса OPC UA. Все фрагменты для запроса и связанного с ним ответа используют один и тот же идентификатор.

1.5. Сервисы

В OPC UA всё дальнейшее взаимодействие между клиентом и сервером основано на вызовах сервисов. Вызовы сервисов состоят из запроса и ответа.

RequestHeader – заголовок запросов:

Поле	Тип данных	Описание
RequestHeader	structure	Общие параметры запросов для всей сессии
authenticationToken	Session AuthenticationToken	Секретный идентификатор сессии
timestamp	UtcTime	Время, когда был отправлен запрос

requestHandle	IntegerId	Идентификатор запроса (возвращается в ответе)
returnDiagnostics	UInt32	Битовая маска, которая идентифицирует вид необходимой диагностики, если она предусмотрена
auditEntryId	String	Идентификатор записи журнала аудита клиента
timeoutHint	UInt32	Тайм-аут, который может использоваться для отмены длительных операций
AdditionalHeader	Extensible Parameter AdditionalHeader	Зарезервировано для использования в будущем

ResponseHeader – заголовок ответов:

Поле	Тип данных	Описание
ResponseHeader	structure	Общие параметры ответов для всей сессии
timestamp	UtcTime	Время, когда был отправлен ответ
requestHandle	IntegerId	Идентификатор запроса, который был получен от клиента
serviceResult	StatusCode	Результат вызова службы
serviceDiagnostics	DiagnosticInfo	Диагностическая информация, если была запрошена клиентом
stringTable[]	String	Параметры диагностической информации
additionalHeader	Extensible Parameter AdditionalHeader	Зарезервировано для использования в будущем

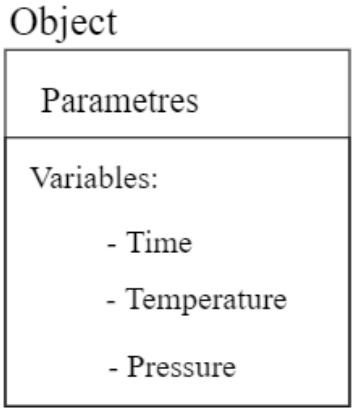
2. Описание лабораторного стенда

2.1. Клиент – сервер.

Для выполнения лабораторной работы была реализована структура клиент-сервер на языке программирования Python 3. Для передачи данных по протоколу OPC UA была использована библиотека freeOPCUA.

В нашем случае в основе адресного пространства OPC UA лежит модель узлов, которая предоставляет серверам удобный способ представления данных клиентам. Объекты и их компоненты представлены в адресном пространстве как набор узлов, описываемых атрибутами и связанных ссылками. Узлы могут быть отдельными объектами, переменными, методами и так дале.

На нашем сервере данные хранятся в качестве переменных объекта Parameters:



Функционал серверного приложения заключается в генерации случайных значений каждые 2 секунды для следующих переменных: температура (в диапазоне от 10 до 50), давление (в диапазоне от 200 до 999) и время (текущее время). Таким образом мы имитируем компоненту АСУ ТП.

Клиент подключается к серверу по порту 4840 и по названию объекта считывает его переменные также каждые 2 секунды.

При получении сигнала прерывания сервер останавливается, прерывая соединение с клиентом. Клиент также прерывает соединение при получении сигнала.

2.2. Виртуальные машины.

Роль клиента и сервера выполняют виртуальные машины Kali Linux 2020.3, которые были созданы в программном продукте виртуализации Oracle VM VirtualBox. Две машины были объединены в одну виртуальную сеть следующим образом:

1. В настройках сети VirtualBox была создана сеть Net.
2. В настройках сети виртуальных машин они обе были подключены к сети Net через один из сетевых адаптеров.

На одной из машин был запущен сервер, а на другой клиент. Трафик фиксировался с помощью программного обеспечения WireShark.

3. Модуль для ПО Suricata. (В стадии разработки)

Suricata — сетевое средство обнаружения и предотвращения вторжений, имеющее открытый исходный код. Suricata позволяет анализировать множество сетевых протоколов, однако очень малое число промышленных протоколов (например, Modbus). Suricata позволяет добавлять собственноручные модули для расширения списка анализируемых протоколов. В рамках лабораторной работы был реализован модуль для протокола OPC UA.

3.1. Файлы app-layer-opcua.[h, c]

Эти файлы отвечают за парсинг полей пакетов и детектирование событий, связанных с протоколом.

- Описание структуры транзакции: (На начальном этапе взяли только поля заголовка)

```
typedef struct OpcuaTransaction_ {
   struct OpcuaState_ *opcua;

   uint32_t    m_type;        /**< тип сообщения */
   uint32_t    length;        /**< длина сообщения */
   uint32_t    channel_id;    /**< id защищённого канала*/
   uint32_t    token_id;      /**< id токена безопасности канала*/
   uint32_t    seq_num;       /**< порядковый номер сообщения */
   uint32_t    req_id;        /**< id запроса*/

   AppLayerDecoderEvents *decoder_events;
   DetectEngineState *de_state;
   AppLayerTxData tx_data;

   TAILQ_ENTRY(ModbusTransaction_) next;
} OpcuaTransaction;
```

- Детектируемые события:

1. Получение пустого сообщения:
{"EMPTY_MESSAGE", OPCUA_DECODER_EVENT_EMPTY_MESSAGE}

2. Последний блок в сообщении (последний байт типа сообщения в этом случае равен F):
{"IS_FINAL", OPCUA_DECODER_EVENT_ IS_FINAL }

3. Неверная длина полей протокола:
{"INVALID_LENGTH", OPCUA_DECODER_EVENT_INVALID_LENGTH}

ПРИЛОЖЕНИЕ

A. OPCUA_server.py

```
from opcua import Server
from random import randint
import datetime
import time
from opcua import ua

server = Server()

url = "opc.tcp://192.168.56.1:4840"
server.set_endpoint(url)

name = "OPCUA_SERVER"
addspace = server.register_namespace(name)

node = server.get_objects_node()

Param = node.add_object(addspace, "Parameters")

Temp = Param.add_variable(addspace, "Temperature", 0)
Press = Param.add_variable(addspace, "Pressure", 0)
Time = Param.add_variable(addspace, "Time", 0)

Temp.set_writable()
Press.set_writable()
Time.set_writable()

server.start()
print("Server started at {}".format(url))

while True:
    try:
        Temperature = randint(10, 50)
        Pressure = randint(200, 999)
        T = datetime.datetime.now()

        print(Temperature, Pressure, T)

        Temp.set_value(Temperature)
        Press.set_value(Pressure)
```

```
            Time.set_value(T)

            time.sleep(2)
        except KeyboardInterrupt:
            print("Closing connection...")
            break
    print()
    server.stop()
    print("Server stopped")
```

B. OPCUA_client.py

```
    from opcua import Client
    import time
    import opcua

    url = "opc.tcp://192.168.56.1:4840"
    client = Client(url)
    client.connect()
    print("Client connected")

    client.load_type_definitions()

    root = client.get_root_node()
    print("Root node is: ", root)
    objects = client.get_objects_node()
    print("Objects node is: ", objects)

    print("Children of root are: ", root.get_children())

    uri = "OPCUA_SERVER"
    idx = client.get_namespace_index(uri)

    while True:
        try:
            Time = root.get_child(["0:Objects", "{}:Parameters".format(idx),
"{}:Time".format(idx)])
            Pressure = root.get_child(["0:Objects", "{}:Parameters".format(idx),
"{}:Pressure".format(idx)])
            Temp = root.get_child(["0:Objects", "{}:Parameters".format(idx),
"{}:Temperature".format(idx)])
            print("Time: ", Time.get_value())
            print("\tPressure: ", Pressure.get_value())
            print("\tTemperature: ", Temp.get_value())
```

```
        time.sleep(2)
    except KeyboardInterrupt:
        print("Closing connection...")
        break

client.disconnect()
print("Connection closed")
```