Practical Pitfalls for Security in OPC UA

Alessandro Erba*†, Anne Müller*†, Nils Ole Tippenhauer*
*CISPA Helmholtz Center for Information Security,
†Saarbrücken Graduate School of Computer Science, Saarland University
[alessandro.erba, anne.mueller, tippenhauer]@cispa.de

***Abstract*—In 2006, the OPC Foundation released the first specification for OPC Unified Architecture protocol, one of the industrial protocols that promises security features such as authentication, authorization, integrity, and confidentiality. Challenges in the practical adoption of those security features by product vendors, libraries implementing the standard, and end-users were not investigated so far.**

In this work, we systematically investigate practical challenges to configure OPC UA securely. In particular, we review 48 artifacts consisting of products and libraries for OPC UA and show that 38 out of the 48 artifacts have one (or more) security issue. In particular, we show that 7 OPC UA artifacts do not support the security features of the protocol at all. In addition, 31 artifacts that partially feature OPC UA security rely on incomplete libraries and come with misleading instructions. Consequently, relying on those products and libraries will result in vulnerable implementations of OPC UA security features. We design, implement and demonstrate attacks in which the attacker can steal credentials exchanged between victims, eavesdrop on process information, manipulate the physical process through sensor values and actuator commands, and prevent the detection of anomalies in the physical process.

I. Introduction

Interconnection of industrial components carries security concerns that need to be addressed at different levels to allow secure industrial plants. One among the others is the security of the communication protocol adopted for Machine to Machine (M2M) communication. In 2006, the OPC Foundation released the first specification for the OPC Unified Architecture (OPC UA) protocol [1], an industrial protocol that promises security features such as authentication, authorization, integrity, and confidentiality. OPC UA is a client/server architecture, e.g., in an ICS the OPC UA server is running on a Programmable Logic Controller (PLC) to publish industrial process sensor readings and control its state. Those values can be read and modified by an OPC UA client running on other industrial devices or end-user systems.

Although OPC UA provides features to ensure security and confidentiality properties, they do not work out of the box. OPC UA requires a correct configuration to guarantee secure communication. For example, OPC UA specifies a number of alternative *securityModes* (i.e., configurations for application authentication and encryption ranging from 'None' to 'SignAndEncrypt'), *securityPolicies* (i.e., algorithms used for asymmetric encryption ranging from 'None' to 'Basic256Sha256') and *UserIdentityToken* (i.e., configurations for user authentication). The main attacks to mitigate are i) Rogue Server (i.e. server impersonation), ii) Rogue Client (i.e. client impersonation), and iii) Middleperson attacks. For example, an attacker can feed wrong information to clients (Rogue Server attack), eavesdrop, and change values which can directly alter the physical process (Rogue Client attack), or both (Middleperson attack). In 2017, the German Federal Office for Information Security (BSI) released the 'OPC UA Security Analysis' [2], reporting that 'No systematic errors could be detected' in the OPC UA standard. In addition, the BSI recommended using only 'Sign' or 'SignAndEncrypt' as security mode, with 'Basic256Sha256' as security policy, and configure a user authentication mechanism.

Nevertheless, there seem to be significant challenges to set up secure OPC UA deployments in practice. In [3], Dahlmanns et al. perform a scan for OPC UA servers reachable over the Internet and find that 92% of the reachable deployments show issues with security configuration. Among those, 44% of the servers are accessible without any authentication requirements. To mitigate that issue, the authors suggest (similar to [2]) to disable insecure security modes and policies and enforce user authentication. Their findings demonstrate that such suggestions are still not applied in a significant number of deployments. The authors believe that this is due to the configuration complexity of the OPC UA security features.

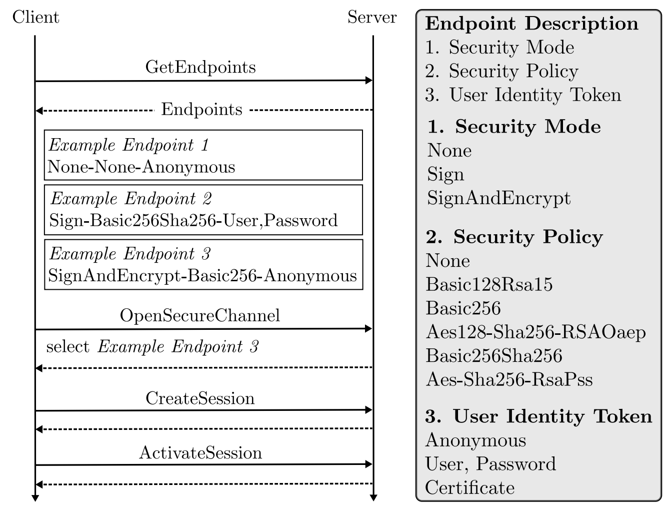
In this work, we systematically investigate practical challenges to configure OPC UA in a secure way and prevent the three attacks outlined above. In particular, we show that many OPC UA products have missing support for security features of the protocol. In addition, products that claim to feature OPC UA security rely on libraries that are incomplete, difficult to use and provide instructions that lead to insecure settings. Lastly, we also show that several popular OPC UA libraries provide vulnerable implementations of OPC UA security features.

To assess the vulnerability of available libraries, we create a framework that implements the three proposed attacks (i.e. Rogue Server, Rogue Client, Middleperson). For each library, we set up a client and/or server application using protocol options that can be expected to provide a secured OPC UA instance, and then attack that system. For cases where the attacks succeed, we further investigate the cause for the vulnerability. In particular, we show that Middleperson attacks are possible if vulnerable servers and clients are present in the same network, and such attacks allow to manipulate the process and prevent reliable control of the system. Surprisingly, we find that even recovery of plaintext credentials from intercepted encrypted communications is easily possible with our Rogue Server and Middleperson attacks. This finding is in contrast to prior work [4], where the authors perform a Middleperson attack on OPC UA applications and conclude that it is impossible to recover user passwords in OPC UA systems (even with security mode 'None').

Contributions. We summarize our contributions as follows

- We systematically analyze proprietary products and identify reoccurring issues with the availability, and setup instructions for OPC UA security features
- We systematically analyze available libraries and identify reoccurring issues with the implementation of security features
- We demonstrate the feasibility of the identified attacks by implementing all three attacks with a proof-of-concept application (we plan to release a proof of concept of our framework as open-source)

The remainder of this work is organized as follows. In II we present the details of the OPC UA protocol. In III we present the system and attacker model, as well as the approach we use to conduct our research. In IV we explain how we build the framework that we use to analyze the available libraries. In V we report the results of our research. A discussion of our findings is presented in VI, followed by conclusions in VIII.



(a) (b)

Fig. 1. Figure (a) gives an overview over the connection establishment procedure between client and server. The server provides a list of endpoints, where each endpoint consist of a Security Mode, a Security Policy and a User Identity Token. The client selects which endpoint it wants to connect to. If the mode Sign or SignAndEncrypt is chosen the server and client each have to send their public certificate along with the endpoint to establish the secure channel. Figure (b) shows the options available for the endpoint configuration. The Security Mode, Security Policy and User Identity Token can be chosen independently from each other.

II. Background

In this section, we present the background related to the OPC UA protocol and its security features. OPC UA features several options to change the level of security achieved for a connection.

A. *OPC Unified Architecture*

OPC UA [1] is an industrial communication protocol developed by the OPC foundation. The protocol allows platform-independent and secure communication by design. The first version of OPC UA Unified Architecture was released in 2006, followed by four other releases up to the current specification 1.04 released in 2018. Eighteen parts constitute the protocol specification, each describing a feature of the protocol. Part 2 describes the security model of OPC UA, and Part 4 describes the services that implement security primitives.

The standard details two communication strategies: a client-server model and a publisher-subscriber model. Both models allow messages to be signed to ensure authenticity or signed and encrypted to add confidentiality. OPC UA does not enforce the use of security features, a mode with no security is also possible.

Figure 1 reports an example of a secure connection establishment in client-server communication. A server can offer several endpoints, each defined by a Security Mode, a Security Policy, and the supported User Identity Token(s). The Security Mode defines how messages are exchanged between parties to achieve authentication, confidentiality, and integrity guarantees. Available Security Modes are None, Sign, SignAndEncrypt (Security Mode Sign and SignAndEncrypt enforce application authentication). Security Policies define the cryptographic primitives used for the specific security mode: None, Basic128Rsa15 (Deprecated), Basic256 (Deprecated), Aes128-Sha256-RsaOaep, Basic256Sha256, Aes256-Sha256-RsaPss. Finally, the UserIdentityToken defines the supported user authentication methods for an endpoint: Anonymous (no user authentication), Username&Password, Certificate [5].

When initiating a connection, the client can choose the endpoint to connect to (from an unauthenticated list of endpoints provided by the server).

B. *Certificate Management in OPC UA*

In order to establish a secure connection (Security Mode different from 'None'), OPC UA applications (Clients and Servers) need to perform application authentication by exchanging their Application Instance Certificates. Upon receiving a certificate, an application needs to decide whether the received application certificate is trustworthy and the connection can be established or not. To this end, each OPC UA application has a list of certificates that are trusted (Certificate Trustlist). This list can contain self-signed certificates or certificates from Certificate Authorities (CA). In the first case, the certificates are often exchanged manually between applications, but the use of a Certificate Manager with a Global Discovery Server (GDS) is also possible.

A GDS can provide two main functionalities. As the name 'Global Discovery' suggests: OPC UA instances can query the GDS to find available applications that have previously registered with the GDS. Additionally, a GDS can offer the CertificateManager and allow applications to push their certificate to the GDS or pull certificates from the GDS to update the Trustlist.

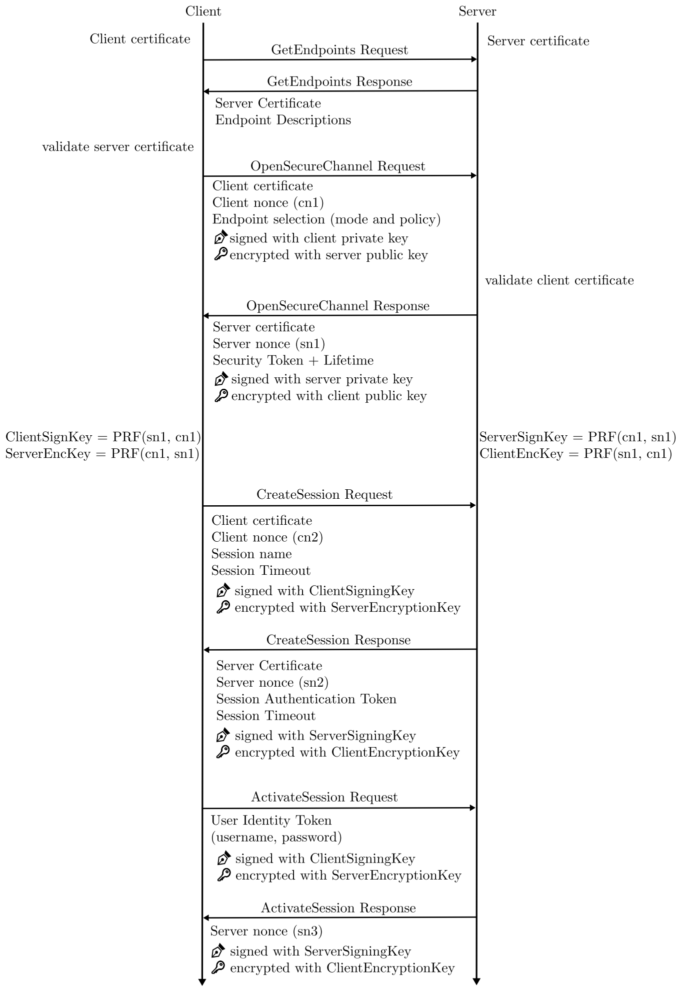


Fig. 2. Connection establishment between client and server using Security Mode SignAndEncrypt.

The server sends its certificate as part of the `GetEndpoints()` response. If the client selects an endpoint with Security Mode Sign or SignAndEncyrpt, the following OpenSecureChannel message is encrypted with the server public key and signed with the client's private key. This message also contains the client certificate. If the server decides to trust the client's certificate, the response is encrypted with the client's public key and signed with the server's private key. The `OpenSecureChannel` Request, and Response messages contain a nonce from client and server in the respective message. Client and server compute the symmetric signing and encryption keys used in the session starting from the nonces [6]. Figure 2 shows the complete secure connection establishment procedure.

C. Service Sets

OPC UA offers its functionalities through 10 service sets. Each service consists of a Service request and Service response, identifiable via RequestHeader and ResponseHeader.

The ten service sets (and therefore the functionalities) offered by the protocol are Discovery, SecureChannel, Session,

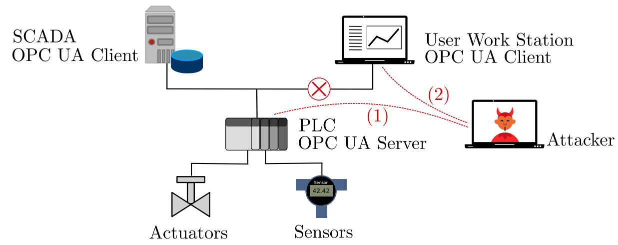


Fig. 3. The figure shows a network consisting of an OPC UA server and two clients. An attacker that wants to establish themselves as a PitM has to (1) pose as an OPC UA client towards the legitimate OPC UA server and (2) as an OPC UA server towards a legitimate client.

NodeManagement, View, Query, Attribute, Method, MonitoredItem, and Subscription.

Each service set specifies a different functionality as defined in Part 4 of the OPC UA protocol [7]. For example, a client can read and write values related to the processes on OPC UA Nodes in the server through the Attribute service set. It is also possible to provide the functionality for the clients to add nodes. Moreover, clients can call methods (through the Method service set) defined for Object Nodes. Through these functionalities, a client can directly influence the plant operation.

The OPC UA server can authorize clients to use its functionalities through user authentication. Session service set provides the user authentication functionalities (e.g., login with username and password). Authentication occurs during session activation, after the initial authentication via certificates.

III. OPC UA Security Assessment Methodology

In this section, we present our approach for the security assessment of OPC UA artifacts. OPC UA features a number of cryptographic options to establish secure channels (see Section II). Like in any system, authentication of parties to each other (application authentication in OPC UA) requires either pre-shared secrets or certificates with public keys. In this work, we argue that in the ICS setting (in which OPC UA is adopted), no public PKI (e.g., with root CAs) is available, so a core issue is how to establish the initial trust between the communicating parties. There are two options in OPC UA to tackle the certificate management and distribution: i) Self-signed certificates that need to be distributed manually among parties that are willing to communicate in the OPC UA network, to build the so-called *Trustlist*. ii) Through *CertificateManager* functionality as described in part 12 of the OPC UA standard. The focus of our security assessment is the application authentication functionality required for the security in OPC UA.

We start presenting our system and attacker model, and research challenges. Then we introduce our framework for OPC UA security assessment.

A. System and Attacker Model

We assume a local ICS network with OPC UA server as depicted in Figure 3. The system operator guarantees security in the OPC UA system by allowing the server to offer only secure endpoints (i.e., Sign or SignAndEncrypt, see Section II) as suggested by the BSI [2]. We note that Intrusion Detection Systems are out of scope in our model as we aim to look at security guarantees from the OPC UA artifacts. We assume the network operator follows the user manual shipped with the deployed ICS hardware/software to configure the OPC UA server and client security. In this work, we consider three attackers with different goals[1].

Rogue Server. A new device is introduced and now needs to establish secure OPC UA communications with other devices. An attacker is present in the network and aims to manipulate OPC UA clients by providing malicious information or stealing of OPC UA user credentials. The attacker creates a server that offers secure endpoints to establish a secure connection with new clients and make them believe that they are communicating with the actual OPC UA server in the network.

Rogue Client. An attacker aims to connect to the OPC UA server to eavesdrop or manipulate the information shared between the server to its clients. The attacker creates a client that attempts to connect to the server although not authorized by the network operator.

Middleperson attack (PitM). An attacker aims to establish themselves as Middleperson in the connection between the client and server, intercepting and manipulating all communications between both. In general, this requires achieving Rogue Client and Server objectives.

B. Research Questions and Challenges

With our research, we aim to address the following research questions. **R1.** What are practical challenges for the correct use of OPC UA security features? **R2.** Are OPC UA security features correctly implemented by the vendors and products? **R3.** What are the consequences of breaking OPC UA security features?

While addressing those research questions, we tackle the following research challenges: i) Partially proprietary products without source code. ii) available OPC UA libraries partially documented, iii) unavailability of products (or real deployments) for testing.

C. Proposed Approach

The focus of our approach is the analysis of the security features implemented by OPC UA software available. In particular, we focus on key management functionalities implemented in OPC UA artifacts i.e. i) Proprietary hardware and software that implements OPC UA stack, ii) available libraries that implement the OPC UA stack.

To address R1, we survey proprietary and available OPC UA enabled products. We investigate practical challenges for correctly configuring secure OPC UA setups in the analyzed products checking availability of features.

[1]We use the gender-neutral 'they' as pronoun in this work

To address R2, we propose a framework to verify the correctness of implementations of OPC UA security features in hardware and software products. For proprietary products: we consult user manuals to verify the correctness of security features. For available libraries, we practically tested the security features by deploying OPC UA clients and server locally. In general, we investigate correct key management in OPC UA enabled products. In particular, with our framework, we tested specific attacks against OPC UA client and server implementations, which are feasible due to erroneous or incomplete key management in OPC UA enabled products. For each library, we create the OPC UA server and clients following the library instructions (first the default configuration, second the optional features) and test their vulnerability to the different attacks presented in the Attacker Model (Section III-A). We plan to release a proof of concept of our framework as open-source.

To address R3, we show that breaking the application authentication of any OPC UA security configuration will have severe consequences on OPC UA security. An attacker will be able to obtain plain-text passwords when legitimate clients connect to Rogue Server or the Middleperson system and perform user authentication, to modify the data seen by OPC UA clients and servers, and to execute function calls on an OPC UA server that can interfere with the physical process (e.g. send commands to actuators). We will discuss this further in the next Section.

IV. Framework and Implementation

In this section, we present our framework for the security assessment of OPC UA artifacts. First, we will describe our framework from a high-level perspective; then, we provide the details about the framework implementation used to test the OPC UA artifacts. The results of our security assessment are presented in Section V.

A. Framework

Our framework tests OPC UA clients and OPC UA servers against Rogue Server, Rogue Client, and Middleperson attacks. For OPC UA servers, the framework identifies vulnerabilities to Rogue Client Attacks, while for OPC UA clients the framework identifies vulnerabilities to Rogue Server Attacks. The framework test for vulnerabilities generated by incorrect (or incomplete) Trustlist management (that deviates from the prescriptions of the OPC UA protocol). When both Rogue Client and Rogue Server vulnerabilities are present in the OPC UA network, Middleperson attacks can be exploited.

Rogue Server. In Figure 4 we report the steps followed by our framework to verify the vulnerability to the Rogue Server. The test employs an OPC UA Rogue Server to test OPC UA clients. The Rogue Server waits for a `getEndpoints()` request from a victim client and replies offering secure endpoints and its self-signed certificate that is not present in the client's Trustlist. The victim client receives the endpoints and the certificate (server Application Instance Certificate). The Trustlist of the OPC UA victim client does not contain the Rogue Server's public key (Application Instance Certificate).

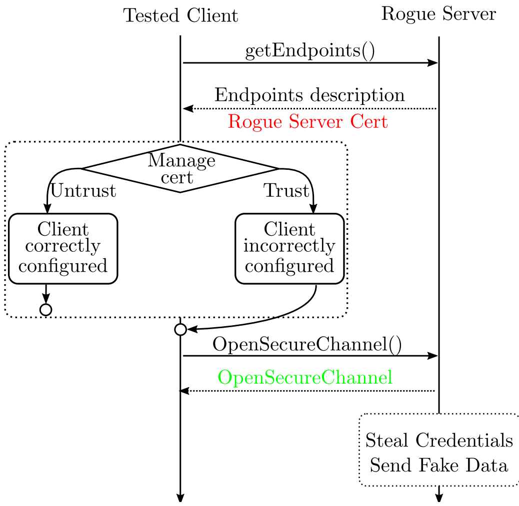


Fig. 4. Rogue Server workflow. The server waits for a new incoming connection. When the Server receives a getEndpoints() request it provides a certificate that was not trusted on the client side upon connection (highlighted in red). If the client trusts the server certificate and performs an OpenSecureChannel() request to the server, the Rogue Server allows the connection (highlighted in green) and the attacker can perpetrate his attacks e.g. steal credentials and send manipulated data. The client is misconfigured and vulnerable to Rogue Server attack.

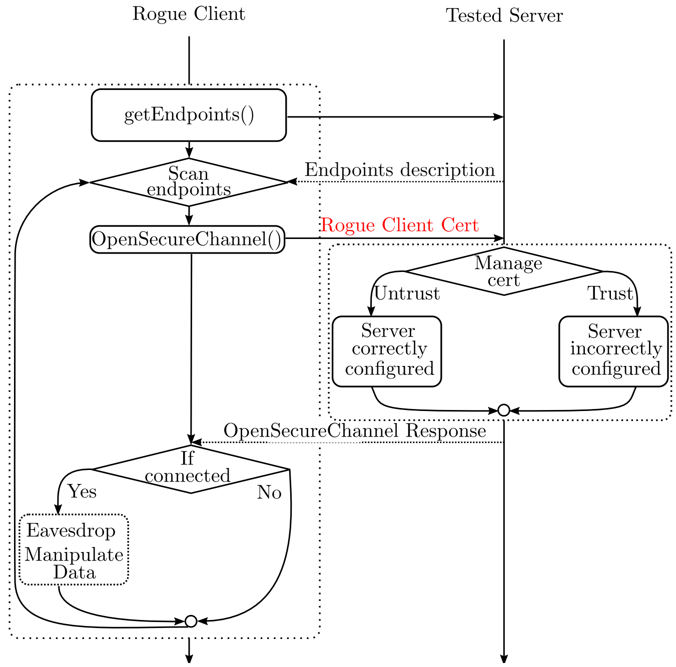


Fig. 5. Rogue Client workflow. The dashed boxes contain the flowchart representing the application server/client logic. Diamonds represent a decisions by the client/server according to the data flow and management. The client lists all the secure endpoints on the server and for each one it tries to connect providing an arbitrary self signed certificate (highlighted in red) that was not shared with the server in advance. If the client successfully connects, the server is not correctly managing certificates.

Hence, the victim client should not establish a secure connection with the untrusted Rogue Server as the root of trust between client and Rogue Server was not established upon connection. If the victim client is correctly configured, it will not continue the interaction with the Rogue Server. Otherwise, the victim client trusts the Rogue Server and instantiates an `OpenSecureChannel()` request that the Rogue Server accepts.

As a consequence of a Rogue Server attack, an attacker can send fake data to the victim client and steal user credentials. User credentials stealing occurs when the `ActivateSession()` request is instantiated by the Client. If the Rogue Server endpoints are configured with 'UserIdentityToken' user and password, the attacked client will be required to authenticate (activate a session) to establish a connection with the Server. OPC UA session activation request (that contains user and password) is encrypted with the keys derived from the Nonces exchanged in the `OpenSecureChannel()` with the Rogue Server. Moreover, the password is encoded in UTF-8 and encrypted with the server public key (i.e., the key received from the untrusted Rogue Server). Despite the different encryption techniques applied to encrypt the password before transmission, the Rogue Server will decrypt them because the client allows the connection with untrusted parties.

Rogue Client. In Figure 5 we report the steps followed by our framework to verify the vulnerability to the Rogue Client in OPC UA servers. The test employs an OPC UA Rogue Client to the test OPC UA server implementations. The Rogue Client scans all the endpoints offered by an OPC UA server and tries to establish a secure connection to all of them, one by one. The Trustlist of the OPC UA server does not contain the Rogue Client's certificate. Hence, the Rogue Client should not be entitled to establish a secure connection with the server because the root of trust between client and server was not acknowledged upon connection. If the client succeeds in connecting, the server implementation is managing erroneously the certificates, and it is deviating from the expected behavior prescribed by OPC UA protocol. The server is then considered vulnerable to Rogue Client attacks.

As a consequence of a Rogue Client attack, an attacker can perform different actions on the server according to the server configuration. The attacker possibilities range from reading values published by the attacked OPC UA server to writing values and executing commands on the server that influence the physical process.

Middleperson. Figure 6 reports the steps followed by our framework to verify the vulnerability to the Middleperson attack. The test leverages the concepts of Rogue Client and Rouge Server used at the same time to achieve the Middleperson attack. The attacker instantiates a Rogue Server in the network. The client requests a secure connection with the Rogue Server (as described in the previous paragraphs). If the attacked server requires user authentication with username and password, the Rogue Server can use the same user identity token to the victim's client and steal the credentials. At this point, the Rogue Client instantiates a connection with the victim's server by providing the stolen credentials and takes

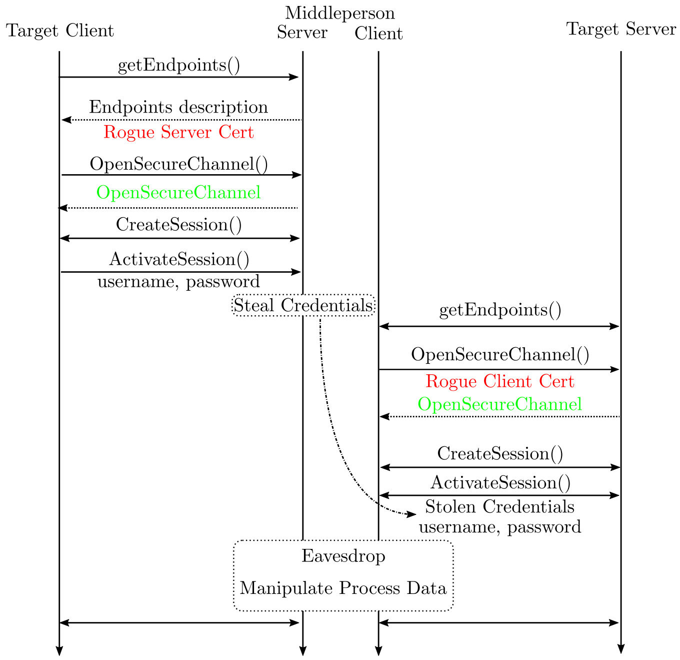


Fig. 6. Example of Middleperson attack with stealing of session credentials (User Identity Token: User, Password). The attacker acts both as Rogue Client and Rougue Server. The Target Client connects to the Middleperson Server, that provides to the client a self signed certificate. The client is vulnerable to Rogue Client attacks and trusts the server cert without verification and instantiates a OpenSecureChannel() request. The Rogue Server allows the connection. When the client instantiates a session the server accepts it and decrypts the provided credentials. At this point the Middleperson connects to the Target Server (that is vulnerable to Rogue Client Attacks) and creates a session providing the credentials stolen from the Target Client.

control of the industrial process (as described in the Rogue Client paragraph). The attacker forwards stealthily (from the process operators) the information received from the victim client to the victim server and vice versa.

Consequences of the Middleperson attack comprise the union of the outcomes identified for Rogue Client and Rogue Server.

B. *Implementation*

We implemented our framework (i.e., Rogue Server, Rogue Client, and Middleperson) using the python opc-ua [8] open source library. The implementation serves as a Proof of concept (POC) of our attacks. We have implemented the POC using the python opc-ua since the library offers support to all the functions required to implement our proposed attacks. Moreover, the functionalities are easy to prototype and deploy.

Our framework consists four python modules, `framework`, `rogueserver`, `rogueclient`, and `utils` for a total of 571 lines of code. For each of the attacks that we implemented (Rogue Server, Rogue Client, and Middleperson), we report a description of how we realized functionalities through the python modules.

Framework module. `framework` module interacts with the attacker. It offers a command-line interface to select the attack to perform (i.e., Rogue Server, Rogue Client, or Middleperson).

Rogue Server attack. When Rogue Server attack is selected, the `rogueserver` module performs the following to mount and start the attack. First, the program executes port scanning in the network to identify available OPC UA servers on port 4840. The attacker selects a benign server that they want to clone with the Rogue Server attack. At this point, the sequence of unencrypted and unauthenticated primitives `FindServers()` and `GetEndpoints()` requests are sent to the benign server to retrieve endpoints information, server information, and the server certificate. A Rogue Server is created with this information. The Rogue Server has the same name, offers the same endpoints (same security mode, security policy), the same user identity token, and provides a self-signed certificate filled with the same information as the benign server. When Rogue Server is configured, port forwarding is enabled in the network to route requests intended to the victim's server to the Rogue Server. Finally, the server is started and waits for a victim client to connect to it without performing certificate validation. If the connection succeeds, the Rogue Server starts publishing fake data, and if the victim's client provides user credentials, the Rogue Client decrypts them.

Rogue Client attack. When Rogue Client attack is selected, the `rogueclient` module performs the following actions to mount and start the attack. First, the attack performs port scanning in the network to identify available OPC UA servers on port 4840. The attacker chooses the victim server that they want to connect through the Rogue Client attack. At this point, the client attempts the `OpenSecureChannel()` request to the victim server endpoints (one by one), providing a self-signed certificate that was not inserted in the victim server Trustlist upon connection. If the cert is trusted, the attack is successful. Furthermore, if the victim server requires the user identity token 'None': the Rogue Client instantiates a `ActivateSession()` request and can start reading (or writing) values at server nodes. If a user identity token with username and password is required, the attacker can input them (e.g., they retrieved them with a Rogue Server attack). Finally, the `ActivateSession()` request is sent to the server.

Middleperson attack. When the Middleperson attack is selected, our POC realizes the attack depicted in Figure 6. The `rogueserver` and `rogueclient` modules are used in parallel (i.e., in threads). First, the Rogue Server is created to capture victim client requests in the network and acquire user credentials. Once the credentials are retrieved, the Rogue Client is instantiated and connects to the victim's server providing the stolen credentials. The attack is successful if both the victim's client and server do not populate the Trustlist upon connection.

V. Assessment Results

In this section, we present the results of our security assessment of 22 OPC UA products by different vendors (i.e., mostly

OPC UA servers for PLCs), and 26 systems we build using 16 available libraries. We start by investigating the availability of features of the OPC UA stack implemented in the considered artifacts. Then we perform a security assessment for OPC UA products with our framework and look for vulnerabilities to Rogue Client, Rogue Server, Middleperson attacks.

A. Adoption of OPC UA features

In this section, we investigate which features of OPC UA stack are implemented by OPC UA enabled artifacts. We consider proprietary products, and available software.

Proprietary OPC UA products. In Table I we report the results of our research about the adoption of OPC UA features that we conducted over 22 proprietary OPC UA products that are offered by vendors to configure their industrial devices. The table is organized into three parts, i) support of publish-subscribe, ii) compatibility with Global Discovery Server, iii) security features. Out of 22 software packages that were analyzed, four vendors rely on Codesys automation software to implement OPC UA features.

None of the considered products support the publish-subscribe model. This feature, announced in the first semester of 2018, is not yet supported by products. In October 2020, Codesys software released the CODESYS OPC UA PubSub SL [31] extension that supports Publish-Subscribe, but at the moment of writing, vendors do not integrate it yet.

Four vendors mention compatibility with GDS servers, but it is not clear if these vendors also offer their implementation of a GDS server or provide functionality for their products to connect to third party software.

Concerning the security features, out of 22 OPC UA servers, three vendors (Beijer, Honeywell, and Yokogawa) do not support security features. It means that deploying an OPC UA network with industrial devices from those brands will always result in an insecure deployment since the only supported security policy is None. A total of 3 vendors (Bachmann, Beckoff, and Lenze) support security with deprecated cryptographic primitives, making their applications de facto insecure. For the remaining 16 products that support security features, we have looked at how the user manual guides the customers through the server configuration. Specifically, we observed that two vendors (Mitsubishi and National Instruments) instruct their users to configure security None. Then, three vendors (B&R, Omron, and Panasonic) discourage the use of None, and six vendors (Codesys, Hitachi, Rockwell, Siemens, Wago, Eaton) do not give recommendations on the preferred policy. Furthermore, four vendors (ABB, General Electric, Schneider, Wiedmüller) recommend a specific Security Policy (Schneider and Weidmüller use security mode SignAndEncrypt as default). Finally, one vendor (Bosch Rexroth) does not support security mode None, thereby enforcing authenticated connections. We report in the table if the certificate Trustlist is supported as required by the OPC UA specification. As we can see from the table, most vendors support it but, there are several problems in the configuration procedure detailed in user manuals that make deployments vulnerable to the three attacks considered in our manuscript. We will detail the configuration issues in the following subsections.

Available OPC UA libraries. In Table II we report the results of our research about the adoption of OPC UA features that we conducted over 16 available libraries to deploy OPC UA in industrial plants. The table is divided into four parts: support of publish-subscribe, compatibility with Global Discovery Server, server security features, client security features.

Out of 16 libraries: 11 libraries implement server features, and 15 libraries implement client features. Specifically, 10 libraries offer server and client features, 5 offer solely client features, and 1 offers solely server features. Publish-Subscribe is implemented by 2 libraries (open62541, S2OPC), also in this case, this feature is not widely adopted. At the moment of writing, the OPC Foundation official implementation (UA .NET) does not support this connection mode. GDS is implemented by 1 library (UA .NET). With respect to security features implemented in servers, 10 out of 11 offer security features. Regarding the security features implemented in clients, 12 out of 15 implement security features. Moreover, we have investigated the correctness of security features implementation. Specifically, we looked at the availability of the Trustlist for certificate verification as described in the OPC UA standard. Among server implementations, 6 (Eclipse Milo, node-opcua, Rust opcua, open62451, S2OPC, and UA .NET) offer this feature, while the other 5 (ASNeG, LibUA, OpenScada UA, Python-opcua) do not allow the server to verify to which clients they are communicating with. Among the client implementations, 6 (Eclipse Milo, Rust opcua, open62541, UA .NET, opc-ua-client, and UAExpert) offer Trustlist functionality, while 4 libraries (LibUA, node-opcua, OpenScadaUA, Python-opcua) do not provide the feature to verify the party that they are communicating with. Regarding the remaining 2 implementations (Golang opcua, and S2OPC), we run into issues while verifying their secure connection functionalities that prevented us from testing the availability of Trustlist features. Finally, we check the security features and verify their correct configuration in the demo applications provided in their repository. The following subsections will detail the pitfalls and problems that we found in the security of available implementations, making them vulnerable to the considered attacks.

B. Vulnerability to Rogue Server

We tested the vulnerability of 15 Client implementations from available libraries, as vendors do not offer OPC UA client functionalities (apart from Bachmann). We found that all available Client demo applications are configured to connect to a security point with mode None, i.e., no security. For 3 libraries (ASNeG, Free OpcUa, opcua ts) no other mode is possible. The ASNeG library will support security features starting from the next release. The 12 remaining libraries support secure connections.

For those 12 libraries, we verified their support to Trustlist for certificate management and tested it with our framework

TABLE I

ADOPTION OF FEATURES OPC UA IN PROPRIETARY PRODUCTS. ● DENOTES THAT THE FEATURE IS SUPPORTED BY THE PRODUCT. ○ DENOTES THAT THE FEATURE IS NOT SUPPORTED BY THE PRODUCT. ◐ DENOTES THAT THE FEATURE IS SUPPORTED BUT THERE ARE PROBLEMS WITH ITS CONFIGURATION THAT MAKE INDUSTRIAL DEPLOYMENTS INSECURE.

Vendor	Platform	Pub-Sub	GDS	Security	Trustlist	Recommended Policy
B&R	ADI OPC UA [9]	○	○	●	●	Not specified
Bachmann	OPC UA Client and Server [10]	○	○	◐	◐	Deprecated protocols
Beckoff	TC3 OPC UA [11]	○	○	◐	◐	Deprecated protocols
Beijer	iX Developer [12]	○	○	○	○	None
Bosch Rexroth	ctrlX CORE [13]	○	○	●	◐	None not supported
General Electric	iFIX [14]	○	compatible	●	◐	Basic256Sha256
Honeywell	ControlEdge Builder [15]	○	○	○	○	None
Lenze	Easy Starter [16]	○	○	◐	◐	Deprecated protocols
Mitsubishi	MX Configurator-R [17]	○	○	●	●	None
National Instruments	InsightCM [18]	○	○	●	●	None
Omron	SYSMAC-SE2 [19]	○	○	●	●	Not specified
Panasonic	HMWIN Studio [20]	○	compatible	●	◐	Not specified
Rockwell	Factory talk linx [21]	○	○	●	●	Not specified
Schneider	Control Expert [22]	○	○	●	●	Basic256Sha256
Siemens	STEP 7 [23]	○	compatible	●	◐	Not specified
Weidmüller	u-create studio [24]	○	○	●	●	Basic256Sha256
Yokogawa	SMARTDAC+ [25]	○	○	○	○	None
Codesys based platforms						
Codesys	Codesys V3.5 [26]	●	○	●	◐	Not specified
ABB	Automation Builder [27]	○	○	●	◐	Basic256Sha256
Eaton	XSOFT-CODESYS [28]	○	○	●	◐	Not specified
Hitachi	HX Codesys [29]	○	○	●	◐	Not specified
Wago	e!cockpit [30]	○	compatible	●	◐	Not specified

implementation. In 4 libraries (LibUA, node-opcua, Open-Scada UA, and python-opcua), the Trustlist is not supported.

In 3 libraries, we had issues configuring and running the client application in a secure configuration. In Eclipse Milo, the demo client does not perform certificate validation and the documentation does not provide details about how to enable the Trustlist on although the feature is present in the source code. In S2OPC the source code features the Trustlist management, but we were not able to test it due to errors and missing details for the configuration. Finally, in Golang opcua we were not able to find information related to the Trustlist neither in the documentation, issues in the repository, or the code, hence we assume that this feature is not supported.

In 5 clients, the Trustlist is supported, but we found two types of insecure behavior that make these clients vulnerable to Rogue Server attacks:

i) Trustlist disabled by default In 4 libraries (Opcua Rust, UA.Net, and open62541, opc-ua-client) the demo client accepts all server certificates by default. The user can disable this option, then the mentioned libraries behave correctly w.r.t. the certificate validation procedure. While this setting is only meant to be used during development and testing, it is an additional hurdle that can lead to a seemingly secure application that is vulnerable to a Rogue Server attack.

ii) Use of unencrypted primitives to perform certificate exchange In one implementation (UAexpert with GUI interface) the instructions guide the user to perform a 'GetEndpoints' request to retrieve the Server certificate. The server replies to the client sending his certificate to the client. The Client prompts the error 'BadCertificateUntrusted' since the server certificate fails the security check. At this point, the client is asked to trust the certificate and re-instantiate a secure connection. This behavior is susceptible to Rogue Server attacks since the certificates are exchanged through an insecure channel. UAexpert offers the option to trust certificates before establishing a connection to servers, but this is not the default workflow proposed in the instructions.

Overall all 12 clients that support security features exhibit vulnerabilities that can be exploited in a Rogue Server attack. Even libraries that have handled security correctly on the server-side are lacking security features on the client-side, forcing users to lower the security properties of their OPC UA deployments.

C. Vulnerability to Rogue Client

In this section, we report the results of our security assessment w.r.t. the vulnerabilities to the Rogue Client for the OPC UA artifacts.

TABLE II

LIBRARIES WITH OPC UA IMPLEMENTATION. ● DENOTES THAT THE FEATURE IS SUPPORTED BY THE PRODUCT. ○DENOTES THAT THE FEATURE IS NOT SUPPORTED BY THE PRODUCT. ◐ DENOTES THAT THE FEATURE IS SUPPORTED BUT THERE ARE PROBLEMS WITH ITS CONFIGURATION. * DENOTES THAT THE FEATURE IS GOING TO BE INTRODUCED IN NEXT RELEASE. THE COLUMN SECURITY REPORTS WHETHER THE LIBRARY IMPLEMENTS SIGNING AND ENCRYPTION. THE COLUMN TRUSTLIST REPORTS WHETHER THE LIBRARY IMPLEMENTS APPLICATION AUTHENTICATION. THE COLUMN DEMO BEHAVIOUR REPORTS WHETHER A SECURE CONNECTION TO THE DEMO APPLICATION IS POSSIBLE.

				Server			Client		
Name	Lang.	Pub-Sub	GDS	Security	Trustlist	Demo Behavior	Security	Trustlist	Demo Behavior
ASNeG [32]	C++	○*	○	●	○	-	○*	-	-
Eclipse Milo [33]	Java	○	○	●	●	●	●	◐	○
Free OpcUA [34]	C++	○	○	-	-	-	○	-	-
LibUA [35]	C#	○	○	●	○	○	●	○	○
node-opcua [36]	.js	○*	○	●	●	◐	●	○	○
opc-ua-client [37]	C#	○	-	-	-	-	●	●	◐
opcua [38]	Rust	○	○	●	●	◐	●	●	◐
opcua [39]	Golang	○	○	-	-	-	●	-	-
opcua [40]	TypeScript	○	-	-	-	-	○	-	-
opcua4j [41]	Java	○	○	○	-	-	-	-	-
open62541 [42]	C	●	○	●	●	◐	●	●	◐
OpenScada UA [43]	C++	○	○	●	○	○	●	○	○
Python-opcua [8]	Python	○	○	●	○	○	●	○	○
S2OPC [44]	C	●	○	●	●	●	●	◐	◐
UA.NET [45]	C#	○	●	●	●	●	●	●	◐
UAexpert [46]	C++	○	○	-	-	-	●	●	◐

Vendors. Since we do not have access to actual devices, our analysis relies on the official user manuals shipped with products. The results for the 22 analyzed products are reported in Table I. The vendors Yokogawa [25], Honeywell [15] and Beijer [12] do not support any security features for OPC UA: this means that they do not offer secure communications channels to send information to clients and will not be able to perform application authentication.

For the remaining 19 companies, we have investigated if they use the Trustlist to enable only certain clients to connect to the server. All the companies implement the Trustlist for certificate verification, only 7 out of 19 correctly instruct the users to configure it. We found that 12 companies report insecure instructions to perform the certificate exchange necessary to build the Trustlist that makes them vulnerable to Rogue Client attack. In particular, we have identified two different issues with the instructions:

i) Trustlist disabled by default The instructions by default guide the user that enables security to configure the server to accept all certificates and optionally the user can configure the Trustlist. If the server has default settings, an attacker can connect a client to the server providing an arbitrary certificate that is not verified upon trusting it. Siemens and Bachmann's products are affected by this issue.

ii) Improper use of Secure Channel primitives to perform certificate exchange The issue resides in the procedure used to exchange certificates. The product affected by this issue leverages the `OpenSecureChannel()` request to move the client certificate from client to server (instead of building the Trustlist before any connection in the network as required in the OPC UA standard). This behavior is an improper use of the OPC UA standard that can be leveraged by an attacker to install its Rogue Client certificate on a target server. On the server-side, the certificate is manually or automatically trusted. An operator would need to carefully check the certificate thumbprint to notice that the installed certificate is not authorized. We identified this as a common error in the instructions from 10 different vendor products (ABB, Beckoff, Bosch Rexroth, Codesys, Eaton, General Electric, Hitachi, Lenze, Panasonic, and Wago). All 4 products analyzed based on Codesys software propagate this error present in the Codesys documentation. This problem may potentially be threatening more than 400 device manufacturers that rely on this Codesys software.

As showed in Section IV if the initial certificate exchange is incorrect, the security guarantees of the OPC UA standard are lost and, the signed/encrypted channels do not guarantee communication security as the authentication of the communicating parties is flawed.

Libraries. We tested the vulnerability to Rogue Client attack of the 10 server libraries that offer security features. This evaluation is done with our framework implementation as described in section IV. To perform our test, we started from demo applications shipped with the libraries and then verified if the library itself has additional capabilities not used in the demo to manage security features. If additional features are available, we add them to the server to test the library.

All libraries provide a demo server or a tutorial explaining how to set up a simple OPC UA server. Out of the considered demo applications, 5 demo applications offer secure and insecure endpoints, and 5 offer exclusively security None. For demo applications where only insecure endpoints are supported, any client can connect to the server via the security mode and policy None (i.e., without securely authenticating).

Next, we tested the certificate management functionalities offered through Trustlist (mode Sign or SignAndEncrypt). We found that 6 libraries support the Trustlist of certificates, and 4 do not support it.

With our framework Rogue Client we tested the 6 demo servers provided by the implementations that support the Trustlist. Our framework tries to establish a connection in mode Sign or SignAndEncrypt where the client provides a self-signed certificate, which was not listed on the server before connecting. According to the OPC UA standard, such connections should be rejected by the server.

In 3 cases the demo server rejects connection from untrusted clients (Eclipse Milo, UA.Net, s2opc). The demo server from Eclipse Milo, available at this address 'opc.tcp://milo.digitalpetri.com:62541/milo', rejects unknown clients, and users are required to upload their client certificates to the server to appear in the server Trustlist and connect. Similarly, the demo servers provided by the UA.Net library and the s2opc library reject connection attempts of unknown clients.

Moreover, we found that 3 demo servers show the same two types of insecure behaviors identified in the vendor section.

i) Trustlist disabled by default Untrusted connections are allowed (by default) in 2 libraries (node-opcua, open62541), which are not enforcing the use of the Trustlist to start the server. Again we found a major flaw in the certificate exchange.

In node-opcua there is a boolean variable 'automatically-AcceptUnknownCertificate' that is turned to 'true' by default when creating OPC UA server. This setting is transparent to users that are allowed to build a server without setting this variable explicitly. In the open62541 library, the user can start the server with or without the Trustlist. The Trustlist can be selected as an optional parameter to start the server from the command-line interface. When the server is started without the Trustlist, any incoming certificate is accepted, the program notifies the user is notified of this behavior.

ii) Improper use of Secure Channel primitives to perform certificate exchange The Rust opcua library uses the Trustlist for certificate validation in demo applications, but the suggested procedure on certificate exchange is vulnerable. After a connection attempt, the client certificate is stored in the 'rejected' list of certificates from where an operator should move it to the trusted folder. An attacker can create a certificate similar to the real client certificate that is difficult to tell apart.

Finally, for the remaining 4 demo applications, the missing support to client certificate authentication is caused by missing features in the library altogether. In particular, we found that there are 4 libraries where the implementation of the Trustlist is missing. In the library LibUA, a function is prepared for the user with a comment to implement certificate authentication. Users of the ASNeG and the Python-opcua library have pointed out the missing security features in issues on GitHub. For the ASNeG library, this feature is planned for the next release [47]. Developers of the Python-opcua library in 2017 acknowledged the issue [48], but it is not available yet.

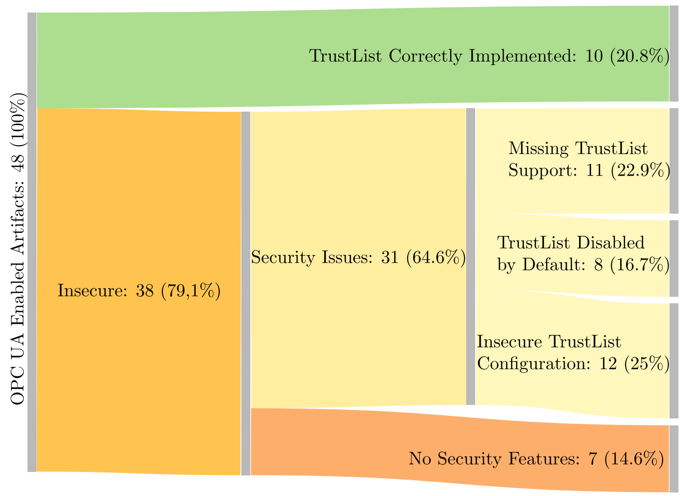


Fig. 7. This chart summarizes the findings of our work. It shows how the security properties for the 48 OPC UA enabled artifacts are distributed. For the artifacts that are insecure the chart details the reason of the insecure behavior. The majority of the OPC UA artifacts present security issues.

D. Vulnerability to Middleperson attack

In Section IV we explained that the Middleperson attack can be performed when there are servers vulnerable to Rogue Client attacks and clients vulnerable to Rogue Server present at the same time in the network. Combinations of aforementioned OPC UA servers and clients that are vulnerable respectively to Rogue Client and Rogue Server attacks would make a deployment that is vulnerable to Middleperson attacks. In particular, we have identified that 19 servers artifacts out of 29 (65%) that support security features are vulnerable to the Rogue Client attack (4 artifacts due to missing Trustlist features and 15 due to insecure instructions Trustlist) and 12 clients out of 12 (100%) that support security, are vulnerable to Rogue Server attack (7 artifacts due to missing Trustlist features and 5 due to insecure instructions to configure Trustlist). As we can see there is a non-negligible risk to deploy an insecure OPC UA network where an attacker can operate as Middleperson.

E. Summary of Findings

We considered a total of 48 OPC UA enabled artifacts: 22 products from vendors and 16 available libraries (of which 11 provide OPC UA servers, 15 provide OPC UA clients). Figure 7 reports a summary of our findings that we detail in the following.

Seven artifacts do not support security features at all (14.6%). Among the 41 remaining artifacts that support security features, we found that 31 artifacts (64.6% of the 48 OPC UA artifacts) show issues or errors in the Trustlist management that enable Rogue Client, Rogue Server, and Middleperson attacks. The other 10 artifacts (20.8% out 48) correctly implement the Trustlist management and instruct users about its configuration, and thus they are not vulnerable to the Rogue Client, Rogue Server, and Middleperson attacks. The 31 artifacts that show security issues with the configuration of the Trustlist can be classified into the following three categories:

- **Missing Support for Trustlist.** 11 artifacts do not implement Trustlist (or do not provide instructions about its configuration) management although they implement OPC UA security features, that is they offer functionality for signing and encryption but not for the validation of certificates. This makes their deployments always vulnerable to Rogue Client, Rogue Server, Middleperson attacks.
- **Trustlist disabled by default.** in 8 artifacts the Trustlist is enabled optionally.
- **Insecure Trustlist configuration.** in 12 artifacts the instructions guide the users to improper use of Secure Channel primitives to perform certificate exchange.

Adoption of features. in our analysis of OPC UA features, we found that while the client-server model is available in all tested products while the publisher-subscriber model is supported by 3 artifacts. Moreover, the features offered by the GDS to manage certificates are supported by 5 artifacts. Security features are widely adopted, but they require the correct distribution of certificates in order to deploy a secure network. As we found, this is not always the case in OPC UA products as many of them do not implement the Trustlist for certificates or do have issues in the configuration procedure.

VI. Countermeasures

Our work showed general missing support and incompleteness of OPC UA security features. This might be due to a general unawareness about the root importance of the Trustlist in the secure channel establishment for OPC UA (or not among motivating drivers during the development [48]). In our manuscript we have identified three common issues in the analyzed artifacts. In this section, we propose some countermeasures to prevent vulnerabilities in OPC UA deployments.

Missing support to Trustlist. This problem can be solved by implementing the Trustlist management as described in the OPC UA standard.

Trustlist disabled by default. This problem can be solved by enabling the Trustlist by default, and by making it mandatory to populate the Trustlist upon the creation of secure channels.

Certificate exchange through Secure Channel primitives. This problem can be solved with a twofold solution i) remove the instructions that lead to this insecure behavior from user manuals, ii) make it mandatory to populate the Trustlist upon the creation of secure channels.

The mentioned countermeasures, although simple, can mitigate the three common issues that we have encountered in our work and avoid vulnerable OPC UA deployments.

VII. Related Work

Security analysis of open source OPC UA libraries. In [49], the authors study the security of four popular open-source libraries (UA .Net Standard, open62541, node-opcua, Python-opcua) which we also inspect in this paper. The analysis focuses on five aspects: dependencies, timeouts, supported Security Policies, message processing, and randomness. The authors identify two vulnerabilities in the implementations: missing upper time limits in Python-opcua leaving the library vulnerable to DoS attacks and missing packet type checks in node-opcua. The problems that we discovered with our work, regarding the Trustlist configuration of the libraries, were not identified.

Neu et al. [50] and Polge et al. [4] have shown the feasibility of a DoS attack by a Rogue Client. They suggest detecting anomalies in the network traffic as a countermeasure. Polge et al. [4] additionally implement a Middleperson attack using Eclipse Milo. The attacked OPC UA applications are using Security Policy None or Aes128-Sha256-RsaOaep. They report that if username and password are sent as part of the ActivateSessionRequest message the password is encrypted even in Security Mode None and can therefore not be recovered (in contrast to the findings presented in this work). The encryption of the password in Mode None is not required according to the Specification [7]. The developers of the library have made the decision to encrypt the password by default even if Security Mode None is used [51]. Based on vulnerabilities we found, a Middleperson attack that recovers plain-text credentials is possible. More specifically, the client examples provided by the Milo library are vulnerable to the attack since arbitrary certificates are accepted. Therefore, a Middleperson attacker can provide their own certificate, and the user credentials are encrypted using the public key associated with the attacker's certificate.

Secure exchange of certificates. The security of the OPC UA protocol relies on certificates that authenticate each application. Prior publications proposed techniques to establish the initial root of trust in OPC UA. Meier et al. [52] propose to connect a new OPC UA application to a physical device with certificate manager functionality to install a certificate and the Trustlist before connecting the application to the network. In [53], a PKI is implemented. The PKI offers functionality to sign certificates or verify the trustworthiness of certificates using the Online Certificate Status Protocol.

VIII. Conclusions

In this work, we have systematically investigated practical challenges faced to use OPC UA securely. To this end, we have introduced a security assessment methodology—our assessment considers three attacks that can target OPC UA deployments, Rogue Server, Rogue Client, Middleperson attacks. Our research poses three research questions and three challenges. With our approach we systematically address the research questions **R1, R2, R3**. To recall, the questions were **R1.** What are practical challenges for the correct use of OPC UA security features? **R2.** Are OPC UA security features correctly implemented by the vendors and products? **R3.** What are the consequences of breaking OPC UA security features?

To address **R1**, we conducted the first systematical survey of 48 OPC UA artifacts provided by vendors or available online. Our survey investigates the availability of OPC UA security, publish-subscribe, and Global Discovery Server functionalities. We showed that publish-subscribe and Global Discovery Server are not widely adopted. Furthermore, we showed that 7 OPC UA do not support for security features of the protocol at all. In addition, we found that 31 artifacts that partially feature OPC UA security rely on incomplete libraries and come with misleading instructions.

To address **R2**, We proposed a framework to investigate the identified security issues. With our framework, we show that the identified issues make OPC UA artifacts are vulnerable to Middleperson, Rogue Client, and Rogue Server attacks. Specifically, We have analyzed 48 OPC UA artifacts, 41 of those artifacts support security features. We found that 31 out 41 artifacts present pitfalls in the Trustlist management required to establish the initial root of trust. We have identified three recurring pitfalls in the Trustlist management that enable vulnerabilities in OPC UA deployments. Those recurring pitfalls are the missing support to Trustlist, Trustlist disabled by default, and certificate exchange through secure channel primitives. For each of the identified recurring pitfalls, we provide possible mitigation.

To address **R3**, we designed, implemented, and demonstrated three types of attacks (Middleperson, Rogue Client, and Rogue Server). The attacks allow the attacker to steal user credentials exchanged between victims, eavesdrop on process information, manipulate the physical process through sensor values and actuator commands, and prevent the detection of anomalies in the physical process.

Our findings demonstrate a major flaw on OPC UA artifacts that threaten the OPC UA security guarantees. We believe that our proposed systematic approach is useful to increase the awareness of the risks produced by the three common pitfalls for the key establishment that we have identified and help users, developers and companies to understand the threats that can be induced by those erroneous configurations of the Trustlist. Finally, our POC implementation can be used as a tool to test for erroneous configurations and improve security in OPC UA deployments.

References

[1] O. Foundation, "Opc unified architecture specification v1.04," https://opcfoundation.org/developer-tools/specifications-unified-architecture, 2018.

[2] BSI, "Open platform communications unified architecture security analysis," https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Studies/OPCUA/OPCUA.html, Bundesamt für Sicherheit in der Informationstechnik, Tech. Rep., 2017.

[3] M. Dahlmanns, J. Lohmöller, I. B. Fink, J. Pennekamp, K. Wehrle, and M. Henze, "Easing the conscience with opc ua: An internet-wide study on insecure deployments," in *Proceedings of the ACM Internet Measurement Conference*, 2020, pp. 101–110.

[4] J. Polge, J. Robert, and Y. Le Traon, "Assessing the impact of attacks on opc-ua applications in the industry 4.0 era," in *2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2019, pp. 1–6.

[5] O. Foundation, "Opc unified architecture specification part 2: Security model v1.04," 2018.

[6] ——, "Opc unified architecture specification part 6: Mappings v1.04," 2017.

[7] ——, "Opc unified architecture specification part 4: Services v1.04," 2017.

[8] F. O. ua, "Opc ua implementation," https://github.com/FreeOpcUa/python-opcua, accessed December 10, 2020.

[9] B. I. A. GmbH, "Adi opc ua server user's manual v1.27," https://www.br-automation.com/es-mx/descargar/industrial-pcs-and-panels/automation-pc-3100/kabylake-system-unit/windows-10-iot-enterprise-2016-ltsb/adi-opc-ua-server-users-manual/, accessed November 3, 2020.

[10] Bachmann, "OPC UA client and server," http://bachmann.bbdevserver.de/fileadmin/media/Produkte/Vernetzung/Produktblaetter/Software_OPC-UA-server_en.pdf, accessed 14 January, 2021.

[11] Beckhoff, "Tc3 opc ua," https://download.beckhoff.com/download/document/automation/twincat3/TF6100_TC3_OPC-UA_EN.pdf, accessed 14 January, 2021.

[12] B. E. A. AB, "Opc ua server - ix developer 2.4 ki00312a," https://www.beijerelectronics.tw/en/Support/file-archive-tree-page?docId=58657, accessed November 3, 2020.

[13] B. R. AG, "Opc ua server application manual," https://www.boschrexroth.com/various/utilities/mediadirectory/download/index.jsp?object_nr=R911403778, accessed November 3, 2020.

[14] G. E. Company, "Certificate management and the opc ua server," https://www.ge.com/digital/documentation/ifix/version61/Subsystems/OPCUASVR/content/opcua_about_the_trust_list.htm, accessed November 3, 2020.

[15] H. I. Sàrl, "Control edge builder protocol configuration reference guide," https://www.honeywellprocess.com/library/support/Public/Documents/ControlEdge-Builder-Protocol-Configuration-Reference-Guide-RTDOC-X288-en-161A.pdf, accessed November 3, 2020.

[16] Lenze, "Lenze opc ua communicationv1.1," https://download.lenze.com/AKB/English/201400321/Commissioning_Lenze_OPC_UA_V1_1.pdf, accessed November 3, 2020.

[17] M. Electric, "Melsec iq-r opc ua server moduleuser's manual (startup)," https://dl.mitsubishielectric.com/dl/fa/document/manual/plc/sh081693eng/sh081693engf.pdf, accessed November 3, 2020.

[18] N. I. CORP., "Insightcm 3.7 manual," https://www.ni.com/documentation/en/insightcm/latest/serverconf/secure-opc-server/, accessed November 3, 2020.

[19] Omron, "Nj-series cpu unit opc ua user's manual (w588)," https://assets.omron.eu/downloads/manual/en/v2/w588_nj-series_cpu_unit_opc_ua_users_manual_en.pdf, accessed November 3, 2020.

[20] P. E. W. E. AG, "Hmwin user manual," https://mediap.industry.panasonic.eu/assets/download-files/import/mn_hmwin_user_peweu_en.pdf, accessed November 3, 2020.

[21] R. Automation, "Factorytalk linx gateway getting results guide," https://literature.rockwellautomation.com/idc/groups/literature/documents/gr/ftlg-gr001_-en-e.pdf, accessed November 3, 2020.

[22] S. Electric, "M580bmenua0100 opc ua embedded moduleinstallation and configuration guide," https://download.schneider-electric.com/files?p_enDocType=User+guide&p_File_Name=PHA83350.00.pdf&p_Doc_Ref=PHA83350, accessed November 3, 2020.

[23] S. AG, "Opc ua .net client for the simatic s7-1500 opc ua server," https://cache.industry.siemens.com/dl/files/901/109737901/att_955026/v3/109737901_OPC_UA_Client_S7-1500_DOKU_V13_en.pdf, accessed December 10, 2020.

[24] W. I. G. . C. KG, "u-create studio system manual," https://mdcop.weidmueller.com/mediadelivery/asset/900_91150, accessed November 3, 2020.

[25] Yokogawa, "Opc-ua server (/e3)user's manual," http://web-material3.yokogawa.com/IM04L51B01-20EN_010.pdf, accessed November 3, 2020.

[26] Codesys, "Opc ua server," https://help.codesys.com/api-content/2/codesys/3.5.14.0/en/_cds_runtime_opc_ua_server/, accessed December 10, 2020.
[27] ABB, "Ac500 user management with v3configuration and handling," https://library.e.abb.com/public/2745203d43f64bf2af9add792d5b4e46/AC500%20-%20Application%20Note%203ADR010315.pdf, accessed November 3, 2020.
[28] Eaton, "Xc300 modular control," https://es-assets.eaton.com/DOCUMENTATION/AWB_MANUALS/MN050005_EN.pdf, accessed December 10, 2020.
[29] Hitachi, "Hx series application manual (software)," https://www.hitachi-da.com/files/pdfs/produkte/SPS/HX/HX-CPU_Software(Tentative).pdf, accessed November 3, 2020.
[30] W. K. G. . C. KG, "Wago-i/o-system 750, 750-8xxx opc ua server," https://www.wago.com/global/d/3597419, accessed November 3, 2020.
[31] CODESYS, "Codesys opc ua pubsub sl," https://store.codesys.com/op-ua-pubsub-sl.html?___store=en&___from_store=default, accessed January 14, 2021.
[32] ASNeG, "Opc ua implementation," https://github.com/ASNeG/OpcUaStack, accessed December 10, 2020.
[33] E. Milo, "Opc ua implementation," https://github.com/eclipse/milo, accessed December 10, 2020.
[34] FreeOpcUa, "Opc ua implementation," http://freeopcua.github.io/, accessed December 10, 2020.
[35] LibUA, "Opc ua implmentation," https://github.com/nauful/LibUA, accessed December 10, 2020.
[36] Node-opcua, "Opc ua implementation," http://node-opcua.github.io/, accessed December 10, 2020.
[37] C. S. LLC, "Opc ua implementaion," https://github.com/convertersystems/opc-ua-client, accessed December 10, 2020.
[38] R. opc UA, "Opc ua implementation," https://github.com/locka99/opcua, accessed December 10, 2020.
[39] G. opcua, "Opc ua implementation," https://github.com/gopcua/opcua, accessed December 10, 2020.
[40] H. B. M. GmbH, "Opc ua implementation," https://github.com/HBM/opcua, accessed December 10, 2020.
[41] opcua4j, "Opc ua implementation," https://code.google.com/p/opcua4j/, accessed December 10, 2020.
[42] Open62541, "Opc ua implementation," http://open62541.org/, accessed December 10, 2020.
[43] O. U. Interface, "Opc ua implementation," http://oscada.org/main/documentation/, accessed December 10, 2020.
[44] S2OPC, "Opc ua implementation," https://gitlab.com/systerel/S2OPC, accessed December 10, 2020.
[45] OPCfoundation, "Opc ua implemntation," https://github.com/OPCFoundation/UA-.NETStandard, accessed December 10, 2020.
[46] U. Automation, "Step-by-step connect example," http://documentation.unified-automation.com/uaexpert/1.4.2/html/first_steps.html#connect_step-by-step, accessed January 14, 2021.
[47] ASNeG, "Client certificates are not validated #44," https://github.com/ASNeG/OpcUaStack/issues/44, accessed January 14, 2021.
[48] P. O. UA, "Client certificates are not validated #392," https://github.com/FreeOpcUa/python-opcua/issues/392, accessed January 14, 2021.
[49] N. Mühlbauer, E. Kirdan, M.-O. Pahl, and G. Carle, "Open-source opc ua security and scalability," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1. IEEE, 2020, pp. 262–269.
[50] C. V. Neu, I. Schiering, and A. Zorzo, "Simulating and detecting attacks of untrusted clients in opc ua networks," in *Proceedings of the Third Central European Cybersecurity Conference*, 2019, pp. 1–6.
[51] E. Milo, "Regression with username/password authentication #244," https://github.com/eclipse/milo/issues/244, accessed January 23, 2021.
[52] D. Meier, F. Patzer, M. Drexler, and J. Beyerer, "Portable trust anchor for opc ua using auto-configuration," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1. IEEE, 2020, pp. 270–277.
[53] G. Karthikeyan and S. Heiss, "Pki and user access rights management for opc ua based applications," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1. IEEE, 2018, pp. 251–257.