

Masterarbeit

**Erweiterung sicherheitsrelevanter  
Software für die automatische Integritäts-  
prüfung von Endgeräten**

Daniel Wuttke  
Sommersemester 2006

Fachhochschule Hannover  
Fachbereich Informatik  
Studiengang Angewandte Informatik



---

# Über diese Masterarbeit

## Autor

Name: Daniel Wuttke  
Anschrift: Lüneburger Weg 9  
30900 Wedemark  
  
Matrikelnummer: 1015791  
  
Telefon: 05130 / 952073  
E-Mail: DanielWuttke@gmx.de

## Prüfer

Erstprüfer: Prof. Dr. Josef von Helden  
Fachhochschule Hannover / Fachbereich Informatik  
  
Zweitprüfer: Prof. Dr. Stefan Wohlfeil  
Fachhochschule Hannover / Fachbereich Informatik

## Selbstständigkeitserklärung

Hiermit erkläre ich, Daniel Wuttke, dass ich die eingereichte Masterarbeit selbstständig und ohne fremde Hilfe verfasst, keine anderen als die von mir angegebenen Quellen und Hilfsmittel benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Hannover, 29. August 2006

---

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Problemstellung und Ziele . . . . .	1
1.2	Aufbau der Arbeit . . . . .	2
1.3	Über diese Masterarbeit . . . . .	3
1.4	Typographische Konventionen . . . . .	4
<b>2</b>	<b>TNC-Architektur</b>	<b>5</b>
2.1	Trusted Computing Group und TNC . . . . .	5
2.2	Allgemeiner Aufbau des Architekturmodells . . . . .	6
2.3	Komponenten im Architekturmodell . . . . .	7
2.3.1	Integrity Measurement Collector (IMC) . . . . .	7
2.3.2	Integrity Measurement Verifier (IMV) . . . . .	8
2.3.3	Weitere Komponenten . . . . .	8
2.4	Schnittstellen im Architekturmodell . . . . .	9
2.4.1	Schnittstelle zwischen IMC und TNC Client (IF-IMC) . . . . .	9
2.4.2	Schnittstelle zwischen IMV und TNC Server (IF-IMV) . . . . .	9
2.4.3	Schnittstelle zwischen IMC und IMV (IF-M) . . . . .	10
2.4.4	Weitere Schnittstellen . . . . .	10
2.5	Eingesetzte Protokolle und Technologien . . . . .	11
2.6	Ablauf einer Integritätsprüfung . . . . .	11
<b>3</b>	<b>Allgemeine API für IMCs und IMVs</b>	<b>15</b>
3.1	Anforderungen an die API . . . . .	15
3.1.1	Funktionale Anforderungen . . . . .	15
3.1.2	Nicht-funktionale Anforderungen . . . . .	16
3.2	Softwarearchitektur der API . . . . .	17
3.2.1	Abstrakter IMC . . . . .	19
3.2.2	Abstrakter IMV . . . . .	20
3.2.3	Logging . . . . .	22
3.2.4	Validierung . . . . .	23
3.2.5	Windows Registry und WMI . . . . .	26
3.2.6	Weitere Building-Blocks . . . . .	28
3.3	Ablauf einer Integritätsprüfung mit der API . . . . .	28

<b>4</b>	<b>Integrity Measurement: Windows Registry</b>	<b>33</b>
4.1	Motivation . . . . .	33
4.2	Anforderungen und Vorgehen . . . . .	33
4.3	Aufbau der XML-Konfiguration . . . . .	34
4.3.1	Werte aus der Registrierung auslesen . . . . .	35
4.3.2	Verschachtelte Einträge . . . . .	37
4.3.3	Werte als Attribute zurückgeben . . . . .	38
4.3.4	Werte aller Unterschlüssel auslesen . . . . .	38
4.4	Basiskonfiguration des IMVs . . . . .	39
4.5	Ablauf der Integritätsprüfung . . . . .	40
4.6	Mögliche Prüfungsergebnisse . . . . .	41
<b>5</b>	<b>Integrity Measurement: Host-Scanner</b>	<b>43</b>
5.1	Motivation . . . . .	43
5.2	Anforderungen und Vorgehen . . . . .	44
5.3	Aufbau der XML-Konfiguration . . . . .	44
5.4	Basiskonfiguration des IMVs . . . . .	46
5.5	Ablauf der Integritätsprüfung . . . . .	47
<b>6</b>	<b>Integrity Measurement: Windows Security Center</b>	<b>49</b>
6.1	Motivation . . . . .	49
6.2	Anforderungen und Vorgehen . . . . .	50
6.3	Ermittlung der AV-Parameter . . . . .	51
6.4	Ermittlung der FW-Parameter . . . . .	52
6.5	Ablauf der Integritätsprüfung . . . . .	54
<b>7</b>	<b>Integrity Measurement: ClamWin Virens Scanner</b>	<b>57</b>
7.1	Motivation . . . . .	57
7.2	Anforderungen und Vorgehen . . . . .	57
7.3	Ermittlung der AV-Parameter . . . . .	58
7.3.1	Installationspfad und Programmversion . . . . .	58
7.3.2	Status des Hintergrundprozesses . . . . .	58
7.3.3	Version der Virendefinitionsdateien . . . . .	60
7.4	Ablauf der Integritätsprüfung . . . . .	61
7.4.1	ClamWin nicht installiert . . . . .	61
7.4.2	Hintergrundprozess läuft nicht . . . . .	62
7.4.3	Hintergrundprozess läuft . . . . .	62
<b>8</b>	<b>Implementierung und Installation</b>	<b>65</b>
8.1	Entwicklungsumgebung . . . . .	65
8.2	Konfiguration der Projekte . . . . .	66
8.3	Spezielle Aspekte der Implementierung . . . . .	67
8.3.1	Threadsicherheit . . . . .	67
8.3.2	WMI-Wrapper . . . . .	69

8.3.3	Logging-API . . . . .	69
8.4	Weitere Abhängigkeiten . . . . .	70
8.5	Installation der TNC-Komponenten . . . . .	70
8.5.1	Notwendigkeit eines Installationstools . . . . .	70
8.5.2	Entwicklungsumgebung . . . . .	70
8.5.3	Anforderungen an das Installationstool . . . . .	71
8.5.4	Ablauf einer Installation . . . . .	71
8.6	Testumgebung . . . . .	73
<b>9</b>	<b>Fazit und Ausblick</b>	<b>75</b>
<b>A</b>	<b>XML-Schema-Beschreibungen</b>	<b>77</b>
A.1	XML-Schema für IMVRegistry-Konfiguration . . . . .	77
A.2	XML-Schema für IMVHostScanner-Konfiguration . . . . .	78
A.3	XML-Schema für Policies . . . . .	80
<b>B</b>	<b>XML-Konfigurationen</b>	<b>81</b>
B.1	XML-Konfiguration für IMVRegistry . . . . .	81
B.2	XML-Konfiguration für IMVHostScanner . . . . .	83
<b>C</b>	<b>Policies für Integrity Measurement Verifiers</b>	<b>85</b>
C.1	Policy für IMVRegistry . . . . .	85
C.2	Policy für IMVHostScanner . . . . .	86
C.3	Policy für IMVClamWin . . . . .	86
C.4	Policies für IMVSecurityCenter . . . . .	87
<b>D</b>	<b>Quellcodes</b>	<b>89</b>
D.1	Quellcode mit Log-Ausgaben . . . . .	89
D.2	Quellcode der TNC-Komponenten . . . . .	90
	<b>Literaturverzeichnis</b>	<b>91</b>
	<b>Abbildungsverzeichnis</b>	<b>93</b>
	<b>Tabellenverzeichnis</b>	<b>95</b>
	<b>Listings</b>	<b>97</b>
	<b>Abkürzungsverzeichnis</b>	<b>99</b>





# 1 Einleitung

Interne Firmennetze sind heutzutage gut nach außen geschützt. Durch den Einsatz mehrerer Paketfilter und demilitarisierter Zonen (DMZ) sowie *Intrusion-Detection-Systeme* (IDS) haben Angreifer aus dem unsicheren Internet kaum noch Chancen einen Angriff auf das interne Netz durchzuführen. Daher gehen die Gefahren vielmehr von dem internen Firmennetz selbst aus. Ob absichtlich oder unabsichtlich sind oft autorisierte Personen für Schäden im lokalen Netzwerk verantwortlich. Aurand beschreibt in [Aur05], wie leicht Sicherheitslücken in LANs bewusst ausgenutzt werden können. Auf der anderen Seite wissen die betroffenen Personen aber oft nicht, dass sie dem internen Netz großen Schaden zufügen, indem sie z.B. Viren oder Würmer von außen einschleppen.

## 1.1 Problemstellung und Ziele

Dies kann geschehen, wenn Mitarbeitern erlaubt wird, ihre Rechner sowohl zu Hause als auch im Firmennetz einzusetzen. Im unsicheren privaten Heimnetzwerk sind die Computer oft ungeschützt Angriffen aus dem Internet ausgesetzt. Der Anwender wird von schädlichen Programmen wie Würmern infiziert und schleust sie später ins vermeintlich sichere Firmennetz ein. Teilweise lässt sich ein Einsatz von Rechnern in mehreren Netzen allerdings nicht verhindern, denkt man z.B. an öffentlichere Netze wie in einer Hochschule. Daher wurde mit TNC<sup>1</sup> eine Architektur entworfen, die es Administratoren erlaubt, Rechner, die sich in ein internes Netz einklinken wollen, einem Integritätscheck zu unterziehen. Dabei kann geprüft werden, ob der Rechner gewisse Standard-Sicherheitsrichtlinien einhält. Erst wenn er diese Prüfung bestanden hat, erlangt er Zugriff auf das interne Netz. Im anderen Fall wird er isoliert oder komplett ausgeschlossen.

---

<sup>1</sup>Trusted Network Connect (siehe auch <https://www.trustedcomputinggroup.org/groups/network/>)

Ziel dieser Arbeit ist die Entwicklung einer API zur client-seitigen Messung und server-seitigen Auswertung von Sicherheitsaspekten, die zur Integritätsbestimmung des Client-Rechners herangezogen werden können. In diesem Zusammenhang sollen auch ausgewählte Merkmale gemessen und bewertet werden. Mit Hilfe der API werden dazu exemplarische *Integrity Measurement Collectors* (IMCs) und *Integrity Measurement Verifiers* (IMVs) entwickelt.

## 1.2 Aufbau der Arbeit

Kapitel 2 beschreibt die der Arbeit zu Grunde liegende TNC-Architektur. Dabei wird es zuerst einen Gesamtüberblick über die Architektur geben. Im Anschluss wird auf die einzelnen Bestandteile genauer eingegangen.

In Kapitel 3 wird die allgemeine API zum Erstellen von IMCs und IMVs, die dem Messen und Auswerten von Sicherheitsaspekten dienen, vorgestellt. Nach den Anforderungen an die API wird eine dazu passende Softwarearchitektur beschrieben.

Kapitel 4 stellt das erste exemplarische IMC/IMV-Paar vor, das unter Verwendung der allgemeinen API entstanden ist. Dabei geht es um das Auslesen von sicherheitsrelevanten Registrierungseinträgen auf Windows-Clients.

In Kapitel 5 wird ein IMC beschrieben, mit dessen Hilfe Port-Scans auf Client-Seite durchgeführt werden können. Ein entsprechender IMV sorgt für die Einhaltung der Sicherheitsrichtlinien.

Wichtige Sicherheitsmerkmale sind die Firewall- und Anti-Virus-Parameter eines Rechners. In Kapitel 6 wird eine Komponente vorgestellt, die auf Client-Seite die Daten des *Windows Security-Centers* ausliest und damit das Vorhandensein und die Aktualität beliebiger Firewall- und Anti-Virus-Software feststellen kann.

In Kapitel 7 wird ein spezielles IMC/IMV-Paar für den Open-Source Virens Scanner *ClamWin* vorgestellt. Dieser unterstützt keine Integration in das *SecurityCenter* von Windows.

Die Vorgehensweise bei der Implementierung der Komponenten wird in Kapitel 9 beschrieben. Dabei geht es um die verwendete Entwicklungsumgebung und die Konfiguration der Projekte. Weiterhin wird das Vorgehen bei der Installation auf einem Testsystem erläutert.

Die Arbeit schließt mit einem Fazit und Ausblick in Kapitel 9 ab. Dabei soll aufgezeigt werden, welche Erweiterungsmöglichkeiten die entstandene Architektur bietet.

## 1.3 Über diese Masterarbeit

Diese Masterarbeit entstand im Rahmen eines hochschulinternen Projekts mit dem Namen *TNC@FHH*. Dabei war eine Kooperation mit verschiedenen Mitgliedern der *Trusted Computing Group* (TCG) möglich. Der Fachbereich Informatik wurde zudem als Mitglied der TCG und der TNC Subgroup aufgenommen.

In diesem Projekt entstand parallel eine zweite Arbeit von Martin Schmiedel [Sch06], die die Entwicklung eines TNC-Clients und -Servers als Ziel hat. Diese Komponenten sind hauptverantwortlich für den Austausch der gemessenen Sicherheitsaspekte zwischen Client und Server. Daher war eine enge Zusammenarbeit notwendig, um die entstandenen Teilkomponenten auf ihre Funktionsfähigkeit hin zu testen.

## 1.4 Typographische Konventionen

In dieser Masterarbeit werden folgende Schreibweisen verwendet:

<i>Kursiv:</i>	Wichtige Begriffe und allgemeine Hervorhebungen werden in kursiver Schrift dargestellt.
Typewriter:	Die Schriftart Typewriter (Courier) wird für Klassen- und Methodenbezeichnungen sowie Befehle verwendet.

Quellcode und XML-Beschreibungen werden in Listings ausgegeben:

```
class Konto
{
    private:
        string kontoNr;
        double saldo;

    public:
        void setKontoNr(string kontoNr)
        {
            this -> kontoNr = kontoNr;
        }

        [...]
}
```

Listing 1.1: Beispiel-Quellcode

## 2 TNC-Architektur

In diesem Kapitel wird die der Arbeit zu Grunde liegende TNC-Architektur mit ihren Zielen und Aufgaben vorgestellt. Zu Beginn wird es einen groben Überblick über die gesamte Architektur geben. Im Folgenden wird auf die Komponenten, die im Zusammenhang mit dieser Arbeit eine wichtige Rolle spielen, gesondert eingegangen.

### 2.1 Trusted Computing Group und TNC

Bei der *Trusted Computing Group* handelt es sich um eine gemeinnützige Organisation, die versucht, offene Standards im Bereich der Computer- und Netzwerksicherheit für zahlreiche Plattformen und Geräte zu definieren. Ziel ist es, Umgebungen zu entwickeln, die das Arbeiten mit Rechnern und Netzwerken sicherer machen sollen, ohne Einschränkungen im Bereich der Funktionalität und Privatsphäre in Kauf nehmen zu müssen. Anwender sollen sicher sein können, dass ihre vertraulichen Daten vor Angriffen von außen geschützt sind.

Die *Trusted Network Connect Sub Group* (TNC-SG) arbeitet an einer offenen Architektur, die es Netzwerk-Administratoren ermöglichen soll, Endgeräte wie PCs oder Laptops vor ihrer Einbindung in ein Netzwerk einer Integritätsprüfung zu unterziehen. Für diese Integritätsprüfung werden Sicherheitsaspekte des Endgerätes client-seitig gemessen bzw. ausgelesen und server-seitig ausgewertet. Diese Sicherheitsaspekte können sich auf alle möglichen Arten von Systemkomponenten wie Hardware, Firmware und Software beziehen. Denkbar wären z.B. Anti-Virus-Parameter, Firewall-Status, Software-Versionen und Systemupdates. Erfüllt ein Endgerät wichtige Anforderungen nicht (hat er z.B. keine Firewall installiert oder sind die Virendefinitionen der Anti-Virus-Software veraltet), schlägt der Integritätstest fehl und das Endgerät wird nicht in das lokale Netzwerk aufgenommen. Dies kann sowohl einen kompletten Ausschluss als auch die Isolierung in einen geschützten Bereich, in dem Updates durchgeführt werden können, bedeuten.

## 2.2 Allgemeiner Aufbau des Architekturmodells

Bei der in Abbildung 2.1 dargestellten TNC-Architektur handelt es sich um ein Schichtenmodell, dem eine Vielzahl von Entitäten, Komponenten und Schnittstellen zu Grunde liegt. Im Folgenden wird auf die einzelnen Bestandteile näher eingegangen.

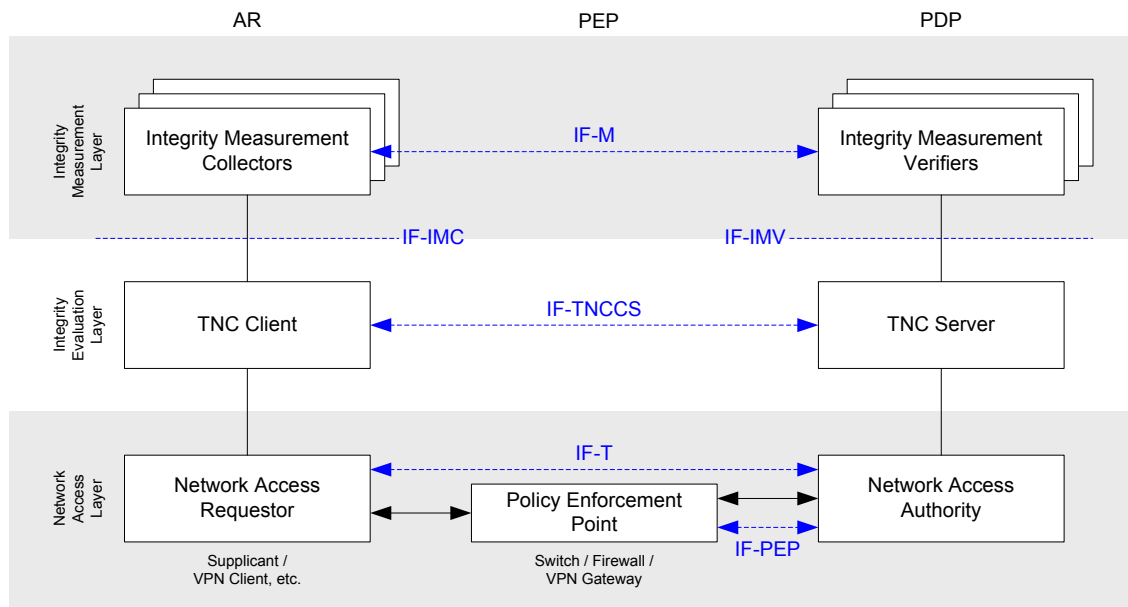


Abbildung 2.1: TNC-Architektur (vgl. [TCG06a])

Das Architekturmodell besteht aus drei Spalten. Diese repräsentieren die Entitäten der TNC-Architektur. Dabei handelt es sich um aktive Netzwerkkomponenten wie Client-PC, Server und Switch. Eine Entität kann dabei aus mehreren Komponenten bestehen, die in der Grafik durch Rechtecke repräsentiert werden. Jede Komponente nimmt in diesem Zusammenhang eine ganz spezielle Aufgabe wahr. Darauf wird in den Abschnitten 2.3.1 bis 2.3.3 gesondert eingegangen.

Die Entitäten der TNC-Architektur im Überblick:

**Access Requestor (AR)** Der Access Requestor ist der Client (z.B. PC oder Laptop), der Zugang zum geschützten internen Netzwerk erlangen möchte.

**Policy Decision Point (PDP)** Beim Policy Decision Point handelt es sich um einen Server, der anhand von Regeln (Policies) entscheiden muss, ob dem Client der Zugang zum Netz gewährt wird.

**Policy Enforcement Point (PEP)** Der Policy Enforcement Point (i.d.R. ein Switch) steuert den Zugriff auf das interne Netz und setzt die Entscheidung vom PDP um; d.h. er nimmt den Client abhängig vom Ergebnis der Integritätsprüfung in das interne Netz auf oder isoliert ihn.

Drei horizontale Schichten gruppieren die Komponenten der Entitäten auf logischer Ebene. Auf diesen Ebenen findet die Kommunikation zwischen den Komponenten der TNC-Architektur statt. Standardisierte Schnittstellen, in der Abbildung durch gestrichelte Linien dargestellt, legen die Art und Weise der Kommunikation fest.

Die logischen Schichten im Einzelnen:

**Network Access Layer** Auf dieser Ebene findet der physikalische Datenaustausch statt. Alle drei Entitäten sind an diesem Datenaustausch beteiligt. Verwendete Technologien können dabei u.a. VPN und 802.1x sein.

**Integrity Evaluation Layer** Die Komponenten in dieser Schicht müssen mit Hilfe von definierten Regeln beurteilen, ob die Integrität des Clients insofern gewährleistet ist, dass er ohne Bedenken in das interne Netz aufgenommen werden darf.

**Integrity Measurement Layer** Diese Ebene besteht aus einer beliebigen Anzahl von Komponenten-Paaren, die Sicherheitsaspekte client-seitig auslesen bzw. messen und server-seitig auswerten.

## 2.3 Komponenten im Architekturmodell

### 2.3.1 Integrity Measurement Collector (IMC)

Diese Software-Komponente ist Bestandteil des Access Requestors (AR) und liegt im *Integrity Measurement Layer*. Sie ist verantwortlich für das Messen und Auslesen von Sicherheitsaspekten. Dabei kann es sich z.B. um Anti-Virus-Parameter, den Status der Personal Firewall, Softwareversionen oder das Vorhandensein von Betriebssystem-Updates handeln. Ein IMC ist in der Regel eine herstellerspezifische Komponente für genau ein Software-Produkt auf dem Client. Daher ist es möglich, mehrere IMCs parallel auf dem Client zu installieren, die dynamisch geladen und aufgerufen werden können.

### 2.3.2 Integrity Measurement Verifier (IMV)

Der Integrity Measurement Verifier (IMV) ist eine Komponente des Policy Decision Points (PDP) und liegt ebenso im *Integrity Measurement Layer*. Zu jedem IMC auf Clientseite muss es ein entsprechendes Gegenstück auf dem PDP geben. Ein IMV nimmt die Nachrichten des IMCs entgegen und wertet den gemessenen Sicherheitsaspekt mit Hilfe von festgelegten Policies aus. Im Folgenden kann der IMV eine Teilaussage zur Integrität des Clients abgeben.

### 2.3.3 Weitere Komponenten

Die weiteren Komponenten der TNC-Architektur liegen in den Schichten *Integrity Evaluation Layer* und *Network Access Layer*. Diese sind nicht Bestandteil dieser Arbeit. Daher wird es an dieser Stelle nur einen kurzen Überblick über die Funktion dieser Komponenten geben. Weitere Informationen zu diesem Thema finden sich in [Sch06] wieder.

**TNC Client (TNCC)** Der TNCC ist für das Laden der IMCs und das Versenden von Nachrichten vom IMC zum IMV verantwortlich. Des Weiteren leitet er Nachrichten von den IMVs an die betroffenen IMCs weiter. Die Übertragung der Nachrichten muss gesichert erfolgen. Der TNCC darf die Nachrichten nicht einsehen oder verändern.

**Network Access Requestor (NAR)** Der NAR ist verantwortlich für den Aufbau der Netzwerkverbindung zum Server. Eine Implementierung kann beispielsweise in einem Supplicant für 802.1x erfolgen.

**Network Access Authority (NAA)** Der NAA muss entscheiden, ob einem Client der Zugriff zum internen Netz gewährt wird. Zu diesem Zweck kann und sollte er einen TNC-Server befragen, welcher den Integritätszustand des Clients mit festen Vorgaben (Policies) vergleichen kann.

**TNC Server (TNCS)** Der TNCS kontrolliert den Nachrichtenfluss zwischen IMCs und IMVs. Weiterhin nimmt er Empfehlungen und Ergebnisse der IMVs entgegen, um eine Gesamtaussage zur Integrität des Clients anstellen zu können. Diese Gesamtsentscheidung wird dem NAA mitgeteilt, welcher selbige umsetzen muss.



## 2.4 Schnittstellen im Architekturmodell

Die Spezifikationen zu den Schnittstellen IF-IMC [TCG05b] und IF-IMV [TCG05c] wurden Anfang Mai 2005 von der TCG veröffentlicht. Mittlerweise gibt es aktualisierte Versionen dieser Spezifikationen (vgl. [TCG06b] und [TCG06c]). Erweiterungen der Schnittstellen wurden in diesem Zusammenhang aber noch nicht umgesetzt. Die Veröffentlichung der Spezifikationen zu den anderen Schnittstellen, die im Abschnitt 2.4.4 vorgestellt werden, erfolgte erstmalig Anfang Mai 2006.

### 2.4.1 Schnittstelle zwischen IMC und TNC Client (IF-IMC)

Die Schnittstelle IF-IMC ist die Verbindung zwischen IMCs und TNC Client. Über sie wird der Nachrichtenaustausch zwischen IMCs und IMVs ermöglicht. Der TNCC sammelt innerhalb eines TNC-Handshakes über mehrere Runden (auch *Batches* genannt) die zu versendenden Nachrichten und schickt sie zum TNC Server. Im Vorfeld muss er sämtliche registrierte IMCs laden, initialisieren und nach einem Verbindungsaufbau zum Server einmalig zum Versenden einer Nachricht auffordern. Letztes erfolgt über einen Funktionsaufruf beim IMC, der den Beginn des TNC-Handshakes kennzeichnet. Zu diesem Zweck wurde innerhalb der TNC-Architektur eine genaue API spezifiziert, in der festgelegt ist, welche Funktionen IMCs und der TNC Client bereitstellen müssen [TCG06b]. Somit wird eine standardisierte Kommunikation zwischen den beiden Komponenten ermöglicht. Dies wiederum bedeutet, dass TNC Client und IMCs nicht zwingend von genau einem Hersteller entwickelt werden müssen. Die wichtige Anforderung der Interoperabilität ist damit prinzipiell gewährleistet. Dennoch schließt die Architektur die Erweiterung der API um herstellerspezifische Bestandteile nicht aus.

### 2.4.2 Schnittstelle zwischen IMV und TNC Server (IF-IMV)

Durch IF-IMC ist die Schnittstelle zwischen IMVs und TNC Server beschrieben. Über sie werden hauptsächlich die Nachrichten der IMCs geleitet, die zur Integritätsbestimmung durch die IMVs ausgewertet werden müssen, sowie die Antwort-Nachrichten der IMVs zurück zum Client. Des Weiteren wird dem TNC Server über diese Verbindung jeweils das Ergebnis einer Prüfung durch einen IMV sowie eine Empfehlung, ob eine Aufnahme ins interne Netz unbedenkbar ist, mitgeteilt. Wurde nach der maximalen Anzahl an

Runden im TNC-Handshake noch keine Empfehlung eines oder mehrerer IMVs ausgesprochen – möglicherweise liegen noch nicht genug oder gar keine Informationen der IMCs vor – fordert der TNC Server die jeweiligen IMVs zur Abgabe einer Bewertung auf. In den meisten Fällen werden sie dann die Aussage treffen, dass sie keine Bewertung anstellen können. Wie der TNC Server diese neutrale Aussage auffasst, hängt von den Policy-Einstellungen des Servers ab. Die Spezifikation der gesamten API ist in [TCG06c] beschrieben.

### 2.4.3 Schnittstelle zwischen IMC und IMV (IF-M)

Die Schnittstelle IF-M ist herstellerspezifisch und durch TNC nicht beschrieben. Sie dient dem logischen Nachrichtenaustausch zwischen IMCs und IMVs. Physikalisch werden sie über das IF-T Interface transportiert. Damit TNC Client und TNC Server die Nachrichten identifizieren können, werden sie durch herstellereigene Nachrichten-Typen gekennzeichnet. Zu diesem Zweck erhält jeder Hersteller von IMC/IMV-Paaren eine eindeutige Hersteller-ID. Diese Hersteller-ID in Verbindung mit einem frei wählbaren *Subtype* ergibt dann eine eindeutige Nachrichten-Kennung. Anhand dieser Kennung wissen der TNCC und TNCS, welchem IMC bzw. IMV sie die Nachrichten zustellen müssen. Zu diesem Zweck müssen sich die IMCs und IMVs für bestimmte Nachrichten-Typen, an denen sie interessiert sind, registrieren.

### 2.4.4 Weitere Schnittstellen

Die weiteren Schnittstellen, die durch die TNC-Architektur beschrieben sind, sind kein Bestandteil dieser Arbeit und werden daher nur kurz aufgelistet. Eine genauere Beschreibung dieser Schnittstellen ist in der Master-Arbeit von Martin Schmiedel [Sch06] zu finden.

Die weiteren Schnittstellen in der TNC-Architektur:

- IF-TNCCS
- IF-T
- IF-PEP

## 2.5 Eingesetzte Protokolle und Technologien

Zum Zeitpunkt der Integritätsprüfung des Clients liegt noch keine aktive LAN-Verbindung zum Switch oder Access Point vor. Durch die Prüfung soll ja erst festgestellt werden, ob dem Client die Aufnahme ins interne Netz gestattet werden darf. Daher stellt TCP/IP keine Möglichkeit dar, Pakete zwischen AR und PDP auszutauschen. Beim IEEE 802.1x Standard handelt es sich dagegen um ein Layer-2-basierendes Protokoll, welches auf höhere Protokolle wie eben IP nicht angewiesen ist. Über 802.1x können beliebige Authentifizierungs-Informationen zwischen Client und Server ausgetauscht werden. Das Protokoll unterscheidet dabei zwischen drei verschiedenen Entitäten:

- Supplicant (Client)
- Authenticator (Switch oder Access Point)
- Authentication Server (i.d.R. RADIUS)

Von der Rollenverteilung her spiegeln sich an dieser Stelle die drei Entitäten der TNC-Architektur wider (AR, PEP und PDP). Daher bietet sich die Verwendung des 802.1x Standards in Verbindung mit dem EAP-Protokoll<sup>1</sup> für die Übertragung von Daten zwischen TNC Client und TNC Server an. Als NAA (siehe Abschnitt 2.3.3) kann innerhalb der TNC-Architektur ebenfalls ein RADIUS-Server eingesetzt werden. Weitere Informationen zum Thema EAP über 802.1x und RADIUS finden sich in [Aur05] und [Sch06] wieder.

## 2.6 Ablauf einer Integritätsprüfung

Der Ablauf einer Integritätsprüfung innerhalb der TNC-Architektur soll durch das Sequenzdiagramm in Abbildung 2.2 verdeutlicht werden. In diesem Beispiel findet die Prüfung unter Verwendung von zwei IMC/IMV-Paaren und eines TNC-Handshakes über vier Runden statt.

Wie zu erkennen ist, werden die IMCs und IMVs vor Beginn eines TNC-Handshakes initialisiert (Schritte 5 bis 8). Dies dient dem Zweck des Versionsabgleichs. Möglicherweise unterstützt ein IMC bereits mehrere Versionen der IF-IMC Schnittstelle, der TNC Client beherrscht aber nicht alle davon (oder umgekehrt). Daher müssen sich beide Parteien vor

---

<sup>1</sup>Extensible Authentication Protocol (RFC 2284)

Beginn eines Handshakes auf eine gemeinsame Version einigen. Das gleiche Verfahren findet auf Server-Seite zwischen TNCS und IMVs statt.

Stellt der TNC Client eine Verbindung zum Server her, beginnt er einen neuen Handshake. Dies teilt er seinen IMCs mit, die daraufhin ihre erste Nachricht für ihren IMV erstellen (Schritte 11 und 13). Innerhalb der ersten Runde sammelt der TNC Client erst alle Nachrichten sämtlicher IMCs ein, bevor er alle gebündelt zum Server schickt. Vor dem Versenden (Schritt 15) teilt er den IMCs mit, dass die aktuelle Runde beendet wird (nicht im Diagramm enthalten). Zu diesem Zeitpunkt haben die IMCs sogar nochmals die Möglichkeit eine Nachricht zu erzeugen, die noch innerhalb des ersten *Batches* versendet werden würde. Die Implementierung dieser Funktionalität auf IMC- und IMV-Seite ist allerdings optional.

Auf Server-Seite nimmt der TNCS die Nachrichten entgegen und verteilt sie gemäß den Registrierungen der Nachrichten-Typen (siehe Abschnitt 2.4.3) an die IMVs (Schritte 17 und 19). Diese erzeugen ihrerseits Antwort-Nachrichten, die vom TNCS eingesammelt und zusammen zurück zum TNC Client geschickt werden (Schritt 21). Damit ist die zweite Runde des TNC-Handshakes beendet und der TNCC beginnt mit der dritten. Das Verfahren der Nachrichtenzustellung wiederholt sich nun auf Client-Seite (Schritte 23 und 25). Wiederum erzeugen die IMCs Nachrichten, die zu ihren IMVs geschickt werden sollen. Nach der Verteilung dieser Nachrichten auf Server-Seite können die IMVs eine Bewertung des Integritätszustands des Clients abgeben (Schritte 29 und 31).

Mit Hilfe dieser Ergebnisse muss der TNCS nun eine Gesamtentscheidung treffen und bestimmen, ob der AR ins interne Netz aufgenommen werden soll. Diese Entscheidung teilt er dem PEP mit, welcher den entsprechenden Port nun freigibt, sperrt oder isoliert. Weiterhin wird der TNC Client über das Ergebnis informiert und die Verbindung abgebaut. Der TNC-Handshake ist damit beendet. Den genauen Ablauf der Kommunikation zwischen AR, PEP und PDP sowie den Aufbau der auf IF-TNCCS basierenden Nachrichten, die zwischen TNCC und TNCS ausgetauscht werden und die Daten der IMCs und IMVs beinhalten, beschreibt Martin Schmiedel in seiner Masterarbeit [Sch06].

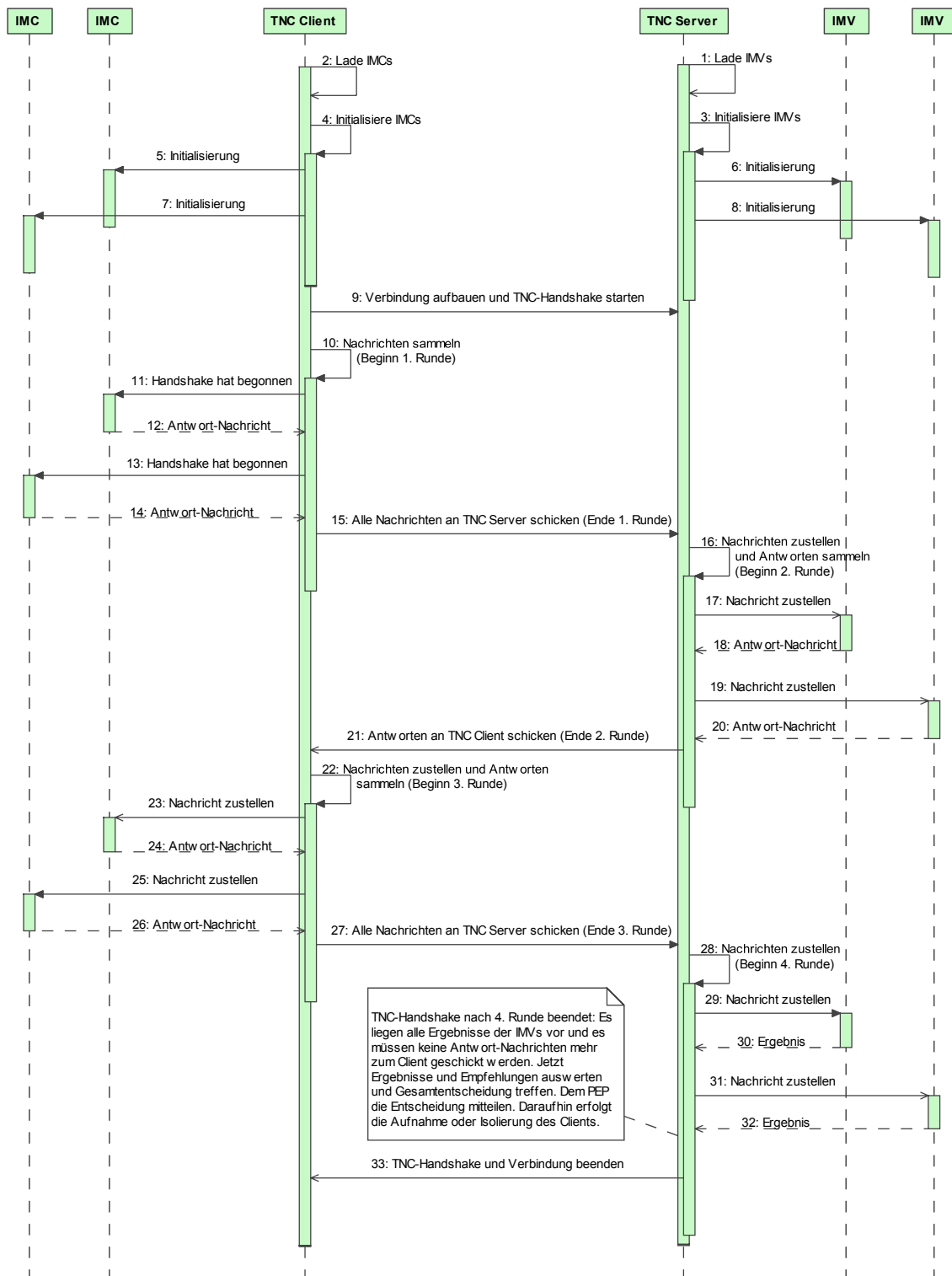


Abbildung 2.2: Ablauf einer Integritätsprüfung mit zwei IMC/IMV-Paaren



## 3 Allgemeine API für IMCs und IMVs

In diesem Kapitel soll die allgemeine API für die Entwicklung von IMCs und IMVs vorgestellt werden. Dazu wird im Vorfeld auf die funktionalen und nicht-funktionalen Anforderungen an eine API eingegangen. Im Anschluss wird die Softwarearchitektur mit ihren Bausteinen und Basisfunktionalitäten beschrieben.

### 3.1 Anforderungen an die API

#### 3.1.1 Funktionale Anforderungen

Folgende funktionale Anforderungen sollten von einer API für die Entwicklung von IMCs und IMVs erfüllt werden:

**Erweiterbarkeit und Flexibilität** Es muss möglich sein, in wenigen Schritten und ohne großen Aufwand weitere IMC/IMV-Paare für neue Sicherheitsaspekte, die zu einer Integritätsprüfung herangezogen werden können, zu entwickeln. Dabei sollte sich der Entwickler nicht mehr mit der Implementierung von Basisfunktionalität beschäftigen müssen; d.h. der Nachrichtenaustausch und die Schnittstelle zum TNC Client bzw. Server sollten bereits durch eine abstrakte Basisklasse implementiert sein. Des Weiteren wäre auch die Bereitstellung zusätzlicher Standardbibliotheken für Logging, Dateizugriffe und XML-Bearbeitung sinnvoll. Im einfachsten Fall sollte ein Entwickler eine abstrakte Klasse erweitern können und diese lediglich um neue Logik erweitern müssen.

**Threadsicherheit** Es muss davon ausgegangen werden, dass mehrere Clients gleichzeitig einer Integritätsprüfung unterzogen werden. Damit muss es auf dem PDP meh-

rere Instanzen derselben IMVs geben können. Die Zustände der jeweiligen Verbindungen müssen separat gespeichert werden können. Auf Clientseite wird es dagegen immer nur genau eine Instanz eines IMCs geben. Threadsicherheit ist in diesem Fall also nicht zwingend erforderlich.

#### 3.1.2 Nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen an die Softwarearchitektur der API lassen sich wie folgt zusammenfassen:

**Open-Source** Da das Projekt keine kommerziellen Ziele verfolgt, soll der entwickelte Source-Code frei zugänglich sein. Um die Software unter einer entsprechenden Open-Source-Lizenz veröffentlichen zu können, dürfen innerhalb des Projekts keine kommerziellen Bibliotheken verwendet werden.

**Windows als Client** Da Microsoft Windows das am häufigsten eingesetzte Betriebssystem für Client-PCs ist, sollen der TNC Client sowie die IMCs vorrangig für einen Einsatz unter Windows XP entwickelt werden.

**Linux als Server** Innerhalb des Fachbereichs Informatik werden linuxbasierte Server für die Bereitstellung zentraler Dienste eingesetzt. Aus diesem Grund sollen der TNC Server und die IMVs für Linux entwickelt werden. Als Linux-Distribution dient SuSE 9.3.

**Verwendung von C++** Da bis zum jetzigen Zeitpunkt lediglich ein C-Binding für die Implementierung der TNC-Architektur veröffentlicht wurde, ist eine Bereitstellung der API als C bzw. C++ Implementierung notwendig.

**Plattformunabhängigkeit und Wiederverwendbarkeit** Durch die Verwendung von ANSI-C bzw. ANSI-C++ ist eine Portierung des Quellcodes von Windows auf andere Systeme wie z.B. Linux möglich. Damit kann ein Großteil des Quellcodes als gemeinsame Sourcenbasis sowohl unter Windows für die IMC-API als auch unter Linux für die IMV-API verwendet werden.

**Übersichtlichkeit** Die Softwarearchitektur der API sollte übersichtlich und gut verständlich sein. Dies erleichtert die Wartbarkeit und den Einstieg neuer Entwickler.



**Gute Performance** Die allgemeine API für IMCs und IMVs sollte schlank und performant implementiert sein. Lange Wartezeiten beim Handshake zwischen TNC Client und TNC Server aufgrund von Performance-Problemen bei IMCs und IMVs verzögern die Aufnahme des Clientrechners in das interne Netz und könnten beim Anwender durchaus eine gewisse Verärgerung hervorrufen.

## 3.2 Softwarearchitektur der API

Die oben genannten Anforderungen an die API erfordern eine gut strukturierte und übersichtliche Softwarearchitektur. Eine Unterteilung in einzelne Building-Blocks<sup>1</sup> zwecks Wiederverwendbarkeit und Portierbarkeit erscheint sinnvoll.

Die Softwarearchitektur für IMCs auf Windows-Systemen unterscheidet sich von der Architektur für IMVs auf linuxbasierten Systemen nur geringfügig. Die Verwendung der Registrierung für das Ablegen von Systemparametern unter Windows erfordert zwei zusätzliche Softwarebausteine, die unter Linux nicht erforderlich sind. Abbildung 3.1 veranschaulicht die Softwarearchitektur für IMCs unter Windows.

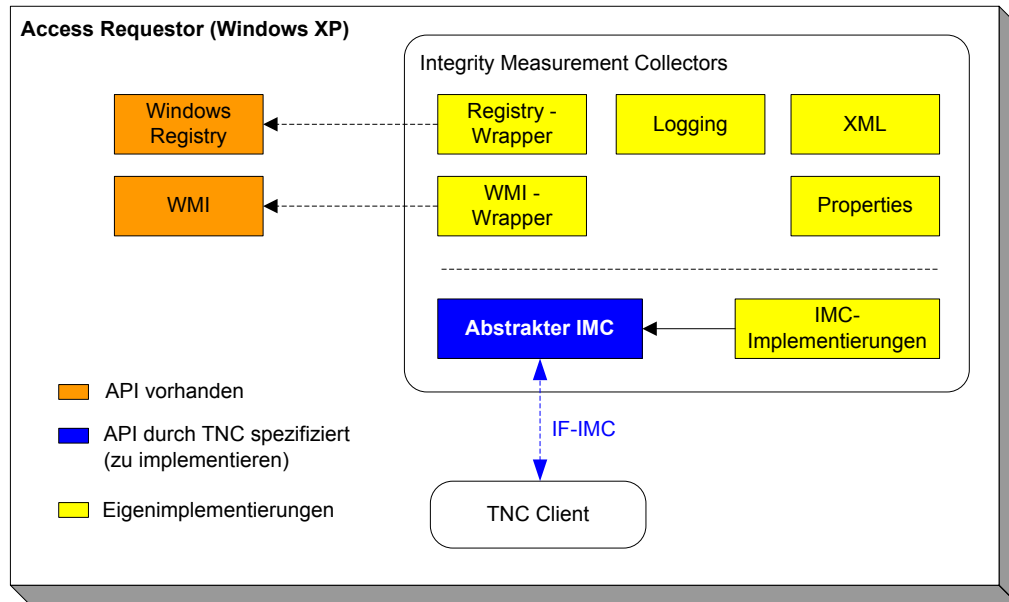


Abbildung 3.1: Architekturübersicht IMC-API

<sup>1</sup>Bei einem Building-Block handelt es sich um eine technische Architekturkomponente.

Wie im vorherigen Kapitel bereits beschrieben wurde, ist die Schnittstelle zwischen IMC und TNC Client durch die TNC-Architektur genau spezifiziert. Da diese Schnittstelle von jeder konkreten IMC-Implementierung benötigt wird, ist an dieser Stelle eine abstrakte Basisimplementierung sinnvoll (Abstrakter IMC). Konkrete IMC-Implementierungen brauchen damit nur noch die Basisimplementierung erweitern und können sich somit voll auf ihre eigene Fachlichkeit konzentrieren. Weitere Softwarebausteine wie Logging, XML und Properties stellen darüber hinaus Standardfunktionen bereit, die den Entwickler von IMCs bei seiner Arbeit unterstützen können. Auf die einzelnen Softwarekomponenten wird im Folgenden noch genauer eingegangen. Durch diese Vorgehensweise wird die Anforderung *Erweiterbarkeit und Flexibilität* umgesetzt.

Die Softwarearchitektur der API für IMVs unter Linux unterscheidet sich nur unwesentlich von der IMC-API. Windows-spezifische Komponenten entfallen. Dafür gibt es einen neuen Building-Block *Validierung*, der den Entwickler von IMVs beim Soll-Ist-Vergleich unter Berücksichtigung einer Policy unterstützen soll. Ein Großteil des Quellcodes kann von der IMC-Implementierung übernommen werden. Lediglich kleine Anpassungen an der abstrakten Basisimplementierung des IMVs sind aufgrund von Abweichungen in der Schnittstelle IF-IMV notwendig. Abbildung 3.2 zeigt die Softwarearchitektur der IMV-API.

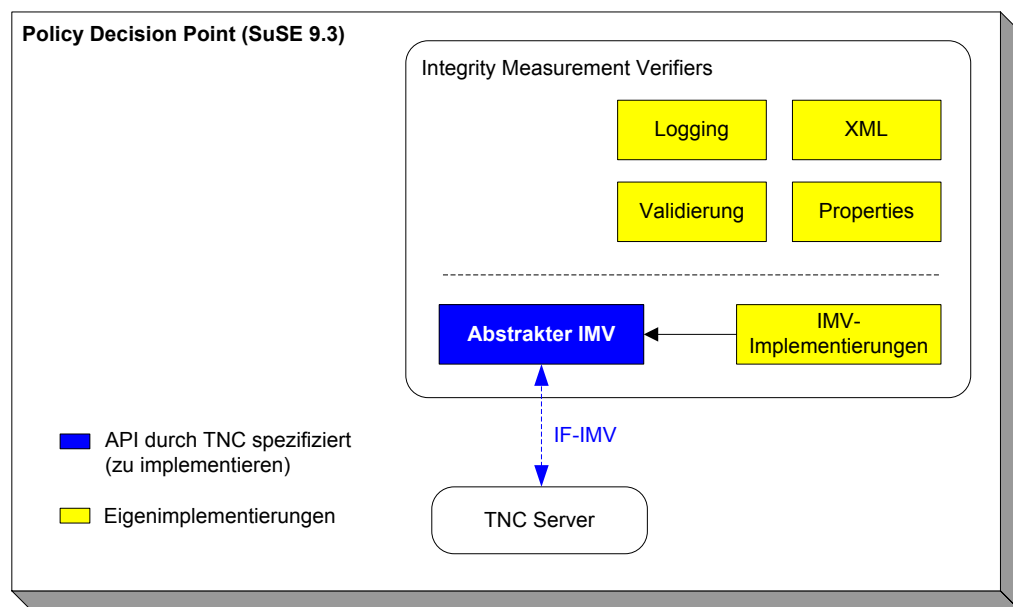


Abbildung 3.2: Architekturübersicht IMV-API

### 3.2.1 Abstrakter IMC

Der abstrakte IMC setzt sich aus zwei Bestandteilen zusammen (siehe Abbildung 3.3). Die Klasse `AbstractIMC` stellt eine abstrakte Basisklasse dar, die von einer konkreten IMC-Klasse erweitert werden muss. Neben statischen Funktionen zum Zusammensetzen und Parsen von Nachrichten-Typen verwaltet die Klasse den aktuellen *Connection State* der Verbindung. Die Methoden `begin()` und `receive()` sind abstrakt und müssen von der konkreten Klasse implementiert werden. Soll die erste Nachricht vom IMC zum IMV geschickt werden, wird `begin()` aufgerufen. Sobald später eine Nachricht vom IMV ein- geht, wird diese über die Methode `receive()` an den konkreten IMC geleitet.

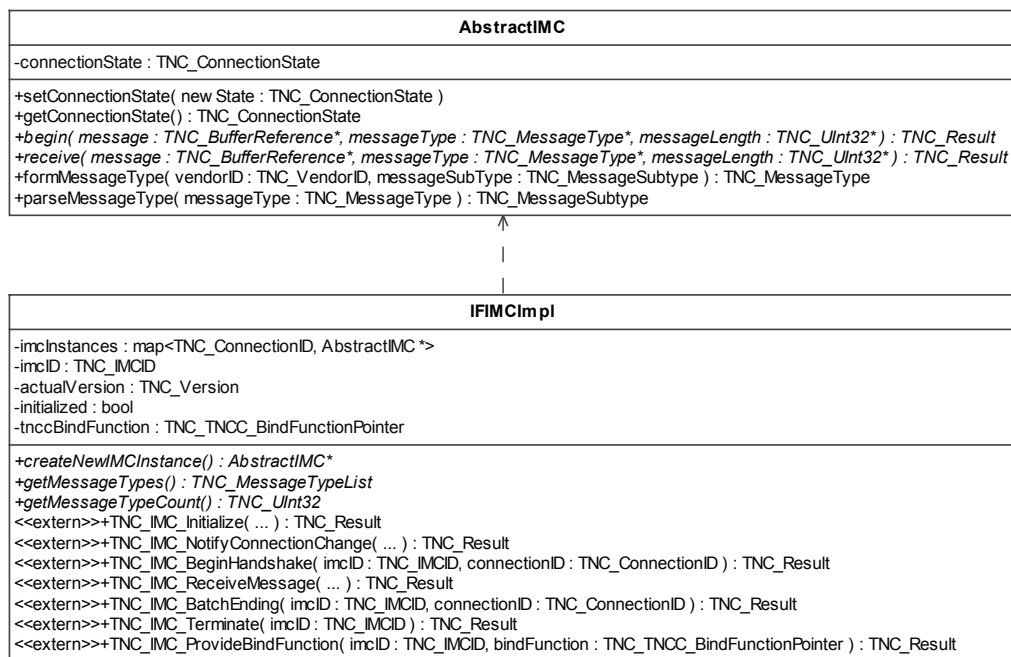


Abbildung 3.3: Abstrakter IMC

Die Klasse `IFIMCImpl` implementiert die Schnittstelle IF-IMC. Bei den durch `<extern>` gekennzeichneten Methoden handelt es sich um die Funktionen, die laut TNC IF-IMC Spezifikation [TCG06b] jeder IMC implementieren muss und beim *Windows DLL Platform Binding* nach außen sichtbar sind. Diese Methoden können nach dem Laden der DLL vom TNC Client aufgerufen werden. Die Klasse ist damit verantwortlich für das Entgegennehmen und Weiterreichen von Nachrichten vom TNC Client zum IMC bzw. umgekehrt. Weiterhin erzeugt sie im Falle eines neuen TNC-Handshakes auch eine neue Instanz der konkreten IMC-Klasse und legt diese innerhalb einer statischen Map ab. Die Instanz kann jederzeit anhand der eindeutigen vom TNC Client vergebenen Connection-

ID identifiziert werden. Sollte eine Veränderung des *Connection States* eintreten, wird der IMC darüber informiert. Da ein TNC Client in der Regel immer nur mit einem TNC Server gleichzeitig kommuniziert, wird es im Normalfall auch immer nur eine Instanz eines IMCs geben.

Drei abstrakte Methoden – in diesem Fall handelt es sich um reine Funktions-Deklarationen – müssen von einer konkreten IMC-Klasse zusätzlich implementiert werden. Während des Linker-Vorgangs werden die Implementierungen zu den Funktions-Deklarationen vom Linker eingebunden. Die Methoden haben folgende Aufgabe:

**createNewIMCInstance()** Diese Funktion gibt einen Zeiger auf eine neue Instanz der eigenen konkreten IMC-Klasse zurück. Innerhalb der Klasse IFIMCImpl wird anschließend nur auf die Schnittstelle der Oberklasse AbstractIMC zugegriffen. Bei diesem Verfahren handelt es sich grob um die Umsetzung des Factory-Patterns. Allerdings muss die abstrakte Implementierung noch nicht einmal den Namen der konkreten IMC-Klasse wissen, um eine neue Instanz erzeugen zu können. Dies wäre in Programmiersprache Java so nicht umsetzbar. Durch dieses Vorgehen ist eine leichte Erweiterung der Basisimplementierung möglich, ohne das z.B. Klassennamen in einer Properties-Datei abgelegt werden müssen.

**getMessageTypes()** Diese Methode liefert alle Nachrichten-Typen zurück, an deren Empfang der IMC interessiert ist. Die Klasse IFIMCImpl ruft diese Funktion unmittelbar nach der Initialisierung des IMCs auf. Da eine Instanz der konkreten IMC-Klasse erst beim Start eines neuen TNC-Handshakes erzeugt wird, kann diese Funktion also keine Instanz-Methode der konkreten IMC-Klasse sein. Auch eine statische Methode ist nicht möglich, da die Basis-Implementierung den Namen der konkreten Klasse nicht kennt. Dieses Problem lässt sich daher genau wie bei `createNewIMCInstance()` sehr elegant über das Linken lösen.

**getMessageTypeCount()** Diese Funktion gibt die Anzahl der Nachrichten-Typen zurück. Die Umsetzung erfolgt genauso wie bei der Methode `getMessageTypes()`.

#### 3.2.2 Abstrakter IMV

Der Aufbau des abstrakten IMVs entspricht fast ganz dem des abstrakten IMCs (siehe Abbildung 3.4). Eine konkrete IMV-Implementierung muss die Basisklasse Abstract-

IMV erweitern. Neben dem *Connection State* werden hier allerdings noch der Zustand der Validierung (*validationFinished*) und das Ergebnis der Validierung sowie eine Empfehlung für den TNC Server, ob der Zugang zum Netz gewährt werden darf, gespeichert. Über entsprechende Funktionen kann die Klasse *IFIMVImpl* das Ergebnis und die Empfehlung abholen und an den TNC Server weitergeben.

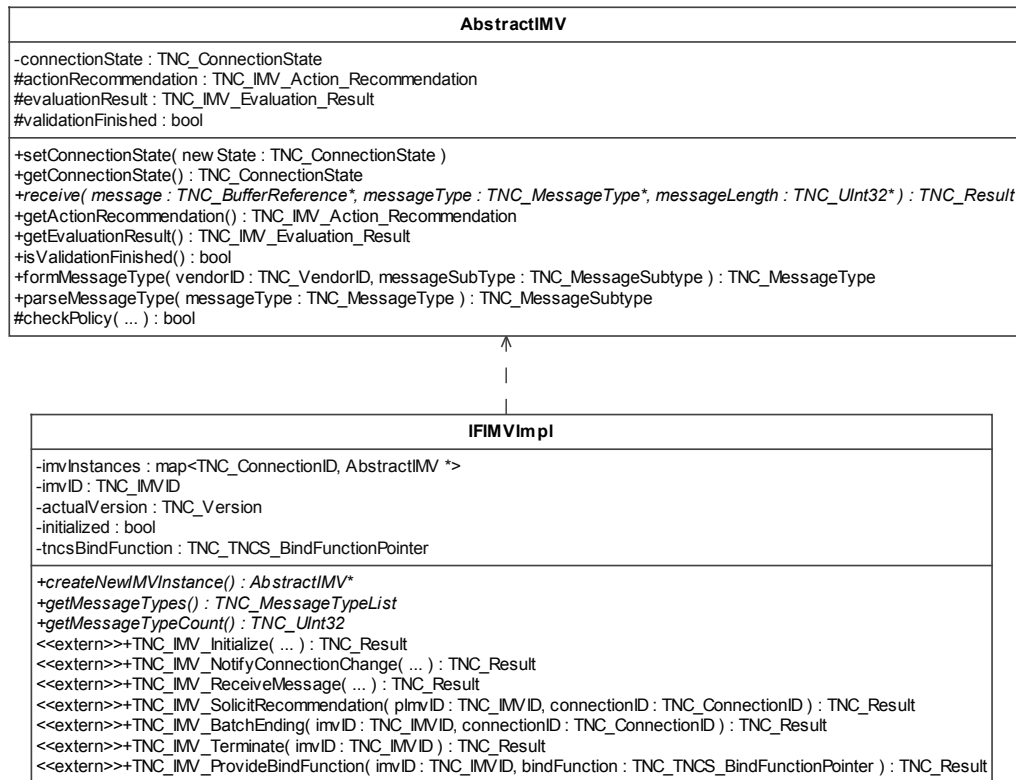


Abbildung 3.4: Abstrakter IMV

Anders als beim abstrakten IMC muss eine konkrete Implementierung nur die *receive()*-Methode überschreiben. Das liegt daran, dass ein IMV von sich aus keine Nachrichten verschicken darf. Er darf erst reagieren, sobald eine Nachricht für ihn eingegangen ist. Die Methode *begin()* kann damit entfallen.

Eine weitere Funktion *checkPolicy()* dient dem konkreten IMVs dazu, ein erhaltenes Ergebnis vom IMC mit einer Policy zu vergleichen. Dabei wird automatisch ein Validierungsergebnis erzeugt und eine Empfehlung ausgesprochen. Bei dieser Funktionalität handelt es sich um einen eigenen Building-Block, der in Abschnitt 3.2.4 genauer erläutert wird.

Die Klasse IFIMVImpl implementiert das Interface IF-IMV und sorgt genau wie sein Gegenüber auf Client-Seite für das Versenden und Entgegennehmen von Nachrichten. Die durch `<extern>` gekennzeichneten Funktionen müssen in diesem Fall allerdings das *Linux Dynamic Linkage Platform Binding* unterstützen, damit der TNC Server die IMVs als *Shared Objects* laden kann. Auch hier werden die IMV-Instanzen innerhalb einer Map verwaltet. Auf Server-Seite ist allerdings damit zu rechnen, dass sich mehrere Clients gleichzeitig verbinden wollen. Damit wird es meistens auch mehrere Instanzen eines IMCs gleichzeitig geben. Daher muss sichergestellt werden, dass der Zugriff auf die Map mit den Instanzen Thread-sicher programmiert wird (dies entspricht der funktionalen Anforderung *Threadsicherheit*). In C++ ist keine Container-Klasse von sich aus Thread-sicher.

Die abstrakten Funktionen `createNewIMVInstance()`, `getMessageTypes()` und `getMessageTypeCount()` entsprechen der Funktionsweise und technischen Umsetzung wie beim abstrakten IMC.

#### 3.2.3 Logging

Eine wichtige Komponente sowohl für IMCs und IMVs als auch für andere TNC-Komponenten stellt die Logging-API dar (siehe Abbildung 3.5). Die Verwendung des Log-Mechanismus erhöht die Übersichtlichkeit und Wartbarkeit des Quellcodes.

TNCLog
<del>-logInstances : map&lt;string, TNCLog *&gt;</del> <del>-className : string</del> <del>-methodName : string</del> <del>-debugLevel : int</del>
+debug( message : string, ... ) +info( message : string, ... ) +warn( message : string, ... ) +error( message : string, ... ) +isDebug() : bool +isInfo() : bool +isWarn() : bool +isError() : bool +getLog( className : string, methodName : string ) : TNCLog& +deleteLogs() +setOutputStream( pOut : ostream& ) +setErrOutputStream( pErr : ostream& )

Abbildung 3.5: Logging-API

Mit Hilfe der Klasse `TNCLog` können innerhalb des Quellcodes Log-Ausgaben entweder auf der Konsole oder in eine Datei erzeugt werden. Durch verschiedene Log-Level kann auf die Menge der auszugebenden Informationen Einfluss genommen werden:

**Error** Es werden nur kritische Fehler ausgegeben, die zu einen Abbruch führen.

**Warn** Neben Fehlern werden auch Warnungen ausgegeben. Ggf. kann die aktuelle Verarbeitung nicht mehr korrekt abgeschlossen werden.

**Info** Neben Fehlermeldungen und Warnungen werden auch reine Informationsmeldungen ausgegeben. In diesem Fall wird z.B. bei einem Methodenaufruf eine entsprechende Hinweismeldung erzeugt.

**Debug** Höchster Log-Level. Neben allen bisherigen Meldungen werden auch Debug-Informationen ausgegeben. Hilft bei der Fehlersuche und der Überprüfung von Ergebnissen.

Für eine produktive Version sollte der Log-Level *Error* oder *Warn* gewählt werden.

Für jede Klasse kann ein separater Log-Level eingestellt werden. Die Zuordnung von Klassennamen zu Log-Levels erfolgt dabei in einer Properties-Datei. Standardmäßig wird versucht die Datei `tnc_log.properties` zu laden. Beim Erzeugen einer Log-Instanz kann alternativ aber auch noch ein Dateiname für eine andere properties-Datei übergeben werden. Die Datei wird mit Hilfe einer weiteren Utility-Klasse (`TNCIOHelper`) ausgelesen, die den Building-Block *Properties* repräsentiert. Listing 3.1 zeigt eine beispielhafte Logging-Konfiguration. Sollte ein Klassenname nicht aufgeführt sein, gilt standardmäßig der Error-Level; d.h. Fehlermeldungen werden in jedem Fall ausgegeben.

### 3.2.4 Validierung

Der Building-Block *Validierung* soll den Entwickler von IMVs beim Auswerten der Daten von den IMCs und dem Vergleich mit einer Policy unterstützen (Soll-Ist-Vergleich). Das Protokoll IF-M macht keinerlei Vorgaben in Bezug auf das Datenaustauschformat. Durch die allgemeine Flexibilität und Plattformunabhängigkeit von XML bietet sich dieses Format zum Austausch von Daten zwischen IMCs und IMVs an. Dass der Datenaustausch zwischen TNCC und TNCS ebenfalls über XML-Nachrichten erfolgt und durch

```
# Log-Level
# Error: 0
# Warn : 1
# Info : 2
# Debug: 3

IFIMCImpl=1
AbstractIMC=1
IMCRegistry=0
IMCHostScanner=2
IMCSecurityCenter=3
```

Listing 3.1: Beispiel für `tnc_log.properties`

die Spezifikation zur Schnittstelle IF-TNCCS auch so vorgeschrieben ist, ist ein weiteres Kriterium, das für die Verwendung von XML an dieser Stelle spricht. Daher legt sich der Building-Block *Validierung* auf XML als Austauschformat fest. Dies hindert Entwickler allerdings nicht daran, auch ein proprietäres Format zu wählen. Dabei muss dann lediglich die Auswertung der Daten und der Vergleich mit einer Policy selbstständig implementiert werden.

Der Building-Block legt den Aufbau einer Policy in XML über eine XML-Schema-Beschreibung fest (siehe Anhang Seite 80). Eine Grammatik zur Beschreibung von Policies für Web-Services wurde bereits durch das WS-Policy-Framework<sup>2</sup> beschrieben. Da es sich um ein sehr allgemeines und gut verständliches Modell handelt, lässt es sich gut auf diesen Anwendungsfall übertragen. Auf diese Weise wird ein bereits existierendes Konzept verwendet und es muss nicht auf eine proprietäre Lösung zurückgegriffen werden. Dies könnte die Akzeptanz des Entwicklers erhöhen. Für die Beschreibung von Policies sollte die *Policy Normal Form* verwendet werden, deren Aufbau in Listing 3.2 beschrieben ist.

```
<wsp:Policy>
  <wsp:ExactlyOne>
    <wsp:All>...</wsp:All> +
  </wsp:ExactlyOne>
</wsp:Policy>
```

Listing 3.2: Policy Normal Form

Die Verwendung dieses Modells erlaubt es, verschiedene mögliche Konfigurationen des Clients festzulegen. Dadurch kann der Client mehrere alternative Konfigurationen zur

---

<sup>2</sup>vgl. <http://www.w3.org/2002/ws/policy>



Auswahl haben, aus der er genau eine erfüllen muss. Listing 3.3 zeigt ein einfaches Beispiel.

```
<wsp:Policy>
  <wsp:ExactlyOne>
    <wsp:All>
      <Firewall name="Kerio Personal Firewall" />
      <AntiVirus name="Sophos AntiVirus" />
    </wsp:All>
    <wsp:All>
      <Firewall name="Windows Internal Firewall" />
      <AntiVirus name="AntiVir Personal Edition" />
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Listing 3.3: Beispiel für eine Policy mit mehreren Alternativen

In diesem Fall hat der Anwender z.B. die Auswahl zwischen zwei Firewall-/Anti-Virus-Software-Kombinationen. Er muss genau eine der beiden Konfigurationen erfüllen, jedoch nicht beide.

### Vergleich von Zahlen und Datums-Angaben

Da in vielen Fällen ein einfacher Vergleich von Zeichenketten nicht ausreichend ist, wird das WS-Policy-Modell erweitert. Neben Strings erlaubt die *TNC-IMV-Policy* auch Zahlen und Datums-Angaben, die auf verschiedene Arten miteinander verglichen werden können. Tabelle 3.1 zeigt die möglichen Typen für Tag-Inhalte.

Typ	Attribut	Beschreibung
Zahl	tncp:type="number"	Erlaubt die Angabe von ganzen Zahlen.
Datum	tncp:type="date"	Erlaubt die Angabe eines Datums im Format Tag/Monat/Jahr.
Zeichenkette	tncp:type="string"	Erlaubt die Angabe einer beliebigen Zeichenkette. Dieses Attribut ist <i>Default</i> und kann weggelassen werden.

Tabelle 3.1: Verschiedene Inhalts-Typen für Policy-Tags

Ohne eine weitere Option würde in allen Fällen ein einfacher Vergleich durchgeführt werden. Stimmen die Inhalte des Soll- und Ist-Tags nicht überein, gäbe es einen Validierungsfehler. Um auf die Art des Vergleichs Einfluss zu nehmen, gibt es ein weiteres Attribut

tncp:compare. Die möglichen Vergleichsoperatoren werden in Tabelle 3.2 vorgestellt. Diese sind sowohl für Zahlen als auch Datums-Angaben anwendbar.

Vergleich	Attribut
gleich	tncp:compare="eq"
größer	tncp:compare="gt"
kleiner	tncp:compare="lt"
größer oder gleich	tncp:compare="ge"
kleiner oder gleich	tncp:compare="le"
ungleich	tncp:compare="ne"

Tabelle 3.2: Vergleichs-Operatoren für Policy-Tags

Listing 3.4 zeigt ein Policy-Beispiel mit verschiedenen Vergleichs-Operatoren. Um Tags vorübergehend von einem Vergleich auszuschließen, kann zusätzlich das Attribut tncp:optional=true angegeben werden. Weiterhin kann durch die Wildcard \* jeder beliebige Inhalt eines Tags zugelassen werden.

```
<tncp:Policy>
  <tncp:ExactlyOne>
    <tncp:All>
      <Firewall>
        <name>Kerio Personal Firewall</name>
        <version tncp:type="number" tncp:compare="eq">2</version>
      </Firewall>
      <AntiVirus>
        <name>*</name>
        <update tncp:type="date" tncp:compare="ge">22/07/2006</update>
        <upToDate tncp:type="string">true</upToDate>
        <version tncp:type="number" tncp:optional="true">1</version>
      </AntiVirus>
    </tncp:All>
  </tncp:ExactlyOne>
</tncp:Policy>
```

Listing 3.4: Beispiel für eine Policy mit Vergleichs-Operatoren

## 3.2.5 Windows Registry und WMI

Die Windows-Registrierung dient seit Windows NT als zentrale Konfigurationsdatenbank für das Betriebssystem<sup>3</sup>. Neben Informationen zum Betriebssystem selbst werden hier

---

<sup>3</sup>vgl. <http://de.wikipedia.org/wiki/Windows-Registrierungsdatenbank>

auch Daten zu installierter Software abgelegt. Dabei kann es sich im einfachsten Fall um benutzerspezifische Einstellungen handeln. Oft werden aber auch Pfad- und Versions-Angaben in der Registry gespeichert.

Für die Entwicklung von IMCs unter Windows ist es sinnvoll, einen Wrapper für den Zugriff auf die Registry bereitzustellen, über den auf konkrete Schlüssel oder eine Menge von Schlüsseln innerhalb der Datenbank zugegriffen werden kann. Durch den Zugriff auf die Registry wird es IMCs ermöglicht, festzustellen, ob ihre zu prüfende Software überhaupt installiert ist, und wenn ja, wo im Dateisystem die benötigten Dateien liegen. Im Anschluss können diese Dateien ausgewertet und die Informationen zum IMV geschickt werden. Oft liegen relevante Informationen wie z.B. die Versionsnummer aber auch direkt in der Registrierung und ein Zugriff auf Dateien der Security-Tools kann damit entfallen.

Bei der Integritätsprüfung eines bestimmten Tools ist ein weiteres Hauptmerkmal die Tatsache, ob das Tool aktuell überhaupt läuft (Dienst, Prozess). Um dies festzustellen bietet sich die Verwendung der WMI-Schnittstelle<sup>4</sup> an. Diese Schnittstelle dient der Administration und Fernwartung von Workstations und Servern über Skriptsprachen wie z.B. VBScript. Die Abfragesprache an sich erinnert dabei stark an SQL. Mit Hilfe von WMI können IMCs feststellen, welche Prozesse laufen und welche Dienste gestartet wurden. Neben Skriptsprachen lässt sich auch über eine API für C++ auf diese Informationen zugreifen. Ein Wrapper für WMI kapselt diese Zugriffe und bildet damit den Building-Block *WMI*. Abbildung 3.6 zeigt die beiden APIs in einer Übersicht.

TNCRegistryHelper
<pre>+getRegString( preservedKey : HKEY, subKey : string, valueName : string ) : string +getRegNumber( preservedKey : HKEY, subKey : string, valueName : string ) : long +getRegBinary( preservedKey : HKEY, subKey : string, valueName : string, buffer : BYTE* ) : int +getSubKeys( preservedKey : HKEY, subKey : string ) : vector&lt;string&gt;*</pre>

WMIAccessHelper
<pre>+openWMINamespace( wmiNamespace : string ) : bool +executeWMIStatement( statement : string, values : string[], valueCount : int ) : string[] +isServiceRunning( serviceName : string ) : string +closeWMINamespace()</pre>

Abbildung 3.6: Registry- und WMI-Wrapper-API

<sup>4</sup>Windows Management Instrumentation (siehe auch <http://msdn.microsoft.com/library/en-us/dnwm/html/mngwmi.asp>)

#### 3.2.6 Weitere Building-Blocks

Folgende weitere Komponenten sollen bei der Entwicklung von IMCs und IMVs unterstützen und von immer wiederkehrenden Standardaufgaben befreien:

**Properties** Konfigurationen können aus Properties-Dateien ausgelesen werden. Dazu bietet die Klasse `TNCIOHelper` Methoden für den Zugriff auf Dateien an. Eine Datei kann dabei falls gewünscht ganz gezielt nach einem bestimmten Property durchsucht werden.

**XML** Durch die allgemeine Flexibilität und Plattformunabhängigkeit von XML bietet sich dieses Format zum Austausch von Daten zwischen IMCs und IMVs an. Die Schnittstelle IF-M macht an dieser Stelle keinerlei Vorgaben. Daher stellt die Klasse `TNCXMLHelper` Methoden für den Zugriff auf XML-Dateien und -Streams bereit. Mit Hilfe von Xerces für C++<sup>5</sup> und DOM wird eine XML-Nachricht dabei als Baumstruktur aufgebaut. Dies erleichtert den Zugriff auf einzelne Elemente. Die Helper-Klasse stellt zahlreiche Funktionen für das Suchen von einzelnen und mehreren Elementen innerhalb dieser Baum-Struktur bereit. Weiterhin können auch neue XML-Nachrichten aufgebaut werden.

### 3.3 Ablauf einer Integritätsprüfung mit der API

Mit Hilfe eines Sequenzdiagrammes lässt sich der Ablauf einer beispielhaften Integritätsprüfung unter Verwendung der vorgestellten API für IMCs und IMVs verdeutlichen. Abbildung 3.7 zeigt einen TNC-Handshake über vier Runden. Aus Gründen der Übersichtlichkeit wurden nur die wesentlichen Änderungen der Connection-States berücksichtigt (Aufruf der Methode `NotifyConnectionChange()`).

Nach dem Laden eines IMCs durch den TNC Client wird die `Initialize()`-Methode der IMC-Basisimplementierung aufgerufen (Schritt 3). Dabei wird die IMC-ID übergeben. Damit der IMC in den Folgeschritten Nachrichten an den IMV senden kann, muss er entsprechende Funktionen vom TNC Client aufrufen können. Durch Aufruf der Methode `ProvideBindFunction()` erhält der IMC einen Zeiger auf eine TNCC-Funktion, über die er Zeiger auf alle anderen nach außen freigegebenen Funktionen des TNC Clients er-

---

<sup>5</sup><http://xml.apache.org/xerces-c>

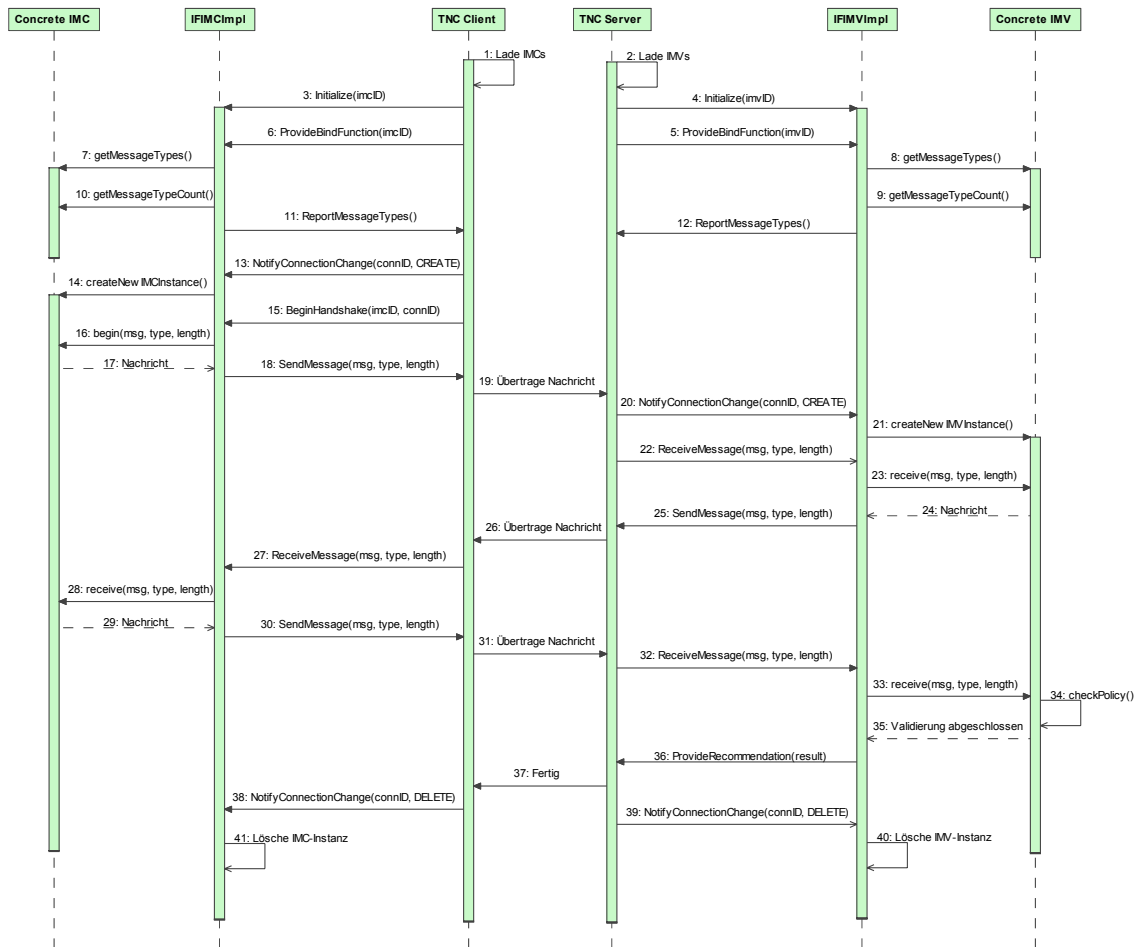


Abbildung 3.7: Ablauf einer Integritätsprüfung mit der API

halten kann (Schritt 6). Der Zeiger wird innerhalb der Basisimplementierung IFIMCImpl gespeichert. Bevor der IMC zum Aufrufer zurückkehrt, muss er die Nachrichten-Typen, an dessen Empfang der konkrete IMC interessiert ist, und deren Anzahl dem TNC Client mitteilen (Schritt 11). Da zu diesem Zeitpunkt aber noch keine Instanz des konkreten IMCs existiert und der Name der Klasse unbekannt ist, muss der konkrete IMC zwei weitere Funktionen implementieren (Schritte 7 und 10), die innerhalb des Header-Files der Klasse IFIMCImpl definiert sind. Während des Linker-Vorgangs beim Bauen werden die Referenzen auf die Implementierungen der Funktionen automatisch gefunden und zugeordnet (siehe auch Abschnitt 3.2.1). Die bisherigen Schritte werden analog auch auf Server-Seite zwischen TNC Server und IMV durchgeführt.

Sobald der Client eine Verbindung zum Server aufgebaut hat, wird ein neuer Handshake begonnen. Dazu wird dem IMC eine Änderung des Status der Netzwerkverbindung mit-

geteilt (Schritt 13). Anhand des Status *CREATE* erkennt der IMC, dass eine neue Verbindung geöffnet wurde. Jetzt erzeugt er eine Instanz der konkreten IMC-Klasse und legt diese innerhalb einer Map ab (Schritt 14). Die Connection-ID dient dabei als eindeutiger Schlüssel. Nach der Rückkehr vom Methodenaufwurf veranlasst der TNC Client den IMC zum Senden der ersten Nachricht an den IMV. Dazu ruft er die Funktion `BeginHandshake()` auf (Schritt 15). Durch die übergebene Connection-ID erkennt der Basis-IMC, um welche Instanz des konkreten IMCs es sich handelt. Durch Aufruf der Funktion `begin()` wird die erste Nachricht erzeugt (Schritt 16); d.h. an dieser Stelle muss der Entwickler eines IMCs das erste mal fachliche Logik implementieren. Das Versenden der Nachricht wird anschließend wieder von der Basisimplementierung übernommen (Schritt 18).

Nach der Übertragung der Nachricht vom TNC Client zum TNC Server (siehe [Sch06]), wird auf Server-Seite analog eine Instanz eines konkreten IMVs erzeugt und in einer Map gespeichert (Schritte 20 und 21). Diese Instanz des IMVs ist jetzt genau für diese eine Netzwerkverbindung verantwortlich. Der TNC Server ruft nun die `ReceiveMessage()`-Methode der IMV-Basisimplementierung auf. Die Klasse `IFIMVImpl` leitet die Nachricht durch Aufruf der Funktion `receive()` an den konkreten IMV weiter (Schritte 22 und 23). An dieser Stelle ist jetzt wieder fachliche Logik des Entwicklers gefordert. Er muss die Nachricht des IMCs auswerten und falls nötig eine Antwort-Nachricht erzeugen. Wäre der IMV anhand der übertragenen Daten jetzt schon in der Lage eine Aussage zur Teil-Integrität des Clients zu treffen, müsste er keine Nachricht mehr zurückschicken. Die Validierung wäre in diesem Fall jetzt bereits beendet. In diesem Beispiel werden allerdings noch weitere Informationen vom IMC erwartet. Daher schickt der IMV eine Nachricht zurück, die dem konkreten IMC über die Funktion `receive()` zugestellt wird (Schritt 28). Nach der nächsten Antwort des IMCs kann der IMV seine Validierung abschließen. Dazu führt er einen Vergleich der erhaltenen Daten mit einer Policy durch (Schritt 34). Er nutzt dazu die Funktionalität *Validierung*, die innerhalb des `AbstractIMV` implementiert ist (siehe dazu auch Abschnitt 3.2.4). Dabei wird automatisch eine Empfehlung für den TNC Server sowie ein Ergebnis der Prüfung erzeugt. Der Entwickler des IMVs kann an dieser Stelle allerdings auch noch Einfluss auf die Entscheidung nehmen. Darüber hinaus ist er nicht gezwungen, die vorgegebene Validierung zu verwenden. Eigenimplementierungen sind hier möglich und in einigen Fällen wahrscheinlich auch unumgänglich.

Nachdem die Klasse `IFIMVImpl` dem TNC Server das Ergebnis der Prüfung automatisch mitgeteilt hat, wartet der Server, bis sämtliche IMVs ihre Empfehlungen ausgesprochen

haben oder bis die maximale Anzahl an Runden erreicht wurde. Im letzten Fall fordert der TNCS die restlichen IMVs auf, eine Entscheidung zu treffen. Im Anschluss führt der TNC Server die Ergebnisse zusammen und trifft eine Gesamtentscheidung, ob der Client ins Netz aufgenommen werden darf. Dies kann anhand unterschiedlicher Kriterien geschehen (siehe dazu [Sch06]).

Liegt eine Gesamtentscheidung vor, kann der TNC-Handshake abgeschlossen werden. Dabei wird sowohl dem IMC als auch dem IMV mitgeteilt, dass die Verbindung mit der entsprechenden Connection-ID beendet wird (Schritt 38 bzw. 39). Die Basisimplementierungen löschen daraufhin die Instanzen zu der Connection-ID.





## 4 Integrity Measurement: Windows Registry

Das IMC/IMV-Paar *Windows Registry* dient dem Zweck, beliebige Registrierungseinträge auf Client-Seite auszulesen, die eine Teilaussage zur Integrität des Windows-Rechners liefern können.

### 4.1 Motivation

Neben installierter Software werden durch Windows auch Systemupdates in der Registrierung eingetragen. Durch einen lesenden Zugriff auf die Registrierung könnte man so auf recht einfache Weise ermitteln, welche Updates auf dem System installiert sind. In der Policy des IMVs ließe sich anschließend festlegen, welche kritischen Updates auf jeden Fall installiert sein müssen, damit der Client-Rechner keine potentielle Gefahr für das interne Netz darstellt.

Weiterhin lässt sich der Registrierung entnehmen, welche Dienste standardmäßig mit Windows zusammen gestartet werden. Auch hier könnte man prüfen, ob sicherheitskritische Dienste aus- oder eingeschaltet sind.

Eine weitere Möglichkeit stellt das Auslesen von Parametern zum Betriebssystem selbst dar. Dazu gehören z.B. Name des Betriebssystems, ServicePack-Level und Version.

### 4.2 Anforderungen und Vorgehen

Um den Zugriff auf die Registrierung möglichst flexibel zu gestalten, sollte die Konfiguration der auszulesenden Daten nicht client-seitig sondern server-seitig erfolgen. Werden

mit der Zeit neue Anforderungen an Clients gestellt (z.B. müssen weitere Systemupdates installiert sein), müssten bei einer client-seitigen Konfiguration die IMCs auf allen Client-Rechnern aktualisiert werden. Dies würde einen hohen administrativen Aufwand bedeuten. Muss dagegen nur der IMV auf dem PDP aktualisiert werden, beschränkt sich der Aufwand auf nur eine einzige Komponente.

Da IMCs immer die erste Nachricht bei einem neuen TNC-Handshake senden müssen, erzeugt der *IMCRegistry* zu Beginn nur eine kurze Info-Nachricht für den IMV. Anhand dieser Nachricht weiß der IMV, dass der IMC auf dem Client installiert und lauffähig ist, also Zugriff auf die Registrierung hat.

```
<FHH_IMCRegistry version="1.0">
  <regAccess allowed="true"/>
</FHH_IMCRegistry>
```

Jetzt kann der IMV dem IMC mittels einer Antwort-Nachricht mitteilen, welche Registrierungseinträge auf dem Client ausgelesen werden sollen. Die Struktur einer solchen Nachricht ist durch XML-Schema eindeutig beschrieben (siehe Anhang Seite 77). Da bei einer derartigen Konfiguration jedoch schnell viele Daten zusammenkommen, ist ein Aufbau der XML-Nachricht innerhalb des C-Quellcodes eher von Nachteil. Daher bietet sich das direkte Auslesen aus einer Konfigurationsdatei an. Dies hat den Vorteil, dass die Konfiguration unabhängig vom IMV und dessen Quellcode angepasst werden kann. Im Folgeabschnitt wird die Konfiguration vorgestellt.

### 4.3 Aufbau der XML-Konfiguration

Analog zum Aufbau der ersten IMC-Nachricht besitzt die IMV-Antwort ein eindeutiges Root-Tag mit der verwendeten Versionsnummer als Attribut.

```
<FHH_IMVRegistry version="1.0">
  ...
</FHH_IMVRegistry>
```

Innerhalb dieses Tags können `<regEntry>`-Tags beliebig tief verschachtelt angegeben werden. Neben einem Namen erwartet das Schema als Kinderelement auch den auszu-

lesenden Schlüssel in der Registrierung. Daneben können entweder auszulesende Werte oder weitere Sub-Entries definiert werden. Beispiel:

```
<FHH_IMVRegistry version="1.0">
  <regEntry>
    <name>WindowsInfo</name>
    <key>HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion</key>
    <values>...</values>
  </regEntry>...
</FHH_IMVRegistry>
```

Bei *HKLM* handelt es sich um einen Stammschlüssel innerhalb der Registrierung. Ausgehend von diesem Schlüssel kann nach Unterschlüsseln gesucht werden. Um die Konfiguration zu vereinfachen wurde an dieser Stelle eine verkürzte Schreibweise verwendet. Folgende Übersicht zeigt die existierenden Stammschlüssel innerhalb der Windows-Registrierung und deren Abkürzungen.

- HKLM = HKEY\_LOCAL\_MACHINE
- HKCU = HKEY\_CURRENT\_USER
- HKCR = HKEY\_CLASSES\_ROOT
- HKU = HKEY\_USERS
- HKCC = HKEY\_CURRENT\_CONFIG

### 4.3.1 Werte aus der Registrierung auslesen

Hinter dem Stammschlüssel kann ein beliebig langer Unterschlüssel folgen. Welche konkreten Werte nun aus diesem Schlüssel ausgelesen werden sollen, wird unter `<values>` definiert. Da es drei verschiedene Typen für Registrierungseinträge gibt (Zeichenfolge, Zahlenwert, Binärwert), muss dies bei der Konfiguration berücksichtigt werden.

```
<values>
  <regString>
    <name>WindowsVersion</name>
    <value>CurrentVersion</value>
```

```
</regString>...
</values>
```

Neben `<regString>` ist demnach auch die Angabe von `<regNumber>` und `<regBinary>` erlaubt. Würde man nun diese Nachricht zum *IMCRegistry* schicken, würde dieser den gesuchten Wert *CurrentVersion* aus der Registrierung auslesen und innerhalb einer Antwortnachricht an den IMV zurückschicken. Abbildung 4.1 zeigt den entsprechenden Beispiel-Eintrag.

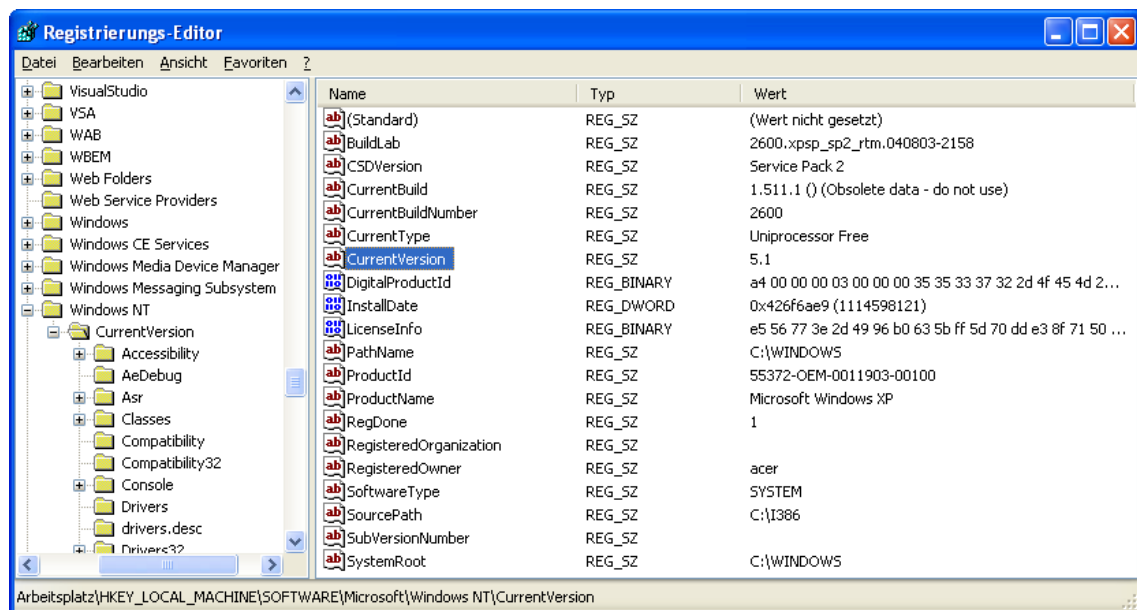


Abbildung 4.1: Beispiel für einen Registrierungseintrag

Möchte man einen anderen Namen für diesen Wert in der Antwort verwenden, kann man diesen optional unter `<name>` spezifizieren. Lässt man die Angabe weg, wird die Bezeichnung unter `<value>` auch für die Antwort verwendet. Die Antwort auf obige Anfrage würde wie folgt aussehen:

```
<FHH_IMCRegistry version="1.0">
  <WindowsInfo>
    <WindowsVersion>5.1</WindowsVersion>
  </WindowsInfo>
</FHH_IMCRegistry>
```

Das `<value>`-Tag erlaubt zusätzlich die Angabe des Attributs *default*, wodurch ein Standardrückgabewert festgelegt werden kann, der verwendet wird, sofern der gesuchte Eintrag in der Registrierung nicht existiert.

```
<value default="5.1">CurrentVersion</value>
```

### 4.3.2 Verschachtelte Einträge

Innerhalb des Tags `<regEntry>` können wie bereits beschrieben weitere Tags dieses Typs definiert werden. Dies erlaubt die Angabe mehrerer Unterschlüssel mit dem selben Schlüssel als Basis. Auf diese Weise muss man für konkrete Unterschlüssel nur noch den relativen Pfad angeben. Dies erleichtert die Konfiguration und macht sie zudem wesentlich übersichtlicher. Beispiel:

```
<regEntry>
  <name>WindowsServices</name>
  <key>HKLM\SYSTEM\CurrentControlSet\Services</key>
  <regEntry>
    <name>SecurityCenter</name>
    <key>wscsvc</key>
    <values>...</values>
  </regEntry>...
</regEntry>
```

Bei der Auswertung der XML-Nachricht baut der IMC die vollständigen Schlüssel dynamisch zusammen. Im obigen Beispiel würde der IMC für den Schlüssel

```
HKLM\SYSTEM\CurrentControlSet\Services\wscsvc
```

die unter `<values>` angegebenen Werte auslesen. Für die Antwort-Nachricht ergibt sich entsprechend folgender Eintrag:

```
<FHH_IMCRegistry version="1.0">
  <WindowsServices>
    <SecurityCenter>
      ...
    </SecurityCenter>
  </WindowsServices>
</FHH_IMCRegistry>
```

```
    </WindowsServices>
</FHH_IMCRegistry>
```

### 4.3.3 Werte als Attribute zurückgeben

Möchte man einen Wert nicht innerhalb eines Tags sondern als Attribut erhalten, kann man dies innerhalb der Konfiguration spezifizieren.

```
<regEntry>
  <name>SecurityCenter</name>
  <key>wscsvc</key>
  <values>
    <regNumber type="attribute">
      <value>start</value>
    </regNumber>...
  </values>
</regEntry>
```

Das Ergebnis würde in diesem Fall so aussehen:

```
<FHH_IMCRegistry version="1.0">
  <WindowsServices>
    <SecurityCenter start="1"/>...
  </WindowsServices>
</FHH_IMCRegistry>
```

### 4.3.4 Werte aller Unterschlüssel auslesen

Um in einem Zug alle Werte mehrerer unbekannter Unterschlüssel auszulesen, lässt die Spezifikation die Angabe einer Wildcard zu.

```
<regEntry>
  <key>*</key>
  <values>
    <regString type="attribute">
      <name>installed</name>
```

```
        <value>InstalledDate</value>
    </regString>...
</values>
</regEntry>
```

## 4.4 Basiskonfiguration des IMVs

In der Basiskonfiguration soll der *IMCRegistry* folgende Parameter aus der Registrierung auslesen:

- Name des Betriebssystems, Version und ServicePack-Level
- Sämtliche installierte Windows-Updates
- Status der Services *SecurityCenter*, *NetMessenger* und Auto-Update
- Status des *Windows Scripting Hosts*
- Auto-Logon-Einstellung

Durch das Auslesen der Betriebssystemparameter wie Name, Version und ServicePack-Level kann z.B. sichergestellt werden, dass auf dem Client-Rechner Windows XP mit SP2 läuft. Durch das Auslesen sämtlicher installierter Updates (dies geschieht per Wildcard) kann innerhalb der Policy für den IMV spezifiziert werden, welche Updates auf dem Client installiert sein müssen. Daneben sollte der Service *SecurityCenter* laufen, der *NetMessenger* dagegen aber überall abgeschaltet sein. Letzterer ist ohne entsprechende Updates anfällig für Angriffe. Der Dienst *Windows Scripting Host* ist u.a. für das Ausführen von VBScripts verantwortlich. Über solche Skripte verbreiten sich oft böswillige Programme, die innerhalb von Firmennetzen großen Schaden anrichten können. Aus diesem Grund empfiehlt es sich, diesen Dienst auszuschalten. Die Logon-Einstellung gibt an, ob am Client-Rechner ein automatisches Login nach dem Start von Windows vorgenommen wird. Da dies in der Regel große Sicherheitsrisiken mit sich bringt, sollte die Angabe eines Benutzernamens mit Passwort Pflicht sein.

Im Anhang auf Seite 81 ist die entsprechende XML-Konfiguration einzusehen, die vom IMV zur Laufzeit aus einer XML-Datei eingelesen und als Nachricht an den IMC geschickt wird.

## 4.5 Ablauf der Integritätsprüfung

Abbildung 4.2 zeigt den Nachrichtenaustausch zwischen IMC und IMV. Die Kommunikation innerhalb des TNC-Handshakes läuft über vier Runden. Zur Vereinfachung wurden TNC Client und TNC Server weggelassen.

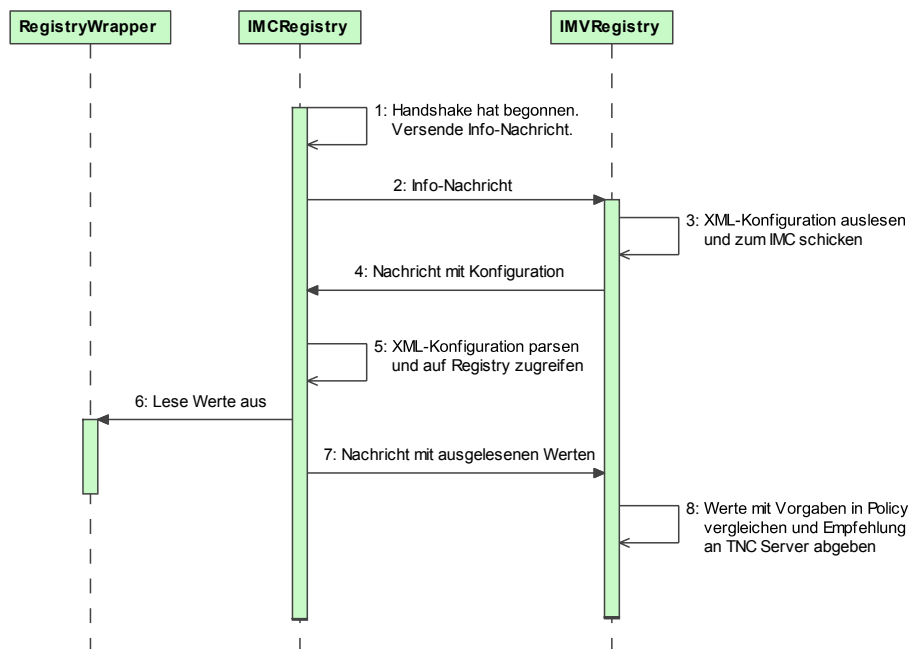


Abbildung 4.2: Nachrichtenaustausch zw. IMCRegistry und IMVRegistry

Wie bereits beschrieben wurde schickt der IMC nach Beginn des Handshakes eine Info-Nachricht an den IMV (Schritt 2). Dieser liest mit Hilfe der Klasse *TNCXMLHelper* die XML-Konfigurationsdatei ein und schickt den Inhalt unverändert an den IMC (Schritte 3 und 4). Dieser parst die Struktur der XML-Nachricht und liest die geforderten Registrierungswerte aus (Schritte 5 und 6). Dazu verwendet er den *RegistryWrapper*, welcher die C-Funktionen für den Zugriff auf die Registrierung kapselt. Während des Parsens wird direkt die Antwort-Nachricht zusammengebaut. Dies geschieht wiederum unter Verwendung des XML-Helpers.

Hat der IMV letztlich die Antwort-Nachricht erhalten, kann er diese mit Hilfe der Funktionalität des Building-Blocks *Validierung* (siehe Kapitel 3.2.4) automatisch mit der Policy vergleichen lassen (Schritt 8).



## 4.6 Mögliche Prüfungsergebnisse

Da auf die Windows-Registrierung generell zumindest lesend zugegriffen werden darf, ist nicht damit zu rechnen, dass der *IMCRegistry* aufgrund fehlender Rechte keinen Zugriff auf die Registrierung hat. Daher sollte der IMV alle geforderten Werte aus der Registrierung auch erhalten. Werden bei der Prüfung Unregelmäßigkeiten festgestellt, wird der Clientrechner als „nicht konform“ angesehen. In diesem Fall spricht sich der IMV gegenüber dem TNCS für „Kein Zugang“ aus.

Sollte die Validierung dagegen keine Abweichungen von der Policy feststellen, wird der Client-PC als „konform“ betrachtet und darf aus Sicht dieses IMVs in das interne Netz aufgenommen werden.

Die Policy für die in Abschnitt 4.4 beschriebene Konfiguration ist im Anhang auf Seite 85 zu finden.



# 5 Integrity Measurement:

## Host-Scanner

Mit Hilfe eines im IMC integrierten Host-Scanners können auf Client-Seite offene Ports aufgespürt werden. Bei einer entsprechenden Konfiguration des IMVs können kritische Ports, die entgegen der Bestimmungen geöffnet sind, zu einem Ausschluss aus dem internen Netz führen.

### 5.1 Motivation

Vor ungefähr drei Jahren wurde bei einigen Microsoft Betriebssystemen (darunter auch Windows XP) eine Sicherheitslücke beim RPC-Dienst<sup>1</sup> bekannt. Angreifen war es über eine Netzwerkverbindung möglich, eine Pufferüberlaufschwachstelle dieses Dienstes auszunutzen, um beliebigen Code mit System-Rechten auszuführen<sup>2</sup>. Durch entsprechende, von Microsoft bereitgestellte, Patches wurde das Problem zwar behoben, in der Zwischenzeit waren aber bereits viele PCs in Mitleidenschaft gezogen worden. Durch diese Sicherheitslücke konnten sich nämlich auch gut Würmer und Viren im Netz verbreiten. Dabei hätte ein Sperren von wenigen Ports (u.a. 135) mittels einer Firewall bereits guten Schutz geboten. Allerdings wurde bei Windows XP vor der Veröffentlichung von Service-Pack 2 standardmäßig keine interne Firewall mit ausgeliefert.

Dieses Beispiel zeigt, dass es nützlich ist, kritische Ports, die vom Anwender in der Regel nicht gebraucht werden, durch eine Firewall zu blocken oder die entsprechenden Dienste ganz abzuschalten. Auf diese Weise wird der Schutz des eigenen Rechners und des gesamten Netzes erhöht. Durch einen client-seitigen Host-Scanner könnten somit kritische Ports auf ihre Verfügbarkeit hin überprüft werden. Eine entsprechende Konfiguration ei-

---

<sup>1</sup>Remote Procedure Call

<sup>2</sup>vgl. <http://cert.uni-stuttgart.de/ticker/article.php?mid=1124>

nes IMVs auf dem PDP würde dafür Sorge tragen, dass bei Verstößen eine Empfehlung zur Isolierung des Client-Rechners ausgesprochen werden würde.

## 5.2 Anforderungen und Vorgehen

Da bei diesem IMC/IMV-Paar mit einer häufigen Veränderung oder Erweiterung der Anforderungen an den Client zu rechnen ist, sollte die Konfiguration der zu überprüfenden Ports auf Server-Seite erfolgen. Damit ist bei Änderungen der Sicherheitsbestimmungen keine Aktualisierung der IMCs auf den Client-Rechnern notwendig. Um auch eine Anpassung des Quellcodes des IMVs zu vermeiden, macht eine Konfiguration per XML-Datei Sinn. Die Struktur dieser XML-Datei ist durch XML-Schema beschrieben (siehe Anhang ab Seite 78).

Zu Beginn eines TNC-Handshakes muss sich der *IMCHostScanner* beim IMV melden. Dies geschieht über eine einfache Info-Nachricht.

```
<FHH_IMCHostScanner version="1.0">
  <HostScanner active="true"/>
</FHH_IMCHostScanner>
```

Hat der IMV diese Nachricht empfangen und eine Versionsübereinstimmung erkannt, liest er die XML-Konfiguration der zu prüfenden Ports ein. Dabei handelt es sich bereits um eine gültige XML-Nachricht, die unverändert zum IMC geschickt werden kann. Durch den im IMC integrierten Host-Scanner können die angegebenen Ports überprüft werden. Eine Antwort-Nachricht teilt dem IMV mit, welche Ports geöffnet und welche Ports geschlossen sind. Da ein Port-Scan unter Umständen sehr lange dauern kann – dies liegt zum größten Teil an der Wartezeit, bis ein Timeout bei einem geschlossenen oder geblockten Port eintritt – sollte die Menge der zu prüfenden Ports mit Bedacht gewählt werden.

## 5.3 Aufbau der XML-Konfiguration

Der Aufbau der IMV-Antwort-Nachricht ist vergleichsweise einfach. Innerhalb des eindeutigen Root-Tags wird das Unterelement `<ports>` erwartet. Im Anschluss kann eine be-

beliebige Anzahl an `<port>`-Tags folgen, wobei als Attribut die zu prüfende Port-Nummer mitgeliefert wird. Beispiel:

```
<FHH_IMVHostScanner version="1.0">
  <ports>
    <port number="80"/>
    <port number="135"/>...
  </ports>
</FHH_IMVHostScanner>
```

Die Antwort-Nachricht des IMCs ist sehr ähnlich aufgebaut. Zusätzlich zur Portnummer werden hier noch die Attribute *open* mit den Werten *true* oder *false* und *service* ergänzt. Letzteres Attribut enthält die Bezeichnung des erkannten Services und wird nur mitgeliefert, sofern der Port geöffnet und der dazugehörige Dienst bekannt ist.

```
<FHH_IMCHostScanner version="1.0">
  <ports>
    <port number="80" open="false"/>
    <port number="135" open="true" service="epmap"/>...
  </ports>
</FHH_IMCHostScanner>
```

## 5.4 Basiskonfiguration des IMVs

In der Basiskonfiguration des IMVs werden die in Tabelle 5.1 aufgeführten Ports geprüft. Im Anhang auf Seite 83 ist die dazugehörige XML-Konfiguration zu finden.

Port	Dienst	Begründung
21	FTP <sup>3</sup>	Ein Clientrechner in einem Firmennetz sollte keinen eigenen FTP-Server bereitstellen.
22	SSH	Auch ein eigener SSH-Server bringt auf einem Client-PC keinen Nutzen und sollte daher nicht vorhanden sein.
25	SMTP <sup>4</sup>	Bei einem Mail-Server handelt es sich um einen zentralen Dienst, der auf Clientrechnern nicht laufen sollte.
42	DNS <sup>5</sup>	Ein Nameserver auf einem Client-PC deutet i.d.R. auf den Versuch einer DNS-Spoofing-Attacke hin. Ein Angreifer versucht in diesem Fall, DNS-Anfragen noch vor dem eigentlichen DNS-Server zu beantworten und die IP-Pakete somit z.B. an seinen eigenen Rechner umzuleiten ( <i>Man-in-the-Middle-Attacke</i> ). Weitere Informationen dazu siehe [Aur05, S. 20ff].
67	BOOTPS	Auf diesem Port nimmt ein BOOTP <sup>6</sup> - bzw. DHCP <sup>7</sup> -Server Requests von Clients entgegen. Da ein solcher Dienst auf Clientrechnern auf den Versuch eines DHCP-basierten Angriffs hindeutet, muss der Port in jedem Fall geschlossen sein. Weitere Informationen zu DHCP-Angriffen liefert Aurand in [Aur05, S. 231-254].
135	RPC Endpoint Mapper	Der RPC-Dienst von Microsoft hat in der Vergangenheit zahlreiche Sicherheitslücken aufgezeigt (siehe Abschnitt 5.1). Daher sollte der Port geschlossen sein.

Tabelle 5.1: Host-Scanner – zu prüfende Ports

<sup>3</sup>File Transfer Protocol (RFC 959)

<sup>4</sup>Simple Mail Transfer Protocol (RFC 821)

<sup>5</sup>Domain-Name-System (RFCs 1034 und 1035)

<sup>6</sup>Bootstrap Protocol (RFC 951)

<sup>7</sup>Dynamic Host Configuration Protocol (RFC 2131)

## 5.5 Ablauf der Integritätsprüfung

Abbildung 5.1 zeigt einen beispielhaften Nachrichtenaustausch zwischen IMC und IMV. Wie bei den Integrity-Measurement-Komponenten zum Auslesen der Windows-Registrierung (siehe Kapitel 4), läuft die Kommunikation maximal über vier Runden. Zur Vereinfachung wurden der TNC Client und der TNC Server beim Kommunikationsverlauf weggelassen.

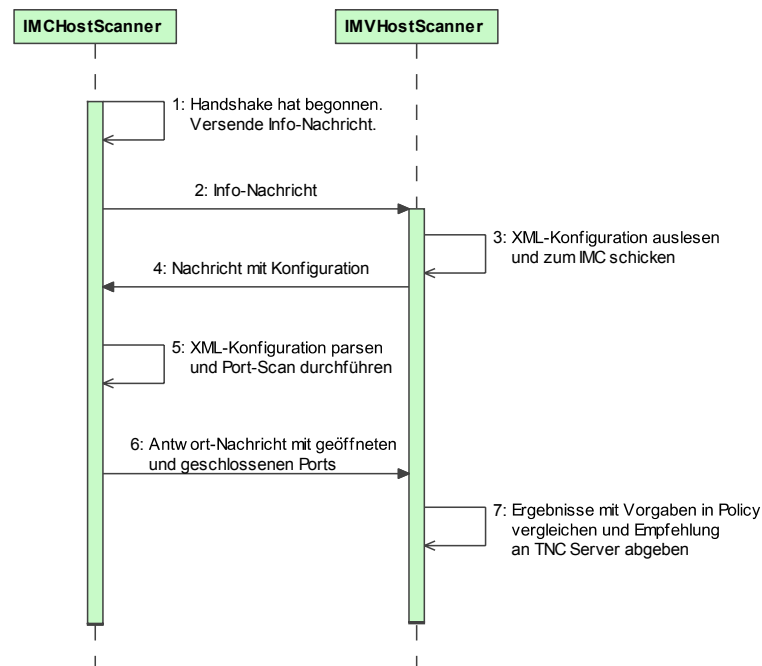


Abbildung 5.1: Nachrichtenaustausch zw. IMCHostScanner und IMVHostScanner

Nach dem Empfang der Info-Nachricht vom IMC an den IMV (Schritt 2), liest der IMV die XML-Konfiguration aus dem Dateisystem ein (Schritt 3). Diese wird unverändert zum IMV zurückgeschickt (Schritt 4). Nach dem Parsen der XML-Nachricht beginnt der IMC mit dem Port-Scan (Schritt 5). Im Anschluss an die Prüfung baut er eine XML-Antwortnachricht mit den gesammelten Information zusammen und sendet diese zum IMV (Schritt 6). Wurde die Antwortnachricht dem IMV durch den TNC Server zugestellt, kann dieser mit der Validierung der Daten beginnen. Dazu nutzt er die bereitgestellte Funktionalität des abstrakten IMV (Building-Block *Validierung*). Nach Abschluss der Validierung liegt das Ergebnis samt Empfehlung vor, welches dem TNC Server mitgeteilt wird (Schritt 7).

Die Policy für die in Abschnitt 5.4 vorgestellte Konfiguration ist im Anhang auf Seite 86 zu finden.



## 6 Integrity Measurement: Windows Security Center

Der IMC für das *Security Center* von Windows XP ist in der Lage die Parameter installierter Anti-Virus-Software sowie einer Personal Firewall auszulesen.

### 6.1 Motivation

Aktuelle Anti-Virus-Software ist heutzutage für Privatanwender genauso unverzichtbar wie für Firmen oder öffentliche Einrichtungen mit einem großen lokalen Netzwerk. Das gleiche gilt für den Einsatz von Firewalls. Auf Heim-PCs sollte stets eine funktionsfähige Personal Firewall installiert sein, die den Anwender vor den meisten Angriffen aus dem Internet schützen sollte.

Das *Security Center* von Windows XP, welches seit Einführung des Service Packs 2 als Hintergrunddienst auf dem Rechner läuft, erkennt das Vorhandensein und die Aktualität eben dieser Security-Tools. Es bietet dem Anwender die Möglichkeit, den Status der Sicherheitskomponenten zentral und in Echtzeit zu überwachen. Dazu liest das *Security Center* Parameter aus der WMI-Datenbank aus.

Zahlreiche Hersteller von Anti-Virus-Software und Firewall-Produkten unterstützen dieses Verfahren und tragen ihre Parameter in die WMI-Datenbank ein. Dazu gehören auch für den Privatanwender frei verfügbare Tools wie z.B. die *Kerio Personal Firewall*<sup>1</sup> und die *AntiVir PersonalEdition Classic*<sup>2</sup>.

Einen genauen Überblick über die Funktionsweise des *Security Centers* von Windows XP gibt der Artikel *Windows Security Center – Managing the State of Security* [Mic04].

---

<sup>1</sup><http://www.sunbelt-software.com/Kerio.cfm>

<sup>2</sup><http://www.free-av.de>

## 6.2 Anforderungen und Vorgehen

Windows XP stellt eine API für den Zugriff auf die WMI-Datenbank via C++ bereit. Der in Kapitel 3.2.5 vorgestellte WMI-Wrapper kapselt den Zugriff auf diese WMI-Schnittstelle. Die Parameter der Security-Tools sind in der WMI-Datenbank unter dem Namespace `root/SecurityCenter` zu finden. Abbildung 6.1 zeigt einen Eintrag für eine installierte und eingeschaltete Firewall.

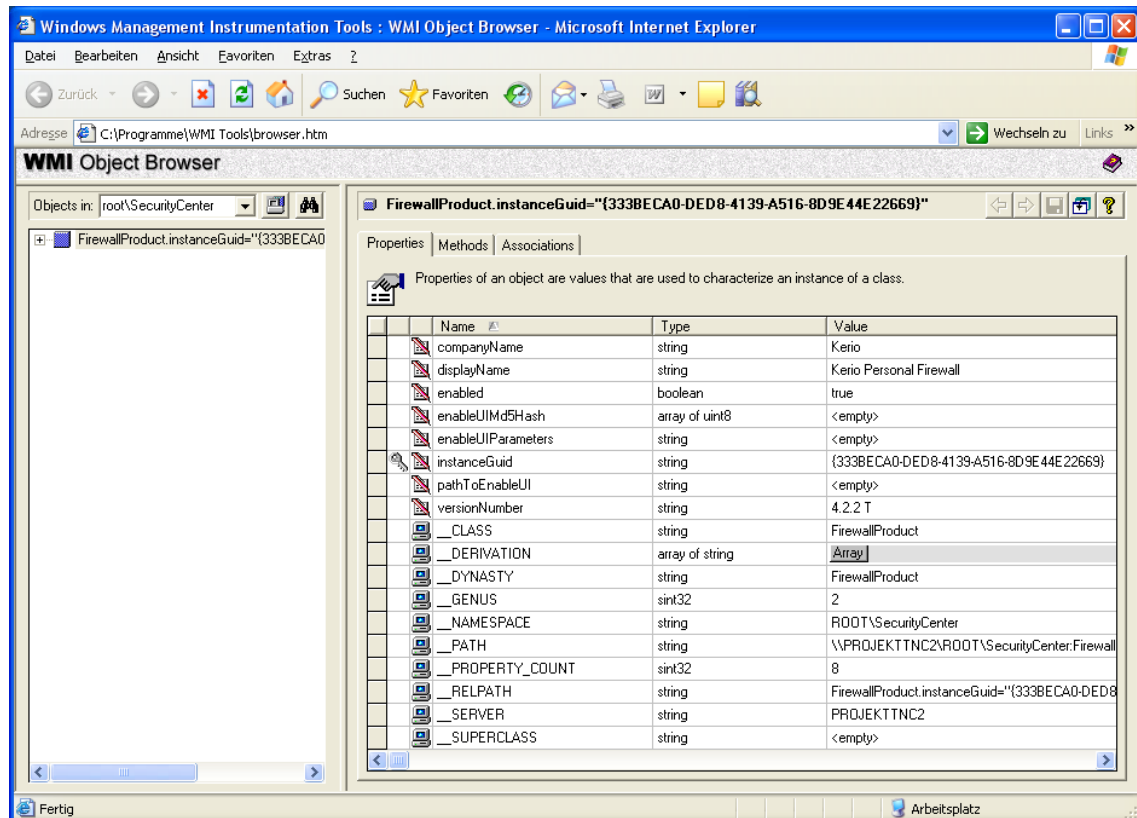


Abbildung 6.1: WMI Object Browser

Der Zugriff auf die WMI-Datenbank zur Feststellung der AV- und FW-Parameter ist für alle potentiellen Security-Tools gleich. Daher muss server-seitig keine Konfiguration erfolgen. Der *IMCSecurityCenter* kann direkt in der ersten Runde des TNC-Handshakes die ausgelesenen Parameter an den IMV zur Auswertung schicken.

Das *Security Center* akzeptiert als Firewall ebenfalls die interne Windows-Firewall. Diese nimmt allerdings keine Eintragung in der WMI-Datenbank vor. Sollte keine Firewall eines anderen Herstellers auf dem System installiert sein – sprich es wurde kein WMI-Eintrag im oben genannten Namespace für eine Firewall gefunden – sollte alternativ geprüft wer-

den, ob zumindest die interne Windows-Firewall als Hintergrundprozess läuft. Zu diesem Zweck stellt der *WMI-Wrapper* entsprechende Funktionalität bereit.

Die ausgelesenen Parameter werden innerhalb einer XML-Nachricht an den IMV geschickt.

```
<FHH_IMCSecurityCenter version="1.0">
  <FirewallProduct>...</FirewallProduct>
  <AntiVirusProduct>...</AntiVirusProduct>
</FHH_IMCSecurityCenter>
```

Sollte keine Firewall und/oder Anti-Virus-Software gefunden worden sein, entfällt der entsprechende Knoten innerhalb der XML-Struktur. In diesem Fall sollte der IMV die Integritätsprüfung allerdings noch nicht direkt beenden und ein negatives Ergebnis an den TNC Server melden. Es besteht die Möglichkeit, dass zwar ein Security-Tool installiert ist, dieses aber keine Eintragung in der WMI-Datenbank vornimmt und somit nicht vom *Security Center* erkannt wird. Für solche Tools muss es dann ein eigenes IMC/IMV-Paar zur Validierung geben. Ein Beispiel ist der *ClamWin*-Virenschanner, dessen Integrity-Measurement-Komponente in Kapitel 7 vorgestellt wird.

Damit der *IMVSecurityCenter* nun mitbekommen kann, dass z.B. doch eine Anti-Virus-Software installiert ist und nur nicht erkannt werden konnte, sollte der IMV des betroffenen Security-Tools eine allgemeine Nachricht versenden, die vom *IMCSecurityCenter* empfangen werden kann. Zur Identifizierung sollte ein eigener *MessageType* ausreichend sein. Diese Information kann der IMC dann an den IMV weiterleiten. Dieser muss sich nun auf das Prüfungsergebnis des anderen IMVs verlassen und kann eine positive Empfehlung gegenüber dem TNC Server aussprechen.

## 6.3 Ermittlung der AV-Parameter

Nach dem Öffnen des Namespaces können die Anti-Virus-Parameter mit folgendem Statement aus der WMI-Datenbank ausgelesen werden:

```
Select * from AntiVirusProduct
```

Für die Integritätsprüfung auf Server-Seite sind die in Tabelle 6.1 aufgeführten Werte relevant.

Wert	Bedeutung
displayName	Gibt den Produktnamen an.
versionNumber	Versionsnummer des Produkts.
productUptoDate	Zeigt an, ob die Viren-Definitions-Dateien auf einem aktuellen Stand sind. Mögliche Werte: <i>true</i> oder <i>false</i>
onAccessScanningEnabled	Gibt an, ob der Echtzeitvirenschutz aktiv ist. In diesem Fall wird bei einem Dateizugriff automatisch ein Scan durchgeführt. Mögliche Werte: <i>true</i> oder <i>false</i>

Tabelle 6.1: AV-Parameter des Security Centers

Die ermittelten Daten werden als eigener XML-Knoten innerhalb der ersten Nachricht an den Server geschickt. Beispiel:

```
<AntiVirusProduct>
  <productName>AntiVir PersonalEdition Classic</productName>
  <version>1.5</version>
  <upToDate>true</upToDate>
  <enabled>true</enabled>
</AntiVirusProduct>
```

## 6.4 Ermittlung der FW-Parameter

Analog zu den AV-Parametern kann man auf die Daten der Firewall in der WMI-Datenbank mit folgendem Statement zugreifen:

```
Select * from FirewallProduct
```

Tabelle 6.2 gibt eine Übersicht über die Werte, die für die Integritätsprüfung herangezogen werden.

Wurde ein Eintrag in der Datenbank gefunden, werden die ermittelten Werte in die XML-Nachricht aufgenommen. Im anderen Fall wird durch eine WMI-Abfrage ermittelt, ob die interne Windows-Firewall als Dienst läuft. Daten zu laufenden Prozessen oder dem Status von Diensten sind im Namespace `root/CIMV2` hinterlegt. Folgendes Statement liefert die

Wert	Bedeutung
displayName	Gibt den Produktnamen an.
versionNumber	Versionsnummer des Produkts.
enabled	Zeigt an, ob die Firewall aktiv ist und den Netzwerkverkehr filtert. Mögliche Werte: <i>true</i> oder <i>false</i>

Tabelle 6.2: FW-Parameter des Security Centers

Informationen zum Service *Windows Firewall/Gemeinsame Nutzung der Internetverbindung*, welcher unter der Kurzbezeichnung *SharedAccess* zu finden ist.

```
Select * from Win32_Service where Name = "SharedAccess"
```

Über den Wert *State* oder *Started* (letzterer ist in der vereinfachten Ausgabe in der WMI-Konsole nicht aufgeführt) kann nun ermittelt werden, ob die interne Firewall läuft. Abbildung 6.2 zeigt eine entsprechende Abfrage über die WMI-Konsole.

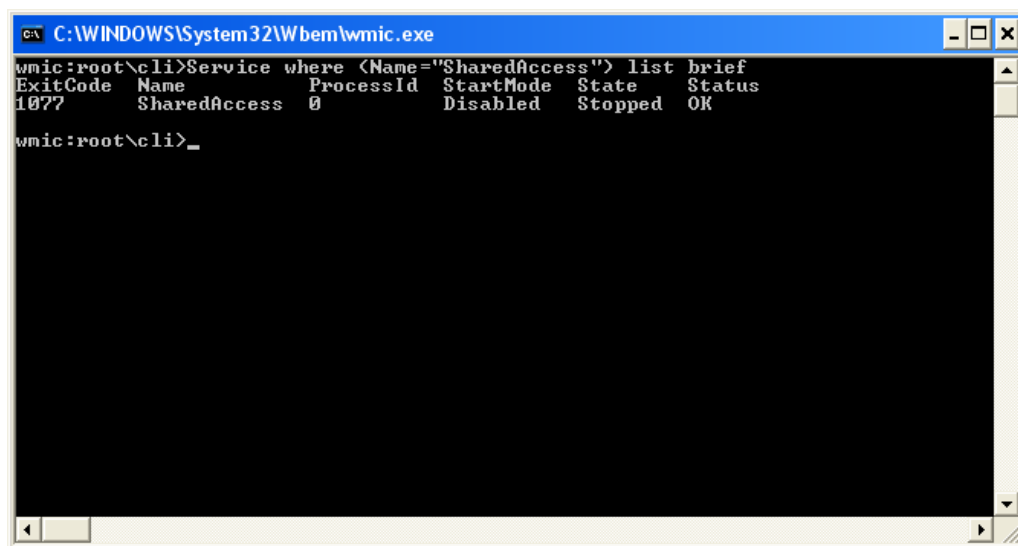


Abbildung 6.2: WMI-Abfrage zur Ermittlung des Status der Windows-Firewall

Läuft die interne Firewall, werden die Daten in die XML-Nachricht als eigener Knoten aufgenommen. Der Aufbau der Nachricht ist in beiden Fällen gleich. Beispiel:

```
<FirewallProduct>
  <productName>Kerio Personal Firewall</productName>
  <version>4.2.2</version>
```

```
<enabled>true</enabled>  
</FirewallProduct>
```

## 6.5 Ablauf der Integritätsprüfung

Sobald der TNC-Handshake begonnen hat, liest der *IMCSecurityCenter* die Anti-Virus- und Firewall-Parameter aus. Je nachdem, ob es Einträge in der WMI-Datenbank gefunden hat, übernimmt er Daten in die XML-Nachricht an den IMV. Abbildung 6.3 verdeutlicht den Ablauf mit den möglichen Entscheidungswegen.

Empfängt der IMV eine Nachricht, in der sowohl Daten zu Anti-Virus-Software als auch zu einer Personal Firewall vorkommen, kann er einen Vergleich der Daten mit der Policy durchführen. Dabei sollten die Werte für *productUptoDate* und *onAccessScanningEnabled* des Viren-Scanners sowie der Wert für *enabled* der Firewall stets auf *true* gesetzt werden. Möchte man auch konkrete Produkte und/oder Versionen für die Security-Tools fordern, kann man diese Daten innerhalb der Policy für *displayName* und *versionNumber* angeben. Im Standardfall sollten aber alle Security-Tools zugelassen werden, sofern sie aktuell und einsatzbereit sind. Die Verwendung einer Wildcard für Name und Version erscheint daher sinnvoll (siehe Kapitel 3.2.4).

Beispielhafte Policies sind im Anhang auf Seite 87 zu finden.

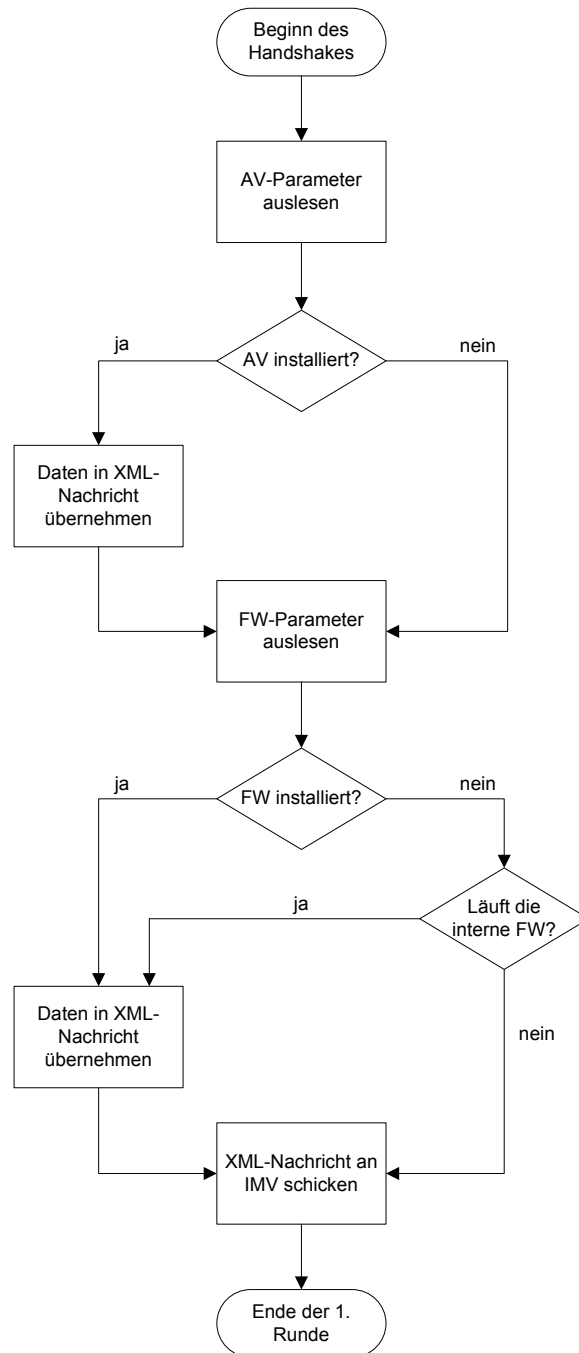


Abbildung 6.3: Entscheidungen im Ablauf des IMCSecurityCenter





## 7 Integrity Measurement: ClamWin Virens Scanner

Diese Integrity-Measurement-Komponente ist verantwortlich für die Prüfung des Open-Source Virens Scanners *ClamWin*<sup>1</sup>.

### 7.1 Motivation

Der *ClamWin*-Virens Scanner ist ein bekannter Vertreter aus dem Open-Source-Bereich für Security-Tools. Es handelt sich um einen Stand-Alone Virens Scanner, dessen Virendatenbanken regelmäßig aktualisiert werden. Das Tool bietet die Möglichkeit, automatische Updates durchzuführen.

Allerdings bietet das Tool keinen Echtzeitschutz an; d.h., dass keine Dateien automatisch geprüft werden. Aus diesem Grund kann das Security-Center von Windows XP nicht erkennen, dass Anti-Viren-Software auf dem System installiert ist (siehe auch Kapitel 6). Eine entsprechende Erweiterung ist zwar geplant, aber ein Einsatz derzeit noch nicht absehbar. Daher macht es Sinn für eine solche Komponente ein separates IMC/IMV-Paar bereitzustellen, welches eine Prüfung der Aktualität des Tools vornimmt.

### 7.2 Anforderungen und Vorgehen

*ClamWin* legt Informationen zur Programmversion und dem Installationspfad innerhalb der Registrierung ab. Eine config-Datei innerhalb des Installationsordners gibt Aufschluss über den Speicherort der Virendefinitions-Dateien. Dabei handelt es sich um Binär-Datei-

---

<sup>1</sup><http://www.clamwin.de>

en, die am Dateianfang zusätzlich Informationen zum Datum des letzten Updates und der Version der Dateien enthalten. Diese Werte lassen sich auslesen und innerhalb einer XML-Nachricht an den IMV schicken.

Da mit Änderungen am Aufbau der Registrierungseinträge sowie der Konfigurationsdatei und Virendefinitions-Dateien nicht zu rechnen ist, bzw. dies nur bei neuen Programmversionen zu erwarten ist, ist eine server-seitige Konfiguration nicht notwendig. Die Anzahl und die Art der auszulesenden Daten sollte in der Regel konstant bleiben. Daher ist dem IMC bekannt, welche Informationen er auslesen und zum IMV schicken muss.

Sollte der IMV eine korrekt installierte und aktuelle Version des *ClamWin*-Virensanners erkannt haben, ist es sinnvoll, eine allgemeine Nachricht an den *IMCSecurityCenter* zu schicken, damit dieser wiederum seinem IMV mitteilen kann, dass doch ein VirenScanner installiert ist. Ansonsten würde der IMV des SecurityCenters fälschlicherweise ein negatives Ergebnis liefern, wodurch der TNC Server ggf. eine Aufnahme ins interne Netz verweigern würde. Ein direkter Nachrichtenaustausch zwischen IMVs ist in der TNC-Architektur nicht vorgesehen.

## 7.3 Ermittlung der AV-Parameter

### 7.3.1 Installationspfad und Programmversion

Um festzustellen, ob und wenn ja wo *ClamWin* installiert ist, muss mit Hilfe des Registry-Wrappers ein Eintrag aus der Windows-Registrierung ausgelesen werden:

```
HKLM\SOFTWARE\ClamWin
```

Der Wert *Path* gibt den Installationsorder an. Zusätzlich lässt sich über *Version* die aktuelle Programmversion ermitteln. Abbildung 7.1 zeigt den entsprechenden Registrierungseintrag.

### 7.3.2 Status des Hintergrundprozesses

Ein weiteres wichtiges Kriterium ist die Tatsache, ob der Hintergrundprozess des Virensanners überhaupt läuft. Wie bereits beschrieben bietet *ClamWin* zwar keinen Echtzeit-

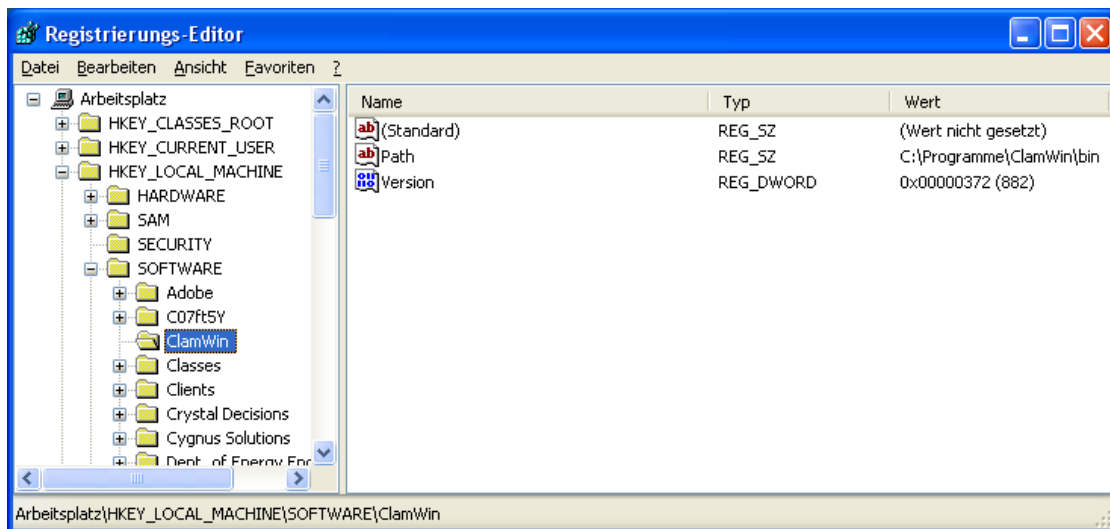


Abbildung 7.1: ClamWin-Installationsorder als Registrierungseintrag

schutz an, dennoch sorgt der Hintergrundprozess für eine Überprüfung der Aktualität des Tools. Sollte es eine neuere Version geben, wird der Anwender darüber informiert (siehe Abbildung 7.2).

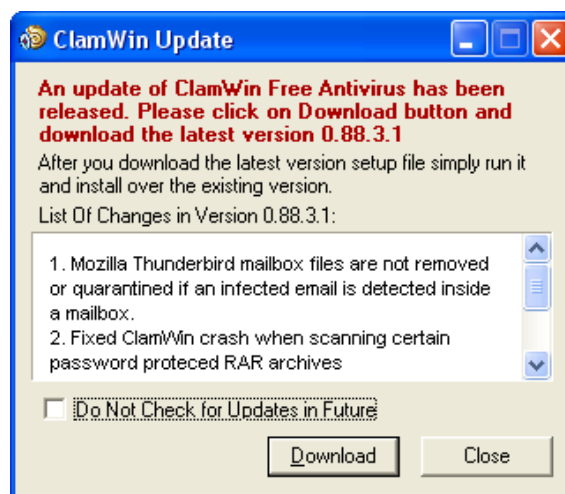
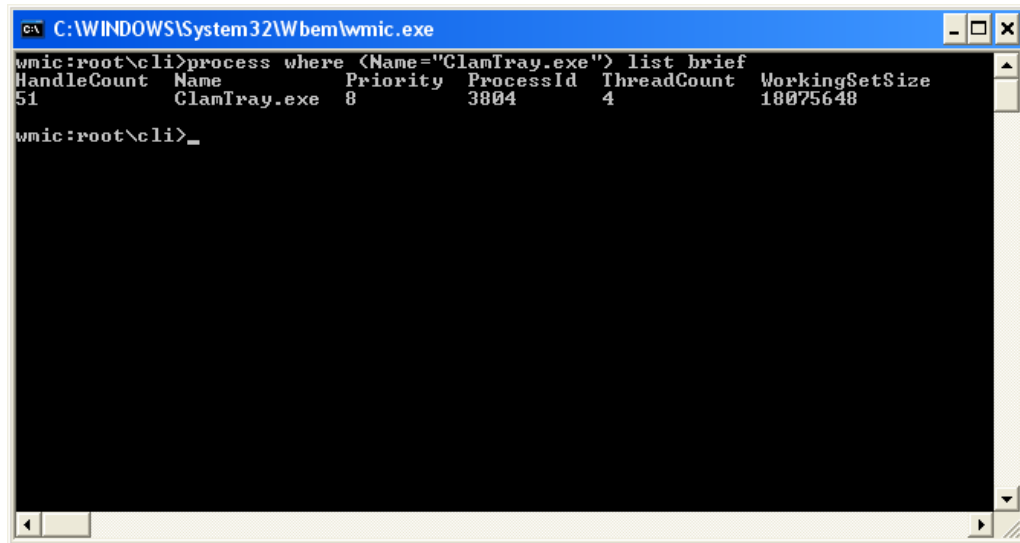


Abbildung 7.2: Hinweis zu verfügbarem ClamWin-Update

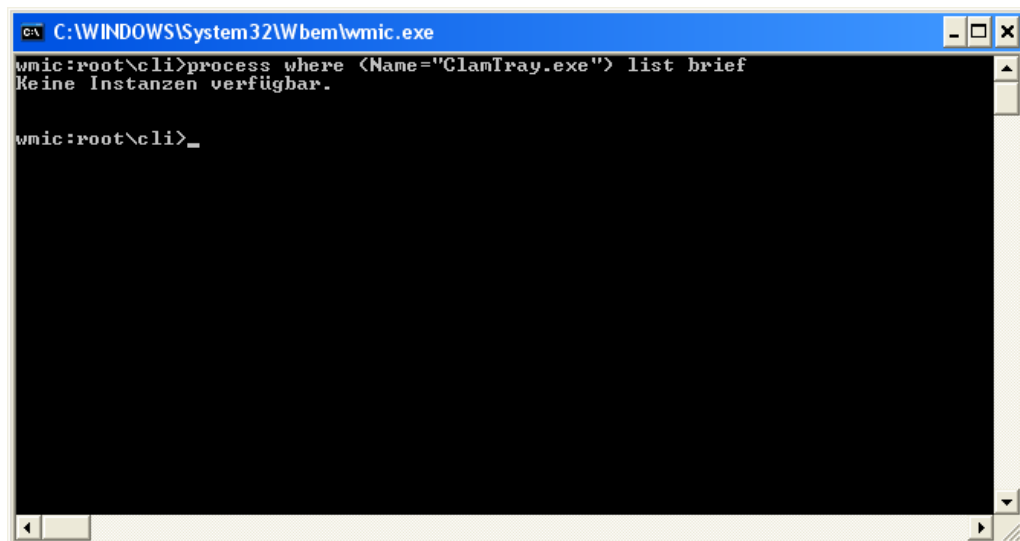
Läuft der Hintergrundprozess nicht, sollte dies als ein KO-Kriterium für den Integritätscheck gewertet werden. Mit Hilfe des WMI-Wrappers lässt sich über eine WMI-Abfrage feststellen, ob der Hintergrundprozess aktiv ist. Wie bereits beschrieben wurde, ähnelt die Syntax der WMI-Abfragesprache stark SQL. Folgende Abbildungen zeigen ein manuelles Abfragen des Prozess-Status über eine WMI-Konsole. Abbildung 7.3 veranschaulicht den

Erfolgsfall; d.h. es wurde ein Eintrag in der Prozessliste für *ClamWin* gefunden. Dagegen zeigt Abbildung 7.4 eine erfolglose Suche. Der Prozess läuft in diesem Fall nicht.



```
C:\WINDOWS\System32\Wbem\wmic.exe
wmic:root\cli>process where <Name="ClamTray.exe"> list brief
HandleCount  Name          Priority  ProcessId  ThreadCount  WorkingSetSize
51           ClamTray.exe    8        3804       4            18075648
wmic:root\cli>_
```

Abbildung 7.3: Abfrage des Prozess-Status über WMI (Prozess läuft)



```
C:\WINDOWS\System32\Wbem\wmic.exe
wmic:root\cli>process where <Name="ClamTray.exe"> list brief
Keine Instanzen verfügbar.
wmic:root\cli>_
```

Abbildung 7.4: Abfrage des Prozess-Status über WMI (Prozess läuft nicht)

### 7.3.3 Version der Virendefinitionsdateien

In der Konfigurationsdatei *ClamWin.conf* ist der Pfad zum Datenbank-Verzeichnis mit den Virendefinitionsdateien als Property hinterlegt. Mit Hilfe der Klasse *TNCIOHelper* lässt sich dieser Wert auslesen.

Jetzt kann auf die beiden Virendefinitionsdateien `main.cvd` und `daily.cvd` zugegriffen werden. Beide Dateien sind gleich aufgebaut. Dies erlaubt einen allgemeinen Algorithmus zur Auswertung. Am Dateianfang stehen jeweils durch einen Doppelpunkt getrennt folgende AV-Parameter:

- Datum der letzten Aktualisierung
- Version der Virendefinitions-Datei
- Anzahl vorhandener Viren-Signaturen

Innerhalb einer Policy auf Server-Seite lassen sich nun Vorgaben für diese AV-Parameter setzen. Es kann z.B. festgelegt werden, wann spätestens das letzte Update erfolgt sein muss und welche Version die beiden Virendefinitionsdateien mindestens haben müssen. Eine beispielhafte Policy ist im Anhang auf Seite 86 zu finden.

## 7.4 Ablauf der Integritätsprüfung

Abhängig von den ermittelten AV-Parametern aus den Abschnitten 7.3.1 und 7.3.2 sind nun folgende Abläufe möglich.

### 7.4.1 ClamWin nicht installiert

Wurde der Eintrag für *ClamWin* nicht in der Registrierung gefunden, geht der IMC davon aus, dass der Viren-Scanner nicht installiert ist. Daher wird folgende Nachricht an den IMV geschickt, der seinerseits dem TNC Server mitteilt, dass er keine Aussage zur Integrität des Clients machen kann. Er enthält sich sozusagen einer Entscheidung.

```
<FHH_IMCClamWin version="1.0">  
  <ClamWin installed="false"/>  
</FHH_IMCClamWin>
```

Weitere Nachrichten werden in diesem TNC-Handshake somit nicht mehr ausgetauscht.

### 7.4.2 Hintergrundprozess läuft nicht

Wurde der Registrierungseintrag zwar gefunden, konnte jedoch nicht festgestellt werden, dass der Hintergrundprozess auch aktiv ist, wird folgende XML-Nachricht an den IMV geschickt. In diesem Fall spricht sich der IMV gegen eine Aufnahme des Client-Rechners in das interne Netz aus.

```
<FHH_IMCClamWin version="1.0">
  <ClamWin installed="true" running="false"/>
</FHH_IMCClamWin>
```

Auch in diesem Fall werden keine weiteren Nachrichten innerhalb des Handshakes mehr ausgetauscht.

### 7.4.3 Hintergrundprozess läuft

Wurde erfolgreich festgestellt, dass der Virens Scanner sowohl installiert als auch einsatzbereit ist, wird folgende Nachricht an den IMV geschickt. Die zuvor aus der Registrierung ausgelesenen Werte für den Installationspfad und die Versionsnummer werden dabei mit übertragen.

```
<FHH_IMCClamWin version="1.0">
  <ClamWin installed="true" running="true">
    <version>882</version>
    <path>C:\Programme\ClamWin\bin</version>
  </ClamWin>
</FHH_IMCClamWin>
```

In diesem Fall antwortet der IMV mit einer einfachen Nachricht ohne konkreten Inhalt. Dies ist möglich, da der Nachricht durch den IMV ein entsprechender *Message-Type* zugeordnet wurde, anhand dessen der IMC feststellen kann, dass er nun die AV-Parameter auslesen und zurückschicken soll.

```
<FHH_IMVClamWin version="1.0"/>
```

Nach dem Auslesen der AV-Parameter wie in Abschnitt 7.3.3 beschrieben, kann der IMC diese Informationen an den IMV schicken. Listing 7.1 zeigt eine beispielhafte Antwort-Nachricht.

```
<FHH_IMCClamWin version="1.0">
  <ClamWin>
    <mainCvd>
      <lastUpdate>9/6/2006</lastUpdate>
      <version>39</version>
      <signatures>58116</signatures>
    </mainCvd>
    <dailyCvd>
      <lastUpdate>20/7/2006</lastUpdate>
      <version>1610</version>
      <signatures>3958</signatures>
    </dailyCvd>
  </ClamWin>
</FHH_IMCClamWin>
```

Listing 7.1: XML-Nachricht mit AV-Parametern des ClamWin-Virens scanners

Nach dem Vergleich mit den Vorgaben – dies geschieht unter Verwendung der Funktionalität des Building-Blocks *Validierung* mit Hilfe einer Policy (siehe Kapitel 3.2.4) – teilt der IMV dem TNC Server sein Prüfungsergebnis mit. Sollte das Ergebnis positiv ausgefallen sein, sendet der IMV innerhalb dieser vierten Runde des TNC-Handshakes zudem noch eine allgemeine Nachricht zurück zum AR. Dieser Nachricht wird ein spezieller *SubType* zugeordnet, der aussagt, dass Anti-Viren-Software gefunden wurde und sich diese in einem aktuellen Zustand befindet. Jedem IMC, der sich für diesen Nachrichten-Typ registriert hat, wird diese Nachricht vom TNC Client zugestellt. So z.B. der *IMCSecurityCenter*, welcher in Kapitel 6 vorgestellt wurde.

Der gesamte Ablauf für diesen Fall soll durch das Sequenzdiagramm in Abbildung 7.5 noch einmal verdeutlicht werden. Zur Vereinfachung wurden die Komponenten TNCC und TNCS weggelassen.

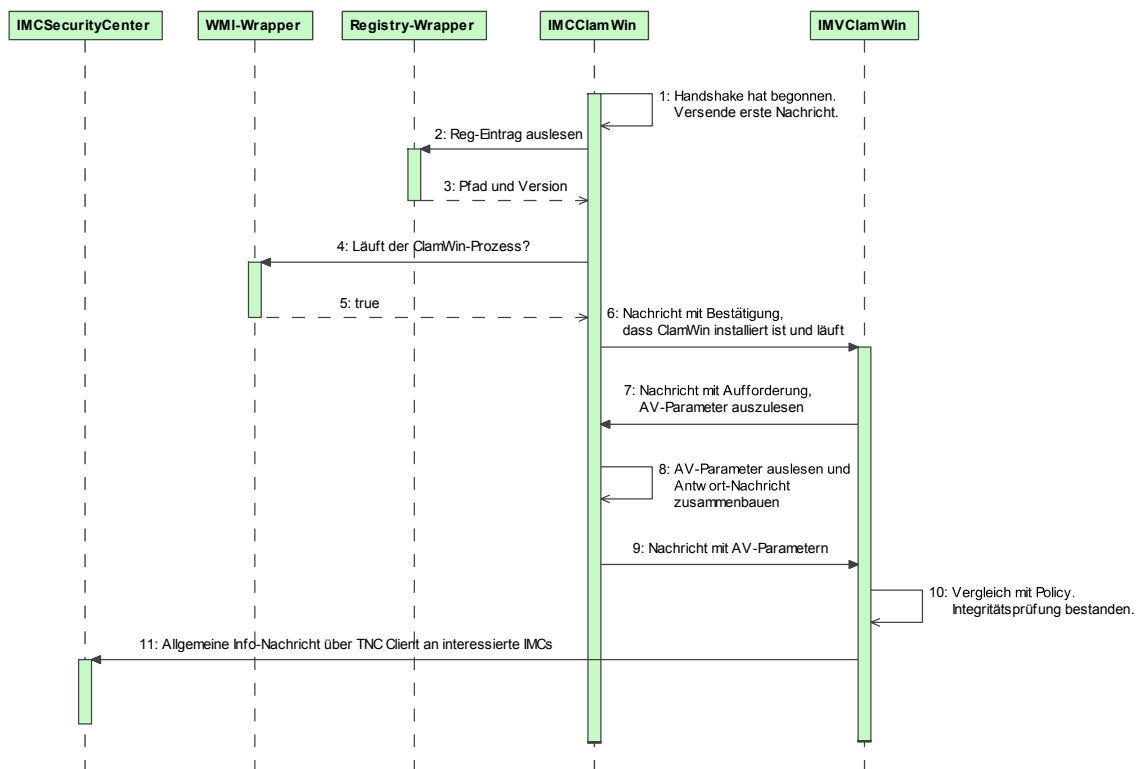


Abbildung 7.5: Nachrichtenaustausch zw. IMCCLamWin und IMVClamWin



## 8 Implementierung und Installation

In diesem Kapitel wird die Vorgehensweise bei der Implementierung der IMCs und IMVs sowie der Hilfsbibliotheken eingegangen. Im Vordergrund steht dabei die verwendete Entwicklungsumgebung und die Konfiguration der einzelnen Projekte. Im Anschluss wird die Installation der TNC-Komponenten vorgestellt.

### 8.1 Entwicklungsumgebung

Für die Entwicklung der TNC-Komponenten wurde Eclipse 3.1<sup>1</sup> in Verbindung mit dem CDT-Plugin<sup>2</sup> verwendet. Bei dem Plugin handelt es sich um ein Open-Source-Projekt von Eclipse. Ein entsprechender C-Compiler wird dabei nicht mit ausgeliefert. Unter Linux wird daher standardmäßig der *Gnu C Compiler* (gcc) für die Kompilierung der Sourcen verwendet. Dieser ist Bestandteil praktisch aller aktuellen Linux-Distributionen. Für die Entwicklung der Client-Komponenten unter Windows XP wird dagegen eine Installation von Cygwin<sup>3</sup> benötigt, welches ein Linux-Betriebssystem simuliert. Auf diese Weise steht der gcc auch unter Windows als Compiler bereit und bietet zudem Zugriff auf die Windows-API. Bei dieser Vorgehensweise muss bei der Installation der TNC-Komponenten allerdings die DLL *cygwin1.dll* mit ausgeliefert werden. Da es sich bei Cygwin aber um ein Open-Source-Projekt handelt, welches unter der GPL<sup>4</sup> steht, stellt die Verwendung dieser Bibliothek keinen Widerspruch zur nicht-funktionalen Anforderung *Open-Source* (siehe Kapitel 3.1.2) dar.

Abbildung 8.1 zeigt den Eclipse-Workspace unter Windows XP zur Entwicklung der Client-Komponenten.

---

<sup>1</sup><http://www.eclipse.org>

<sup>2</sup>C/C++ Development Tools (<http://www.eclipse.org/cdt>)

<sup>3</sup><http://www.cygwin.com>

<sup>4</sup>GNU General Public License

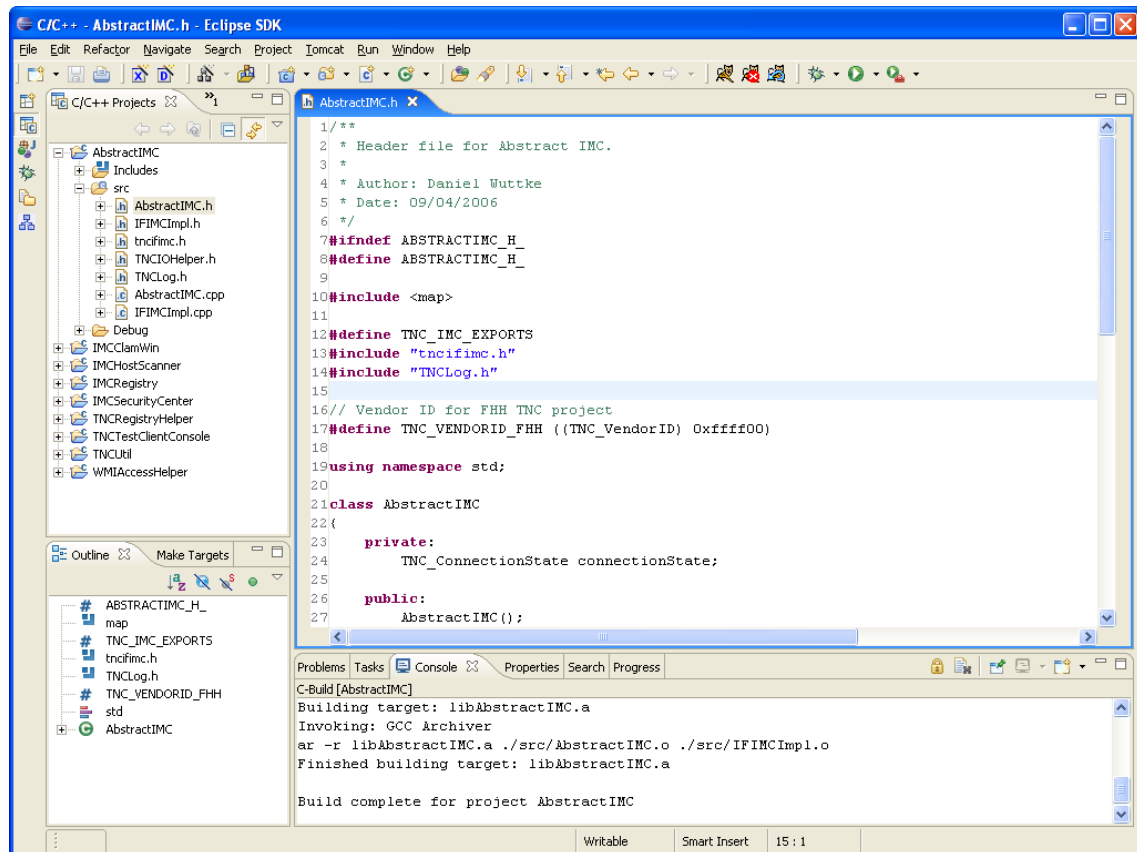


Abbildung 8.1: Eclipse-Workspace

## 8.2 Konfiguration der Projekte

Da sowohl client- als auch server-seitig die gleiche Entwicklungsumgebung verwendet wird, ist die Konfiguration der Projekte in beiden Fällen identisch. Neben ausführbaren Dateien können in C/C++ auch dynamische und statische Bibliotheken erzeugt werden. Dynamische Bibliotheken werden dabei erst zur Laufzeit geladen. Die statischen Bibliotheken müssen dagegen schon während des Linker-Vorgangs zur Verfügung stehen und werden fest in den Output eingebunden. In beiden Fällen werden zur Compiler-Zeit Header-Dateien mit den Deklarationen z.B. der Klassen und Funktionen benötigt. Abbildung 8.2 zeigt den Dialog zur Auswahl der Projekt-Art.

Bei sämtlichen IMC- und IMV-Implementierung handelt es sich um Projekte, die eine dynamische Bibliothek als Output erzeugen. Somit können die IMCs und IMVs dynamisch zur Laufzeit durch den TNC Client bzw. TNC Server, wie von der TNC-Architektur gefor-

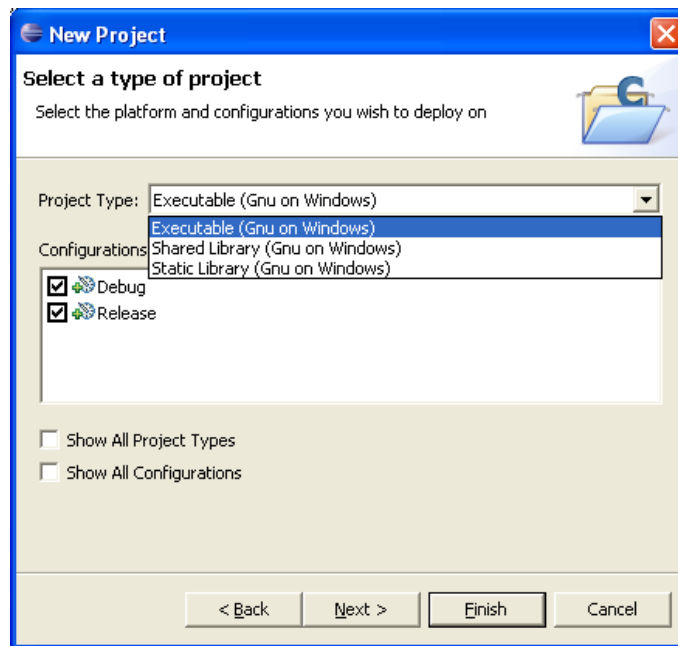


Abbildung 8.2: Auswahl der Projekt-Art

dert, geladen werden. Bei allen weiteren Projekten mit Ausnahme des *TNC-Test-Clients*<sup>5</sup> werden statische Bibliotheken erzeugt, die von anderen Projekten eingebunden werden können. An erster Stelle steht dabei der abstrakte IMC bzw. IMV. Daneben existieren aber noch weitere Projekte, die die in Kapitel 3.2 vorgestellten Building-Blocks repräsentieren. Tabelle 8.1 fasst sämtliche Projekte für den Client, die im Zuge dieser Masterarbeit entstanden sind, zusammen. In der Regel existiert für jedes dieser Projekte ein entsprechendes Pendant auf Server-Seite, so dass diese zur Vereinfachung weggelassen wurden. Abweichungen sind entsprechend vermerkt.

## 8.3 Spezielle Aspekte der Implementierung

### 8.3.1 Threadsicherheit

Wie in Kapitel 3.1.1 beschrieben wurde, ist eine funktionale Anforderung die *Threadsicherheit* auf Server-Seite. Es muss damit gerechnet werden, dass Funktionen der IF-IMV-Schnittstelle in Form der Klasse IFIMVImpl durch den TNC Server parallel aufgerufen werden. Da es pro Verbindung zu einem Client von jedem IMV eine eigene Instanz gibt,

<sup>5</sup>Einfache Konsolen-Anwendung zum Testen einzelner IMCs

Name	Projekt-Art	Beschreibung
AbstractIMC	Statische Bibliothek	Die Basisimplementierung für einen konkreten IMC. Muss in jedem IMC-Projekt eingebunden werden. Building-Block: <i>Abstrakter IMC</i>
IMCclamWin	Dynamische Bibliothek	Konkrete IMC-Implementierung (siehe Kapitel 7). Wird zur Laufzeit vom TNC Client geladen. Abhängigkeiten: TNCUtil, TNCRegistryHelper, WMIAccessHelper
IMCHostScanner	Dynamische Bibliothek	Konkrete IMC-Implementierung (siehe Kapitel 5). Wird zur Laufzeit vom TNC Client geladen. Abhängigkeiten: TNCUtil
IMCRegistry	Dynamische Bibliothek	Konkrete IMC-Implementierung (siehe Kapitel 4). Wird zur Laufzeit vom TNC Client geladen. Abhängigkeiten: TNCUtil, TNCRegistryHelper
IMCSecurityCenter	Dynamische Bibliothek	Konkrete IMC-Implementierung (siehe Kapitel 6). Wird zur Laufzeit vom TNC Client geladen. Abhängigkeiten: TNCUtil, TNCRegistryHelper, WMIAccessHelper
TNCRegistryHelper	Statische Bibliothek	Hilfs-Bibliothek, die die Windows-API für den Zugriff auf die Registrierung kapselt (siehe Kapitel 3.2.5). Steht ausschließlich unter Windows zur Verfügung und muss bei Bedarf ins IMC-Projekt eingebunden werden. Building-Block: <i>Registry-Wrapper</i>
TNCUtil	Statische Bibliothek	Hilfs-Bibliothek, die Funktionen für den Zugriff auf Dateien, das Logging und das Parsen und Erzeugen von XML-Strukturen bereitstellt. Building-Blocks: <i>Logging, XML, Properties</i>
WMIAccessHelper	Statische Bibliothek	Hilfs-Bibliothek, die die Windows-API für den Zugriff auf WMI kapselt (siehe Kapitel 3.2.5). Steht ausschließlich unter Windows zur Verfügung und muss bei Bedarf ins IMC-Projekt eingebunden werden. Building-Block: <i>WMI-Wrapper</i>

Tabelle 8.1: Eclipse-Projekte für Windows-Client

ist die Verwaltung der *Connection States* und Validierungsergebnisse bereits geschützt. Allerdings verwendet die Klasse *IFIMVImpl* Vektoren zum Speichern der Referenzen auf die IMVs. Da sämtliche Container-Klassen in C++ nicht threadsicher sind, muss der schreibende Zugriff auf die Vektoren geschützt werden. Eine Möglichkeit zur Realisierung stellt die Verwendung von Semaphoren<sup>6</sup> dar. Damit wird sichergestellt, dass stets nur ein Thread gleichzeitig auf einen Vector zugreifen darf. Die Umsetzung in C++ erfolgte nach den Angaben aus [Sun05].

### 8.3.2 WMI-Wrapper

Da Cygwin die API für den Zugriff auf WMI nicht bereitstellt, war das Kompilieren mit dem gcc nicht möglich. Daher wurde unter Verwendung der *Microsoft Visual C++ Express Edition* eine separate DLL für den Zugriff auf WMI entwickelt. Diese wird bei Bedarf zur Laufzeit geladen und wird u.a. vom *IMCSecurityCenter* genutzt (siehe Kapitel 6). Der Zugriff auf die DLL wird durch den *WMIAccessHelper* gekapselt.

### 8.3.3 Logging-API

Die Funktionen der in Kapitel 3.2.3 vorgestellten Logging-API verfügen über variable Parameterlisten. Wie bei der C-Funktion `printf()` wird als erster Parameter ein Formatstring erwartet. Dieser String beinhaltet neben den auszugebenden Zeichen auch Steuersequenzen. Für jede Steuersequenz muss ein zusätzlicher Parameter beim Funktionsaufruf übergeben werden (siehe auch [Wac01, S. 274f]). Ein Beispiel-Quellcode mit Log-Ausgaben befindet sich im Anhang auf Seite 89.

---

<sup>6</sup>siehe auch [http://de.wikipedia.org/wiki/Semaphor\\_%28Informatik%29](http://de.wikipedia.org/wiki/Semaphor_%28Informatik%29)

## 8.4 Weitere Abhängigkeiten

Wie bereits in Abschnitt 8.1 erwähnt, benötigen alle client-seitigen TNC-Komponenten eine Cygwin-DLL zur Laufzeit. Daneben wird für die Erstellung und das Parsen von XML-Dokumenten der Xerces-Parser von Apache verwendet. Da der Austausch von Informationen zwischen IMCs und IMVs stets über XML-Nachrichten abläuft, muss die Xerces-Bibliothek in jedem IMC- und IMV-Projekt eingebunden werden. Zusätzlich benötigt der Parser eine DLL bzw. ein SO zur Laufzeit.

## 8.5 Installation der TNC-Komponenten

### 8.5.1 Notwendigkeit eines Installationstools

Für die Installation der TNC-Komponenten unter Windows – dies umfasst sowohl die IMCs als auch den im Zusammenhang mit der Masterarbeit von Martin Schmiedel [Sch06] entwickelten TNC Client – wurde eine separate Anwendung mit grafischer Benutzeroberfläche entworfen. Dies ist sinnvoll, da eine Vielzahl von Client-PCs zu erwarten ist und eine entsprechende manuelle Administration mit hohem Aufwand verbunden wäre. Da bei der Installation Zugriffe auf das Dateisystem und die Registrierung getätigt werden müssen und eine Portierung des Quellcodes nach Linux nicht notwendig ist<sup>7</sup>, bot sich die Verwendung von C# für die Entwicklung des Installationstools an. C# bietet einfache APIs für den Zugriff auf das Dateisystem und die Windows-Registrierung an. Zudem ist die Erstellung schlanker und flexibler GUIs möglich.

### 8.5.2 Entwicklungsumgebung

Für die Entwicklung von C#-Anwendungen stellt Microsoft das frei verfügbare .NET-Framework zur Verfügung. Neben der Laufzeitumgebung *.NET Framework 2.0 Redistributable (x86)*<sup>8</sup> wird für das Erzeugen von C#-Anwendungen das *.NET Framework SDK*

---

<sup>7</sup>Da i.d.R. nur ein PDP existiert und sich der Administrationsaufwand daher auf einen einzelnen Rechner beschränkt, wurde auf ein Installationstool für die TNC-Komponenten auf Server-Seite verzichtet.

<sup>8</sup><http://www.microsoft.com/downloads/details.aspx?displaylang=de&FamilyID=0856eacb-4362-4b0d-8edd-aab15c5e04f5>

2.0 (x86)<sup>9</sup> benötigt. Als Entwicklungsumgebung dient die frei verfügbare IDE *SharpDevelop*<sup>10</sup> in der Version 2.0.

### 8.5.3 Anforderungen an das Installationstool

Die Anforderungen an das Installationstool lassen sich wie folgt zusammenfassen:

**Automatisierte Installation** Nach der Auswahl des Zielordners und der zu installierenden Komponenten muss die Installation vollkommen automatisiert ablaufen; d.h. Verzeichnisse müssen erstellt, Dateien kopiert und Registrierungseinträge erzeugt werden.

**Automatisierte Deinstallation** Ein sauberes Entfernen der installierten TNC-Komponenten muss möglich sein. Dazu gehören das Löschen der Dateien und Ordner sowie das Entfernen der Registrierungseinträge.

**Erweiterbarkeit** Das Installationstool muss leicht um neue Komponenten erweitert werden können.

**Einfache Benutzerführung** Der Installationsprozess muss für den Anwender einfach durchführbar und nachvollziehbar sein.

### 8.5.4 Ablauf einer Installation

Zu Beginn des Installationsprozesses muss der Anwender den Installationsordner auswählen. Sofern zuvor bereits eine Installation von TNC-Komponenten durchgeführt wurde und Einträge in der Registrierung vorhanden sind, wird das Eingabefeld für das Zielverzeichnis vorbelegt. Die Option *Uninstall* steht ebenfalls nur zur Verfügung, wenn eine vorhandene Installation erkannt wurde. Abbildung 8.3 zeigt die Einstiegsmaske des Installationstools.

Auf der Folgemaske kann der Anwender die zu installierenden TNC-Komponenten auswählen. Als erste Komponente ist der TNC Client von Martin Schmiedel [Sch06] aufgeführt. Wichtige Laufzeitbibliotheken wie die Cygwin-DLL und Xerces wurden innerhalb

---

<sup>9</sup><http://www.microsoft.com/downloads/details.aspx?displaylang=de&FamilyID=FE6F2099-B7B4-4F47-A244-C96D69C35DEC>

<sup>10</sup><http://www.icsharpcode.net/OpenSource/SD>

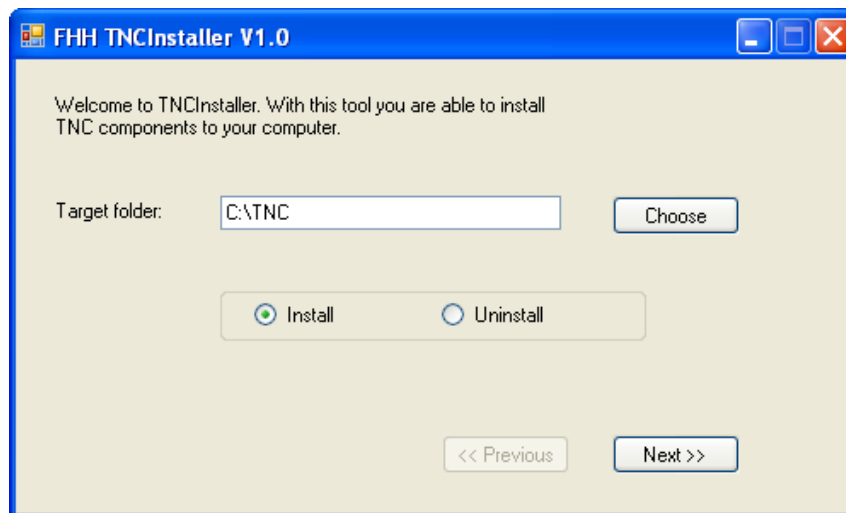


Abbildung 8.3: TNCInstaller – Auswahl der Zielordners

der TNC-Komponente *CygwinLibs* zusammengefasst. Da beide unverzichtbare Bestandteile des AR sind, müssen diese Komponenten in jedem Fall installiert werden. Welche konkreten IMCs auf den Client kopiert und in der Registrierung vermerkt werden sollen, bleibt aber dem Anwender überlassen (siehe auch Abbildung 8.4).

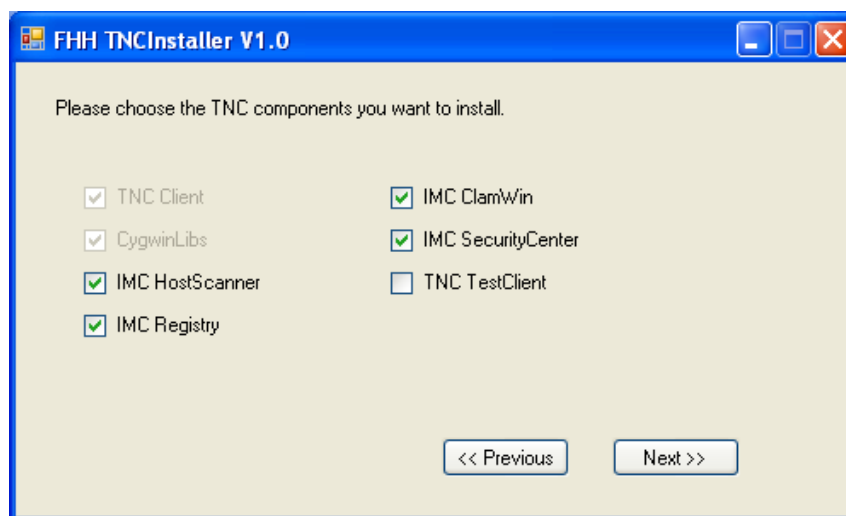


Abbildung 8.4: TNCInstaller – Auswahl der zu installierenden Komponenten

Die letzte Seite zeigt dem Anwender ein Installationsprotokoll, welche Aktivitäten im Einzelnen durchgeführt werden. Dazu gehört das Erstellen eines Verzeichnisses, das Kopieren einer Datei und das Anlegen eines Registrierungseintrags. Nach dem Abschluss der Installation wird eine entsprechende Hinweismeldung eingeblendet. Abbildung 8.5 zeigt ein Beispiel.



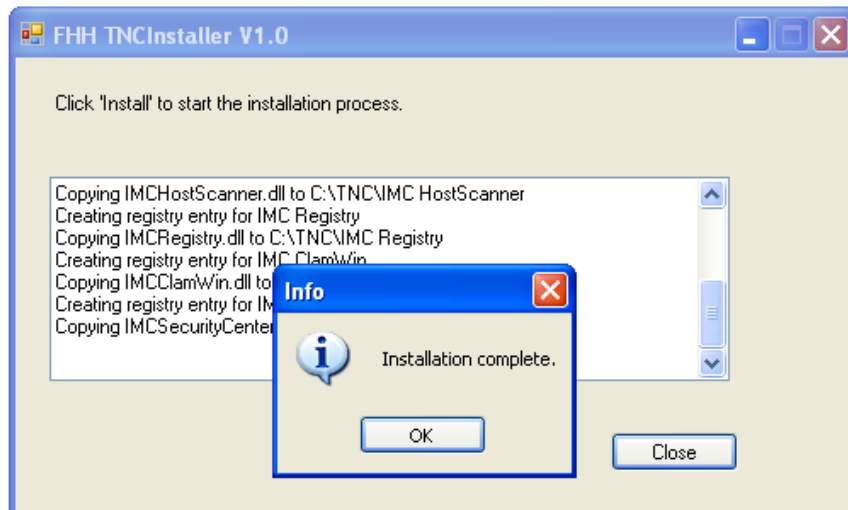


Abbildung 8.5: TNCInstaller – Abschluss der Installation

## 8.6 Testumgebung

Zur Überprüfung der Interaktionsfähigkeit zwischen TNC Client und IMCs bzw. TNC Server und IMVs wurde eine Testumgebung aufgebaut, die aus zwei PCs und einem Switch vom Typ *HP Procurve 5368xl* besteht. Abbildung 8.6 veranschaulicht den Aufbau.

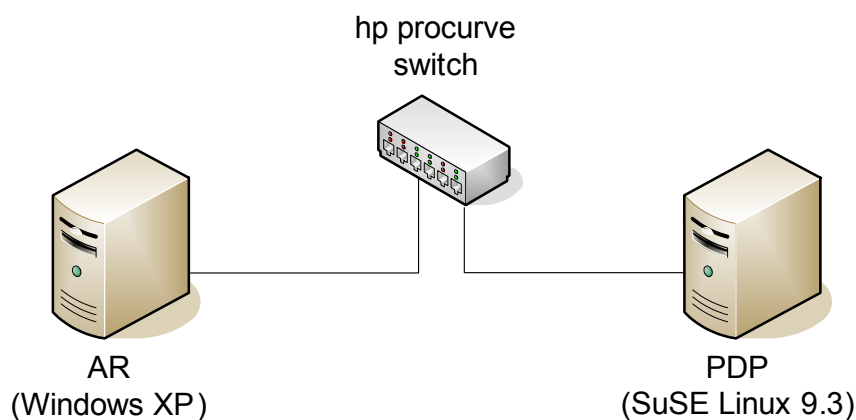


Abbildung 8.6: Aufbau Testumgebung

Beim AR handelt es sich um einen Windows-Rechner mit installiertem Service-Pack 2. Mit Hilfe des Installationstools *TNCInstaller* (siehe Abschnitt 8.5) wurden die client-seitigen TNC-Komponenten auf dem AR installiert. Da sämtliche benötigte Laufzeit-Bibliotheken mit ausgeliefert werden, ist eine Installation von Cygwin auf dem Testrechner nicht notwendig.

Server-seitig wurde die Installation des TNC Servers sowie der IMVs manuell vorgenommen. Dazu wurde u.a. eine Konfigurationsdatei angelegt, in der sämtliche vorhandene IMVs eingetragen werden müssen (vgl. [TCG06c, S. 42f]). Die Konfigurations- und Policy-Dateien der IMVs wurden unter `/etc/tnc` abgelegt. Dieses Verzeichnis ist im Quellcode der IMVs fest angegeben.

Weitere Komponenten, die für die Kommunikation zwischen TNCC und TNCS über den Switch von Bedeutung sind und auf dem AR bzw. PDP installiert werden müssen, beschreibt Martin Schmiedel in seiner Masterarbeit [Sch06].

## 9 Fazit und Ausblick

Im Rahmen dieser Arbeit entstand eine API für die Entwicklung von IMCs und IMVs auf Basis der TNC-Architektur. Wie die Implementierung der vier Beispiel-Komponenten gezeigt hat, können mit Hilfe dieser Basisarchitektur auf einfache Weise neue IMC/IMV-Paare entwickelt werden. Dabei kann sich der Entwickler fast ausschließlich auf die reine Fachlichkeit konzentrieren, ohne sich z.B. Gedanken über den Nachrichtenaustausch zwischen IMCs und IMVs machen zu müssen. Auch die plattformabhängige Implementierung der Schnittstelle für eine dynamische Bibliothek wird dem Entwickler durch die API komplett abgenommen. Weiterhin unterstützt ihn die Basisarchitektur bei der Validierung der ausgelesenen bzw. gemessenen Sicherheitsaspekte des Clients.

Umfangreiche Tests innerhalb der Testumgebung haben bewiesen, dass die Integration der IMCs bzw. IMVs in den TNC Client bzw. TNC Server problemlos möglich war. Dies zeigt auch, dass bei einer genauen Einhaltung der in der TNC-Architektur beschriebenen API eine Interoperabilität mit Produkten anderer Hersteller gewährleistet sein sollte.

Dennoch bietet die entstandene API großes Erweiterungspotential für zukünftige Projekte. Aus Zeitgründen konnten nicht alle Aspekte der TNC-Architektur umgesetzt werden. Somit unterstützt die API zur Zeit noch keine Isolation von Clients in separate Teilnetze, um etwa Updates durchzuführen und sich auf diese Weise zu „sanieren“. Auch ist noch keine Wiederholung des TNC-Handshakes möglich. Die dafür vorgesehenen Funktionen innerhalb der TNC-Architektur sind optional und in der API aktuell nicht implementiert.

Ein weiteres wichtiges Merkmal, das bisher unberücksichtigt geblieben ist, ist die Integrität der TNC-Komponenten – speziell die der IMCs – selbst. Nur wenn die Software an sich vertrauenswürdig ist und nicht von Dritten ausgetauscht oder manipuliert werden kann, macht ein Integritätstest des Clientrechners Sinn. Innerhalb der TCG existiert mit der *TPM Work Group* eine weitere Gruppe, die sich genau dieser Problemstellung angenommen hat. Das *Trusted Platform Module* (TPM) ist ein Hardwaremodul, welches sich auf dem Mainboard eines Rechners befindet. Innerhalb des TPM existieren geschützte

Bereiche, in denen u.a. kryptographische Schlüssel und Informationen zur Plattformintegrität abgelegt werden können. Über spezielle Kommandos kann auf diesen geschützten Speicherbereich zugegriffen werden.

Die TNC-Architektur beschreibt in [TCG06a, S. 26f], wie mit Hilfe eines TPM die Vertrauenswürdigkeit von IMCs und TNC Clients sichergestellt werden kann. Wäre es möglich, ein TPM in die API zur Entwicklung von IMCs zu integrieren, könnten diese keine falschen Zustände mehr vortäuschen. Dies wäre ein weiterer wichtiger Schritt zur Sicherstellung der Client-Integrität innerhalb lokaler Netze.

# Anhang A

## XML-Schema-Beschreibungen

### A.1 XML-Schema für IMVRegistry-Konfiguration

Die Listings A.1 und A.2 zeigen das Schema, das die Syntax für die XML-Konfiguration des *IMVRegistry* beschreibt (vgl. Kapitel 4.3).

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema elementFormDefault="qualified"
  targetNamespace="http://www.inform.fh-hannover.de/IMVRegistry"
  xmlns="http://www.inform.fh-hannover.de/IMVRegistry"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="FHH_IMVRegistry">
    <xsd:complexType>
      <xsd:sequence maxOccurs="unbounded" minOccurs="1">
        <xsd:element ref="regEntry"/>
      </xsd:sequence>
      <xsd:attribute name="version" type="xsd:string" use="required"/>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="regEntry">
    <xsd:complexType>
      <xsd:sequence maxOccurs="1" minOccurs="1">
        <xsd:element ref="name"/>
        <xsd:element name="key" type="xsd:string"/>
        <xsd:choice>
          <xsd:element ref="regEntry" maxOccurs="unbounded"/>
          <xsd:element ref="values" maxOccurs="1"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  [...]
```

Listing A.1: XML-Schema für die Konfiguration des IMVRegistry (Teil 1)

```

[...]

<xsd:element name="name" type="xsd:string"/>

<xsd:element name="values">
  <xsd:complexType>
    <xsd:choice maxOccurs="unbounded" minOccurs="1">
      <xsd:element name="regString" type="regType"/>
      <xsd:element name="regNumber" type="regType"/>
      <xsd:element name="regBinary" type="regType"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>

<xsd:complexType name="regType">
  <xsd:sequence>
    <xsd:element ref="name" minOccurs="0"/>
    <xsd:element name="value" type="xsd:string"/>
  </xsd:sequence>
  <xsd:attribute name="type" type="xsd:string" use="optional"/>
</xsd:complexType>

</xsd:schema>

```

Listing A.2: XML-Schema für die Konfiguration des IMVRegistry (Teil 2)

## A.2 XML-Schema für IMVHostScanner-Konfiguration

Die Listings A.3 und A.4 zeigen das Schema, das die Syntax für die XML-Konfiguration des *IMVHostScanner* beschreibt (vgl. Kapitel 5.3).

```

<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema elementFormDefault="qualified"
  targetNamespace="http://www.inform.fh-hannover.de/IMVHostScanner"
  xmlns="http://www.inform.fh-hannover.de/IMVHostScanner"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="FHH_IMVHostScanner">
    <xsd:complexType>
      <xsd:sequence maxOccurs="1" minOccurs="1">
        <xsd:element ref="ports"/>
      </xsd:sequence>
      <xsd:attribute name="version" type="xsd:string" use="required"/>
    </xsd:complexType>
  </xsd:element>

  [...]

```

Listing A.3: XML-Schema für die Konfiguration des IMVHostScanner (Teil 1)

```
[...]

<xsd:element name="ports">
  <xsd:complexType>
    <xsd:sequence maxOccurs="unbounded" minOccurs="1">
      <xsd:element ref="port"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="port">
  <xsd:complexType>
    <xsd:attribute name="number" type="xsd:decimal" use="required"/>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```

Listing A.4: XML-Schema für die Konfiguration des IMVHostScanner (Teil 2)

## A.3 XML-Schema für Policies

Das Listing A.5 zeigt das Schema, welches die Grammatik einer Policy beschreibt. Policies werden für die Validierung innerhalb von IMVs genutzt (siehe Kapitel 3.2.4).

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema elementFormDefault="qualified"
  targetNamespace="http://www.inform.fh-hannover.de/TNC-IMV-Policy"
  xmlns="http://www.inform.fh-hannover.de/TNC-IMV-Policy"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="Policy">
    <xsd:complexType>
      <xsd:sequence maxOccurs="1" minOccurs="1">
        <xsd:element ref="ExactlyOne"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="ExactlyOne">
    <xsd:complexType>
      <xsd:sequence maxOccurs="unbounded" minOccurs="1">
        <xsd:element ref="All"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="All">
    <xsd:complexType>
      <xsd:sequence maxOccurs="unbounded" minOccurs="1">
        <xs:any namespace="##any" processContents="lax"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

Listing A.5: XML-Schema für Policies



# Anhang B

## XML-Konfigurationen

### B.1 XML-Konfiguration für IMVRegistry

Die Listings B.1 und B.2 zeigen die in Kapitel 4.4 vorgestellte Basiskonfiguration für den *IMVRegistry*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<FHH_IMVRegistry version="1.0">
  <regEntry>
    <name>WindowsInfo</name>
    <key>HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion</key>
    <values>
      <regString>
        <name>ProductName</name>
        <value>ProductName</value>
      </regString>
      <regString>
        <name>WindowsVersion</name>
        <value>CurrentVersion</value>
      </regString>
      <regString>
        <name>ServicePackLevel</name>
        <value>CSDVersion</value>
      </regString>
    </values>
  </regEntry>
  <regEntry>
    <name>WindowsServices</name>
    <key>HKLM\SYSTEM\CurrentControlSet\Services</key>
    <regEntry>
      <name>SecurityCenter</name>
      <key>wscsvc</key>
    </regEntry>
  </regEntry>
  [...]
</FHH_IMVRegistry>
```

Listing B.1: Basiskonfiguration des IMVRegistry (Teil 1)

```
[...]

    <values>
      <regNumber type="attribute">
        <value>start</value>
      </regNumber>
    </values>
  </regEntry>
  <regEntry>
    <name>NetMessenger</name>
    <key>Messenger</key>
    <values>
      <regNumber type="attribute">
        <value>start</value>
      </regNumber>
    </values>
  </regEntry>
</regEntry>
<regEntry>
  <name>WindowsUpdates</name>
  <key>HKLM\SOFTWARE\Microsoft\Updates\Windows XP\SP3</key>
  <regEntry>
    <key>*</key>
    <values>
      <regString type="attribute">
        <name>type</name>
        <value>Type</value>
      </regString>
    </values>
  </regEntry>
</regEntry>
<regEntry>
  <name>WindowsScriptingHost</name>
  <key>HKLM\SOFTWARE\Microsoft\Windows Script Host\Settings</key>
  <values>
    <regNumber type="attribute">
      <name>enabled</name>
      <value default="1">Enabled</value>
    </regNumber>
  </values>
</regEntry>
<regEntry>
  <name>AutoLogon</name>
  <key>HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\WinLogon</key>
  <values>
    <regString type="attribute">
      <name>enabled</name>
      <value default="0">AutoAdminLogon</value>
    </regString>
  </values>
</regEntry>
</FHH_IMVRegistry>
```

Listing B.2: Basiskonfiguration des IMVRegistry (Teil 2)

## B.2 XML-Konfiguration für IMVHostScanner

Das Listing B.3 zeigt die in Kapitel 5.4 vorgestellte Basiskonfiguration für den *IMV-HostScanner*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<FHH_IMVHostScanner version="1.0">
  <ports>
    <port number="21" />      <!-- ftp -->
    <port number="22" />      <!-- ssh -->
    <port number="25" />      <!-- smtp -->
    <port number="42" />      <!-- nameserver -->
    <port number="67" />      <!-- bootps / dhcp -->
    <port number="135" />     <!-- RPC Endpoint Mapper -->
  </ports>
</FHH_IMVHostScanner>
```

Listing B.3: Basiskonfiguration des IMVHostScanner



# Anhang C

## Policies für Integrity Measurement Verifiers

### C.1 Policy für IMVRegistry

Das Listing C.1 zeigt die Policy für die in Kapitel 4.4 vorgestellte Basiskonfiguration des *IMVRegistry*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<tncp:Policy xmlns:tncp="http://www.inform.fh-hannover.de/TNC-IMV-Policy">
  <tncp:ExactlyOne>
    <tncp:All>
      <WindowsInfo>
        <ProductName>Microsoft Windows XP</ProductName>
        <WindowsVersion>5.1</WindowsVersion>
        <ServicePackLevel>Service Pack 2</ServicePackLevel>
      </WindowsInfo>
      <WindowsServices>
        <SecurityCenter start="2" /> <!-- automatic -->
        <UpdateService start="2" /> <!-- automatic -->
        <NetMessenger start="4" /> <!-- deactivated -->
      </WindowsServices>
      <WindowsUpdates>
        <KB888302 type="Update" />
        <KB896688 type="Update" />
        <KB902400 type="Update" />
        <KB908531 type="Update" />
      </WindowsUpdates>
      <WindowsScriptingHost enabled="0" /> <!-- disabled -->
      <AutoLogon enabled="0" /> <!-- disabled -->
    </tncp:All>
  </tncp:ExactlyOne>
</tncp:Policy>
```

Listing C.1: Policy für IMVRegistry

## C.2 Policy für IMVHostScanner

Das Listing C.2 zeigt die Policy für die in Kapitel 5.4 vorgestellte Basiskonfiguration des *IMVHostScanner*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<tncp:Policy xmlns:tncp="http://www.inform.fh-hannover.de/TNC-IMV-Policy">
  <tncp:ExactlyOne>
    <tncp:All>
      <port number="21" open="false" />      <!-- ftp -->
      <port number="22" open="false" />      <!-- ssh -->
      <port number="25" open="false" />      <!-- smtp -->
      <port number="42" open="false" />      <!-- nameserver -->
      <port number="67" open="false" />      <!-- bootps / dhcp -->
      <port number="135" open="false" />     <!-- RPC Endpoint Mapper -->
    </tncp:All>
  </tncp:ExactlyOne>
</tncp:Policy>
```

Listing C.2: Policy für IMVHostScanner

## C.3 Policy für IMVClamWin

Das Listing C.3 zeigt eine beispielhafte Policy für die in Kapitel 7 vorgestellte Integrity-Measurement-Komponente für den *ClamWin*-Virens scanner.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<tncp:Policy xmlns:tncp="http://www.inform.fh-hannover.de/TNC-IMV-Policy">
  <tncp:ExactlyOne>
    <tncp:All>
      <mainCvd>
        <lastUpdate tncp:type="date" tncp:compare="ge">9/6/2006</lastUpdate>
        <version tncp:type="number" tncp:compare="ge">39</version>
        <signatures tncp:type="number" tncp:compare="ge">58116</signatures>
      </mainCvd>
      <dailyCvd>
        <lastUpdate tncp:type="date" tncp:compare="ge">12/7/2006</lastUpdate>
        <version tncp:type="number" tncp:compare="ge">1280</version>
        <signatures tncp:type="number" tncp:compare="ge">2815</signatures>
      </dailyCvd>
    </tncp:All>
  </tncp:ExactlyOne>
</tncp:Policy>
```

Listing C.3: Policy für IMVClamWin

## C.4 Policies für IMVSecurityCenter

Die Listings C.4 und C.5 zeigen beispielhafte Policies für die Firewall- und Anti-Virus-Parameter des in Kapitel 6 vorgestellten *IMVSecurityCenter*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<tncp:Policy xmlns:tncp="http://www.inform.fh-hannover.de/TNC-IMV-Policy">
  <tncp:ExactlyOne>
    <tncp:All>
      <FirewallProduct>
        <productName>Kerio</productName>
        <version>4.2.2 T</version>
        <enabled>true</enabled>
      </FirewallProduct>
    </tncp:All>
    <tncp:All>
      <FirewallProduct>
        <productName>Internal Windows Firewall</productName>
        <enabled>true</enabled>
      </FirewallProduct>
    </tncp:All>
  </tncp:ExactlyOne>
</tncp:Policy>
```

Listing C.4: Policy für IMVSecurityCenter (FW-Parameter)

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<tncp:Policy xmlns:tncp="http://www.inform.fh-hannover.de/TNC-IMV-Policy">
  <tncp:ExactlyOne>
    <tncp:All>
      <AntiVirusProduct>
        <productName>AntiVir</productName>
        <version>*</version>
        <upToDate>true</upToDate>
        <enabled>true</enabled>
      </AntiVirusProduct>
    </tncp:All>
    <tncp:All>
      <AntiVirusProduct>
        <productName>Sophos Anti-Virus</productName>
        <version>*</version>
        <upToDate>true</upToDate>
        <enabled>true</enabled>
      </AntiVirusProduct>
    </tncp:All>
  </tncp:ExactlyOne>
</tncp:Policy>
```

Listing C.5: Policy für IMVSecurityCenter (AV-Parameter)





# Anhang D

## Quellcodes

### D.1 Quellcode mit Log-Ausgaben

Listing D.1 zeigt einen Beispiel-Quellcode mit Log-Ausgaben. Dabei wird die in Kapitel 3.2.3 vorgestellte Logging-API verwendet.

```
/**
 * Abstract IMC implementation.
 *
 * Author: Daniel Wuttke
 * Date: 09/04/2006
 */
#include "AbstractIMC.h"

// Classname for debugging and logging
#define CLASSNAME "AbstractIMC"

AbstractIMC::AbstractIMC() {
    TNLog::getLog(CLASSNAME, "Constructor");
}

AbstractIMC::~AbstractIMC() {
    TNLog::getLog(CLASSNAME, "~");
}

/**
 * Sets a new connection state.
 */
void AbstractIMC::setConnectionState(TNC_ConnectionState newState) {
    TNLog &log = TNLog::getLog(CLASSNAME, "setConnectionState");

    log.debug("New connection state: %d", newState);
    connectionState = newState;
}

[...]
```

Listing D.1: Beispiel-Quellcode mit Log-Ausgaben

## D.2 Quellcode der TNC-Komponenten

Der gesamte Quellcode der im Rahmen dieses Projekts entstandenen TNC-Komponenten ist auf der beiliegenden CD-ROM zu finden. Enthalten ist auch eine lauffähige Version der Entwicklungsumgebung Eclipse 3.1.0 mit einem vorkonfigurierten Workspace.

Die CD-ROM enthält eine README-Datei mit einer Auflistung der Inhalte und einer Installationsanleitung für die beigefügte Eclipse-Version.

# Literaturverzeichnis

- [Aur05] AURAND, ANDREAS: *LAN-Sicherheit - Schwachstellen, Angriffe und Schutzmechanismen in lokalen Netzwerken - am Beispiel von Cisco Catalyst Switches*. dpunkt.verlag, Heidelberg, 2005. 1, 11, 46
- [Dra03] DRAYTON, PETER UND ALBAHARI, BEN: *C# in a Nutshell*. O'Reilly Verlag, Köln, 2003.
- [Lou02] LOUIS, DIRK UND STRASSER, SHINJA: *C# in 21 Tagen*. Markt + Technik Verlag, München, 2002.
- [Mic04] MICROSOFT CORPORATION: *Windows Security Center – Managing the State of Security*, Published: September 29, 2004. <http://www.microsoft.com/windowsxp/sp2/wsoverview.mspx>. 49
- [Pet99] PETZOLD, CHARLES: *Windows-Programmierung - Das Entwicklerhandbuch zur WIN32-API*. Microsoft Press Deutschland, 1999.
- [Red03] RED HAT, INC.: *Cygwin User's Guide*, 2003.
- [Sch06] SCHMIEDEL, MARTIN: *Entwicklung einer Client- / Server-basierten Software für die Prüfung der Vertrauenswürdigkeit von Endgeräten*. Masterarbeit, 2006. FH Hannover / Fachbereich Informatik. 3, 8, 10, 11, 12, 30, 31, 70, 71, 74
- [Sun05] SUN MICROSYSTEMS, INC.: *Multithreaded Programming Guide*, 2005. <http://docs.sun.com/app/docs/doc/816-5137>. 69
- [TCG05a] TCG TRUSTED NETWORK CONNECT: *TNC Architecture for Interoperability*, 3 May 2005. Specification Version 1.0 Revision 4.
- [TCG05b] TCG TRUSTED NETWORK CONNECT: *TNC IF-IMC*, 3 May 2005. Specification Version 1.0 Revision 3. 9

- [TCG05c] TCG TRUSTED NETWORK CONNECT: *TNC IF-IMV*, 3 May 2005. Specification Version 1.0 Revision 3. 9
- [TCG06a] TCG TRUSTED NETWORK CONNECT: *TNC Architecture for Interoperability*, 1 May 2006. Specification Version 1.1 Revision 2. 6, 76
- [TCG06b] TCG TRUSTED NETWORK CONNECT: *TNC IF-IMC*, 1 May 2006. Specification Version 1.1 Revision 5. 9, 19
- [TCG06c] TCG TRUSTED NETWORK CONNECT: *TNC IF-IMV*, 1 May 2006. Specification Version 1.1 Revision 5. 9, 10, 74
- [Wac01] WACHTLER, KLAUS: *0 ist false, alles andere true - Eine Einführung in die Programmierung mit ANSI-C*, Oktober 2001. 69
- [Wil97] WILLMS, ANDRÉ: *C++ Programmierung - Programmierung, Programmier-technik, Datenorganisation*. Addison-Wesley, Bonn, 1997.

# Abbildungsverzeichnis

2.1	TNC-Architektur . . . . .	6
2.2	Ablauf einer Integritätsprüfung mit zwei IMC/IMV-Paaren . . . . .	13
3.1	Architekturübersicht IMC-API . . . . .	17
3.2	Architekturübersicht IMV-API . . . . .	18
3.3	Abstrakter IMC . . . . .	19
3.4	Abstrakter IMV . . . . .	21
3.5	Logging-API . . . . .	22
3.6	Registry- und WMI-Wrapper-API . . . . .	27
3.7	Ablauf einer Integritätsprüfung mit der API . . . . .	29
4.1	Beispiel für einen Registrierungseintrag . . . . .	36
4.2	Nachrichtenaustausch zw. IMCRegistry und IMVRegistry . . . . .	40
5.1	Nachrichtenaustausch zw. IMCHostScanner und IMVHostScanner . . . . .	47
6.1	WMI Object Browser . . . . .	50
6.2	WMI-Abfrage zur Ermittlung des Status der Windows-Firewall . . . . .	53
6.3	Entscheidungen im Ablauf des IMCSecurityCenter . . . . .	55
7.1	ClamWin-Installationsorder als Registrierungseintrag . . . . .	59
7.2	Hinweis zu verfügbarem ClamWin-Update . . . . .	59
7.3	Abfrage des Prozess-Status über WMI (Prozess läuft) . . . . .	60
7.4	Abfrage des Prozess-Status über WMI (Prozess läuft nicht) . . . . .	60
7.5	Nachrichtenaustausch zw. IMCCLamWin und IMVClamWin . . . . .	64
8.1	Eclipse-Workspace . . . . .	66
8.2	Auswahl der Projekt-Art . . . . .	67
8.3	TNCInstaller – Auswahl der Zielordners . . . . .	72
8.4	TNCInstaller – Auswahl der zu installierenden Komponenten . . . . .	72

8.5	TNCInstaller – Abschluss der Installation . . . . .	73
8.6	Aufbau Testumgebung . . . . .	73

# Tabellenverzeichnis

3.1	Verschiedene Inhalts-Typen für Policy-Tags . . . . .	25
3.2	Vergleichs-Operatoren für Policy-Tags . . . . .	26
5.1	Host-Scanner – zu prüfende Ports . . . . .	46
6.1	AV-Parameter des Security Centers . . . . .	52
6.2	FW-Parameter des Security Centers . . . . .	53
8.1	Eclipse-Projekte für Windows-Client . . . . .	68





# Listings

1.1	Beispiel-Quellcode . . . . .	4
3.1	Beispiel für <code>tnc_log.properties</code> . . . . .	24
3.2	Policy Normal Form . . . . .	24
3.3	Beispiel für eine Policy mit mehreren Alternativen . . . . .	25
3.4	Beispiel für eine Policy mit Vergleichs-Operatoren . . . . .	26
7.1	XML-Nachricht mit AV-Parametern des ClamWin-Virenschanners . . . . .	63
A.1	XML-Schema für die Konfiguration des IMVRegistry (Teil 1) . . . . .	77
A.2	XML-Schema für die Konfiguration des IMVRegistry (Teil 2) . . . . .	78
A.3	XML-Schema für die Konfiguration des IMVHostScanner (Teil 1) . . . . .	78
A.4	XML-Schema für die Konfiguration des IMVHostScanner (Teil 2) . . . . .	79
A.5	XML-Schema für Policies . . . . .	80
B.1	Basiskonfiguration des IMVRegistry (Teil 1) . . . . .	81
B.2	Basiskonfiguration des IMVRegistry (Teil 2) . . . . .	82
B.3	Basiskonfiguration des IMVHostScanner . . . . .	83
C.1	Policy für IMVRegistry . . . . .	85
C.2	Policy für IMVHostScanner . . . . .	86
C.3	Policy für IMVClamWin . . . . .	86
C.4	Policy für IMVSecurityCenter (FW-Parameter) . . . . .	87
C.5	Policy für IMVSecurityCenter (AV-Parameter) . . . . .	87
D.1	Beispiel-Quellcode mit Log-Ausgaben . . . . .	89



# Abkürzungsverzeichnis

AR .....	Access Requestor
BOOTP .....	Bootstrap Protocol
CDT .....	C/C++ Development Tools
DHCP .....	Dynamic Host Configuration Protocol
DLL .....	Dynamic Link Library
DNS .....	Domain-Name-System
EAP .....	Extensible Authentication Protocol
FTP .....	File Transfer Protocol
GPL .....	GNU General Public License
IDS .....	Intrusion Detection System
IMC .....	Integrity Measurement Collector
IMV .....	Integrity Measurement Verifier
PDP .....	Policy Decision Point
PEP .....	Policy Enforcement Point
RADIUS .....	Remote Authentication Dial In User Service
RFC .....	Request For Comment
RPC .....	Remote Procedure Call
SMTP .....	Simple Mail Transfer Protocol
SO .....	Shared Object
SQL .....	Structured Query Language
TCG .....	Trusted Computing Group
TNC .....	Trusted Network Connect
TNC-SG .....	Trusted Network Connect Sub Group
TNCC .....	TNC Client
TNCS .....	TNC Server
TPM .....	Trusted Platform Module
VPN .....	Virtual Private Network

WMI .....	Windows Management Instrumentation
XML .....	Extensible Markup Language