Bundesamt
für Sicherheit in der
Informationstechnik

Sicherheitsanalyse OPC UA

25.04.2016

Autoren

Die Studie Sicherheitsanalyse von OPC UA wurde durch das Bundesamt für Sicherheit in der Informationstechnik (BSI) beauftragt und unter Federführung des TÜV SÜD Rail als Konsortialführer im Zeitraum vom 14.1.2015 bis 02.12.2015 durchgeführt. Weitere im Konsortium beteiligte Firmen und Personen waren in alphabetischer Reihenfolge:

ascolab

- Herr Damm
- Herr Gappmeier
- Herr Zugfil

SIGNON

- Herr Plöb

TÜV SÜD

- Herr Fiat (Projektleiter)
- Herr Störtkuhl

Danksagung

Die Autoren danken der OPC-Foundation für die offene, konstruktive Zusammenarbeit. Alle in dieser Studie dargestellten Verbesserungsvorschläge wurden durch die OPC-Foundation aufgegriffen und intensiv diskutiert. Diese Studie führte zu einer Anpassung von Spezifikation und Referenzimplementierung, die in die nächsten Aktualisierungen einfließen wird.

Eine Übersicht über die Rückmeldungen der OPC Foundation zu dieser Studie ist nachfolgend zu finden:

https://opcfoundation.org/security/

Bundesamt für Sicherheit in der Informationstechnik
Postfach 20 03 63
53133 Bonn
Tel.: +49 22899 9582-0
E-Mail: ics-sec@bsi.bund.de
Internet: https://www.bsi.bund.de

Inhalt

1 Managementzusammenfassung

OPC Unified Architecture (OPC UA) ist der zentrale Standard in der Umsetzung der Zukunftsstrategie Industrie 4.0 und wird bereits jetzt bei der Vernetzung vorhandener Industrieanlagen immer häufiger eingesetzt. Sicherheit war von Anfang an eines der Kernziele von OPC UA als Protokoll der Zukunft: Es bietet die Möglichkeit, herstellerübergreifend Netze über verschiedene Einsatzebenen von der Steuerungs- bis hin zur Unternehmensebene zu verbinden. Außerdem bringt OPC UA, im Gegensatz zu vielen anderen Industrieprotokollen, integrierte Sicherheitsfunktionalität zur Absicherung der Kommunikation mit.

Ziele und Vorgehen

Ziel der aktuellen Studie war es, eine Bestandsaufnahme der IT Sicherheit von OPC UA durchzuführen. Zu diesem Zweck wurden im wesentlichen zwei Analysen durchgeführt: Im ersten Teil des Projekts wurde die Spezifikation des OPC UA Protokolls in der Version 1.02 auf systematische Fehler analysiert. Diese Analyse wurde in folgende Teilschritte gegliedert:

- Analyse bereits durchgeführter Untersuchungen der IT Sicherheit von OPC UA
- Bedrohungsanalyse (Analyse der Schutzziele und Bedrohungen, Analyse der Bedrohungen und Maßnahmen)
- Analyse der OPC UA Spezifikation im Detail mit den Schwerpunkten auf den Parts 2, 4, 6, 7 und 12

Für die Analyse der Spezifikation wurden keine formalen oder semiformalen Methoden eingesetzt. Die OPC UA Kommunikation wurde bezüglich der Dienste SecureChannel, Session und Discovery (Bestandteile des Kommunikationsstacks der OPC Foundation) gemäß Spezifikation jedoch bis auf die Ebene der Parameter systematisch erfasst und analysiert.

Aufbauend auf diese Spezifikationsanalyse wurde im zweiten Teil des Projekts die von der OPC Foundation angebotene Referenzimplementierung in ANSI C des OPC UA Kommunikationsstacks in der Version 1.02.344.5 folgenden Sicherheitstests unterzogen:

- Zertifikatstests
- statische Codeanalyse
- Fuzzing
- dynamische Codeanalyse

Wichtigste Ergebnisse

Die durchgeführte *Spezifikationsanalyse* hat gezeigt, dass OPC UA, im Gegensatz zu den meisten anderen Industrieprotokollen, ein hohes Maß an Sicherheit bietet.

Es konnten keine systematischen Fehler entdeckt werden.

Bei der *Analyse der Referenzimplementierung* wurden im Wesentlichen folgende Probleme festgestellt:

- Ein wichtiger Mechanismus zum Schutz gegen replay Angriffe fehlt, weil die sequenceNumber nicht ausgewertet wird.
- Speicherlecks können für Denial of Service Angriffe genutzt werden.
- Fehler bei Zertifikatstests, die – abhängig von der verwendeten Rahmenapplikation – evtl. ausgenutzt werden können
- Keine umfassende Dokumentation über implementierte (Sicherheits-)funktionalitäten im OPC UA Kommunikationsstack

Nichtsdestotrotz lief der Stack während aller Tests sehr stabil, denn es wurden keine Abstürze beobachtet.

Empfohlene Maßnahmen

Bei der Absicherung der Kommunikation mit dem OPC UA Protokoll sind folgende drei Einstellungen von zentraler Bedeutung:

- **securityMode:** Der securityMode sollte 'Sign' (signieren von Nachrichten) oder 'SignAndEncrypt' (signieren und verschlüsseln von Nachrichten) sein. Damit wird unter anderem eine Authentifizierung auf Applikationsebene erzwungen. securityMode 'None' bietet keinerlei Schutz. securityMode 'SignAndEncrypt' ist dann zu verwenden, wenn über Integrität hinaus, vertrauliche Daten zu schützen sind.
- **Wahl der kryptographischen Algorithmen:** Die sicherste securityPolicy '`Basic256Sha256`' sollte gewählt werden, sofern dies technisch möglich ist. Die schwächsten securityPolicies verwenden teilweise veraltete Algorithmen und sollten nicht eingesetzt werden.
- **Benutzerauthentifizierung:** Die Möglichkeit der Anmeldung mit der Kennung 'anonymous' sollte unterbunden werden, weil diese keinen Schutz bietet. Zum einen ist bei Verwendung dieser generischen Kennung nicht nachvollziehbar, wer auf Serverseite z. B. Daten oder die Konfiguration geändert hat. Zum anderen könnte ein Angreifer diese Kennung mißbrauchen, um unerlaubt Daten zu lesen oder zu schreiben, falls keine ausreichende Eingrenzung der Rechte der 'anonymous' Kennung konfiguriert wurde.

Neben der unmittelbaren sicheren Konfiguration der Kommunikation selbst sind weitere, zusätzliche Maßnahmen zum Schutz der Infrastruktur erforderlich. In dieser Studie wird davon ausgegangen, dass der Betreiber von OPC UA Kommunikationen in Automatisierungs- und Steuerungsanlagen Best Practice Ansätze, wie sie im ICS Security Kompendium [1] beschrieben sind, umgesetzt hat, betreibt und ständig verbessert.

In dieser Studie wurde detailliert auf verschiedene Aspekte der IT Sicherheit des OPC UA Protokolls eingegangen: Es wurde sowohl die Spezifikation auf systematische Fehler geprüft, als auch die Referenzimplementierung des Kommunikationsstacks mit verschiedenen Werkzeugen untersucht. So ist ein präziseres Bild entstanden, welche Punkte in der Spezifikation und der Referenzimplementierung verbesserungsbedürftig sind und worauf zu achten ist, um ein hohes Maß an IT Sicherheit beim Einsatz von OPC UA zu erzielen. Außerdem wird ein Ausblick auf weitere Themen gegeben, die in weiterführenden Untersuchungen vertieft werden könnten.

2 Gegenstand der Analyse

2.1 Untersuchungsgegenstand

Der Untersuchungsgegenstand des Projekts „Sicherheitsanalyse von OPC UA“ ist das Protokoll OPC UA in der Spezifikation in der Version 1.02 (1.02.47 für Part 12) und eine Referenzimplementierung der OPC Foundation des Kommunikationsstacks in ANSI C in der Version 1.02.344.5.

Die Referenzimplementierung des Kommunikationsstacks ist alleine nicht lauffähig und wurde deshalb ohne Veränderungen des Stacks in eine UA Serverapplikation eingebunden, um Tests durchführen zu können.

Das OPC UA Protokoll wird durch die in Tabelle 1 genannten Dokumente spezifiziert.

Nr.	Titel
[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

Tabelle 1: Dokumente der OPC UA Spezifikation

Im Rahmen dieser Studie wird an manchen Stellen auf die Folgeversion 1.03 der Spezifikation verwiesen. Neun von 13 Parts der neuen Spezifikation wurden im Juli 2015 veröffentlicht. Die fehlenden vier Teile liegen zu diesem Zeitpunkt noch nicht als Veröffentlichung vor.

Am 10.11.2015 hat die OPC Foundation eine aktualisierte ANSI C Referenzimplementierung des Stacks mit der Versionsnummer 1.02.336 für die Spezifikationsversion 1.03 zur Verfügung gestellt.

2.2 Ziele der Analyse

Mit dem Projekt „Sicherheitsanalyse von OPC UA“ wurde das generelle Ziel verfolgt, belastbare Aussagen über die IT Sicherheit des OPC UA Protokolls treffen zu können. Hauptziele des Projekts waren somit einerseits die OPC UA Spezifikation bzgl. IT Sicherheit zu überprüfen und zu validieren. Andererseits wurde eine Referenzimplementierung des Kommunikationsstacks mittels ausführlicher Tests, insbesondere auch auf der Basis der Erkenntnisse der Analyse der OPC UA Spezifikation, geprüft. So wurden sowohl etwaige systematische Schwachstellen als auch Schwachstellen der Implementierung identifiziert und bewertet. Gegenmaßnahmen wurden, wo immer möglich, skizziert.

Aus den dargestellten Hauptzielen leiteten sich folgende Detailziele ab:

- OPC UA Spezifikation
  - Herausarbeitung und Darstellung der bzgl. IT Sicherheit wichtigsten Komponenten von OPC UA
  - Aufdecken von Sicherheitslücken
  - Aufdecken von Widersprüchen (auch zwischen den verschiedenen Teilen der Spezifikation)
  - Empfehlung von Verbesserungen
  - Sonstige Review-Ergebnisse wie Syntaxfehler, Fehler in Bildern, unverständliche oder zweideutige Textpassagen, usw.
- Test einer Referenzimplementierung des OPC UA Kommunikationsstacks
  - Identifizierung von Schwachstellen, wenn möglich mit Proof-of-Concept
  - Aussagen über die Robustheit des Stacks
  - Empfehlungen von Gegenmaßnahmen, wo immer möglich

Die genannten Ziele wurden mit Blick auf die Interessenvertreter

- BSI
- Betreiber von OPC UA Infrastrukturen
- OPC Foundation

gewählt, an die sich die hier vorliegende Studie im wesentlichen wendet.

2.3 Umfang der Analyse

Entsprechend den in Abschnitt 2.2 genannten Detailzielen bestand der Umfang der Analyse aus folgenden zwei größeren Teilaufgaben:

- Analyse der OPC UA Spezifikation bzgl. IT Sicherheit
- Durchführung von Sicherheitstests auf eine Referenzimplementierung des OPC UA Kommunikationsstacks der OPC Foundation

Für die Analyse der Spezifikation und die Sicherheitstests wurden die in Tabelle 2 angegebenen Dokumente herangezogen.

Nr.	Quelle	Titel
[1]	BSI	ICS Security Kompendium
[15]	FIRST (Forum for Incident Response and Security Teams)	CVSS v2.0 (Common Vulnerability Scoring System)

Tabelle 2: Informationsquellen für die Durchführung von Sicherheitsanalysen

2.3.1 Analyse der OPC UA Spezifikation

Die Analyse der Spezifikation wurde in folgende Teilschritte gegliedert:

- Bestandsaufnahme des Kenntnisstands bzgl. der IT Sicherheit von OPC UA
- Bedrohungsanalyse (Analyse der Schutzziele und Bedrohungen, Analyse der Bedrohungen und Maßnahmen)
- Analyse der OPC UA Spezifikation im Detail

Bestandsaufnahme des Kenntnisstands bzgl. der IT Sicherheit von OPC UA

Als Grundlage für die Spezifikationsanalyse wurde eine Bestandsaufnahme des aktuellen Wissensstands zum Thema IT Sicherheit bei OPC UA durchgeführt. Die Recherche bezog sich auf die einschlägige Literatur bzw. Veröffentlichungen und Informationen aus dem Internet. Dabei wurde untersucht, welche Aspekte aus der Spezifikation von anderen Experten als problematisch angesehen werden. Diese Ergebnisse wurden bei der Durchführung der Spezifikationsanalyse mitberücksichtigt.

Bedrohungsanalyse

Zur Unterstützung der Spezifikationsanalyse wurde eine Bedrohungsanalyse für den Anwendungsfall (siehe Abbildung 1) des OPC UA Protokolls durchgeführt, um folgende Aspekte zu adressieren:

- der Anwendungsfall enthält die wesentlichen Kommunikationsvarianten, die im Umfeld OPC UA in industriellen Steuerungsumgebungen von Bedeutung sind:
  - Kommunikation zwischen OPC UA Server und OPC UA Client, die eine volle Absicherung der Kommunikation gemäß OPC UA Spezifikation über Verschlüsselung und digitale Signatur erlauben
  - Kommunikation zwischen OPC UA Server und OPC UA Client, die keine volle Absicherung der Kommunikation erlauben, da nur ein minimaler Client zum Beispiel auf Sensoren oder Aktoren installiert und damit keine Verschlüsselung und digitale Signatur gemäß OPC UA Spezifikation möglich ist
  - Kommunikation eines Clients aus einem nicht vertrauenswürdigen Netz (zum Beispiel dem Internet) mit einem OPC UA Server, der über eine DMZ (Demilitarized Zone) geschützt ist.
- Die OPC UA Spezifikation kann nur IT Sicherheitsmaßnahmen definieren, die die Kommunikation unmittelbar schützen. Bedrohungen, die zum Beispiel auf das Betriebssystem wirken, muss mit anderen IT Sicherheitsmaßnahmen begegnet werden. Diese IT Sicherheitsmaßnahmen können durch eine Bedrohungsanalyse des Use case abgeleitet werden.

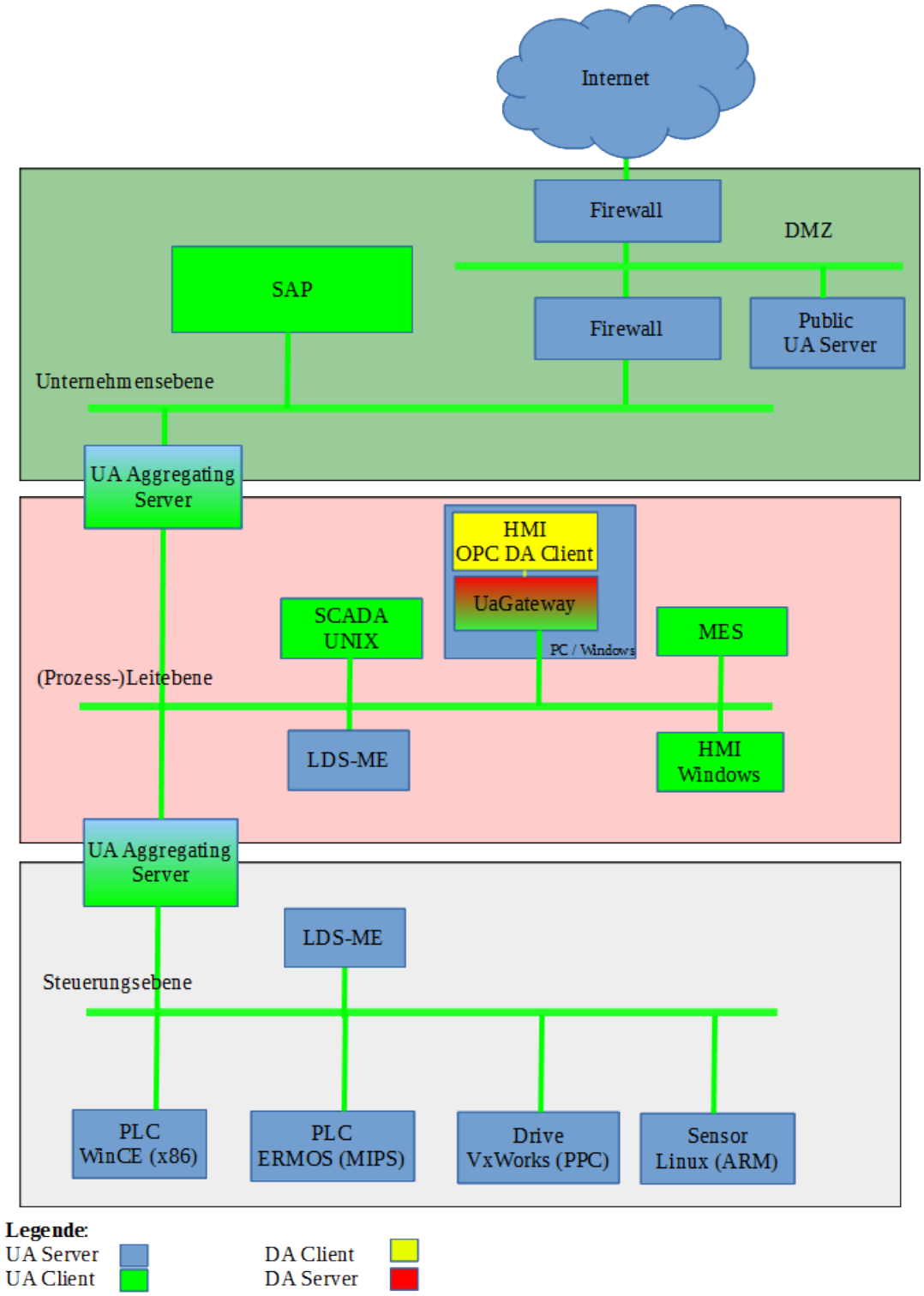

Abbildung 1: Anwendungsfall in einer beispielhaften OPC UA Umgebung

Dabei ist der Anwendungsfall hier nicht so zu verstehen, dass er ein genaues Abbild von Installationen in Steuerungsanlagen zum Beispiel in der Fertigung darstellt. Der Anwendungsfall beinhaltet aber die dargestellten Varianten von Kommunikationen, die in typischen Automatisierungs- und Steuerungsanlagen verwendet werden. Ziel ist es, aus dem Anwendungsfall Bedrohungen zu identifizieren, die im OPC UA Alltag tatsächlich eine Rolle spielen und nicht eine theoretische Randerscheinung sind, da OPC UA in dieser Konstellation nicht eingesetzt wird.

Analyse der Spezifikation

Die Identifizierung systematischer Fehler in der Spezifikation erfolgte anhand folgender Kriterien:

a Die Liste der Maßnahmen ist fehlerhaft oder unvollständig

b Die Zuordnung von Maßnahme zu Bedrohung ist falsch oder unvollständig

c Die Parametrisierung der Maßnahme ist falsch oder unzureichend

Jeder systematische Fehler wird mit seiner Kritikalität gemäß der in Kapitel 3 beschriebenen Metrik bewertet.

Der Fokus wurde dabei auf folgende Dokumente der Spezifikation wegen ihrer Relevanz für die IT Sicherheit des OPC UA Kommunikationsprotokolls gelegt:

- Part 2 – Security Model: Dieser Teil der Spezifikation bietet einen Überblick über die Sicherheitsarchitektur von OPC UA, die möglichen Angriffsszenarien und die Schutzmechanismen, die bei OPC UA definiert sind.
- Part 4 – Services: Hier werden die Inhalte der Nachrichten definiert, die als Dienste zwischen den Kommunikationspartnern OPC UA Client und Server ausgetauscht werden. Ein Großteil der Schutzmechanismen, die in Part 2 aufgeführt sind, wird durch diese Spezifikation zumindest abstrakt abgedeckt.
- Part 6 – Mappings: Dieser Teil der Spezifikation definiert die protokollspezifische Umsetzung des Sicherheitsmodells aus Part 2 und der abstrakten Serviceparameter aus Part 4. Darüber hinaus werden die protokollspezifische Kodierung der Datenstrukturen aus Part 4, die Informationsmodelle sowie die Header für die konkreten Netzwerkprotokolle definiert.
- Part 7 – Profiles: Hier werden unter anderem Sicherheitsprofile definiert, die vorgeben, welcher Satz von kryptografischen Algorithmen und Schlüssellängen zum Einsatz kommt.
- Part 12 – Discovery: Dieser Teil der Spezifikation definiert unter anderem verschiedene Möglichkeiten, wie OPC UA Clients die verfügbaren OPC UA Server im Netzwerk finden können.

2.3.2 Durchführung von Sicherheitstests auf eine Referenzimplementierung

Die Referenzimplementierung der OPC Foundation in der Version 1.02.344.5 wurde folgenden Tests unterzogen:

- Zertifikatstests
- statische Codeanalyse
- Fuzzing
- dynamische Codeanalyse

Die Beschreibung der Tests und deren Ergebnisse erfolgt in den Abschnitten 8.4 bis 8.7.

2.4 Ausschluss und nicht betrachtete Aspekte

2.4.1 Ausschlüsse

Die IT Sicherheit von „Classic OPC“ ist nicht Bestandteil des Projekts. Dabei handelt es sich um die Vorgängerversion des OPC UA Protokolls, die sich stark von OPC UA unterscheidet. Als Protokolle wird nur die Kombination bestehend aus UA TCP, UA Secure Conversation und UA Binary untersucht.

Da die aktuelle Version des Kommunikationsstacks der OPC Foundation die securityPolicy `RSA256SHA256` nicht unterstützt, konnte das Verschlüsseln mit AES256 im CBC Modus und das Signieren mit unter anderem SHA256 nicht getestet werden.

Von den existierenden OPC UA Diensten wurden ausschließlich SecureChannel, Session und Discovery getestet, da nur diese vom OPC UA Stack verarbeitet werden. Alle anderen Dienste werden zwar auch vom

OPC UA Stack deserialisiert, aber von Schichten außerhalb des OPC UA Stacks verarbeitet, typischerweise von SDKs oder der OPC UA Applikation selber. D.h. potentielle Fehler im OPC UA Stack Deserializer bei der Verwendung der übrigen Dienste wurden nicht untersucht.

Ebenso wurden selten genutzte Funktionalitäten wie z. B. die Kerberos Benutzerauthentifizierung auch nicht getestet. Diese setzt sowieso eine erfolgreiche Applikationsauthentifizierung voraus.

2.4.2 Unsicherheiten

Bei der dynamischen Codeanalyse wurden durch *Valgrind* Fehlermeldungen erzeugt, deren Auswertung mit folgenden Unsicherheiten behaftet ist:

- Nicht in allen Fällen kann man eindeutig analysieren, ob die Ursache einer Fehlermeldung sich im Kommunikationsstack der OPC Foundation oder in der Rahmenapplikation befindet.
- Die von Valgrind gemeldeten ca. 400 Fehler können potentielle Schwachstellen bezüglich der IT Sicherheit darstellen und wurden deshalb weiter analysiert. Dabei wurden die in Abschnitt 8.7 dargestellten Schwachstellen identifiziert. Die durchgeführte Analyse muss jedoch durch weitergehende Codeanalysen verfeinert werden, um sicherzustellen, dass alle Schwachstellen identifiziert wurden und die Auswirkungen dieser Schwachstellen bezüglich der IT Sicherheit klar definiert werden kann.

3 Metrik zur Bewertung der Kritikalität von Schwachstellen

Etwaige identifizierte Schwachstellen müssen bezüglich ihrer Kritikalität bewertet werden, um Gewichtungen z. B. bei den in OPC UA vorgesehenen IT Sicherheitsmaßnahmen vornehmen zu können. Um beurteilen zu können, wie kritisch diese Schwachstellen sind, wird eine Metrik benötigt. Die durch die Spezifikationsanalyse identifizierten Schwachstellen wurden durch Tests während wo immer möglich verifiziert. Daher wurde eine Metrik so gestaltet, dass die Bewertung einer Schwachstelle gemäß den Ergebnissen der Tests einfach angepasst werden konnte.

Als Grundlage für eine solche Metrik wurde CVSS v2.0 gewählt. Das Common Vulnerability Scoring System ist international anerkannt und seit Jahren erprobt und wird im Folgenden kurz beschrieben.

In CVSS werden drei Bereiche betrachtet: Base Metric Group, Temporal Metric Group, Environmental Metric Group. Diese Metrikgruppen beinhalten wiederum Parameter, die im Kontext des angegebenen Bereichs zu bewerten sind. Die Zusammenhänge sind in nachfolgenden Tabellen gezeigt:

Base Metric Group	
Access Vector	Confidentiality Impact
Access Complexity	Integrity Impact
Authentication	Availability Impact

Temporal Metric Group
Exploitability
Remediation Level
Report Conficence

Environemental Metric Group	
Collateral Damage Potential	Confidentiality Requirement
Target Distribution	Integrity Requirement
	Availability Requirement

Tabelle 3: CVSS Metrikgruppen (Quelle: [15] Seite 3)

Die Werte der Parameter gehen in Funktionen ein, mit denen ein Wert (score) zwischen 0 und 10 berechnet wird.

Für diese Studie wurde CVSS v2.0 wie folgt für dieses Projekt angepasst:

3.1 Base Metric Group

Alle Parameter aus der 'Base Metric Group' werden verwendet und demnach, wie in CVSS beschrieben, bewertet. Hier eine kurze Beschreibung der Parameter, weitere Details können in [15] nachgelesen werden:

- *Access Vector:* Der Parameter sagt etwas über den benötigten Zugang, um die Schwachstelle auszunutzen (z. B. lokal oder über das Netzwerk), aus.
- *Access Complexity:* Der Parameter stellt dar, wie schwierig die Ausführung des Angriffs ist, wenn der Angreifer sich den benötigten Zugang verschafft hat. Müssen besondere Bedingungen erfüllt sein oder ist der Angriff immer durchführbar? Ein Beispiel hierfür wäre eine Sitzung, die automatisch nach fünf Minuten abläuft, wodurch der Angreifer nur eine begrenzte Zeit hätte, um seinen Angriff auszuführen.
- *Authentication:* Hier wird gezählt, wie oft sich der Angreifer authentifizieren muss, um den Angriff durchzuführen (gar nicht bis mehrfach).
- *Confidentiality Impact, Integrity Impact, Availability Impact:* Diese Parameter stellen dar, wie stark das jeweilige Schutzziel bei einem erfolgreichen Angriff kompromittiert wird.

3.2 Temporal Metric

In dieser Gruppe wird nur der erste Parameter Exploitability für die Bewertung einer Schwachstelle hinzugezogen:

- *Exploitability:* Der Parameter besagt, ob die Schwachstelle nur theoretisch beschrieben ist oder ob es schon einen ausführbaren Angriffscode gibt. Vorteil für dieses Projekt ist, dass die bei der Spezifikationsanalyse identifizierten Schwachstellen erst mit 'Unproven' bewertet werden könnten. Nach einer Verifizierung der tatsächlichen Ausführbarkeit in der Testphase wird die Bewertung der Schwachstelle entsprechend angepasst, indem hier der Parameter auf 'Proof-of-Concept', 'Functional' oder 'High' hochgestuft wird.
- *Remediation Level:* Hier wird bewertet, ob bereits eine (temporäre) Lösung des Problems vorhanden ist. Da die im Rahmen dieses Projekts gefundene Schwachstellen unbekannt sind, kann es dafür noch keine Lösung geben. Entsprechend wurde dieser Parameter immer mit 'Not Defined' bewerten, was bedeutet, dass der Parameter keinen Einfluss auf die Bewertung der Schwachstelle hat.
- *Report Confidence:* Der Parameter sagt etwas darüber aus, wie zuverlässig die Quellen, die von einer Schwachstelle berichten, sind. Weil in diesem Projekt die Quelle immer das Projekt-Konsortium ist, wird auch dieser Parameter aus der Bewertung ausgeklammert, sprich immer mit 'Not Defined' in die Berechnungsfunktion gemäß CVSS eingesetzt.

3.3 Environmental Metrics

Die drei Parameter dieser optionalen Metrikgruppe Collateral Damage Potential, Target Distribution und IT Sicherheit Requirements helfen z. B. Betreibern einzuschätzen, wie kritisch eine Schwachstelle in ihrer eigenen Umgebung ist, indem unter anderem berücksichtigt wird, wie viele Geräte in relevanten Infrastrukturen betroffen sind.

Eine solche Bewertung der Kritikalität einer Schwachstelle ist hier nicht sinnvoll anwendbar, da keine konkrete Umgebung vorliegt. Entsprechend wird auf diese Metrikgruppe ganz verzichtet.

3.4 Berechnung des Wertes

Aus den vorangegangenen Überlegungen ergibt sich folgende Berechnungsgrundlage:

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


TemporalScore = round_to_1_decimal(BaseScore*Exploitability)
```

Die Parameterwerte werden wie folgt verwendet und festgelegt:

Name	Beschreibung	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), AvailImpact (A)	none (N)	0
	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

Tabelle 4: Für die Bewertung der Kritikalität verwendete CVSS Parameter und ihre Werte

Wichtig für die Nachvollziehbarkeit der Bewertung einer Schwachstelle ist, dass der Wert immer mit dem Vektor der Parameterwerte angezeigt wird:

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]

Eine Schwachstelle mit hohem Schadenspotential, gekennzeichnet durch Impact-Werte P oder C, hat nicht zwingend einen höheren CVSS Wert als eine Schwachstelle mit niedrigerem Schadenspotential: Ist zum Beispiel Schwachstelle mit niedrigem Schadenspotential deutlich einfacher auszuführen (Parameter AccessVector und AccessComplexity), kann ein höherer CVSS Gesamtwert als für die Schwachstelle mit hohem Schadenspotential erreicht werden.

4 Beschreibung der Testumgebung

Folgendes Diagramm zeigt die Architektur der Testumgebung für die Sicherheitstests des Kommunikationsstacks der OPC Foundation:

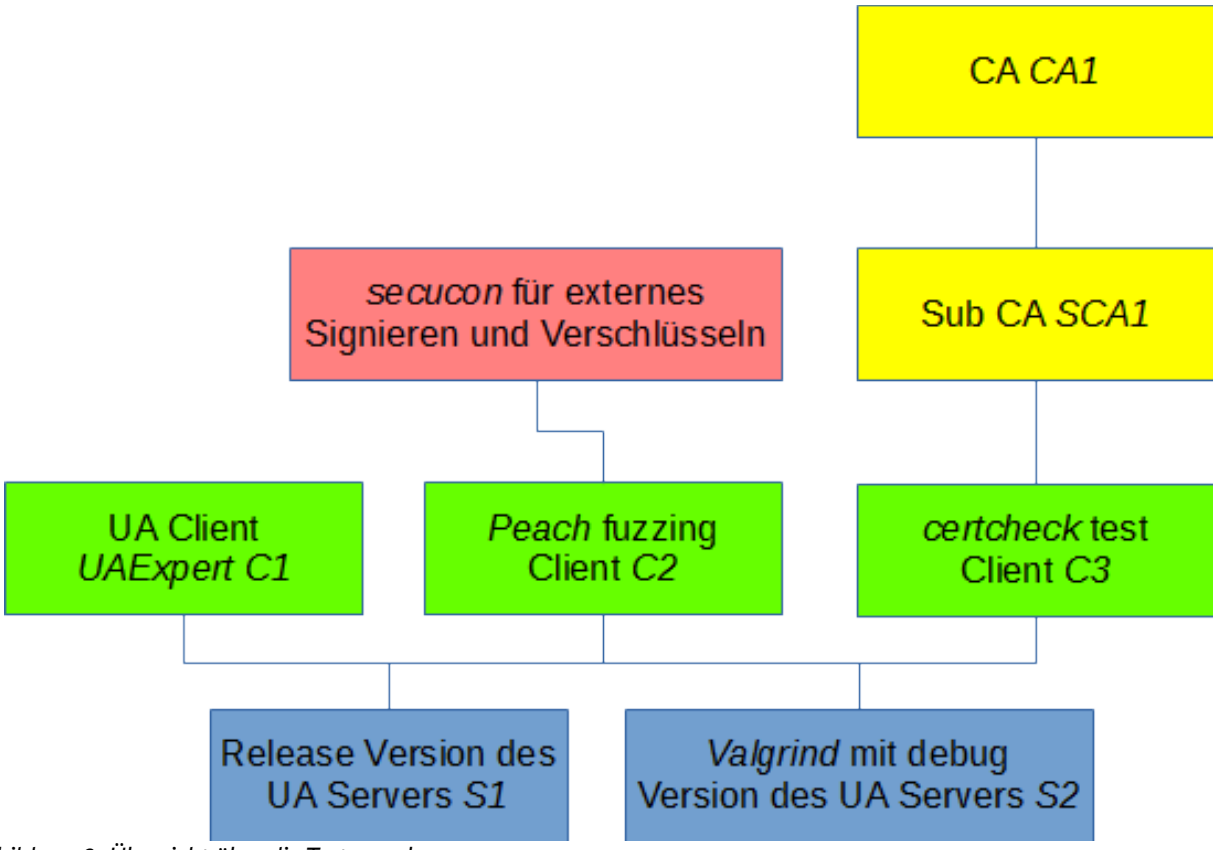

Abbildung 2: Übersicht über die Testumgebung

Folgende Elemente sind dort abgebildet:

- In grün sind die Clients zu sehen:
  - UA client *UAExpert C1:* Dieser Client ist Teil des Softwarepakets von Unified Automation und wurde anfangs lediglich zur Prüfung der Verbindung zum Server verwendet. Dieser Client wurde in AP 4 nicht weiter eingesetzt.
  - *Peach* fuzzing client *C2*: Das Fuzzingframework Peach wurde für die Fuzzingtests und die dynamische Codeanalyse in AP 4 ausgiebig verwendet. Mit Peach wurde die Clientseite simuliert und damit der OPC UA Server getestet. Weitere Details zu den Fuzzingtests und der dynamischen Codeanalyse sind jeweils in den Abschnitten 8.6 und 8.7 nachzulesen.
  - *certcheck* test client *C3*: Um die Implementierung der Zertifikatsprüfungsmechanismen zu testen, wurden Werkzeuge von Unified Automation eingesetzt, mit denen der Verbindungsaufbau eines Clients mit einem OPC UA Server simuliert wird. Dabei ermöglicht die Konfiguration der Werkzeuge das Testen von unterschiedlichen PKI (Public Key Infrastruktur) Umgebungen. Weitere Details zu diesen Tests sind in Abschnitt 8.4 beschrieben.
- Die Server werden in blau dargestellt:
  - Release Version des OPC UA Servers *S1*: Dieser Server wurde sowohl beim Fuzzing als auch bei den Tests der Zertifikatsprüfmechanismen (siehe hierzu Abschnitt 8.4) verwendet.
  - *Valgrind* mit debug Version des OPC UA Servers S2: Dieser Server wurde bei der dynamischen Codeanalyse und zur Messung der Codeabdeckung eingesetzt. Mehr zur Codeabdeckung in Abschnitt 8.8.

- Für Zertifikatstests mit dem Werkzeug *certcheck* benötigte Elemente der PKI sind in gelber Farbe abgebildet:
  - Root CA (Certificate Authority) *CA1:* Wenn die getesteten Zertifikate nicht selbstsigniert sind, wird eine CA benötigt. Die Zertifikate der Root CA sind selbstsigniert.
  - Sub CA SCA1: In manchen Zertifikatstests wird eine mehrstufige PKI simuliert. Hierfür ist eine Sub CA notwendig. Dabei wurden die Zertifikate der Sub CA von der Root CA signiert.
- Lachsfarben ist außerdem noch die externe Applikation *secucon* zu sehen. Dieses Werkzeug wurde von ascolab im Rahmen dieses Projekts entwickelt, um bei Bedarf in *Peach* erzeugte Nachrichten signieren und verschlüsseln zu können und die Antworten vom getesteten Server zu entschlüsseln und die enthaltene Signatur zu entfernen. Weitere Details zur Kommunikation zwischen *Peach* und *secucon* sind in Abschnitt 8.3 zu finden.

Alle in Abbildung 2 dargestellten Bausteine der Testumgebung sind in einer *VirtualBox* VM (virtuellen Maschine) mit 32-Bit Kali Linux Betriebssystem enthalten. Bis zu sechs Instanzen dieser VM liefen parallel auf unterschiedlichen Rechern, um die Laufzeit der Fuzzingstests zu reduzieren.

Instanz	Hostbetriebssystem	Rechner	VirtualBox Version
1	Win7 32-Bit	Arbeitslaptop	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

Tabelle 5: Verteilung der VM Instanzen

Als Grundlage für die Erzeugung der Instanzen 2 bis 6 wurde die Instanz Nr. 1 der VM verwendet.

Die Tests wurden mit *bash* Skripten ausgeführt und gesteuert. *Wireshark* wurde zur Visualisierung und Analyse des Netzwerkverkehrs benutzt, *openSSL* zur Erzeugung der Zertifikate. Der in der Testumgebung verwendete Compiler für die Server *S1* und *S2* ist *gcc*.

Weitere wesentliche Werkzeuge zur eigentlichen Durchführung der Tests sind in den Abschnitten 8.4 bis 8.8 detailliert beschrieben.

Nachfolgende Tabelle fasst die eingesetzten Werkzeuge zusammen:

Werkzeug	Zweck	Plattform	Version
Wireshark	Visualisierung und Analyse des Netzwerkverkehrs	Linux 32-Bit	1.99.8
openSSL	Erzeugung von Zertifikaten	Linux 32-Bit	1.0.1e
cppcheck	Statische Codeanalyse	Linux 32-Bit	1.54
Valgrind	Dynamische Codeanalyse	Linux 32-Bit	3.8.1
gcc	Compiler	Linux 32-Bit	4.7.2
lcov + genhtml	Messung und graphische Darstellung der Codeabdeckung	Linux 32-Bit	1.9
Peach	Fuzzingframework	Linux 32-Bit	3.1.124.0
bash	Steuerung der Tests	Linux 32-Bit	4.2.37

Tabelle 6: Liste der wesentlichen Werkzeuge, die für die Tests eingesetzt wurden

5 Bestandsaufnahme des Kenntnisstands bzgl. der IT Sicherheit von OPC UA

Im Folgenden (für alle weiteren Kapitel) werden unter Feststellungen gewisse Verbesserungsmöglichkeiten verstanden. Feststellungen bedeuten aber noch keinen systematischen Fehler der Spezifikation bzgl. IT Sicherheit. Systematische Fehler bedeuten, dass die Spezifikation eine Sicherheitslücke in der Definition des OPC UA Protokolls beinhaltet.

Die Internetrecherche hat gezeigt, dass in der Vergangenheit schon vereinzelt Schwachstellen bei Implementierungen von OPC UA Applikationen gemeldet und behoben wurden:

- Ausschnitt aus [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."
- Ausschnitt aus [17]: „OPCUA wrapper SP3 Enhancement : XML parser cyber-security vulnerability fix"

Die interessantesten Ergebnisse befinden sich aber in [18], [19] und [20]. Im Nachfolgenden bedeuten rot markierte Texte Aspekte, die noch offen sind und bisher nicht von der OPC Foundation berücksichtigt wurden, grün markierte Texte Aspekte, die als behoben eingeschätzt wurden.

- Unter [18] gibt Digital Bond Auskunft über die Ergebnisse einer bereits im August 2008 abgeschlossenen Analyse der IT Sicherheit von OPC UA basierend auf die damalige Spezifikation. Außerdem wurde im Rahmen des gleichen Auftrags die IT Sicherheit eines von der OPC Foundation zur Verfügung gestellten Softwarepakets untersucht. Es wird vermutet, dass es sich dabei um eine Beispielimplementierung von der OPC Foundation in C# (für die Microsoft .NET Umgebung) handelte. Die Untersuchung zeigt somit nicht zwangsläufig Fehler oder Aspekte, die auch die ANSI C Referenzimplementierung betreffen.
  Die Spezifikationsanalyse hat damals folgendes ergeben:
  - Digital Bond schrieb, dass selbstsignierten Zertifikaten nicht automatisch, also ohne zusätzliche Prüfung vertraut werden darf.
    Dies ist aus unserer Sicht mit der Einführung der Trust List erfüllt, wenn diese geeignet vor Manipulation geschützt wird.
  - Secure Channels sollten nicht so heißen, falls sie mit der SecurityPolicy-Einstellung 'None' keine Sicherheit anbieten.
    Dieser wichtige Hinweis wurde in einer Arbeitsgruppe der OPC Foundation mehrfach besprochen. Diese kam aber zu dem Schluss, dass eine Änderung des Namens nicht mehr möglich ist, weil die Bezeichnung u.a. in APIs enthalten ist. Um zu verhindern, dass ein falsches Gefühl der Sicherheit vermittelt wird, wurden Hinweise in der Spezifikation hinzugefügt, die explizit darauf aufmerksam machen, dass bei securityMode 'None' keine Sicherheit enthalten ist.
  - Falls RegisterServer mit securityMode 'None' erlaubt ist, kann dies von einem Angreifer missbraucht werden, um einen vertrauenswürdigen Server vorzutäuschen.
    Dies wird in Part 7 Tabelle 3 explizit verboten. In der Version 1.03 ist auch in Part 4 ein weiterer Hinweis geplant.
  - Es waren in der Spezifikation keine Vorgaben für Zufallszahlengeneratoren enthalten. Die Spezifikation verlässt sich darauf, dass sie mit ausreichender Entropie erzeugt werden.
  - Damals gab es auch Widersprüche in der Spezifikation, ob die Pakete bei *OpenSecureChannel* und *CloseSecureChannel* signiert und verschlüsselt sein müssen.
    Es konnten in der Version 1.02 keine Widersprüche gefunden werden. Allerdings sollte explizit erwähnt werden, dass OpenSecureChannel requests und responses auch bei securityMode Sign verschlüsselt werden.
    Hier sollte auf die Gefahren von schlechter Entropie hingewiesen und in Form von Funktionalitätsklassen Mindestanforderungen an die Zufallszahlengeneratoren gestellt werden. Siehe [21] Abschnitt 9 und [22] Abschnitt 4 für weitere Informationen.

Die Prüfung der Software durch [18] hatte folgende Probleme gezeigt:
  - heap- und stack-Überläufe
  - Abstürze bei fuzzing Tests

  Digital Bond war zuversichtlich, dass die OPC Foundation die meisten Schwachstellen in der nächsten Version der Spezifikation und des Quellcodes beheben würde.
- NIST hat im Rahmen seines Smart Grid Interoperability Panels (SGIP) im dritten Quartal 2012 die Ergebnisse der Cyber Security Working Group (CSWG) bezüglich der Evaluierung der IT Sicherheit von OPC UA [19] veröffentlicht. Diese Evaluierung basierte auf den damaligen Stand der Spezifikation. Folgende wesentliche Punkte wurden in den Ergebnisberichten angemerkt:
  - Ein Ausfall der Auditfunktionalität kann zu Lücken bei einer späteren forensischen Analyse führen. Deshalb sollte dies bei der Umsetzung der Auditfunktionalität berücksichtigt werden.
  - Es sollte explizit erwähnt werden, dass Cipher Suites, die nicht die Anforderungen von FIPS 140-2, NIST SP 800-131A oder vergleichbaren Referenzdokumenten erfüllen, nur in Ausnahmefällen eingesetzt werden sollten.
    Was die Schlüssellängen und Protokollversionen bei TLS angeht, wird auf NIST Empfehlungen hingewiesen. Auf veraltete oder unsichere Cipher Suites wird in der Spezifikation überhaupt nicht eingegangen.
  - In Part 6 Anhang D Tabelle D2 sollte SHA-1 ersetzt werden.
    Diesbezüglich hat noch keine Aktualisierung stattgefunden.
  - In Part 7 sollte explizit darauf hingewiesen werden, dass SHA-1 als veraltet gilt und stattdessen SHA-256 verwendet werden sollte.
    Diesbezüglich hat noch keine Aktualisierung stattgefunden.
  - In Part 7 fehlen jegliche Informationen, welche TLS Cipher Suites veraltet sind und welche empfohlen werden.
    Diesbezüglich hat noch keine Aktualisierung stattgefunden.
  - In Part 7 sollte explizit darauf hingewiesen werden, dass Passwörter nie im Klartext übertragen werden dürfen.
    In [5] und [7] wird konsequent darauf hingewiesen, dass die Authentifizierung mit geheimnisbasierte userIdentityTokens nur verschlüsselt stattfinden darf.
- In [20] untersuchen die Autoren des Papers von Ende 2014, wie die IT Sicherheitsmechanismen in OPC UA umgesetzt sind.
  - Daraus folgern sie, dass einige der in den Profilen für das Verschlüsseln und Signieren gewählten Algorithmen nicht mehr zeitgemäß sind.
    Gemäß [21] und [23] sind folgende Empfehlungen des BSI zu berücksichtigen:

    - SHA-1 sollte gar nicht mehr verwendet werden. Einzige Ausnahme ist HMAC-SHA1.
    - Als Formatierungsverfahren wird für RSA EME-OAEP empfohlen.
    - TLS 1.0 darf nicht mehr eingesetzt werden.
    - Dass RSA Schlüssel mindestens 2.000 bit lang sein sollten, ist in [8] zufriedenstellend adressiert. Allerdings ist zu bezweifeln, dass das Personal, das für die Konfiguration von OPC UA Clients und Server verantwortlich ist, solche Hinweise aus der Spezifikation zur Kenntnis nimmt.

  - Außerdem ergibt ihre Untersuchung, dass die Umsetzung unnötig viel Rechenleistung in Anspruch nimmt, was sich bei Feldgeräten mit kleinen Prozessoren negativ auswirken kann.
  - Der gleiche private Schlüssel wird zum Signieren und Entschlüsseln verwendet.
  - Die auf elliptische Kurven basierenden Algorithmen (ECDSA und ECDH) können auf Grund der doppelten Verwendung der privaten Schlüssel bei OPC UA nicht eingesetzt werden.

Zusammenfassung der wesentlichen Erkenntnisse:

ID	Feststellung	Empfehlung
1	Der Name „Secure Channel“ suggeriert IT Sicherheit, die bei securityMode None aber nicht enthalten ist. Leider kann der Name nicht mehr geändert werden.	keine
2	Es werden für Zufallszahlengeneratoren keine Hinweise auf Mindestanforderungen in der OPC UA Spezifikation angegeben.	Es sollten für Zufallszahlengeneratoren Hinweise auf Mindestanforderungen von anerkannten Institutionen (zum Beispiel BSI) in der OPC UA Spezifikation angegeben werden.
3	Es werden keine Hinweise zu veralteten oder als unsicher geltenden Algorithmen, insbesondere SHA-1, und Cipher Suites, gegeben.	Es sollten Hinweise zu veralteten oder als unsicher geltenden Algorithmen, insbesondere SHA-1, und Cipher Suites, gegeben werden, z. B. Links auf anerkannte Institutionen (z. B. BSI), die zu kryptographischen Algorithmen regelmäßig Empfehlungen abgeben.
4	Der private Schlüssel wird bei OPC UA sowohl für die Signatur als auch für die Entschlüsselung verwendet.	Für Signatur und Verschlüsselung sollten verschiedene Schlüsselpaare in der OPC UA Spezifikation gefordert werden.
5	Derzeit besteht keine Möglichkeit auch auf elliptische Kurven basierende Algorithmen zurückzugreifen.	Diese Möglichkeit sollte OPC UA in Zukunft vorsehen, um auch auf Geräten mit geringer Prozessorleistung ausreichende Sicherheit zu ermöglichen.

Tabelle 7: Wesentliche Erkenntnisse aus der Bestandsaufnahme

6 Redaktionelle Anmerkungen und Korrekturvorschläge

Im Rahmen der einer redaktionellen Analyse sind verschiedene Punkte aufgefallen, die eventuell zu Verständnis- und/oder Implementierungsproblemen führen können (z. B. einfache Tippfehler, missverständliche Formulierungen, Inkonsistenzen, usw.) oder sogar Relevanz für die IT Sicherheit haben. Die IT Sicherheit des Protokolls wird in Kapitel 7 dieser Studie weiter detailliert analysiert.

Tabelle 8 fasst die gefundenen Punkte zusammen: In der ersten Spalte wird eine eindeutige ID festgelegt. Die folgenden vier Spalten in Tabelle 8 dienen der Findung der Textstelle, Tabelle oder Abbildung in den verschiedenen Dokumenten der Spezifikation. In der Spalte 'Feststellung' wird erläutert, was unklar oder falsch ist. Bei Bedarf werden diese Anmerkungen mit Empfehlungen, wie die angemahnten Stellen eventuell verbessert werden könnten, ergänzt. Die nach unserer Einschätzung (ohne explizites Bewertungsschema) wichtigeren Punkte wurden gelb markiert, sicherheitsrelevante Punkte sind rot markiert. In Tabelle 9 werden die wichtigsten Erkenntnisse, insbesondere die sicherheitsrelevanten Punkte, zusammengefasst.

ID	Dokument	Seite	Kapitel	Abschnitt/ Tabelle/ Diagramm	Feststellung	Empfehlung
1	Part 2	8	4.2		Die Definitionen von Begriffen aus der IT Sicherheit entsprechen nicht den Definitionen aus international anerkannten Standards wie z. B. ISO 27000 [24].	Dies wird empfohlen, da Begriffe einheitlich verwendet werden sollten, um Missverständnisse zu vermeiden.
2	Part 2	9	4.3.2	Abschnitt 1	Warum wird nur Server flooding erwähnt? Es kommt unter Umständen auch Client flooding als Bedrohung in Frage.	Client flooding bitte ergänzen
3	Part 2	16	4.12.1	Abschnitt 1	Der letzte Satz ist unvollständig.	
4	Part 2	22	5.2.2.1		Die Identifizierung findet mit dem X.509 Zertifikat statt, die Authentifizierung mit dem privaten Schlüssel.	
5	Part 2	22	5.2.2.2	Abschnitt 1	Der userIdentityToken kommt nicht in der openSecureChannel, sondern in der ActivateSession Anfrage vor.	
6	Part 2	22	5.2.4-5.2.5	Abschnitt 2	Die PKI ist nicht für die Verwaltung der symmetrischen Schlüssel zuständig. Diese werden anhand der Client und Server nonces abgeleitet.	
7	Part 2	25	6.11	Abschnitt 4	Der erste Satz stimmt in dieser allgemeinen Form nicht.	Die Aussage sollte auf den Einsatz von Zertifikaten bei OPC UA eingeschränkt werden.
8	Part 2	28	6.11	Abschnitt 2	„unique key used to create and verify digital signatures“ ist falsch: Der private Schlüssel wird zur Erzeugung der Signatur verwendet, der öffentliche zur Verifizierung. Ein Schlüssel kann aber nicht für beides verwendet werden.	
9	Part 4	12	5.4.1	Fig. 9	Die Abbildung macht nicht deutlich, dass für FindServers und GetEndpoints erst ein SecureChannel mit securityMode 'None' aufgebaut werden muss.	Diese Information sollte entweder in der Abbildung dargestellt werden oder im Text erläutert werden.
10	Part 4	16	5.4.3.1	Fig. 10	„CreateSecureChannel“ kommt an mehreren Stellen in Abbildungen, Tabellen oder im Text vor.	Durch “OpenSecureChannel“ ersetzen
11	Part 4	19	5.4.4.2	Tabelle 5	Es steht, dass der authenticationToken weggelassen („omitted“) werden soll, auch auf den Seiten 14, 16 und 23.	Dies soll durch „null“ oder „empty“ ersetzt werden.
12	Part 4	22	5.5.2.1	Abschnitt 2	Die Signatur findet mit dem privaten Schlüssel statt, nicht mit	

ID	Dokument	Seite	Kapitel	Abschnitt/ Tabelle/ Diagramm	Feststellung	Empfehlung
					dem Zertifikat. Die Verschlüsselung findet mit dem öffentlichen Schlüssel statt, nicht mit dem Zertifikat.	
13	Part 4	23	5.5.2.2	Tabelle 7	Nicht Zertifikat, sondern privater Schlüssel	
14	Part 4	23	5.5.2.2	Tabelle 7	Bei der secureChannelId werden in Part 4 und 6 zwei Datentypen erwähnt: ByteString und Uint32. Dies gilt auch für Tabelle 9.	Laut ascolab ist Part 6 ausschlaggebend. Tabelle 7 sollte überarbeitet oder gelöscht werden, damit keine Widersprüche mehr bestehen.
15	Part 4	23	5.5.2.2	Tabelle 7	“channelId“ sollte durch „secureChannelId“ ersetzt werden.	
16	Part 4	23	5.5.2.2	Tabelle 7	Für den Parameter requestedLifetime fehlen Unter- und/oder Obergrenzen. (In der Referenzimplementierung des Kommunikationsstacks wird eine Stunde verwendet.)	Bitte Werte(bereiche) empfehlen
17	Part 4	24	5.5.3.2	Tabelle 9	Laut Part 6 S. 47 wird keine Antwort auf eine closeSecureChannel Anfrage geschickt.	Die Stelle in Part 4 sollte umformuliert werden.
18	Part 4	27	5.6.2.2	Tabelle 11	Für maxResponsesMessageSize fehlen Angaben zu Defaultwerten.	Empfehlungen für eine Untergrenze oder einen Defaultwert bei maxResponsesMessageSize
19	Part 4	28	5.6.2.2	Tabelle 11	Bei serverSoftwareCertificates steht, dass ein leeres array verwendet werden muss.	Es sollte genau spezifiziert sein, was „omitted“, „empty“ und „null“ bedeutet.
20	Part 4	28	5.6.2.2	Tabelle 11	Falls securityMode 'None' verwendet wird, liegt kein Client Zertifikat für die Berechnung von der serverSignature vor.	Das Problem wurde in der Version 1.03 bereits behoben.
21	Part 4	99	6.1.5		Der letzte Satz ist zu allgemein bzw. falsch, da bei geheimnisbasierten userIdentityTokens keine Signatur als Beweis erzeugt wird.	
22	Part 4	147	7.29	Fig. 30	In der Abbildung steht “GetSecureChannelInfo”. Diese Funktion ist aber nirgends definiert bzw. erläutert.	ascolab empfiehlt folgende oder eine ähnliche Formulierung „information requested by the server application from the communication stack“.
23	Part 4	147	7.29	Fig. 30	Die SessionId ist kein Parameter in ActivateSession Anfragen.	
24	Part 4	148	7.31	Tabelle 161	Sollte bei keyUsage zweimal „may“ durch „shall“ ersetzt werden?	
25	Part 4	154	7.35.1		Für geheimnisbasierte userIdentityTokens wird verlangt, dass sie	Es sollte zwingend erforderlich sein („shall“), dass

ID	Dokument	Seite	Kapitel	Abschnitt/ Tabelle/ Diagramm	Feststellung	Empfehlung
					verschlüsselt werden. Bei signaturbasierten userIdentityTokens steht nichts.	immer entweder eine userTokenPolicy oder die securityPolicyUri nicht 'None' ist, damit zur Erstellung der Signatur ein Algorithmus zur Auswahl steht.
26	Part 4	155	7.35.3	Tabelle 171	„Zertifikat“ sollte durch „öffentlicher Schlüssel“ ersetzt werden.	
27	Part 4	156	7.35.5	Tabelle 173	„Zertifikat“ sollte durch „öffentlicher Schlüssel“ ersetzt werden.	
28	Part 6	6	5.1.2	Tabelle 1	Die byte Definition ist falsch.	Durch „0 bis 255“ ersetzen
29	Part 6	7	5.1.6	Abschnitt 3	Laut Spezifikation können variant arrays ineinander verschachtelt werden. Das gleiche gilt für diagnosticInfo.	Es sollte eine sinnvolle maximal zulässige Rekursionstiefe festgelegt werden, z. B. 10.
30	Part 6	36	6.7.1	Abschnitt 2	Sollte der Wert 8196 oder 8192 lauten?	Laut ascolab sollte es 8192 heißen.
31	Part 6	38	6.7.2	Tabelle 30	MaxCertificateSize ist nirgends definiert.	Es sollte erläutert werden, wie oder wo dieser Parameter konfiguriert wird.
32	Part 6	38	6.7.2	Tabelle 30	Datentyp von SecurityPolicyUriLength und ReceiverCertificateThumbprintLength auf byte oder UInt32 ändern. Datentypen von SenderCertificateLength auf UInt32 ändern.	Es wird generell bei Parametern empfohlen, alle integer Datentypen als unsigned zu definieren, falls der jeweilige Parameter keine negativen Werte annehmen kann. In diesem speziellen Fall ist es allerdings sinnvoller, die drei Parameter SecurityPolicyUri, SenderCertificate und ReceiverCertificateThumbprint als ByteStrings zu definieren, was der Implementierung entspricht, weil die Länge -1 wie bei NULL ByteStrings zulässig ist.
33	Part 6	38	6.7.2	Tabelle 30	Die Angaben widersprechen sich: ReceiverCertificateThumbprintLength ist immer 20 und ReceiverCertificateThumbprint ist null, falls die Nachricht nicht verschlüsselt wird.	
34	Part 6	39	6.7.2		Was passiert, wenn eine Lücke bei der sequenceNumber entdeckt	Es sollte explizit beschrieben werden, wie eine

ID	Dokument	Seite	Kapitel	Abschnitt/ Tabelle/ Diagramm	Feststellung	Empfehlung
					wird?	OPC UA Applikation auf einen solchen Fall reagieren soll (Fehlermeldung, Verbindungsabbruch, usw.).
35	Part 6	41	6.7.4	Tabelle 35	Der erste Satz auf S. 41 und die Tatsache, dass in Tabelle 35 andere Parameter mit anderen Datentypen stehen, als in Part 4 spezifiziert, kann für Verwirrung sorgen.	Es sollte ausführlicher darauf hingewiesen werden, dass Part 4 eine allgemeine Beschreibung der Dienste darstellt und in Part 6 auf die protokollabhängigen Details eingegangen wird. Dementsprechend hat Part 6 Vorrang, falls widersprüchliche Aussagen zustande kommen. Es sollte aber bevorzugt Part 4 nach Möglichkeit so überarbeitet werden, dass keine Widersprüche entstehen.
36	Part 6	41	6.7.4		OpenSecureChannel Anfragen und Antworten werden auch bei securityMode 'Sign' verschlüsselt.	Dies sollte explizit erwähnt werden.
37	Part 6	42	6.7.5		Es fehlen Angaben über die Qualität oder Mindestanforderungen von Zufallszahlengeneratoren.	Empfehlungen zur Qualität oder Mindestanforderungen von Zufallszahlengeneratoren
38	Part 6	43	6.7.6	Abschnitt 3	Wegen der unterschiedlichen Datentypdefinitionen ByteString und UInt32 unterscheiden sich auch die Defaultwerte bei einem neuem SecureChannel: 0 und null.	Es sollte verstärkt auf die Konsistenz von Datentypen und Defaultwerten zwischen Part 4 und 6 geachtet werden. Außerdem sollte genau spezifiziert sein, was „null“ tatsächlich bedeutet: Es kann beispielsweise für 0 bei integer, eine initialisierte leere Datenstruktur (Länge 0) oder eine nicht initialisierte Datenstruktur (null pointer) stehen.
39	Part 6	63	D.2	Tabelle D1	Im ersten Satz „UntrustedIssuerStore“ durch „IssuerStore“ und „UntrustedIssuerList“ durch „IssuerList“ ersetzen.	
40	Part 6	64	D.3	Tabelle D2	Der Datentyp für ValidationOptions sollte auf byte oder UInt32 geändert werden, da keine negativen Werte zulässig sind.	

ID	Dokument	Seite	Kapitel	Abschnitt/ Tabelle/ Diagramm	Feststellung	Empfehlung
41	Part 6	67	D.6	Tabelle D6	Die Tabelle gibt Optionen an, die eine unsichere Konfiguration der Kommunikation darstellen.	Es sollte explizit darauf hingewiesen werden, dass manche dieser Optionen unsicher sind.
42	Part 7	15	5.3	Tabelle 11	Unter Security Basic 128Rsa15: http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/dk/p_sha1 existiert nicht mehr.	
43	Part 7	18-19	5.3	Tabelle 11	Die “security levels” 1 bis 3 sind nirgends definiert. Was ist ihr Nutzen?	Erläuterungen zu den security levels hinzufügen
44	Part 7	37	5.5	Tabelle 21	Es könnte eine Kategorie „documentation – security best practices/recommended settings“ hinzugefügt werden.	
45	Part 7	78	6.5.124-126		Warum wird für diese Profile eine PKI benötigt? Können selbstsignierte Zertifikate mit trust lists nicht verwendet werden?	
46	Part 12	4	4.2.2	Abschnitt 2	RegisterServer2 ist in der Version 1.02.47 nicht definiert.	
47	Part 12	5	4.3.1		FindServersOnNetwork ist in der Version 1.02.47 nicht definiert.	
48	Part 12	13	6.1		QueryServers ist in der Version 1.02.47 nicht definiert.	
49	Part 12	19	7.2	Fig. 13	Die Abbildung fehlt.	In der Version 1.03.53 ist sie vorhanden.
50	Part 12	27	7.6.4		Es fehlen Angaben, wie privateKeyPassword bestimmt wird.	Hinzufügen von Mindestanforderungen oder Empfehlungen über die Qualität des Passworts
51	Part 12	32	7.7.3	Abschnitt 2	Es fehlen Angaben zum Defaultwert von MaxTrustListSize.	Empfehlungen für den Defaultwert von MaxTrustListSize
52	Part 12	40	B.3	Tabelle 23	Der letzte Satz im Feld „scheme“ ist unvollständig.	Dies wurde in der Version 1.03 bereits behoben.
53	Part 12	40	B.3		Falls TTL für Time To Live steht, dann handelt es sich um einen byte. Entsprechend ist die Obergrenze 255, üblicherweise wird 64 als Defaultwert empfohlen.	Steht TTL für etwas anderes, sollte dies definiert werden. Ansonsten sollte der Text korrigiert werden.
54	Part 12	47	F.2		Wie werden rogue Server im provisioning state erkannt?	
55	Part 12	47	F.2	Abschnitt 2	Was ist das „OPC UA interface“?	ascolab schlägt vor, „serverConfigurationType“ stattdessen zu verwenden.

ID	Dokument	Seite	Kapitel	Abschnitt/ Tabelle/ Diagramm	Feststellung	Empfehlung
56	Part 12	48	F.2	Abschnitt 3	„authorized“ sollte durch „unauthorized“ ersetzt werden. Dies ist in Version 1.03 bereits geschehen.	
57	allgemein				Derzeit wird folgende Einstellungskombination erlaubt: securityMode 'Sign' oder 'SignAndEncrypt' und securityPolicyUri 'None'. Dies führt zu einer unsicheren Kommunikation.	Es wird empfohlen, bei securityMode 'Sign' und 'SignAndEncrypt' als securityPolicyUri 'None' zu verbieten und diese Validierung der Konfiguration auch als Konformitätstest hinzuzufügen.
58	allgemein				Es wird empfohlen, an allen Stellen „X509“ durch „X.509 v3“ zu ersetzen.	
59	allgemein				Adressierung der in Abschnitt 7.4 gefunden Schwachstellen	
60	allgemein				Zur Konfiguration einer OPC UA Applikation gehört eine Reihe von Parametern, die Unter- oder Obergrenzen z. B. für Puffergrößen oder Anzahl der Nachrichten festlegen. Es fehlt allerdings meist die Angabe eines Defaultwerts oder eine Empfehlung, in welchem Bereich sinnvolle Werte liegen. Es würde Entwicklern und Administratoren die Arbeit deutlich erleichtern, wenn in der Spezifikation eine Liste von solchen Defaultwerten vorhanden wäre.	Wir empfehlen eine Liste von solchen Defaultwerten. Dabei könnten zwei Geräteklassen unterschieden werden: eingebettete Geräte mit begrenzter Leistung einerseits und PCs oder Workstations andererseits.

Tabelle 8: Fragen und Anmerkungen zur Spezifikation

Zusammenfassung der wesentlichen Erkenntnisse:

ID	Feststellung	Empfehlung
1	Inkonsistenzen zwischen Part 4 und 6: Sowohl Part 4, als auch Part 6, beschreibt insbesondere die Dienste, die für einen sicheren Verbindungsaufbau auf der Grundlage eines SecureChannels und einer Session, entscheidend sind. Allerdings ist Part 4 allgemein gehalten, wohingegen Part 6 auf die Besonderheiten der zur Auswahl stehenden Protokolle eingeht. Dadurch unterscheiden sich teilweise die Parameter, die in den verschiedenen Teilen einer Nachricht enthalten sind. Auch die Zuordnung mancher Datentypen, die zur Definition der Parameter dient, ist unterschiedlich. Dies ist verwirrend und fehlerträchtig.	Die angesprochenen Inkonsistenzen zwischen Part 4 und Part 6 sollten beseitigt werden.
2	Unklarheiten bei nicht gebrauchten Parametern: Durch die Möglichkeit, über den securityMode das Signieren und Verschlüsseln ein- oder auszuschalten, werden in bestimmten Fällen nicht benötigte Parameter in einer Nachricht mitübertragen. Es ist aber nicht eindeutig, welche Werte in solchen Fällen bei den betroffenen Parametern einzusetzen sind. Dies kann zu Implementierungsfehlern und Inkompatibilitäten zwischen OPC UA Applikationen von verschiedenen Herstellern führen.	Auch für nicht benötigte Parameter einer Nachricht sollten die zu setzenden Parameterwerte definiert werden.
3	Fehlende Liste sinnvoller Defaultwerte: Zur Konfiguration einer OPC UA Applikation gehört eine Reihe von Parametern, die Unter- oder Obergrenzen z. B. für Puffergrößen oder Anzahl der Nachrichten festlegen. Es fehlt allerdings meist die Angabe eines Defaultwerts oder eine Empfehlung, in welchem Bereich sinnvolle Werte liegen. Es würde Entwicklern und Administratoren die Arbeit deutlich erleichtern, wenn in der Spezifikation eine Liste von solchen Defaultwerten vorhanden wäre. Man könnte dabei zwei Geräteklassen unterscheiden: eingebettete Geräte mit begrenzter Leistung einerseits und PCs oder Workstations andererseits.	Die Defaultwerte von Parametern sollten immer festgelegt werden. Eine Liste von solchen Defaultwerten sollte erstellt werden und zwei Geräteklassen unterscheiden: eingebettete Geräte mit begrenzter Leistung einerseits und PCs oder Workstations andererseits.
4	Besserer Schutz vor Angriffen auf geheimnisbasierten userIdentityToken: Zur Zeit schreibt die Spezifikation nicht vor, wie bei wiederholter fehlgeschlagener Benutzerauthentifizierung vorzugehen ist.	Ein Vorschlag zur weiteren Erhöhung der Sicherheit wäre, dass Server die Einstellung erlauben, bei jeder Fehlanmeldung den Zeitraum bis zur Bearbeitung der nächsten Benutzerauthentifizierung zu verdoppeln und z. B. bei einer Minute zu kappen. Bei erfolgreicher Anmeldung wird diese Grenze wieder auf eine Sekunde zurückgesetzt.

ID	Feststellung	Empfehlung
5	Derzeit wird folgende Einstellungskombination erlaubt: securityMode 'Sign' oder 'SignAndEncrypt' und securityPolicyUri 'None'. Dies führt zu einer unsicheren Kommunikation.	Es wird empfohlen, bei securityMode 'Sign' und 'SignAndEncrypt' als securityPolicyUri 'None' zu verbieten und diese Validierung der Konfiguration auch als Konformitätstest hinzuzufügen.
6	Part 4, S. 154, Kap. 7.35.1 Für geheimnisbasierte userIdentityTokens wird verlangt, dass sie verschlüsselt werden. Bei signaturbasierten userIdentityTokens steht nichts.	Es sollte zwingend erforderlich sein („shall“), dass immer entweder eine userTokenPolicy oder die securityPolicyUri nicht 'None' ist, damit zur Erstellung der Signatur ein Algorithmus zur Auswahl steht.
7	Part 6, S. 7, Kap. 5.1.6, Abschnitt 3 Laut Spezifikation können variant arrays ineinander verschachtelt werden. Das gleiche gilt für diagnosticInfo.	Es sollte eine sinnvolle maximal zulässige Rekursionstiefe festgelegt werden, z. B. 10.
8	Part 6, S. 39, Kap. 6.7.2 Was passiert, wenn eine Lücke bei der sequenceNumber entdeckt wird?	Es sollte explizit beschrieben werden, wie eine OPC UA Applikation auf einen solchen Fall reagieren soll (Fehlermeldung, Verbindungsabbruch, usw.).
9	Part 6, S. 41, Kap. 6.7.4 OpenSecureChannel Anfragen und Antworten werden auch bei securityMode 'Sign' verschlüsselt.	Dies sollte explizit erwähnt werden.
10	Part 6, S. 67, Kap. D6, Tabelle D6 Die Tabelle D6 gibt Optionen an, die eine unsichere Konfiguration der Kommunikation darstellen.	Es sollte explizit darauf hingewiesen werden, dass manche dieser Optionen unsicher sind.

Tabelle 9: Wesentliche redaktionelle Anmerkungen und sicherheitskritische Feststellungen

7 Ergebnisse aus der Spezifikationsanalyse

Die hier dargestellte Analyse umfasst folgende Punkte:

1 Überprüfung der Schutzziele und Bedrohungstypen (siehe Abschnitt 7.1)
Ziel ist hier die Korrektheit bzw. Lücken bezüglich Schutzziele und Bedrohungstypen, wie sie in Part 2 der Spezifikation beschrieben sind, festzustellen. Zudem wird geprüft, ob die Zuordnung zwischen Bedrohungen und Schutzzielen korrekt sind. Die Analyse führt zu einem Verbesserungsvorschlag für Schutzziele und Bedrohungstypen, der in der Bedrohungsanalyse (siehe Punkt 2) verwendet wird.

2 Bedrohungsanalyse für das OPC UA Protokoll (siehe Abschnitt 7.2)
Die Bedrohungsanalyse umfasst zwei Unterpunkte:

   a Analyse gemäß Bedrohungen aus Part 2 (siehe Abschnitt 7.2.1)
   Ziel ist hier die Kritikalität gemäß CVSS Bewertungsschema zu identifizieren. Zudem soll die Analyse zeigen, wie die Bedrohungen aus Part 2 für OPC UA konkretisiert werden können. Die Ergebnisse der Analyse sind in Tabelle 16 gezeigt. Weiter kann diese Bedrohungsanalyse zur Ableitung von Testfällen für die Referenzimplementierung des OPC UA Kommunikationsstacks verwendet werden.

   b Analyse von Bedrohungen auf Elemente der OPC UA Infrastruktur (siehe Abschnitt 7.2.2)

3 Analyse der Schutzmechanismen des OPC UA Protokoll auf Parameterebene (siehe Abschnitt 7.3)
Hier wird analysiert, ob die Sicherheitsmaßnahmen des OPC UA Protokolls den Bedrohungen (siehe Punkt 1 und Punkt 2) entgegenwirken. Die detaillierte Analyse bezieht auch die Parameterebene des Protokolls mit ein. Mit dieser Analyse wird die Kette Schutzmechanismus – Bedrohung – Schutzziel geschlossen und erlaubt Aussagen darüber, ob Schutzziele erreicht werden bzw. Bedrohungen ausreichend begegnet wird.

Die Ergebnisse der Analyse sind in Abschnitt 7.5 zusammengefasst.

7.1 Überprüfung der Schutzziele und Bedrohungstypen

7.1.1 Analyse mittels STRIDE

In Part 2 wurden bereits eine Reihe an Bedrohungstypen berücksichtigt. Dank unabhängiger threat modelling Methodiken wie STRIDE kann man diese Liste eventuell noch ergänzen. In der folgenden Tabelle wird aufgeschlüsselt, welche sechs Bedrohungstypen sich hinter STRIDE verbergen. Außerdem wird dargestellt, ob durch die in Part 2 beschriebenen Bedrohungstypen alle STRIDE Bedrohungstypen abgedeckt sind:

	S	T	R	I	D	E
Abbreviation	Spoofing	Tampering	Repudiation	Information Disclosure	Denial of Service	Elevation of Privilige
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

Tabelle 10: Zuordnung von Bedrohungen aus Part 2 zu STRIDE

Daraus ergibt sich folgende Bewertung (zur Interpretation der Bedrohungen siehe Abschnitt 7.1.3):

- '*spoofing*' wird in Part 2 mit '*message spoofing*' abgedeckt
- '*tampering*' ist breiter gefasst, als '*message alteration*' aus Part 2. Welche neuen Angriffe beim Bedrohungstyp '*tampering*' sich außerhalb der reinen Kommunikation ergeben, ist in Tabelle 17 beschrieben.
- '*repudiation*' fehlt in Part 2 und wurde deshalb in Tabelle 16 hinzugefügt. Zudem lässt sich hier auch das Schutzziel Nichtabstreitbarkeit ableiten, das in Part 2 nicht angegeben wird. Auch Standards wie zum Beispiel ISO/IEC 27000 [24] definieren dieses Schutzziel.
- '*information disclosure*' ist auf der Kommunikationsebene mit '*eavesdropping*' aus Part 2 abgedeckt. Weitere Angriffe werden in Tabelle 17 geschildert.
- '*denial of service*' umfasst mehr als '*message flooding*' aus Part 2. Deshalb wurde der Bedrohungstyp 'message flooding' durch 'denial of service' ersetzt.
- '*elevation of privilege*' ist durch '*compromising user credentials*' für den Bereich der Kommunikation abdeckt. Andere Aspekte, wie das Überschreiben von im Arbeitsspeicher geladenen Genehmigungen, wurden in Tabelle 17 betrachtet. Bedrohungen, die eine Manipulation von Rechten auf Address Space Ebene beinhalten, werden ebenfalls in Tabelle 17 erfasst.

Zusammenfassung:

Die Schutzziele aus Part 2 wurden um das Schutzziel ´*Nichtabstreitbarkeit*´ ergänzt.

Der neue Bedrohungstyp '*repudiation*´ und der weiter gefasste Bedrohungstyp ´*denial of service*´ wurden für die weitere Analyse verwendet.

Entsprechend werden in Tabelle Tabelle 14 das Schutzziel ´*Nichtabstreitbarkeit*´ und die Bedrohungstypen '*repudiation*´ und ´*denial of service*´ verwendet.

7.1.2 Prüfung der Definitionen der Schutzziele

In nachfolgender Tabelle werden die Definitionen der Schutzziele aus Part 2 den Definitionen aus ISO/IEC 27000 gegenübergestellt:

Schutzziel	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 *Client*, *Server*, 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

Tabelle 11: Vergleich von Definitionen aus der IT Sicherheit

Der ISO/IEC Standard gibt keine Definitionen für ´*Auditability*´ oder ´*Authorization*´ an. Die Definitionen für ´*Authentication, Availability, Confidentiality*´ und ´*Integrity*´ sind im ISO/IEC 27000 weiter gefasst als in Part 2. Part 2 fokussiert auf Kommunikation und Daten.

Die Definition von ´*Authorization*´ erscheint ungenügend, da nicht einfach Zugriffsrechte auf eine Ressource vergeben werden sollen, sondern es sollen nur die absolut notwendigen Zugriffsrechte (nach dem need-to-know Prinzip) korrekt an eine Ressource vergeben werden.

7.1.3 Prüfung der Definitionen der Bedrohungstypen

Im Folgenden wurden die Definitionen der Bedrohungstypen aus Part 2 bei Bedarf eingeschränkt oder verschärft, so dass die daraus resultierenden Bedrohungstypen disjunkte Mengen bilden.

Message flooding:

'*message flooding*' wird durch den Bedrohungstyp '*denial of service*' ersetzt, der nicht nur Angriffe auf die Kommunikation direkt umfasst, sondern allgemein Angriffe bezeichnet, die zum Ziel haben, die Verfügbarkeit eines IT Systems, einer Anwendung etc. zu stören oder ganz auszuschalten.

Message spoofing:

Der Bedrohungstyp message spoofing entspricht nicht der Definition von spoofing, wie sie in der IT Sicherheit üblicherweise verstanden wird. Spoofing beinhaltet immer das Vorgaukeln von Identitäten (Person, Applikation, Prozess, usw.) und diese Interpretation von message spoofing wurde für die weitere Analyse zugrunde gelegt. Um message spoofing von session hijacking zu unterscheiden, wurde davon ausgegangen, dass beim message spoofing ausschließlich ein neuer Kommunikationskanal aufgebaut wird. Das Eingreifen in eine bestehende Kommunikation fällt unter session hijacking.

Malformed messages:

Für die weitere Analyse wurde folgende Annahme getroffen: Bei der Bedrohung malformed messages erzeugt oder modifiziert ein Angreifer eine Nachricht so, dass sie vom Format her fehlerhaft ist. Eine Nachricht mit falscher Signatur würde nach dieser Definition unter message spoofing fallen (der Angreifer gaukelt eine Identität vor), eine Nachricht mit falschen Längenangaben von Feldgrößen unter malformed messages.

Rogue server:

Die in Part 2 enthaltene Definition von rogue server kann eng oder sehr weit interpretiert werden. Wenn ein beliebiger OPC UA Server als rogue server in die OPC UA Kommunikationsinfrastruktur eingeschleust werden kann, dann sind prinzipiell alle Schutzziele bedroht. Wenn der rogue server sich zudem als ein anderer, autorisierter OPC UA Server ausgeben kann, sind die Schutzziele deutlich stärker bedroht. Wie ein rogue server zu verstehen ist, wird in der Spezifikation aber nicht weiter beschrieben. In der weiteren Analyse wird die Bedrohung rogue server so interpretiert, dass der Angreifer einen OPC UA Server einschleusen kann, dieser aber nicht einen anderen, autorisierten OPC UA Server vorgaukeln kann.

Compromising user credentials:

Die Bedrohung compromising user credentials wurde so interpretiert, dass es nicht nur um eine Kompromittierung der bei der Benutzerauthentifizierung benötigten Daten geht, sondern auch um die Kompromittierung der bei der Applikationsauthentifizierung benötigten Daten.

7.1.4 Prüfung der Zuordnung Schutzziele versus Bedrohungen

Die Zuordnungen *Schutzziele versus Bedrohungstypen* gemäß Part 2 der Spezifikation wird in folgender Tabelle (Zuordnungen gemäß [3], Abschnitt 4.3 der Spezifikation) angegeben:

Bedrohung / **Schutzziel**	Message Flooding	Eaves-dropping	Message Spoofing	Message Alteration	Message Replay	Mal-formed Messages	Server Profiling	Session Hijacking	Rogue Server	Compromising User Credentials
Authentifizierung		(X)					(X)	X	X	
Autorisierung		(X)	X	X	X		(X)	X	X	X
Vertraulichkeit		X					(X)	X	X	X
Integrität		(X)	X	X		X	(X)	X		
Prüffähigkeit		(X)					(X)	X	X	
Verfügbarkeit	X	(X)				X	(X)	X	X	

Tabelle 12: Schutzziele versus Bedrohungstypen laut Spezifikation
Erläuterung: Die Kreuze in Klammern stellen indirekte Bedrohungen dar.

7.1.5 Ergebnisse

Aufgrund der teilweise unscharfen Definitionen von Bedrohungstypen (siehe Abschnitt 7.1.3) ist eine Zuordnung von Bedrohungstypen zu Schutzzielen nicht eindeutig möglich. Unsere Einschätzung der Auswirkung von Bedrohungstypen auf Schutzziele weicht von der der OPC Foundation ab. Entsprechend ergibt sich aus unserer Sicht folgende Tabelle:

Bedrohung / **Schutzziel**	Denial of Service	Eaves-dropping	Message Spoofing	Message Alteration	Message Replay	Mal-formed Messages	Server Profiling	Session Hijacking	Rogue Server	Compromising User Credentials	Repudiation
Authentifizierung		X		X	X		(X)	X	X	X	
Autorisierung		X	X	X	X		(X)	X	X		
Vertraulichkeit		X					(X)	X	X	X	
Integrität				X			(X)	X			
Prüffähigkeit				X			(X)	X	X		
Verfügbarkeit	X					X	(X)	X	X		
Nicht-abstreitbarkeit				X			(X)	X			X

Tabelle 13: Schutzziele versus Bedrohungstypen laut aktueller Analyse
Erläuterung: Die Kreuze in Klammern stellen indirekte Bedrohungen dar.

Die Liste der Schutzziele wurde um *´Nichtabstreitbarkeit´* erweitert: OPC UA besitzt Mechanismen, Anfragen mit zu protokollieren, und die Authentifizierung von Applikation und Benutzer auf der Basis von kryptographischen Signaturen bei securityMode 'Sign' und 'SignAndEncrypt', die das Schutzziel *´Nichtabstreitbarkeit´* erfüllen. Auch die Liste der Bedrohungstypen wurde aufgrund STRIDE um die Kategorie *´repudiation´* ergänzt. Der Bedrohungstyp message flooding wurde druch die Bedrohungstyp *´denial of service´* ersetzt. Für die Tabelle Tabelle 14 und im Folgenden gelten die in Abschnitt 7.1.3 erläuterten Interpretationen für die Bedrohungstypen.

Hier ein paar Beispiele zur Erläuterung unserer weiteren Änderungen:

- *Authentifizierung*:
  Kreuz bei '*message replay*': Angenommen, es wird bei einem Verbindungsaufbau eine Authentifizierung durchgeführt. Dann ist es grundsätzlich möglich, dass ein Angreifer die versendeten Nachrichten des einen Kommunikationspartners bei einem erfolgreichen Verbindungsaufbau aufzeichnet und zu einem späteren Zeitpunkt missbraucht, um erneut eine Verbindung samt Authentifizierung erfolgreich aufzubauen.
- Kreuz bei '*message alteration*': Auch die Fähigkeit, abgefangene Nachrichten zu ändern, ist eine Bedrohung für die Authentifizierung, weil ein Angreifer unter Umständen in der Lage ist, z. B. nach Änderung eines Zeitstempels sich erneut zu authentifizieren oder dank der Manipulation der Authentifizierungsdaten sich als jemand anderes zu authentifizieren.
- Vertraulichkeit:
  Kreuz bei '*compromising user credentials*': Ist ein Angreifer in der Lage Benutzermeldedaten zu kompromittieren, kann er sich eventuell über Anfragen an einen Server Zugang zu vertraulichen Daten verschaffen. Also ist die Kompromittierung von Benutzermeldedaten eine Bedrohung für das Schutzziel Vertraulichkeit.

Zusammenfassung der wesentlichen Erkenntnisse:

ID	Feststellung	Empfehlung
1	Die Definitionen der Schutzziele unterscheiden sich vom international anerkannten Standard ISO/IEC 27000 [24] für die Schutzziele *´Authentication, Availability, Confidentiality´* und *´Integrity´*	Es sollten die Definitionen des Standards ISO/IEC 27000 verwendet werden.
2	Das Schutzziel *´Nichtabstreitbarkeit´* fehlt	Die Schutzziele sollten um das Schutzziel *´Nichtabstreitbarkeit´*ergänzt werden.
3	Das Schutzziel *´Authorization´* ist ungenau definiert.	Die Definition sollte hervorheben, dass Rechte nach dem Need-to-Know Prinzip vergeben werden müssen.
4	Der Bedrohungstyp Abstreitbarkeit ist in der OPC UA Spezifikation nicht verwendet worden.	Erweiterung der Bedrohungstypen um die Abstreitbarkeit
5	Der Bedrohungstyp message flooding ist zu eng gefasst.	Erweiterung des Bedrohungstyps message flooding auf denial of service, da message flooding eine Untermenge von denial of service ist
6	Definitionen der Bedrohungstypen entsprechen nicht den üblichen Definitionen aus der IT Sicherheit oder sind zu unscharf	Die Definitionen sollten wie zum Beispiel in 7.1.3 vorgeschlagen genauer definiert werden und nicht nur auf die Kommunikation fokussieren (auch wenn sie dann nur auf die Kommunikation für OPC UA in der Spezifikation angewendet werden)
7	Die Zuordnung Schutzziel versus Bedrohungen ist sehr interpretierbar	Die Zuordnung Schutzziel versus Bedrohungen sollte überarbeitet werden.

Tabelle 14: Wesentliche Erkenntnisse aus der Prüfung der Schutzziele und Bedrohungstypen

7.2 Bedrohungsanalyse für das OPC UA Protokoll

7.2.1 Analyse gemäß Bedrohungen aus Part 2

Auf Grund der in Abschnitt 7.1 erläuterten Analyse der Schutzziele und Bedrohungstypen ergeben sich prinzipiell die in Tabelle 16 aufgeführten Bedrohungen. Dabei ist folgendes zu beachten:

Die Analyse der in Part 2 berücksichtigten Elemente, nämlich die Schutzziele und Bedrohungstypen, hat ergeben, dass der Fokus dieser Elemente ausschließlich der Absicherung der OPC UA Kommunikation gilt. Für die weitere Analyse verwenden wir die in Tabelle 14 ausgewiesenen Bedrohungen.
Der threat modelling Ansatz nach STRIDE ist etwas breiter ausgerichtet und stellt sicher, dass auch Bedrohungen, die nicht direkt auf die Kommunikation selbst abzielen, Berücksichtigung finden. Entsprechend wurden Bedrohungen auf Elemente der OPC UA Infrastruktur in Tabelle 17 aufgenommen. Schutz der Kommunikation und Schutz der Infrastruktur sind gleich wichtig und ergänzen sich. Wird ein Aspekt vernachlässigt, ist die Sicherheit insgesamt gefährdet. Entsprechend müssen beide in das Sicherheitskonzept eingebunden werden.

Für nachfolgende Tabellen 16 und 17 werden folgende Regeln für die CVSS Bewertung benutzt:

- AccessVector (AV): Bei den meisten Angriffen wird ein Zugang zum lokalen Netzsegment (A) benötigt, um mitzulauschen oder Nachrichten mit einer OPC UA Applikation auszutauschen. In vereinzelten Fällen ist ein lokaler Zugriff auf das System (L) erforderlich.
- AccessComplexity (AC): Angriffe, die nur das Versenden von Nachrichten ohne weitere Voraussetzungen beinhalten, wurden als einfach (L) eingestuft. Angriffe, bei denen mitgelauscht wird, wurden als mittelschwer (M) eingestuft. Komplexe Angriffe, die z. B. das Knacken von kryptographischen Schlüsseln erfordern, wurden als schwierig (C) bewertet.
- Authentication (Au): Die meisten Angriffe kommen ohne Authentifizierung (N) aus.
- ConfImpact (C): Wird die Verschlüsselung geknackt oder Zugang zu vertraulichen Daten ermöglicht, wurde der Parameter mit der höchsten Stufe (C) bewertet.
- IntegImpact (I): Wird die Signatur geknackt oder Nachrichten von einer anderen OPC UA Applikation vorgetäuscht, wurde der Parameter mit der höchsten Stufe (C) bewertet.
- AvailImpact (A): Alle denial of service Angriffe wurden mit der höchsten Stufe (C) bewertet.
- Exploitability (E): Alle Angriffe wurden mit 'Unproven' (U) bewertet, weil sie nur anhand der Analyse der Spezifikation erarbeitet wurde und noch nicht getestet werden konnten.

Wegen der konstanten Exploitability auf niedrigstem Niveau (U) kann ein Angriff maximal den Wert 8,5 erreichen. Angriffe, die eine tatsächlich Bedrohung darstellen, also nicht gleichzeitig C, I und A auf 'None' haben, erreichen einen Mindestwert von 0,7. Der Wert 0 bedeutet keine Bedrohung. Deshalb schlagen wir folgende Einstufung der Kritikalität anhand eines Ampelsystems vor:

niedrige Kritikalität	mittlere Kritikalität	hohe Kritikalität
0,7 – 3,3	3,4 – 5,9	6,0 – 8,5

Tabelle 15: CVSS Kritikalitätswerte für OPC UA

Bedrohungstyp	ID	Bedrohung	Kurzbeschreibung	Security-Mode	CVSS Vektor	CVSS Wert
Denial of service von nicht vertrauenswürdigem Client						
	1	HEL flooding	Der Client schickt dem Server wiederholt HEL Nachrichten. Reaktion: Der Server antwortet jeweils mit einer ACK Nachricht. Auswirkung: hohe Netzwerklast, geringe Prozessorlast	alle	AV:A/AC:L/Au:N/C:N/I:N/A:C/E:U	5,2
	2	ACK oder ERR flooding	Der Client schickt dem Server wiederholt ACK oder ERR Nachtrichten. Reaktion: Der Server antwortet jeweils mit einer ERR Nachricht. Auswirkung: hohe Netzwerklast, geringe Prozessorlast	alle	AV:A/AC:L/Au:N/C:N/I:N/A:C/E:U	5,2
	3	HEL + OPN request flooding	Der Client schickt dem Server wiederholt HEL und OPN Nachtrichten. Reaktion: Der Server antwortet mit ACK und ERR Nachrichten. Auswirkung: - hohe Netzwerklast, geringe Prozessorlast bei securityMode None - hohe Netzwerklast, mittlere Prozessorlast bei securityMode Sign (Verifizierung der Signatur) - hohe Netzwerklast, hohe Prozessorlast bei securityMode SignAndEncrypt (Entschlüsselung und Verifizierung der Signatur)	alle	AV:A/AC:L/Au:N/C:N/I:N/A:C/E:U	5,2
	4	FindServers() oder GetEndpoints() request flooding	Der Client baut einen SecureChannel mit securityMode None auf und schickt dem Server wiederholt FindServers() oder GetEndpoints() requests. Reaktion: Der Server antwortet jeweils mit einem FindServers() oder GetEndpoints() response. Auswirkung: hohe Netzwerklast, geringe Prozessorlast	alle	AV:A/AC:L/Au:N/C:N/I:N/A:C/E:U	5,2
	5	CLO request flooding	Der Client schickt dem Server wiederholt CLO Nachtrichten. Reaktion: Der Server antwortet jeweils mit einer ERR Nachricht. Auswirkung: hohe Netzwerklast, geringe Prozessorlast	alle	AV:A/AC:L/Au:N/C:N/I:N/A:C/E:U	5,2

Bedrohungstyp	ID	Bedrohung	Kurzbeschreibung	Security-Mode	CVSS Vektor	CVSS Wert
	6	flooding bestehend aus fehlerhaften Nachrichten	Der Client schickt dem Server wiederholt fehlerhafte Nachtrichten. Reaktion: Der Server antwortet jeweils mit einer ERR Nachricht. Auswirkung: hohe Netzwerklast, geringe Prozessorlast	alle	AV:A/AC:L/Au:N/C:N/I:N/A:C/E:U	5,2
Denial of service von vertrauenswürdigem Client						
	7	denial of service nach Aufbau einer gültigen Session	Der Client baut eine Session auf und schickt dem Server anschließend beliebige, komplexe Anfragen. Reaktion: Der Server wird die Anfragen nach und nach bearbeiten. Auswirkung: hohe Netzwerklast, hohe Prozessorlast	alle	AV:A/AC:L/Au:M/C:N/I:N/A:C/E:U	4,3
Denial of service von nicht vertrauenswürdigem Server						
	8	OPN response flooding	Der Server antwortet auf ein OPN request mit einem OPN response flooding. Reaktion: Der Client verwirft die Nachrichten nach Auswertung der requestId. Auswirkung: - hohe Netzwerklast, geringe Prozessorlast bei securityMode None - hohe Netzwerklast, mittlere Prozessorlast bei securityMode Sign (Verifizierung der Signatur) - hohe Netzwerklast, hohe Prozessorlast bei securityMode SignAndEncrypt (Entschlüsselung und Verifizierung der Signatur)	alle	AV:A/AC:L/Au:N/C:N/I:N/A:C/E:U	5,2
Weitere denial of service Angriffe (nicht message flooding)						
	9	Überlastung des Arbeitsspeichers	Ein Angreifer kann versuchen, den Arbeitsspeicher eines Servers zu überlasten, indem er z. B. die maximale Anzahl an Sessions, Subscriptions, monitoredItems und Queues verwendet. Eine andere Möglichkeit ist die Nutzung einer großen Anzahl an Variablen mit variabler Länge (string, array). Beide Angriffe setzen aber den erfolgreichen Aufbau von Sessions voraus.	None	AV:A/AC:M/Au:N/C:N/I:N/A:C/E:U	4,8

Bedrohungstyp	ID	Bedrohung	Kurzbeschreibung	Security-Mode	CVSS Vektor	CVSS Wert
	10	Vollschreiben der Festplatte	Verursacht der Angreifer, dass sehr große Datenmengen geschrieben werden, z. B. durch ein Missbrauch der file transfer Funktionalität (siehe [6] Annex C), kann es eventuell dazu führen, dass die Festplatte vollläuft.	None	AV:A/AC:M/Au:N/C:N/I:N/A:C/E:U	4,8
Eavesdropping						
	11	Mitlauschen	Ist ein Angreifer in der Lage, im lokalen Subnetz bei der Kommunikation zwischen zwei OPC UA Applikationen mitzulauschen, kann er vertrauliche Informationen abgreifen.	None, Sign	AV:A/AC:M/Au:N/C:P/I:N/A:N/E:U	2,5
	12	Mitlauschen	Hat ein Angreifer, z. B. ein Innentäter, physischen Zugang zum Monitoring Port eines Switchs, kann er auf diese Weise die OPC UA Kommunikation mithören.	None, Sign	AV:L/AC:L/Au:N/C:P/I:N/A:N/E:U	1,8
Message spoofing						
	13	Erzeugen von Nachrichten als Client	Findet keine Applikationsauthentifizierung statt, kann ein Angreifer Nachrichten erzeugen und sich als Client ausgeben.	None	AV:A/AC:L/Au:N/C:P/I:P/A:N/E:U	4,1
Message alteration						
	14	Manipulation von abgegriffenen Nachrichten	Ist ein Angreifer in der Lage, Nachrichten zwischen einem Client und einem Server abzugreifen, kann er diese manipulieren, bevor er sie an den eigentlichen Empfänger weiterleitet.	None	AV:A/AC:M/Au:N/C:C/I:C/A:N/E:U	6,2
	15	Unterschlagen von abgegriffenen Nachrichten	Ist ein Angreifer in der Lage, Nachrichten zwischen einem Client und einem Server abzugreifen, kann er bestimmte unterschlagen.	None	AV:A/AC:M/Au:N/C:N/I:C/A:N/E:U	4,8
Message replay						
	16	UA TCP Nachrichten	Ein Angreifer kann HEL und ACK Nachrichten erneut versenden. Dadurch wird die Verbindung abgebrochen und muss neu aufgebaut werden.	alle	AV:A/AC:M/Au:N/C:N/I:N/A:C/E:U	4,8

Bedrohungstyp	ID	Bedrohung	Kurzbeschreibung	Security-Mode	CVSS Vektor	CVSS Wert
	17	UASC Nachrichten	Erneut versendete UASC Nachrichten werden wegen der falschen sequenceNumber sofort erkannt und verworfen und haben somit keine Auswirkung auf die Sicherheit.	alle	AV:A/AC:M/Au:N/C:N/I:N/A:N/E:U	0
Malformed message						
	18	kaputte Nachricht	Hat ein Angreifer Zugang zum lokalen Netz, kann er kaputte Nachrichten versenden.	None	AV:A/AC:L/Au:N/C:N/I:C/A:N/E:U	5,2
Server profiling						
	19	UA TCP Nachrichten	Ein Angreifer kann HEL und ACK Nachrichten verwenden, um Informationen über die Konfiguration des Servers zu erlangen und daraus eventuell Rückschlüsse über den Server ziehen.	alle	AV:A/AC:L/Au:N/C:P/I:N/A:N/E:U	2,8
	20	FindServers() oder GetEndpoints() requests	Da diese Discovery Dienste immer unsigniert und unverschlüsselt verwendet werden, kann ein Angreifer sie, unabhängig vom securityMode missbrauchen, um z. B. Produktinformationen wie Hersteller- und Produktname zu erlangen.	alle	AV:A/AC:L/Au:N/C:P/I:N/A:N/E:U	2,8
	21	UASC Nachrichten	Findet keine Applikationsauthentifizierung statt, kann ein Angreifer zumindest dank des Aufbaus eines SecureChannels weitere Informationen über den Server ausfindig machen. Wird für den Aufbau einer Session nicht zwingend eine Benutzerauthentifizierung benötigt, können auch dank dieses Vorgangs Informationen über den Server gewonnen werden.	None	AV:A/AC:L/Au:N/C:P/I:N/A:N/E:U	2,8
Session hijacking						
	22	Übernahme eines SecureChannels oder einer Session	Müssen Nachrichten nicht signiert werden, kann ein Angreifer versuchen, die bestehende Kommunikation zwischen einem Client und einem Server zu kapern. Hierfür muss er verschiedene Parameterwerte erraten oder durch Mitlauschen in Erfahrung bringen.	None	AV:A/AC:H/Au:N/C:P/I:C/A:P/E:U	4,9

Bedrohungstyp	ID	Bedrohung	Kurzbeschreibung	Security-Mode	CVSS Vektor	CVSS Wert
			Rogue server			
	23	Betrieb eines nicht vertrauenswürdigen Servers	Ist ein Angreifer in der Lage, seinen eigenen OPC UA Server aufzusetzen, kann er eventuell vertrauliche Informationen abgreifen, wenn sich Clients mit dem Server verbinden.	None	AV:A/AC:M/Au:N/C:P/I:P/A:N/E:U	3,7
	24	Übernahme eines vertrauenswürdigen Servers	Ist ein Angreifer in der Lage, die Kontrolle über einen bestehenden Server zu übernehmen, kann er den ausgehenden Informationsfluss beeinflussen und den eingehenden mitlesen.	alle	AV:A/AC:H/Au:N/C:P/I:C/A:N/E:U	4,5
Compromising user credentials						
	25	brute force Angriff auf private Schlüssel bei Signatur	Werden Nachrichten signiert, aber nicht verschlüsselt, kann ein Angreifer anhand der mitgeschnittenen Inhalte versuchen, die zu den ApplicationInstance Zertifikaten passenden privaten Schlüssel zu errechnen.	Sign	AV:A/AC:H/Au:N/C:N/I:C/A:N/E:U	3,9
	26	brute force Angriff auf private Schlüssel	Werden Nachrichten verschlüsselt, kann ein Angreifer anhand der mitgeschnittenen Inhalte versuchen, die zu den ApplicationInstance Zertifikaten passenden privaten Schlüssel zu errechnen.	SignAnd-Encrypt	AV:A/AC:H/Au:N/C:C/I:C/A:N/E:U	5,3
	27	brute force Angriff auf symmetrischen Signaturschlüssel	Werden Nachrichten signiert, kann ein Angreifer anhand der mitgeschnittenen Inhalte versuchen, die von Client und Server nonces abgeleiteten Signaturschlüssel zu errechnen.	Sign	AV:A/AC:H/Au:N/C:N/I:C/A:N/E:U	3,9
	28	brute force Angriff auf symmetrischen Verschlüsselungs-schlüssel	Werden Nachrichten verschlüsselt, kann ein Angreifer anhand der mitgeschnittenen Inhalte versuchen, die von Client und Server nonces abgeleiteten Verschlüsselungsschlüssel zu errechnen. Auch ein geheimnisbasiertes userIdentityToken gilt dann als geknackt, zertifikatsbasierte userIdentityTokens sind davon unbetroffen.	SignAnd-Encrypt	AV:A/AC:H/Au:N/C:C/I:N/A:N/E:U	3,9

Bedrohungstyp	ID	Bedrohung	Kurzbeschreibung	Security-Mode	CVSS Vektor	CVSS Wert
	29	brute force Angriff auf private Schlüssel zertifikatsbasierter userIdentityToken	Werden Nachrichten nicht verschlüsselt, kann ein Angreifer anhand der mitgeschnittenen Inhalte versuchen, die zu den zertifikatsbasierten userIdentityTokens passenden privaten Schlüssel zu errechnen.	None, Sign	AV:A/AC:H/Au:N/C:C/I:N/A:N/E:U	3,9
	30	dictionary Angriff auf geheimnisbasierten userIdentityToken	Ein Innentäter mit Zugang zu einem vertrauenswürdigen Client hat die Möglichkeit, beliebig viele Passwörter zur Authentifizierung eines Benutzers mit mehr Rechten, als er selbst hat, durchzuprobieren.	alle	AV:L/AC:L/Au:S/C:C/I:N/A:N/E:U	3,9
Repudiation						
	31	Ausschalten der Auditfunktionalität	Hat ein Angreifer Zugriff auf die Konfiguration einer Applikation, kann er z. B. das Auditing ausschalten, so dass seine anschließenden böswilligen Aktionen zumindest auf der einen Seite nicht aufgezeichnet werden.	None	AV:L/AC:L/Au:N/C:N/I:P/A:N/E:U	1,8

Tabelle 16: potentielle Bedrohungen auf die OPC UA Kommunikation

7.2.2 Analyse der Bedrohungen auf Elemente der OPC UA Infrastruktur

Bedrohungstyp	ID	Bedrohung	Kurzbeschreibung	Security-Mode	CVSS Vektor	CVSS Wert
Angriffe auf die OPC UA Infrastruktur						
	32	Auslesen der privaten Schlüssel	Kann sich ein Angreifer Zugriff auf das Dateisystem eines Geräts verschaffen, auf dem eine OPC UA Applikation läuft, so kann er unter Umständen dort abgelegte private Schlüssel auslesen.	alle	AV:L/AC:L/Au:N/C:C/I:C/A:N/E:U	5,6
	33	Manipulieren von trust oder issuer lists	Kann sich ein Angreifer Zugriff auf das Dateisystem eines Geräts verschaffen, auf dem eine OPC UA Applikation läuft, so kann er unter Umständen dort abgelegte trust oder issuer lists manipulieren, so dass ein vom Angreifer erzeugtes Zertifikat bei der Prüfung als vertrauenswürdig angesehen wird.	Sign, SignAnd-Encrypt	AV:L/AC:L/Au:N/C:C/I:C/A:N/E:U	5,6
	34	Löschen der trust oder issuer lists	Kann sich ein Angreifer Zugriff auf das Dateisystem eines Geräts verschaffen, auf dem eine OPC UA Applikation läuft, so kann er unter Umständen dort abgelegte trust oder issuer lists löschen oder leeren, so dass überhaupt keine Zertifikate mehr bei der Prüfung als vertrauenswürdig angesehen werden.	Sign, SignAnd-Encrypt	AV:L/AC:L/Au:N/C:N/I:N/A:C/E:U	4,2
	35	Manipulieren von Konfigurationsdateien	Kann sich ein Angreifer Zugriff auf das Dateisystem eines Geräts verschaffen, auf dem eine OPC UA Applikation läuft, so kann er unter Umständen sicherheitsrelevante Parameter in dort abgelegten Konfigurationsdateien manipulieren, wie z. B. die Änderung des Pfads zum CertificateStore.	alle	AV:L/AC:L/Au:N/C:C/I:C/A:N/E:U	5,6
	36	Manipulieren oder Löschen von Auditdaten	Kann sich ein Angreifer Zugriff auf ein System verschaffen, auf dem OPC UA Auditdaten abgelegt werden, so kann er unter Umständen Einträge löschen oder manipulieren, um seine Aktivitäten zu verschleiern.	alle	AV:L/AC:L/Au:N/C:N/I:C/A:N/E:U	4,2
	37	Manipulieren der Uhrzeit	Kann ein Angreifer die Uhrzeit eines Systems manipulieren, auf dem eine OPC UA Applikation läuft, dann werden von dieser	alle	AV:A/AC:L/Au:N/C:N/I:P/A:N/E:U	2,8

Bedrohungstyp	ID	Bedrohung	Kurzbeschreibung	Security-Mode	CVSS Vektor	CVSS Wert
			Applikation erzeugte Auditdaten gefälscht.			
	38	Nichterreichbarkeit der CRLs	Ist der Abruf aktueller CRLs bei der Prüfung von Zertifikaten zwingend erforderlich, so kann ein Angreifer dies ausnutzen, indem er die Erreichbarkeit der CRLs unterbricht.	Sign, SignAnd-Encrypt	AV:A/AC:M/Au:N/C:N/I:N/A:C/E:U	4,8
	39	Angriffe auf das Betriebssystem	Erlangt ein Angreifer die Kontrolle über das Betriebssystem, auf dem eine OPC UA Applikation läuft, so kann er u.a. die Applikation beenden, den Arbeitsspeicher auslesen, usw.	alle	AV:A/AC:M/Au:N/C:C/I:C/A:C/E:U	6,7
	40	Angriffe auf die Implementierung kryptographischer Algorithmen oder auf Zufallszahlen-generatoren	Sind Schwachstellen in der Kryptobibliothek, auf die eine OPC UA Applikation zurückgreift, bekannt oder ist die Entropie zur Erzeugung von Zufallszahlen unzureichend, so kann ein Angreifer sich unter Umständen Zugang zu geschützten Daten verschaffen.	Sign, SignAnd-Encrypt	AV:A/AC:M/Au:N/C:C/I:N/A:N/E:U	4,8
	41	Manipulation von Zugriffsrechten im Address Space	Gelingt einem Angreifer die Manipulation von Zugriffsrechten auf den Address Space des Servers, kann der Angreifer vertrauliche Daten einsehen oder Daten manipulieren.	alle	AV:A/AC:M/Au:N/C:P/I:P/A:N/E:U	3,7

Tabelle 17: potentielle Bedrohungen auf die OPC UA Infrastruktur

7.2.3 Ergebnisse

In folgender Tabelle werden nur die als kritisch angesehenen Bedrohungen genannt und welche Gegenmaßnahmen gegen die Bedrohung schützen können.

ID	Bedrohung	Gegenmaßnahmen
1	Manipulation von abgegriffenen Nachrichten: Ist ein Angreifer in der Lage, Nachrichten zwischen einem Client und einem Server abzugreifen, kann er diese manipulieren, bevor er sie an den eigentlichen Empfänger weiterleitet.	**securityMode:** securityMode None sollte komplett deaktiviert werden, um spätere Fehlkonfigurationen oder downgrade Angriffe zu unterbinden. Spielt die Vertraulichkeit des über OPC UA laufenden Nachrichtenaustauschs eine Rolle, sollte SignAndEncrypt gewählt werden. **Wahl der kryptographischen Algorithmen:** Als securityPolicyUri wird die zur Zeit sicherste empfohlen, nämlich `Basic256Sha256`.
2	Angriffe auf das Betriebssystem Erlangt ein Angreifer die Kontrolle über das Betriebssystem, auf dem eine OPC UA Applikation läuft, so kann er u.a. die Applikation beenden, den Arbeitsspeicher auslesen, usw.	**Härtung des Betriebssystems:** Ist ein Angreifer in der Lage, auf Teile des Arbeitsspeichers zuzugreifen (Bedrohung 39), in denen eine OPC UA Applikation läuft, kann er unter Umständen an sonst geheime Informationen wie z. B. den privaten Schlüssel gelangen oder Autorisierungen überschreiben. Entsprechend ist auch die Sicherheit des darunter liegenden Systems wichtig. Diese kann durch Härtungsmaßnahmen erheblich verbessert werden.

Tabelle 18: Besonders kritische Bedrohungen

7.3 Analyse der Schutzmechanismen auf Parameterebene

7.3.1 Erläuterung der Analysetabelle

Basierend auf den Ausführungen in [3] Abschnitt 5.2 und unserer Ergänzung bezüglich der Nichtabstreitbarkeit scheint OPC UA Mechanismen zum Schutz gegen alle in Tabelle 14 aufgelisteten Bedrohungstypen zu bieten. Dies wurde anhand einer detaillierten Analyse bis auf Parameterebene geprüft. Die Ergebnisse dieser Analyse der Sicherheit von OPC UA auf der Grundlage der in den ausgetauschten Nachrichten enthaltenen Parameter sind aufgrund des Umfangs in getrennt herunterladbarer Datei [25] dargestellt.

Das Dokument ist wie folgt aufgebaut:

Welchen Schutz OPC UA bietet, hängt entscheidend vom Parameter securityMode ab. Deshalb wurde die Darstellung der Ergebnisse der Übersichtlichkeit halber in drei Tabellen aufgeteilt: Über die Reiter kann man sich die Ergebnisse für securityMode 'None', 'Sign' und 'SignAndEncrypt' anzeigen lassen. Der vierte Reiter fasst die Ergebnisse aller Modi zusammen und ist weiter unten erklärt.

Die drei ersten Tabellen sind gleich aufgebaut: Die Zeilen stellen in Form einer links auf- und zuklappbaren Baumstruktur dar, wie die Nachrichten bei OPC UA TCP und UASC (UA Secure Conversation) aufgebaut sind. Es wurden die Nachrichten ausgewählt, die für die Kommunikation entscheidend sind und somit eine wichtige Rolle bei der IT Sicherheit spielen, nämlich die Findung der Kommunikationspartner (Discovery Dienst) und der Verbindungsaufbau (SecureChannel und Session): Bei OPC UA TCP handelt es sich um drei mögliche Nachrichtentypen (HEL, ACK und ERR). Bei UASC wurden die Nachrichten für den SecureChannel, die Session und den Discovery Dienst berücksichtigt.

Die Spalten stellen zwei Arten von Informationen dar: Die linken Spalten (A bis L) dienen der Beschreibung der Nachrichtenparameter, die rechten (M bis X) zeigen die Ergebnisse bezüglich der Auswertung der Sicherheit an.

Hier eine detailliertere Aufschlüsselung der Spalteninhalte:

- Die Spalten A bis F stellen die verschiedenen Ebenen der Protokolle, Nachrichtenteile und Parameter dar.
- In der Spalte G sind die Datentypen der Parameter spezifiziert, Spalte H zeigt, falls vorhanden, passende Defaultwerte an.
- Die Spalten I und J heben farblich hervor, welche Teile der jeweiligen Nachrichten signiert, respektive verschlüsselt sind:
- Spalte I: grau = unsigniert blau = signiert
- Spalte J: grau = unverschlüsselt grün = verschlüsselt
- Außerdem wird innerhalb der Flächen angezeigt, mit welchem Schlüssel die Operation durchgeführt wurde.
- Die Spalten K und L enthalten Erläuterungen zum besseren Verständnis der Parameter.
- In den Spalten M bis W werden die Bedrohungen aus Part 2, ergänzt durch die Abstreitbarkeit, aufgelistet. Die letzte Spalte ermöglicht es, die Einträge in den Spalten mit den Bedrohungen zu erläutern.

Die Ergebnisse der Analyse sind folgendermaßen in der Tabelle eingetragen: Unterstützt ein Parameter beim Schutz gegen eine bestimmte Bedrohung, wird in der jeweiligen Zeile (Parameter) und Spalte (Bedrohung) eine '1' eingetragen. Beispiel: Die SecureChannelId trägt zum Schutz gegen die Wiederverwendung von Nachrichten bei. Deshalb ist in der Zelle Q44 eine '1' eingetragen.

Zusammenfassend werden außerdem die Einträge für die einzelnen Bedrohungen auf Nachrichtenebene aufsummiert. Kommt dabei eine Zahl größer als 0 heraus, bedeutet dies, dass dieser Nachrichtentyp einen Schutz gegen diese Bedrohung bietet. Diese Aufsummierung der Einträge je Bedrohung wurde auch auf Dienst- und Protokollebene fortgesetzt. Die Ergebnisse auf Dienstebene wurden in der Tabelle im letzten Reiter übertragen.

Die Interpretation, ob ein Parameter den Schutz gegen eine Bedrohung erhöht, ist relativ weit gefasst: Es bedeutet insbesondere nicht, dass ein Parameter alleine ausreichend ist, um gegen die jeweilige Bedrohung zu schützen, sondern lediglich, dass er dazu beiträgt. Und die Tatsache, dass mehrere Parameter zum Schutz gegen eine Bedrohung beitragen, bedeutet auch nicht automatisch, dass die Bedrohung damit vollständig abgewehrt ist.

Beispiele:

- Die Parameter SecureChannelId (Zelle Q44), SequenceNumber (Zelle Q53) und authenticationToken (Zelle Q57) tragen alle drei dazu bei, und zwar unabhängig vom securityMode, dass replay Angriffe durch ein erneutes Versenden einer Nachricht ohne Anpassung unmöglich sind.
- Das Ziel eines Angreifers bei einem flooding Angriff kann nicht nur die Erzeugung einer hohen Netzwerklast, sondern auch die einer hohen Prozessorlast sein. Beim securityMode 'SignAndEncrypt' müssen die meisten Nachrichten erst entschlüsselt und anschließend deren Signaturen verifiziert werden. Durch ein hohes Nachrichtenaufkommen steigt die Prozessorlast also auch stark.
- Der Parameter SecureChannelId alleine bietet etwas Schutz gegen solche flooding Angriffe (Zelle M44): Der Parameter wird nämlich noch vor der Entschlüsselung einer Nachricht ausgewertet. Passt die ID zu keinem bestehenden SecureChannel, wird die Nachricht nicht weiter ausgewertet. Dadurch hält sich der Anstieg der Prozessorlast in Grenzen, die hohe Netzwerklast kann mit diesem Parameter aber nicht vermieden werden.
- Ähnlich ist es bei der Aushandlung der Puffer- und Nachrichtengrößen und der maximalen Anzahl an Teilnachrichten in OPC UA TCP 'HEL' (Zellen M10 bis M13) und 'ACK' (Zellen M23 bis M26) Nachrichten: Sie verhindern, dass ein Angreifer beliebig große und beliebig viele Teilnachrichten schickt, bieten aber keinen ganzheitlichen Schutz gegen flooding Angriffe an.

7.3.2 Detaillierte Erläuterung der Ergebnisse der Analysetabelle

Fügt man die Zahlen nach securityMode sortiert auf Diensteebene zusammen, ergibt sich folgende Tabelle:

security-Mode	Layer or Service	Denial of Service	Eaves-dropping	Message Spoofing	Message Alteration	Message Replay	Malformed Messages	Server Profiling	Session Hijacking	Rogue Server	Compromising User credentials	Repudiation
	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

Tabelle 19: Wirksamkeit der OPC UA Schutzmaßnahmen

: kein Schutz

: geringer Schutz

: Schutz, der die Möglichkeiten eines Angreifers einschränkt, diese Art von Angriffen aber nicht verhindert

: wirksamer Schutz (Angriffe dieser Art setzen kryptographische Angriffe voraus)

Es ist wichtig zu betonen, dass die Zahlen lediglich als Hilfe für die Interpretation herangezogen wurden: Eindeutig ist nur, falls bei einem securityMode für eine Bedrohung bei den drei Diensten eine 0 steht, dann kann kein Schutz vorhanden sein. Aber eine quantitative Auswertung ist nicht möglich: Höhere Werte deuten nicht zwangsläufig auf einen besseren Schutz hin.

OPC UA TCP:

Da der Parameter securityMode Teil von UASC ist, hat er keine Auswirkung auf die darunter liegende Schicht OPC UA TCP.

denial of service: Wie bereits im letzten Beispiel beschrieben, bietet OPC UA TCP nur minimalen Schutz gegen flooding Angriffe.

malformed message: Die Festlegung von Obergrenzen für Puffer- und Nachrichtengrößen und der maximalen Anzahl an Teilnachrichten bietet auch sehr eingeschränkt Schutz gegen fehlerhafte Nachrichten, weil z. B. eine Nachricht mit unzulässiger Größe früh verworfen wird.

server profiling: Weil die Anzahl der weltweit verfügbaren unterschiedlichen OPC UA SDKs recht übersichtlich ist, ist nicht ausgeschlossen, dass allein die auf OPC UA TCP Ebene gewonnenen Informationen zur Erstellung eines eindeutigen Serverprofils ausreichen könnten.

securityMode None:

denial of service: Die Schutzvorkehrungen gegen flooding Angriffe sind für die Parameter die gleichen, wie bei den zwei anderen Modi auch, allerdings mit unterschiedlichen Auswirkungen: Einerseits ist es für einen Angreifer schwieriger, das Opfer zu aufwendigen Rechenoperationen zu zwingen, da weder signiert, noch verschlüsselt wird. Andererseits hat ein Angreifer durch das Wegfallen der Applikationsauthentifizierung mehr Möglichkeiten, den Arbeitsspeicher zu überlasten und die Festplatte vollzuschreiben.

eavesdropping: Weil, mit der Ausnahme von geheimnisbasierten userIdentityTokens, bei securityMode None keine Verschlüsselung stattfindet, ist das Schutzziel Vertraulichkeit nicht erfüllt.

message spoofing, message alteration: Gegen das Erzeugen und die Änderung von Nachrichten, wie z. B. bei einem Man-in-the-Middle Angriff, schützt securityMode None wegen der fehlenden Applikationsauthentifizierung überhaupt nicht.

message replay und session hijacking: Gegen das erneute Versenden von Nachrichten und die Übernahme von Sessions greifen geeignete Mechanismen, wie z. B. die Prüfung der sequenceNumber. Allerdings setzt dies zwingend voraus, dass der Angreifer nicht zusätzlich mitlauscht.

malformed message: Durch die nicht vorhandene Applikationsauthentifizierung hat ein Angreifer viele Möglichkeiten, fehlerhafte Nachrichten zu erzeugen und vom Opfer auswerten zu lassen. Allein die Benutzerauthentifizierung, falls zwingend erforderlich, verhindert den Aufbau einer Session.

server profiling: Der Schutz gegen die Erstellung eines Serverprofils beschränkt sich auf die Tatsache, dass die Benutzerauthentifizierung, falls zwingend erforderlich, den Aufbau einer Session verhindert.

rogue server: Die in Part 7 Tabelle 3 vorgeschriebene Applikationsauthentifizierung beim RegisterServer Discovery Dienst verhindert, dass ein Angreifer seinen nicht autorisierten Server bei Discoveryservern anmeldet und somit dass dieser auf diesem Weg über FindServers von Clients gefunden wird. Allerdings wird in [13] auch die Möglichkeit geboten, dass sich ein rogue server über mDNS bekannt macht.

Durch die fehlende Applikationsauthentifizierung hat ein Client anschließend keine Möglichkeit mehr, einen nicht vertrauenswürdigen Server von einem vertrauenswürdigen zu unterscheiden.

compromising user credentials: Ist, wie von der Spezifikation gefordert, entweder die securityPolicyUri oder eine userTokenPolicy gesetzt, also nicht 'None', so ist die Sicherheit eines geheimnisbasierten userIdentityTokens bei der Benutzerauthentifizierung gewährleistet.

repudiation: Dadurch ist die Nichtabstreitbarkeit bezüglich der Benutzerauthentifizierung auch gegeben. Die Nichtabstreitbarkeit bezüglich der Applikation kann nicht eingehalten werden, da keine solche Authentifizierung stattfindet.

securityMode Sign und SignAndEncrypt:

eavesdropping: Wie erwartet, unterscheiden sich die zwei Modi im Wesentlichen in dem Schutz der Vertraulichkeit: Im securityMode Sign werden zwar die OpenSecureChannel Anfragen und Antworten verschlüsselt, der weitere Nachrichtenaustausch findet aber unverschlüsselt statt. Insbesondere die Inhalte aller Anfragen und Antworten, die im Rahmen einer bestehenden Session ausgetauscht werden, können von einem mitlauschenden Angreifer gelesen werden, weil sie im Klartext versendet werden. Nur geheimnisbasierte userIdentityTokens, die bei activateSession Anfragen zur Authentifizierung des Benutzers gesendet werden, sind verschlüsselt und somit geschützt.

Dagegen werden bei SignAndEncrypt alle Nachrichten beim SecureChannel und der Session verschlüsselt. Allein beim Discovery Dienst müssen alle Applikationen, unabhängig vom sonst eingestellten securityMode, FindServers und GetEndpoints Anfragen und Antworten auch ohne Sicherheit unterstützen, also unverschlüsselt und unsigniert.

denial of service: Bestimmte Parameter schränken die Möglichkeiten eines Angreifers bei flooding Angriffen ein. Insbesondere die erzwungene Applikationsauthentifizierung erschwert ganz erheblich einen flooding Angriff auf Sessionebene bzw. macht diesen ohne Kenntnis weiterer Informationen wie privaten Schlüsseln unmöglich. Nichtsdestotrotz hat der Angreifer gerade wegen der Schutzmaßnahmen, die aufwendige Entschlüsselungs- und Signaturverifizierungsschritte mit sich bringen, die Möglichkeit, zusätzlich zu einer hohen Netzwerklast auch den Prozessor deutlich stärker zu beanspruchen, als es mit securityMode None der Fall ist. Also bietet UASC nur einen unzureichenden Schutz gegen flooding Angriffe an.

Diese Schwachstelle muss jedoch relativiert werden, weil ein Angreifer mit deutlich geringerem Aufwand durch flooding auf IP oder TCP Protokollebene vermutlich vergleichbare Ergebnisse erzielen würde.

server profiling: Durch die erforderliche Applikationsauthentifizierung hat ein Angreifer weniger Möglichkeiten über den SecureChannel Informationen zu gewinnen, wobei auch eine fehlgeschlagene Authentifizierung eventuell Informationen über den Server preisgibt. Sessions kann ein Angreifer bei den securityModes nicht, aber FindServers und GetEndpoints sind ohne Authentifizierung möglich.

Gegen alle anderen Bedrohungen schützen beide Modi ausreichend, mit minimal besserem Schutz bei SignAndEncrypt. Spielt die Vertraulichkeit keine Rolle, reicht wahrscheinlich securityMode Sign.

7.3.3 Schlussfolgerung

securityMode None bietet wenig bis keinen Schutz vor IT Sicherheitsangriffen und sollte unbedingt vermieden werden. Spielt die Vertraulichkeit keine Rolle, bietet securityMode Sign einen angemessenen Schutz gegen die betrachteten Bedrohungen. Werden vertrauliche Daten ausgetauscht, ist securityMode SignAndEncrypt erforderlich.

Die Bedrohungen ´*denial of service*´ und ´*server profiling*´ können mit OPC UA Schutzmechanismen nur abgemildert, aber nicht komplett abgewehrt werden. Entsprechend müssen zusätzliche Maßnahmen außerhalb von der OPC UA Infrastruktur zum Schutz gegen diese Bedrohungen beitragen.

Zudem zeigt die Analyse, dass mit securityMode SignAndEncrypt allen Bedrohungen bis auf ´*denial of service*´ und ´*server profiling*´ ausreichend begegnet werden kann. Somit werden auch die Schutzziele bis auf die Verfügbarkeit erreicht.

7.4 Weitere Feststellungen bzgl. Design

Folgende Designentscheidungen beim Entwurf des Protokolls erschweren aus unserer Sicht das Erreichen eines hohen Maßes an Sicherheit:

ID	Feststellung	Empfehlung
1	Mangelnde Flexibilität beim Profilkonzept: Die Spezifikation sieht aktuell vor, dass alle zulässigen Kombinationen von asymmetrischen und symmetrischen Signatur- und Verschlüsselungsalgorithmen samt Schlüssellängen in Part 7 als Profile aufgelistet werden. Allerdings erscheinen nur ca. alle drei Jahre neue Versionen des OPC UA Standards. Dies bedeutet, dass innerhalb eines solchen Zeitraums keine Anpassungen bezüglich der Kryptographie möglich sind: Es können weder neue, sicherere und/oder effizientere Algorithmen oder Schlüssellängen aufgenommen werden, noch können als unsicher geltende entfernt werden.	Es sollte überlegt werden, wie notwendige Anpassungen bzgl. Kryptographie zeitnah in die Spezifikation integriert werden können.
2	Fehlende forward secrecy: Das aktuelle Verfahren zur Erzeugung der symmetrischen Schlüssel unterstützt keine forward secrecy. Ist ein Angreifer in der Lage, die beim Aufbau eines SecureChannels ausgetauschten Client- und Servernonces herauszufinden, so kann er nachträglich Nachrichten entschlüsseln, die über diesen SecureChannel ausgetauscht wurden. Es ist offen, warum nicht auf Verfahren zurückgegriffen wurde, die sich seit Jahrzehnten bewährt haben und forward secrecy unterstützen, wie z. B. DHE-RSA oder DHE-DSS, die auf den Diffie-Hellman Schlüsselaustausch basieren.	Es sollten bewährte Verfahren für den Schlüsselaustausch verlangt werden.
3	Einführung von mDNS: In Part 12 Abschnitt 4.3.5 wird beschrieben, wie mDNS beim Discovery Dienst eingesetzt wird. Die Untersuchung der Sicherheit von mDNS ist außerhalb des Rahmens dieses Projekts, kann aber zusätzliche Risiken bergen.	Die IT Sicherheit von mDNS sollte untersucht werden.

Tabelle 20: Wesentliche Erkenntnisse bezüglich Design

Allerdings kann man manche Kritikpunkte angesichts des aktuellen Einsatzgebiets von OPC UA relativieren: forward secrecy z. B. ist nur dann relevant, wenn streng vertrauliche Daten ausgetauscht werden, die über längere Zeiträume weiter geschützt bleiben müssen. Dies ist nach unserem Verständnis im Bereich der Automatisierung, wo OPC UA weit verbreitet ist, nicht gegeben. Es ist aber nicht ausgeschlossen, dass das Protokoll in Zukunft vermehrt in anderen Bereichen eingesetzt wird, in denen solche Anforderungen gelten.

7.5 Zusammenfassung der Ergebnisse der Gesamtanalyse

Die durchgeführte Spezifikationsanalyse hat gezeigt, dass OPC UA ein hohes Maß an Sicherheit bietet, wenn securityMode Sign und vor allen Dingen securityMode SignAndEncrypt verwendet wird. Es konnten keine systematischen Fehler entdeckt werden. Dies hat insbesondere die Analyse der Schutzmechanismen auf Parameterebene aus Abschnitt 7.3 gezeigt.

Allein die Abwehr von ´denial of service´ Angriffen erweist sich naturgemäß als schwierig. Dieses Problem ist aber nicht OPC UA spezifisch, sondern betrifft Netzwerkprotokolle allgemein. ´denial of service´ Angriffen kann nur durch eine angemessene IT Sicherheitsarchitektur begegnet werden.

Wie in den Abschnitten 5, 6, 7.1.5 und 7.4 erläutert, besteht für die Spezifikation Verbesserungspotential, das in Tabelle Tabelle 21 nochmals zusammenfassend dargestellt ist.

ID	Feststellung	Empfehlung
Kapitel 5, Bestandsaufnahme		
1	Der Name „Secure Channel“ suggeriert IT Sicherheit, die bei securityMode None aber nicht enthalten ist. Leider kann der Name nicht mehr geändert werden.	keine
2	Es werden für Zufallszahlengeneratoren keine Hinweise auf Mindestanforderungen in der OPC UA Spezifikation angegeben.	Es sollten für Zufallszahlengeneratoren Hinweise auf Mindestanforderungen von anerkannten Institutionen (zum Beispiel BSI) in der OPC UA Spezifikation angegeben werden.
3	Es werden keine Hinweise zu veralteten oder als unsicher geltenden Algorithmen, insbesondere SHA-1, und Cipher Suites, gegeben.	Es sollten Hinweise zu veralteten oder als unsicher geltenden Algorithmen, insbesondere SHA-1, und Cipher Suites, gegeben werden, z. B. Links auf anerkannte Institutionen (z. B. BSI), die zu kryptographischen Algorithmen regelmäßig Empfehlungen abgeben.
4	Der private Schlüssel wird bei OPC UA sowohl für die Signatur als auch für die Entschlüsselung verwendet.	Für Signatur und Verschlüsselung sollten verschiedene Schlüsselpaare in der OPC UA Spezifikation gefordert werden.
5	Derzeit besteht keine Möglichkeit, auf auf elliptische Kurven basierende Algorithmen zurückzugreifen.	Diese Möglichkeit sollte OPC UA in Zukunft vorsehen, um auch auf Geräten mit geringer Prozessorleistung ausreichende Sicherheit zu ermöglichen.

ID	Feststellung	Empfehlung
Kapitel 6, redaktionelle Analyse		
6	Inkonsistenzen zwischen Part 4 und 6: Sowohl Part 4, als auch Part 6, beschreibt insbesondere die Dienste, die für einen sicheren Verbindungsaufbau auf der Grundlage eines SecureChannels und einer Session, entscheidend sind. Allerdings ist Part 4 allgemein gehalten, wohingegen Part 6 auf die Besonderheiten der zur Auswahl stehenden Protokolle eingeht. Dadurch unterscheiden sich teilweise die Parameter, die in den verschiedenen Teilen einer Nachricht enthalten sind. Auch die Zuordnung mancher Datentypen, die zur Definition der Parameter dient, ist unterschiedlich. Dies ist verwirrend und fehlerträchtig.	Die angesprochenen Inkonsistenzen zwischen Part 4 und Part 6 sollten beseitigt werden.
7	Unklarheiten bei nicht gebrauchten Parametern: Durch die Möglichkeit, über den securityMode das Signieren und Verschlüsseln ein- oder auszuschalten, werden in bestimmten Fällen nicht benötigte Parameter in einer Nachricht mitübertragen. Es ist aber nicht eindeutig, welche Werte in solchen Fällen bei den betroffenen Parametern einzusetzen sind. Dies kann zu Implementierungsfehlern und Inkompatibilitäten zwischen OPC UA Applikationen von verschiedenen Herstellern führen.	Auch für nicht benötigte Parameter einer Nachricht sollten die zu setzenden Parameterwerte definiert werden.
8	Fehlende Liste sinnvoller Defaultwerte: Zur Konfiguration einer OPC UA Applikation gehört eine Reihe von Parametern, die Unter- oder Obergrenzen z. B. für Puffergrößen oder Anzahl der Nachrichten festlegen. Es fehlt allerdings meist die Angabe eines Defaultwerts oder eine Empfehlung, in welchem Bereich sinnvolle Werte liegen. Es würde Entwicklern und Administratoren die Arbeit deutlich erleichtern, wenn in der Spezifikation eine Liste von solchen Defaultwerten vorhanden wäre. Man könnte dabei zwei Geräteklassen unterscheiden: eingebettete Geräte mit begrenzter Leistung einerseits und PCs oder Workstations andererseits.	Die Defaultwerte von Parametern sollten immer festgelegt werden. Eine Liste von solchen Defaultwerten sollte erstellt werden und sollte zwei Geräteklassen unterscheiden: eingebettete Geräte mit begrenzter Leistung einerseits und PCs oder Workstations andererseits.
9	Besserer Schutz vor Angriffen auf geheimnisbasierten userIdentityToken: Zur Zeit schreibt die Spezifikation nicht vor, wie bei wiederholter fehlgeschlagener Benutzer-authentifizierung vorzugehen ist.	Ein Vorschlag zur weiteren Erhöhung der Sicherheit wäre, dass Server die Einstellung erlauben, bei jeder Fehlanmeldung den Zeitraum bis zur Bearbeitung der nächsten Benutzerauthentifizierung zu verdoppeln und z. B. bei einer Minute zu kappen. Bei erfolgreicher Anmeldung wird diese Grenze wieder auf eine Sekunde zurückgesetzt.

ID	Feststellung	Empfehlung
10	Derzeit wird folgende Einstellungskombination erlaubt: securityMode 'Sign' oder 'SignAndEncrypt' und securityPolicyUri 'None'. Dies führt zu einer unsicheren Kommunikation.	Es wird empfohlen, bei securityMode 'Sign' und 'SignAndEncrypt' als securityPolicyUri 'None' zu verbieten und diese Validierung der Konfiguration auch als Konformitätstest hinzuzufügen.
11	Part 4, S. 154, Kap. 7.35.1 Für geheimnisbasierte userIdentityTokens wird verlangt, dass sie verschlüsselt werden. Bei signaturbasierten userIdentityTokens steht nichts.	Es sollte zwingend erforderlich sein („shall“), dass immer entweder eine userTokenPolicy oder die securityPolicyUri nicht 'None' ist, damit zur Erstellung der Signatur ein Algorithmus zur Auswahl steht.
12	Part 6, S. 7, Kap. 5.1.6, Abschnitt 3 Laut Spezifikation können variant arrays ineinander verschachtelt werden. Das gleiche gilt für diagnosticInfo.	Es sollte eine sinnvolle maximal zulässige Rekursionstiefe festgelegt werden, z. B. 10.
13	Part 6, S. 39, Kap. 6.7.2 Was passiert, wenn eine Lücke bei der sequenceNumber entdeckt wird?	Es sollte explizit beschrieben werden, wie eine OPC UA Applikation auf einen solchen Fall reagieren soll (Fehlermeldung, Verbindungsabbruch, usw.).
14	Part 6, S. 41, Kap. 6.7.4 OpenSecureChannel Anfragen und Antworten werden auch bei securityMode 'Sign' verschlüsselt.	Dies sollte explizit erwähnt werden.
15	Part 6, S. 67, Kap. D6, Tabelle D6 Die Tabelle D6 gibt Optionen an, die eine unsichere Konfiguration der Kommunikation darstellen.	Es sollte explizit darauf hingewiesen werden, dass manche dieser Optionen unsicher sind.
Kapitel 7.1.5, Analyse Schutzziele und Bedrohungstypen		
16	Die Definitionen der Schutzziele unterscheiden sich vom international anerkannten Standard ISO/IEC 27000 [24] für die Schutzziele *´Authentication, Availability, Confidentiality´* und *´Integrity´*	Es sollten die Definitionen des Standards ISO/IEC 27000 verwendet werden
17	Das Schutzziel *´Nichtabstreitbarkeit´* fehlt	Die Schutzziele sollten um das Schutzziel *´Nichtabstreitbarkeit´*ergänzt werden.
18	Das Schutzziel *´Authorization´* ist ungenau definiert.	Die Definition sollte hervorheben, dass Rechte nach dem Need-to-Know Prinzip vergeben werden müssen.
19	Der Bedrohungstyp Abstreitbarkeit ist in der OPC UA Spezifikation nicht verwendet worden.	Erweiterung der Bedrohungstypen um die Abstreitbarkeit
20	Der Bedrohungstyp message flooding ist zu eng gefasst.	Erweiterung des Bedrohungstyps message flooding auf denial of service, da message flooding eine Untermenge von denial of service ist
21	Definitionen der Bedrohungstypen entsprechen nicht den üblichen Definitionen aus der IT Sicherheit oder sind zu unscharf	Die Definitionen sollten wie zum Beispiel in 7.1.3 vorgeschlagen genauer definiert werden und nicht nur auf die Kommunikation fokussieren (auch wenn sie dann nur auf die Kommunikation für OPC UA in der Spezifikation angewendet werden)

ID	Feststellung	Empfehlung
22	Die Zuordnung Schutzziel versus Bedrohungen ist sehr interpretierbar	Die Zuordnung Schutzziel versus Bedrohungen sollte überarbeitet werden.
Kapitel 7.4, Feststellungen bzgl. Design		
23	Mangelnde Flexibilität beim Profilkonzept: Die Spezifikation sieht aktuell vor, dass alle zulässigen Kombinationen von asymmetrischen und symmetrischen Signatur- und Verschlüsselungsalgorithmen samt Schlüssellängen in Part 7 als Profile aufgelistet werden. Allerdings erscheinen nur ca. alle drei Jahre neue Versionen des OPC UA Standards. Dies bedeutet, dass innerhalb eines solchen Zeitraums keine Anpassungen bezüglich der Kryptographie möglich sind: Es können weder neue, sicherere und/oder effizientere Algorithmen oder Schlüssellängen aufgenommen werden, noch können als unsicher geltende entfernt werden.	Es sollte überlegt werden, wie notwendige Anpassungen bzgl. Kryptographie zeitnah in die Spezifikation integriert werden können.
24	Fehlende forward secrecy: Das aktuelle Verfahren zur Erzeugung der symmetrischen Schlüssel unterstützt keine forward secrecy. Ist ein Angreifer in der Lage, die beim Aufbau eines SecureChannels ausgetauschten Client- und Servernonces herauszufinden, so kann er nachträglich Nachrichten entschlüsseln, die über diesen SecureChannel ausgetauscht wurden. Es ist offen, warum nicht auf Verfahren zurückgegriffen wurde, die sich seit Jahrzehnten bewährt haben und forward secrecy unterstützen, wie z. B. DHE-RSA oder DHE-DSS, die auf den Diffie-Hellman Schlüsselaustausch basieren.	Es sollten bewährte Verfahren für den Schlüsselaustausch verlangt werden.
25	Einführung von mDNS: In Part 12 Abschnitt 4.3.5 wird beschrieben, wie mDNS beim Discovery Dienst eingesetzt wird. Die Untersuchung der Sicherheit von mDNS ist außerhalb des Rahmens dieses Projekts, kann aber zusätzliche Risiken bergen.	Die IT Sicherheit von mDNS sollte untersucht werden.

Tabelle 21: Wesentliche Erkenntnisse der Gesamtanalyse

8 Ergebnisse aus der Analyse der Referenzimplementierung

In diesem Kapitel werden die Tests, Testergebnisse und, wo immer möglich, die Ausnutzung von identifizierten Schwachstellen beschrieben.

8.1 Beschreibung des getesteten OPC UA Kommunikationsstacks

Analysiert wurde die Referenzimplementierung des OPC UA Kommunikationsstacks der OPC Foundation in ANSI C in der Version 1.02.344.5. Es liegen mehrere Referenzimplementierungen der OPC Foundation in verschiedenen Programmiersprachen vor. Die Entscheidung für die ANSI C Implementierung wurde vom BSI getroffen. Die Version 1.02.344.5 war zu Beginn des Projekts die neueste vorliegende Version und entspricht auch der Version 1.02 der Spezifikation, die im Rahmen dieses Projekts analysiert wurde.

Es wurde nach Rücksprache mit dem BSI entschieden, den Stack in eine 32-Bit single-threaded UA Server Rahmenapplikation einzubinden und in einer Linux Umgebung zu testen. Die Einbindung in einen UA Server war notwendig, um die in Abschnitt 8.3 beschriebenen Tests durchführen zu können. Linux wurde wegen der zum Einsatz kommenden Werkzeuge und aus Lizenzgründen als Testplattform favorisiert. Die 32-Bit Variante wurde der 64-Bit Alternative bevorzugt, da beide Varianten aus IT Sicherheitssicht gleichwertig sind, 32-Bit Applikationen aber weiter verbreitet sind.

Der ANSI C Stack kann im single-threaded oder im multi-threaded Modus kompiliert und betrieben werden. In dieser Studie wurde der Prüfgegenstand im single-threaded Modus in einer single-threaded Rahmenapplikation betrieben, um die Komplexität zu reduzieren. Außerdem standen die Sicherheitsfunktionen und deren Logik sowie die En- und Decoder im Fokus der Analyse und nicht das Locking und die Parallelität der Rahmenapplikation.

8.2 Angriffe auf in der Spezifikationsanalyse identifizierten Schwachstellen

Es wurden keine systematischen Fehler bei der Spezifikationsanalyse (siehe Abschnitt 7.5) festgestellt. Daher werden hier keine Angriffe auf Schwachstellen/systematische Fehler beschrieben.

8.3 Vorgehensweise

Die Analyse der Referenzimplementierung basiert auf folgenden Tests des in Abschnitt 8.1 beschriebenen OPC UA Stacks:

- Zertifikatstests
- statische Codeanalyse
- Fuzzing
- dynamische Codeanalyse

Diese Tests dienen einerseits zur Prüfung, inwiefern die Sicherheitsmaßnahmen aus der Spezifikation im Stack vollständig und korrekt implementiert sind. Zum anderen wird die Codequalität mit der Hilfe von Werkzeugen untersucht.

Alle Tests, die eine laufende Umgebung voraussetzen, wurden gegen einen OPC UA Server gefahren. Die Clientseite des Kommunikationsstacks wurde, mit Ausnahme der statischen Codeanalyse, nicht untersucht.

Zertifikatstests:

Bei den Zertifikatstests wird geprüft, ob der Server bei unterschiedlichen PKI Konfigurationen so reagiert, wie es in der Spezifikation vorgesehen ist. Entscheidend ist dabei insbesondere die Einhaltung der in der OPC UA Spezifikation Part 4 Abschnitt 6.1.3 beschriebenen Schritte.

Bei diesen Tests werden Verbindungsversuche eines Clients an einen Server mit bestimmten, speziell konfigurierten PKI Umgebungen simuliert: Der Client versucht z. B., sich mit einem abgelaufenen ApplicationInstanceCertificate zu verbinden, eine CRL ist nicht verfügbar oder das root Zertifikat ist nicht vorhanden, usw.

Es werden sowohl Positivtests, die zu einem erfolgreichen Verbindungsaufbau führen sollten, als auch Negativtests, die zu einer Fehlermeldung des Servers führen sollten, durchlaufen.

Die einzelnen Tests werden in Anhang A detailliert angegeben.

Statische Codeanalyse:

Die statische Codeanalyse ermöglicht eine Untersuchung der Codequalität mittels des Werkzeugs *cppcheck*, mit dem der Quellcode analysiert werden kann. Zum Beispiel werden dadurch Parameterübergaben an Funktionen überprüft.

Der Quellcode des OPC Foundation Kommunikationsstacks enthält sowohl die Client- als auch die Serverseite der Kommunikation. Somit wird durch die statische Codeanalyse Client- und Servercode geprüft.

Fuzzing:

In diesem Projekt wurde durch das Fuzzing geprüft, wie der Server auf unterschiedlichste meist ungültige Nachrichten reagiert. Hierfür wird eine große Anzahl von Testiterationen mit variablen Nachrichteninhalten automatisiert ausgeführt. Zur Implementierung dieser Tests wurde das Fuzzingframework *Peach* verwendet.

Im Fuzzingframework wurden Tests für die verschiedenen securityModes des OPC Foundation Stacks None, Sign und SignAndEncrypt implementiert. Für die securityModes Sign und SignAndEncrypt wurde eine externe Applikation namens *secucon* entwickelt, die es ermöglicht, in *Peach* erzeugte Nachrichten zu signieren und zu verschlüsseln.

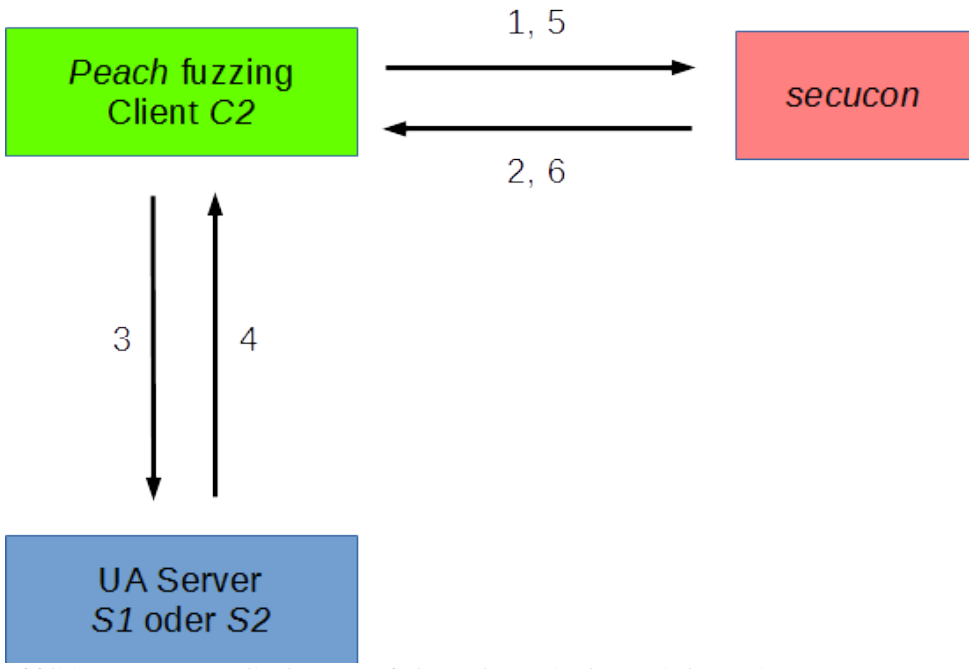

Abbildung 3: Kommunikationswege bei securityMode Sign und SignAndEncrypt

Erläuterung der Pfeile in Abbildung 3:

1: *Peach* schickt *secucon* eine gefuzzte Nachricht

2: *secucon* schickt diese Nachricht signiert und bei Bedarf verschlüsselt an *Peach* zurück

3: *Peach* leitet die Nachricht an den Server weiter

4: der Server antwortet mit einer signierten und bei Bedarf verschlüsselten Nachricht

5: *Peach* leitet die Antwort vom Server an *secucon* weiter

6: *secucon* schickt *Peach* die Antwort vom Server, entschlüsselt und ohne Signatur

Dynamische Codeanalyse:

Die dynamische Codeanalyse wird, ähnlich wie die statische Codeanalyse, zur Identifizierung von Fehlern bestimmter Programmierfehlerklassen benutzt. Während bei der statischen Codeanalyse die zu testende Software nicht ausgeführt wird, setzt die dynamische Codeanalyse die Ausführung des Codes des OPC Foundation Kommunikationsstacks voraus. Um möglichst viele Fehler im Code aufdecken zu können, wurde bei der dynamischen Codeanalyse eine möglichst hohe Codeabdeckung angestrebt.

Details zur Codeabdeckung sind in Abschnitt 8.8 beschrieben.

8.4 Zertifikatstests

8.4.1 Beschreibung der Tests

Bei den Zertifikatstests geht es darum, die in der Spezifikation beschriebene PKI Funktionalität zu verifizieren. Dabei sind insbesondere Part 4 Abschnitt 6.1.3 der OPC UA Spezifikation und die dort abgebildete Tabelle 101 von Bedeutung: Diese Tabelle beschreibt, welche Schritte zur Prüfung der Gültigkeit und Vertrauenswürdigkeit eines Zertifikats erforderlich sind und in welcher Reihenfolge sie durchzuführen sind. In folgender Tabelle sind die zehn Validierungsschritte und ihre Beschreibung, wie sie in Part 4 Tabelle 101 zu finden sind, als Referenz aufgelistet. Die rechte Spalte gibt darüber Auskunft, ob Fehler beim jeweiligen Validierungsschritt unterdrückt werden können. Wird eine solche Unterdrückung z. B. bei der *validity period* eingeschaltet, gilt die Validierung eines Zertifikats als erfolgreich, auch wenn sie diesem Schritt einen Fehler auslöst.

Validierungsschritt	Beschreibung	Fehler unterdrückbar?
Certificate Structure	The certificate structure is verified.	nein
Validity Period	The current time shall be after the start of the validity period and before the end.	ja
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.	ja
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.	nein
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).	ja
Trust List Check	No further checks are required if the Certificate is in the	--

Validierungsschritt	Beschreibung	Fehler unterdrückbar?
	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 reported if the Administrator disables revocation checks for a CA Certificate.	ja
Revocation Check	The Certificate has been revoked and may not be used.	nein

Tabelle 22: Schritte zur Validierung von Zertifikaten (Quelle: [5] Tabelle 101)

Anhand dieser Tabelle lassen sich verschiedene Parameter ableiten, die eine Rolle bei der Validierung von Zertifikaten eine Rolle spielen:

- Struktur der PKI: Handelt es sich um selbstsignierte Zertifikate, um eine einfache PKI mit einer CA oder um eine mehrstufige PKI mit einer root CA und sub CAs?
- Gültigkeit einzelner Zertifikate: Sind sie gültig, abgelaufen, noch nicht gültig, mit fehlerhafter Signatur versehen und/oder widerrufen? Sieht die Konfiguration vor, dass die zeitliche Gültigkeit (abgelaufen, noch nicht gültig) ignoriert wird?
- Verifizierung der Zertifikatskette: Sind die Glieder der Zertifikatskette im OPC UA PKI store vorhanden oder werden sie in der Nachricht mitgeschickt? Ist die Zertifikatskette vollständig? Wird dem selbstsignierten oder dem CA Zertifikat vertraut?
- CRLs: Kommen CRLs zum Einsatz? Falls ja, wie wird mit fehlenden CRLs umgegangen?

In Tabelle 34 in Anhang A sind die 92 durchgeführten Zertifikatstests zu finden, bei denen verschiedene Kombinationen der hier erwähnten Parameter getestet werden.

Zur Durchführung der Tests wurde das Werkzeug *certcheck* eingesetzt, das leihweise von Unified Automation zur Verfügung gestellt wurde. Damit kann getestet werden, wie ein Server auf unterschiedlich konfigurierte Clientanfragen reagiert.

In diesem Projekt wurde als OPC UA PKI store unter Linux eine Ordnerstruktur im Dateisystem verwendet.

8.4.2 Analyse der Ergebnisse

Die Testergebnisse lassen sich wie folgt zusammenfassen:

- 53 Tests waren erfolgreich: Der Server hat die erwartete Antwort geschickt.
- 39 Tests sind fehlgeschlagen. Hier kann man die Tests in drei Kategorien unterteilen:

  1. Der StatusCode in der Antwort des Servers weicht vom erwarteten ab (12 Tests). Es handelt sich also um eine Abweichung der Implementierung des Stacks von der Spezifikation. Diese Abweichung hat aber aus IT Sicherheitsgesichtspunkten keine große Relevanz und wird deshalb nicht weiter untersucht.
  2. Der Server lehnt fälschlicherweise das Zertifikat des Clients und somit auch einen legitimen Verbindungsaufbau ab (8 Tests). Die Verfügbarkeit der Verbindung ist in diesen Fällen also beeinträchtigt.
  3. Der Server stuft das Client Zertifikat fälschlicherweise als gültig ein und lässt die Verbindung vom Client zu (19 Tests). Hier wird also die Sicherheit des Authentifizierungsmechanismus umgangen.

In folgender Tabelle sind alle fehlgeschlagenen Tests aufgelistet und nach den eben beschriebenen Kategorien sortiert:

Testnr.	Testname	Erwarteter StatusCode	Tatsächlicher StatusCode
Falscher StatusCode			
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
Verbindung verweigert			
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_	Good	BadCertificateTime-Invalid

Testnr.	Testname	Erwarteter StatusCode	Tatsächlicher StatusCode
	CaIssuersCrl_ScaIssuersExpiredCrl_ IgnInvTime_Good		
Verbindung zugelassen			
15	RootSigned3Trusted_CaIssuers_Bad	BadCertificate-Revocation Unknown	Good
16	RootSigned4TrustedRevoked_ CaIssuersCrl_Bad	BadCertificateRevoked	Good
20	RootSigned8_CaTrusted_Bad	BadCertificate-Revocation Unknown	Good
21	RootSigned9Revoked_ CaTrustedCrl_Bad	BadCertificateRevoked	Good
24	RootSigned12Trusted_CaTrusted_ Bad	BadCertificate-Revocation Unknown	Good
25	RootSigned13TrustedRevoked_ CaTrustedCrl_Bad	BadCertificateRevoked	Good
27	RootSigned16Trusted_CaChain_Bad	BadCertificate-Revocation Unknown	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-Revocation Unknown	Good
61	SecondarySigned13_CaIssuersCrl_ ScaTrusted_Bad	BadCertificate-Revocation Unknown	Good
62	SecondarySigned14_CaTrustedCrl_ ScaIssuers_Bad	BadCertificate-Revocation Unknown	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_Ca IssuersCrl_ScaIssuersCrl_Bad	BadCertificateRevoked	Good
76	SecondarySigned28TrustedRevoked_Ca IssuersCrl_ScaTrustedCrl_Bad	BadCertificateRevoked	Good
77	SecondarySigned29TrustedRevoked_Ca	BadCertificateRevoked	Good

Testnr.	Testname	Erwarteter StatusCode	Tatsächlicher StatusCode
	TrustedCrl_ScaIssuersCrl_Bad		

Tabelle 23: Fehlgeschlagene Zertifikatstests

Detailliertere Analyse der Ergebnisse:

Bei allen fehlgeschlagenen Tests der Kategorie 2, bei denen eine Verbindung fälschlicherweise unterbunden wurde, ist die Ursache immer die gleiche: Es wird versucht, mit noch nicht gültigen oder bereits abgelaufenen Zertifikaten eine Verbindung aufzubauen. Dabei ist zu beachten, dass bei all diesen Tests die Option 'IgnoreInvalidTime' gesetzt ist. Dies sollte dazu führen, dass die Zertifikate als gültig anerkannt werden, obwohl die 'validity period' nicht gültig ist, und somit die Verbindung zugelassen wird. Die getestete UA Serverrahmenapplikation unterstützt die Unterdrückung von Fehlern bei abgelaufenen und noch nicht gültigen Zertifikaten nicht. Deshalb lehnt die Serverapplikation den Verbindungsaufbau immer mit dem Status-Code 'BadCertificateTimeInvalid' (`0x80140000`): „The Certificate has expired or is not yet valid.“ ab.

Bei der Option 'IgnoreInvalidTime' handelt es sich um eine Funktionalität, die außerhalb des Stacks implementiert wurde. Entsprechend spielen diese fehlgeschlagenen Tests für die IT Sicherheit des Kommunikationsstacks keine Rolle.

Auch bei den Tests der dritten Kategorie, bei denen eine Verbindung fälschlicherweise zugelassen wird, stellte sich heraus, dass das Fehlverhalten auf eine Ursache zurückzuführen ist. Folgende StatusCodes werden bei allen fehlgeschlagenen Tests erwartet:

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

Es geht also immer darum, die Funktionalität der CRL Prüfung zu testen: Entweder sollte erkannt werden, dass die CRL fehlt, oder dass das Zertifikat in der vorhandenen CRL als widerrufen markiert ist. Diese Tests schlagen alle fehl, weil die Funktionalität der CRL Prüfung in der getesteten Rahmenapplikation unvollständig implementiert ist. Entsprechend ist keine Aussage über die Implementierung der CRL Funktionalität im Stack mit den in Tabelle 34 beschriebenen Tests möglich.

Zusammenfassung:

Die Analyse der Zertifikatstests hat ergeben, dass keine für die IT Sicherheit relevanten Fehler im Kommunikationsstack der OPC Foundation in der getesteten Version 1.02.344.5 festgestellt werden konnte. Es wurden lediglich bei zwölf Tests Abweichungen bei den StatusCodes beobachtet.

Diese Ergebnisse spiegeln allerdings nur teilweise den tatsächlichen Stand der Zertifikatsvalidierung im Stack wider. Die Implementierung der Prüfung von CRLs im Stack konnte nicht getestet werden.

8.5 Statische Codeanalyse

Bei der statischen Codeanalyse wurde das open source Analysewerkzeuge *cppcheck* verwendet. Der Aufruf lautete:

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

Die Parameter mit `-std=` schalten zusätzliche Prüfungen für den jeweiligen Standard ein.

In der *cppcheck* Ausgabe waren ca. 340 Fehlermeldungen enthalten, wovon die meisten jedoch für die IT Sicherheit keine Rolle spielen. Folgende vier Fehlermeldungstypen wurden nicht weiter ausgewertet:

1. **(style) The scope of the variable '<variable_name>' can be reduced.**
   Diese Meldung kam 45 Mal vor. Die Tatsache, dass der Gültigkeitsbereich einer Variablen hätte kleiner gewählt werden können, hat keine Auswirkung auf die IT Sicherheit.
2. **(style) The function '<function_name>' is never used**
   Diese Meldung kam 267 Mal vor. Dies deutet auf toten oder noch nicht benutzen Quellcode hin. Die hohe Anzahl an Meldungen kann unter Umständen ein Hinweis für mangelnde Pflege des Code sein, hat aber keine direkte Relevanz für die IT Sicherheit.
3. **(style) Variable '<variable_name>' is assigned a value that is never used**
   Diese Meldung kam sechs Mal vor. Auch hier handelt es sich um einen Hinweis, dass wahrscheinlich toter Quellcode vorliegt, ist aber ohne direkten Einfluss auf die IT Sicherheit.
4. Elf Meldungen bezogen sich auf Quellcode aus dem Linux Platform Layer `platforms/linux/*`, der kein Bestandteil des OPC Foundation Stacks ist, und wurden deshalb ignoriert.

Die restlichen Meldungen wurden im Folgenden genauer analysiert und in Gruppen zusammengefasst und bewertet:

1 **(error) Possible null pointer dereference**
Diese Meldung tauchte dreimal auf:

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`
Dies ist kein Fehler: `pFile` wird im Code in Zeile 401 oder 404 schon geprüft und kann nicht NULL sein.

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`
Dies ist kein Fehler: Der Wert, der `pCertificateStoreCfg` zugewiesen wird, wird in Zeile 96 des Codes geprüft.

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`
Dies ist kein Fehler: `a_ppConnection` wird in Zeile 1767 des Codes geprüft.

2 **(warning) Using size of pointer <pointer_name> instead of size of its data.**
Diese Meldung tauchte einmal auf:

```
[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)
```

Dies ist kein Fehler: Es handelt sich hier um eine Struktur von Zeigern. Entsprechend stimmt die Berechnung der Größe der Struktur hier.

3 **(warning) Redundant assignment of "<variable_name>" in switch**
Diese Meldung tauchte einmal auf:

```
[transport/https/opcua_httpsstream.c:2989]: (warning) Redundant assi-
gnment of "bParseAgain" in switch
```

Die zweite Zuweisung ist redundant. Dies ist unnötig, stellt aber kein Problem dar.

4 **(style) Defensive programming**
Diese Meldung tauchte einmal auf:

```
[platforms/win32/opcua_p_utilities.c:391]: (style) Defensive program-
ming: 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.
```

Dies ist kein Fehler: Der String Terminator wird vorher in Zeile 390 im Code geprüft.

5 ***(style) Finding the same code for an if branch and an else branch is suspicious***
Diese Meldung tauchte zweimal auf:

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.
```

Dies ist kein Fehler: Diese Stelle wurde für debugging Zwecke verwendet.

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.
```

Dies ist kein Fehler: Es liegt an den Makros, dass für *cppcheck* beide Zweige gleich aussehen.

6 ***(style) Found obsolete function***
Diese Meldung tauchte einmal auf:

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

Die Nutzung der alten Funktion wird durch ein Makro gesteuert; falls die alte Funktion verwendet wird, kann es bei multi-threaded Applikationen zu Problemen kommen, weil unter Umständen mehrere Threads gleichzeitig auf eine globale Struktur zugreifen.
Die Folgen aus IT Sicherheitssicht hängen von der Implementierung der Funktion `gethostbyname` auf dem jeweiligen Betriebssystem ab und können deshalb nicht klar eingegrenzt werden. Abstürze werden aber als sehr unwahrscheinlich eingestuft.

7 ***(portability) Found non reentrant function <function_name>***

Diese Meldung tauchte einmal auf:

```
[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'
```

Dies ist ein Fehler: Siehe Punkt Fehler: Referenz nicht gefunden.

Zusammenfassung:

Lediglich die zwei zusammenhängenden Fehlermeldungen Fehler: Referenz nicht gefunden und Fehler: Referenz nicht gefunden stellen ein tatsächliches Problem dar und können eventuell bei einem multi-threaded Serverbetrieb Auswirkungen auf die IT Sicherheit haben.

8.6 Fuzzing

8.6.1 Programmierung in Peach

Aufbau einer pit Datei

Zentraler Bestandteil des Testens mit *Peach* ist die sogenannte pit Datei, die vorgibt, wie das Fuzzingframework zu arbeiten hat. Wesentliche Elemente in der Gliederung einer pit Datei sind:

- das Datenmodell: Hier wird spezifiziert, wie das zu testende Protokoll strukturiert ist. Dabei werden die Nachrichten, die das Protokoll ausmachen, z. B. in header und body dargestellt und diese bestehen wiederum aus komplexen Datenstrukturen, die sich anhand von Grunddatentypen wie String und Number modellieren lassen.
- das Zustandsmodell: Sind die Strukturen der einzelnen Nachrichten definiert, ist außerdem wichtig zu spezifizieren, welche Nachrichten in welcher Reihenfolge erwartet werden und wie die Zusammenhänge zwischen den Nachrichten sind, insbesondere zwischen Anfragen und Antworten. Dies wird in *Peach* in einem Zustandsautomaten modelliert.
- die Spezifikation des Testlaufs: Hier wird u.a. festgelegt, welche Parameter einer Nachricht beim Fuzzen verändert werden dürfen und nach welchen Regeln eine solche Veränderung stattfindet.

Einschränkung beim Datenmodell

Peach unterstützt keine rekursiven Datentypen oder -strukturen. Entsprechend wurde auf das Prüfen von möglichen Rekursionstiefen bei den OPC UA Datentypen **Variant** und **DiagnosticInfo** im Rahmen der Fuzzingtests verzichtet. Das Verwenden von Rekursionen kann z. B. zu Applikationsabstürzen führen. Diese Art von Fehlern konnte also nicht in Peach reproduziert werden.

Fuzzingstrategien

Peach bietet für alle Grunddatentypen, die vom Framework unterstützt werden, sogenannte Mutators, die definieren, wie ein Datentyp beim Fuzzing verändert wird. Wie die Parameter einer Nachricht gefuzzt werden, hängt von der Wahl der Fuzzingstrategie ab. Es stehen folgende zwei zur Auswahl:

- sequential: Hier wird jeder Parameter einer Nachricht, einzeln und der Reihenfolge nach, mit allen für diesen Datentyp zutreffenden Mutators und allen definierten Werten für den jeweiligen Mutator gefuzzt. Daraus ergibt sich eine bestimmte Anzahl an Iterationen für den Testlauf, nach der alle zulässigen Kombinationen aus Parametern, Mutators und Werten getestet wurden.
- random: Bei random sind grundsätzlich die gleichen Kombinationen aus Parametern, Mutators und Werten wie bei sequential möglich. Der Unterschied ist, dass bei jeder Iteration zufällig eine neue Kombination für das Triple (Parameter, Mutator, Wert) gewählt wird. Die Anzahl der gewünschten Iterationen muss man selbst vorgeben, weil Peach sonst endlos läuft. Zur Begründung für die bei den durchgeführten Fuzzingtests gewählten Anzahl an Iterationen, siehe Abschnitt 8.8.2.
  Als zusätzliche Einstellung hat man bei random die Möglichkeit, statt wie bei sequential immer nur einen Parameter pro Iteration zu fuzzen, je Iteration mehrere Parameter gleichzeitig zu verändern.

Zusammenfassend stehen also drei Arten von Tests der Fuzzingstrategien zur Verfügung: sequential, random mit einem veränderbaren Parameter und random mit mehreren gleichzeitig veränderbaren Parametern.

Grundsätzlich ist die Fuzzingstrategie sequential gegenüber random mit einem veränderbaren Parameter zu bevorzugen: Es werden alle möglichen Triple (Parameter, Mutator, Wert) exakt ein Mal durchlaufen. Bei random kann es vorkommen, dass manche Triple gar nicht getestet werden, andere wiederum mehrfach.

Einschränkung durch choices

Die Begründung für die Verwendung der drei zur Verfügung stehenden Varianten an Fuzzingstrategien ist folgende: Bei der Modellierung von komplexen Datentypen in *Peach* ist es sinnvoll, mit sogenannten 'choice'

Strukturen zu arbeiten. Diese ermöglichen es, die Inhalte von Datenstrukturen in Abhängigkeit eines spezifizierten Parameters flexibel zu gestalten.

Ein anschauliches Beispiel für die Notwendigkeit von choices sind UA Strings. Sie bestehen aus zwei Grunddatentypen: einer Zahl zur Längenangabe der Zeichenkette und der Zeichenkette selbst. Diese Struktur gilt allerdings nicht für NULL Strings, die nur aus der Längenangabe mit dem Wert -1 bestehen und keine Zeichenkette enthalten. Um diese Fallunterscheidung zu modellieren, greift man auf das Hilfsmittel 'choice' zurück, das abhängig von der Längenangabe (> -1 oder = -1) der jeweiligen Instanz eines UA Strings eine Zeichenkette hinzufügt oder nicht.

Das Problem ist, dass *Peach* bei der Fuzzingstrategie sequential beim Fuzzen eines komplexen Datentyps mit choice immer nur die an erster Stelle definierte Variante des choices auswählt und nie die anderen Abzweigungen durchläuft. Bei einem Datentyp mit fünf Varianten innerhalb eines choices wird also nur eine getestet, wenn man ausschließlich mit der Fuzzingstrategie sequential arbeitet.

Diese Einschränkung kann bei der Fuzzingstrategie random überwunden werden, indem man *Peach* mittels sogenannten Datensets dazu zwingt, eine andere als die erste verfügbare Variante bei der Instanziierung eines komplexen Datentyps zu wählen.

Zusammenfassend kann man also festhalten:

- Mit sequential hat man eine optimale Nutzung jeder Iteration, weil keine Wiederholungen vorkommen. Allerdings wird man bei komplexen Datentypen nur eine Variante testen können.
- Mit random werden bestimmte Triple (Parameter, Mutator, Wert) mehrfach, andere im Rahmen eines endlichen Testlaufs gar nicht getestet. Man kann jedoch erzwingen, dass *Peach* alle definierten Varianten von komplexen Datentypen testet.

Modellierte Nachrichten

Der Grundgedanke beim Aufbau der Tests war, aus Client Sicht die für die IT Sicherheit von OPC UA entscheidenden Dienste SecureChannel, Session und Discovery zu testen. Deshalb wurden in *Peach* alle Nachrichten simuliert, die Client und Server zum Verbindungsaufbau austauschen:

- auf UA TCP Ebene:
  - Hello Anfrage (Name 'HEL' in Tabelle 35)
  - Acknowledge und Error Antwort
- auf UASC Ebene:
  - SecureChannel Dienst:
    - OpenSecureChannel Anfrage und Antwort (Name 'OPN' in Tabelle 35)
    - CloseSecureChannel Anfrage und Antwort
  - Session Dienst:
    - CreateSession Anfrage und Antwort
    - ActivateSession Anfrage und Antwort
    - CloseSession Anfrage und Antwort
  - Discovery Dienst:
    - FindServers Anfrage und Antwort
    - GetEndpoints Anfrage und Antwort

Als Ausgangspunkt für das Fuzzen wurden Nachrichten mit gültigen Inhalten verwendet, bei denen dann ein oder mehrere Parameter verändert wurden.

Abhängigkeit vom securityMode

Zusätzlich zu den zur Auswahl stehenden Nachrichten ist ein wesentlicher Aspekt für das Fuzzen der OPC UA Kommunikation, mit welchem securityMode die Verbindung aufgebaut wird: Wie in Abschnitt 8.3 beschrieben, simuliert *Peach* einen Client und schickt bei securityMode None seine Nachrichten direkt an den zu testenden Server. Bei securityMode SignAndEncrypt hingegen müssen die *Peach* Nachrichten zum Signieren und Verschlüsseln erst an *secucon* geschickt werden.

Es wurde entschieden, die Grundfunktionalität der OPC UA Kommunikation zwischen Client und Server im securityMode None ausgiebig zu testen. Ergänzend dazu wurden einige dieser Tests im securityMode SignAndEncrypt wiederholt, um auch die Funktionalität des Signierens und Verschlüsselns zu testen.
Es wurde davon ausgegangen, dass die Funktionalität, die bei securityMode Sign zum Einsatz kommt, auch bei SignAndEncrypt verwendet wird. Siehe Abschnitt 8.8.2 für die Verifizierung dieser Annahme.

8.6.2 Auswahl der Fuzzingtests

Bei der Auswahl der Fuzzingtests wurden vier unterschiedliche Ziele verfolgt:

- dynamische Codeanalyse: Einige der Tests wurden zur Gewinnung von Daten für die dynamische Codeanalyse durchgeführt. Diese erfordert, dass ein mit debug Informationen kompilierter Server getestet wird. Außerdem wurde bei diesen Tests die Codeabdeckung mit *lcov* gemessen. Hierfür wurde der Quellcode mit zusätzlichen Compilerflags (siehe Abschnitt 8.8) kompiliert.
- Produktionsähnlicher Server: Das Laufzeitverhalten von Testapplikationen kann sich je nach Compiler-Einstellungen, wie z. B. mit oder ohne debug Informationen, unterscheiden. Deshalb wurden bestimmte Tests mit einer Serverapplikation wiederholt, die mit Einstellungen kompiliert wurde, wie man sie in Produktionsumgebungen vorfinden würde.
- Erhöhung der Codeabdeckung: Für manche Tests wurden die Anzahl der Iterationen deutlich erhöht, um eine möglichst hohe Codeabdeckung zu erreichen. Diese Tests hatten eine Laufzeit von mehreren Wochen.
- Verifizierung der Annahmen bezüglich der Codeabdeckung: Zur Prüfung von in Abschnitt 8.6.1 getroffenen Annahmen der Codeabdeckung wurden zusätzliche Tests durchgeführt. Damit konnte nachgewiesen werden, dass die Auswahl der Kriterien bei den Fuzzingtests keinen negativen Einfluss auf die Codeabdeckung hatte.

Im Anhang A sind in Tabelle 35 alle durchgeführten Fuzzingtests aufgelistet: Es wurden Tests mit den drei in Abschnitt 8.6.1 beschriebenen Fuzzingstrategien, mit den drei securityModes und mit den debug und release Versionen des Servers durchgeführt.

Die Testserie 009 zur Erhöhung der Codeabdeckung hatte eine Laufzeit von ca. 39 Tagen. Auf Grund von technischen Problemen wurden Test 03 nach wenigen Stunden und Test 02 nach ca. elf Tagen abgebrochen. Die Tests 04 und 07 liefen ohne Unterbrechung über den gesamten Zeitraum. Somit konnten folgende Anzahlen an Iterationen erreicht werden:

- Test 02: ca. 1,92 Millionen
- Test 04: ca. 1,56 Millionen
- Test 07: ca. 7,56 Millionen

8.6.3 Ergebnisse

Bei der Modellierung identifizierte Fehler

Bei der Modellierung der Datenstrukturen und der Entwicklung der Tests mit dem Fuzzingframework *Peach* wurden sowohl Abweichungen der Implementierung von der Spezifikation, als auch Inkonsistenzen bei der Spezifikation festgestellt. Bei folgenden Listen hat keine Gewichtung stattgefunden.

Abweichungen:

1. Laut Part 6 Tabelle 13 sollte ein ExtensionObject für message chunks nach der NodeId auch ein 'Encoding' Byte enthalten. Die Nachrichten vom Server sind aber ohne 'Encoding' Byte. Nach der NodeId kommt sofort der authenticationToken.
2. Die Stackimplementierung erwartet bei securityMode None für die Parameter SecurityPolicyUri-Length, SenderCertificate und ReceiverCertificateThumbprint NULL strings, also der Länge -1.

3. In Part 6 Tabelle 30 der Spezifikation sind diese Parameter nicht als ByteString oder UA String definiert, obwohl sie so aufgebaut sind.
4. Mit UA binary codierte SecureChannel und Session Anfragen mit einem xml encoding TypeId identifier werden trotzdem erfolgreich decodiert.
5. Der Server akzeptiert Werte kleiner als 8.192 für die Parameter ReceiveBufferSize und SendBufferSize.
6. In der Spezifikation Part 6 S. 41 steht, dass bei securityMode None der nonce NULL sein sollte. Der Server antwortet aber mit einem nonce der Länge 1 mit Wert 1.
7. Laut Part 4 Tabelle 9 der Spezifikation sollte in der CloseSecureChannel Anfrage die secureChannelId zweimal vorkommen: als UInt32 und als ByteString. Diese Redundanz ist in der Implementierung aber nicht vorgesehen.
8. Der Server akzeptiert Nachrichten mit falscher sequenceNumber. Laut Part 6 S. 39 sollte dies ein transport error auslösen.
9. Der Server akzeptiert OpenSecureChannel Anfragen mit Protokollversionen, die von der in der Hello Nachricht geschickten Version abweichen. Dies entspricht nicht der Spezifikation (siehe Part 6 S. 41).
10. Laut Part 4 Tabelle 13 der Spezifikation muss bei der ActivateSession Anfrage ein userIdentityToken enthalten sein. Obwohl der Server in den EndpointDescriptions der CreateSession Antwort meldet, dass nur drei userIdentityToken Typen (Anonymous, UserName, Certificate) unterstützt werden, wird die ActivateSession Anfrage ohne userIdentityToken akzeptiert.
    Das Problem ist bekannt und wurde in der neuen Spezifikation bereits behoben, indem hinzugefügt wurde, dass NULL oder leere userIdentityTokens als Anonymous userIdentityToken interpretiert werden sollen.
11. In Part 6 Tabelle 24 der Spezifikation wird der Parameter SignedSoftwareCertificate als ByteString definiert. In der Implementierung hingegen sind es zwei ByteStrings, nämlich CertificateData und Signature.
12. Laut Part 4 Abschnitt 7.30 der 1.02 Spezifikation besteht SignatureData aus den Elementen
    signature
    algorithm
    In der Implementierung ist die Reihenfolge umgekehrt.
13. Die Reihenfolge in der 1.03 Spezifikation wurde angepasst, um mit der Implementierung übereinzustimmen.

Inkonsistenzen:

1. Die Definitionen für den Datentyp ChannelSecurityToken in Part 4 Tabelle 7 und Part 6 Tabelle 35 sind inkonsistent.
2. Es wird grundsätzlich empfohlen zu prüfen, ob beide Datentypen UtcTime und DateTime benötigt werden, da sie redundant zu sein scheinen.
3. In Part 3 S. 65-66 ist localizedText als Datentyp bestehend aus zwei strings definiert. In Part 4 Tabelle 123 wird localizedText als Name eines Int32 Parameters verwendet. Es wird empfohlen, den Parameternamen auf localizedTextIndex zu ändern, um möglichen Verwechselungen vorzubeugen.
4. Beim Parameter LocalizedText sind in Part 3 Tabelle 22 und Part 6 Tabelle 12 unterschiedliche Reihenfolgen für Text und Locale definiert.

    Anmerkung: In der 1.02 Spezifikation ist der Aufruf von CloseSession ohne ActivateSession nicht vorgesehen. In der 1.03 Spezifikation werden einem Client mehr Möglichkeiten eingeräumt, indem auf einem CreateSession nun auch ein CloseSession folgen kann.

Die meisten hier festgestellten Abweichungen und Inkonsistenzen spielen für die IT Sicherheit des OPC UA Protokolls keine Rolle. Ausnahmen sind die Abweichungen 7 und 8, die beide Mechanismen betreffen, die zur Sicherheit und Stabilität des Protokolls beitragen sollten. Gerade die sequenceNumber spielt eine wichtige Rolle bei der Abwehr von replay Angriffen. Die fehlende Prüfung der Versionsnummer könnte die Stabilität von OPC UA Applikationen beeinträchtigen, falls Clients und Server mit unterschiedlichen Protokollversionen versuchen, miteinander zu kommunizieren.

Bei den Tests gefundene Fehler

Bei den allermeisten durchgeführten Fuzzingtests wurden Iterationen in der Größenordnung von 100.000 bis 200.000 durchlaufen. Somit summiert sich die Anzahl der Iterationen je Testserie auf ca. 1 – 1,5 Millionen. Zusammen mit der auf mehrere Wochen angelegten Testserie 009 wurden insgesamt bei den Fuzzingtests deutlich über 15 Millionen Iterationen durchlaufen.

Abgesehen von den Daten für die dynamische Codeanalyse, hat die reine Ausführung der Fuzzingtests zu keinem Erkenntnisgewinn bezüglich Implementierungs- oder systematischen Fehlern geführt. Es fanden insbesondere keine durch Fehler im Stack verursachten Abstürze statt.

Zusammenfassung

Die Fuzzingtests der Client Server Nachrichten der drei Dienste SecureChannel, Session und Discovery haben gezeigt, dass die Stackimplementierung auch bei unerwarteten Inhalten stabil läuft.

Während der Modellierungsphase konnte jedoch festgestellt werden, dass manche in der Spezifikation für die IT Sicherheit und Stabilität vorgesehenen Mechanismen nicht, unvollständig oder fehlerhaft implementiert wurden. Insbesondere die vollständige und korrekte Auswertung der sequenceNumber muss dringend nachgeholt werden.

Als Ausweitung der aktuellen Analyse könnte eine formale und systematische Modellierung des Protokolls mit einem Beweiser erwägt werden.

8.7 Dynamische Codeanalyse

Die dynamische Codeanalyse wurde mit dem open source Werkzeug *Valgrind* durchgeführt. Für diese Zwecke wurde der UA Server mit Debuginformationen und Flags zur Messung der Codeabdeckung kompiliert. Diese Anwendung wurde dann mit *Valgrind* gestartet. Folgende Fuzzingtests wurden mit *Valgrind* ausgeführt und vollständig ausgewertet (siehe Tabelle 35 für weitere Details zu den einzelnen Testserien):

Testseriennr.	Testserienname	securityMode
001	*sequential individual parameter fuzzing*	None
002	*random individual parameter fuzzing*	None
003	*random multiple parameter fuzzing*	None
007	*sequential individual parameter fuzzing*	SignAndEncrypt

Tabelle 24: Liste der bei der dynamischen Codeanalyse vollständig ausgewerteten Fuzzingtests

Folgende Testserien wurden auch mit Valgrind aufgeführt, allerdings wurde hier ausschließlich Serverabstürze betrachtet:

Testseriennr.	Testserienname	securityMode
009	*random multiple parameter fuzzing*	SignAndEncrypt
015	*sequential individual parameter fuzzing*	Sign
016	*random multiple parameter fuzzing*	None

Tabelle 25: Liste der bei der dynamischen Codeanalyse teilweise ausgewerteten Fuzzingtests

8.7.1 Ergebnisse:

Insgesamt kamen über 400 Fehlermeldungen von *Valgrind* zusammen. Die Auswertung ergab, dass viele Meldungen auf die gleichen Ursachen zurückzuführen waren und dass die Implementierungsfehler bei einigen dieser identifizierten Ursachen außerhalb des Untersuchungsgegenstands lagen, also nicht den Kommunikationsstack betrafen.

Die Fehler im Stack lassen sich folgenden vier Fehlerklassen zuordnen:

Speicherleck bei NodeId Dekodierung

Klasse: Speicherleck mit beeinflussbarer Größe
Funktion: `OpcUa_BinaryDecoder_ReadNodeIdBody()`
Referenzen in Valgrind logs:

Valgrind log	Zeilen
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

Valgrind log	Zeilen
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

Tabelle 26: Auftreten des Speicherlecks in den Valgrind logs

Technische Detailbeschreibung:

Der Typ der `NodeId` wird erst nach dem Deserialisieren gesetzt. Tritt beim Deserialisieren ein Fehler auf (vom Sender ausgelöst), kann der allozierte Speicher nicht mehr freigegeben werden. Bei `NodeIds` vom Typ *String* und *ByteString* ist die Größe des allozierten Speichers vom Sender vorgegeben.

Auswirkung:

Ein Angreifer hat die Möglichkeit, durch dauerhafte Belegung des Arbeitsspeichers den Server in einen Zustand zu bringen, in dem er nur noch kleine oder gar keine Anfragen mehr beantworten kann. Sollte das Fehlerhandling bei fehlgeschlagenen Allokationen teilweise fehlerhaft sein, wären Folgefehler denkbar.

Das am leichtesten erreichbare Ziel für einen Angreifer wäre somit, den Server in seiner Erreichbarkeit einzuschränken, oder gänzlich unerreichbar zu machen.

Fehler beim Parsen der Zertifikatskette

Klasse: möglicher Speicherzugriffsfehler
Funktion: `PKIProvider::SplitCertificateChain()`
Referenzen in Valgrind logs:

Valgrind log	Zeilen
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

Tabelle 27: Auftreten des Fehlers beim Parsen von Zertifikatsketten in den Valgrind logs

Technische Detailbeschreibung:

Beim empfangenen Sender Zertifikat kann es sich um eine Zertifikatskette handeln, in der mehrere Zertifikate hintereinandergereiht werden. Um auf die einzelnen Elemente zugreifen zu können, wird ein Index erstellt, der aus einem Zeiger auf den jeweiligen Beginn eines Zertifikats und dessen Länge besteht. Bei der Erstellung dieses Index wird nicht überprüft, ob das letzte Zertifikat vollständig enthalten ist. Ist dies nicht der Fall, ist die Längenangabe ungültig. Ein Zugriff auf die volle Länge dieses Speicherbereichs (z. B. mit memcpy) führt zu einem Speicherzugriffsfehler.

Auswirkung:

Die konkrete Auswirkung der fehlerhaften Speicherblocklänge hängt vom weiteren Umgang mit dem Speicherbereich ab. Möglich ist zum Beispiel die Weitergabe an OpenSSL zur Validierung. Wird hierbei der Speicherbereich mit Hilfe eines ASN.1 Parsers ausgewertet, könnten darin enthaltene Fehler u.a. zu Abstürzen führen.

Der fehlerhafte Index wird im weiteren Verlauf an die Applikation übergeben, in der durch die weitere Verarbeitung zusätzliche Fehler auftreten können.

In den absehbaren Fällen könnte es zu Lesezugriffen auf den ungültigen Speicher und somit evtl. zu Abstürzen führen. Ein Angreifer könnte den Server somit unerreichbar machen.

Fehler beim Senden eines ServiceFaults durch unerwarteten Request Body

Klasse: Speicherzugriffsfehler mit konstanter Größe 4 Byte
Funktion: `OpcUa_Endpoint_BeginProcessRequest()`
Referenzen in Valgrind logs:

Valgrind log	Zeilen
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,

Valgrind log	Zeilen
	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

Tabelle 28: Auftreten des Fehlers beim Senden eines serviceFaults in den Valgrind logs

Technische Detailbeschreibung:

Senden eines gültigen serialisierten Datentyps mit einer Speichergröße kleiner der von OpcUa_RequestHeader. Durch eine ungeprüfte Konvertierung in OpcUa_RequestHeader greift ein folgender Zugriff auf RequestHandle über den gültigen Speicherbereich des ursprünglichen Datentyps hinaus. Die maximale Länge beträgt 4 Byte. Diese Daten werden im ResponseHeader als RequestHandle an den Client zurückgesendet.

Auswirkung:

Die konkreten Auswirkungen beim Zugriff auf ungültigen Speicher sind systemspezifisch. Kommt es nicht zum Absturz, wird der Speicherinhalt der Länge 4 Byte an den Client gesendet. Gezieltes Auslesen von Speicherinhalten ist aufgrund der geringen Größe unwahrscheinlich. Im Falle eines Absturzes könnte der Angreifer den Server unerreichbar machen.

Fehlende Initialisierung einer Stackvariable

Klasse: Sprung auf Basis unitialisierter Daten und evtl. Speicherfreigabe mit ungültiger Adresse
Funktion: `OpcUa_SecureListener_ProcessOpenSecureChannelRequest()`
Referenzen in Valgrind logs:

Valgrind log	Zeilen
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

Tabelle 29: Auftreten des Zugriffs auf eine nicht initialisierte Variable in den Valgrind logs

Technische Detailbeschreibung:

Tritt beim Dekodieren des Nachrichten Headers einer OpenSecureChannel Nachricht ein Fehler auf, bevor der String mit der Security Policy dekodiert wurde, wird die Methode `OpcUa_String_Clear()` auf die uninitialisierte Stackvariable angewandt. Abhängig vom zufälligen Speicherinhalt der Struktur kann es zu einem Freigabeversuch einer zufälligen Speicheradresse kommen. Das weitere Verhalten ist von der Speicheradresse abhängig.

Auswirkung:

Die exakte Auswirkung hängt von zufälligem Speicherinhalt ab. Im schlimmsten Fall wird versucht, eine ungültige Speicheradresse freizugeben, was in den meisten Fällen zu einem Absturz führt. Ein Angreifer könnte somit versuchen, den Server unerreichbar zu machen.

In seltenen Fällen könnte dies zu einem double-free exploit führen. Das wird aber für sehr unwahrscheinlich erachtet, da die Adresse, welche an `free()` übergeben wird nicht von außen beeinflussbar ist.

Zusammenfassung:

Die bei der dynamischen Codeanalyse identifizierten Fehler wirken sich durch Vollschreiben des Arbeitsspeichers oder Abstürze hauptsächlich negativ auf die Verfügbarkeit des Servers aus. Diese Fehler können aber potentiell auch andere Auswirkungen haben, die schwer abzuschätzen sind.

Für eine abschließende und endgültige Beurteilung der Fehlermeldungen und Auswirkungen der beschriebenen Fehler ist eine weitere detailliertere Analyse notwendig.

8.8 Codeabdeckung

8.8.1 Messung

Zur Messung der Codeabdeckung zu Laufzeiten werden beim Kompilieren der Serverapplikation folgende C Flags benötigt: `-fprofile-arcs -ftest-coverage`

Diese Einstellungen bewirken, dass mit einem Werzeug namens *lcov* gemessen werden kann, wie oft jede Zeile Code ausgeführt wird und welche Abzweigungen durchlaufen werden. Auch die Anzahl der aufgerufenen Funktionen wird festgehalten.

Die Codeabdeckung kann bei einzelnen Tests gemessen und anschließend für ganze Testserien aggregiert werden. Mit einem weiteren Werkzeug *genhtml* können die Ergebnisse übersichtlich graphisch dargestellt werden, wobei hierfür die Ordnerstruktur und Dateien des Quellcodes verwendet werden.

Die mit debug Informationen kompilierte Version des Servers wurde auch immer so kompiliert, dass gleichzeitig die Codeabdeckung gemessen werden konnte. Entsprechend liegen für folgende Testserien Zahlen vor:

Testseriennr.	Testserienname	securityMode	Testziel
001	*sequential individual parameter fuzzing*	None	dynamische Codeanalyse
002	*random individual parameter fuzzing*	None	dynamische Codeanalyse
003	*random multiple parameter fuzzing*	None	dynamische Codeanalyse
007	*sequential individual parameter fuzzing*	SignAndEncrypt	dynamische Codeanalyse
009	*random multiple parameter fuzzing*	SignAndEncrypt	Erhöhung Codeabdeckung

Testseriennr.	Testserienname	securityMode	Testziel
013	*certificate tests*	SignAndEncrypt	Zertikatstests
015	*sequential individual parameter fuzzing*	Sign	Verifizierung Annahmen Codeabdeckung
016	*random multiple parameter fuzzing*	None	Verifizierung Annahmen Codeabdeckung

Tabelle 30: Liste der Testserien, bei denen die Codeabdeckung gemessen wurde

Die ersten vier Testserien 001, 002, 003 und 007 dienten der eigentlichen dynamischen Codeanalyse. Die Testserie 009 diente dazu, eine weitere Erhöhung der Codeabdeckung zu erreichen. Bei der Testserie 013 wurde die Funktionalität zur Validierung von Zertifikaten geprüft und hierbei die Codeabdeckung gemessen. Die Testserien 015 und 016 dienten der Prüfung, ob mit einer anderen Parameterwahl bei den Fuzzingtests eine höhere Codeabdeckung hätte erreicht werden können. Somit dienten diese Tests der Verifizierung folgender Annahmen zur Codeabdeckung:

- Annahme 1: Die Tests mit securityMode Sign (Testserie 015) erhöhen die Codeabdeckung nicht wesentlich.
- Annahme 2: Die Erhöhung der Anzahl der Iterationen (Testserie 009) erhöht die Codeabdeckung nicht wesentlich.
- Annahme 3: Die Erhöhung der Anzahl der Iterationen je choice (Testserie 016) erhöht die Codeabdeckung nicht wesentlich.

Auf Grund von technischen Problemen (Absturz von *Peach* und der Serverrahmenapplikation) liegt bei der Testserie 009 keine Messung der Codeabdeckung für die Tests 02 und 03 vor.

Die Tabellen 31, 32 und 33 listen abhängig von den durchgeführten Tests die Codeabdeckung für Zeilen, Funktionen und Abzweigungen, jeweils als Prozentzahl vom gesamten Quellcode und als absoluten Wert auf. Die grauen Zeilen in diesen drei Tabellen dienen als Vergleichsgrundlage zur Verifizierung der Annahmen.

#	Grundlage für die Messung	Zeilen- abdeckung		Funktionen- abdeckung		Abzweigungen- abdeckung	
1	Server starten und ohne Tests wieder beenden	9,8%	3157	11,8%	255	3,8%	1051
2	Testserie 001	35.3%	11328	42.4%	913	17.8%	4935
3	Testserie 002	31.9%	10251	36.2%	779	16.5%	4574
4	Testserie 003	34.1%	10953	38.5%	829	18.4%	5114
5	Testserie 007	38.3%	12314	44.7%	962	19.4%	5380
6	Testserien 001, 002, 003, 007	41,8%	13428	49,5%	1066	22,7%	6287
7	Testserien 001, 002, 003, 007 und Testserie 015 mit securityMode Sign (Verifizierung Annahme 1)	41,8%	13436	49,5%	1066	22,7%	6293
8	Testserien 001, 002, 003, 007 und Testserie 013 Zertifikatstests	41,9%	13442	49,5%	1066	22,7%	6301
	Anzahl für gesamten Quellcode	**100%**	**32119**	**100%**	**2153**	**100%**	**27754**

Tabelle 31: Erreichte Codeabdeckung abhängig von den durchgeführten Testserien (Annahme 1)

Verifizierung der Annahme 1: Die Auswertung der Testserien 001, 002, 003, 007 und 015 (Tabelle 31 Zeile 7) führt zu keiner höheren Codeabdeckung im Vergleich zur Codeabdeckung der Testserien 001, 002, 003 und 007 (Tabelle 31 Zeile 6).

Anmerkung zu Zeile 8 in Tabelle 31: Die kaum veränderte Codeabdeckung nach Durchführung der Zertifikatstests lässt sich dadurch erklären, dass ein Großteil der Validierung der Zertifikate in OpenSSL, also der eingebundenen Kryptobibliothek, ausgelagert ist.

#	Grundlage für die Messung	Zeilen- abdeckung		Funktionen- abdeckung		Abzweigungen- abdeckung	
1	Test 04 der Testserien 001, 002, 003, 007	36.8%	11822	42.2%	909	18.9%	5246
2	Test 04 der Testserien 001, 002, 003, 007 und Testserie 009 langer Testlauf (Verifizierung Annahme 2)	39.0%	12533	45.9%	988	20.4%	5672
3	Test 07 der Testserien 001, 002, 003, 007	36.8%	11814	42.0%	904	19.0%	5266
4	Test 07 der Testserien 001, 002, 003, 007 und Testserie 009 langer Testlauf (Verifizierung Annahme 2)	43.0%	13797	52.8%	1137	23.6%	6552
	Anzahl für gesamten Quellcode	**100%**	**32119**	**100%**	**2153**	**100%**	**27754**

Tabelle 32: Erreichte Codeabdeckung abhängig von der Gesamtanzahl der Iterationen (Annahme 2)

Verifizierung der Annahme 2: Die Auswertung der Tests 04 und 07 der Testserien 001, 002, 003, 007 und 009 (Tabelle 32 Zeilen 2 und 4) führt zwar zu einer höheren Codeabdeckung im Vergleich zur Codeabdeckung der gleichen Tests für die Testserien 001, 002, 003 und 007 (Tabelle 32 Zeilen 1 und 3). Die Erhöhung der Codeabdeckung relativiert sich jedoch, wenn man die kompletten Testserien 001, 002, 003 und 007 betrachtet. Außerdem muss berücksichtigt werden, dass gerade beim Test 07, bei dem die Steigerung der Codeabdeckung am deutlichsten ausfällt, die Anzahl der Iterationen bei der Testserie 009 um ein Faktor 18,3 die Summe der Anzahl an Iterationen für die Testserien 001, 002, 003 und 007 übersteigt.

#	Grundlage für die Messung	Zeilen- abdeckung		Funktionen- abdeckung		Abzweigungen- abdeckung	
1	Test 02 der Testserien 001, 002, 003, 007	33.9%	10888	39.9%	859	17.4%	4829
2	Test 02 der Testserien 001, 002, 003, 007 und Testserie 016 mit weiteren 10k Iterationen je choice (Verifizierung Annahme 3)	34.0%	10905	39.9%	859	17.5%	4853
3	Test 04 der Testserien 001, 002, 003, 007	36.8%	11822	42.2%	909	18.9%	5246
4	Test 04 der Testserien 001, 002, 003, 007 und Testserie 016 mit weiteren 10k Iterationen je choice (Verifizierung Annahme 3)	37.1%	11904	42.6%	917	19.0%	5287
5	Test 07 der Testserien 001, 002, 003, 007	36.8%	11814	42.0%	904	19.0%	5266
6	Test 07 der Testserien 001, 002, 003, 007 und Testserie 016 mit weiteren 10k Iterationen je choice (Verifizierung Annahme 3)	37.0%	11872	42.4%	913	19.1%	5296
	Anzahl für gesamten Quellcode	**100%**	**32119**	**100%**	**2153**	**100%**	**27754**

Tabelle 33: Erreichte Codeabdeckung abhängig von der Anzahl der Iterationen je choice (Annahme 3)

Verifizierung der Annahme 3: Beim Fuzzing mit der Fuzzingstrategie random wurde die Anzahl der Iterationen je choice auf 5.000 für die Testserien 002 und 003 festgelegt. Für die Testserie 016 wurde die Anzahl der Iterationen je choice auf 10.000 erhöht. Die Verdreifachung der Anzahl der Iterationen je choice in den Zeilen 2, 4 und 6 in Tabelle 33 führte aber nicht zu einer deutlichen Erhöhung der Codeabdeckung im Vergleich zu den Ergebnissen mit 5.000 Iterationen je choice in den Zeilen 1, 3 und 5.

8.8.2 Analyse der Ergebnisse

Die Testserien 001, 002, 003 und 007 weisen eine Codeabdeckung von ca. 43% bei den Zeilen und ca. 24% bei den Abzweigungen auf. Dies kann folgendermaßen erklärt werden:

- Die Testapplikation ist ein single-threaded Server. Dies hat zur Folge, dass folgende Codebestandteile nicht durchlaufen werden:
  - Code, der nur bei Multithreading aktiv ist (Threading; Synchronisation)
  - Synchrone APIs; diese bezeichnen Funktionen, die blockierend arbeiten, was nur im Multithread Betrieb möglich ist. Stattdessen werden asynchrone APIs verwendet.
  - Funktionen, die ausschließlich Clientfunktionalität enthalten. Da nur die Serverseite getestet wurde, wurden keine Clientfunktionen durchlaufen.
    Die Analyse der Codeabdeckung kann mit der graphischen Darstellung der Ergebnisse mittels *genhtml* auf die Ebene der Quellcodedateien verfeinert werden. Dabei weisen Dateinamen mit den Bestandteilen 'Connection' und 'Channel' auf Clientcode hin, die Namensbestandteile 'Listener' und 'Endpoint' hingegen auf Servercode hin.
- Bestimmte Funktionalitäten wurden bei den Tests gar nicht oder nur beschränkt untersucht:
  - Wegen ihrer Relevanz für die IT Sicherheit wurden die Tests so priorisiert, dass ausschließlich die drei Dienste SecureChannel, Session und Discovery betrachtet wurden. Entsprechend wurde der Code für die weiteren Dienste wie z. B. Attribute oder Subscription nicht ausgeführt. Dies gilt auch für die große Anzahl an Datenstrukturen, die nur bei diesen nicht untersuchten Diensten zum Einsatz kommt.
  - In der Testumgebung stand immer ausreichend Arbeitsspeicher zur Verfügung, so dass keine Abzweigungen wegen fehlgeschlagener Speicherallozierungen durchlaufen wurden.
  - Mechanismen, wie das Aufteilen von besonders langen Inhalten auf mehrere Nachrichten (chunking), die Erneuerung von SecureChannels, Verbindungsabbrüche, usw. wurden nicht getestet.
  - An manchen Stellen enthält der Quellcode bereits Teile von Funktionalitäten, wie bestimmte Kryptofunktionen, die aber erst in künftigen Versionen vervollständigt werden und entsprechend erst dort zum Einsatz kommen können.

8.9 Proof-of-Concept

Im Rahmen dieses Projekts wurde kein Proof-of-Concept realisiert. Es wird empfohlen, im Nachgang zu diesem Projekt die Ausnutzbarkeit folgender Schwachstellen durch Proof-of-Concepts zu belegen:

- replay Angriffe unter Ausnutzung der fehlenden sequenceNumber Prüfung
- Ausnutzung von kompromittierten Zertifikaten, deren Gültigkeit wegen der nicht stattfindenden CRL Prüfung nicht erkannt wird
- Denial of Service Angriffe unter Ausnutzung von Speicherlecks

8.10 Zusammenfassung

Folgende Ergebnisse der Tests der Referenzimplementierung des Kommunikationsstacks der OPC Foundation sind festzuhalten.

Zertifikatstests

Die Analyse der Zertifikatstestergebnisse hat ergeben, dass der Kommunikationsstack der OPC Foundation in der getesteten Version 1.02.344.5 beim Einsatz von selbstsignierten Zertifikaten sich spezifikationskonform bezüglich des Erlaubens oder Ablehnens von Verbindungen verhält.

Die Funktionalität zur Prüfung von CRLs konnte in der vorliegenden Rahmenapplikation nicht getestet werden.

Statische Codeanalyse

Die statische Codeanalyse führte zu ca. 340 Fehlermeldungen, von denen die allermeisten in Bezug auf IT Sicherheit irrelevant sind: Die nicht weiter betrachteten Fehlermeldungen bezogen sich auf die mögliche Einschränkung des Gültigkeitsbereichs von Variablen, auf toten oder nicht benutzen Quellcode oder auf Quellcode des Linux Plattform Layers, der nicht Bestandteil des OPC Foundation Stacks ist. Diese Fehler im Code sind zwar verbesserbar, haben aber nach unserer Einschätzung keine direkte Relevanz für die IT Sicherheit des Kommunikationsstacks.

Die restlichen Meldungen wurden genauer analysiert. Dabei ergaben sich folgende zwei für die IT Sicherheit bedeutsame Fehler: Die Funktion `gethostbyname` ist obsolet und nicht wiedereintrittsfähig (Engl. 'reentrant'). Dies kann bei multi-threaded Servern in seltenen Fällen zu Verbindungsproblemen führen. Auch Abstürze können je nach Implementierung der Funktion nicht vollkommen ausgeschlossen werden.

Die identifizierten Fehler wurden als nicht so gravierend bezüglich IT Sicherheit eingestuft, dass ein Proof-of-Concept zu entwickeln gewesen wäre.

Fuzzing

Die Entwicklung der Tests in *Peach* (Modellierung) führte zur Identifizierung folgender Fehler:

- Die sequenceNumber wird in UASC nicht geprüft. Dies stellt eine Sicherheitslücke dar, deren Ausnutzbarkeit im Rahmen eines Proof-of-Concepts noch zu zeigen ist.
- Mehrere Abweichungen zwischen Spezifikation und Implementierung des OPC Foundation Stacks, sowie Inkonsistenzen in der Spezifikation

Trotz umfangreicher Tests mit mehr als 15 Millionen Iterationen führten die Fuzzingtests zu keiner weiteren Identifikation von Implementierungsfehlern.

Dynamische Codeanalyse

Die bei der dynamischen Codeanalyse identifizierten Fehler wirken sich durch Ausschöpfung des Arbeitsspeichers oder Abstürze hauptsächlich negativ auf die Verfügbarkeit des Servers aus. Diese Fehler können aber potentiell auch andere Auswirkungen haben, die zum jetzigen Zeitpunkt der Analyse schwer abzuschätzen sind.

Für eine abschließende und endgültige Beurteilung der Fehlermeldungen und Auswirkungen der beschriebenen Fehler ist eine weitere detailliertere Analyse notwendig.

Codeabdeckung

Die durchgeführten Testserien führten zu einer Codeabdeckung von ca. 43% bei den Zeilen und ca. 23% bei den Abzweigungen. Der Grund für die relativ niedrige Codeabdeckung ist darauf zurückzuführen, dass z. B. multi-threading Codeabschnitte und Clientfunktionalitäten nicht durchlaufen werden (Test eines single-threaded Servers). Zudem wurden unter anderen weitere OPC UA Dienste wie z. B. der Subscription Dienst nicht getestet, da auf die drei Dienste SecureChannel, Session und Discovery wegen ihrer Bedeutung für die IT Sicherheit fokussiert wurde.

Des Weiteren wurde verifiziert, dass durch die Erhöhung der Anzahl der Iterationen (auch in choices) und das Testen mit securityMode Sign die Codeabdeckung nicht erhöht werden kann.

9 Ausblick

In diesem Projekt konnten nicht alle Bereiche der IT Sicherheit analysiert und getestet werden. Jedoch können als Erkenntnisgewinn folgende weitere Arbeitspunkte für die Untersuchung der IT Sicherheit der OPC UA Spezifikation und der Implementierung des Kommunikationsstacks angegeben werden. Die Themen wurden nach unserer Einschätzung in absteigender Priorität sortiert:

- weitere Analyse der Valgrind Fehlermeldungen: Da Wechselwirkungen zwischen Implementierungsfehlern bei Testläufen mit mehreren zehntausend Iterationen nicht ausgeschlossen sind, wird empfohlen, die identifizierten Fehler im Quellcode zu beheben. Anschließend kann eine erneute dynamische Codeanalyse mit neugewonnenen Daten durchgeführt werden.
- Außerdem kann durch eine Weiterführung der Analyse der bereits identifizierten Fehler ermittelt werden, welche Auswirkungen diese genau haben.
- LDS (local discovery server) mit RegisterServer testen: Beim Discovery Dienst wurden die Nachrichten eines Clients an einen Server getestet. Diese Tests können erweitert werden, indem auch die RegisterServer Nachricht von einem Server an einen LDS geprüft wird.
- Verifizierung der Ausnutzbarkeit von identifizierten Schwachstellen der Implementierung mittels Proof-of-Concepts
- Testen von Mechanismen zur Verwaltung von Verbindungen: Mechanismen wie das chunking, die Erneuerung von SecureChannels, die Simulation von Verbindungsabbrüchen und -wiederaufbau oder Übertragungsfehlern können zusätzlich noch getestet werden.
- weitere Werkzeuge für statische und dynamische Codeanalyse: Der Einsatz von unterschiedlichen Werkzeugen kann sowohl bei der statischen als auch bei der dynamischen Codeanalyse zu neuen Erkenntnissen führen.
- weitere Datenstrukturen auf Serverseite testen: Die bisherige Prüfung der Referenzimplementierung beschränkt sich auf die drei Dienste SecureChannel, Session und Discovery. Entsprechend wurden nur die Datenstrukturen modelliert und getestet, die für diese Dienste erforderlich sind. Um die encoder/decoder Funktionalität ausführlicher zu testen, können weitere Datenstrukturen in der Datenmodellierung hinzugefügt werden.
  Auch das in Abschnitt 8.6.1 erwähnte Testen von rekursiven Datenstrukturen könnte ergänzend hinzugenommen werden.
- Clientseite testen: Bei den in AP4 durchgeführten Tests wurde ausschließlich die Clientseite simuliert, um die Serverseite zu testen. Es kann jedoch Sinn machen, auch die Codestellen unter die Lupe zu nehmen, die nicht von Servern sondern von Clients ausgeführt werden.
- weitere Dienste auf Serverseite testen: Die bisherige Prüfung beschränkte sich auf das Testen der Referenzimplementierung des Kommunikationsstacks. Da der Stack aber nur die Dienste SecureChannel, Session und Discovery abdeckt, kann es sinnvoll sein, die Untersuchung auf alle weiteren Dienste der OPC UA Spezifikation zu erweitern.
- Testen von korrekten und fehlerhaften Nachrichten in falscher Reihenfolge: Bei den bisherigen Tests wurde die von OPC UA vorgegebenen Reihenfolge der Nachrichten bei einem Verbindungsaufbau respektiert. Diese Tests können erweitert werden, indem nicht nur die Inhalte, sondern auch die Reihenfolge der Nachrichten variiert wird.
- Veränderung der Testumgebung: Alternativen sind 64-Bit Architekturen, multi-threaded Applikationen und Windows Plattformen.
- Modellierung mit einem Werkzeug für die Verifizierung von Protokollen: Zur exakten Prüfung der IT Sicherheit des OPC UA Protokolls kann die Modellierung mit formalen Methoden beitragen, um eventuelle systematische Fehler zu identifizieren.
- andere Programmiersprachen: Es wurde im Rahmen dieses Projekts die Referenzimplementierung in ANSI C des OPC UA Kommunikationsstacks geprüft. Es liegen aber weitere Implementierungen in anderen Programmiersprachen vor.
- neue 1.03 Spezifikation: Bisher wurde die bestehende Spezifikation in der Version 1.02 untersucht. Im Laufe des Projekts wurden allerdings Teile der neuen Spezifikation veröffentlicht. Sobald alle Dokumente vorhanden sind, kann auch die neue Version untersucht werden.

Anhang A: Ergänzungen zu Abschnitt 8

Ergänzungen zu Abschnitt 8.4 Zertifikatstests:

Tabelle 34 listet die 92 durchgeführten Zertifikatstests auf und ist folgendermaßen aufgebaut:

Zur besseren Lesbarkeit der Tabelle wurden Spalten, die zusammengehören, farblich hervorgehoben. So ergeben sich drei Gruppen, jeweils für die CA (Spalten 3 und 4: CA, CRL), für die Sub CA (Spalten 5 bis 8: SCA, Fehler, CRL, Widerrufen) und für das Client Zertifikat (Spalten 9 bis 11: Cert, Fehler, Widerrufen).

Bedeutung der einzelnen Spalten:

Nr.: eindeutige Testnummer

Typ: bezieht sich auf die PKI Struktur. 'Self' steht für selbstsignierte Zertifikate, 'Root' für eine PKI mit einer einfachen CA und 'Secondary' für eine PKI mit root CA und sub CAs.

CA: zeigt an, ob ein CA Zertifikat zur Verifizierung der Zertifikatskette vorhanden ist und falls ja, woher das CA Zertifikat kommt und ob ihm vertraut wird. 'Chain' bedeutet, dass das CA Zertifikat in der Nachricht enthalten ist. 'Trusted' bedeutet, dass das CA Zertifikat im PKI store abgelegt ist und ihm bereits vertraut wird. 'Issuers' bedeutet ebenfalls, dass das CA Zertifikat im PKI store vorhanden ist, allerdings wird dem Zertifikat hier nicht vertraut.

CRL: zeigt an, ob eine CRL für die CA (Spalte 4) oder die sub CA (Spalte 7) vorhanden ist.

SCA: zeigt an, ob ein sub CA Zertifikat zur Verifizierung der Zertifikatskette vorhanden ist. Die drei möglichen Werte 'Chain', 'Trusted' und 'Issuers' haben die gleiche Bedeutung, wie in der Spalte CA.

Fehler: zeigt an, ob das sub CA Zertifikat (Spalte 6) oder das Client Zertifikat (Spalte 10) gültig ist oder nicht. 'Signature' bedeutet, dass die Signatur fehlerhaft ist, 'NotYet', dass das Zertifikat noch nicht gültig ist, 'Expired', dass das Zertifikat bereits abgelaufen ist.

Widerrufen: zeigt an, ob das sub CA Zertifikat (Spalte 8) oder das Client Zertifikat (Spalte 11) widerrufen ist.

Cert: zeigt an, ob dem Client Zertifikat vertraut wird oder nicht.

Konfiguration: werden bestimmte Fehlerklassen ignoriert. Ist 'IgnInvTime' gesetzt, löst ein zeitlich ungültiges Zertifikat keinen Fehler aus. Ist 'IgnMissCRL' gesetzt, löst eine fehlende CRL keinen Fehler aus.

Serverantwort: erwarteter Statuscode, mit dem der Server antworten sollte. Kommt es hier zu einer Abweichung, wird der jeweilige Test als fehlgeschlagen markiert. Eine farbliche Unterlegung bedeutet, dass die Tests fehlgeschlagen sind. Bei blauen Zellen lag die Ursache für den Fehler außerhalb des Stacks. Die gelb markierten Felder weisen auf Abweichungen beim StatusCode ohne Auswirkung auf die IT Sicherheit hin.

In der Tabelle bedeutet ein Minuszeichen, dass kein Wert gesetzt ist.

Nr.	Typ	CA	CRL	SCA	Fehler	CRL	Wider-rufen	Cert	Fehler	Wider-rufen	Konfiguration	Serverantwort
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

Nr.	Typ	CA	CRL	SCA	Fehler	CRL	Wider-rufen	Cert	Fehler	Wider-rufen	Konfiguration	Serverantwort
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, Issuers	yes	-	-	-	-	Trusted	-	-	-	Good
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

Nr.	Typ	CA	CRL	SCA	Fehler	CRL	Wider-rufen	Cert	Fehler	Wider-rufen	Konfiguration	Serverantwort
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
56	Secondary	Trusted	yes	-	-	-	-	-	-	-	-	CertificateUntrusted
57	Secondary	Issuers	-	Issuers	-	yes	-	Trusted	-	-	-	IssuerRevocationUnknown
58	Secondary	Issuers	-	Trusted	-	yes	-	-	-	-	-	IssuerRevocationUnknown
59	Secondary	Trusted	-	Issuers	-	yes	-	-	-	-	-	IssuerRevocationUnknown

Nr.	Typ	CA	CRL	SCA	Fehler	CRL	Wider-rufen	Cert	Fehler	Wider-rufen	Konfiguration	Serverantwort
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, Trusted	-	-	-	-	-	-	IgnMissCRL	Good

Nr.	Typ	CA	CRL	SCA	Fehler	CRL	Wider-rufen	Cert	Fehler	Wider-rufen	Konfiguration	Serverantwort
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

Tabelle 34: Zertifikatstests mit erwartetem Ergebnis

Ergänzungen zu Abschnitt 8.6.2 Auswahl der Fuzzingtests:

In folgender Tabelle 35 sind alle durchgeführten Fuzzingtests aufgelistet. In der letzten Spalte steht in Kurzform, welches der vier in Abschnitt 8.6.2 erwähnten Ziele bei der jeweiligen Testserie verfolgt wurde. Die Spalte Umgebung beschreibt, welcher Client und welcher Server benutzt wurden, wobei aus Gründen der Übersichtlichkeit folgende Abkürzungen verwendet werden:

- 'debug' steht für:
  - Client: *Peach* v3.1.124.0
  - Server: *Valgrind* mit debug Version des Servers
- 'release' steht für:
  - Client: *Peach* v3.1.124.0
  - Server: release Version des Servers

Test-seriennr.	Testserienname	Testnr.	Testname	securityMode	Umgebung	Testziel
001	sequential individual parameter fuzzing	01	HEL message fuzzing	–	debug	dynamische Codeanalyse
001	sequential individual parameter fuzzing	02	OPN message fuzzing	None	debug	dynamische Codeanalyse
001	sequential individual parameter fuzzing	03	CreateSession message fuzzing	None	debug	dynamische Codeanalyse
001	sequential individual parameter fuzzing	04	ActivateSession message fuzzing	None	debug	dynamische Codeanalyse
001	sequential individual parameter fuzzing	05	CloseSession message fuzzing	None	debug	dynamische Codeanalyse
001	sequential individual parameter fuzzing	06	CloseChannel message fuzzing	None	debug	dynamische Codeanalyse
001	sequential individual parameter fuzzing	07	FindServers message fuzzing	None	debug	dynamische Codeanalyse
001	sequential individual parameter fuzzing	08	GetEndpoints message fuzzing	None	debug	dynamische

Test-seriennr.	Testseriename	Testnr.	Testname	securityMode	Umgebung	Testziel
						Codeanalyse
002	random individual parameter fuzzing	01	HEL message fuzzing (5.000 Iterationen je choice)	--	debug	dynamische Codeanalyse
002	random individual parameter fuzzing	02	OPN message fuzzing (5.000 Iterationen je choice)	None	debug	dynamische Codeanalyse
002	random individual parameter fuzzing	03	CreateSession message fuzzing (5.000 Iterationen je choice)	None	debug	dynamische Codeanalyse
002	random individual parameter fuzzing	04	ActivateSession message fuzzing (5.000 Iterationen je choice)	None	debug	dynamische Codeanalyse
002	random individual parameter fuzzing	05	CloseSession message fuzzing (5.000 Iterationen je choice)	None	debug	dynamische Codeanalyse
002	random individual parameter fuzzing	06	CloseChannel message fuzzing (5.000 Iterationen je choice)	None	debug	dynamische Codeanalyse
002	random individual parameter fuzzing	07	FindServers message fuzzing (5.000 Iterationen je choice)	None	debug	dynamische Codeanalyse
002	random individual parameter fuzzing	08	GetEndpoints message fuzzing (5.000 Iterationen je choice)	None	debug	dynamische Codeanalyse
003	random multiple parameter fuzzing	01	HEL message fuzzing (5.000 Iterationen je choice)	--	debug	dynamische Codeanalyse
003	random multiple parameter fuzzing	02	OPN message fuzzing (5.000 Iterationen je choice)	None	debug	dynamische Codeanalyse
003	random multiple parameter fuzzing	03	CreateSession message fuzzing (5.000 Iterationen je choice)	None	debug	dynamische Codeanalyse
003	random multiple parameter fuzzing	04	ActivateSession message fuzzing (5.000 Iterationen je choice)	None	debug	dynamische Codeanalyse

Test-seriennr.	Testseriename	Testnr.	Testname	securityMode	Umgebung	Testziel
003	random multiple parameter fuzzing	05	CloseSession message fuzzing (5.000 Iterationen je choice)	None	debug	dynamische Codeanalyse
003	random multiple parameter fuzzing	06	CloseChannel message fuzzing (5.000 Iterationen je choice)	None	debug	dynamische Codeanalyse
003	random multiple parameter fuzzing	07	FindServers message fuzzing (5.000 Iterationen je choice)	None	debug	dynamische Codeanalyse
003	random multiple parameter fuzzing	08	GetEndpoints message fuzzing (5.000 Iterationen je choice)	None	debug	dynamische Codeanalyse
004	sequential individual parameter fuzzing	01	HEL message fuzzing	--	release	Produktionsserver
004	sequential individual parameter fuzzing	02	OPN message fuzzing	None	release	Produktionsserver
004	sequential individual parameter fuzzing	03	CreateSession message fuzzing	None	release	Produktionsserver
004	sequential individual parameter fuzzing	04	ActivateSession message fuzzing	None	release	Produktionsserver
004	sequential individual parameter fuzzing	05	CloseSession message fuzzing	None	release	Produktionsserver
004	sequential individual parameter fuzzing	06	CloseChannel message fuzzing	None	release	Produktionsserver
004	sequential individual parameter fuzzing	07	FindServers message fuzzing	None	release	Produktionsserver
004	sequential individual parameter fuzzing	08	GetEndpoints message fuzzing	None	release	Produktionsserver
005	random individual parameter fuzzing	01	HEL message fuzzing (5.000 Iterationen je choice)	--	release	Produktionsserver
005	random individual parameter fuzzing	02	OPN message fuzzing (5.000 Iterationen je choice)	None	release	Produktionsserver
005	random individual parameter fuzzing	03	CreateSession message fuzzing (5.000 Iterationen je choice)	None	release	Produktionsserver
005	random individual parameter fuzzing	04	ActivateSession message fuzzing (5.000 Iterationen je choice)	None	release	Produktionsserver

Test-seriennr.	Testseriename	Testnr.	Testname	securityMode	Umgebung	Testziel
005	random individual parameter fuzzing	05	CloseSession message fuzzing (5.000 Iterationen je choice)	None	release	Produktionsserver
005	random individual parameter fuzzing	06	CloseChannel message fuzzing (5.000 Iterationen je choice)	None	release	Produktionsserver
005	random individual parameter fuzzing	07	FindServers message fuzzing (5.000 Iterationen je choice)	None	release	Produktionsserver
005	random individual parameter fuzzing	08	GetEndpoints message fuzzing (5.000 Iterationen je choice)	None	release	Produktionsserver
006	random multiple parameter fuzzing	01	HEL message fuzzing (5.000 Iterationen je choice)	--	release	Produktionsserver
006	random multiple parameter fuzzing	02	OPN message fuzzing (5.000 Iterationen je choice)	None	release	Produktionsserver
006	random multiple parameter fuzzing	03	CreateSession message fuzzing (5.000 Iterationen je choice)	None	release	Produktionsserver
006	random multiple parameter fuzzing	04	ActivateSession message fuzzing (5.000 Iterationen je choice)	None	release	Produktionsserver
006	random multiple parameter fuzzing	05	CloseSession message fuzzing (5.000 Iterationen je choice)	None	release	Produktionsserver
006	random multiple parameter fuzzing	06	CloseChannel message fuzzing (5.000 Iterationen je choice)	None	release	Produktionsserver
006	random multiple parameter fuzzing	07	FindServers message fuzzing (5.000 Iterationen je choice)	None	release	Produktionsserver
006	random multiple parameter fuzzing	08	GetEndpoints message fuzzing (5.000 Iterationen je choice)	None	release	Produktionsserver
007	sequential individual parameter fuzzing	02	OPN message fuzzing	SignAndEncrypt	debug	dynamische

Test-seriennr.	Testseriename	Testnr.	Testname	securityMode	Umgebung	Testziel
						Codeanalyse
007	sequential individual parameter fuzzing	03	CreateSession message fuzzing	SignAndEncrypt	debug	dynamische Codeanalyse
007	sequential individual parameter fuzzing	04	ActivateSession message fuzzing	SignAndEncrypt	debug	dynamische Codeanalyse
007	sequential individual parameter fuzzing	05	CloseSession message fuzzing	SignAndEncrypt	debug	dynamische Codeanalyse
007	sequential individual parameter fuzzing	06	CloseChannel message fuzzing	SignAndEncrypt	debug	dynamische Codeanalyse
007	sequential individual parameter fuzzing	07	FindServers message fuzzing	SignAndEncrypt	debug	dynamische Codeanalyse
007	sequential individual parameter fuzzing	08	GetEndpoints message fuzzing	SignAndEncrypt	debug	dynamische Codeanalyse
009	random multiple parameter fuzzing (long run)	02	OPN message fuzzing	SignAndEncrypt	debug	Erhöhung Codeabdeckung
009	random multiple parameter fuzzing (long run)	03	CreateSession message fuzzing	SignAndEncrypt	debug	Erhöhung Codeabdeckung
009	random multiple parameter fuzzing (long run)	04	ActivateSession message fuzzing	None	debug	Erhöhung Codeabdeckung
009	random multiple parameter fuzzing (long run)	07	FindServers message fuzzing	None	debug	Erhöhung Codeabdeckung
015	sequential individual parameter fuzzing	02	OPN message fuzzing	Sign	debug	Verifizierung Annahmen Codeabdeckung
015	sequential individual parameter fuzzing	03	CreateSession message fuzzing	Sign	debug	Verifizierung

Test-seriennr.	Testseriennname	Testnr.	Testname	securityMode	Umgebung	Testziel
						Annahmen Codeabdeckung
015	sequential individual parameter fuzzing	04	ActivateSession message fuzzing	Sign	debug	Verifizierung Annahmen Codeabdeckung
015	sequential individual parameter fuzzing	05	CloseSession message fuzzing	Sign	debug	Verifizierung Annahmen Codeabdeckung
015	sequential individual parameter fuzzing	06	CloseChannel message fuzzing	Sign	debug	Verifizierung Annahmen Codeabdeckung
015	sequential individual parameter fuzzing	07	FindServers message fuzzing	Sign	debug	Verifizierung Annahmen Codeabdeckung
015	sequential individual parameter fuzzing	08	GetEndpoints message fuzzing	Sign	debug	Verifizierung Annahmen Codeabdeckung
016	random individual parameter fuzzing	02	OPN message fuzzing (10.000 Iterationen je choice)	None	debug	Verifizierung Annahmen Codeabdeckung
016	random individual parameter fuzzing	04	ActivateSession message fuzzing (10.000 Iterationen je choice)	None	debug	Verifizierung Annahmen Codeabdeckung
016	random individual parameter fuzzing	07	FindServers message fuzzing (10.000 Iterationen je choice)	None	debug	Verifizierung Annahmen Codeabdeckung

Tabelle 35: Liste der durchgeführten Fuzzingtests

Anhang B: Compilerflags

Vollständige Liste der compiler flags für die debug und release Versionen des Servers.

Die Dateien der debug Version wurden mit folgendem Befehl kompiliert:

```
/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_SECURITY-
POLICY_BASIC256=1 -DOPCUA_SUPPORT_SECURITYPOLICY_NONE=1
-DOPCUA_HAVE_HTTPS=0 -DOPCUA_P_SOCKETMANAGER_SUPPORT_SSL=0 -DOPCUA_SUP-
PORT_PKI_WIN32=0 -DUASERVER_SERVICES_HISTORYREAD=1
-DUASERVER_SERVICES_HISTORYUPDATE=1 -DUASERVER_SERVICES_CALL=1 -DUASER-
VER_SUPPORT_EVENTS=1 -DOPCUA_MULTITHREADED=0
-DOPCUA_USE_SYNCHRONISATION=1 -DUASERVER_SUPPORT_AUTHORIZATION=1 -DUASER-
VER_SUPPORT_AUTHENTICATION_INTERNAL=1 -DOPCUA_USE_STATIC_PLATFORM_INTER-
FACE=1 -DOPCUA_HAVE_CLIENTAPI=1 -DOPCUA_HAVE_SERVERAPI=1 -DUASERVER_SUP-
PORT_AUTHENTICATION_PAM=0 -DUASERVER_SUPPORT_AUTHENTICATION_SASL=0
-DUASERVER_SUPPORT_DISCOVERY=1 -DOPCUA_GUID_STRING_USE_CURLYBRACE=1
-D_UA_STACK_BUILD_DLL -DOPCUA_SUPPORT_PKI=1
-DOPCUA_ENCODEABLE_OBJECT_COMPARE_SUPPORTED=0
-DOPCUA_ENCODEABLE_OBJECT_COPY_SUPPORTED=0 -DOPCUA_SUPPORT_SECURITYPO-
LICY_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 -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/opcua/ascolab_webdav/sdk/src/uastack/platforms/linux
-I/root/opcua/ascolab_webdav/sdk/src/uastack/core
-I/root/opcua/ascolab_webdav/sdk/src/uastack/stackcore
-I/root/opcua/ascolab_webdav/sdk/src/uastack/securechannel
-I/root/opcua/ascolab_webdav/sdk/src/uastack/transport/tcp
-I/root/opcua/ascolab_webdav/sdk/src/uastack/proxystub/clientproxy
-I/root/opcua/ascolab_webdav/sdk/src/uastack/proxystub/serverstub -Wall
-fno-strict-aliasing -Wno-format -Wfloat-equal -Wextra -o <Dateiname>.o
-c <Dateiname>
```

Die Optionen, die mit `-DOPCUA` und `-DUASERVER` anfangen, dienen der Einstellung von OPC UA interner Funktionalität.

Die Dateien der release Version des Servers wurden wie folgt kompiliert:

```
/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_SECURITY-
POLICY_BASIC256=1 -DOPCUA_SUPPORT_SECURITYPOLICY_NONE=1
-DOPCUA_HAVE_HTTPS=0 -DOPCUA_P_SOCKETMANAGER_SUPPORT_SSL=0 -DOPCUA_SUP-
PORT_PKI_WIN32=0 -DUASERVER_SERVICES_HISTORYREAD=1
-DUASERVER_SERVICES_HISTORYUPDATE=1 -DUASERVER_SERVICES_CALL=1 -DUASER-
VER_SUPPORT_EVENTS=1 -DOPCUA_MULTITHREADED=0
-DOPCUA_USE_SYNCHRONISATION=1 -DUASERVER_SUPPORT_AUTHORIZATION=1 -DUASER-
VER_SUPPORT_AUTHENTICATION_INTERNAL=1 -DOPCUA_USE_STATIC_PLATFORM_INTER-
FACE=1 -DOPCUA_HAVE_CLIENTAPI=1 -DOPCUA_HAVE_SERVERAPI=1 -DUASERVER_SUP-
PORT_AUTHENTICATION_PAM=0 -DUASERVER_SUPPORT_AUTHENTICATION_SASL=0
-DUASERVER_SUPPORT_DISCOVERY=1 -DOPCUA_GUID_STRING_USE_CURLYBRACE=1
-D_UA_STACK_BUILD_DLL -DOPCUA_SUPPORT_PKI=1
-DOPCUA_ENCODEABLE_OBJECT_COMPARE_SUPPORTED=0
-DOPCUA_ENCODEABLE_OBJECT_COPY_SUPPORTED=0 -DOPCUA_SUPPORT_SECURITYPO-
LICY_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/opcua/ascolab_webdav/sdk/src/uastack/platforms/linux
-I/root/opcua/ascolab_webdav/sdk/src/uastack/core
-I/root/opcua/ascolab_webdav/sdk/src/uastack/stackcore
-I/root/opcua/ascolab_webdav/sdk/src/uastack/securechannel
-I/root/opcua/ascolab_webdav/sdk/src/uastack/transport/tcp
-I/root/opcua/ascolab_webdav/sdk/src/uastack/proxystub/clientproxy
-I/root/opcua/ascolab_webdav/sdk/src/uastack/proxystub/serverstub -Wall
-fno-strict-aliasing -Wno-format -Wfloat-equal -Wextra -o <Dateiname>.o
-c <Dateiname>
```

Anhang C: Literatur- und Quellenverzeichnis

[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/opcua.html

Anhang D: Abbildungs- und Tabellenverzeichnis

Anhang E: Abkürzungsverzeichnis

BSI	Bundesamt für Sicherheit in der Informationstechnik
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 mit multicast-Erweiterung (Zeroconf)
MES	Manufacturing Execution System
OPC DA	OPC Data Access (Classic COM based OPC)
OPC UA	Open Platform Communications Unified Architecture
PKI	Public Key Infrastruktur
SCADA	Supervisory Control and Data Acquisition
SDK	Software Development Kit
SGIP	Smart Grid Interoperability Panels
UASC	UA Secure Conversation

Tabelle 36: Abkürzungsverzeichnis