



Bundesamt
für Sicherheit in der
Informationstechnik

OPC UA Security Analysis

02/03/2017



Authors

The OPC UA security analysis study was commissioned by the Federal Office for Information Security (BSI) and performed under the consortium leadership of TÜV SÜD Rail from 14 January 2015 until 2 December 2015. Listed in alphabetical order, the other companies and people involved in the consortium were as follows:

ascolab

- Mr. Damm
- Mr. Gappmeier
- Mr. Zugfil

SIGNON

- Mr. Plöb

TÜV SÜD

- Mr. Fiat (project manager)
- Mr. Störtkuhl

Acknowledgements

The authors would like to thank the OPC Foundation for the open and constructive cooperation. All suggestions for improvement presented in this study were taken up by the OPC Foundation and discussed intensively. This study led to an adjustment of specification and reference implementation which will be incorporated into the next updates.

An overview of the OPC Foundation's feedback on this study can be found at the following address:

<https://opcfoundation.org/security/>

Content

1 MANAGEMENT SUMMARY.....	6
2 SUBJECT OF THE ANALYSIS.....	8
2.1 SUBJECT UNDER EXAMINATION.....	8
2.2 OBJECTIVES OF THE ANALYSIS.....	9
2.3 SCOPE OF THE ANALYSIS.....	9
2.3.1 Analysis of the OPC UA Specification.....	10
Inventory of the level of knowledge with respect to the IT security of OPC UA.....	10
Threat analysis.....	10
Analysis of the specification.....	12
2.3.2 Performing of security tests on a reference implementation.....	12
2.4 EXCLUSION AND ASPECTS THAT ARE NOT TAKEN INTO ACCOUNT.....	12
2.4.1 Exclusions.....	12
2.4.2 Uncertainties.....	13
3 METRIC FOR ASSESSING THE CRITICALITY OF VULNERABILITIES.....	14
3.1 BASE METRIC GROUP.....	14
3.2 TEMPORAL METRIC.....	14
3.3 ENVIRONMENTAL METRICS.....	15
3.4 CALCULATION OF THE SCORE.....	15
4 DESCRIPTION OF THE TEST ENVIRONMENT.....	17
5 INVENTORY OF THE LEVEL OF KNOWLEDGE WITH RESPECT TO THE IT SECURITY OF OPC UA.....	19
6 EDITORIAL COMMENTS AND SUGGESTIONS FOR CORRECTION.....	22
7 RESULTS OF THE SPECIFICATION ANALYSIS.....	30
7.1 REVIEW OF SECURITY OBJECTIVES AND THREAT TYPES.....	30
7.1.1 Analysis by means of STRIDE.....	30
7.1.2 Review of the security objective definitions.....	32
7.1.3 Review of the threat type definitions.....	33
7.1.4 Review of the assignment of security objectives versus threats.....	34
7.1.5 Results.....	34
Summary of the main findings:.....	35
7.2 THREAT ANALYSIS FOR THE OPC UA PROTOCOL.....	36
7.2.1 Analysis according to the threats from Part 2.....	36
7.2.2 Analysis of the threats for elements of the OPC UA infrastructure.....	42
7.2.3 Results.....	44
7.3 ANALYSIS OF THE PROTECTION MECHANISMS ON THE PARAMETER LEVEL.....	44
7.3.1 Explanation of the analysis table.....	44
7.3.2 Detailed explanation of the results of the analysis table.....	45
7.3.3 Conclusion.....	48
7.4 FURTHER OBSERVATIONS REGARDING THE DESIGN.....	48
7.5 SUMMARY OF THE OVERALL ANALYSIS RESULTS.....	50
8 RESULTS OF THE REFERENCE IMPLEMENTATION ANALYSIS.....	54
8.1 DESCRIPTION OF THE OPC UA COMMUNICATION STACK TESTED.....	54
8.2 ATTACKS TO VULNERABILITIES IDENTIFIED IN THE SPECIFICATION ANALYSIS.....	54
8.3 PROCEDURE.....	54
8.4 CERTIFICATE TESTS.....	56
8.4.1 Description of the tests.....	56
8.4.2 Analysis of the results.....	57

8.5 STATIC CODE ANALYSIS.....	60
8.6 FUZZING.....	62
8.6.1 <i>Programming in Peach</i>	62
8.6.2 <i>Choice of fuzzing tests</i>	64
8.6.3 <i>Results</i>	64
8.7 DYNAMIC CODE ANALYSIS.....	66
8.7.1 <i>Results</i>	66
8.8 CODE COVERAGE.....	70
8.8.1 <i>Measurement</i>	70
8.8.2 <i>Analysis of the results</i>	73
8.9 PROOF OF CONCEPT.....	73
8.10 SUMMARY.....	73
9 OUTLOOK.....	75

1 Management summary

OPC Unified Architecture (OPC UA) is the central standard when implementing the Industry 4.0 future strategy and has already been used more and more often for the networking of existing industrial plants. From the very beginning, security was one of the core objectives of OPC UA as a protocol of the future: It offers the opportunity of connecting networks via different levels from the control through to the corporate level in a manufacturer-independent manner. Moreover, OPC UA, in contrast to many other industrial protocols, is equipped with integrated security functionality to secure the communication.

Objective and procedure

The objective of the current study was to carry out an inventory of the IT security of OPC UA. For this purpose, basically two analyses were performed: In the first part of the project, the specification of the version 1.02 OPC UA protocol was analysed with regard to systematic errors. This analysis was divided into the following substeps:

- Analysis of existing studies of the IT security of OPC UA which have already been carried out
- Threat analysis (analysis of security objectives and threats, analysis of threats and measures)
- Detailed analysis of the OPC UA Specification, focussing on the Parts 2, 4, 6, 7 and 12

For the analysis of the specification, no formal or semiformal methods were used. The OPC UA communication was analysed systematically with regard to the SecureChannel, Session and Discovery services (components of the communication stack of the OPC Foundation) according to the specification, however, except for the parameter level.

Based on this specification analysis, the reference implementation offered by the OPC Foundation in ANSI C of the version 1.02.344.5 OPC UA communication stack was subjected to the following security tests in the second part of the project:

- Certificate tests
- Static code analysis
- Fuzzing
- Dynamic code analysis

Main results

The *specification analysis* performed has shown that OPC UA, in contrast to many other industrial protocols, provides a high level of security.

No systematic errors could be detected.

When *analysing the reference implementation*, basically the following problems were identified:

- An important mechanism to protect against replay attacks is missing, since the sequenceNumber is not evaluated.
- Memory leaks can be used for denial-of-service attacks.
- Errors during certificate tests which might be exploited, depending on the framework application used
- No comprehensive documentation on the implemented (security) functionalities in the OPC UA communication stack

Nevertheless, the stack ran in a very stable manner during all tests, since no crashes were observed.

Recommended measures

When securing the communication with the OPC UA protocol, the following three settings are of central importance:

- **securityMode:** The securityMode should be 'Sign' (signing messages) or 'SignAndEncrypt' (signing and encrypting messages). Among other things, authentication at the application level is forced. securityMode 'None' does not provide any protection. securityMode 'SignAndEncrypt' must be used if not only integrity, but also confidential data is to be protected.
- **Selection of cryptographic algorithms:** The most secure securityPolicy 'Basic256Sha256' should be chosen provided that this is technically possible. The weakest securityPolicies sometimes use obsolete algorithms and should not be used.
- **User authentication:** The possibility of logging in with the identifier 'anonymous' should be prevented, since it does not provide any protection. On the one hand, it is not possible to comprehend who has changed, for example, the data or configuration on the server side when this generic identifier is used. On the other, an attacker could misuse this identifier to read or write data in an unauthorised manner if no adequate restriction of the rights of the identifier 'anonymous' was configured.

In addition to the immediate secure configuration of the communication itself, other, additional measures are required to protect the infrastructure. In this study, it is assumed that the operator of OPC UA communications has implemented, operates and continuously improves best-practice approaches, as described in the ICS Security Compendium [1], in automation and control systems.

This study addressed different aspects of the IT security of the OPC UA protocol in detail: The specification was not only tested for systematic errors, but the reference implementation of the communication stack was also examined with different tools. This led to a more precise picture of which points in the specification and reference implementation have to be improved and which aspects have to be taken into consideration in order to achieve a high level of IT security when using OPC UA. Furthermore, an outlook on other topics is given, which could be examined in more detail in further examinations.

2 Subject of the analysis

2.1 Subject under examination

The subject under examination of the "OPC UA security analysis" project is the OPC UA protocol in the version 1.02 (1.02.47 for Part 12) specification and a reference implementation of the OPC Foundation of the communication stack in ANSI C in the version 1.02.344.5.

The reference implementation of the communication stack cannot be run by itself and was thus integrated into a UA server application without making changes to the stack in order to be able to perform tests.

The OPC UA protocol is specified by the documents mentioned in Table 1.

No.	Title
[2]	Part 1: OPC UA Specification: Part 1 – Overview and Concepts
[3]	Part 2: OPC UA Specification: Part 2 – Security Model
[4]	Part 3: OPC UA Specification: Part 3 – Address Space Model
[5]	Part 4: OPC UA Specification: Part 4 – Services
[6]	Part 5: OPC UA Specification: Part 5 – Information Model
[7]	Part 6: OPC UA Specification: Part 6 – Mappings
[8]	Part 7: OPC UA Specification: Part 7 – Profiles
[9]	Part 8: OPC UA Specification: Part 8 – Data Access
[10]	Part 9: OPC UA Specification: Part 9 – Alarms and Conditions
[11]	Part 10: OPC UA Specification: Part 10 – Programs
[12]	Part 11: OPC UA Specification: Part 11 – Historical Access
[13]	Part 12: OPC UA Specification: Part 12 – Discovery
[14]	Part 13: OPC UA Specification: Part 13 – Aggregates

Table 1: Documents of the OPC UA Specification

As part of this study, the subsequent version 1.03 of the specification is referred to in some places. Nine of 13 Parts of the new specification were published in July 2015. The missing four parts have not yet been published at this time.

On 10 November 2015, the OPC Foundation made an updated ANSI C reference implementation of the stack available with the version number 1.02.336 for the specification version 1.03.

2.2 Objectives of the analysis

With the "OPC UA security analysis" project, the general objective of being able to make sound statements on the IT security of the OPC UA protocol was pursued. The main objectives of the project were thus to verify and validate the OPC UA Specification with regard to IT security on the one hand. On the other, a reference implementation of the communication stack was checked by means of comprehensive testing, especially also based on the findings of the OPC UA Specification analysis. Both any systematic vulnerabilities and vulnerabilities of the implementation were identified and assessed. Countermeasures were outlined wherever possible.

The following detail objectives were derived from the main objectives presented:

- OPC UA Specification
 - Identification and presentation of the most important components of OPC UA in terms of IT security
 - Detection of security vulnerabilities
 - Detection of contradictions (also between different parts of the specification)
 - Recommendation of improvements
 - Other review results such as syntax errors, errors in pictures, incomprehensible or ambiguous text passages etc.
- Testing of a reference implementation of the OPC UA communication stack
 - Identification of vulnerabilities if possible with proof of concept
 - Statements on the robustness of the stack
 - Recommendation of countermeasures wherever possible

The mentioned objectives were chosen with the following stakeholders

- BSI
- Operators of OPC UA infrastructures
- OPC Foundation

in mind, who are primarily addressed by this study.

2.3 Scope of the analysis

According to the detail objectives in Section 2.2, the scope of the analysis consisted of the following two major parts:

- Analysis of the OPC UA Specification regarding IT security
- Performing of security tests for a reference implementation of the OPC UA communication stack of the OPC Foundation

For the analysis of the specification and the security tests, the documents referred to in Table 2 were used.

No.	Source	Title
[1]	BSI	ICS Security Compendium
[15]	FIRST (Forum for Incident Response and Security Teams)	CVSS v2.0 (Common Vulnerability Scoring System)

Table 2: Sources of information for performing security analyses

2.3.1 Analysis of the OPC UA Specification

The analysis of the specification was divided into the following substeps:

- Inventory of the level of existing knowledge with respect to the IT security of OPC UA
- Threat analysis (analysis of security objectives and threats, analysis of threats and measures)
- Analysis of the OPC UA Specification in detail

Inventory of the level of knowledge with respect to the IT security of OPC UA

As a basis for the specification analysis, an inventory of the current level of knowledge regarding the topic of IT security of OPC UA was carried out. The research included relevant literature and/or publications and information from the Internet. It was examined which aspects of the specification are considered a problem by other experts. These results were also taken into account when the specification analysis was performed.

Threat analysis

To support the specification analysis, a threat analysis was carried out for the use case (see Figure 1) of the OPC UA protocol in order to address the following aspects:

- The use case includes the essential communication variants which are important with regard to OPC UA in industrial control environments:
 - Communication between OPC UA server and OPC UA client which allow full securing of the communication according to the OPC UA Specification via encryption and digital signature
 - Communication between OPC UA server and OPC UA client which do not allow full securing of the communication, since only a minimum client is installed, for example on sensors or actuators and encryption and digital signature according to the OPC UA Specification is thus not possible
 - Communication of a client from an untrusted network (the Internet, for example) with an OPC UA server which is protected via a DMZ (Demilitarized Zone).
- The OPC UA Specification can only define IT security measures which protect the communication directly. Threats which have an impact, for example, on the operating system must be counteracted with other IT security measures. These IT security measures can be derived by performing a threat analysis of the use case.

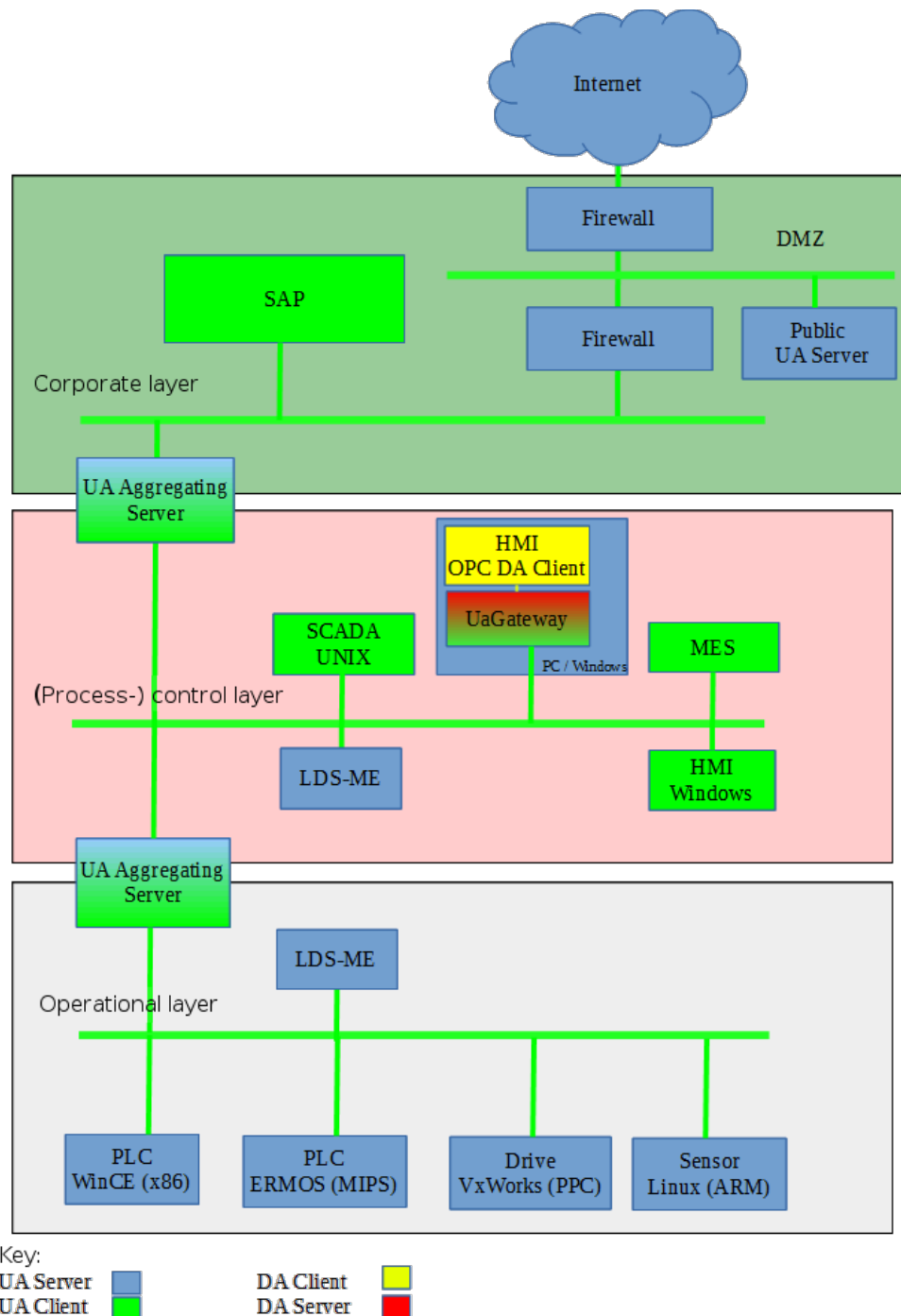


Figure 1: Use case in an exemplary OPC UA environment

Here, the use case should not be understood to reflect an exact image of the installations in control systems, for example in production. The use case, however, includes the presented communication options which are used in typical automation and control systems. The objective is to identify threats based on the use case which actually play a role in OPC UA daily routine and are not a theoretical peripheral matter, since OPC UA is not used in this constellation.

Analysis of the specification

Systematic errors in the specification were identified on the basis of the following criteria:

- a The list of measures is incorrect or incomplete
- b The assignment of the measure to the threat is incorrect or incomplete
- c The parameterisation of the measure is incorrect or incomplete

Each systematic error is assessed with its criticality according to the metric described in Chapter 3.

The focus was placed on the following documents of the specification due to their relevance to the IT security of the OPC UA communication protocol:

- Part 2 – Security Model: This part of the specification provides an overview of the OPC UA security architecture, the potential attack scenarios and the protection mechanisms defined at OPC UA.
- Part 4 – Services: In this part, the contents of the messages are defined which are exchanged between the communication partners, OPC UA client and server, as services. A large part of the protection mechanisms which are listed in Part 2 are covered by this specification at least in an abstract manner.
- Part 6 – Mappings: This part of the specification defines the protocol-specific implementation of the security model from Part 2 and the abstract service parameters from Part 4. Moreover, the protocol-specific encoding of the data structures from Part 4, the information models as well as the headers for the specific network protocols are defined.
- Part 7 – Profiles: Here, among other things, security profiles are defined which specify which set of cryptographic algorithms and key lengths is used.
- Part 12 – Discovery: This part of the specification defines, among other things, different options how OPC UA clients can find the available OPC UA servers in the network.

2.3.2 Performing of security tests on a reference implementation

The version 1.02.344.5 reference implementation of the OPC Foundation was subjected to the following tests:

- Certificate tests
- Static code analysis
- Fuzzing
- Dynamic code analysis

The tests and their results is described in Sections 8.4 to 8.7.

2.4 Exclusion and aspects that are not taken into account

2.4.1 Exclusions

The IT security of "Classic OPC" is not part of this project. It is the previous version of the OPC protocol which differs greatly from OPC UA. As protocols, only the combination consisting of UA TCP, UA Secure Conversation and UA Binary is examined.

Since the current version of the communication stack of the OPC Foundation does not support the security-Policy `RSA256SHA256`, encryption with AES256 in CBC mode and signing with SHA256, among other things, could not be tested.

Of the existing OPC UA services, only SecureChannel, Session and Discovery were tested, as only these services are processed by the OPC UA stack. All other services are also deserialised by the OPC UA stack, but processed by layers outside the OPC UA stacks, typically by software development kits (SDKs) or the OPC UA

application itself. This means that potential errors in the OPC UA Stack deserialiser were not examined when the other remaining services were used.

Rarely used functionalities such as the Kerberos user authentication were not tested either. It requires successful application authentication in any case.

2.4.2 Uncertainties

During the dynamic code analysis, error messages were generated by *Valgrind* the evaluation of which involves the following uncertainties:

- It is not possible to analyse in all cases if the cause of an error message is in the communication stack of the OPC Foundation or in the framework application.
- The approx. 400 errors reported by *Valgrind* can be potential vulnerabilities with respect to IT security and were thus analysed further. The vulnerabilities shown in Section 8.7 were identified. The analysis performed, however, must be refined by further code analyses in order to ensure that all vulnerabilities were identified and the impacts of these vulnerabilities can be defined clearly with respect to IT security.

3 Metric for assessing the criticality of vulnerabilities

Any identified vulnerabilities must be assessed regarding their criticality in order to be able to assign a weight, for example, the IT security measures provided in OPC UA. In order to be able to assess how critical these vulnerabilities are, a metric is required. The vulnerabilities identified by the specification analysis were verified by tests wherever possible. Therefore, the metric was designed in such a way that the assessment of a vulnerability could be adjusted easily according to the results of the tests.

CVSS v2.0 was chosen as the basis for such a metric. The Common Vulnerability Scoring System is recognised internationally and has been tried and tested for years. It is described briefly below.

Three areas are examined in CVSS: Base Metric Group, Temporal Metric Group, Environmental Metric Group. These metric groups include parameters which are to be assessed in the context of the area specified. The respective relations are shown in the following tables:

Base Metric Group		Temporal Metric Group	Environmental Metric Group	
Access Vector	Confidentiality Impact	Exploitability	Collateral Damage Potential	Confidentiality Requirement
Access Complexity	Integrity Impact	Remediation Level	Target Distribution	Integrity Requirement
Authentication	Availability Impact	Report Confidence		Availability Requirement

Table 3: CVSS Metric Groups (source: [15] page 3)

The values of the parameters are included in functions with which a value (score) between 0 and 10 is calculated.

For this study, CVSS v2.0 was adjusted for this project as follows:

3.1 Base Metric Group

All parameters from the 'Base Metric Group' are used and thus assessed as described in CVSS. A brief description of the parameters is provided below. Further details can be read in [15]:

- *Access Vector*: The parameter provides some information about the access required to exploit the vulnerability (e.g. locally or via the network).
- *Access Complexity*: The parameter indicates how difficult it is to perform the attack when the attacker has gained the access required. Do special conditions have to be met or can the attack always be carried out? An example of this would be a session which expires automatically after five minutes, in which case the attacker would only have a limited amount of time to carry out their attack.
- *Authentication*: Here, it is counted how often the attacker has to authenticate themselves in order to perform the attack (ranging from not at all to several times).
- *Confidentiality Impact, Integrity Impact, Availability Impact*: These parameters indicate how severely the respective security objective is compromised in the event of a successful attack.

3.2 Temporal Metric

In this group, only the first parameter "Exploitability" is used to assess a vulnerability:

- *Exploitability*: The parameter indicates whether the vulnerability is only described theoretically or a feasible attack code is already available. The advantage for this project is that the vulnerabilities identified

during the specification analysis could be assessed as 'Unproven' first. After the actual feasibility has been verified in the test phase, the assessment of the vulnerability is adjusted accordingly by grading the parameter to the higher levels 'Proof-of-Concept', 'Functional' or 'High'.

- *Remediation Level*: Here, it is assessed if a (temporary) solution of the problem is already available. Since the vulnerabilities detected within the framework of this project are unknown, a solution cannot be available yet. Accordingly, this parameter was always assessed 'Not Defined', which means that the parameter does not have an impact on the assessment of the vulnerability.
- *Report Confidence*: The parameter provides some information about how reliable the sources reporting a vulnerability are. Since the source is always the project consortium in this project, this parameter is also excluded from the assessment, which means that 'Not Defined' is used in the calculation function according to CVSS.

3.3 Environmental Metrics

The three parameters of this optional metric group, i.e. Collateral Damage Potential, Target Distribution and IT Security Requirements, help, for example, operators to evaluate how critical a vulnerability is in their own environment by taking into account, among other things, how many devices are affected in relevant infrastructures.

Such an assessment of the criticality of a vulnerability cannot be applied reasonably in this case, since there is no specific environment. Accordingly, this metric group is not used at all.

3.4 Calculation of the score

Based on the considerations above, the basis for the calculation is as follows:

```
BaseScore = round_to_1_decimal(((0.6*Impact)+(0.4*Accessibility)-1.5)*f(Impact))
Impact = 10.41*(1-(1-ConfImpact)*(1-IntegImpact)*(1-AvailImpact))
Accessibility = 20* AccessVector*AccessComplexity*Authentication
f(impact)= 0 if Impact=0, 1.176 otherwise
```

```
TemporalScore = round_to_1_decimal(BaseScore*Exploitability)
```

The parameter values are used and defined as follows:

Name	Description	Score
AccessVector (AV)	requires local access (L)	0.395
	adjacent network accessible (A)	0.646
	network accessible (N)	1
AccessComplexity (AC)	high (H)	0.35
	medium (M)	0.61
	low (L)	0.71
Authentication (Au)	multiple instances (M)	0.45
	single instance (S)	0.56
	no authentication (N)	0.704
ConfImpact (C), IntegImpact (I),	none (N)	0

AvailImpact (A)	partial (P)	0.275
	complete (C)	0.66
Exploitability (E)	unproven (U)	0.85
	proof-of-concept (POC)	0.9
	functional (F)	0.95
	high (H)	1

Table 4: CVSS parameters used for assessing the criticality and their scores

To be able to understand the assessment of a vulnerability, it is important that the score is always shown with the vector of the parameter scores:

Base: AV:[L,A,N]/AC:[H,M,L]/Au:[M,S,N]/C:[N,P,C]/I:[N,P,C]/A:[N,P,C]
 Temporal: E:[U,POC,F,H]

A vulnerability with a high potential for damage, marked by the Impact scores P or C, does not necessarily have a higher CVSS score than a vulnerability with a lower potential for damage: If, for example, it is significantly easier to perform a vulnerability with a low potential for damage (parameters AccessVector and AccessComplexity), a higher CVSS overall score can be achieved than for the vulnerability with a high potential for damage.

4 Description of the test environment

The following diagram shows the architecture of the test environment for the security tests of the communication stack of the OPC Foundation:

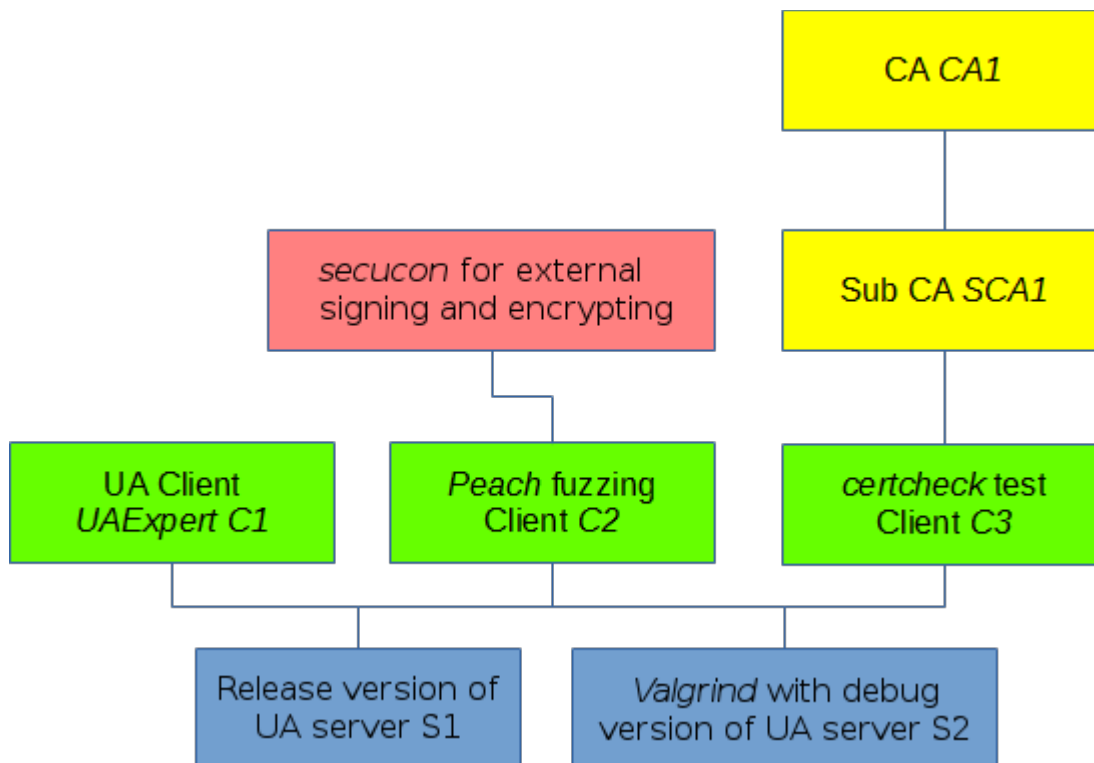


Figure 2: Overview of the test environment

The following elements are shown:

- The clients are presented in green:
 - UA client *UAExpert C1*: This client is part of the Unified Automation software package and was merely used in the beginning to test the connection to the server. This client was no longer used in AP 4.
 - *Peach* fuzzing client *C2*: The fuzzing framework *Peach* was used extensively for the fuzzing tests and the dynamic code analysis in AP 4. With *Peach*, the client side was simulated and thus the OPC UA server tested. Further details on the fuzzing tests and the dynamic code analysis can be read in Sections 8.6 and/or 8.7.
 - *certcheck* test client *C3*: To test the implementation of the certificate verification mechanisms, Unified Automation tools which simulate the establishment of the connection from a client to an OPC UA server, were used. The configuration of the tools makes it possible to test different PKI (Public Key Infrastructure) environments. Further details on these tests are described in Section 8.4.
- The servers are shown in blue:
 - Release version of the OPC UA server *S1*: This server was used both for fuzzing and for the tests of the certificate verification mechanisms (for more details, see Section 8.4).
 - *Valgrind* with debug version of the OPC UA server *S2*: This server was used for the dynamic code analysis and to measure code coverage. More information on code coverage can be found in Section 8.8.
- PKI elements required for certificate tests with the *certcheck* tool are marked in yellow:
 - Root CA (Certificate Authority) *CA1*: If the tested certificates are not self-signed, a CA is required. The certificates of the Root CA are self-signed.

- Sub CA SCA1: In some certificate tests, a multi-level PKI is simulated. For this purpose, a Sub CA is needed. The Sub CA certificates were signed by the Root CA.
- Moreover, the external application *secucon* is highlighted in salmon. This tool was developed by ascolab within the framework of this project to be able to sign and encrypt messages generated in *Peach* if necessary and to decrypt the responses from the tested server and remove the contained signature. Further details on the communication between *Peach* and *secucon* can be found in Section 8.3.

All components of the test environment shown in Figure 2 are included in a *VirtualBox* VM (virtual machine) with 32-bit Kali Linux operating system. Up to six instances of this VM ran in parallel on different computers to reduce the runtime of the fuzzing tests.

Instance	Host operating system	Computer	VirtualBox version
1	Win7 32-bit	Work laptop	4.2.18
2	Win7 64-bit	lab laptop 1	5.0.2
3	Linux 64-bit	lab laptop 2	4.3.20
4	Win7 64-bit	pentest laptop 1	4.3.10
5	Win7 64-bit	pentest laptop 2	5.0.2
6	Windows Server 2008 R2 64-bit	server 1	4.3.8

Table 5: Distribution of VM instances

As the basis for generating the instances 2 up to 6, the instance no. 1 of the VM was used.

The tests were performed and controlled with *bash* scripts. *Wireshark* was used to visualise and analyse the network traffic, *openssl* to generate the certificates. The compiler used in the test environment for the servers *S1* and *S2* is *gcc*.

Other key tools for the actual carrying out of the tests are described in detail in Section 8.4 to 8.8.

The following table summarises the tools used:

Tool	Purpose	Platform	Version
<i>Wireshark</i>	Visualisation and analysis of the network traffic	Linux 32-bit	1.99.8
<i>openssl</i>	Generation of certificates	Linux 32-bit	1.0.1e
<i>cppcheck</i>	Static code analysis	Linux 32-bit	1.54
<i>Valgrind</i>	Dynamic code analysis	Linux 32-bit	3.8.1
<i>gcc</i>	Compiler	Linux 32-bit	4.7.2
<i>lcov + genhtml</i>	Measurement and graphical representation of code coverage	Linux 32-bit	1.9
<i>Peach</i>	Fuzzing framework	Linux 32-bit	3.1.124.0
<i>bash</i>	Control of the tests	Linux 32-bit	4.2.37

Table 6: List of key tools used for the tests

5 Inventory of the level of knowledge with respect to the IT security of OPC UA

Below (for all further chapters), the term "observations" refer to certain potential improvements. Observations, however, do not necessarily mean systematic errors of the specification regarding IT security. Systematic errors mean that the specification includes a security vulnerability in the definition of the OPC UA protocol.

Internet research has shown that vulnerabilities in the implementations of OPC UA applications have already been reported and eliminated in a few cases in the past:

- Excerpt from [16]: "The SCADA OPC-UA TCP protocol contains issues that could allow an unauthenticated, remote attacker to spoof network communications or cause buffer overflows on a targeted system."
- Excerpt from [17]: "OPCUA wrapper SP3 Enhancement: XML parser cyber-security vulnerability fix"

The most interesting results, however, can be found in [18], [19] and [20]. Below, texts marked in red refer to aspects which are still outstanding and have not been taken into account by the OPC Foundation so far, whereas tests marked in green are aspects which were considered eliminated.

- In [18], Digital Bond provides information on the results of an IT security analysis of OPC UA which was already completed in August 2008 based on the specification valid at that time. Furthermore, the IT security of a software package made available by the OPC Foundation was examined as part of the same mandate. It is presumed that it was an exemplary implementation by the OPC Foundation in C# (for the Microsoft .NET environment). Thus, the examination does not necessarily show errors or aspects which also affect the ANSI C reference implementation.

At that time, the specification analysis has revealed the following:

- Digital Bond wrote that self-signed certificates should not be trusted automatically, which means without an additional verification.

From our point of view, this requirement has been met with the introduction of the Trust List if it provides adequate protection against manipulation.

- Secure Channels should not be called such if they do not provide security with the securityPolicy setting 'None'.

This important note was discussed several times in a working group of the OPC Foundation. This working group, however, came to the conclusion that changing the name is no longer possible, because the designation is included in APIs, among other things. In order to prevent a false feeling of security from being conveyed, notes explicitly pointing out the fact that no security is provided in the securityMode 'None' were integrated into the specification.

- If RegisterServer is allowed with the securityMode 'None', this can be misused by an attacker to fake a trusted server.

This is prohibited explicitly in Part 7 Table 3. A further note is also planned in Part 4 in version 1.03.

- The specification did not include guidelines for random number generators. The specification relies on these numbers being generated with adequate entropy.
- At that time, there were also contradictions in the specification as to whether the packages had to be signed and encrypted in the case of *OpenSecureChannel* and *CloseSecureChannel*.

No contradictions could be found in version 1.02. However, it should be mentioned explicitly that *OpenSecureChannel* requests and responses are also encrypted in the securityMode Sign.

Here, the risks of poor entropy should be pointed out and minimum requirements for the random number generators set in the form of functionality classes. For more information, see [21] Section 9 and [22] Section 4.

- The software testing by [18] had revealed the following problems:
 - Heap and stack overflows

- Crashes in the case of fuzzing tests
Digital Bond was confident that the OPC Foundation would eliminate most vulnerabilities in the next version of the specification and the source code.
- A part of its Smart Grid Interoperability Panel (SGIP) in the third quarter of 2012, NIST published the results of the Cyber Security Working Group (CSWG) with respect to the evaluation of the IT security of OPC UA [19]. This evaluation based on the specification valid at that time. The following essential points were noted in the result reports:
 - A failure of the audit functionality may result in gaps in a subsequent forensic analysis. Therefore, this should be taken into consideration when implementing the audit functionality.
 - It should be mentioned explicitly that cipher suites which do not meet the requirements of FIPS 140-2, NIST SP 800-131A or comparable reference documents should only be used in exceptional cases.
With respect to the key lengths and protocol versions in TLS, the NIST recommendation are referred to. Outdated or insecure cipher suites are not addressed at all in the specification.
 - In Part 6 Annex D Table D2, SHA-1 should be replaced.

There has not been an update yet concerning this matter.

- In Part 7, it should be pointed out explicitly that SHA-1 is considered outdated and SHA-256 should be used instead.

There has not been an update yet concerning this matter.

- Any information on which TLS cipher suites are outdated and which are recommended is missing in Part 7.

There has not been an update yet concerning this matter.

- In Part 7, it should be pointed out explicitly that passwords must never be transmitted as plaintext.

In [5] and [7], it is consequently referred to the fact that authentication with secret-based userIdentityTokens may be carried out only in an encrypted form.

- In [20], the authors of the paper examined at the end of 2014 how the IT security mechanisms are implemented in OPC UA.
 - They conclude that some of the algorithms chosen in the profiles for the encryption and signing are outdated.

According to [21] and [23], the following BSI recommendations must be taken into account:

- ☐ SHA-1 should not be used at all. The only exception is HMAC-SHA1.
- ☐ EME-OAEP is recommended as formatting procedure for RSA.
- ☐ TLS 1.0 may not be used any more.
- ☐ The fact that RSA keys should have a length of at least 2000 bits is addressed in a satisfactory manner in [8]. However, it is doubtful that the personnel who is responsible for the configuration of OPC UA clients and servers take notice of such notes from the specification.
- Moreover, their examination has shown that the implementation requires unnecessarily high amounts of computing power, which may have an adverse impact in the case of field devices with small processors.
- The same private key is used for signing and decryption.
- The algorithms based on elliptic curves (ECDSA and ECDH) cannot be used in OPC UA due to the dual use of the private keys.

Summary of the main findings:

ID	Observation	Recommendation
1	The name "Secure Channel" suggests IT security which, however, is not included in the securityMode None. Unfortunately, the name cannot be changed any more.	None
2	For random number generators, no notes regarding minimum requirements are provided in the OPC UA Specification.	Notes regarding minimum requirements of recognised institutions (BSI, for example) in the OPC UA Specification should be specified for random number generators.
3	There are no notes regarding outdated algorithms or algorithms considered insecure, especially SHA-1 and cipher suites.	Notes regarding outdated algorithms or algorithms considered insecure, especially SHA-1, and cipher suites, should be provided, e.g. links to recognised institutions (e.g. BSI) which give recommendations for cryptographic algorithms at regular intervals.
4	The private key is used in OPC UA both for the signature and for the encryption.	Different key pairs for the signature and encryption should be requested in the OPC UA Specification.
5	At the moment, it is not possible to also use algorithms based on elliptic curves.	OPC UA should provide for this option in future in order to also allow for adequate security on devices with low processing power.

Table 7: Main findings of the inventory

6 Editorial comments and suggestions for correction

The editorial analysis has revealed different aspects which could lead to comprehension and/or implementation problems (e.g. simple typing errors, ambiguous wording, inconsistencies etc.) or even be relevant to IT security. The IT security of the protocol is analysed in more detail in Chapter 7 of this study.

Table 8 summarises the aspects found: In the first column, a unique ID is defined. The following four columns in Table 8 are used to provide information on where the text passage, table or figure can be found in the different documents of the specification. In the 'Observation' column, it is explained what is unclear or incorrect. If necessary, these comments are supplemented with recommendations regarding how the mentioned passages could be improved. The aspects which were more important from our point of view (without an explicit assessment scheme) were marked in yellow. Security-relevant aspects are marked in red. The most important findings, especially the security-relevant aspects, are summarised in 9.

ID	Docu- ment	Page	Chapt er	Section/ Table/ Diagram	Observation	Recommendation
1	Part 2	8	4.2		The definitions of IT security terms does not correspond to the definitions of internationally recognised standards such as ISO 27000 [24].	This is recommended, since terms should be used consistently in order to avoid misunderstandings.
2	Part 2	9	4.3.2	Section 1	Why is only "server flooding" mentioned? Under certain circumstances, "client flooding" might also be worth considering as a threat.	Please add "client flooding"
3	Part 2	16	4.12.1	Section 1	The last sentence is incomplete.	
4	Part 2	22	5.2.2.1		Identification takes place with the X.509 certificate and authentication with the private key.	
5	Part 2	22	5.2.2.2	Section 1	The userIdentityToken does not occur in the openSecureChannel request, but in the ActivateSession request.	
6	Part 2	22	5.2.4-5.2.5	Section 2	The PKI is not responsible for symmetric key management. They are derived based on the client and server nonces.	
7	Part 2	25	6.11	Section 4	The first sentence is not correct in this general form.	The statement should be restricted to the use of certificates in OPC UA.
8	Part 2	28	6.11	Section 2	"unique key used to create and verify digital signatures" is not correct: The private key is used to create the signature, whereas the public key is used for verification. However, a key cannot be used for both.	
9	Part 4	12	5.4.1	Fig. 9	The figure does not make it clear that a SecureChannel with securityMode 'None' has to be established first for FindServers and GetEndpoints.	This information should either be shown in the figure or explained in the text.
10	Part 4	16	5.4.3.1	Fig. 10	"CreateSecureChannel" occurs several times in figures, tables or text passages.	Replace by "OpenSecureChannel"
11	Part 4	19	5.4.4.2	Table 5	The text says that the authenticationToken should be "omitted", also on page 14, 16 and 23.	This should be replaced by "null" or "empty".
12	Part 4	22	5.5.2.1	Section 2	The signature is carried out with the private key, and not with the certificate. The encryption is performed with the public key, and not with the certificate.	

13	Part 4	23	5.5.2.2	Table 7	Not certificate, but private key	
14	Part 4	23	5.5.2.2	Table 7	For secureChannelId, two data types are mentioned in Part 4 and 6: ByteString and Uint32. The same applies to Table 9.	According to ascolab, Part 6 is decisive. Table 7 should be revised or deleted to ensure that there are no contradictions any more.
15	Part 4	23	5.5.2.2	Table 7	"channelId" should be replaced by "secureChannelId".	
16	Part 4	23	5.5.2.2	Table 7	Lower and/or upper limits are missing for the requestedLifetime parameter. (In the reference implementation of the communication stack, one hour is used.)	Please recommend (ranges of) values
17	Part 4	24	5.5.3.2	Table 9	According to Part 6 p. 47, no response to the closeSecureChannel request is sent.	The passage in Part 4 should be reworded.
18	Part 4	27	5.6.2.2	Table 11	For maxResponsesMessageSize, information on default values is missing.	Recommendations for a lower limit or a default value for maxResponsesMessageSize
19	Part 4	28	5.6.2.2	Table 11	For serverSoftwareCertificates, it can be read that an empty array has to be used.	It should be specified in detail what "omitted", "empty" and "null" means.
20	Part 4	28	5.6.2.2	Table 11	If securityMode 'None' is used, no client certificate is available for the calculation of the serverSignature.	The problem has already been eliminated in version 1.03.
21	Part 4	99	6.1.5		The last sentence is too general and/or incorrect, since no signature is created as proof in the case of secret-based userIdentityTokens.	
22	Part 4	147	7.29	Fig. 30	"GetSecureChannelInfo" occurs in the figure. However, this function is not defined and/or explained anywhere.	ascolab recommends using the following wording "information requested by the server application from the communication stack" or a similar wording.
23	Part 4	147	7.29	Fig. 30	The SessionId is not a parameter in ActivateSession requests.	
24	Part 4	148	7.31	Table 161	Should "may" be replaced twice by "shall" for keyUsage?	
25	Part 4	154	7.35.1		For secret-based userIdentityTokens, it is required that they are encrypted. There is no such requirement for signature-based userIdentityTokens.	It should be mandatory ("shall") that a userTokenPolicy or the securityPolicyUri is not 'None' at all times to ensure that an algorithm is available for creating the signature.
26	Part 4	155	7.35.3	Table 171	"Certificate" should be replaced by "public key".	
27	Part 4	156	7.35.5	Table 173	"Certificate" should be replaced by "public key".	

28	Part 6	6	5.1.2	Table 1	The byte definition is wrong.	Replace by "0 to 255"
29	Part 6	7	5.1.6	Section 3	According to the specification, variant arrays can be nested. The same applies to diagnosticInfo.	A reasonable maximum permissible recursion depth should be specified, e.g. 10.
30	Part 6	36	6.7.1	Section 2	Should the value be 8196 or 8192?	According to ascolab, it should be 8192.
31	Part 6	38	6.7.2	Table 30	MaxCertificateSize is defined nowhere.	It should be explained how and where this parameter is configured.
32	Part 6	38	6.7.2	Table 30	Change the data type of SecurityPolicyUriLength and ReceiverCertificateThumbprintLength to byte or UInt32. Change data type of SenderCertificateLength to UInt32.	It is generally recommended for parameters to define all integer data types as unsigned if the respective parameter cannot have a negative value. In this special case, however, it makes more sense to define the three parameters SecurityPolicyUri, SenderCertificate and ReceiverCertificateThumbprint as ByteStrings, which corresponds to the implementation, since the length -1 is permissible as in the case of NULL ByteStrings.
33	Part 6	38	6.7.2	Table 30	The information is contradictory: ReceiverCertificateThumbprintLength is always 20 and ReceiverCertificateThumbprint is null if the message is not encrypted.	
34	Part 6	39	6.7.2		What happens if a gap is detected for the sequenceNumber ?	It should be described explicitly how an OPC UA application should respond to such a case (error message, loss of connection etc.).
35	Part 6	41	6.7.4	Table 35	The first sentence on p. 41 and the fact that other parameters with other data types are in Table 35 differing from than those specified in Part 4 can cause confusion.	It should be pointed out in more detail Part 4 provide a general description of the services and that the protocol-dependent details are addressed in Part 6. Accordingly, Part 6 takes priority if there are contradictory statements. If possible, however, Part 4 should preferably be revised in such a way that there are no contradictions.
36	Part 6	41	6.7.4		OpenSecureChannel requests and responses are also encrypted in the securityMode 'Sign'.	This should be mentioned explicitly.

37	Part 6	42	6.7.5		Information on the quality or minimum requirements of random number generators are missing.	Recommendations on the quality or minimum requirements of random number generators
38	Part 6	43	6.7.6	Section 3	Due to the different data type definitions ByteString and UInt32, the default values also differ in the case of a new SecureChannel: 0 and null.	More attention should be paid to the consistency of data types and default values between Part 4 and 6. Moreover, it should be specified precisely what "null" actually means: For example, it can be 0 for integer, an initialised empty data structure (length 0) or an uninitialized data structure (null pointer).
39	Part 6	63	D.2	Table D1	Replace "UntrustedIssuerStore" by "IssuerStore" and "UntrustedIssuerList" by "IssuerList" in the first sentence.	
40	Part 6	64	D.3	Table D2	The data type for ValidationOptions should be changed to byte or UInt32, as no negative values are permissible.	
41	Part 6	67	D.6	Table D6	The table provides options which are an insecure configuration of the communication.	It should be pointed out explicitly that some of these options are insecure.
42	Part 7	15	5.3	Table 11	At Security Basic 128Rsa15: http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/dk/p_sha1 does no longer exist.	
43	Part 7	18-19	5.3	Table 11	The "security levels" 1 to 3 are not defined anywhere. What is their benefit?	Add explanations to the security levels
44	Part 7	37	5.5	Table 21	A category "documentation – security best practices/recommended settings" could be added.	
45	Part 7	78	6.5.124-126		Why is a PKI required for these profiles? Can self-signed certificates not be used with trust lists?	
46	Part 12	4	4.2.2	Section 2	RegisterServer2 is not defined in version 1.02.47.	
47	Part 12	5	4.3.1		FindServersOnNetwork is not defined in version 1.02.47.	
48	Part 12	13	6.1		QueryServers is not defined in version 1.02.47.	
49	Part 12	19	7.2	Fig. 13	The figure is missing.	It is available in version 1.03.53.
50	Part 12	27	7.6.4		Information on how the privateKeyPassword is determined is missing.	Addition of minimum requirements or recommendations for the quality of the password
51	Part 12	32	7.7.3	Section 2	Information on the default value of MaxTrustListSize is missing.	Recommendations for the default value of MaxTrustListSize

52	Part 12	40	B.3	Table 23	The last sentence in the field "scheme" is incomplete.	The problem has already been eliminated in version 1.03.
53	Part 12	40	B.3		If TTL stands for Time To Live, it is a byte. Accordingly, the upper limit is 255; usually, 64 is recommended as a default value.	If TTL stands for something else, it should be defined. Otherwise, the text should be corrected.
54	Part 12	47	F.2		How are rogue servers identified in the provisioning state?	
55	Part 12	47	F.2	Section 2	What is the "OPC UA interface"?	ascolab suggests using "serverConfigurationType" instead.
56	Part 12	48	F.2	Section 3	"authorized" should be replaced by "unauthorized". This has already been done in version 1.03.	
57	Generally				At the moment, the following setting combination is allowed: securityMode 'Sign' or 'SignAndEncrypt' and securityPolicyUri 'None'. This results in insecure communication.	It is recommended that securityPolicyUri 'None' is prohibited in the securityMode 'Sign' and 'SignAndEncrypt' and that this validation of the configuration is also added as conformity test.
58	Generally				It is recommended that "X509" is replaced by "X.509 v3" in all places.	
59	Generally				Addressing of the vulnerabilities found in Section 7.4	
60	Generally				The configuration of an OPC UA application includes a number of parameters which determine lower or upper limits, e.g. for buffer sizes or number of messages. In most cases, however, default value information or a recommendation regarding the range of reasonable values is missing. It would make the work of developers and administrators significantly easier if a list of such default values would be available in the specification.	We recommend using a list of such default values. A distinction could be made between two device classes: embedded devices with limited power on the one hand and PCs or workstations on the other.

Table 8: Questions and comments with respect to the specification

Summary of the main findings:

ID	Observation	Recommendation
1	<p>Inconsistencies between Part 4 and 6: Both Part 4 and Part 6 particularly describe those services which are crucial for the secure establishment of the connection on the basis of a SecureChannel and a Session. Part 4, however, is quite general, whereas Part 6 addresses the particularities of the available protocols. This is why the parameters which are contained in the different parts of a message partially differ. The assignment of some data types which are used to define the parameters is also different. This is confusing and prone to errors.</p>	<p>The mentioned inconsistencies between Part 4 and Part 6 should be eliminated.</p>
2	<p>Uncertainties in the case of parameters which are not used: Due to the possibility of switching on or off signing and encryption via the securityMode, parameters which are not required are also transmitted in a message in certain cases. It is not clear, however, which values are to be used for the parameters affected in these cases. This can result in implementation errors and incompatibilities between OPC UA applications of different manufacturers.</p>	<p>For the parameters of a message which are not required, the parameter values to be set should also be defined.</p>
3	<p>Missing list of reasonable default values: The configuration of an OPC UA application includes a number of parameters which determine lower or upper limits, e.g. for buffer sizes or number of messages. In most cases, however, default value information or a recommendation regarding the range of reasonable values is missing. It would make the work of developers and administrators significantly easier if a list of such default values would be available in the specification. A distinction could be made between two device classes: embedded devices with limited power on the one hand and PCs or workstations on the other.</p>	<p>The default values of parameters should always be determined. A list of such default values should be created and a distinction made between two device classes: embedded devices with limited power on the one hand and PCs or workstations on the other.</p>
4	<p>Better protection against attacks to secret-based userIdentityTokens: At the moment, the specification does not define how to proceed in the case of repeatedly failed user authentication.</p>	<p>In order to increase the security further, one suggestion would be that servers support doubling the period of time delay for processing of the next authentication for each failed login, until some maximum time is reached. For example, when one minute has been reached. Upon successful login, this limit is reset to one second. This function should be able to be enabled on a server.</p>
5	<p>At the moment, the following setting combination is allowed: securityMode 'Sign' or 'SignAndEncrypt' and securityPolicyUri 'None'.</p>	<p>It is recommended that securityPolicyUri 'None' be prohibited in the securityMode 'Sign' and 'SignAndEncrypt' and that this validation of the</p>

	This results in insecure communication.	configuration also be added as a conformity test.
6	Part 4, p. 154, Chap. 7.35.1 For secret-based userIdentityTokens, encryption is required. There is no such requirement for signature-based userIdentityTokens.	It should be mandatory ("shall") that either a userTokenPolicy or the securityPolicyUri not be 'None' at all times to ensure that an algorithm is available for creating the signature.
7	Part 6, S. 7, Chap. 5.1.6, Section 3 According to the specification, variant arrays can be nested. The same applies to diagnosticInfo.	A reasonable maximum permissible recursion depth should be specified, e.g. 10.
8	Part 6, p. 39, Chap. 6.7.2 What happens if a gap is detected for the sequenceNumber?	It should be described explicitly how an OPC UA application should respond to such a case (error message, loss of connection etc.).
9	Part 6, p. 41, Chap. 6.7.4 OpenSecureChannel requests and responses are also encrypted in the securityMode 'Sign'.	This should be mentioned explicitly.
10	Part 6, p. 67, Chap. D6, Table D6 Table D6 provides options which are an insecure configuration of the communication.	It should be pointed out explicitly that some of these options are insecure.

Table 9: Essential editorial comments and security-critical observations

7 Results of the specification analysis

The analysis presented here includes the following items:

- 1 Review of security objectives and threat types (see Section 7.1)
Here, the objective is to determine the correctness and/or gaps regarding security objectives and threat types, as they are described in Part 2 of the specification. Furthermore, it is reviewed if the assignment of threats to security objectives is correct. The analysis leads to a suggestion for improvement with respect to the security objectives and threat types, which is used in the threat analysis (see item 2).
- 2 Threat analysis for the OPC UA protocol (see Section 7.2)
The threat analysis consists of two sub items:
 - a Analysis according to the threats from Part 2 (see Section 7.2.1)
Here, the objective is to identify the criticality according to the CVSS assessment scheme. Moreover, the analysis should demonstrate how the threats from Part 2 can be put in more specific terms for OPC UA. The results of the analysis are shown in Table 16. In addition, this threat analysis can be used to derive test cases for the reference implementation of the OPC UA communication stack.
 - b Analysis of threats for elements of the OPC UA infrastructure (see Section 7.2.2)
- 3 Analysis of the protection mechanisms of the OPC UA protocol on the parameter level (see Section 7.3)
Here, it is analysed whether the security measures of the OPC UA protocol counteract the threats (see item 1 and item 2). The detailed analysis also includes the parameter level of the protocol. This analysis closes the "protection mechanism – threat – security objective" chain and makes it possible to make statements as to whether security objectives are achieved and/or threats are counteracted adequately.

The results of the analysis are summarised in Section 7.5.

7.1 Review of security objectives and threat types

7.1.1 Analysis by means of STRIDE

In Part 2, a number of threat types have already been taken into account. Thanks to independent threat modelling methods such as STRIDE, this list may even be supplemented. In the following table, which six threat types are hidden behind STRIDE is broken down. Furthermore, it is shown whether all STRIDE threat types are covered by the threat types described in Part 2:

	S	T	R	I	D	E
Abbreviation	Spoofing	Tampering	Repudiation	Information Disclosure	Denial of Service	Elevation of Privilege
Definition	Pretending to be something or someone other than yourself	Modifying something on disk, on a network, or in memory	Claiming that you didn't do something, or were not responsible	Providing information to someone not authorized to see it	Absorbing resources needed to provide service	Allowing someone to do something they're not authorized to do
Violating	Authentication	Integrity	Non-Repudiation	Confidentiality	Availability	Authorization
Message Flooding					X	
Eavesdropping				X		
Message Spoofing	X					
Message Alteration		X				
Message Replay		X				X
Malformed Message		X				
Server Profiling				X		
Session Hijacking						X
Rogue Server	X			X	X	X
Compromising User Credentials						X

Table 10: Mapping of threats from Part 2 to STRIDE

This results in the following assessment (for the interpretation of the threats, see Section 7.1.3):

- 'spoofing' is covered by 'message spoofing' in Part 2.
- The term 'tampering' is broader than 'message alteration' in Part 2. Which new attacks arise beyond pure communication in the case of the threat type 'tampering' is described in Table 17.
- 'repudiation' is missing in Part 2 and was therefore added to Table 16. Furthermore, the security objective 'non-repudiation' which is not specified in Part 2 can be derived here. Standards such as ISO/IEC 27000 [24] also define this security objective.
- 'information disclosure' is covered on the communication level with 'eavesdropping' in Part 2. Further attacks are described in Table 17.
- 'denial of service' is broader than 'message flooding' in Part 2. Therefore, the threat type 'message flooding' was replaced by 'denial of service'.
- 'elevation of privilege' is covered by 'compromising user credentials' for the field of communication. Other aspects, such as overwriting permits loaded in memory, were considered in Table 17. Threats which include the manipulation of rights on the address space level are also listed in Table 17.

Summary:

The security objectives in Part 2 were supplemented by the security objective 'non-repudiation'.

The new threat type 'repudiation' and the broader threat type 'denial of service' were used for the further analysis.

Correspondingly, the security objective '*non-repudiation*' and the threat types '*repudiation*' and '*denial of service*' are used in Table 14.

7.1.2 Review of the security objective definitions

In the following table, the security objective definitions in Part 2 are compared to the definitions in ISO/IEC 27000:

Security objective	Definition Part 2 [3]	Definition ISO/IEC 27000, 2009-05-01 [24]
Auditability	a security objective that assures that any actions or activities in a system can be recorded.	
Authentication	a security objective that assures the identity of an entity such as a <i>Client</i> , <i>Server</i> , or user can be verified.	provision of assurance that a claimed characteristic of an entity is correct
Authorization	the ability to grant access to a system resource.	
Availability	a security objective that assures a system is running normally; meaning no services have been compromised in such a way to become unavailable or severely degraded.	property of being accessible and usable upon demand by an authorized entity
Confidentiality	a security objective that assures the protection of data from being read by unintended parties.	property that information is not made available or disclosed to unauthorized individuals, entities, or processes
Integrity	a security objective that assures that information has not been modified or destroyed in an unauthorized manner, see IS Glossary	property of protecting the accuracy and completeness of assets

Table 11: Comparison of IT security definitions

The ISO/IEC standard does not provide definitions for '*Auditability*' or '*Authorization*'. The definitions for '*Authentication*', '*Availability*', '*Confidentiality*' and '*Integrity*' are broader in ISO/IEC 27000 than in Part 2. Part 2 focuses on communication and data.

The definition of '*Authorization*' seems to be insufficient, since access rights to a resource should not simply be granted, but only the absolutely necessary access rights (according to the need-to-know principle) to a resource should be granted correctly.

7.1.3 Review of the threat type definitions

Below, the threat type definitions in Part 2 were restricted or strengthened as needed so that the threat types resulting from this form disjoint sets.

Message flooding:

'*message flooding*' is replaced by the threat type '*denial of service*' which not only directly covers attacks to communication, but also generally refers to attacks with the objective of disrupting the availability of an IT system, an application etc. or disabling it completely.

Message spoofing:

The threat type 'message spoofing' does not correspond to the definition of spoofing as it is usually understood in IT security. Spoofing always includes feigning identities (person, application, process etc.) and this interpretation of message spoofing was taken as the basis for the further analysis. In order to distinguish message spoofing from session hijacking, it was assumed that in the case of message spoofing, a new communication channel is established. Intervention in an existing communication falls under session hijacking.

Malformed messages:

For the further analysis, the following assumptions were made: In the case of the threat 'malformed messages', an attacker creates or modifies a message in such a way that its format is incorrect. According to this definition, a message with an incorrect signature would fall under message spoofing (the attacker feigns an identity), whereas a message with incorrect specification of field lengths falls under malformed messages.

Rogue server:

The definition of rogue server included in Part 2 can be interpreted in a narrow or in a very broad manner. If any OPC UA server can be injected as a rogue server into the OPC UA communication infrastructure, all security objectives are threatened in principle. If the rogue server can also pretend to be another authorised OPC UA server, the security objectives are threatened to a significantly larger extent. How a rogue server is to be understood, however, is not described in the specification in more detail. In the further analysis, the threat 'rogue server' is interpreted in such a way that the attacker is able to inject an OPC UA server which, however, cannot pose as another authorised OPC UA server.

Compromising user credentials:

The threat 'compromising user credentials' was interpreted in such a manner that it is not only about the compromising of the data required for the user authentication, but also about the compromising of the data required for the application authentication.

7.1.4 Review of the assignment of security objectives versus threats

The assignments of *security objectives versus threat types* according to Part 2 of the specification are provided in the following table (assignments according to [3], Section 4.3 of the specification):

Threat	Messa ge Floodi ng	Eavesd roppin g	Messa ge Spoofi ng	Message Alteration	Mes sage Rep lay	Malfor med Messag es	Server Profilin g	Session Hijacking	Rog ue Serv er	Comprom ising User Credential s
Security objective										
Authentication		(X)					(X)	X	X	
Authorisation		(X)	X	X	X		(X)	X	X	X
Confidentiality		X					(X)	X	X	X
Integrity		(X)	X	X		X	(X)	X		
Auditability		(X)					(X)	X	X	
Availability	X	(X)				X	(X)	X	X	

Table 12: Security objectives versus threat types according to the specification

Explanation: The crosses in brackets are indirect threats.

7.1.5 Results

Due to the sometimes vague threat type definitions (see Section 7.1.3), threat types cannot be mapped to security objectives in an unambiguous manner. Our evaluation of the impact of threat types on security objectives differs from that of the OPC Foundation. Accordingly, the following table results from our point of view:

Threat	Deni al of Serv ice	Eav esd rop pin g	Messag e Spoofi ng	Messag e Alterati on	Messa ge Repla y	Mal-fo rmed Messag es	Server Profilin g	Session Hijacki ng	Rogue Server	Compro mising User Credenti als	Reput iation
Security objective											
Authenticati on		X		X	X		(X)	X	X	X	
Authorisatio n		X	X	X	X		(X)	X	X		
Confidentiali ty		X					(X)	X	X	X	
Integrity				X			(X)	X			
Auditability				X			(X)	X	X		
Availability	X					X	(X)	X	X		
Non-reputia tion				X			(X)	X			X

Table 13: Security objectives versus threat types according to the current analysis

Explanation: The crosses in brackets are indirect threats.

The list of the security objectives was supplemented by 'Non-repudiation': OPC UA is equipped with mechanisms in order to log requests and the authentication of the application and user on the basis of cryptographic signatures in the case of securityMode 'Sign' and 'SignAndEncrypt' which meet the security objective 'Non-repudiation'. The list of threat types was also supplemented by the category 'repudiation' due to STRIDE. The threat type 'message flooding' was replaced by the threat type 'denial of service'. To the Table 14 and below, the interpretations for the threat types explained in Section 7.1.3 apply.

Here are some examples to explain our further changes:

- **Authentication:**
Cross at '*message replay*': Assuming that authentication is performed when a connection has been established, it is generally possible that an attacker records the messages sent by one communication partner when a connection has been established successfully and misuses them at a later point of time in order to successfully establish a connection including authentication.
- Cross at '*message alteration*': The ability to change intercepted messages is also a threat to authentication, since an attacker might be able, for example, to authenticate themselves again after a time stamp has been changed or to authenticate themselves as someone else thanks to the manipulation of the authentication data.
- **Confidentiality:**
Cross at '*compromising user credentials*': If an attacker is able to compromise user credentials, they might obtain access to confidential data via requests sent to a server. This means that compromising user credentials is a threat to the security objective 'confidentiality'.

Summary of the main findings:

ID	Observation	Recommendation
1	The definitions of the security objectives differ from the internationally recognised standard ISO/IEC 27000 [24] for the security objectives ' <i>Authentication, Availability, Confidentiality</i> ' and ' <i>Integrity</i> '	The definitions of the standard ISO/IEC 27000 should be used.
2	The security objective ' <i>Non-repudiation</i> ' is missing	The security objectives should be supplemented by the security objective ' <i>Non-repudiation</i> '.
3	The security objective ' <i>Authorization</i> ' is not defined precisely enough.	The definition should highlight that rights have to be granted according to the need-to-know principle.
4	The threat type 'repudiation' is not used in the OPC UA Specification.	Extension of the threat types by 'repudiation'
5	The threat type 'message flooding' is too narrow.	Extension of the threat type 'message flooding' to 'denial of service', since 'message flooding' is a subset of 'denial of service'
6	The definitions of the threat types do not correspond to the usual IT security definitions or are too vague	The definitions should be defined in more detail, for example as suggested in 7.1.3, and not only focus on the communication (even if they are only applied to the communication for OPC UA in the specification)
7	The assignment of security objectives versus threats is very much open to interpretation	The assignment of security objectives versus threats should be revised.

Table 14: Main findings of the review of security objectives and threat types

7.2 Threat analysis for the OPC UA protocol

7.2.1 Analysis according to the threats from Part 2

In principle, the analysis of the security objectives and threat types explained in Section 7.1 results in the threats listed in Table 16. Here, the following must be taken into consideration:

The analysis of the elements taken into account in Part 2, i.e. the security objectives and threat types, has shown that the focus of these elements is solely on securing the OPC UA communication. For the further analysis, we use the threats referred to in Table 14.

The threat modelling approach according to STRIDE is broader and ensures that threats which are not directed directly at the communication itself are also taken into consideration. Accordingly, threats to elements of the OPC UA infrastructure were included in Table 17. The protection of the communication and the protection of the infrastructure are of equal importance and complement each other. If an aspect is neglected, security as a whole is threatened. Accordingly, both must be integrated into the security concept.

For Table 16 and Table 17 below, the following rules are used for the CVSS assessment:

- AccessVector (AV): For most attacks, access to the local network segment (A) is needed in order to eavesdrop on or exchange messages with an OPC UA application. In some cases, local access to the system (L) is required.
- AccessComplexity (AC): Attacks which involve the sending of messages without further requirements were classified with a low level of criticality (L). Eavesdropping attacks were assessed with a medium level of criticality (M). Complex attacks requiring, for example, the cracking of cryptographic keys were assessed with a high level of criticality (C).
- Authentication (Au): Most attacks do not require authentication (N).
- ConfImpact (C): If the encryption is cracked or access to confidential data allowed, the parameter was assessed with highest level of criticality (C).
- IntegImpact (I): If the signature was cracked or messages spoofed by another OPC UA application, the parameter was assessed with the highest level of criticality (C).
- AvailImpact (A): All denial-of-service attacks were assessed with the highest level of criticality (C).
- Exploitability (E): All attacks were assessed as 'Unproven' (U), since they were only developed based on the analysis of the specification and could not have been tested yet.

Due to the constant exploitability on the lowest level (U), an attack may achieve the maximum score 8.5. Attacks which are an actual threat, i.e. for which C, I and A is not 'None' at the same time, achieve the minimum score 0.7. Score 0 means that there is no threat. Therefore, we suggest the following classification of the criticality on the basis of a traffic light colour system:

Low criticality	Medium criticality	High criticality
0.7 – 3.3	3.4 – 5.9	6.0 – 8.5

Table 15: CVSS criticality scores for OPC UA

Threat Type	ID	Threat	Brief Description	Security-Mode	CVSS Vector	CVSS Score
<i>Denial of service of an untrusted client</i>						
	1	HEL flooding	The client repeatedly sends HEL messages to the server. Response: The server responds with an ACK message in each case. Impact: high network load, low processor load	All	AV:A/AC:L/Au:N/C:N/I:N/A:C/E:U	5.2
	2	ACK or ERR flooding	The client repeatedly sends ACK or ERR messages to the server. Response: The server responds with an ERR message in each case. Impact: high network load, low processor load	All	AV:A/AC:L/Au:N/C:N/I:N/A:C/E:U	5.2
	3	HEL + OPN request flooding	The client repeatedly sends HEL and OPN messages to the server. Response: The server responds with ACK and ERR messages. Impact: - high network load, low processor load with securityMode None - high network load, medium processor load with securityMode Sign (verification of the signature) - high network load, high processor load with securityMode SignAndEncrypt (encryption and verification of the signature)	All	AV:A/AC:L/Au:N/C:N/I:N/A:C/E:U	5.2
	4	FindServers() or GetEndpoints() request flooding	The client establishes a SecureChannel with securityMode None and repeatedly sends FindServers() or GetEndpoints() requests to the server. Response: The server respond with a FindServers() or GetEndpoints() response in each case. Impact: high network load, low processor load	All	AV:A/AC:L/Au:N/C:N/I:N/A:C/E:U	5.2
	5	CLO request flooding	The client repeatedly sends CLO messages to the server. Response: The server responds with an ERR message in each case. Impact: high network load, low processor load	All	AV:A/AC:L/Au:N/C:N/I:N/A:C/E:U	5.2
	6	flooding consisting of incorrect messages	The client repeatedly sends incorrect messages to the server. Response: The server responds with an ERR message in each case. Impact: high network load, low processor load	All	AV:A/AC:L/Au:N/C:N/I:N/A:C/E:U	5.2
<i>Denial of service of a trusted client</i>						
	7	denial of service after	The Client establishes a session and sends arbitrary, complex	All	AV:A/AC:L/Au:M/C:	4.3

		a valid session has been established	requests to the server afterwards. Response: The server will process the requests one after the other. Impact: high network load, high processor load		N/I:N/A:C/E:U	
<i>Denial of service of an untrusted server</i>						
	8	OPN response flooding	The server responds to an OPN request with OPN response flooding. Response: The client rejects the messages after the requestId has been evaluated. Impact: - high network load, low processor load with securityMode None - high network load, medium processor load with securityMode Sign (verification of the signature) - high network load, high processor load with securityMode SignAndEncrypt (encryption and verification of the signature)	All	AV:A/AC:L/Au:N/C:N/I:N/A:C/E:U	5.2
<i>Further denial-of-service attacks (other than message flooding)</i>						
	9	Overload of the memory	An attacker can try to overload the memory of a server by using, for example, the maximum number of sessions, subscriptions, monitoredItems and Queues. Another possibility is to use a large number of variables with a variable length (string, array). Both attacks, however, require the successful establishment of sessions.	None	AV:A/AC:M/Au:N/C:N/I:N/A:C/E:U	4.8
	10	Filling the hard disk by writing	If the attacker causes the writing of very large amounts of data, for example by misusing the file transfer functionality (see [6] Annex C), it may be the case that the hard disk becomes full.	None	AV:A/AC:M/Au:N/C:N/I:N/A:C/E:U	4.8
<i>Eavesdropping</i>						
	11	Eavesdropping	If an attacker is able to eavesdrop on the communication between two OPC UA applications in the local subnetwork, they can intercept confidential information.	None, Sign	AV:A/AC:M/Au:N/C:P/I:N/A:N/E:U	2.5
	12	Eavesdropping	If an attacker, an insider for example, has physical access to the monitoring port of a switch, they can eavesdrop on the OPC UA communication in this manner.	None, Sign	AV:L/AC:L/Au:N/C:P/I:N/A:N/E:U	1.8
<i>Message spoofing</i>						
	13	Creating messages as	If there is no application authentication, an attacker can create	None	AV:A/AC:L/Au:N/C:P	4.1

		client	messages and pretend to be the client.		/I:P/A:N/E:U	
<i>Message alteration</i>						
	14	Manipulation of intercepted messages	If an attacker is able to intercept messages between a client and a server, they can manipulate them before they forward them to the actual recipient.	None	AV:A/AC:M/Au:N/C:C/I:C/A:N/E:U	6.2
	15	Suppression of intercepted messages	If an attacker is able to intercept messages between a client and a server, they can suppress some of these messages.	None	AV:A/AC:M/Au:N/C:N/I:C/A:N/E:U	4.8
<i>Message replay</i>						
	16	UA TCP messages	An attacker can send HEL and ACK messages again. Thus, the connection is interrupted and must be re-established.	All	AV:A/AC:M/Au:N/C:N/I:N/A:C/E:U	4.8
	17	UASC messages	UASC messages that have been sent again are identified immediately due to the incorrect sequenceNumber and rejected, and have thus an impact on the security.	All	AV:A/AC:M/Au:N/C:N/I:N/A:N/E:U	0
<i>Malformed message</i>						
	18	Malformed message	If an attacker has access to the local network, they can send malformed messages.	None	AV:A/AC:L/Au:N/C:N/I:C/A:N/E:U	5.2
<i>Server profiling</i>						
	19	UA TCP messages	An attacker can use HEL and ACK messages in order obtain information on the configuration of the server and possibly draw conclusions on the server from this.	All	AV:A/AC:L/Au:N/C:P/I:N/A:N/E:U	2.8
	20	FindServers() or GetEndpoints() requests	Since these Discovery services are always used in an unsigned and unencrypted form, an attacker can misuse them irrespective of the securityMode in order to obtain, for example, product information such as manufacturer and product names.	All	AV:A/AC:L/Au:N/C:P/I:N/A:N/E:U	2.8
	21	UASC messages	If there is no application authentication, an attacker can at least find further information on the server thanks to the establishment of a SecureChannel. If user authentication is not necessarily required to establish a session, information on the server can also be obtained thanks to this process.	None	AV:A/AC:L/Au:N/C:P/I:N/A:N/E:U	2.8
<i>Session hijacking</i>						
	22	Takeover of a SecureChannel or	If messages do not have to be signed, an attacker can try to capture the existing communication between a client and a	None	AV:A/AC:H/Au:N/C:P/I:C/A:P/E:U	4.9

		Session	server. For this purpose, they must guess different parameter values or find them out by eavesdropping.			
<i>Rogue server</i>						
	23	Operation of an untrusted server	If an attacker is able to implement their own OPC UA server, they might be able to intercept confidential information when the clients connect the server.	None	AV:A/AC:M/Au:N/C:P/I:P/A:N/E:U	3.7
	24	Takeover of a trusted server	If an attacker is able to take over the control of an existing server, they can influence the outgoing flow of information and read the incoming flow of information.	All	AV:A/AC:H/Au:N/C:P/I:C/A:N/E:U	4.5
<i>Compromising user credentials</i>						
	25	Brute-force attack to private keys during signing	If messages are signed, but not encrypted, an attacker can try to calculate private keys that are suitable for the ApplicationInstance certificates on the basis of the contents recorded.	Sign	AV:A/AC:H/Au:N/C:N/I:C/A:N/E:U	3.9
	26	Brute-force attack to private keys	If messages are encrypted, an attacker can try to calculate private keys that are suitable for the ApplicationInstance certificates on the basis of the contents recorded.	SignAnd-Encrypt	AV:A/AC:H/Au:N/C:C/I:C/A:N/E:U	5.3
	27	Brute-force attack to symmetric keys	If messages are signed, an attacker can try to calculate the signature keys derived from the client and server nonces on the basis of the contents recorded.	Sign	AV:A/AC:H/Au:N/C:N/I:C/A:N/E:U	3.9
	28	Brute-force attack to symmetric encryption keys	If messages are encrypted, an attacker can try to calculate the encryption keys derived from the client and server nonces on the basis of the contents recorded. Secret-based userIdentityTokens are then also considered cracked; certificate-based userIdentityTokens are not affected by this.	SignAnd-Encrypt	AV:A/AC:H/Au:N/C:C/I:N/A:N/E:U	3.9
	29	Brute-force attack to private keys of certificate-based userIdentityTokens	If messages are not encrypted, an attacker can try to calculate private keys that are suitable for the certificate-based userIdentityTokens on the basis of the contents recorded.	None, Sign	AV:A/AC:H/Au:N/C:C/I:N/A:N/E:U	3.9

	30	Dictionary attack to secret-based userIdentityTokens	An insider with access to a trusted client can try out any number of passwords to authenticate a user with more rights than they have themselves.	All	AV:L/AC:L/Au:S/C:C/I:N/A:N/E:U	3.9
<i>Repudiation</i>						
	31	Disabling the audit functionality	If an attacker has access to the configuration of an application, they can disable the auditing for example so that their subsequent malicious actions are not recorded at least on the one side.	None	AV:L/AC:L/Au:N/C:N/I:P/A:N/E:U	1.8

Table 16: Potential threats for the OPC UA communication

7.2.2 Analysis of the threats for elements of the OPC UA infrastructure

Threat Type	ID	Threat	Brief Description	Security-Mode	CVSS Vector	CVSS Score
<i>Attacks to the OPC UA infrastructure</i>						
	32	Reading the private keys	If an attacker can obtain access to the file system of a device on which an OPC UA application runs, they might be able to read private keys stored there.	All	AV:L/AC:L/Au:N/C:C/I:C/A:N/E:U	5.6
	33	Manipulation of trust or issuer lists	If an attacker can obtain access to the file system of a device on which an OPC UA application runs, they might be able to manipulate the trust or issuer lists stored there so that a certificate created by an attacker is considered trusted during the testing.	Sign, SignAnd-Encrypt	AV:L/AC:L/Au:N/C:C/I:C/A:N/E:U	5.6
	34	Deletion of trust or issuer lists	If an attacker can obtain access to the file system of a device on which an OPC UA application runs, they might be able to delete or empty the trust or issuer lists stored there so that no certificates at all are considered trusted during the testing.	Sign, SignAnd-Encrypt	AV:L/AC:L/Au:N/C:N/I:N/A:C/E:U	4.2
	35	Manipulation of configuration files	If an attacker can obtain access to the file system of a device on which an OPC UA application runs, they might be able to manipulate security-relevant parameters in configuration files stored there, such as changing the path to the CertificateStore.	All	AV:L/AC:L/Au:N/C:C/I:C/A:N/E:U	5.6
	36	Manipulation or deletion of audit data	If an attacker can obtain access to a system on which the OPC UA audit data is stored, they might be able to delete or manipulate entries in order to cover up their activities.	All	AV:L/AC:L/Au:N/C:N/I:C/A:N/E:U	4.2
	37	Manipulation of the time	If an attacker can manipulate the time of a system on which an OPC UA application runs, audit data created by this application is falsified.	All	AV:A/AC:L/Au:N/C:N/I:P/A:N/E:U	2.8
	38	Unavailability of the CRLs	If the retrieval of current CRLs is mandatory during the testing of certificates, an attacker can exploit this by interrupting the availability of the CRLs.	Sign, SignAnd-Encrypt	AV:A/AC:M/Au:N/C:N/I:N/A:C/E:U	4.8
	39	Attacks on the operating system	If an attacker gains control of the operating system on which an OPC UA application runs, they can, among other things,	All	AV:A/AC:M/Au:N/C:C/I:C/A:C/E:U	6.7

			terminate the application, read the memory etc.			
	40	Attacks on the implementation of cryptographic algorithms or on random number generators	If vulnerabilities in the cryptographic library which an OPC UA application uses are known or if the entropy for the generation of random numbers is insufficient, an attacker might obtain access to protected data.	Sign, SignAnd-Encrypt	AV:A/AC:M/Au:N/C:C/I:N/A:N/E:U	4.8
	41	Manipulation of access rights in the address space	If an attacker succeeds in manipulating access rights to the server's address space, the attacker can read confidential data or manipulate data.	All	AV:A/AC:M/Au:N/C:P/I:P/A:N/E:U	3.7

Table 17: Potential threats for the OPC UA infrastructure

7.2.3 Results

The following Table only shows the threats considered critical and which countermeasures can be taken as protection against the respective threat.

ID	Threat	Countermeasures
1	Manipulation of intercepted messages: If an attacker is able to intercept messages between a client and a server, they can manipulate them before they forward them to the actual recipient.	securityMode: securityMode None should be deactivated completely to prevent subsequent mis-configurations or downgrade attacks from being carried out. If the confidentiality of the exchange of information taking place via OPC UA is important, SignAndEncrypt should be chosen. Selection of cryptographic algorithms: As securityPolicyUri, the one which is most secure at the moment, i.e. Basic256Sha256, is recommended.
2	Attacks to the operating system If an attacker gains control of the operating system on which an OPC UA application runs, they can, among other things, terminate the application, read the memory etc.	Hardening of the operating system: If an attacker is able to access parts of the memory (threat 39) in which an OPC UA application runs, they might obtain access to otherwise secret information such as the private key or overwrite authorisations. Accordingly, the security of the underlying system is also important. This can be improved significantly by taking hardening measures.

Table 18: Particularly critical threats

7.3 Analysis of the protection mechanisms on the parameter level

7.3.1 Explanation of the analysis table

Based on the explanations in [3] Section 5.2 and our supplement regarding non-repudiation, OPC UA seems to provide mechanisms as protection against all threat types listed in Table 14. This was tested up to the parameter level on the basis of a detailed analysis. Due to their extent, the results of this analysis of the OPC UA security based on the parameters contained in the exchanged messages are presented in a file that can be downloaded separately [25].

The document is structured as follows:

Which protection OPC UA offers crucially depends on the securityMode parameter. Therefore, the results are presented in three tables for the sake of clarity: Using the tabs, the results for securityMode 'None', 'Sign' and 'SignAndEncrypt' can be displayed. The fourth tab summarises the results of all modes and is explained in more detail below.

The first three tables have the same structure: In the form of a tree structure that can be opened and closed on the left side, the lines represent how the messages are structured in OPC UA TCP and UASC (UA Secure Conversation). The messages which are crucial for the communication and thus play an important role in IT security were chosen, i.e. finding the communication partners (Discovery service) and establishing the connection (SecureChannel and Session): In OPC UA TCP, there are three possible message types (HEL, ACK and ERR). In UASC, the messages for the SecureChannel, Session and the Discovery services were taken into account.

The columns represent two types of information: The columns on the left side (A to L) are used to describe the message parameters, whereas the columns on the right side (M to X) show the results regarding the evaluation of the security.

Below, a detailed breakdown of the column contents can be found:

- The columns A to F represent the different levels of the protocols, message parts and parameters.
- In column G, the data types of the parameters are specified and column H shows the corresponding default values if available.
- By using colours, the columns I and J highlight which parts of the respective messages are signed or encrypted:
- Column I: grey = unsigned blue = signed
- Column J: grey = unencrypted green = encrypted
- Moreover, inside the surfaces it is shown with which key the operation was carried out.
- The columns K and L contain explanations for a better understanding of the parameters.
- In the columns M to W, the threats from Part 2, supplemented by 'repudiation', are listed. The last column makes it possible to explain the entries in the columns with the threats.

The results of the analysis are entered in the table as follows: If a parameter supports the protection against a specific threat, '1' is entered in the respective line (parameter) and column (threat). Example: The SecureChannelId contributes to the protection against the re-use of messages. Therefore, '1' has been entered into cell Q44.

In summary, the entries for the individual threats on the message level are added up. If this results in a number greater than 0, this means that this message type provides protection against this threat. This addition of the entries per threat was also continued on the service and protocol level. The results on the service level were transferred to the last tab in the table.

The interpretation as to whether a parameter increases the protection against a threat is defined in relatively broad terms: In particular, this does not mean that one parameter alone is sufficient as protection against the respective threat, but only that it contributes to such protection. Furthermore, the fact that several parameters contribute to the protection against a threat does not automatically mean that the threat is thus fended off completely.

Examples:


- All three parameters SecureChannelId (cell Q44), SequenceNumber (cell Q53) and authenticationToken (cell Q57) contribute, irrespectively of the securityMode, to the fact that replay attacks cannot be performed by sending a message again without making adjustments.
- In the event of a flooding attack, the target of an attacker may not only be generating a high network load, but also generating a high processor load. In the securityMode 'SignAndEncrypt', most messages must be encrypted first and their signatures verified afterwards. A high message volume also results in a considerable increase in the processor load.
- The parameter SecureChannelId alone does provide some protection against such flooding attacks (cell M44): This is due to the fact that the parameter is evaluated even before a message is encrypted. If the ID does not match an existing SecureChannel, the message will not be evaluated any further. This limits the increase in the processor load, but the network load cannot be avoided with this parameter.
- The situation is similar when negotiating the buffer and message sizes and the maximum number of partial messages in OPC UA TCP 'HEL' (cells M10 to M13) and 'ACK' (cells M23 to M26) messages: They prevent an attacker from sending partial messages in any size and number, but they do not provide comprehensive protection against flooding attacks.


7.3.2 Detailed explanation of the results of the analysis table


Adding the numbers sorted according to the securityMode on the service level results in the following table:


security-Mode	Layer or Service	Denial of Service	Eavesdropping	Message Spoofing	Message Alteration	Message Replay	Malformed Messages	Server Profiling	Session Hijacking	Rogue Server	Compromising User credentials	Reputation
OPC UA TCP		8	0	0	0	0	8	0	0	0	0	0
None												
SecureChannel		17	0	0	0	23	1	0	22	0	0	0
Session		14	0	2	0	26	3	4	23	0	2	2
Discovery		12	0	2	2	21	5	4	18	3	0	3
Sign												
SecureChannel		17	9	15	15	32	16	26	37	11	13	17
Session		14	0	12	8	31	12	14	28	6	4	18
Discovery		13	0	3	3	22	5	12	19	4	1	6
SignAndEncrypt												
SecureChannel		17	18	15	15	32	16	26	40	11	18	17
Session		14	18	12	8	31	12	14	46	6	22	18
Discovery		13	7	3	3	22	5	12	25	4	7	6

Table 19: Effectiveness of the OPC UA measures

: no protection

: low protection

: Protection which restricts the possibilities of an attacker, but does not prevent this type of attack

: effective protection (attacks of this type require cryptographic attacks)

It is important to stress that the numbers were only used as an aid for the interpretation: Only one thing, however is clear: if the number 0 is entered for one threat for the three services in one securityMode, there can be no protection. However, a quantitative evaluation is not possible: Higher scores do not necessarily refer to better protection.

OPC UA TCP:

Since the parameter `securityMode` is part of UASC, it does not have any impact on the underlying layer OPC UA TCP.

denial of service: As it has already been described in the last example, OPC UA TCP provides only minimum protection against flooding attacks.

malformed message: Defining upper limits for buffer and message sizes and the maximum number of partial messages also provides protection against incorrect messages in a very restricted manner, since a message of an impermissible size, for example, is rejected at an early stage.

server profiling: Since the number of different OPC UA SDKs available worldwide is relatively small, it is not excluded that the information gained on the OPC UA TCP level alone could be sufficient to create a clear server profile.

securityMode None:

denial of service: The precautions against flooding attacks are the same for the parameters as for the two other modes, but with different impacts: On the one hand, it is more difficult for an attacker to force the victim to carry out complex computer operations, since there is no signing nor encryption. On the other, an attacker has more options of overloading the memory and filling the hard disk by writing due to the fact that there is no application authentication.

eavesdropping: Since, with the exception of secret-based `userIdentityTokens`, there is no encryption in the `securityMode None`, the security objective 'confidentiality' is not complied with.

message spoofing, message alteration: `securityMode None` does not provide any protection at all against the creation and alteration of messages, such as in the event of a man-in-the-middle attack, due to the lack of application authentication.

message replay and session hijacking: Adequate mechanisms, such as verifying the `sequenceNumber` provide protection against messages being sent again and the takeover of sessions. This requires, however, that the attacker does not eavesdrop additionally.

malformed message: Due to the missing application authentication, an attacker has many options of creating incorrect messages and having them evaluated by the victim. User authentication alone, if absolutely necessary, prevents a session from being established.

server profiling: The protection against the creation of a server profile is limited to the fact that user authentication, if absolutely necessary, prevents a session from being established.

rogue server: Application authentication for the RegisterServer Discovery service prescribed in Part 7 Table 3 prevents an attacker from registering their unauthorised server with Discovery servers and thus from being found via FindServers of clients in this manner. In [13], however, it is also possible that a rogue server introduces itself via mDNS.

Due to the lack of application authentication, a Client is then no longer able to distinguish an untrusted server from a trusted server.

compromising user credentials: If the `securityPolicyUri` or a `userTokenPolicy` is set as required by the specification, i.e. not 'None', the security of a secret-based `userIdentityToken` is ensured for user authentication.

repudiation: Thus, non-repudiation regarding user authentication is also provided. Non-repudiation regarding the application cannot be complied with, since there is no such authentication.

securityMode Sign and SignAndEncrypt:

eavesdropping: As expected, the two modes basically differ in the protection of confidentiality: In the `securityMode Sign`, the OpenSecureChannel requests and responses themselves are encrypted, but the further ex-

change of information is performed in unencrypted form. In particular, the contents of all requests and responses which are exchanged within an existing session can be read by an eavesdropping attacker, because they are sent in plaintext. Only secret-based userIdentityTokens which are sent for the authentication of the users in case of activateSession requests are encrypted and thus protected.

With SignAndEncrypt, however, all messages are encrypted for SecureChannel and the Session. Only for the Discovery service, all applications, irrespective of the securityMode set otherwise, must support FindServers and GetEndpoints requests and responses even without security, i.e. unencrypted and unsigned.

denial of service: Certain parameters restrict the possibilities of an attacker in the case of flooding attacks. Forced application authentication in particular makes it considerably more difficult to perform a flooding attack on the session level and/or makes it impossible without knowing further information such as private keys. Especially due to the measures involving complex decryption and signature verification steps, the attacker has nevertheless the possibility of also putting a considerably stronger load on the processor, as is the case with securityMode None, in addition to a high network load. This means that UASC only offers inadequate protection against flooding attacks.

This vulnerability, however, must be qualified, since an attacker would probably achieve comparable results with significantly less effort by flooding on the IP or TCP protocol level.

server profiling: Due to the required application authentication, an attacker has less possibilities of gaining information via a SecureChannel, whereas failed authentication might also disclose information about the server. With the securityModes, an attacker cannot establish sessions, but FindServers and GetEndpoints are also possible without authentication.

The two modes offer adequate protection against all other threats, with slightly better protection when SignAndEncrypt is chosen. If confidentiality does not play a role, securityMode Sign is probably sufficient.

7.3.3 Conclusion

securityMode None provides little to no protection against IT security attacks and should be avoided in all cases. If confidentiality does not play a role, securityMode Sign offers adequate protection against the threats examined. If confidential data is exchanged, securityMode SignAndEncrypt is required.

The threats 'denial of service' and 'server profiling' can only be reduced using OPC UA protection mechanisms, but not fended off completely. Accordingly, additional measures outside the OPC UA infrastructure must contribute to the protection against these threats.

Moreover, the analysis has shown that all threats except for 'denial of service' and 'server profiling' can be counteracted adequately with the securityMode SignAndEncrypt. Thus, the security objectives are complied with except for 'availability'.

7.4 Further observations regarding the design

From our point of view, the following design decisions when drawing up the protocol make it more difficult to achieve a high level of security:

ID	Observation	Recommendation
1	Poor flexibility in the profile concept: At the moment, the specification requires that all permissible combinations of asymmetric and symmetric signature and encryption algorithms including key lengths are listed as profile in Part 7. However, new versions of the OPC UA standard are only published approx. every three years. This means that it is not possible to make	It should be considered how necessary adjustments regarding the cryptography can be integrated into the specification in a timely manner.

	any adjustments regarding the cryptography within this period of time: Neither new, more secure and/or more efficient algorithms or key lengths can be included nor algorithms or key lengths considered insecure removed.	
2	<p>Lack of forward secrecy:</p> <p>The current procedure used for the generation of the symmetric keys does not support forward secrecy. If an attacker is able to find out the client and server nonces that are exchanged when a SecureChannel is established, they can also subsequently decrypt the messages which were exchanged via this SecureChannel. It is not clear why procedures and practices were not used, which have been successful for years and support forward secrecy, such as DHE-RSA or DHE-DSS which are based on the Diffie-Hellman key exchange.</p>	Good key exchange practices should be required.
3	<p>Introduction of mDNS:</p> <p>Part 12 Section 4.3.5 describes how mDNS is used for the Discovery service. The examination of mDNS security is outside the scope of this project, but can bear additional risks.</p>	The IT security of mDNS should be examined.

Table 20: Main findings regarding the design

However, some points of criticism can be qualified due to the current field of application of OPC UA: forward secrecy, for instance, is only relevant if strictly confidential data which must still remain protected over longer periods of time is exchanged. In our opinion, this is not the case in the field of automation, in which OPC UA is widely used. It cannot be excluded, however, that the protocol will be used increasingly in the future in other areas to which such requirements apply.

7.5 Summary of the overall analysis results

The specification analysis performed has shown that OPC UA offers a high level of security if security-Mode Sign and, above all, securityMode SignAndEncrypt is used. No systematic errors could be detected. This has become apparent particularly from the analysis of the protection mechanisms on the parameter level in Section 7.3.

Even just fending off 'denial of service' attacks is by nature difficult. This problem, however, is not specific to OPC UA, but affects network protocols in general. 'denial of service' attacks can only be counteracted by an adequate IT security architecture.

As explained in Section 5, 6, 7.1.5 and 7.4, there is potential for improvement in the specification, which has been summarised again in Table 21.

ID	Observation	Recommendation
Chapter 5, inventory		
1	The name "Secure Channel" suggests IT security which, however, is not included in the securityMode None. Unfortunately, the name cannot be changed any more.	None
2	For random number generators, no notes regarding minimum requirements are provided in the OPC UA Specification.	Notes regarding minimum requirements of recognised institutions (BSI, for example) in the OPC UA Specification should be specified for random number generators.
3	There are no notes regarding outdated algorithms or algorithms considered insecure, especially SHA-1 and cipher suites.	Notes regarding outdated algorithms or algorithms considered insecure, especially SHA-1, and cipher suites, should be provided, e.g. links to recognised institutions (e.g. BSI) which give recommendations for cryptographic algorithms at regular intervals.
4	The private key is used in OPC UA both for the signature and for the encryption.	Different key pairs for the signature and encryption should be requested in the OPC UA Specification.
5	At the moment, it is not possible to also use algorithms based on elliptic curves.	OPC UA should provide for this option in future in order to also allow for adequate security on devices with low processing power.

ID	Observation	Recommendation
Chapter 6, editorial analysis		
6	Inconsistencies between Part 4 and 6: Both Part 4 and Part 6 particularly describe those services which are crucial for the secure establishment of the connection on the basis of a SecureChannel and a Session. Part 4, however, is quite general, whereas Part 6 addresses the particularities of the available protocols. This is why the parameters which are contained in the different parts of a message partially differ. The assignment of some data types which used to define the parameters is also different. This is	The mentioned inconsistencies between Part 4 and Part 6 should be eliminated.

	confusing and prone to errors.	
7	<p>Uncertainties in the case of parameters which are not used:</p> <p>Due to the possibility of switching on or off signing and encryption via the securityMode, parameters which are not required are also transmitted in a message in certain cases. It is not clear, however, which values are to be used for the parameters affected in these cases. This can result in implementation errors and incompatibilities between OPC UA applications of different manufacturers.</p>	For the parameters of a message which are not required, the parameter values to be set should also be defined.
8	<p>Missing list of reasonable default values:</p> <p>The configuration of an OPC UA application includes a number of parameters which determine lower or upper limits, e.g. for buffer sizes or number of messages. In most cases, however, default value information or a recommendation regarding the range of reasonable values is missing. It would make the work of developers and administrators significantly easier if a list of such default values would be available in the specification. A distinction could be made between two device classes: embedded devices with limited power on the one hand and PCs or workstations on the other.</p>	The default values of parameters should always be determined. A list of such default values should be created and a distinction made between two device classes: embedded devices with limited power on the one hand and PCs or workstations on the other.
9	<p>Better protection against attacks to secretbased userIdentityTokens:</p> <p>At the moment, the specification does not define how to proceed in the case of repeatedly failed user authentication.</p>	In order to increase the security further, one suggestion would be that servers allow the setting of doubling the period of time up until the processing of the next authentication for each failed login and stopping this process, for example, when one minute has been reached. Upon successful login, this limit is reset to one second.
10	At the moment, the following setting combination is allowed: securityMode 'Sign' or 'SignAndEncrypt' and securityPolicyUri 'None'. This results in insecure communication.	It is recommended that securityPolicyUri 'None' be prohibited in the securityMode 'Sign' and 'SignAndEncrypt' and that this validation of the configuration also be added as conformity test.
11	<p>Part 4, p. 154, Chap. 7.35.1</p> <p>For secret-based userIdentityTokens, encryption is required. There is no such requirement for signature-based userIdentityTokens.</p>	It should be mandatory ("shall") that a userTokenPolicy or the securityPolicyUri is not 'None' at all times to ensure that an algorithm is available for creating the signature.
12	<p>Part 6, S. 7, Chap. 5.1.6, Section 3</p> <p>According to the specification, variant arrays can be nested. The same applies to diagnosticInfo.</p>	A reasonable maximum permissible recursion depth should be specified, e.g. 10.
13	<p>Part 6, p. 39, Chap. 6.7.2</p> <p>What happens if a gap is detected for the sequenceNumber?</p>	It should be described explicitly how an OPC UA application should respond to such a case (error message, loss of connection etc.).
14	<p>Part 6, p. 41, Chap. 6.7.4</p> <p>OpenSecureChannel requests and responses are also encrypted in the securityMode 'Sign'.</p>	This should be mentioned explicitly.
15	Part 6, p. 67, Chap. D6, Table D6	It should be pointed out explicitly that some of

	Table D6 provides options which are an insecure configuration of the communication.	these options are insecure.
Chapter 7.1.5, analysis of security objectives and threat types		
16	The definitions of the security objectives differ from the internationally recognised standard ISO/IEC 27000 [24] for the security objectives 'Authentication, Availability, Confidentiality' and 'Integrity'	The definitions of the standard ISO/IEC 27000 should be used.
17	The security objective 'Non-repudiation' is missing	The security objectives should be supplemented by the security objective 'Non-repudiation'.
18	The security objective 'Authorization' is not defined precisely enough.	The definition should highlight that rights have to be granted according to the need-to-know principle.
19	The threat type 'repudiation' is not used in the OPC UA Specification.	Extension of the threat types by 'repudiation'
20	The threat type 'message flooding' is too narrow.	Extension of the threat type 'message flooding' to 'denial of service', since 'message flooding' is a subset of 'denial of service'
21	The definitions of the threat types do not correspond to the usual IT security definitions or are too vague	The definitions should be defined in more detail, for example, as suggested in 7.1.3 and not only focus on the communication (even if they are only applied to the communication for OPC UA in the specification)
22	The assignment of security objectives versus threats is very much open to interpretation	The assignment of security objectives versus threats should be revised.
Chapter 7.4, observations regarding the design		
23	Poor flexibility in the profile concept: At the moment, the specification requires that all permissible combinations of asymmetric and symmetric signature and encryption algorithms including key lengths be listed as profile in Part 7. However, new versions of the OPC UA standard are only published approx. every three years. This means that it is not possible to make any adjustments regarding the cryptography within this period of time: Neither new, more secure and/or more efficient algorithms or key lengths can be included nor algorithms or key lengths considered insecure removed.	It should be considered how necessary adjustments regarding the cryptography can be integrated into the specification in a timely manner.
24	Lack of forward secrecy: The current procedure used for the generation of the symmetric keys does not support forward secrecy. If an attacker is able to find out the client and server nonces that are exchanged when a SecureChannel is established, they can also subsequently decrypt the messages which were exchanged via this SecureChannel. It is not clear why procedures and practices which have been successful for years and support forward secrecy, such as DHE-RSA or DHE-DSS	Good key exchange practices should be required.

	which are based on the Diffie-Hellman key exchange, were not used.	
25	Introduction of mDNS: Part 12 Section 4.3.5 describes how mDNS is used for the Discovery service. The examination of mDNS security is outside the scope of this project, but can bear additional risks.	The IT security of mDNS should be examined.

Table 21: Main findings of the overall analysis

8 Results of the reference implementation analysis

This chapter describes the tests, test results and, wherever possible, the exploitation of identified vulnerabilities.

8.1 Description of the OPC UA communication stack tested

The reference implementation of the OPC UA communication stack of the OPC Foundation in ANSI C in the version 1.02.344.5 was analysed. Several reference implementations of the OPC Foundation in different programming languages are available. The decision in favour of the ANSI C implementation was taken by the BSI. The version 1.02.344.5 was the latest version available at the beginning of the project and also corresponds to the version 1.02 of the specification which was analysed within the framework of this project.

Upon consultation with the BSI, it was decided to integrate the stack into a 32-bit single-threaded UA server frame application and to test it in a Linux environment. The integration into a UA server was necessary in order to be able to perform the tests described in Section 8.3. Due to the tools used and for licence reasons, Linux was favoured as test platform. The 32-bit version was preferred the 64-bit alternative, since both versions are equivalent from an IT security perspective, but 32-bit applications are used more widely.

The ANSI C stack can be compiled and operated in the single-threaded or in the multi-threaded mode. In this study, the object under examination was operated in the single-threaded mode in a single-threaded framework application in order to reduce the complexity. Moreover, the focus of the analysis was the security functions and their logic as well as the encoders and decoders and not the locking and the parallelism of the frame application.

8.2 Attacks to vulnerabilities identified in the specification analysis

When analysing the specification, no systematic errors were found (see Section 7.5). Therefore, no attacks to vulnerabilities/systematic errors are described here.

8.3 Procedure

The analysis of the reference implementation is based on the following tests of the OPC UA stack described in Section 8.1:

- Certificate tests
- Static code analysis
- Fuzzing
- Dynamic code analysis

On the one hand, these tests are used to verify to what extent the security measures from the specification have been implemented completely and correctly in the stack. On the other, the code quality was examined by means of tools.

All tests requiring a running environment were operated with an OPC UA server. The client side of the communication stack was not examined except for the static code analysis.

Certificate tests:

In the certificate tests, it is checked if the server responds in different PKI configurations according to the specification. The decisive factor here is the compliance with the steps described in Part 4 Section 6.1.3 of the OPC UA Specification.

In these tests, connection attempts of a client to a server with certain, specifically configured PKI environments are simulated: For example, the client tries to connect to an expired ApplicationInstanceCertificate, a CRL is not available or the root certificate is not available etc.

Both positive tests which should result in a successful establishment of the connection and negative tests which should lead to an error message of the server are run.

The individual tests are described in detail in Annex A.

Static code analysis:

The static code analysis, using the *cppcheck* tool, makes it possible to examine the code quality of the source code. For example, parameter handling by functions is checked.

The source code of the OPC Foundation communication stack contains both the client and the server side of the communication. Thus, the client and server code is checked by the static code analysis.

Fuzzing:

As part of this project, it was tested, by means of fuzzing, how the server responds to a wide variety of mostly invalid messages. For this purpose, a large number of test iterations with variable message contents were performed automatically. To implement these tests, the fuzzing framework *Peach* was used.

In the fuzzing framework, tests for the different securityModes of the OPC Foundation stack None, Sign and SignAndEncrypt were implemented. For the securityModes Sign and SignAndEncrypt, an external application was developed (*secucon*) which allows signing and encrypting messages created in *Peach*.

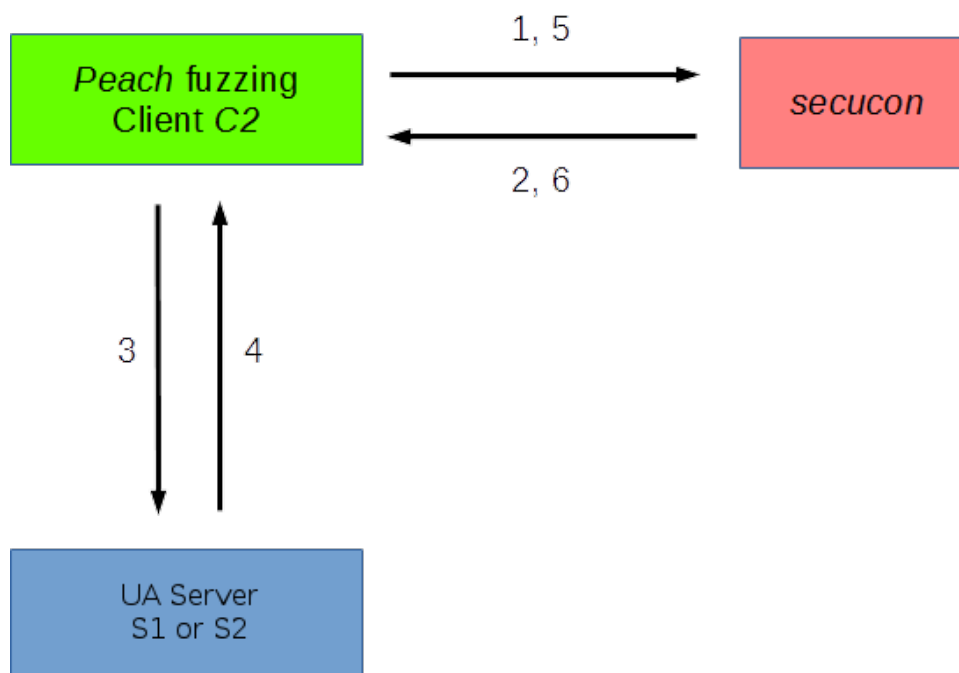


Figure 3: Communication channels with securityMode Sign and SignAndEncrypt

Explanations of the arrows in Figure 3:

- 1: *Peach* sends a fuzzed message to *secucon*
- 2: *secucon* sends this message back to *Peach*, signed and, if necessary, encrypted
- 3: *Peach* forwards the message to the server

4: The server responds with a signed and, if necessary, encrypted message

5: *Peach* forwards the response of server to *secucon*

6: *secucon* sends the response of the server, decrypted and without a signature, to *Peach*

Dynamic code analysis:

Similarly to the static code analysis, the dynamic code analysis is used to identify errors of certain programming error classes. Whereas the software to be tested is not run in the static code analysis, the dynamic code analysis requires that the code of the OPC Foundation communication stack be run. In order to be able to detect as many errors as possible in the code, the highest possible code coverage was aimed at when performing the dynamic code analysis.

Details regarding the code coverage are described in Section 8.8.

8.4 Certificate tests

8.4.1 Description of the tests

The goal of the certificate tests is to verify the PKI functionality described in the specification. In this respect, especially Part 4 Section 6.1.3 of the OPC UA Specification and Table 101 presented there are relevant: This table describes which steps are required to test the validity and trustworthiness of a certificate and in which order they are to be carried out. In the following table, the ten validation steps and their description, as they can be found in Part 4 Table 101, are listed as reference. The column on the right side provides information whether errors can be suppressed in the respective validation step. If such suppression is enabled, for example, for the *validity period*, the validation of a certificate is considered successful even if an error is triggered in this step.

Validation step	Description	Can the error be suppressed?
Certificate Structure	The certificate structure is verified.	no
Validity Period	The current time shall be after the start of the validity period and before the end.	yes
Host Name	The HostName in the URL used to connect to the Server shall be the same as one of the HostNames specified in the Certificate.	yes
URI	Application and Software Certificates contain an application or product URI that shall match the URI specified in the ApplicationDescription provided with the Certificate. This check is skipped for CA Certificates.	no
Certificate Usage	Each Certificate has a set of uses for the Certificate (see Part 6). These uses shall match use requested for the Certificate (i.e. Application, Software or CA).	yes
Trust List Check	No further checks are required if the Certificate is in the Trust List. The Administrator shall completely validate any Certificate before placing it in the Trust List.	--
Find Issuer Certificate	A Certificate cannot be trusted if the Issuer Certificate is unknown. A self-signed Certificate is its own issuer.	--
Signature	A Certificate with an invalid signature shall always be rejected.	--
Find Revocation List	Each CA Certificate may have a revocation list. This check fails if this list is not available (i.e. a network interruption prevents the Application from accessing the list). No error is	yes

	reported if the Administrator disables revocation checks for a CA Certificate.	
Revocation Check	The Certificate has been revoked and may not be used.	no

Table 22: Certificate validation steps (source: [5] Table 101)

Based on this table, different parameters which play a role in the validation of certificates can be derived:

- Structure of the PKIs: Is a self-signed certificate, a simple PKI with one CA or a multi-level PKI with one root CA and sub CAs referred to?
- Validity of individual certificates: Are they valid, have they expired, are they not valid yet, have they been provided with an incorrect signature and/or revoked? Does the configuration provide for the option that the temporal validity (expired, not valid yet) is ignored?
- Verification of the certificate chain: Are the links of the certificate chain available in the OPC UA PKI store or are they sent together with the message? Is the certificate chain complete? Is the self-signed or the CA certificate trusted?
- CRLs: Are CRLs used? If CRLs are used, how are missing CRLs dealt with?

In Table 34 in Annex A, the 92 certificate tests performed can be found, in which different combinations of the parameters mentioned here are tested.

In order to carry out the tests, the *certcheck* tool which was made available by Unified Automation on a loan basis was used. With this, it can be tested how a server responds to differently configured client requests.

In this project, a folder structure in the file system was used as OPC UA PKI store under Linux.

8.4.2 Analysis of the results

The test results can be summarised as follows:

- 53 tests were completed successfully: The server sent the expected response.
- 39 tests failed. Here, the tests can be divided into three categories:
 1. The StatusCode in the response of the server deviates from the expected code (12 tests). This means that the implementation of the stack deviates from the specification. This deviation, however, does not have great relevance from the IT security point of view and is therefore not examined any further.
 2. The server erroneously rejects the certificate of the client and thus also a legitimate establishment of the connection (8 tests). This means that the availability of the connection is impaired in these cases.
 3. The server erroneously classifies the client certificate as valid and allows the connection to the client to be established (19 tests). Thus, the security of the authentication mechanism is bypassed here.

In the following table, all failed tests are listed and sorted according the categories described above:

TestNo.	TestName	Expected StatusCode	Actual StatusCode
<i>Incorrect StatusCode</i>			
3	SelfSigned3TrustedSignature_Bad	BadSecurityChecks-Failed	BadCertificateUntrusted
4	SelfSigned4Signature_Bad	BadSecurityChecks-Failed	BadCertificateUntrusted
6	SelfSigned6NotYet_Bad	BadCertificateTime-Invalid	BadCertificateUntrusted
10	SelfSigned10Expired_Bad	BadCertificateTime-Invalid	BadCertificateUntrusted
47	RootSigned37NotYet_CaIssuerCrl_IgnInvTime_Bad	BadCertificateUntrusted	BadCertificateTime-Invalid
48	RootSigned38Expired_CaIssuerCrl_IgnInvTime_Bad	BadCertificateUntrusted	BadCertificateTime-Invalid

83	SecondarySigned35Trusted_CaIssuersCrl_ScaIssuersNotYetCrl_Bad	BadCertificateIssuer-TimeInvalid	BadCertificateTime-Invalid
85	SecondarySigned37_CaIssuersCrl_ScaIssuersNotYet_Bad	BadCertificateIssuer-TimeInvalid	BadCertificateTime-Invalid
86	SecondarySigned38_CaIssuersCrl_ScaIssuersNotYet_IgnInvTime_Bad	BadCertificateUntrusted	BadCertificateTime-Invalid
87	SecondarySigned39Trusted_CaIssuersCrl_ScaIssuersExpiredCrl_Bad	BadCertificateIssuer-TimeInvalid	BadCertificateTime-Invalid
89	SecondarySigned41_CaIssuersCrl_ScaIssuersExpired_Bad	BadCertificateIssuer-TimeInvalid	BadCertificateTime-Invalid
90	SecondarySigned42_CaIssuersCrl_ScaIssuersExpired_IgnInvTime_Bad	BadCertificateUntrusted	BadCertificateTime-Invalid
<i>Connection rejected</i>			
7	SelfSigned7TrustedNotYet_IgnInvTime_Good	Good	BadCertificateTime-Invalid
11	SelfSigned11TrustedExpired_IgnInvTime_Good	Good	BadCertificateTime-Invalid
41	RootSigned31TrustedNotYet_CaTrustedCrl_IgnInvTime_Good	Good	BadCertificateTime-Invalid
42	RootSigned32NotYet_CaTrustedCrl_IgnInvTime_Good	Good	BadCertificateTime-Invalid
45	RootSigned35TrustedExpired_CaTrustedCrl_IgnInvTime_Good	Good	BadCertificateTime-Invalid
46	RootSigned36Expired_CaTrustedCrl_IgnInvTime_Good	Good	BadCertificateTime-Invalid
84	SecondarySigned36Trusted_CaIssuersCrl_ScaIssuersNotYetCrl_IgnInvTime_Good	Good	BadCertificateTime-Invalid
88	SecondarySigned40Trusted_CaIssuersCrl_ScaIssuersExpiredCrl_IgnInvTime_Good	Good	BadCertificateTime-Invalid
<i>Connection granted</i>			
15	RootSigned3Trusted_CaIssuers_Bad	BadCertificate-RevocationUnknown	Good
16	RootSigned4TrustedRevoked_CaIssuersCrl_Bad	BadCertificateRevoked	Good
20	RootSigned8_CaTrusted_Bad	BadCertificate-RevocationUnknown	Good
21	RootSigned9Revoked_CaTrustedCrl_Bad	BadCertificateRevoked	Good
24	RootSigned12Trusted_CaTrusted_Bad	BadCertificate-RevocationUnknown	Good
25	RootSigned13TrustedRevoked_CaTrustedCrl_Bad	BadCertificateRevoked	Good
27	RootSigned16Trusted_CaChain_Bad	BadCertificate-RevocationUnknown	Good
57	SecondarySigned9Trusted_CaIssuers_ScaIssuersCrl_Bad	BadCertificateIssuer-RevocationUnknown	Good
58	SecondarySigned10_CaIssuers_ScaTrustedCrl_Bad	BadCertificateIssuerRevocationUnknown	Good

59	SecondarySigned11_CaTrusted_ScaIssuersCrl_Bad	BadCertificateIssuer-RevocationUnknown	Good
60	SecondarySigned12Trusted_CaIssuersCrl_ScaIssuers_Bad	BadCertificate-RevocationUnknown	Good
61	SecondarySigned13_CaIssuersCrl_ScaTrusted_Bad	BadCertificate-RevocationUnknown	Good
62	SecondarySigned14_CaTrustedCrl_ScaIssuers_Bad	BadCertificate-RevocationUnknown	Good
72	SecondarySigned24Trusted_CaIssuersCrl_ScaIssuersCrlRevoked_Bad	BadCertificateIssuer-Revoked	Good
73	SecondarySigned25Trusted_CaIssuersCrl_ScaTrustedCrlRevoked_Bad	BadCertificateIssuer-Revoked	Good
74	SecondarySigned26Trusted_CaTrustedCrl_ScaIssuersCrlRevoked_Bad	BadCertificateIssuer-Revoked	Good
75	SecondarySigned27TrustedRevoked_CaIssuersCrl_ScaIssuersCrl_Bad	BadCertificateRevoked	Good
76	SecondarySigned28TrustedRevoked_CaIssuersCrl_ScaTrustedCrl_Bad	BadCertificateRevoked	Good
77	SecondarySigned29TrustedRevoked_CaTrustedCrl_ScaIssuersCrl_Bad	BadCertificateRevoked	Good

Table 23: Failed certificate tests

Detailed analysis of the results:

All failed tests of category 2 for which a connection was prevented erroneously have always the same cause: It is tried to establish a connection with certificates that either have not been valid yet or have already expired. In this respect, it must be noted that the 'IgnoreInvalidTime' option has been set for all these tests. This should result in the certificates being recognised as valid, although the 'validity period' is not valid and the establishment of the connection being granted. The tested UA server frame application does not support the suppression of errors in the event of certificates that either have not been valid yet or have already expired. Therefore, the server application always rejects the establishment of the connection with the StatusCode 'BadCertificateTimeInvalid' (0x80140000): "The Certificate has expired or is not yet valid."

The 'IgnoreInvalidTime' option is a functionality which was implemented outside the stack. Accordingly, these failed tests are not relevant to the IT security of the communication stack.

For the tests of the third category for which the establishment of the connection was granted erroneously, it turned out that the malfunction can be traced back to a cause. The following StatusCodes are expected for all failed tests:

0x801B0000	BadCertificateRevocationUnknown
0x801C0000	BadCertificateIssuerRevocationUnknown
0x801D0000	BadCertificateRevoked
0x801E0000	BadCertificateIssuerRevoked

The aim is always to test the functionality of the CRL testing: It should be identified either that the CRL is missing or that the certificate is marked as revoked in the available CRL. All these tests failed, since the functionality of the CRL testing had been implemented incompletely in the tested frame application. Accordingly, it is not possible to make a statement regarding the implementation of the CRL functionality in the stack with the tests described in Table 34.

Summary:

The analysis of the certificate tests has shown that no errors relevant to IT security could be detected in the communication stack of the OPC Foundation in the tested version 1.02.344.5. Only in twelve tests were deviations in the StatusCodes observed.

These results, however, only partially reflect the actual state of the certificate validation in the stack. The implementation of the CRL testing in the stack could not be tested.

8.5 Static code analysis

For the static code analysis, the open source analysis tool *cppcheck* was used. The call was as follows:

```
cppcheck --enable=all --std=c99 --std=c++11 --std=posix -f -v  
-rp=/root/opcua/ascolab_webdav/sdk/src/uastack /root/opcua/ascolab_webdav/sdk/src/uastack 2> results_uastack_raw.txt
```

The parameters with `-std=` activate additional tests for the respective standard.

The *cppcheck* output contained approx. 340 error messages, most of which are irrelevant to IT security. The following four types of error messages were not evaluated further:

1. **(style) The scope of the variable '<variable_name>' can be reduced.**
This message occurred 45 times. The fact that a smaller scope of a variable could have been chosen does not have an impact on IT security.
2. **(style) The function '<function_name>' is never used**
This message occurred 267 times. This refers to source code that is either dead or has not been used yet. The high number of messages might indicate poor maintenance of the code, but is not directly relevant to IT security.
3. **(style) Variable '<variable_name>' is assigned a value that is never used**
This message occurred six times. In this case, too, it is indicated that the source code is probably dead, but this does not have a direct influence on IT security.
4. Eleven messages referred to the source code from the Linux Platform Layer `platforms/linux/*` which is not part of the OPC Foundation stack and was therefore ignored.

Below, the remaining messages were analysed in more detail, summarised in groups and assessed:

1 (error) Possible null pointer dereference

This message occurred three times:

- a [platforms/win32/opcua_p_openssl_pki.c:449]: (error) Possible null pointer dereference: `pFile` - otherwise it is redundant to check if `pFile` is null at line 437
This is not an error: `pFile` has already been tested in line 401 or 404 of the code and cannot be NULL.
- b [platforms/win32/opcua_p_win32_pki.c:108]: (error) Possible null pointer dereference: `pCertificateStoreCfg` - otherwise it is redundant to check if `pCertificateStoreCfg` is null at line 102
This is not an error: The value which is assigned to `pCertificateStoreCfg` is tested in line 96 of the code.
- c [transport/tcp/opcua_tcpconnection.c:1787]: (error) Possible null pointer dereference: `a_ppConnection` - otherwise it is redundant to check if `a_ppConnection` is null at line 1795
This is not an error: `a_ppConnection` is tested in line 1767 of the code.

2 (warning) Using size of pointer <pointer_name> instead of size of its data.

This message occurred once:

[platforms/win32/opcua_p_socket_interface.c:557]: (warning) Using size of pointer OpcUa_P_Socket_g_pSocketManagers instead of size of its data. This is likely to lead to a buffer overflow. You probably intend to write sizeof(*OpcUa_P_Socket_g_pSocketManagers)

This is not an error: It is a structure of indicators. Accordingly, the calculation of the structure size is correct here.

3 (warning) Redundant assignment of "<variable_name>" in switch

This message occurred once:

[transport/https/opcua_httpsstream.c:2989]: (warning) Redundant assignment of "bParseAgain" in switch

The second assignment is redundant. This is not necessary, but is not a problem.

4 (style) Defensive programming

This message occurred once:

[platforms/win32/opcua_p_utilities.c:391]: (style) Defensive programming: The variable nIndex1 is used as array index and then there is a check that it is within limits. This can mean that the array might be accessed out-of-bounds. Reorder conditions such as '(a[i] && i < 10)' to '(i < 10 && a[i])'. That way the array will not be accessed when the index is out of limits.

This is not an error: The string terminator is tested above in line 390 of the code.

5 (style) Finding the same code for an if branch and an else branch is suspicious

This message occurred twice:

a [platforms/win32/opcua_p_mutex.c:231] ->
[platforms/win32/opcua_p_mutex.c:225]: (style) Finding the same code for an if branch and an else branch is suspicious and might indicate a cut and paste or logic error. Please examine this code carefully to determine if it is correct.

This is not an error: This passage was used for debugging purposes.

b [platforms/win32/opcua_p_socket_interface.c:494] ->
[platforms/win32/opcua_p_socket_interface.c:476]: (style) Finding the same code for an if branch and an else branch is suspicious and might indicate a cut and paste or logic error. Please examine this code carefully to determine if it is correct.

This is not an error: It is due to the macros that the two branches look the same for *cppcheck*.

6 (style) Found obsolete function

This message occurred once:

[platforms/win32/opcua_p_utilities.c:409]: (style) Found obsolete function 'gethostbyname'. It is recommended that new applications use the 'getaddrinfo' function

The usage of the obsolete function is controlled by a macro; if the obsolete function is used, problems can occur with multi-threaded applications, because several threads might access a global structure at the same time.

From the IT security perspective, the consequences depend on the implementation of the function `gethostbyname` on the respective operating system and can therefore not be limited clearly. Crashes, however are classified as highly unlikely.

7 (portability) Found non reentrant function <function_name>

This message occurred once:

```
[platforms/win32/opcua_p_utilities.c:409]: (portability) Found non reentrant function 'gethostbyname'. For threadsafe applications it is recommended to use the reentrant replacement function 'gethostbyname_r'
```

This is an error: See point 6.

Summary:

Only the two related error messages and are an actual problem and could have impacts on IT security in the case of a multi-threaded server operation.

8.6 Fuzzing

8.6.1 Programming in Peach

Structure of a pit file

The central part of the testing with *Peach* is the so-called pit file which specifies how the fuzzing framework has to work. The main elements in the structure of a pit file include:

- the data model: Here, it is specified how the protocol to be tested is structured. The messages which constitute the protocol are, for example, shown in header and body and these, in turn, consist of complex data structures which can be modelled based on basic data types such as String and Number.
- the state model: If the structures of the individual messages are defined, it is also important to specify which messages are expected in which order and how the messages are related, especially the requests and responses. This is modelled in a state machine in *Peach*.
- the specification of the test run: Here, it is specified, among other things, which parameters of a message may be changed during fuzzing and according to which rules such a change is made.

Restriction for the data model

Peach does not support recursive data types or structures. Accordingly, the testing of possible recursion depths for the OPC UA data types **Variant** and **DiagnosticInfo** as part of the fuzzing tests was dispensed with. The usage of recursions might result in application crashes, for instance. This means that this type of error could not be reproduced in *Peach*.

Fuzzing strategies

For all basic data types which are supported by the framework, *Peach* offers so-called mutators which define how a data type is changed during fuzzing. How the parameters of a message are fuzzed depends on the fuzzing strategy chosen. The following two strategies are available:

- sequential: Here, each parameter of a message is fuzzed individually and in their respective order with all mutators applicable to this data type and all defined values for the respective mutator. This results in a certain number of iterations for the test run according to which all permissible combinations of parameters, mutators and values are tested.
- random: For the random strategy, the same combinations of parameters, mutators and values are generally possible as for the sequential strategy. The difference is that a new combination is randomly chosen for the triple (parameter, mutator, value) for each iteration. You must define the number of desired iterations yourself, since *Peach* will run endlessly otherwise. The reasons for the number of iterations chosen for the fuzzing tests performed can be found in Section 8.8.2.

As an additional setting, it is possible to change several parameters simultaneously for each iteration with the random strategy instead of fuzzing only one parameter for each iteration as with the sequential strategy.

In summary, three types of fuzzing strategy tests are available: sequential, random with one changeable parameter and random with several simultaneously changeable parameters.

In general, the fuzzing strategy sequential is to be preferred to the random strategy with one changeable parameter: All possible triples (parameter, mutator, value) are run exactly once. With the random strategy, it is possible that some triples will not be tested at all, whereas others can be tested several times.

Restriction by choices

The reasons for the usage of the three available versions of fuzzing strategies are as follows: When modelling complex data types in *Peach*, it makes sense to work with so-called 'choice' structures. They make it possible to design the contents of data structures depending on a specified parameter in a flexible manner.

An illustrative example of the necessity of choices are UA strings. They consist of two basic data types: one number for the length specification of the character string and the character string itself. However, this structure does not apply to NULL strings which only consist of the length specification with the value -1 and do not contain any character string. In order to model this case analysis, the resource 'choice' is used, which adds one character string or not depending on the length specification (> -1 or $= -1$) of the respective instance of a UA string.

The problem is that *Peach*, when fuzzing a complex data type with choice with the sequential fuzzing strategy, always chooses only the choice version defined in the first place and never runs the other branches. In the case of a data type with five versions within one choice, only one version will be tested if only the sequential fuzzing strategy is worked with.

This restriction can be overcome for the random fuzzing strategy by forcing *Peach* by means of so-called data sets to choose another version than the first available version when instantiating a complex data type.

This can be summarised as follows:

- With the sequential strategy, each iteration is used optimally, since there are no repetitions. For complex data types, however, it will only be possible to test one version.
- With the random strategy, certain triples (parameter, mutator, value) are tested several times, whereas others as part of a finite test run are not tested at all. It is possible, however, to force *Peach* to test all defined versions of complex data types.

Modelled messages

The basic idea when setting up the tests was to test the SecureChannel, Session and Discovery services which are crucial for the IT security of OPC UA from the client point of view. Therefore, all messages which are exchanged between the client and the server in order to establish the connection were simulated in *Peach*:

- on UA TCP level:
 - Hello request (name 'HEL' in Table 35)
 - Acknowledge and Error response
- on UASC level:
 - SecureChannel service:
 - OpenSecureChannel request and response (name 'OPN' in Table 35)
 - CloseSecureChannel request and response
 - Session service:
 - CreateSession request and response
 - ActivateSession request and response
 - CloseSession request and response
 - Discovery service:
 - FindServers request and response
 - GetEndpoints request and response

As the starting point for the fuzzing, messages with valid contents were used, for which one or several parameters were changed afterwards.

Dependency on the securityMode

In addition to the messages available, an essential aspect for the fuzzing of the OPC UA communication is with which securityMode the connection is established: As described in Section 8.3, *Peach* simulates a client and, in the securityMode None, sends its messages directly to the server to be tested. In the securityMode SignAndEncrypt, however, the *Peach* messages are sent to *secucon* first for signing and encryption.

It was decided to extensively test the basic functionality of the OPC UA communication between the client and the server in the securityMode None. In addition to this, some of these tests were repeated in the securityMode SignAndEncrypt in order to also test the signature and encryption functionalities.

It was assumed that the functionality which is used in the securityMode Sign is also used in the SignAndEncrypt. For the verification of this assumption, see Section 8.8.2.

8.6.2 Choice of fuzzing tests

When choosing the fuzzing tests, four different objectives were pursued:

- dynamic code analysis: Some of the tests were carried out to obtain data for the dynamic code analysis. This requires that a server compiled with debug information be tested. Moreover, the code coverage was measured with *lcov* in these tests. For this purpose, the source code was compiled with additional compiler flags (see Section 8.8).
- Production-like server: The runtime behaviour of test applications can differ depending on the compiler settings, such as with or without debug information. Therefore, certain tests were repeated with a server application that was compiled with settings such as could be found in production environments.
- Increase in code coverage: For some tests, the number of iterations was increased significantly in order to achieve the highest possible code coverage. These tests had a runtime of several weeks.
- Verification of the assumptions regarding code coverage: To verify the code coverage assumptions made in Section 8.6.1, additional tests were performed. Thus, it could be demonstrated that the choice of the criteria did not have a negative impact on the code coverage in the fuzzing tests.

In Annex A, all fuzzing tests performed are listed in Table 35: Tests with the three fuzzing strategies described in Section 8.6.1, with the three securityModes and with the debug and release versions of the server were carried out.

The test series 009 regarding the increase in the code coverage had a runtime of approx. 39 days. Due to technical problems, test 03 was cancelled after a few hours and test 02 after approx. eleven days. The tests 04 and 07 ran without any interruption over the entire test period. Thus, the following numbers of iterations could be achieved:

- Test 02: approx. 1.92 million
- Test 04: approx. 1.56 million
- Test 07: approx. 7.56 million

8.6.3 Results

Error identified during the modelling process

When modelling the data structures and developing the tests with the fuzzing framework *Peach*, both deviations of the implementation from the specification and inconsistencies in the specification were identified. There was no weighting in the following lists.

Deviations:

1. According to Part 6 Table 13, an ExtensionObject should also contain an 'Encoding' byte after the NodeId for message chunks. The messages of the server, however, are without an 'Encoding' byte. The NodeId is followed immediately by the authenticationToken.

2. In the securityMode None, the stack implementation expects NULL strings, i.e. the length -1, for the parameters SecurityPolicyUriLength, SenderCertificate and ReceiverCertificateThumbprint.
3. In Part 6 Table 30 of the specification, these parameters are not defined as ByteString or UA string, although they are structured like this.
4. SecureChannel and Session requests coded with UA binary with an xml encoding TypeId identifier are still decoded successfully.
5. The server accepts values smaller than 8,192 for the parameters ReceiveBufferSize and SendBufferSize.
6. In Part 6 p. 41 of the specification, it is defined that the nonce should be NULL in the securityMode None. The server, however, responds with a nonce of the length 1 with the value 1.
7. According to Part 4 Table 9 of the specification, the secureChannelId should occur twice in the CloseSecureChannel request: as UInt32 and as ByteString. This redundancy, however, is not provided in the implementation.
8. The server accepts messages with an incorrect sequenceNumber. According to Part 6 p. 39, this should trigger a transport error.
9. The server accepts OpenSecureChannel requests with protocol versions which deviate from the version sent in the Hello message. This does not correspond to the specification (see Part 6 S. 41).
10. According to Part 4 Table 13 of the specification, a userIdentityToken must be included in the ActivateSession request. Although the server reports in the EndpointDescriptions of the CreateSession response that only three userIdentityToken types (Anonymous, UserName, Certificate) are supported, the ActivateSession request is accepted without a userIdentityToken.
The problem is known and has already been eliminated in the new specification by adding that NULL or empty userIdentityTokens should be interpreted as Anonymous userIdentityTokens.
11. In Part 6 Table 24 of the specification, the parameter SignedSoftwareCertificate is defined as ByteString. In the implementation, however, two ByteStrings, i.e. CertificateData and Signature, are defined.
12. According to Part 4 Section 7.30 of the version 1.02 specification, SignatureData consists of the following elements:
signature
algorithm
In the implementation, however, the order is reversed.

The order was adjusted in the version 1.03 specification in order to comply with the implementation.

Inconsistencies:

1. The definitions for the data type ChannelSecurityToken in Part 4 Table 7 and Part 6 Table 35 are inconsistent.
2. In general, it is recommended to test whether the two data types UtcTime and DateTime are required, as they seem to be redundant.
3. In Part 3 p. 65-66, localizedText is defined as data type consisting of two strings. In Part 4 Table 123, localizedText is used as the name of an Int32 parameter. It is recommended to change the parameter name to localizedTextIndex in order to avoid any possible confusion.
4. For the parameter LocalizedText, different orders for Text and Locale are defined in Part 3 Table 22 and Part 6 Table 12.

Comment: In the version 1.02 specification, calling CloseSession without ActivateSession is not provided for. In the version 1.03 specification, a client is given more possibilities, since CreateSession can now also be followed by CloseSession.

Most of the deviations and inconsistencies found here are not relevant to the IT security of the OPC UA protocol. Exceptions are the deviations 7 and 8, both of which affect mechanisms that should contribute to the security and stability of the protocol. `sequenceNumber` in particular plays an important role in fending off replay attacks. The missing verification of the version number could impair the stability of OPC UA applications if clients and servers with different protocol versions try to communicate with each other.

Errors found when carrying out the tests

In the vast majority of the fuzzing tests performed, iterations in the size of 100,000 to 200,000 were run. Thus, the number of iterations adds up to approx. 1 – 1.5 million per test series. Together with the test series 009 scheduled for several weeks, a total of well over 15 million iterations were run in the fuzzing tests.

Apart from the data for the dynamic code analysis, solely performing the fuzzing tests did not lead to further findings with respect to implementation or systematic errors. In particular, there were no crashes caused by errors in the stack.

Summary

The fuzzing tests of the client server messages of the three service `SecureChannel`, `Session` and `Discovery` have shown that the stack implementation runs in a stable manner even with unexpected contents.

During the modelling phase, however, it could be observed that some mechanisms provided in the specification for IT security and stability were not implemented or were implemented incompletely or incorrectly. The complete and correct evaluation of the `sequenceNumber` in particular must urgently be ensured.

To extend this analysis, the formal and systematic modelling of the protocol using a prover could be taken into consideration.

8.7 Dynamic code analysis

The dynamic code analysis was carried out with the *Valgrind* open source tool. For these purposes, the UA server was compiled with debug information and flags in order to measure the code coverage. This application was then started with *Valgrind*. The following fuzzing tests were carried out with *Valgrind* and evaluated completely (see Table 35 for further details on the individual test series):

Test series no.	Test series name	securityMode
001	<i>sequential individual parameter fuzzing</i>	None
002	<i>random individual parameter fuzzing</i>	None
003	<i>random multiple parameter fuzzing</i>	None
007	<i>sequential individual parameter fuzzing</i>	SignAndEncrypt

Table 24: List of the fuzzing test evaluated completely in the dynamic code analysis

The following test series were also carried out with *Valgrind*, but only server crashes were examined here:

Test series no.	Test series name	securityMode
009	<i>random multiple parameter fuzzing</i>	SignAndEncrypt
015	<i>sequential individual parameter fuzzing</i>	Sign
016	<i>random multiple parameter fuzzing</i>	None

Table 25: List of the fuzzing test evaluated partially in the dynamic code analysis

8.7.1 Results:

In total, there were more than 400 error messages of *Valgrind*. The evaluation showed that many messages had the same causes and that the implementation errors were not part of the subject under examination for some of these causes identified, which means that they did not affect the communication stack.

The errors in the stack can be assigned to the following four error classes:

Memory leak with NodeId decoding

Class: Memory leak with influenceable size

Function: OpcUa_BinaryDecoder_ReadNodeIdBody()

References in Valgrind logs:

Valgrind log	Lines
001_seq_indi_param_fuzzing\02_OPN_fuzzing	26, 40, 54
001_seq_indi_param_fuzzing\03_CRE_fuzzing	66, 80, 94
001_seq_indi_param_fuzzing\04_ACT_fuzzing	66, 80
001_seq_indi_param_fuzzing\05_CLO_SESS_fuzzing	66, 80
001_seq_indi_param_fuzzing\06_CLO_CHAN_fuzzing	12, 26, 40, 54
001_seq_indi_param_fuzzing\07_FIND_fuzzing	66, 80, 94, 108
001_seq_indi_param_fuzzing\08_GET_fuzzing	66, 80, 94, 108
002_rand_indi_param_fuzzing\02_OPN_fuzzing	53, 67, 81, 95, 109
002_rand_indi_param_fuzzing\03_CRE_fuzzing	66, 80, 94, 108, 122
002_rand_indi_param_fuzzing\04_ACT_fuzzing	80, 108, 122, 164, 178, 192, 206, 220, 248
002_rand_indi_param_fuzzing\05_CLO_SESS_fuzzing	66, 80, 94, 108, 122
002_rand_indi_param_fuzzing\06_CLO_CHAN_fuzzing	12, 40, 54, 68, 82, 96, 110, 124, 138
002_rand_indi_param_fuzzing\07_FIND_fuzzing	66, 80, 94, 108, 122
002_rand_indi_param_fuzzing\08_GET_fuzzing	66, 80, 94, 108, 122
003_rand_multi_param_fuzzing\02_OPN_fuzzing	53, 67, 81, 95, 109
003_rand_multi_param_fuzzing\03_CRE_fuzzing	66, 94, 108, 122, 136
003_rand_multi_param_fuzzing\04_ACT_fuzzing	66, 80, 122, 136, 150, 164, 178, 192, 220
003_rand_multi_param_fuzzing\05_CLO_SESS_fuzzing	66, 80, 94, 108, 122, 136, 150, 164
003_rand_multi_param_fuzzing\06_CLO_CHAN_fuzzing	12, 26, 40, 54, 68, 82, 96, 110, 124
003_rand_multi_param_fuzzing\07_FIND_fuzzing	66, 80, 94, 108, 122, 136, 150
003_rand_multi_param_fuzzing\08_GET_fuzzing	66, 80, 94, 108, 122, 136
007_seq_indi_param_fuzzing\02_OPN_fuzzing	2459, 2473, 2487, 2501
007_seq_indi_param_fuzzing\03_CRE_fuzzing	12
007_seq_indi_param_fuzzing\03_CRE_fuzzing	75, 89, 103, 117
007_seq_indi_param_fuzzing\04_ACT_fuzzing	75, 89
007_seq_indi_param_fuzzing\05_CLO_SESS_fuzzing	75, 89
007_seq_indi_param_fuzzing\06_CLO_CHAN_fuzzing	12, 26, 40, 54
007_seq_indi_param_fuzzing\07_FIND_fuzzing	75, 89, 103, 117
007_seq_indi_param_fuzzing\08_GET_fuzzing	75, 89, 103, 117

Table 26: Occurrence of the memory leak in the Valgrind logs

Technical detailed description:

The type of `NodeId` is set only after deserialisation. If an error occurs during deserialisation (triggered by the sender), the allocated memory space can no longer be freed. With type *String* and *ByteString* `NodeIds`, the size of the allocated memory space is preset by the sender.

Impact:

Due to the permanent allocation of memory space, an attacker is able to set the server into a state in which only small or even no requests at all can be responded to. If the handling of errors is partially incorrect in the case of failed allocations, consequential errors would be possible.

The most easily achievable target for an attacker would thus be to restrict the availability of the server or to make it completely unavailable.

Error when parsing the certificate chain

Class: Possible memory access error

Function: PKIProvider::SplitCertificateChain()

References in Valgrind logs:

Valgrind log	Lines
001_seq_indi_param_fuzzing\03_CRE_fuzzing	34
001_seq_indi_param_fuzzing\04_ACT_fuzzing	34
001_seq_indi_param_fuzzing\05_CLO_SESS_fuzzing	34
001_seq_indi_param_fuzzing\07_FIND_fuzzing	34
001_seq_indi_param_fuzzing\08_GET_fuzzing	34
002_rand_indi_param_fuzzing\03_CRE_fuzzing	7
002_rand_indi_param_fuzzing\04_ACT_fuzzing	7
002_rand_indi_param_fuzzing\05_CLO_SESS_fuzzing	7
002_rand_indi_param_fuzzing\07_FIND_fuzzing	7
002_rand_indi_param_fuzzing\08_GET_fuzzing	7
003_rand_multi_param_fuzzing\03_CRE_fuzzing	34
003_rand_multi_param_fuzzing\04_ACT_fuzzing	34
003_rand_multi_param_fuzzing\05_CLO_SESS_fuzzing	7
003_rand_multi_param_fuzzing\07_FIND_fuzzing	7
003_rand_multi_param_fuzzing\08_GET_fuzzing	7

Table 27: Occurrence of the error when parsing certificate chains in the Valgrind logs

Technical detailed description:

The sender certificate received can be a certificate chain in which several certificates are arranged in a row. In order to be able to access the individual elements, an index is created, which consists of a pointer to the respective start of a certificate and its length. When this index is created, it is not checked whether the last certificate is contained completely. If this is not the case, the length specification is invalid. Accessing the full length of this memory area (e.g. with memcpy) results in a memory access error.

Impact:

The specific impact of the incorrect memory block length depends on the further handling of the memory area. An option would be the transfer to OpenSSL for validation. If the memory area is evaluated using an ASN.1 parser, errors contained therein might result in crashes, among other things.

In the further course, the incorrect index is transferred to the application, in which additional errors can occur caused by the further processing.

In the foreseeable cases, this could lead to read access to the invalid memory space and thus possibly to crashes. An attacker could thus make the server unavailable.

Errors when sending a ServiceFault caused by an unexpected Request Body

Class: Memory access error with the constant size of 4 bytes

Function: OpcUa_Endpoint_BeginProcessRequest()

References in Valgrind logs:

Valgrind log	Lines
001_seq_indi_param_fuzzing\03_CRE_fuzzing	7
001_seq_indi_param_fuzzing\04_ACT_fuzzing	7
001_seq_indi_param_fuzzing\05_CLO_SESS_fuzzing	7
001_seq_indi_param_fuzzing\07_FIND_fuzzing	7
001_seq_indi_param_fuzzing\08_GET_fuzzing	7
002_rand_indi_param_fuzzing\02_OPN_fuzzing	21
002_rand_indi_param_fuzzing\03_CRE_fuzzing	34
002_rand_indi_param_fuzzing\04_ACT_fuzzing	34

002_rand_indi_param_fuzzing\05_CLO_SESS_fuzzing	34
002_rand_indi_param_fuzzing\07_FIND_fuzzing	34
002_rand_indi_param_fuzzing\08_GET_fuzzing	34
003_rand_multi_param_fuzzing\02_OPN_fuzzing	21
003_rand_multi_param_fuzzing\03_CRE_fuzzing	7
003_rand_multi_param_fuzzing\04_ACT_fuzzing	7
003_rand_multi_param_fuzzing\05_CLO_SESS_fuzzing	34
003_rand_multi_param_fuzzing\07_FIND_fuzzing	34
003_rand_multi_param_fuzzing\08_GET_fuzzing	34
007_seq_indi_param_fuzzing\02_OPN_fuzzing	7, 34, 61, 88, 108, 128, 155, 182, 209, 236, 263, 290, 317, 337, 355, 375, 395, 412, 430, 452, 474, 493, 512, 531, 551, 571, 592, 609, 628, 647, 666, 687, 708, 729, 748, 768, 788, 809, 830, 851, 868, 886, 904, 924, 944, 962, 980, 998, 1016, 1035, 1053, 1071, 1089, 1107, 1125, 1143, 1161, 1179, 1200, 1219, 1238, 1257, 1276, 1295, 1314, 1335, 1354, 1373, 1392, 1411, 1430, 1449, 1468, 1487, 1506, 1525, 1544, 1563, 1582, 1601, 1619, 1640, 1658, 1676, 1694, 1712, 1730, 1748, 1766, 1784, 1802, 1820, 1838, 1856, 1874, 1892, 1910, 1928, 1949, 1970, 1991, 2008, 2029, 2050, 2071, 2090, 2109, 2126, 2143, 2160, 2177, 2198, 2219, 2239, 2260, 2281, 2302, 2339, 2360, 2380, 2400, 2427
007_seq_indi_param_fuzzing\03_CRE_fuzzing	7
007_seq_indi_param_fuzzing\03_CRE_fuzzing	7
007_seq_indi_param_fuzzing\04_ACT_fuzzing	7
007_seq_indi_param_fuzzing\05_CLO_SESS_fuzzing	7
007_seq_indi_param_fuzzing\07_FIND_fuzzing	7
007_seq_indi_param_fuzzing\08_GET_fuzzing	7

Table 28: Occurrence of the error when sending a serviceFault in the Valgrind logs

Technical detailed description:

Sending a valid serialised data type with a memory space size smaller than the one of the OpcUa_Request-Header. Due to the unverified conversion in OpcUa_RequestHeader, subsequent access to the RequestHandle exceeds the valid memory area of the original data type. The maximum length is 4 bytes. This data is sent back to the client in the ResponseHeader as RequestHandle.

Impact:

The concrete impacts when accessing invalid memory space are specific to the respective system. If there is no crash, the memory content with the length of 4 bytes is sent to the client. Reading the memory contents in a targeted manner is not likely due the small size. In the event of a crash, the attacker could make the server unavailable.

Missing initialisation of a stack variable

Class: Jump based on uninitialised data and possibly memory freed with an invalid address

Function: OpcUa_SecureListener_ProcessOpenSecureChannelRequest ()

References in Valgrind logs:

Valgrind log	Lines
001_seq_indi_param_fuzzing\02_OPN_fuzzing	7

002_rand_indi_param_fuzzing\02_OPN_fuzzing	7
003_rand_multi_param_fuzzing\02_OPN_fuzzing	7
007_seq_indi_param_fuzzing\02_OPN_fuzzing	2331, 2335

Table 29: Occurrence of the access to an uninitialised variable in the Valgrind logs

Technical detailed description:

If an error occurs when decoding the message header of an OpenSecureChannel message, before the string with the security policy was decoded, the `OpCua_String_Clear()` method was applied to the uninitialised stack variable. Depending on the random memory content of the structure, there might be a free attempt of a random memory address. The further behaviour depends on the memory address.

Impact:

The exact impact depends on the random memory content. In the worst case, an attempt is made to free an invalid memory address, which results in a crash in most cases. An attacker could thus try to make the server unavailable.

In rare cases, this could result in a double-free exploit. However, this is considered to be very unlikely, as the address which is transferred to `free()` cannot be influenced from the outside.

Summary:

The errors identified in the dynamic code analysis mainly have a negative impact on the availability of the server due to the filling of the memory by writing or crashes. These errors, however, might also have other impacts that are difficult to predict.

For a final and definite evaluation of the error messages and impacts of the errors described, a further, more detailed analysis must be performed.

8.8 Code coverage

8.8.1 Measurement

In order to measure the code coverage for runtimes, the following C Flags are required when compiling the server application: `-fprofile-arcs -ftest-coverage`

These settings have the effect that, using a tool called *lcov*, it can be measured how often each line of code is executed and which branches are run. The number of functions called is also documented.

The code coverage can be measured for individual tests and aggregated afterwards for entire test series. With another tool called *genhtml*, the results can be visualised clearly, with the folder structure and files of the source code being used for this purpose.

The server version compiled with debug information was also always compiled in such a manner that the code coverage could have been measured at the same time. Accordingly, figures are available for the following test series:

Test series no.	Test series name	securityMode	Test objective
001	<i>sequential individual parameter fuzzing</i>	None	Dynamic code analysis
002	<i>random individual parameter fuzzing</i>	None	Dynamic code analysis
003	<i>random multiple parameter fuzzing</i>	None	Dynamic code analysis
007	<i>sequential individual parameter fuzzing</i>	SignAndEncrypt	Dynamic code analysis

009	<i>random multiple parameter fuzzing</i>	SignAndEncrypt	Increase in code coverage
013	<i>certificate tests</i>	SignAndEncrypt	Certificate tests
015	<i>sequential individual parameter fuzzing</i>	Sign	Verification of the assumptions regarding the code coverage
016	<i>random multiple parameter fuzzing</i>	None	Verification of the assumptions regarding the code coverage

Table 30: List of test series for which the code coverage was measured

The first four test series 001, 002, 003 and 007 were used for the actual dynamic code analysis. The test series 009 were used to achieve an increase in the code coverage. With the test series 013, the functionality for the validation of certificates was tested and the code coverage measured. The test series 015 and 016 were used to test if a higher code coverage could have been reached with another selection of parameters in the fuzzing tests. These tests were thus used to verify the following assumptions regarding the code coverage:

- Assumption 1: The tests with securityMode Sign (test series 015) do not increase the code coverage significantly.
- Assumption 2: The increase in the number of iterations (test series 009) does not increase the code coverage significantly.
- Assumption 3: The increase in the number of iterations per choice (test series 016) does not increase the code coverage significantly.

Due to technical problems (crash of *Peach* and the server frame application), no measurement of the code coverage for the tests 02 and 03 is available for the test series 009.

Table 31, 32 and 33 list, depending on the tests carried out, the code coverage for lines, functions and branches, each as a percentage of the entire source code and as an absolute value. The grey lines in these three tables are used as basis for comparison in order to verify the assumptions.

#	Basis for the measurement	Line coverage		Function coverage		Branch coverage	
1	Starting the server and stopping it without any tests	9.8%	3157	11.8%	255	3.8%	1051
2	Test series 001	35.3%	11328	42.4%	913	17.8%	4935
3	Test series 002	31.9%	10251	36.2%	779	16.5%	4574
4	Test series 003	34.1%	10953	38.5%	829	18.4%	5114
5	Test series 007	38.3%	12314	44.7%	962	19.4%	5380
6	Test series 001, 002, 003, 007	41.8%	13428	49.5%	1066	22.7%	6287
7	Test series 001, 002, 003, 007 and test series 015 with securityMode Sign (verification of assumption 1)	41.8%	13436	49.5%	1066	22.7%	6293
8	Test series 001, 002, 003, 007 and test series 013 certificate tests	41.9%	13442	49.5%	1066	22.7%	6301
	Number for the entire source code	100%	32119	100%	2153	100%	27754

Table 31: Code coverage achieved depending on the test series performed (assumption 1)

Verification of assumption 1: The evaluation of the test series 001, 002, 003, 007 and 015 (Table 31 Line 7) does not result in a higher code coverage as compared to the code coverage of the test series 001, 002, 003 and 007 (Table 31 Line 6).

Comment regarding Line 8 in Table 31: The hardly changed code coverage after the certificate tests have been carried out can be explained by the fact that a large part of the validation of the certificates is out-sourced in OpenSSL, i.e. the integrated cryptographic library.

#	Basis for the measurement	Line coverage		Function coverage		Branch coverage	
1	Test 04 of the test series 001, 002, 003, 007	36.8%	11822	42.2%	909	18.9%	5246
2	Test 04 of the test series 001, 002, 003, 007 and test series 009 long test run (verification of assumption 2)	39.0%	12533	45.9%	988	20.4%	5672
3	Test 07 of the test series 001, 002, 003, 007	36.8%	11814	42.0%	904	19.0%	5266
4	Test 07 of the test series 001, 002, 003, 007 and test series 009 long test run (verification of assumption 2)	43.0%	13797	52.8%	1137	23.6%	6552
	Number for the entire source code	100%	32119	100%	2153	100%	27754

Table 32: Code coverage achieved depending on the total number of iterations (assumption 2)

Verification of assumption 2: The evaluation of the tests 04 and 07 of the test series 001, 002, 003, 007 and 009 (Table 32 Lines 2 and 4) results in a higher code coverage as compared to the code coverage of the same tests for the test series 001, 002, 003 and 007 (Table 32 Lines 1 and 3). The increase in the code coverage, however, is qualified if the entire test series 001, 002, 003 and 007 are examined. Moreover, it must be taken into consideration that, for the test 07 in particular, for which the increase in the code coverage is the most significant, the number of iterations exceeds for the test series 009 the sum of the number of iterations for the test series 001, 002, 003 and 007 by the factor of 18.3.

#	Basis for the measurement	Line coverage		Function coverage		Branch coverage	
1	Test 02 of the test series 001, 002, 003, 007	33.9%	10888	39.9%	859	17.4%	4829
2	Test 02 of the test series 001, 002, 003, 007 and test series 016 with additional 10k iterations per choice (verification of assumption 3)	34.0%	10905	39.9%	859	17.5%	4853
3	Test 04 of the test series 001, 002, 003, 007	36.8%	11822	42.2%	909	18.9%	5246
4	Test 04 of the test series 001, 002, 003, 007 and test series 016 with additional 10k iterations per choice (verification of assumption 3)	37.1%	11904	42.6%	917	19.0%	5287
5	Test 07 of the test series 001, 002, 003, 007	36.8%	11814	42.0%	904	19.0%	5266
6	Test 07 of the test series 001, 002, 003, 007 and test series 016 with additional 10k iterations per choice (verification of assumption 3)	37.0%	11872	42.4%	913	19.1%	5296
	Number for the entire source code	100%	32119	100%	2153	100%	27754

Table 33: Code coverage achieved depending on the number of iterations per choice (assumption 3)

Verification of assumption 3: When fuzzing with the random fuzzing strategy, the number of iterations per choice was set to 5,000 for the test series 002 and 003. For the test series 016, the number of the iterations per choice is increased to 10,000. The tripling of the number of iterations per choice in the line 2, 4 and 6 in Table

33, however, did not result in a significant increase in the code coverage as compared to the results with 5,000 iterations per choice in the lines 1, 3 and 5.

8.8.2 Analysis of the results

The test series 001, 002, 003 and 007 have a code coverage of approx. 43% for the lines and approx. 24% for the branches. This can be explained as follows:

- The test application is a single-threaded server. This means that the following code components are not run:
 - Code which is only active in the case of multi-threading (threading; synchronisation)
 - Synchronous APIs; they refer to functions which work in a blocking manner, which is only possible in multi-thread operations. Asynchronous APIs are used instead.
 - Functions which only contain the client functionality. Since only the server side was tested, no client functions were run.

The analysis of the code coverage can be refined with the visualisation of the results by means of *genhtml* to the level of the source code files. The file names with the components 'Connection' and 'Channel' refer to client code, whereas the name components 'Listener' and 'Endpoint' refer to server code.
- In the tests, certain functionalities were not examined at all or only to a limited extent:
 - Due to their relevance to IT security, the tests were prioritised so that only the three services SecureChannel, Session and Discovery were examined. Accordingly, the code for further services, such as attributes or subscription, was not executed. The same also applies for the large number of data structures which are only used for these services which have not been examined.
 - In the test environment, sufficient memory was always available so that no branches due to failed memory allocations were run.
 - Mechanisms such as the splitting of particularly long contents to several messages (referred to as "chunking"), the renewal of SecureChannels, interrupted connections etc. were not tested.
 - In some places, the source code has already included parts of functionalities, such as certain cryptographic functions, which, however, will only be completed in future versions and can thus be only used there.

8.9 Proof of concept

Within the framework of this project, no proof of concept was realised. It is recommended to demonstrate proof of the following vulnerabilities by means of proofs of concept as a follow-up to this project:

- Replay attacks by exploiting the missing testing of the sequenceNumber
- Exploitation of compromised certificates the validity of which is not detected due to the missing testing of CRLs
- Denial-of-service attacks by exploiting memory leaks

8.10 Summary

The following results of the tests of the reference implementation of the OPC Foundation communication stack must be noted.

Certificate tests

The analysis of the certificate test results has shown that the OPC Foundation communication stack in the tested version 1.02.344.5 complies with the specification with respect to allowing or denying connections when self-signed certificates are used.

The functionality used for the testing of CRLs could not be tested in this framework application.

Static code analysis

The static code analysis resulted in approx. 340 error messages the vast majority of which are irrelevant with respect to IT security: The error messages that were not examined further related to the possible restriction of the scope of variables, source code that is dead or not used or to source code of the Linux platform layer which is not part of the OPC Foundation stack. These errors in the code can and should be improved, but do not have any direct relevance to the IT security of the communication stack from our point of view.

The remaining messages were analysed in more detail. This resulted in the following two errors which are essential to IT security: The function `gethostbyname` is obsolete and not 'reentrant'. In rare cases, this causes connection problems if multi-threaded servers are used. Depending on the implementation of the function, crashes cannot be excluded completely either.

The errors identified were not classified as so severe regarding IT security that a proof of concept would have to be developed.

Fuzzing

The development of the tests in *Peach* (modelling) led to the identification of the following errors:

- The sequenceNumber is not tested in UASC. This is a security vulnerability the exploitability of which remains to be shown as part of a proof of concept.
- Several deviations between the specification and implementation of the OPC Foundation stack as well as inconsistencies in the specification

Despite comprehensive tests with more than 15 millions of iterations, the fuzzing tests did not identify any further implementation errors.

Dynamic code analysis

The errors identified in the dynamic code analysis mainly have a negative impact on the availability of the server due to the utilisation of the memory or crashes. These errors, however, might also have other impacts that are difficult to predict at this point of the analysis.

For a final and definite evaluation of the error messages and impacts of the errors described, a further, more detailed analysis must be performed.

Code coverage

The test series performed showed a code coverage of approx. 43% for the lines and approx. 23% for the branches. The relatively low code coverage is due to the fact that, for example, multi-threading code sections and client functionalities are not run (test of a single-threaded server). Moreover, other OPC UA services, such as the Subscription service, among other things, were not tested, since the three services SecureChannel, Session and Discovery were focussed on because of their relevance to IT security.

Furthermore, it was verified that the code coverage cannot be increased by increasing the number of iterations (also in choices) and testing with the securityMode Sign.

9 Outlook

As part of this project, not all IT security areas could have been analysed and tested. However, the following additional work items for examining the IT security of the OPC UA specification and the implementation of the communication stack can be mentioned as further findings. The topics were sorted according to their priority in descending order from our point of view:

- Further analysis of Valgrind error messages: Since interactions between implementation errors for test runs with several ten thousands of iterations cannot be excluded, it is recommended to eliminate the errors identified in the source code. Afterwards, a new dynamic code analysis with new data can be carried out.
- Moreover, it is possible to determine the exact impacts of already identified errors by continuing the analysis of these errors.
- Testing LDS (local discovery server) with RegisterServer: For the Discovery service, the messages of a client sent to a server were tested. These tests can be extended by also testing the RegisterServer message from a server to an LDS.
- Verification of the exploitability of identified vulnerabilities in the implementation by means of proofs of concept
- Testing of mechanisms for the management of connections: Mechanisms, such as chunking, the renewal of SecureChannels, the simulation of interrupted and re-established connections or transmission errors can still be tested additionally.
- Other tools for static and dynamic code analyses: The use of different tools can lead to new findings both when carrying out the static and the dynamic code analysis.
- Testing other data structures on the server side: This previous testing of the reference implementation is restricted to the three services SecureChannel, Session and Discovery. Accordingly, only the data structures required for these services are modelled and tested. In order to test the encoder/decoder functionality in more detail, further data structures can be added in the data model.
The testing of recursive data structures mentioned in Section 8.6.1 could also be added on a supplementary basis.
- Testing the client side: For the tests performed in AP4, only the client side was simulated to test the server side. It can make sense, however, to also examine carefully those code sections which are not executed by servers, but clients.
- Testing other services on the server side: This previous test was restricted to testing the reference implementation of the communication stack. Since the stack, however, only covers the SecureChannel, Session and Discovery services, it can make sense to extend the examination to all other services of the OPC UA Specification.
- Testing correct and incorrect messages in the wrong order: In these previous tests, the order of the messages prescribed by OPC UA when establishing a connection was respected. These tests can also be extended by not only varying the contents, but also the order of the messages.
- Changing the test environment: Alternatives are 64-bit architectures, multi-threaded applications and Windows platforms.
- Modelling by means of a tool used to verify protocols: For the precise testing of the IT security of the OPC UA protocol, modelling using formal methods can contribute to the identification of any systematic errors.
- Other programming languages: Within the framework of this project, the reference implementation in ANSI C of the OPC UA communication stack was tested. However, other implementations in other programming languages are available.
- New version 1.03 specification: So far, the existing specification (version 1.02) has been examined. In the course of the project, however, parts of the new specification were published. As soon as all documents are available, the new version can also be examined.

Annex A: Supplements to Section 8

Supplements to Section 8.4 Certificate tests:

Table 34 lists the 92 certificate tests and has the following structure:

For a better readability of the table, the columns belonging together were highlighted in colour. This means that there are three groups, with one group each for the CA (columns 3 and 4: CA, CRL), for the sub CA (columns 5 to 8: SCA, Error, CRL, Revoked) and for the client certificate (columns 9 to 11: Cert, Error, Revoked).

Meaning of the individual columns:

No.: unambiguous test number

Type: refers to the PKI structure. 'Self' stands for self-signed certificates, 'Root' for a PKI with a single CA and 'Secondary' for a PKI with root CA and sub CAs.

CA: indicates whether a CA certificate is available for the verification of the certificate chain and, if this is the case, where the CA certificate comes from and if it is trusted. 'Chain' means that the CA certificate is contained in the message. 'Trusted' means that the CA certificate is stored in the PKI store and that it has already been trusted. 'Issuers' also means that the CA certificate is available in the PKI store, but the certificate is not trusted in this case.

CRL: indicates whether a CRL is available for the CA (column 4) or the sub CA (column 7).

SCA: indicates whether a sub CA certificate is available for the verification of the certificate chain. The three possible values 'Chain', 'Trusted' and 'Issuers' have the same meaning as in column CA.

Error: indicates whether the sub CA certificate (column 6) or the client certificate (column 10) is valid or not. 'Signature' means that the signature is incorrect; 'NotYet' means that the certificate is not valid yet; 'Expired' means that the certificate has already expired.

Revoked: indicates whether the sub CA certificate (column 8) or the client certificate (column 11) is revoked.

Cert: indicates whether the client certificate is trusted or not.

Configuration: indicates whether certain error classes are ignored. If 'IgnInvTime' is set, a certificate invalid in terms of time does not trigger an error. If 'IgnMissCRL' is set, a missing CRL does not trigger an error.

Server response: Expected status code with which the server should respond. If a deviation occurs here, the respective test will be marked as failed. If the fields are highlighted in colour, this means that the tests failed. For the blue cells, the cause for the error was outside the stack. The fields marked in yellow refer to deviations in the StatusCode without having an impact on IT security.

In the table, a minus sign means that no value is set.

N o.	Type	CA	CR L	SCA	Error	CRL	Revoked	Cert	Error	Revoked	Configuration	Server response
1	Self	-	-	-	-	-	-	Trusted	-	-	-	Good
2	Self	-	-	-	-	-	-	-	-	-	-	CertificateUntrusted
3	Self	-	-	-	-	-	-	Trusted	Signature	-	-	SecurityChecksFailed
4	Self	-	-	-	-	-	-	-	Signature	-	-	SecurityChecksFailed
5	Self	-	-	-	-	-	-	Trusted	NotYet	-	-	CertificateTimeInvalid
6	Self	-	-	-	-	-	-	-	NotYet	-	-	CertificateTimeInvalid
7	Self	-	-	-	-	-	-	Trusted	NotYet	-	IgnInvTime	Good
8	Self	-	-	-	-	-	-	-	NotYet	-	IgnInvTime	CertificateUntrusted
9	Self	-	-	-	-	-	-	Trusted	Expired	-	-	CertificateTimeInvalid
10	Self	-	-	-	-	-	-	-	Expired	-	-	CertificateTimeInvalid
11	Self	-	-	-	-	-	-	Trusted	Expired	-	IgnInvTime	Good
12	Self	-	-	-	-	-	-	-	Expired	-	IgnInvTime	CertificateUntrusted
13	Root	Issuers	yes	-	-	-	-	Trusted	-	-	-	Good
14	Root	Issuers	-	-	-	-	-	Trusted	-	-	IgnMissCRL	Good
15	Root	Issuers	-	-	-	-	-	Trusted	-	-	-	RevocationUnknown
16	Root	Issuers	yes	-	-	-	-	Trusted	-	yes	-	CertificateRevoked
17	Root	-	yes	-	-	-	-	Trusted	-	-	-	Good
18	Root	Trusted	yes	-	-	-	-	-	-	-	-	Good
19	Root	Trusted	-	-	-	-	-	-	-	-	IgnMissCRL	Good
20	Root	Trusted	-	-	-	-	-	-	-	-	-	RevocationUnknown
21	Root	Trusted	yes	-	-	-	-	-	-	yes	-	CertificateRevoked
22	Root	Trusted	yes	-	-	-	-	Trusted	-	-	-	Good
23	Root	Trusted	-	-	-	-	-	Trusted	-	-	IgnMissCRL	Good
24	Root	Trusted	-	-	-	-	-	Trusted	-	-	-	RevocationUnknown
25	Root	Trusted	yes	-	-	-	-	Trusted	-	yes	-	CertificateRevoked
26	Root	Chain	-	-	-	-	-	Trusted	-	-	IgnMissCRL	Good
27	Root	Chain	-	-	-	-	-	Trusted	-	-	-	RevocationUnknown
28	Root	Chain, Trusted	yes	-	-	-	-	Trusted	-	-	-	Good
29	Root	Chain,	yes	-	-	-	-	Trusted	-	-	-	Good

		Issuers										
30	Root	Issuers	yes	-	-	-	-	-	-	-	-	CertificateUntrusted
31	Root	Issuers	-	-	-	-	-	-	-	-	IgnMissCRL	CertificateUntrusted
32	Root	Issuers	-	-	-	-	-	-	-	-	-	CertificateUntrusted
33	Root	Issuers	yes	-	-	-	-	-	-	yes	-	CertificateUntrusted
34	Root	Chain, Trusted	yes	-	-	-	-	-	-	-	-	Good
35	Root	Chain, Trusted	-	-	-	-	-	-	-	-	IgnMissCRL	Good
36	Root	Chain, Issuers	yes	-	-	-	-	-	-	-	-	CertificateUntrusted
37	Root	Chain, Issuers	-	-	-	-	-	-	-	-	IgnMissCRL	CertificateUntrusted
38	Root	Chain	-	-	-	-	-	-	-	-	IgnMissCRL	CertificateUntrusted
39	Root	Trusted	yes	-	-	-	-	Trusted	NotYet	-	-	CertificateTimeInvalid
40	Root	Trusted	yes	-	-	-	-	-	NotYet	-	-	CertificateTimeInvalid
41	Root	Trusted	yes	-	-	-	-	Trusted	NotYet	-	IgnInvTime	Good
42	Root	Trusted	yes	-	-	-	-	-	NotYet	-	IgnInvTime	Good
43	Root	Trusted	yes	-	-	-	-	Trusted	Expired	-	-	CertificateTimeInvalid
44	Root	Trusted	yes	-	-	-	-	-	Expired	-	-	CertificateTimeInvalid
45	Root	Trusted	yes	-	-	-	-	Trusted	Expired	-	IgnInvTime	Good
46	Root	Trusted	yes	-	-	-	-	-	Expired	-	IgnInvTime	Good
47	Root	Issuers	yes	-	-	-	-	-	NotYet	-	IgnInvTime	CertificateUntrusted
48	Root	Issuers	yes	-	-	-	-	-	Expired	-	IgnInvTime	CertificateUntrusted
49	Secondary	Issuers	yes	Issuers	-	yes	-	Trusted	-	-	-	Good
50	Secondary	Issuers	yes	Trusted	-	yes	-	-	-	-	-	Good
51	Secondary	Trusted	yes	Issuers	-	yes	-	-	-	-	-	Good
52	Secondary	Issuers	yes	Issuers	-	yes	-	-	-	-	-	CertificateUntrusted
53	Secondary	-	-	Issuers	-	yes	-	Trusted	-	-	-	Good
54	Secondary	Issuers	yes	-	-	-	-	Trusted	-	-	-	Good
55	Secondary	-	-	Trusted	-	yes	-	-	-	-	-	Good

				d								
56	Secondary	Trusted	yes	-	-	-	-	-	-	-	-	CertificateUntrusted
57	Secondary	Issuers	-	Issuers	-	yes	-	Trusted	-	-	-	IssuerRevocationUnknown
58	Secondary	Issuers	-	Trusted	-	yes	-	-	-	-	-	IssuerRevocationUnknown
59	Secondary	Trusted	-	Issuers	-	yes	-	-	-	-	-	IssuerRevocationUnknown
60	Secondary	Issuers	yes	Issuers	-	-	-	Trusted	-	-	-	RevocationUnknown
61	Secondary	Issuers	yes	Trusted	-	-	-	-	-	-	-	RevocationUnknown
62	Secondary	Trusted	yes	Issuers	-	-	-	-	-	-	-	RevocationUnknown
63	Secondary	Issuers	-	Issuers	-	yes	-	Trusted	-	-	IgnMissCRL	Good
64	Secondary	Issuers	-	Trusted	-	yes	-	-	-	-	IgnMissCRL	Good
65	Secondary	Trusted	-	Issuers	-	yes	-	-	-	-	IgnMissCRL	Good
66	Secondary	Issuers	yes	Issuers	-	-	-	Trusted	-	-	IgnMissCRL	Good
67	Secondary	Issuers	yes	Trusted	-	-	-	-	-	-	IgnMissCRL	Good
68	Secondary	Trusted	yes	Issuers	-	-	-	-	-	-	IgnMissCRL	Good
69	Secondary	Issuers	-	Issuers	-	-	-	Trusted	-	-	IgnMissCRL	Good
70	Secondary	Issuers	-	Trusted	-	-	-	-	-	-	IgnMissCRL	Good
71	Secondary	Trusted	-	Issuers	-	-	-	-	-	-	IgnMissCRL	Good
72	Secondary	Issuers	yes	Issuers	-	yes	yes	Trusted	-	-	-	CertificateIssuerRevoked
73	Secondary	Issuers	yes	Trusted	-	yes	yes	Trusted	-	-	-	CertificateIssuerRevoked
74	Secondary	Trusted	yes	Issuers	-	yes	yes	Trusted	-	-	-	CertificateIssuerRevoked
75	Secondary	Issuers	yes	Issuers	-	yes	-	Trusted	-	yes	-	CertificateRevoked
76	Secondary	Issuers	yes	Trusted	-	yes	-	Trusted	-	yes	-	CertificateRevoked
77	Secondary	Trusted	yes	Issuers	-	yes	-	Trusted	-	yes	-	CertificateRevoked
78	Secondary	Chain	-	Chain	-	-	-	Trusted	-	-	IgnMissCRL	Good
79	Secondary	Chain	-	Chain, Truste	-	-	-	-	-	-	IgnMissCRL	Good

				d								
80	Secondary	Chain, Trusted	-	Chain	-	-	-	-	-	-	IgnMissCRL	Good
81	Secondary	Trusted	-	Trusted	-	-	-	-	-	-	IgnMissCRL	Good
82	Secondary	Trusted	-	Trusted	-	-	-	Trusted	-	-	IgnMissCRL	Good
83	Secondary	Issuers	yes	Issuers	NotYet	yes	-	Trusted	-	-	-	CertificateIssuer-TimeInvalid
84	Secondary	Issuers	yes	Issuers	NotYet	yes	-	Trusted	-	-	IgnInvTime	Good
85	Secondary	Issuers	yes	Issuers	NotYet	-	-	-	-	-	-	CertificateIssuer-TimeInvalid
86	Secondary	Issuers	yes	Issuers	NotYet	-	-	-	-	-	IgnInvTime	CertificateUntrusted
87	Secondary	Issuers	yes	Issuers	Expired	yes	-	Trusted	-	-	-	CertificateIssuer-TimeInvalid
88	Secondary	Issuers	yes	Issuers	Expired	yes	-	Trusted	-	-	IgnInvTime	Good
89	Secondary	Issuers	yes	Issuers	Expired	-	-	-	-	-	-	CertificateIssuer-TimeInvalid
90	Secondary	Issuers	yes	Issuers	Expired	-	-	-	-	-	IgnInvTime	CertificateUntrusted
91	Secondary	Chain	-	Chain, Issuers	-	-	-	-	-	-	IgnMissCRL	CertificateUntrusted
92	Secondary	Chain, Issuers	-	Chain	-	-	-	-	-	-	IgnMissCRL	CertificateUntrusted

Table 34: Certificate tests with expected result

Supplements to Section 8.6.2 Choice of fuzzing tests:

In Table 35 below, all fuzzing tests that were carried out are listed. The last column specifies in brief which of the four objectives mentioned in Section 8.6.2 was pursued for the respective test series. The column 'Environment' describes which client and which server were used, with the following abbreviations being used for reasons of clarity:

- 'debug' stands for:
 - Client: *Peach* v3.1.124.0
 - Server: *Valgrind* with debug version of the server
- 'release' stands for:
 - Client: *Peach* v3.1.124.0
 - Server: release version of the server

Test series no.	Test series name	TestNo.	TestName	securityMode	Environment	Test objective
001	sequential individual parameter fuzzing	01	HEL message fuzzing	–	debug	Dynamic code analysis
001	sequential individual parameter fuzzing	02	OPN message fuzzing	None	debug	Dynamic code analysis
001	sequential individual parameter fuzzing	03	CreateSession message fuzzing	None	debug	Dynamic code analysis
001	sequential individual parameter fuzzing	04	ActivateSession message fuzzing	None	debug	Dynamic code analysis
001	sequential individual parameter fuzzing	05	CloseSession message fuzzing	None	debug	Dynamic code analysis
001	sequential individual parameter fuzzing	06	CloseChannel message fuzzing	None	debug	Dynamic code analysis
001	sequential individual parameter fuzzing	07	FindServers message fuzzing	None	debug	Dynamic code analysis
001	sequential individual parameter fuzzing	08	GetEndpoints message fuzzing	None	debug	Dynamic code analysis
002	random individual parameter fuzzing	01	HEL message fuzzing (5,000 iterations per choice)	--	debug	Dynamic code analysis
002	random individual parameter fuzzing	02	OPN message fuzzing (5,000	None	debug	Dynamic code

			iterations per choice)			analysis
002	random individual parameter fuzzing	03	CreateSession message fuzzing (5,000 iterations per choice)	None	debug	Dynamic code analysis
002	random individual parameter fuzzing	04	ActivateSession message fuzzing (5,000 iterations per choice)	None	debug	Dynamic code analysis
002	random individual parameter fuzzing	05	CloseSession message fuzzing (5,000 iterations per choice)	None	debug	Dynamic code analysis
002	random individual parameter fuzzing	06	CloseChannel message fuzzing (5,000 iterations per choice)	None	debug	Dynamic code analysis
002	random individual parameter fuzzing	07	FindServers message fuzzing (5,000 iterations per choice)	None	debug	Dynamic code analysis
002	random individual parameter fuzzing	08	GetEndpoints message fuzzing (5,000 iterations per choice)	None	debug	Dynamic code analysis
003	random multiple parameter fuzzing	01	HEL message fuzzing (5,000 iterations per choice)	--	debug	Dynamic code analysis
003	random multiple parameter fuzzing	02	OPN message fuzzing (5,000 iterations per choice)	None	debug	Dynamic code analysis
003	random multiple parameter fuzzing	03	CreateSession message fuzzing (5,000 iterations per choice)	None	debug	Dynamic code analysis
003	random multiple parameter fuzzing	04	ActivateSession message fuzzing (5,000 iterations per choice)	None	debug	Dynamic code analysis
003	random multiple parameter fuzzing	05	CloseSession message fuzzing (5,000 iterations per choice)	None	debug	Dynamic code analysis
003	random multiple parameter fuzzing	06	CloseChannel message fuzzing (5,000 iterations per choice)	None	debug	Dynamic code analysis
003	random multiple parameter fuzzing	07	FindServers message fuzzing (5,000 iterations per choice)	None	debug	Dynamic code analysis
003	random multiple parameter fuzzing	08	GetEndpoints message fuzzing (5,000 iterations per choice)	None	debug	Dynamic code analysis
004	sequential individual parameter fuzzing	01	HEL message fuzzing	--	release	Production server
004	sequential individual parameter fuzzing	02	OPN message fuzzing	None	release	Production server
004	sequential individual parameter fuzzing	03	CreateSession message fuzzing	None	release	Production server
004	sequential individual parameter fuzzing	04	ActivateSession message fuzzing	None	release	Production server

004	sequential individual parameter fuzzing	05	CloseSession message fuzzing	None	release	Production server
004	sequential individual parameter fuzzing	06	CloseChannel message fuzzing	None	release	Production server
004	sequential individual parameter fuzzing	07	FindServers message fuzzing	None	release	Production server
004	sequential individual parameter fuzzing	08	GetEndpoints message fuzzing	None	release	Production server
005	random individual parameter fuzzing	01	HEL message fuzzing (5,000 iterations per choice)	--	release	Production server
005	random individual parameter fuzzing	02	OPN message fuzzing (5,000 iterations per choice)	None	release	Production server
005	random individual parameter fuzzing	03	CreateSession message fuzzing (5,000 iterations per choice)	None	release	Production server
005	random individual parameter fuzzing	04	ActivateSession message fuzzing (5,000 iterations per choice)	None	release	Production server
005	random individual parameter fuzzing	05	CloseSession message fuzzing (5,000 iterations per choice)	None	release	Production server
005	random individual parameter fuzzing	06	CloseChannel message fuzzing (5,000 iterations per choice)	None	release	Production server
005	random individual parameter fuzzing	07	FindServers message fuzzing (5,000 iterations per choice)	None	release	Production server
005	random individual parameter fuzzing	08	GetEndpoints message fuzzing (5,000 iterations per choice)	None	release	Production server
006	random multiple parameter fuzzing	01	HEL message fuzzing (5,000 iterations per choice)	--	release	Production server
006	random multiple parameter fuzzing	02	OPN message fuzzing (5,000 iterations per choice)	None	release	Production server
006	random multiple parameter fuzzing	03	CreateSession message fuzzing (5,000 iterations per choice)	None	release	Production server
006	random multiple parameter fuzzing	04	ActivateSession message fuzzing (5,000 iterations per choice)	None	release	Production server
006	random multiple parameter fuzzing	05	CloseSession message fuzzing (5,000 iterations per choice)	None	release	Production server
006	random multiple parameter fuzzing	06	CloseChannel message fuzzing (5,000 iterations per choice)	None	release	Production server
006	random multiple parameter fuzzing	07	FindServers message fuzzing (5,000 iterations per choice)	None	release	Production server

			iterations per choice)			
006	random multiple parameter fuzzing	08	GetEndpoints message fuzzing (5,000 iterations per choice)	None	release	Production server
007	sequential individual parameter fuzzing	02	OPN message fuzzing	SignAndEncrypt	debug	Dynamic code analysis
007	sequential individual parameter fuzzing	03	CreateSession message fuzzing	SignAndEncrypt	debug	Dynamic code analysis
007	sequential individual parameter fuzzing	04	ActivateSession message fuzzing	SignAndEncrypt	debug	Dynamic code analysis
007	sequential individual parameter fuzzing	05	CloseSession message fuzzing	SignAndEncrypt	debug	Dynamic code analysis
007	sequential individual parameter fuzzing	06	CloseChannel message fuzzing	SignAndEncrypt	debug	Dynamic code analysis
007	sequential individual parameter fuzzing	07	FindServers message fuzzing	SignAndEncrypt	debug	Dynamic code analysis
007	sequential individual parameter fuzzing	08	GetEndpoints message fuzzing	SignAndEncrypt	debug	Dynamic code analysis
009	random multiple parameter fuzzing (long run)	02	OPN message fuzzing	SignAndEncrypt	debug	Increase in code coverage
009	random multiple parameter fuzzing (long run)	03	CreateSession message fuzzing	SignAndEncrypt	debug	Increase in code coverage
009	random multiple parameter fuzzing (long run)	04	ActivateSession message fuzzing	None	debug	Increase in code coverage
009	random multiple parameter fuzzing (long run)	07	FindServers message fuzzing	None	debug	Increase in code coverage
015	sequential individual parameter fuzzing	02	OPN message fuzzing	Sign	debug	Verification of the assumptions regarding the code coverage
015	sequential individual parameter fuzzing	03	CreateSession message fuzzing	Sign	debug	Verification of the assumptions regarding the code coverage

015	sequential individual parameter fuzzing	04	ActivateSession message fuzzing	Sign	debug	Verification of the assumptions regarding the code coverage
015	sequential individual parameter fuzzing	05	CloseSession message fuzzing	Sign	debug	Verification of the assumptions regarding the code coverage
015	sequential individual parameter fuzzing	06	CloseChannel message fuzzing	Sign	debug	Verification of the assumptions regarding the code coverage
015	sequential individual parameter fuzzing	07	FindServers message fuzzing	Sign	debug	Verification of the assumptions regarding the code coverage
015	sequential individual parameter fuzzing	08	GetEndpoints message fuzzing	Sign	debug	Verification of the assumptions regarding the code coverage
016	random individual parameter fuzzing	02	OPN message fuzzing (10,000 iterations per choice)	None	debug	Verification of the assumptions regarding the code coverage
016	random individual parameter fuzzing	04	ActivateSession message fuzzing (10,000 iterations per choice)	None	debug	Verification of the assumptions regarding the code coverage
016	random individual parameter fuzzing	07	FindServers message fuzzing (10,000 iterations per choice)	None	debug	Verification of the assumptions regarding the code coverage

Table 35: List of fuzzing tests performed

Annex B: Compiler flags

Complete list of the compiler flags for the debug and release versions of the server.

The files of the debug version were compiled with the following command:

```
/usr/bin/gcc -Duastack_EXPORTS -DOPCUA_P_TIMER_NO_OF_TIMERS=50
-DOPCUA_GUID_STRING_USE_CURLYBRACE=1 -DOPCUA_SUPPORT_PKI=1
-DOPCUA_GUID_STRING_USE_CURLYBRACE=1 -DOPCUA_SUPPORT_SECURITYPOLICY_BA-
SIC128RSA15=1 -DOPCUA_SUPPORT_SECURITYPOLICY_BASIC256=1
-DOPCUA_SUPPORT_SECURITYPOLICY_NONE=1 -DOPCUA_HAVE_HTTPS=0
-DOPCUA_P_SOCKETMANAGER_SUPPORT_SSL=0 -DOPCUA_SUPPORT_PKI_WIN32=0 -DU-
ASERVER_SERVICES_HISTORYREAD=1 -DUASERVER_SERVICES_HISTORYUPDATE=1 -DU-
ASERVER_SERVICES_CALL=1 -DUASERVER_SUPPORT_EVENTS=1 -DOPCUA_MULTI-
THREADED=0 -DOPCUA_USE_SYNCHRONISATION=1 -DUASERVER_SUPPORT_AUTHORIZA-
TION=1 -DUASERVER_SUPPORT_AUTHENTICATION_INTERNAL=1
-DOPCUA_USE_STATIC_PLATFORM_INTERFACE=1 -DOPCUA_HAVE_CLIENTAPI=1
-DOPCUA_HAVE_SERVERAPI=1 -DUASERVER_SUPPORT_AUTHENTICATION_PAM=0 -DU-
ASERVER_SUPPORT_AUTHENTICATION_SASL=0 -DUASERVER_SUPPORT_DISCOVERY=1
-DOPCUA_GUID_STRING_USE_CURLYBRACE=1 -DUA_STACK_BUILD_DLL -DOPCUA_SUP-
PORT_PKI=1 -DOPCUA_ENCODEABLE_OBJECT_COMPARE_SUPPORTED=0 -DOPCUA_ENCODE-
ABLE_OBJECT_COPY_SUPPORTED=0 -DOPCUA_SUPPORT_SECURITYPOLICY_BA-
SIC128RSA15=1 -DOPCUA_SUPPORT_SECURITYPOLICY_BASIC256=1
-DOPCUA_SUPPORT_SECURITYPOLICY_NONE=1 -DOPCUA_HAVE_HTTPS=0
-DOPCUA_P_SOCKETMANAGER_SUPPORT_SSL=0 -DOPCUA_MULTITHREADED=0
-DOPCUA_USE_SYNCHRONISATION=1 -DOPCUA_USE_STATIC_PLATFORM_INTERFACE=1
-DOPCUA_HAVE_CLIENTAPI=1 -DOPCUA_HAVE_SERVERAPI=1
-DOPCUA_TCPLISTENER_MAXCONNECTIONS=100 -DOPCUA_P_SOCKETMANAGER_NUMBEROF-
SOCKETS=102 -D_DEBUG -DHAVE_TIMEGM -DOPCUA_HAVE_OPENSSL -march=i686 -fPIC
-fno-strict-aliasing -fprofile-arcs -ftest-coverage -Wextra -Wno-unused-
but-set-variable -g -fPIC
-I/root/opcu/ascalab_webdav/sdk/src/uastack/platforms/linux
-I/root/opcu/ascalab_webdav/sdk/src/uastack/core
-I/root/opcu/ascalab_webdav/sdk/src/uastack/stackcore -I/root/opcu/as-
colab_webdav/sdk/src/uastack/securechannel
-I/root/opcu/ascalab_webdav/sdk/src/uastack/transport/tcp
-I/root/opcu/ascalab_webdav/sdk/src/uastack/proxystub/clientproxy
-I/root/opcu/ascalab_webdav/sdk/src/uastack/proxystub/serverstub -Wall
-fno-strict-aliasing -Wno-format -Wfloat-equal -Wextra -o <file name>.o
-c <file name>
```

The options starting with `-DOPCUA` and `-DUASERVER` are used to set the OPC UA-internal functionality.

The files of the release version of the server were compiled as follows:

```
/usr/bin/gcc -Duastack_EXPORTS -DOPCUA_P_TIMER_NO_OF_TIMERS=50
-DOPCUA_GUID_STRING_USE_CURLYBRACE=1 -DOPCUA_SUPPORT_PKI=1
-DOPCUA_GUID_STRING_USE_CURLYBRACE=1 -DOPCUA_SUPPORT_SECURITYPOLICY_BASIC128RSA15=1 -DOPCUA_SUPPORT_SECURITYPOLICY_BASIC256=1
-DOPCUA_SUPPORT_SECURITYPOLICY_NONE=1 -DOPCUA_HAVE_HTTPS=0
-DOPCUA_P_SOCKETMANAGER_SUPPORT_SSL=0 -DOPCUA_SUPPORT_PKI_WIN32=0 -DUASERVER_SERVICES_HISTORYREAD=1 -DUASERVER_SERVICES_HISTORYUPDATE=1 -DUASERVER_SERVICES_CALL=1 -DUASERVER_SUPPORT_EVENTS=1 -DOPCUA_MULTITHREADED=0 -DOPCUA_USE_SYNCHRONISATION=1 -DUASERVER_SUPPORT_AUTHORIZATION=1 -DUASERVER_SUPPORT_AUTHENTICATION_INTERNAL=1
-DOPCUA_USE_STATIC_PLATFORM_INTERFACE=1 -DOPCUA_HAVE_CLIENTAPI=1
-DOPCUA_HAVE_SERVERAPI=1 -DUASERVER_SUPPORT_AUTHENTICATION_PAM=0 -DUASERVER_SUPPORT_AUTHENTICATION_SASL=0 -DUASERVER_SUPPORT_DISCOVERY=1
-DOPCUA_GUID_STRING_USE_CURLYBRACE=1 -DUA_STACK_BUILD_DLL -DOPCUA_SUPPORT_PKI=1 -DOPCUA_ENCODEABLE_OBJECT_COMPARE_SUPPORTED=0 -DOPCUA_ENCODEABLE_OBJECT_COPY_SUPPORTED=0 -DOPCUA_SUPPORT_SECURITYPOLICY_BASIC128RSA15=1 -DOPCUA_SUPPORT_SECURITYPOLICY_BASIC256=1
-DOPCUA_SUPPORT_SECURITYPOLICY_NONE=1 -DOPCUA_HAVE_HTTPS=0
-DOPCUA_P_SOCKETMANAGER_SUPPORT_SSL=0 -DOPCUA_MULTITHREADED=0
-DOPCUA_USE_SYNCHRONISATION=1 -DOPCUA_USE_STATIC_PLATFORM_INTERFACE=1
-DOPCUA_HAVE_CLIENTAPI=1 -DOPCUA_HAVE_SERVERAPI=1
-DOPCUA_TCPLISTENER_MAXCONNECTIONS=100 -DOPCUA_P_SOCKETMANAGER_NUMBEROF_SOCKETS=102 -DHAVE_TIMEGM -DOPCUA_HAVE_OPENSSL -march=i686 -fPIC -fno-strict-aliasing -Wextra -Wno-unused-but-set-variable -O3 -DNDEBUG -fPIC
-I/root/opcu/ascolab_webdav/sdk/src/uastack/platforms/linux
-I/root/opcu/ascolab_webdav/sdk/src/uastack/core
-I/root/opcu/ascolab_webdav/sdk/src/uastack/stackcore -I/root/opcu/ascolab_webdav/sdk/src/uastack/securechannel
-I/root/opcu/ascolab_webdav/sdk/src/uastack/transport/tcp
-I/root/opcu/ascolab_webdav/sdk/src/uastack/proxystub/clientproxy
-I/root/opcu/ascolab_webdav/sdk/src/uastack/proxystub/serverstub -Wall
-fno-strict-aliasing -Wno-format -Wfloat-equal -Wextra -o <file name>.o
-c <file name>
```

Annex C: References and bibliography

- [1] BSI, ICS Security Kompendium,
https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/ICS/ICS-Security_kompendium_pdf.pdf
- [2] OPC Foundation, OPC UA Specification: Part 01 - Overview
- [3] OPC Foundation, OPC UA Specification: Part 02 - Security Model
- [4] OPC Foundation, OPC UA Specification: Part 03 - Address Space Model
- [5] OPC Foundation, OPC UA Specification: Part 04 - Services
- [6] OPC Foundation, OPC UA Specification: Part 05 - Information Model
- [7] OPC Foundation, OPC UA Specification: Part 06 - Mappings
- [8] OPC Foundation, OPC UA Specification: Part 07 - Profiles
- [9] OPC Foundation, OPC UA Specification: Part 08 - Data Access
- [10] OPC Foundation, OPC UA Specification: Part 09 - Alarms and Conditions
- [11] OPC Foundation, OPC UA Specification: Part 10 - Programs
- [12] OPC Foundation, OPC UA Specification: Part 11 - Historical Access
- [13] OPC Foundation, OPC UA Specification: Part 12 - Discovery
- [14] OPC Foundation, OPC UA Specification: Part 13 - Aggregates
- [15] Forum for Incident Response and Security Teams, A Complete Guide to the Common Vulnerability Scoring System
- [16] Cisco Systems, SCADA OPC-UA TCP Protocol Issues, <http://tools.cisco.com/security/center/viewAlert.x?alertId=34634>
- [17] Schneider Electric, OPC Factory Server V3.40 - Service Pack3 - Update Description,
http://www.schneider-electric.com/download/IN/EN/details/141070211-OPC-Factory-Server-V340--Service-Pack-3/?showAsIframe=true&reference=OFS_3_40_2811-%28SP3%29
- [18] Dale G. Peterson, OPC UA Assessment Series,
<https://www.digitalbond.com/blog/2008/08/14/opc-ua-assessment-series-part-1/>
- [19] CSWG STANDARDS SUBGROUP, Document Reviews - Documents Assessed in Q3 2012,
https://collaborate.nist.gov/twiki-sggrid/bin/view/SmartGrid/CSCTGStandards#Documents_Assessed_in_Q3_2012
- [20] Melanie Gallinat, Stefan Hausmann, Markus Köster, Stefan Heiss, OPC-UA: Ein kritischer Vergleich der IT-Sicherheitsoptionen
- [21] BSI, TR-02102-1 "Kryptographische Verfahren: Empfehlungen und Schlüssellängen"
- [22] NIST, Elaine Barker and Allen Roginsky, Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths
- [23] BSI, TR-02102-2 Verwendung von Transport Layer Security (TLS)
- [24] ISO/IEC, Information Security Management Systems standards, "ISO/IEC 27000 series"
- [25] BSI, OPC UA Protokollbetrachtung,
<https://www.bsi.bund.de/DE/Publikationen/Studien/OPCUA/opcu.html>

Annex D: Table of abbreviations and list of tables

Figure 1: Use case in an exemplary OPC UA environment.....	10
Figure 2: Overview of the test environment.....	16
Figure 3: Overview of the test environment.....	16
Figure 4: Communication channels with securityMode Sign and SignAndEncrypt.....	56
Table 1: Documents of the OPC UA Specification.....	8
Table 2: Sources of information for performing security analyses.....	9
Table 3: CVSS Metric Groups (source: [15] page 3).....	14
Table 4: CVSS parameters used for assessing the criticality and their scores.....	16
Table 5: Distribution of VM instances.....	18
Table 6: List of key tools used for the tests.....	18
Table 7: Main findings of the inventory.....	21
Table 8: Questions and comments with respect to the specification.....	27
Table 9: Essential editorial comments and security-critical observations.....	29
Table 10: Mapping of threats from Part 2 to STRIDE.....	31
Table 11: Comparison of IT security definitions.....	32
Table 12: Security objectives versus threat types according to the specification Explanation: The crosses in brackets are indirect threats.....	34
Table 13: Security objectives versus threat types according to the current analysis Explanation: The crosses in brackets are indirect threats.....	34
Table 14: Main findings of the review of security objectives and threat types.....	35
Table 15: CVSS criticality scores for OPC UA.....	36
Table 16: Potential threats for the OPC UA communication.....	41
Table 17: Potential threats for the OPC UA infrastructure.....	43
Table 18: Particularly critical threats.....	44
Table 19: Effectiveness of the OPC UA measures.....	46
Table 20: Main findings regarding the design.....	49
Table 21: Main findings of the overall analysis.....	53
Table 22: Certificate validation steps (source: [5] Table 101).....	57
Table 23: Failed certificate tests.....	59
Table 24: List of the fuzzing test evaluated completely in the dynamic code analysis.....	66
Table 25: List of the fuzzing test evaluated partially in the dynamic code analysis.....	66
Table 26: Occurrence of the memory leak in the Valgrind logs.....	67
Table 27: Occurrence of the error when parsing certificate chains in the Valgrind logs.....	68
Table 28: Occurrence of the error when sending a serviceFault in the Valgrind logs.....	69
Table 29: Occurrence of the access to an uninitialised variable in the Valgrind logs.....	70
Table 30: List of test series for which the code coverage was measured.....	71
Table 31: Code coverage achieved depending on the test series performed (assumption 1).....	72
Table 32: Code coverage achieved depending on the total number of iterations (assumption 2).....	72
Table 33: Code coverage achieved depending on the number of iterations per choice (assumption 3).....	72
Table 34: Certificate tests with expected result.....	80
Table 35: List of fuzzing tests performed.....	85
Table 36: Table of abbreviations.....	90

Annex E: Table of abbreviations

BSI	Federal Office for Information Security
CA	Certificate Authority
CRL	Certificate Revocation List
CSWG	Cyber Security Working Group
CVSS	Common Vulnerability Scoring System
DMZ	Demilitarized Zone
HMI	Human Machine Interface
ICS	Industrial Control System
LDS	Local Discovery Server
LDS-ME	LDS with multicast extension (Zeroconf)
MES	Manufacturing Execution System
OPC DA	OPC Data Access (Classic COM based OPC)
OPC UA	Open Platform Communications Unified Architecture
PKI	Public Key Infrastructure
SCADA	Supervisory Control and Data Acquisition
SDK	Software Development Kit
SGIP	Smart Grid Interoperability Panels
UASC	UA Secure Conversation

Table 36: Table of abbreviations