Security Analysis of Vendor Implementations of the OPC UA Protocol for Industrial Control Systems

Alessandro Erba
CISPA Helmholtz Center for Information Security
Saarbrücken Graduate School of Computer Science, Saarland University
Saarbrücken, Germany
alessandro.erba@cispa.de

Anne Müller
CISPA Helmholtz Center for Information Security
Saarbrücken Graduate School of Computer Science, Saarland University
Saarbrücken, Germany
anne.mueller@cispa.de

Nils Ole Tippenhauer
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
tippenhauer@cispa.de

ABSTRACT

The OPC UA protocol is an upcoming de-facto standard for building Industry 4.0 processes in Europe, and one of the few industrial protocols that promises security features to prevent attackers from manipulating and damaging critical infrastructures. Despite the importance of the protocol, challenges in the adoption of OPC UA's security features by product vendors, libraries implementing the standard, and end-users were not investigated so far.

In this work, we systematically investigate 48 publicly available artifacts consisting of products and libraries for OPC UA and show that 38 out of the 48 artifacts have one (or more) security issues. 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. To verify our analysis, 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.

CCS CONCEPTS

• **Security and privacy → Key management**; **Usability in security and privacy**; **Authentication**.

KEYWORDS

OPC UA, Security Analysis, Key management, Attacks

ACM Reference Format:
Alessandro Erba, Anne Müller, and Nils Ole Tippenhauer. 2022. Security Analysis of Vendor Implementations of the OPC UA Protocol for Industrial Control Systems. In *Proceedings of the 4th Workshop on CPS & IoT Security and Privacy (CPSIoTSec '22), November 7, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3560826.3563380

CPSIoTSec '22, November 7, 2022, Los Angeles, CA, USA

ACM ISBN 978-1-4503-9876-3/22/11…$15.00
https://doi.org/10.1145/3560826.3563380

1 INTRODUCTION

The increasing interconnection of industrial components critically relies on the security of the communication protocol adopted for Machine to Machine (M2M) communication to prevent adversarial manipulation of the process, leading to significant monetary loss and physical damage [59]. The OPC Unified Architecture (OPC UA) protocol [20] is often considered the de-facto standard for building Industry 4.0 processes and it is the reference communication protocol in RAMI4.0 (the reference model for Industry 4.0 [66] in European countries such as France, Italy and Germany [23]). In 2006, the OPC Foundation released the first OPC UA specification featuring security mechanisms such as authentication, authorization, integrity, and confidentiality. The German Federal Office for Information Security (BSI) released the 'OPC UA Security Analysis' [9], reporting that 'No systematic errors could be detected' in the OPC UA standard.

Nevertheless, there seem to be significant challenges to set up secure OPC UA deployments in practice. In [14], the authors perform a scan for OPC UA servers reachable over the Internet and find that 92% of the 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 [9]) to disable insecure security modes and policies and enforce user authentication. The authors of [14] suggest that their findings are due to the configuration complexity of OPC UA, but no root cause analysis was provided. Indeed, OPC UA requires a correct configuration to prevent attacks such as: i) attackers feeding incorrect information to clients (Rogue Server attack); ii) eavesdrop and change values which can directly alter the physical process (Rogue Client attack); iii) or both (Middleperson attack).

Security threats through configuration complexity are in general investigated in the field of usable security [2, 26, 35]. For example, studies demonstrated that a little set of demo applications or poor documentation resulted in insecure deployments [35]. Challenges in setting up OPC UA networks were not investigated so far.

In this work, we are the first to systematically investigate the security of a range of OPC UA libraries and products. Despite the popularity of OPC UA and its advanced security concept (compared to other industrial protocols), we show that in practice many OPC UA artifacts have missing support for security features of the protocol. Those limitations make the security configuration infeasible, or give a false sense of security (i.e., traffic encryption occurs with untrusted parties). Several of the artifacts that claim to feature OPC

UA security rely on libraries that are incomplete, insecure and provide instructions that lead to insecure settings. While such issues are implementation-specific, we show that the OPC UA standard is insufficient, as it supports those insecure behaviors. Lastly, we propose several practical countermeasures.

To assess the vulnerability of libraries, we create a framework that implements Rogue Server, Rogue Client, and Middleperson attacks. 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. We show that Middleperson attacks are possible and 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 contrasts with prior work [55], 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 22 products and identify significant recurring (for 15 out of 22) issues with the availability OPC UA security features, or their setup instructions.
- We systematically analyze 16 libraries and for all of them we identify at least one reoccurring issue with the implementation of security features (either at client side, at server side or both).
- We demonstrate the feasibility of the identified attacks by implementing the attacks with custom proof-of-concept applications[1].

Disclosure. We discussed our findings with the OPC foundation, leading to improvements of the specifications. Our work was based on publicly available information extracted from user manuals, library documentation and the OPC UA standard. As the feasibility of our attacks depends on (past) end-user configuration process carried out in the target system and not on a specific software/hardware vulnerability, there was no disclosure to vendors required. Nevertheless, our findings already led to improvements of at least one artifact after public discussion of this work.

Organization. The remainder of this work is organized as follows. In Section 2 we present the details of the OPC UA protocol. In Section 3 we present our research methodology together with system and attacker model. In Section 4 we explain how we build the framework that we use to analyze the libraries. In Section 5 we report the results of our research. A discussion of countermeasures is presented in Section 6. Related work is presented in Section 7 followed by conclusions in Section 8.

2 BACKGROUND ON OPC UA

OPC UA [20] is an industrial communication protocol developed by the OPC foundation that allows platform-independent and secure communication by design. Those two characteristics make this protocol stand out when compared to other common industrial protocols. Adopting this protocol in an industrial plant allows integration between heterogeneous hardware, which is instead a constraint brought by proprietary protocols (e.g., Siemens S7). At the same time, the modern *security-by-design* features offered by the protocol promise to mitigate some of the common security threats for CPS (e.g., no message authentication or encryption). The last OPC UA specification 1.04 was released in 2018 and is divided into eighteen parts. Part 2 describes the security model of OPC UA, and Part 4 describes the services that implement security primitives. Two communication strategies are possible: client-server and publisher-subscriber. Both allow messages to be signed to ensure authenticity and encrypted to add confidentiality. OPC UA does not enforce the use of security, messages can be exchanged without security. Figure 1(a)(b) reports an example of secure connection establishment in client-server. A server offers several endpoints, each defined by a Security Mode, a Security Policy, and the supported User Identity Token(s). When initiating a connection, the client chooses the endpoint (from an unauthenticated list of endpoints). The Security Mode defines how messages are exchanged to achieve authentication, confidentiality, and integrity. Available Security Modes are None, Sign, SignAndEncrypt. Security Policies define the cryptographic primitives used for the specific security mode. The UserIdentityToken defines the supported user authentication methods for an endpoint: Anonymous (no user authentication), Username&Password, Certificate [49].

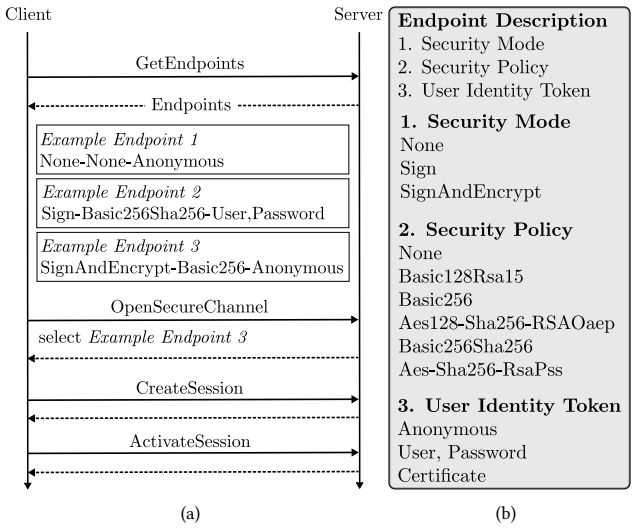

Figure 1: (a) Overview of the connection establishment procedure. The server provides a list of endpoints. The client selects which endpoint to connect to. If a secure endpoint is chosen, the application authentication through certificates is performed. (b) Options available for the endpoint configuration. The Security Mode, Security Policy and User Identity Token are chosen independently from each other.

Certificate Management. Secure connection establishment (Security Mode not 'None') requires OPC UA applications (clients and

[1]The framework is available at https://github.com/scy-phy/OPC-UA-attacks-POC

servers) to exchange their Application Instance Certificate. Upon receiving a certificate, an application needs to decide whether the received certificate is trustworthy. To this end, each OPC UA application has a list of certificates that are trusted (Certificate Trustlist). This list contains 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. GDS provides two main functionalities: i) Application discovery: OPC UA applications use the GDS to find available applications that have previously registered with the GDS. ii) Certificate management: applications push and pull certificate to the GDS to update Trustlist.

Secure Connection Establishment. To establish a secure connection, the server sends its certificate through the `GetEndpoints` response. If the client trusts the server and selects a secure endpoint, the `OpenSecureChannel` message carries the client certificate. Upon trusting the client's certificate on server side, the secure channel is established. The `OpenSecureChannel` request and response use asymmetric encryption and messages contain nonces used to compute the symmetric signing and encryption keys used in sessions [48]. Due to space constraints further details can be found in Appendix A.

3 OPC UA SECURITY ASSESSMENT METHODOLOGY

OPC UA features a number of cryptographic options to establish secure channels (see Section 2). 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 parties. On the Internet, the initial distribution of public keys is commonly solved by shipping devices (or OS) with certificates of a set of core root certificate authorities (CAs). Servers (identified by unique DNS names) then provide certificates authenticated directly (or indirectly) by those root CAs. Such a solution is not possible for ICS networks, as they are air-gapped and do not provide (externally verifiable) unique addressing. Local self-signed CAs are an alternative, but their certificates will not be shipped together with devices newly introduced into the system. For this reason, bootstrapping the security in an OPC UA system relies on the manual certificates distribution. In OPC UA there are two ways to perform certificate management and distribution: i) Through self-signed certificates, ii) through the GDS *CertificateManager* (see Section 2). The focus of our security assessment is the application authentication functionality required for the security in OPC UA.

3.1 System and Attacker Model

We assume a local ICS network with OPC UA server as depicted in Figure 2. 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 2) as suggested by the BSI [9]. 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

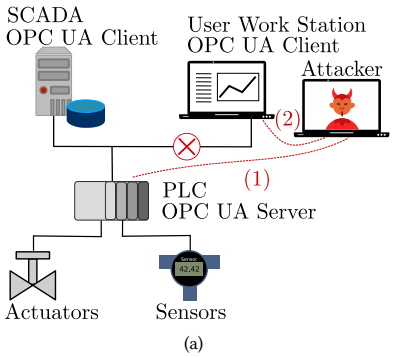

Figure 2: Example OPC UA network with a server and two clients. An attacker that wants to establish themselves as a PitM has to (1) pose as an OPC UA client towards the legitimate OPC UA server and (2) as an OPC UA server towards a legitimate client.

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(this is a common assumption in related usable security works, e.g., [2]). In this work, we consider three attackers with different goals corresponding to the OPC UA model introduced in the Part 2 of the standard [49].

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 OPC UA user credentials. They[2] create 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 and clients. They create 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 between the client and server, intercepting and manipulating all communications between both. This requires achieving Rogue Client and Server objectives.

3.2 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.

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

3.3 Proposed Approach

Our approach focuses on the analysis of security features implemented by OPC UA compatible artifacts. We focus on implemented key management functionalities i.e., i) Proprietary hardware and software that implements OPC UA stack, ii) Open-source libraries that implement the OPC UA stack.

To address R1, we survey proprietary and open source OPC UA enabled products. We investigate practical challenges for correctly configuring secure OPC UA setups in the analyzed products checking availability of features. Investigating challenges to set up security is common practice in usable security, where resources available to developers are analyzed to understand if they help configuring secure systems [2, 26, 34, 35].

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 as we had no access to physical products. For libraries, we practically tested the security features deploying OPC UA clients and server locally. We tested the three attacks presented in the Attacker Model, which are feasible due to erroneous or incomplete key management features. For each library, we set up OPC UA server and client following the instructions and demo applications provided with libraries. Testing the application is useful because, a) not all functionality provided by products was sufficiently described in documentation to be able to classify its features for our paper; b) documentation and code does not always match, so we double-checked with our implementation and demos. We note that, demo applications, user manuals and documentations are resources used by developers to build their applications, so it is realistic to follow them to identify poor security implementations and propagation of security pitfalls [2, 35]. Recently, it was reported that it was possible to alter the infotainment system firmware on a 2021 Hyundai Ioniq car using publicly known AES 128-bit CBC private keys provided as an example in NIST documents [61].

To address R3, we show that breaking the application authentication 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).

4 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 of our framework implementation.

4.1 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 checks for

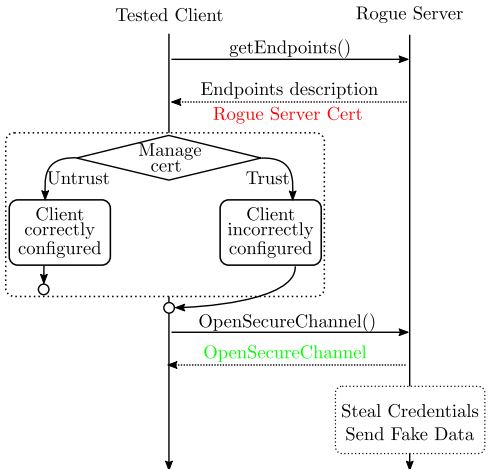

Figure 3: Rogue Server workflow. The server waits for a new connection. When the Server receives a getEndpoints request it provides a certificate that was not trusted on the client side upon connection (in red). If the client trusts the server certificate and sends an OpenSecureChannel to the server, the Rogue Server allows the connection (in green). The attacker can, for example, steal credentials and send manipulated data.

vulnerabilities generated by incorrect (or incomplete) Trustlist management. When both Rogue Client and Rogue Server vulnerabilities are present in the OPC UA network, Middleperson attacks can be exploited.

Rogue Server. In Figure 3 we report the steps followed 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 application instance certificate that is not present in the client's Trustlist. The victim client receives the endpoints and the certificate. The Trustlist of the OPC UA victim client does not contain the Rogue Server's certificate. 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, an attacker can send fake data to the victim client and steal user credentials. User credentials stealing occurs when the `ActivateSession` request with 'UserIdentityToken' 'user&password' is instantiated by the Client. The request (containing user and password) is encrypted with the keys derived from the `OpenSecureChannel` Nonces. 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.

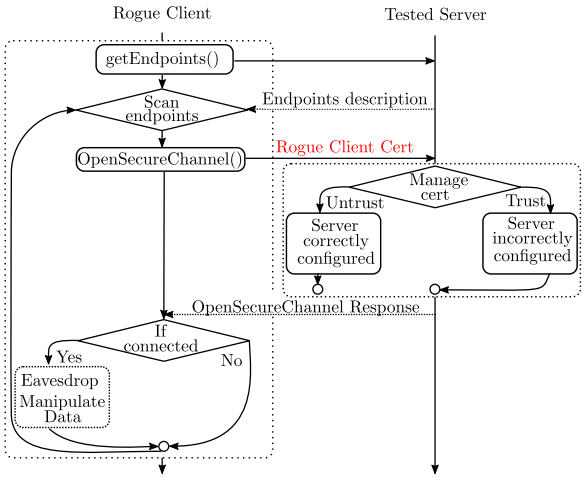

Figure 4: Rogue Client workflow. The dashed boxes contain the flowchart representing the application server/client logic. Diamonds represent a decision by the client/server according to the data flow and management. The client lists all the secure endpoints on the server and for each one, it tries to 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.

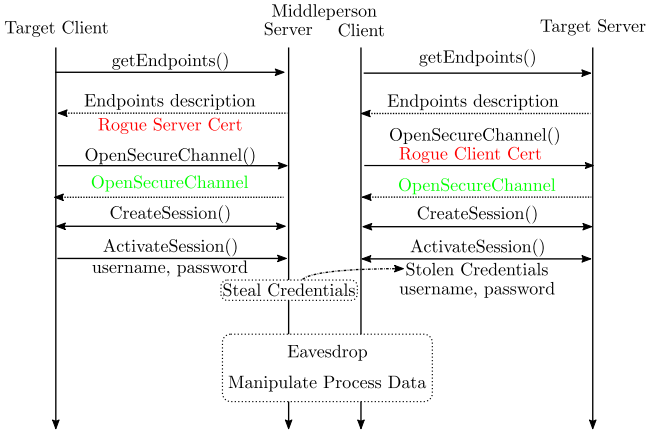

Figure 5: Example of Middleperson attack with stealing of session credentials (User Identity Token: User&Password). The attacker acts both as Rogue Client and Server. The Target Client connects to the Middleperson Server. The client is vulnerable to Rogue Server attack and trusts the server cert without verification and instantiates a OpenSecureChannel() request. The Rogue Server allows the connection. When the client instantiates a session the Rogue Server accepts it and decrypts the provided credentials. The Middleperson connects to the Target Server (vulnerable to Rogue Client Attacks) and creates a session providing the stolen credentials

Rogue Client. In Figure 4 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, it means that 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 5 reports the steps followed by our framework to verify the vulnerability to the Middleperson attack. The test leverages the concepts of Rogue Client and Rogue 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 request the user identity token from 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 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.

4.2 Implementation

We implemented our framework (i.e., Rogue Server, Rogue Client, and Middleperson) using the python opc-ua [21] 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 of four python modules, `framework`, `utils`, `rogueclient`, and `rogueserver` for a total of 571 lines of code. For each of the three attacks, we report a description of how we realized functionalities through the python modules.

Framework module. The `framework` module interacts with the attacker. It offers a command-line interface to select the attack (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 (except fingerprints). 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 5. 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.

5 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 libraries. We start by reporting the selection criteria of the artifacts considered in this work. Next, we investigate 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.

5.1 Artifacts Selection Criteria

For the two categories of OPC UA artifacts considered in our work, we used the following selection criteria. Note that not all the artifacts are certified by the OPC Foundation.

Proprietary products. We assessed only proprietary products with publicly accessible manuals that provide details on the availability of OPC UA security features, as we had no access to hardware products.

Libraries. We assessed libraries based on the availability as open-source or free (i.e., unlimited unpaid access) versions. For open-source libraries, we consulted GitHub and selected the libraries based on their popularity (e.g., number of stars), and we consulted the list at this repository[3].

5.2 Adoption of OPC UA features

For the artifacts considered we investigate which features of OPC UA are implemented.

Proprietary products. In Table 1 we summarize the adoption of OPC UA features of the 22 proprietary OPC UA products that are offered by vendors to configure their industrial devices. The table is organized into four parts: Certification by OPC Foundation, support of publish-subscribe, compatibility with Global Discovery Server, 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 products considered 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 OPC UA PubSub SL [12] extension that supports Publish-Subscribe, currently 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. Two vendors (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 seven vendors (Bachmann, 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

[3] open62541: List of Open Source OPC UA Implementations (version pushed on March 10, 2020)

Table 1: OPC UA in proprietary products. ●/○ denotes if the product supports/not supports a feature. ◐ denotes that there are problems with feature configuration.

Vendor	Platform	OPC Cert.	Pub-Sub	GDS	Security	Trustlist	Recommended Policy
B&R	ADI OPC UA [8]	●	○	○	●	●	Not specified
Bachmann	OPC UA Client/Serv. [4]	○	○	○	◐	◐	Not specified
Beckoff	TC3 OPC UA [5]	○	○	○	◐	◐	Deprecated protocols
Beijer	iX Developer [6]	○	○	○	○	○	None
Bosch Rexroth	ctrlX CORE [7]	○	○	○	●	◐	None not supported
General Electric	iFIX [24]	○	○	●	●	◐	Basic256Sha256
Honeywell	ControlEdge Builder [28]	○✢	○	○	○	○	None
Lenze	Easy Starter [36]	○	○	○	◐	◐	Deprecated protocols
Mitsubishi	MX Configurator-R [40]	●	○	○	●	●	None
National Instr.	InsightCM [42]	○	○	○	●	●	None
Omron	SYSMAC-SE2 [45]	●	○	○	●	●	Not specified
Panasonic	HMWIN Studio [54]	○	○	●	●	◐	Not specified
Rockwell	Factory talk linx [56]	○	○	○	●	●	Not specified
Schneider	Control Expert [18]	●	○	○	●	●	Basic256Sha256
Siemens	STEP 7 [60]	●	○	●	●	◐	Not specified
Weidmüller	u-create studio [64]	○	○	○	●	●	Basic256Sha256
Yokogawa	SMARTDAC+ [65]	○	○	○	○	○	None
Codesys based platforms							
Codesys	Codesys V3.5 [11]	○	●	○	●	◐	Not specified
ABB	Automation Builder [1]	○	○	○	●	◐	Basic256Sha256
Eaton	XSOFT-CODESYS [16]	○	○	○	●	◐	Not specified
Hitachi	HX Codesys [27]	○	○	○	●	◐	Not specified
Wago	e!cockpit [63]	●	○	●	●	◐	Not specified

✢State of the documentation consulted during the investigation. After a preprint release of this manuscript, the documentation related to the product was updated. Now it supports security and it is certified.

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.

Libraries. In Table 2 we report the results of our research about the adoption of OPC UA features that we conducted over 16 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

Table 2: OPC UA Libraries. ●/○ denotes if the product supports/not supports a feature. ◐ denotes that there are problems with feature configuration. Security column reports if the library implements security features. Trustlist column reports if the library implements application authentication. Demo column reports if demo application supports secure connection

		OPC	Pub		Server			Client		
Name	Lang.	Cert.	Sub	GDS	Security	Trustlist	Demo App.	Security	Trustlist	Demo App.
ASNeG [3]	C++	○	○*	○	●	○	-	○*	-	-
Eclipse Milo [17]	Java	○	○	○	●	●	●	●	◐	○
Free OpcUA [22]	C++	○	○	○	-	-	-	○	-	-
LibUA [37]	C#	○	○	○	●	○	○	●	○	○
node-opcua [44]	.js	○	○*	○	●	●	◐	●	○	○
opc-ua-client [13]	C#	○	○	-	-	-	-	●	●	◐
opcua [57]	Rust	○	○	○	●	●	◐	●	●	◐
opcua [25]	Golang	○	○	○	-	-	-	●	-	-
opcua [29]	TypeScript	○	○	-	-	-	-	○	-	-
opcua4j [51]	Java	○	○	○	○	-	-	-	-	-
open62541 [52]	C	◐+	●	○	●	●	◐	●	●	◐
OpenScada UA [53]	C++	○	○	○	●	○	○	●	○	○
Python-opcua [21]	Python	○	○	○	●	○	○	●	○	○
S2OPC [58]	C	◐+	●	○	●	●	●	●	◐	◐
UA.NET [50]	C#	●	○	●	●	●	●	●	●	◐
UAexpert [62]	C++	●	○	○	-	-	-	●	●	◐

+Server certified, client not certified. *Denotes that the feature is going to be introduced in the next release

they are communicating with. Regarding the remaining 2 implementations (Golang opcua, and S2OPC), we ran 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.

5.3 Vulnerability to Rogue Server

We tested the vulnerability of 15 Client implementations from 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 an endpoint with security 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 implementation. In 4 libraries (LibUA, node-opcua, OpenScada 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, nor in issues in the repository, nor in 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 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 Secure Channel 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 a connection to server, but this is not the default workflow 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.

5.4 Vulnerability to Rogue Client

Proprietary products. 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 1. The vendors Yokogawa [65], Honeywell [28] and Beijer [6] 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) 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 unauthenticated `OpenSecureChannel` request to move the client certificate from client to server (instead of building the Trustlist before any connection in the network). This behavior 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 behavior 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 insecure behavior present in the Codesys documentation. This problem may potentially threaten more than 400 device manufacturers that rely on this Codesys software.

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 4. To perform our test, we started with 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. Five demo applications offer secure and insecure endpoints, and five 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 Eclipse Milo demo server reachable online 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 do not enforce the use of the Trustlist to start the server. Again, we found a major flaw in the certificate management. In node-opcua there is a Boolean variable 'automaticallyAcceptUnknownCertificate' 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) Use of Secure Channel primitives to perform certificate exchange. The Rust opcua library uses the Trustlist for certificate validation in demo applications, in our opinion the suggested procedure on certificate exchange is insecure. 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 like 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 [31]. Developers of the Python-opcua

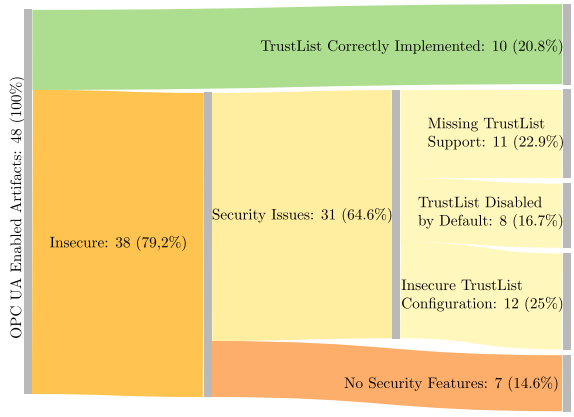

Figure 6: Summary of findings. It shows how the security properties for the 48 OPC UA enabled artifacts are distributed. For the artifacts that are insecure the chart details the reason of the insecure behavior. The majority of the OPC UA artifacts present security issues.

library in 2017 acknowledged the issue [30], but it is not available yet.

5.5 Vulnerability to Middleperson attack

In Section 4 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.

5.6 Summary of Findings

We considered a total of 48 OPC UA enabled artifacts: 22 products from vendors and 16 libraries (of which 11 provide OPC UA servers, 15 provide OPC UA clients). Figure 6 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 of 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 disabled by default. This behavior puts OPC UA deployments at risk as applications will accept any incoming certificate, making them vulnerable to the three considered attacks.
- **Certificate exchange through Secure Channel primitives.** In 12 artifacts the instructions guide the users to use unauthenticated Secure Channel primitives to perform certificate exchange. The user is guided to initiate a connection such that the connecting applications send their certificates to each other. Then the certificates are manually trusted for each device. Since the certificates are sent via an insecure channel an adversary can leverage this behavior to mount an attack. The OPC UA standard allows this behavior [46] assuming that only trained personnel are authorized to trust incoming certificates exchanged through insecure channels. In our opinion, this recommendation is flawed as insecure channels allow simple manipulation of the certificates before acceptance by the administrator (via Middleperson attacks), and humans have been shown to be bad at detecting manipulated certificates [10, 15].

Adoption of features. in our analysis of OPC UA features, we found that 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 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.

Our investigation is the first about security of OPC UA artifacts. Our findings and considerations are consistent with prior work in usable security of cryptographic APIs [2, 26], which showed that lack of comprehensive documentation and demo applications will result in poor security configurations implemented in the final deployments.

6 COUNTERMEASURES

Our work showed general missing support and incompleteness of OPC UA security features in artifacts that make the certificate management often not possible or not required by default. Moreover, we discovered some insecure behaviors allowed by the OPC UA standard. In this section, we discuss countermeasures to prevent vulnerabilities in OPC UA deployments and achieve an initial secure key distribution.

6.1 Certificate exchange via insecure channels

As explained in Section 3, initial key distribution for air-gapped ICS is a non-trivial problem as devices cannot be shipped with root CAs installed. OPC UA Global Discovery Servers with Certificate Manager would be an alternative to manual certificate exchange, but as we showed in Section 5 this feature is not widely implemented and, in any case, the GDS certificates will not be shipped together with devices newly introduced into the system. That implies that bootstrapping the security in an OPC UA system critically relies on manual pre-distribution of certificates even when GDS is deployed.

Currently the OPC UA standard does not provide effective solutions to overcome this challenge, instead (as explained in Section 5) it guides the users to exchange certificates through insecure channels. Given the technological challenges posed by ICS networks, we suggest to i) update the standard (and manuals) to instruct users to rely on out-of-band secure channels for initial certificate distribution, ii) enforce Trustlist population before use of OPC UA secure channels and not through unsecured primitives as found commonly in the consulted user manuals and the standard itself.

Out-of-band certificate exchange can be achieved with different solutions e.g., through secure protocols such as SSH, or through physical solutions such as USB sticks and QR codes [38]. Each of those viable solutions has different trade-offs in terms of usability and security but in any case, offer better guarantees than exchanging certificates through unsecured primitives. Of course, exchanging OPC UA certificates through SSH requires that the communicating devices have already shared SSH secrets. USB sticks are a widely supported solution to distribute certificates in the industrial environment, as a drawback the users are required to physically move around the plant to distribute the certificates to the devices that are expected to communicate with each other. Finally, QR codes represent another potentially easy to use and deploy solution to distribute certificates in the industrial environment [38]. In particular, industrial devices could be shipped with a private public key pair, the public key can be printed on a QR code sticker and physically applied to the device. During the installation process the operator scans the QR code and installs the certificate in the Trustlist of the authorized devices in the industrial environment. As a drawback of this method, an attacker could deploy a malicious device in the industrial plant while performing a supply chain attack and replace the QR code sticker with his own certificate on legitimate hardware.

6.2 Missing support for Trustlist

End users should only use products and libraries that implement the Trustlist management as described in the OPC UA standard. In particular, products certified by the OPC can be expected to support this feature. Artifacts that do not provide this feature should instead implement it in order to allow communicating parties to trust each other upon connection.

6.3 Trustlist disabled by default

End users should enable the Trustlist if disabled upon OPC UA network configuration. Vendors recommend enabling the Trustlist by default, and thus making it mandatory to populate the Trustlist upon the creation of secure channels. Enabling the Trustlist functionality will require the user to perform additional steps to set up a new device on the OPC UA network but those steps are of crucial importance for the security of the ICS.

7 RELATED WORK

7.1 Security of open source OPC UA libraries

Muhlbauer et al. [41] 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 making it vulnerable to DoS attacks and missing packet type checks in node-opcua. The issues related to the Trustlist configuration of the libraries investigated our work were not identified.

Neu et al. [43] and Polge et al. [55] showed the feasibility of DoS attacks by a Rogue Client. They suggest network traffic anomaly detection as a countermeasure. Polge et al. [55] implemented 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 our findings). The encryption of the password in Mode None is optional in the Specification [47], this design choice was taken by [32]. Encrypting credentials with untrusted certificates is susceptible to Rogue Server attacks. Based on our findings, a Middleperson attack that recovers plain-text credentials is possible. 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.

7.2 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 [33], a PKI is implemented that offers functionality to sign certificates or verify certificates using the Online Certificate Status Protocol. Meier et al. [39] 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.

7.3 Usability issues leading to insecure systems

Usability of security features is an active research topic that relates to the usability of OPC UA security features. Several studies point out that the integration of security features into programs and systems has to be facilitated [26].

Acar et al. [2] compare the usability of five Python cryptographic libraries. They find that APIs which offer fewer options lead to better security results. In addition, the documentation of the library and the availability of example code had a stronger influence on the successful completion of the task than the experience of the participant.

Krombholz et al. [35] conducted a usability study on the configuration of HTTPS. In the study 28 system administrators were

asked to configure a web server with HTTPS. They found that participants struggled to find reliable resources to learn the process, a large number of configuration options were difficult to comprehend, and the default configuration only offered weak security. Additionally, the security benefits which a protocol such as HTTPS offers are misunderstood or underestimated. Based on misconceptions some administrators decide against using secure options [19, 34].

8 CONCLUSIONS

Usable security of libraries is an active research area [2, 26, 34, 35] and showed how important features and documentation are for secure deployments. In this work, we have systematically investigated practical challenges faced to use OPC UA securely. To this end, we introduced a security assessment methodology—our assessment considers three attacks that can target OPC UA deployments, Rogue Server, Rogue Client, and Middleperson attacks.

We systematically address three 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?

To address **R1**, we conducted the first systematical survey of 48 OPC UA artifacts provided by vendors or open source. 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 artifacts do not support security features of the protocol.

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 vulnerable to the considered attacks. We analyzed 48 OPC UA artifacts, 41 of those artifacts support security features. We found that 31 out 41 artifacts present three recurring pitfalls in the Trustlist management to establish the initial root of trust.

To address **R3**, we designed, implemented, and demonstrated three types of attacks. 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.

Our findings demonstrate major security flaws in OPC UA artifacts that threaten the OPC UA security guarantees. Those results imply that there are a significant number of OPC UA deployments in industry that either do not provide full security guarantees or rely on the absence of the attacker at configuration time for new devices to bootstrap their authentication process. We believe that our proposed systematic approach is useful to increase awareness among users, developers and companies about the threats that can be produced by the three identified pitfalls in the initial key establishment and Trustlist configuration. As a complementary result, our POC tool can be used as a tool to probe for erroneous configurations and improve security in OPC UA deployments.

REFERENCES

[1] ABB. accessed November 3, 2020. AC500 USER MANAGEMENT WITH V3CONFIGURATION AND HANDLING, version 3ADR010315. https://library.e.abb.com/public/2745203d43f64bf2af9add792d5b4e46/AC500%20-%20Application%20Note%203ADR010315.pdf.

[2] Yasemin Acar, Michael Backes, Sascha Fahl, Simson Garfinkel, Doowon Kim, Michelle L Mazurek, and Christian Stransky. 2017. Comparing the usability of cryptographic APIs. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 154–171.

[3] ASNeG. accessed December 10, 2020. OPC UA implementation, version 3.8.1. https://github.com/ASNeG/OpcUaStack.

[4] Bachmann. accessed 14 January, 2021. OPC UA Client and Server, version 09/2020. https://www.bachmann.info/fileadmin/media/Produkte/Vernetzung/Produktblaetter/Software_OPC-UA-client_server_en.pdf.

[5] Beckhoff. accessed 14 January, 2021. TC3 OPC UA, version 2.9. https://download.beckhoff.com/download/document/automation/twincat3/TF6100_TC3_OPC-UA_EN.pdf.

[6] Beijer Electronics Automation AB. accessed November 3, 2020. OPC UA Server - iX Developer 2.4 KI00312A. https://www.beijerelectronics.tw/en/Support/file-archive-tree-page?docId=58657.

[7] Bosch Rexroth AG. accessed November 3, 2020. OPC UA Server Application Manual, R911403778 Edition 01. https://www.boschrexroth.com/various/utilities/mediadirectory/download/index.jsp?object_nr=R911403778.

[8] B&R Industrial Automation GmbH. accessed November 3, 2020. ADI OPC UA server User's manual, version 1.27. 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/.

[9] BSI. 2017. *Open Platform Communications Unified Architecture Security Analysis*. Technical Report. Bundesamt für Sicherheit in der Informationstechnik.

[10] Mauro Cherubini, Alexandre Meylan, Bertil Chapuis, Mathias Humbert, Igor Bilogrevic, and Kévin Huguenin. 2018. Towards usable checksums: Automating the integrity verification of web downloads for the masses. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 1256–1271.

[11] Codesys. accessed December 10, 2020. OPC UA Server, version 3.5.14.0. https://help.codesys.com/api-content/2/codesys/3.5.14.0/en/_cds_runtime_opc_ua_server/.

[12] CODESYS. accessed January 14, 2021. CODESYS OPC UA PubSub SL. https://store.codesys.com/op-ua-pubsub-sl.html?___store=en&___from_store=default.

[13] Converter Systems LLC. accessed December 10, 2020. OPC UA implementaion. https://github.com/convertersystems/opc-ua-client.

[14] Markus Dahlmanns, Johannes Lohmöller, Ina Berenice Fink, Jan Pennekamp, Klaus Wehrle, and Martin Henze. 2020. Easing the Conscience with OPC UA: An Internet-Wide Study on Insecure Deployments. In *Proceedings of the ACM Internet Measurement Conference*. 101–110.

[15] Sergej Dechand, Dominik Schürmann, Karoline Busse, Yasemin Acar, Sascha Fahl, and Matthew Smith. 2016. An empirical study of textual key-fingerprint representations. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*. 193–208.

[16] Eaton. accessed December 10, 2020. XC300 Modular control, version MN050005EN. https://es-assets.eaton.com/DOCUMENTATION/AWB_MANUALS/MN050005_EN.pdf.

[17] Eclipse Milo. accessed December 10, 2020. OPC UA implementation, version 0.5.1. https://github.com/eclipse/milo.

[18] Schneider Electric. accessed November 3, 2020. M580 BMENUA0100 OPC UA Embedded Module Installation and Configuration Guide, version 10/2019. https://download.schneider-electric.com/files?p_enDocType=User+guide&p_File_Name=PHA83350.00.pdf&p_Doc_Ref=PHA83350.

[19] Sascha Fahl, Yasemin Acar, Henning Perl, and Matthew Smith. 2014. Why Eve and Mallory (also) love webmasters: A study on the root causes of SSL misconfigurations. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*. 507–512.

[20] OPC Foundation. 2018. OPC Unified Architecture Specification v1.04. https://opcfoundation.org/developer-tools/specifications-unified-architecture.

[21] Free OPC UA. accessed December 10, 2020. OPC UA implementation, version 0.98.12. https://github.com/FreeOpcUa/python-opcua.

[22] FreeOpcUa. accessed December 10, 2020. OPC UA implementation. http://freeopcua.github.io/.

[23] French Ministry of Economy and Finance, Alliance Industrie du Futur, German Federal Ministry for Economic Affairs and Energy (BMWi), Platform Industrie 4.0, Italian Ministero dello Sviluppo Economico and Piano Nazionale Impresa 4.0. accessed July 25, 2021. The Structure of the Administration Shell: TRILATERAL PERSPECTIVES from France, Italy and Germany. https://www.de.digital/DIGITAL/Redaktion/EN/Publikation/the-structure-of-the-administration-shell.pdf?__blob=publicationFile&v=3.

[24] General Electric. accessed November 3, 2020. Certificate Management and the OPC UA Server, version 6.1. https://www.ge.com/digital/documentation/ifix/version61/Subsystems/OPCUASVR/content/opcua_about_the_trust_list.htm.

[25] Go opcua. accessed December 10, 2020. OPC UA implementation, version 0.1.13. https://github.com/gopcua/opcua.

[26] Matthew Green and Matthew Smith. 2016. Developers are not the enemy!: The need for usable security APIs. *IEEE Security & Privacy* 14, 5 (2016), 40–46.
[27] Hitachi. accessed November 3, 2020. HX Series Application Manual, verison NJI-638(X). https://www.hitachi-da.com/files/pdfs/produkte/SPS/HX/HX-CPU_Software(Tentative).pdf.
[28] Honeywell International. accessed November 3, 2020. Control Edge Builder Protocol Configuration Reference Guide, version RTDOC-X288-en-161A. https://www.honeywellprocess.com/library/support/Public/Documents/ControlEdge-Builder-Protocol-Configuration-Reference-Guide-RTDOC-X288-en-161A.pdf.
[29] Hottinger Baldwin Messtechnik GmbH. accessed December 10, 2020. OPC UA implementation. https://github.com/HBM/opcua.
[30] Issue. accessed January 14, 2021. Client certificates are not validated #392. https://github.com/FreeOpcUa/python-opcua/issues/392.
[31] Issue. accessed January 14, 2021. Client certificates are not validated #44. https://github.com/ASNeG/OpcUaStack/issues/44.
[32] Issue. accessed January 23, 2021. Regression with username/password authentication #244. https://github.com/eclipse/milo/issues/244.
[33] Gajasri Karthikeyan and Stefan Heiss. 2018. 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, 251–257.
[34] Katharina Krombholz, Karoline Busse, Katharina Pfeffer, Matthew Smith, and Emanuel von Zezschwitz. 2019. "If HTTPS Were Secure, I Wouldn't Need 2FA"-End User and Administrator Mental Models of HTTPS. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 246–263.
[35] Katharina Krombholz, Wilfried Mayer, Martin Schmiedecker, and Edgar Weippl. 2017. "I Have No Idea What I'm Doing"–On the Usability of Deploying HTTPS. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*. 1339–1356.
[36] Lenze. accessed November 3, 2020. Lenze OPC UA Communication, version v1.1. https://download.lenze.com/AKB/English/201400321/Commissioning_Lenze_OPC_UA_V1_1.pdf.
[37] LibUA. accessed October, 2020. OPC UA implmentation. https://github.com/nauful/LibUA.
[38] Tobias Marktscheffel, Wolfram Gottschlich, Wolfgang Popp, Philemon Werli, Simon Dominik Fink, Arne Bilzhause, and Hermann de Meer. 2016. QR code based mutual authentication protocol for Internet of Things. In *2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. 1–6. https://doi.org/10.1109/WoWMoM.2016.7523562
[39] David Meier, Florian Patzer, Matthias Drexler, and Jürgen Beyerer. 2020. 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, 270–277.
[40] Mitsubishi Electric. accessed November 3, 2020. MELSEC iQ-R OPC UA Server ModuleUser's Manual, version RD81OPC96-SW1DND-ROPCUA-E. https://dl.mitsubishielectric.com/dl/fa/document/manual/plc/sh081693eng/sh081693engf.pdf.
[41] Nikolas Mühlbauer, Erkin Kirdan, Marc-Oliver Pahl, and Georg Carle. 2020. Open-Source OPC UA Security and Scalability. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Vol. 1. IEEE, 262–269.
[42] National Instruments. accessed November 3, 2020. InsightCM 3.7 Manual. https://www.ni.com/documentation/en/insightcm/latest/serverconf/secure-opc-server/.
[43] Charles Varlei Neu, Ina Schiering, and Avelino Zorzo. 2019. Simulating and detecting attacks of untrusted clients in OPC UA networks. In *Proceedings of the Third Central European Cybersecurity Conference*. 1–6.
[44] Node-opcua. accessed December 10, 2020. OPC UA implementation, version 2.13.0. http://node-opcua.github.io/.
[45] Omron. accessed November 3, 2020. 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.
[46] OPC Foundation. 2017. OPC Unified Architecture Specification Part 12/section 7.7.7: GetRejectedList v1.04.
[47] OPC Foundation. 2017. OPC Unified Architecture Specification Part 4: Services v1.04.
[48] OPC Foundation. 2017. OPC Unified Architecture Specification Part 6: Mappings v1.04.
[49] OPC Foundation. 2018. OPC Unified Architecture Specification Part 2: Security Model v1.04.
[50] OPC Foundation. accessed December 10, 2020. OPC UA implemntation, version 1.4.363.49. https://github.com/OPCFoundation/UA-.NETStandard.
[51] opcua4j. accessed December 10, 2020. OPC UA implementation. https://code.google.com/p/opcua4j/.
[52] Open62541. accessed December 10, 2020. OPC UA implementation, version 1.1.2. http://open62541.org/.
[53] OpenScada UA Interface. accessed December 10, 2020. OPC UA implementation, version 1.7. http://oscada.org/main/documentation/.
[54] Panasonic Electric Works. accessed November 3, 2020. HMWIN User Manual, verison v208. https://mediap.industry.panasonic.eu/assets/download-files/import/mn_hmwin_user_peweu_en.pdf.
[55] Julien Polge, Jeremy Robert, and Yves Le Traon. 2019. 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, 1–6.
[56] Rockwell Automation. accessed November 3, 2020. FactoryTalk Linx Gateway Getting Results Guide, version FTLG-GR001C-EN-E. https://literature.rockwellautomation.com/idc/groups/literature/documents/gr/ftlg-gr001_-en-e.pdf.
[57] Rust opc UA. accessed December 10, 2020. OPC UA implementation, version 0.7. https://github.com/locka99/opcua.
[58] S2OPC. accessed December 10, 2020. OPC UA implementation, version 1.1.0. https://gitlab.com/systerel/S2OPC.
[59] Ahmad-Reza Sadeghi, Christian Wachsmann, and Michael Waidner. 2015. Security and privacy challenges in industrial Internet of Things. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1145/2744769.2747942.
[60] Siemens AG. accessed December 10, 2020. OPC UA .NET Client for the SIMATIC S7-1500 OPC UA Server, version 109737901 V1.3. https://cache.industry.siemens.com/dl/files/901/109737901/att_955026/v3/109737901_OPC_UA_Client_S7-1500_DOKU_V13_en.pdf.
[61] The Register. accessed August 26, 2022. Software developer cracks Hyundai car security with Google search. https://www.theregister.com/2022/08/17/software_developer_cracks_hyundai_encryption.
[62] Unified Automation. accessed January 14, 2021. Step-by-Step Connect Example, version 1.4.2. http://documentation.unified-automation.com/uaexpert/1.4.2/html/first_steps.html#connect_step-by-step.
[63] WAGO Kontakttechnik GmbH. accessed November 3, 2020. WAGO-I/O-SYSTEM 750, 750-8xxx OPC UA Server, version 1.0.0. https://www.wago.com/global/d/3597419.
[64] Weidmüller Interface GmbH. accessed November 3, 2020. u-create studio System manual, version 2696790000/02/04.2020. https://mdcop.weidmueller.com/mediadelivery/asset/900_91150.
[65] Yokogawa. accessed November 3, 2020. OPC-UA Server (/E3)User's Manual, version IM 04L51B01-20EN. http://web-material3.yokogawa.com/IM04L51B01-20EN_010.pdf.
[66] ZVEI – German Electrical and Electronic Manufacturers' Association Automation Division. accessed July 25, 2021. Status Report Reference Architecture Model Industrie 4.0 (RAMI4.0). https://www.zvei.org/fileadmin/user_upload/Themen/Industrie_4.0/Das_Referenzarchitekturmodell_RAMI_4.0_und_die_Industrie_4.0-Komponente/pdf/5305_Publikation_GMA_Status_Report_ZVEI_Reference_Architecture_Model.pdf.

A 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, 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 [47]. 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 (Method service set) defined for Object Nodes. Through these functionalities, a client can directly influence the plant operations. The OPC UA server authorizes clients to use its services via user authentication. Session service set provides the user authentication functionalities. User authentication occurs during session activation, after the initial application authentication.