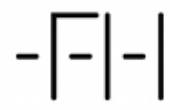
Fachhochschule Hannover
University of Applied Sciences and Arts

Fakultät IV - Wirtschaft und Informatik
Abteilung Informatik

Anbindung von Open Vulnerability Assessment System (OpenVAS) an eine Metadata Access Point (MAP)-Infrastruktur

Bachelorarbeit

Ralf Steuerwald
Juli 2011



Beteiligte

Erstprüfer
Prof. Dr. rer. nat. Josef von Helden
Fachhochschule Hannover
Ricklinger Stadtweg 120
30459 Hannover
E-Mail: josef.vonhelden@fh-hannover.de

Zweitprüfer
Bastian Hellmann M.Sc.
Fachhochschule Hannover
Ricklinger Stadtweg 120
30459 Hannover
E-Mail: bastian.hellmann@fh-hannover.de

Autor
Ralf Steuerwald
Leonhardring 29
31319 Sehnde
E-Mail: ralf.steuerwald@gmx.de

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die eingereichte Bachelorarbeit selbständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Ort, Datum Unterschrift

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

1 Einleitung

1.1 Motivation

Der störungsfreie Betrieb einer komplexen IT-Infrastruktur ist für die meisten Organisationen unerlässlich. Eine große Gefahr für den Betrieb dieser IT-Umgebungen stellen Angriffe von außen, aber auch von innen heraus dar. Diese Angriffe können verschiedene Ausprägungen besitzen, welche abhängig von den Zielen und der Motivation der Angreifer sind. Das Spektrum von Angriffen reicht dabei von Einbrüchen in IT-Systeme, um Daten zu stehlen, bis hin zur vollständigen Unterbrechung der normalen Funktionalität.

Wie anfällig Systeme internationaler Unternehmen und öffentlicher Einrichtungen sind und vor allem welche enormen Kosten durch diese verwundbare Systeme entstehen können, zeigt im Jahr 2011 eine Vielzahl von Angriffen [Rei11] gegen Unternehmen und Organisationen wie Sony [Jur11], GEMA [ore11] und viele weitere. Neben den direkten Kosten für den Ausfall wichtiger Systeme und damit verbundenen Umsatzverlusten, entsteht auch ein schwerwiegender Imageverlust, z.B. durch den Verlust von personenbezogenen Daten.

Um unternehmenskritische Infrastruktur, vor solchen Angriffen zu schützen, werden verschiedene Anstrengungen unternommen. Dazu gehören unter anderem eher passive Sicherheitsmaßnahmen wie zum Beispiel das Aufstellen von Firewalls an den Grenzen zu anderen Netzen oder zwischen Bereichen mit unterschiedlichen Sicherheitsansprüchen. Durch die Ausrichtung am Netzwerkperimeter können diese Komponenten eine Vielzahl von äußeren Angriffen verhindern.

Zusätzlich gibt es auch die Möglichkeit Komponenten in das Netzwerk zu integrieren, die durch reaktive oder sogar proaktive Funktionen die Sicherheit steigern können.

In den Bereich der reaktiven Komponenten fallen zum Beispiel Netzwerküberwachsysteme wie Snort oder Nagios. Diese Systeme sind in der Lage den Zustand des Netzwerkes zu überwachen und im Falle eines Angriffs oder einer ungewöhnlichen Situation Maßnahmen einzuleiten. Zu diesen Maßnahmen gehören z.B. das Senden einer Mitteilung an einen Administrator.

Ein weiterer, enorm wichtiger, Bereich sind die proaktiven Komponenten. In diese Kategorie fallen alle Werkzeuge, die dabei helfen Sicherheitslücken zu finden, bevor eine konkrete Gefährdungssituation eintritt.

Ein Werkzeug aus dieser Familie ist das Open Vulnerability Assessment System (OpenVAS), welches einen der Hauptbestandteile dieser Arbeit darstellt. Die wichtigsten Grundlagen zu OpenVAS werden in Kapitel 4 vermittelt.

Ein Nachteil, den die meisten oben genannten Systeme mit sich bringen, ist ihre isolierte Arbeitsweise. Wie oben schon erwähnt sind viele dieser Anwendungen in der Lage, Benachrichtigungen zu versenden, sobald sie eine Sicherheitsverletzung feststellen. Dies bedeutet, dass dann der Administrator manuell Maßnahmen ergreifen muss. Unter Umständen werden diese aber erst nach einer zeitlichen Verzögerung umgesetzt.

In der Vergangenheit wurden verschiedene Ansätze entwickelt, um die automatisierte Kommunikation zwischen Netzwerkgeräten zu verbessern. Dazu gehören z.B. Dienste wie syslog mit denen zentralisiert Logmeldungen verschiedenster Anwendungen gesammelt werden, um dann auf Anomalien untersucht zu werden. [Plö05]

Diese Ansätze haben aber alle den Nachteil, dass sie oft nur für eine Teilmenge an Diensten zur Verfügung stehen oder auf proprietäre Schnittstellen aufbauen. Außerdem entsteht durch die Verwendung dieser verschiedenen Kommunikationsprotokolle (z.B. syslog) eine Kommunikationsstruktur, die sehr stark gekoppelt ist und somit auch komplex und schwer zu warten.

Das von der Trusted Computing Group entworfene IF-MAP-Protokoll versucht diese Kommunikation zu vereinfachen indem alle, für die Sicherheitskoordination, benötigten Daten über eine zentrale Instanz geleitet werden. Ein weiterer großer Vorteil ist die herstellerunabhängige und offene Definition des Protokolls. IF-MAP bildet den zweiten Hauptbestandteil dieser Arbeit und wird im Kapitel 3 näher erläutert.

1.2 Ziel und Aufgabenstellung

Im Rahmen dieser Bachelorarbeit soll eine exemplarische Implementation einer Anbindung von OpenVAS an eine MAP-Infrastruktur entstehen. Dabei soll zum einen untersucht werden welche Ergebnisse eines Sicherheitsscans durch OpenVAS in einem MAP-Server veröffentlicht werden können. Zum anderen ist auch die Möglichkeit zu prüfen, wie OpenVAS dynamisch reagieren kann, wenn als Grundlage für diese Reaktion Daten dienen, die aus einem MAP-Server gewonnen werden können.

1.3 Aufbau dieser Arbeit

Im Folgenden werden zunächst in Kapitel 2 alle wichtigen Grundlagen, die für das Verständnis der folgenden Kapitel notwendig sind vermittelt. Anschließend wird in Kapitel 3 das IF-MAP-Protokoll analysiert. Danach wird im Kapitel 4 das Open Vulnerability Assessment System vorgestellt. Im Anschluss werden in Kapitel 5 die Anforderungen an eine Anbindung von OpenVAS an eine MAP-Infrastruktur aufgestellt. In Kapitel 6

wird ein Konzept für die Anbindung vorgestellt. Zu dem Konzept wird in Kapitel 7 dann eine exemplarische Realisierung beschrieben. Anschließend wird in Kapitel 8 betrachtet, ob das Konzept und die Implementierung alle gestellten Anforderungen erfüllen. Zum Abschluss wird in Kapitel 9 ein Ausblick in die zukünftige Entwicklung gegeben.

1.4 Typografische Konventionen

- Alle Auszüge aus Konfigurations-, Quelltext- und XML-Dateien werden in `grau hinterlegten` Listings abgebildet.
- Klassennamen, Methodennamen, XML-Elemente sowie Auszüge aus Konfigurationsdateien werden im Text `hervorgehoben`.
- Zu betonende Textstellen werden *kursiv* dargestellt.

2 Grundlagen

In diesem Kapitel werden kurz die Technologien und Begriffe beschrieben, die sowohl als Grundlage für die entstandene Implementierung als auch für die verwendeten Komponenten wichtig sind. Dieses Kapitel gibt nur einen Überblick und eine kurze Einführung in die entsprechenden Technologien und Begriffe. Die Erläuterungen in diesem Kapitel sollen nur das Verständnis der folgenden Kapitel sicher stellen. Für alle Abschnitte dieses Kapitels gibt es zahlreiche Werke, welche die teils sehr komplexen Themengebiete umfassend behandeln. Auf eine Auswahl dieser Quellen für detailliertere Informationen wird an entsprechender Stelle verwiesen, außerdem wird am Ende jedes Abschnittes auf den Bezug zu dieser Arbeit hingewiesen.

Da hier nicht alle Grundlagen vermittelt werden können, sind für das Verständnis der weiteren Kapitel einige Kenntnisse nötig, dazu gehören:

- Grundlagen in der Programmiersprache Java,
- das Verständnis für die Funktion und die Arbeitsweise von TCP/IP-Netzwerken,
- Grundverständnis für die Basisbegriffe der asymmetrischen Kryptographie,
- Grundkenntnisse über Anwendungsprotokolle (z.B. HTTP [FGM+97]) und deren Einsatzbereiche und
- Grundkenntnisse in der Socketprogrammierung [Kre02].

2.1 XML

Die Extensible Markup Language (XML) ist eine Auszeichnungssprache die vom World Wide Web Consortium (W3C) [wcc] Spezifiziert wurde [BPM+08] [MCP+06]. XML wird angewendet um Daten in einem menschenlesbaren und gleichzeitig maschinenlesbaren Format auszuzeichnen.

Ein XML-Dokument ist menschenlesbar, da innerhalb des Dokuments die Daten und Auszeichnungselemente als Text dargestellt werden. Eine einfache Verarbeitung durch Maschinen ist möglich, weil die Daten in einer festgelegten und sehr stark strukturierten Form angeordnet sind. Die Gesamtstruktur eines XML-Dokuments entspricht einem Baum. Ein Beispiel eines XML-Dokumentes ist in Listing 2.1 dargestellt.

Die wichtigsten Begriffe, die im XML-Umfeld benutzt werden, sind:

```
<?xml version="1.0" encoding="UTF-8" ?>
<buch xmlns="http://inform.fh-hannover.de/~vasmap"
  author="Holger Reibold">
    <titel>OpenVAS kompakt</titel>
    <kapitel>OpenVAS - der Einstieg</kapitel>
    <kapitel>OpenVAS in Betrieb nehmen</kapitel>
    <kapitel>Der OpenVAS-Client</kapitel>
    <kapitel>Die Scan-Einstellungen im Detail</kapitel>
    <kapitel>Berichte verstehen und interpretieren</kapitel>
    <kapitel>Die Zukunft des Scannens: GSA</kapitel>
    <kapitel>OpenVAS fuer Fortgeschrittene</kapitel>
    <kapitel>Tipps und Tricks fuer den Praxiseinsatz</kapitel>
    <kapitel>Eigene Tests schreiben</kapitel>
    <anhang>
          <konfigurationsdatei file="pfad/zur/datei">
                  ... Inhalt der Konfigurationsdatei ...
          </konfigurationsdatei>
    </anhang>
</buch>
```

Listing 2.1: Beispiel XML-Dokument

Element Elemente stellen einen Abschnitt in einem XML-Dokument dar. Dabei können Elemente wiederum andere Elemente enthalten oder Daten in Form von normalem Text. In Listing 2.1 stellen unter anderem `kapitel` und `anhang` Elemente dar.

Tag Der Name eines Elements eingeschlossen in `< >` wird als Tag bezeichnet. Jedes Element mit Inhalt besitzt ein Start- und einen End-Tag. Zum Beispiel ist `<titel>` das Start-Tag und `</titel>` das End-Tag des Elements `titel`. Leere Elemente werden durch ein Empty-Element-Tag `<elementName/>` dargestellt.

Attribute Attribute sind zusätzliche Informationen über ein Element. Sie werden innerhalb des Element-Tags angegeben. Im Beispiel stellt `file` ein Attribut des Elements `konfigurationsdatei` dar.

Namensräume Da verschiedene XML-Dokumente miteinander kombiniert werden können, kann es schnell zu Konflikten bei der Benennung von Elementen kommen. Um zwei Elemente mit gleichem Namen, aber unterschiedlicher Semantik unterscheiden zu können werden Namensräume eingesetzt. Im Beispiel wird für das gesamte Dokument der Namensraum `http://inform.fh-hannover.de/~vasmap` als Default-Namensraum definiert. Um verschiedene Namensräume zu kombinieren wird jedem Namensraum ein Prefix zugeordnet, z.B. durch `xmlns:vasmap="http://inform.fh-hannover.de/~vasmap"`. Jedes Element, welches sich in diesem Namensraum befindet muss nun ebenfalls dieses Prefix in Start- und End-Tag enthalten.

Damit ein XML-Dokument möglichst einfach verarbeitet werden kann, müssen folgende Bedingungen erfüllt sein [Bru11]:

- Die XML-Version des Dokuments muss angegeben sein (Im Beispiel in Zeile 1).

- Es existiert genau ein Wurzelelement (Im Beispiel `buch`).
- Jedes Start-Tag hat genau ein End-Tag.
- Die Schachtelung der Start- und End-Tags ist korrekt. Dies bedeutet, dass zu jedem Start-Tag auf der selben Schachtelungsebene ein End-Tag folgt.
- Die Werte von Attributen sind in `"`-Paare oder `'`-Paare eingeschlossen.
- Attributnamen sind innerhalb eines Elements einmalig.

Sind diese Bedingungen alle erfüllt wird das XML-Dokument auch als *wohlgeformt* (engl. well-formed) bezeichnet.

Zusätzlich zur Wohlgeformtheit kann ein XML-Dokument *gültig* sein, was auch als *valide* bezeichnet wird. Diese Eigenschaft ist erfüllt, wenn die Struktur des Dokuments der selben Struktur entspricht, die für dieses Dokument definiert wurde. Die Definition kann dabei auf verschiedene Arten stattfinden. Zwei dieser Möglichkeiten werden in den folgenden Abschnitten vorgestellt.

Wie schon oben erwähnt soll hier nur ein minimaler Einstieg gegeben werden, weitere Informationen können aus den XML-Spezifikationen [BPM+08] und [MCP+06] sowie aus [rrz10], [McL07] und [McL02] entnommen werden.

Bezug: XML ist für diese Arbeit von Bedeutung, da sowohl IF-MAP (3), als auch das OpenVAS Management Protocol (4.3.3) XML zur Auszeichnung der Operationen und Daten verwenden.

2.1.1 XML Schema

Mit einem XML Schema ist es möglich den Inhalt und die Struktur eines XML-Dokuments einzuschränken. Im vorherigen Abschnitt wurde der grundsätzliche Aufbau eines XML-Dokuments beschrieben. Die Regeln zur Wohlgeformtheit enthalten jedoch keine Vorschriften, die z.B. den Wertebereich von Elementen einschränken können, wie man es aus Programmiersprachen kennt. Für die Definition dieser Einschränkungen kann unter anderem XML Schema verwendet werden.

XML Schema stellt selbst auch ein XML-Dokument dar, wie in Listing 2.2 gezeigt. Das dargestellte XML Schema beschreibt dabei welche Datentypen und Elemente ein XML-Dokument enthalten darf. XML-Dokumente, die nach diesem XML Schema valide (oder gültig) sind werden auch als Instanzen des XML Schemas bezeichnet.

Das Element `xsd:element` mit dem Attribut `name="buch"` beschreibt, welcher Inhalt in einer Instanz dieses XML Schemas enthalten sein darf. Es wird festgelegt, dass eine Folge von verschiedenen Elementen Inhalt eines Elements `buch` sein darf. Diese Folge besteht aus:

- Einem Element `titel` vom Typ `xsd:string`,
- mindestens einem bis beliebig vielen `kapitel`-Elementen und

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://inform.fh-hannover.de/~vasmap"
  xmlns="hhttp://inform.fh-hannover.de/~vasmap"
  elementFormDefault="qualified"
>

  <xsd:element name="buch">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="titel" type="xsd:string" />
        <xsd:element name="kapitel" type="xsd:string" minOccurs="1"
              maxOccurs="unbounded" />
        <xsd:element name="anhang" type="anhangType" minOccurs="0" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:complexType name="anhangType">
    <xsd:sequence>
      <xsd:any minOccurs="1" maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>

  <xsd:element name="konfigurationsdatei">
    <xsd:complexType>
      <xsd:simpleContent>
        <xsd:extension base="xsd:string">
          <xsd:attribute name="file" type="xsd:string" />
        </xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

Listing 2.2: Beispiel XML Schema

- einem optionalem Anhang vom Typ `anhangType`.

Im Element `xsd:complexType` wird beschrieben, dass der Anhang aus mindestens einem, bis beliebig vielen, nicht weiter definierten Elementen `xsd:any` bestehen darf. Als letztes wird eine Element `konfigurationsdatei` von Typ `xsd:string` mit einem Attribut `file` auch vom Typ `xsd:string` definiert.

Die letzte Definition deutet schon an, dass XML Schema eine sehr komplexe Sprache ist und hier nicht umfassend behandelt werden kann. Außerdem ist das Beispiel sehr stark vereinfacht und würde so nicht in der Praxis eingesetzt werden. Deshalb sei auch hier wieder auf die offizielle XML Schema Spezifikationen [SMTM+09] und [PGB+09] sowie auf [rrz10], [McL07] und [McL02] verwiesen.

Bezug: XML Schema wird zur Definition der erlaubten Operationen und Daten des IF-MAP-Protokolls (3) verwendet.

2.1.2 RELAX NG

Bei RELAX NG handelt es sich, genau wie bei XML Schema, um eine Sprache die den Inhalt und die Struktur eines XML-Dokuments definiert. Im Gegensatz zu XML Schema ist RELAX NG nicht ganz so umfangreich, dadurch aber auch kompakter und leichter zu erlernen. RELAX NG kann in zwei verschiedenen Formen dargestellt werden. Die normale Variante stellt, wie XML Schema auch, ein XML-Dokument dar. Die kompakte Syntax hingegen orientiert sich sehr stark an einer Backus-Naur-Form und ist selbst kein XML. Informationen zu RELAX NG können aus den Spezifikationen zur normalen Syntax [CM01] und der kompakten Syntax [Cla02] entnommen werden.

Bezug: Analog zu XML Schema und IF-MAP, wird RELAX NG zur Definition der erlaubten Operationen und Daten im OpenVAS Management Protocol verwendet.

2.1.3 SAX

Die Simple API for XML (SAX) stellt ein Interface für Programmiersprachen bereit um XML-Dokumente verarbeiten zu können. Dabei wurde SAX zunächst für Java entwickelt, ist aber inzwischen auch für viele andere Programmiersprachen verfügbar.

Bei der Verarbeitung mit SAX wird ein XML-Dokument sequenziell eingelesen und beim Erreichen bestimmter Positionen innerhalb des XML-Dokuments werden sogenannte Callback-Methoden aufgerufen. Das Einlesen und Verarbeiten von XML-Dokumenten wird auch als parsen bezeichnet.

Die wichtigsten dieser Callback-Methoden sind:

startDocument: Diese Methode wird aufgerufen sobald begonnen wird das Dokument zu parsen.

startElement: Diese Methode wird für jedes Element, welches geparst wird, aufgerufen. Innerhalb dieser Methode ist der Zugriff auf den Namen des Elements, die Attribute des Elements sowie einiger weitere Eigenschaften möglich.

endElement: Diese Methode wird aufgerufen sobald das Ende eines Elements erreicht wurde. Auch in dieser Methode ist unter anderem der Zugriff auf den Elementnamen möglich.

characters: Befinden sich innerhalb eines Elements Textdaten wird diese Methode aufgerufen und die Textdaten übergeben.

endDocument: Diese Methode wird am Ende des XML-Dokuments aufgerufen, z.B. um Ressourcen, die während des parsens belegt wurden, wieder frei zu geben.

Um SAX für die Verarbeitung von XML-Dokumenten in Java einsetzen zu können, sind folgende Schritte notwendig:

1. Implementieren der oben genannten Callback-Methoden in einer Klasse die `org.xml.sax.helpers.DefaultHandler` erweitert.
2. Erzeugen eines Objektes der Klasse `javax.xml.parsers.SAXParser`.
3. Aufrufen der Methode `SAXParser.parse` mit dem XML-Dokument und einer Instanz der Klasse die von `org.xml.sax.helpers.DefaultHandler` abgeleitet wurde.

Eine weitere Technik um XML-Dokumente zu verarbeiten ist Document Object Model (DOM). Weitere Informationen und Beispiele zu DOM und SAX können in [rrz10], [McL07] und [McL02] gefunden werden.

Bezug: Im weiteren Verlauf dieser Arbeit wird SAX zur Verarbeitung von OMP-Nachrichten eingesetzt (7.2.3).

2.1.4 JAXB

Java Architecture for XML Binding (JAXB) verfolgt einen anderen Ansatz zur XML-Verarbeitung als SAX. Während bei SAX die gesamte Verarbeitung sehr stark am XML-Dokument ausgerichtet ist, versucht JAXB hier noch weiter zu abstrahieren. Dies geschieht dadurch, dass mit Hilfe von JAXB aus einem XML Schema Java Klassen generiert werden können, welche die Struktur des XML Schemas abbilden. Die Generierung erfolgt dabei mit dem Tool `xjc`.

Die generierten Java Klassen entsprechen also dem XML Schema. Daraus folgt, dass einzelne Objekte (Instanzen) der Java Klassen genau den XML-Instanzen entsprechen, die das XML Schema vorschreibt. Da hier nun eine direkte Abbildung vorhanden ist (welche auch als Datenbindung bezeichnet wird) kann der gesamte Vorgang des Umwandeln von Java-Objekten in XML-Dokumente und umgekehrt vollständig automatisiert geschehen. Der Entwickler muss also nicht mehr wie bei SAX die Methoden, die für das Verarbeiten einzelner XML-Tags zuständig sind, selbst implementieren. Das Umwandeln von Java-Objekten in XML-Dokumente wird auch als Marshalling bezeichnet, der umgekehrte Weg, also das Umwandeln von XML-Dokumenten in die entsprechenden Java-Objekte wird auch als Demarshalling bezeichnet.

Bezug: Eine Datenbindung wie beschrieben wird in dieser Arbeit für das Erzeugen von Protokollnachrichten verwendet (7.2.3).

2.2 SOAP

Mit SOAP wird ein Protokoll bezeichnet, welches zum einfachen Austausch von strukturierten Daten entwickelt wurde. Dabei ist SOAP noch über den Anwendungsprotokollen im OSI-Referenzmodell angeordnet. SOAP selbst kann für die Übertragung der

Daten diese anderen Anwendungsprotokolle, z.B. HTTP oder SMTP, benutzen. Außerdem ist es mit SOAP sehr einfach möglich entfernte Methodenaufrufe umzusetzen. Dabei enthält die SOAP-Nachricht den Namen der Methode oder des Dienstes, der auf einem anderen Rechner aufgerufen werden soll. Zusätzlich werden mit dieser Anfrage alle Parameter übertragen, die für den Aufruf nötig sind. Die SOAP-Nachricht, die als Antwort übertragen wird, enthält den Rückgabewert der Methode die aufgerufen wurde.

Zur Auszeichnung der Daten wird XML verwendet, was die einfache Verarbeitung von SOAP-Nachrichten, mit den in den vorherigen Abschnitten vorgestellten Technologien, möglich macht.

Der typische Aufbau einer SOAP-Nachricht besteht aus einem Umschlag, dem SOAP-Envelope, einem Kopf dem SOAP-Header und dem eigentlichen Nachrichteninhalt dem SOAP-Body. Der SOAP-Header ist optional und kann verschiedene Informationen enthalten, welche für jede Anwendung unterschiedlich ausfallen können. Beispiele für Daten, die im SOAP-Header enthalten sein können, sind Transaktionsnummern bei SOAP-Nachrichten, die sich über mehrere Aufrufe erstrecken.

Ein Begriff der im Zusammenhang mit SOAP eine wichtige Rolle spielt ist Web Services Description Language (WSDL). WSDL ist selbst auch ein XML-Dokument, welches die Schnittstelle beschreibt die durch SOAP-Nachrichten aufgerufen werden kann. Weitere Informationen zum Thema SOAP und WSDL können aus [ML07] entnommen werden.

Bezug: Für die Übertragung von IF-MAP-Nachrichten wird SOAP verwendet (3.2.5).

2.3 SSL/TLS

Das Secure Sockets Layer (SSL) Protokoll und das Nachfolgeprotokoll Transport Layer Security (TLS) sind Verschlüsselungsprotokolle, die es ermöglichen Anwendungsprotokolle verschlüsselt zu übertragen. Um dies zu ermöglichen, wird SSL/TLS zwischen der eigentlichen Transportschicht und den Anwendungsprotokollen (z.B. HTTP oder SMTP) eingesetzt.

Die Schutzziele die dabei von SSL/TLS umgesetzt werden sind Authentizität, Integrität und Vertraulichkeit der übertragenen Daten. Um diese Ziele durchzusetzen baut SSL/TLS auf dem Public-Key-Verfahren der asymmetrischen Kryptographie auf.

Die Authentizität wird dadurch sicher gestellt, dass beim Verbindungsaufbau (Handshake) zwischen den beiden Teilnehmern Zertifikate (üblicherweise X.509 Zertifikate) ausgetauscht werden, die ein sicheres Identifizieren des Kommunikationspartners ermöglichen. Dabei ist es immer vorgesehen, dass der Server sich beim anfragenden Client mit seinem Zertifikat authentisiert, umgekehrt ist die Authentisierung des Clients beim Server mit Hilfe eines Client Zertifikats optional.

Die Integrität wird durch den Einsatz von kryptografischen Prüfsummen ermöglicht.

Die eigentliche Verschlüsselung der übertragenen Daten wird durch einen symmetrischen Verschlüsselungsalgorithmus umgesetzt. Dies geschieht hauptsächlich aus Performancegründen, da symmetrische Verfahren um ein Vielfaches schneller sind als asymmetrische Verfahren. Damit beiden Kommunikationsteilnehmer über den selben Schlüssel zum Ver- und Entschlüsseln (der symmetrisch verschlüsselten Daten) verfügen, wird zu Beginn der Kommunikation im SSL-Handshake ein gemeinsamer Schlüssel ausgehandelt.

Detailiertere Informationen zu SSL/TLS können aus [DR08] sowie [Plö05] und [CKC05] entnommen werden.

Bezug: Die gesamten Netzwerkkommunikation im weiteren Verlauf dieser Arbeit wird durch das SSL/TLS Protokoll verschlüsselt.

2.4 IRON

Das Projekt Intelligent Reaction on Network Events (IRON), ist ein Forschungsprojekt, welches an der Fachhochschule Hannover durchgeführt wurde, um die Grundlagen für eine MAP-Infrastruktur zu schaffen. Im Rahmen dieses Projektes ist der MAP-Server irond entstanden, welcher eine Open Source Implementierung des IF-MAP 2.0 Standard darstellt. Weitere Anwendungen, die für den Einsatz in einer MAP-Infrastruktur an der Fachhochschule Hannover Entwickelt wurden sind irongui und irondhcp. Informationen sind auf der Trust@FHH-Webseite[1] zu finden. Eine ausführliche Einführung in das IF-MAP-Protokoll befindet sich in Kapitel 3.

Bezug: Zum Testen der Anbindung von OpenVAS wird irond verwendet.

2.5 Common Vulnerabilities and Exposures

Als Common Vulnerabilities and Exposures (CVE) wird ein Standard bezeichnet der eine einheitliche Bezeichnung von Sicherheitslücken und Verwundbarkeiten in verschiedensten IT-Systemen bereit stellt. Durch die einheitliche Benennung wird die Zusammenarbeit zwischen verschiedenen Sicherheitsorganisationen und vor allem der automatische Austausch von Informationen über Sicherheitslücken erleichtert.

CVEs bestehen im wesentlichen aus der eindeutigen Kennung und einer Beschreibung der Sicherheitslücke. Nachfolgend ist ein Beispiel einer CVE abgebildet.

```
CVE-ID: CVE-2008-0166

Description:
OpenSSL 0.9.8c-1 up to versions before 0.9.8g-9 on
Debian-based operating systems uses a random number
```

[1] https://trust.inform.fh-hannover.de/

```
generator that generates predictable numbers, which
makes it easier for remote attackers to conduct
brute force guessing attacks against cryptographic keys.
```

Bezug: OpenVAS stellt für alle gefundenen Schwachstellen eine CVE Bezeichnung bereit.

3 Analyse von IF-MAP

3.1 Trusted Computing Group

Die Trusted Computing Group (TCG) wurde im Jahr 2003 gegründet und hat das Ziel eine möglichst umfassende Definition von Standards bereit zu stellen, welche alle Teilbereiche des Trusted Computing [SWB08] abdecken. Zu den Standards der Trusted Computing Group gehören unter anderem hardwarenahe Komponenten, wie das Trusted Plattform Modul (TPM), welches eine Vertrauensbasis auf Hardwareebene bietet. Es wird aber auch durch eine Sammlung von verschiedenen Spezifikationen eine umfangreiche Architektur vorgeschrieben, die den sicheren und geregelten Zugriff auf Netzwerke beschreibt, sowie die koordinierte Zusammenarbeit verschiedener Netzwerkkomponenten in Echtzeit ermöglicht. Diese Architektur wird als Trusted Network Connect (TNC)[1] bezeichnet. Der sichere und vor allem authentisierte Zugriff auf ein Netzwerk von außen wird durch eine Erweiterung des IEEE 802.1X Standards [IEE04] ermöglicht. Diese Erweiterung sieht vor, dass in die üblichen Authentisierungsmechanismen zusätzliche Messwerte über den anfragenden Client eingehen, z.B. kann mit Hilfe des TPM festgestellt werden ob Veränderungen am Betriebssystemkern gemacht wurden, welche die Vertrauenswürdigkeit des Clients einschränken und deshalb ein Zugriff nicht gestattet wird.

Um innerhalb dieser Architektur den Austausch von Informationen (auch Metadaten genannt) zu ermöglichen, wurde das Interface for Metadata Access Points (IF-MAP) entworfen. Durch IF-MAP ist es möglich alle sicherheitsrelevanten Informationen in einem Netzwerk zentralisiert zu sammeln, zu strukturieren und für andere Netzwerkgeräte zugreifbar zu machen.

Die Daten, die die Komponenten in der TNC Architektur austauschen, werden mit dem IF-MAP-Protokoll übertragen und in einem MAP-Server gespeichert. Die Gesamtstruktur aus MAP-Server und einer Vielzahl an verschiedenen Clients, die unterschiedliche Aufgaben erfüllen wird auch als Metadata Access Point (MAP)-Infrastruktur bezeichnet. Da der Fokus dieser Arbeit nicht auf der gesamten TNC-Struktur liegt sei an dieser Stelle für weitere Informationen auf [SWB08] und [tcg] verwiesen. Die IF-MAP- Protokollversion auf die sich alle nachfolgenden Abschnitte beziehen ist Version 2.0 [Tru10a], auf Ausnahmen davon, die sich auf frühere Versionen beziehen, wird an entsprechender Stelle hingewiesen.

[1] `http://www.trustedcomputinggroup.org/developers/trusted_network_connect`

Zwei Begriffe aus TNC müssen für die spätere Verwendung definiert werden:

Policy Decision Point (PDP): Eine Netzwerkkomponente die aktiv Sicherheitsentscheidungen trifft. Die Entscheidung wird dabei aufgrund von verschiedenen Integritäts- und Sicherheitsinformationen getroffen, die aus einem MAP-Server gewonnen werden können.

Policy Enforcement Point (PEP): Eine Netzwerkkomponente (z.B. ein Switch oder ein WLAN-Access-Point) die von einem PDP getroffenen Entscheidungen durchsetzt. Dazu gehören z.B. das Isolieren von Clients in spezielle Netzwerksegmente oder das Ausschließen von Clients aus dem Netzwerk.

3.2 Grundlagen des IF-MAP-Protokolls

Das IF-MAP-Protokoll stellt eine herstellerunabhängige Schnittstelle bereit, die es verschiedenen Netzwerkkomponenten erlaubt Daten auszutauschen. Dabei war als Hauptanwendungsfall zunächst der Austausch von sicherheitsrelevanten Daten in der TNC Architektur vorgesehen. Jedoch ist IF-MAP, aufgrund der offenen Spezifikation und den Fokus auf Erweiterbarkeit, nicht auf diesen Anwendungsfall beschränkt.

Das Kommunikationsmodell welches Verwendung findet, wird als „publish/subscribe“ bezeichnet. Dabei sind „Publisher“ Kommunikationsteilnehmer, die Daten über einen Kanal veröffentlichen. Der zweite Typ von Teilnehmern sind die „Subscriber“. Diese „Subscriber“ können sich für bestimmte Daten anmelden und erhalten dann passend zu ihren registrierten Interessen Informationen, die von den „Publishern“ veröffentlicht wurden. Der Austausch der Daten findet dabei über einen zentralen Kommunikationskanal statt. Die „Subscriber“ und „Publisher“ kommunizieren nicht direkt miteinander, sondern nur indirekt über einen zentralen Server.

In einer IF-MAP-Umgebung werden die „Publisher“ und „Subscriber“ durch beliebige Netzwerkteilnehmer dargestellt. Dabei kann jedes Geräte sowohl „Subscriber“ als auch „Publisher“ oder auch nur eines darstellen.

Im folgendem werden „Publisher“ und „Subscriber“ gleichbedeutend auch als Client oder IF-MAP-Client bezeichnet.

Der zentrale Server wird als Metadata Access Point Server bezeichnet (im weiteren Verlauf auch MAP-Server genannt). Es ergibt sich also eine Kommunikationsstruktur wie in Abbildung 3.1 dargestellt. Eine Vielzahl von verschiedenen Clients, mit den unterschiedlichsten Aufgaben, greift auf einen zentralen MAP-Server zu.

Diese zentralisierte Ausrichtung der Kommunikation vermeidet Kommunikationsstrukturen, wie sie durch den Einsatz von verschiedenen Protokollen (z.B. syslog) entstehen. Diese alten Strukturen wie in Abbildung 3.2 abgebildet sind komplexer, schwerer zu Warten und unter Umständen nicht standardisiert.

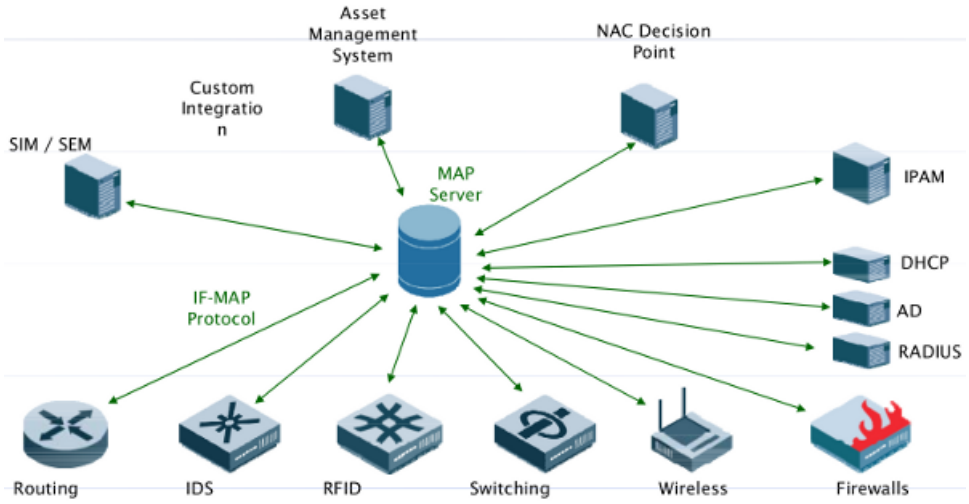


Abbildung 3.1: MAP-Infrastruktur Quelle: [tcg09]

Auf einem MAP-Server können von IF-MAP-Clients beliebige Daten veröffentlicht werden. Gleichzeitig können sich IF-MAP-Clients am MAP-Server für bestimmte Daten registrieren. Diese Clients erhalten dann automatisch Informationen vom MAP-Server falls von einem anderem Client passende Informationen veröffentlicht wurden. Innerhalb des MAP-Server werden die Daten zu einem ungerichtetem Graphen verknüpft. In Abbildung 3.3 ist eine mögliche Ausprägung eines solchen Graphen dargestellt. Dabei repräsentiert ein grau hinterlegtes Oval einen Identifier (3.2.1), Rechtecke stellen Metadaten (3.2.2) dar und die Verbindungslinien werden als Links (3.2.3) bezeichnet.

Als Protokollgrundlage dient dabei XML zur Auszeichnung der Daten und Operationen. Beschränkt wird das grundlegende Protokoll durch ein XML Schema, welches in [Tru10a] beschrieben ist. Die gesamte Kommunikation der IF-MAP-Clients mit dem MAP-Server findet über eine verschlüsselte Verbindung statt.

Zunächst werden nun einige Begriffe erläutert, die für das Verständnis dieses Kapitels notwendig sind und auch im weiteren Verlauf dieser Arbeit immer wieder eine wichtige Rolle spielen werden. Dabei ist zu beachten, dass aufgrund des Umfangs hier nicht alle Elemente, Attribute und Typen aufgelistet werden. Die exakten Definitionen können aus der Spezifikation [Tru10a] entnommen werden.

3.2.1 Identifier

Identifier sind die Datentypen innerhalb des IF-MAP-Protokolls die für alle Operationen als Startpunkt dienen. Zur Zeit sind 5 Identifiertypen spezifiziert mit denen es möglich ist verschiedene Entitäten innerhalb einer Netzinfrastruktur zu adressieren.

ip-address: Eine IP-Adresse, wahlweise aus dem IPv4 oder IPv6 Protokoll.

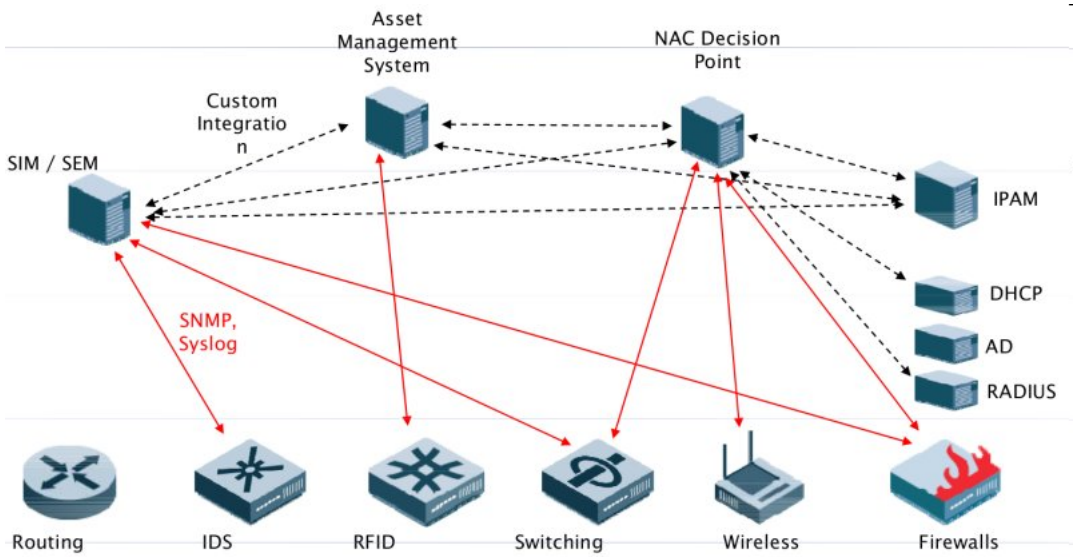


Abbildung 3.2: Komplexe Kommunikationsstrukturen Quelle: [tcg09]

mac-address: Die physikalische Adresse eines Endgerätes.

device: Ein Endgerät welches durch eine eindeutige Namen identifizierbar ist.

identitiy: Eine Identität die nicht nur auf natürliche Personen beschränkt ist, sondern unter anderem auch durch einen DNS-Namen repräsentiert werden kann.

access-request: Eine Anfrage für den Zugriff auf ein geschütztes Netzwerk.

3.2.2 Metadaten

Metadaten enthalten Informationen über die Identifier an die sie assoziiert sind. In [Tru10a] sind keine speziellen Metadaten spezifiziert. In einer zusätzlichen Spezifikation [Tru10b] ist das XML Schema und die Verwendung einer grundlegenden Menge von Metadatentypen speziell für die Anwendung im Bereich der Netzwerksicherheit im TNC Umfeld beschrieben.

Außerdem ist das IF-MAP XML Schema so aufgebaut, dass die Menge an Metadaten beliebig für spezielle Anwendungsbereiche erweitert werden kann.

Genauere Informationen über alle Metadatentypen können aus [Tru10b] entnommen werden. Hier wird nur eine kleine Auswahl vorgestellt, die im weiteren Verlauf der Arbeit verwendet wird.

event: Ein Netzwerkereignis welches unterschiedliche Ursachen haben kann und für die Sicherheit eines Netzwerkes von Relevanz ist.

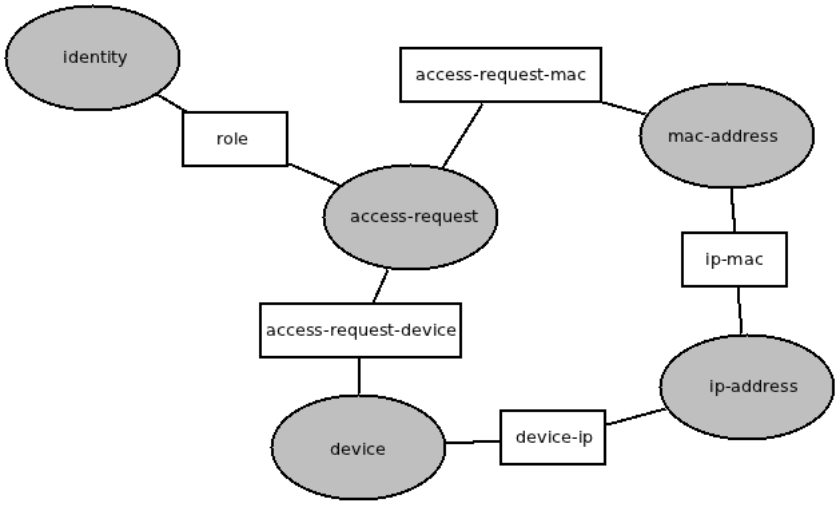


Abbildung 3.3: IF-MAP-Graph

request-for-investigation: Die Aufforderung eines Policy Decision Point nach einer genaueren Untersuchung der assoziierten MAC-Adresse oder IP-Adresse.

Ein wichtiges XML-Attribut welches für alle Metadaten vorgesehen ist und welches Auswirkungen auf die Verarbeitung der Metadaten hat, ist `ifmap-cardinality`. Für dieses Attribut sind zwei Werte vorgesehen. Der Wert `singleValue` bedeutet, dass bei einer `update`-Operation alle vorhandenen Metadaten, bezüglich des selben Identifiers, mit den neuen Metadaten aus dieser `update`-Operation ersetzt werden. Die Angabe von `multiValue` hingegen bedeutet, dass die neuen Metadaten zu den bereits bestehenden hinzugefügt werden.

3.2.3 Links

Wie schon erwähnt werden die Daten innerhalb des MAP-Servers durch einen Graphen repräsentiert. Die Knoten des Graphen stellen Identifier oder Metadaten dar. Die Kanten, die diese Knoten verbinden, werden als Links bezeichnet. Dabei sind folgende Kombinationen von Links, Metadaten und Identifiern möglich:

- Zwei Identifier, verbunden durch einen Link auf dem sich mindestens ein Metadatum befindet (Abbildung 3.4).
- Ein Identifier der nur mit einem Link verbunden ist auf dem sich mindestens ein Metadatum befindet (Abbildung 3.5).

Zwei Identifier, die nur über einen Link verbunden sind, welcher keine Metadaten besitzt sind im IF-MAP-Protokoll nicht vorgesehen. Genau wie ein Identifier, der nur mit einem Link verbunden ist, der auch keine Metadaten enthält.

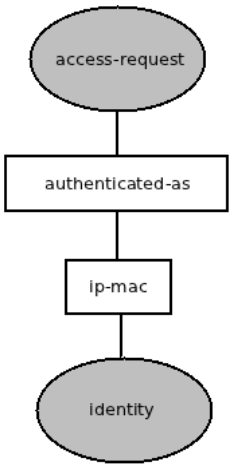


Abbildung 3.4: Zwei Identifier mit Metadaten

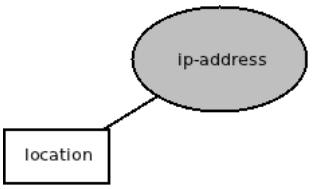


Abbildung 3.5: Ein Identifier mit Metadaten

3.2.4 Operationen

Im Folgenden sollen nun die Operationen vorgestellt werden, die eine Interaktion mit dem MAP-Server ermöglichen. Dabei gibt es zwei verschiedene Kanäle auf denen ein Client mit dem MAP-Server kommunizieren kann.

Synchronous Send-Receive Channel (SSRC): Auf dieser Verbindung werden alle synchronen Operationen ausgeführt. Darunter fallen alle Anfragen an den Server auf die dieser unmittelbar antworten kann, z.B. den Aufbau einer neuen Verbindung oder das Veröffentlichen von Metadaten auf dem MAP-Server.

Asynchronous Receive Channel (ARC): Diese Verbindung wird für Operationen verwendet auf die der MAP-Server unter Umständen nicht sofort antworten kann. Dazu gehört aktuell nur der `poll`-Request (siehe 3.2.4). Die Antwort wird asynchron zur Anfrage übertragen, dies bedeutet, dass der Client nach der Anfrage zunächst andere Aufgaben erfüllen kann und die Antwort zu einem undefiniertem späteren Zeitpunkt übertragen wird.

Session Management

Jede IF-MAP-Operation muss innerhalb einer gültigen Session stattfinden. Ausgenommen davon ist der `newSession`-Request. Der MAP-Server vergibt beim Verbindungsaufbau eines Clients eine `session-id`. Diese `session-id` wird dann bei jeder Operation vom Client mit übertragen.

Folgende Operationen sind für das Verwalten einer Session zuständig:

newSession: Baut eine neue Verbindung zum MAP-Server auf. Der Server sendet als Antwort die `session-id` sowie eine `ifmap-publisher-id` die den Client identifiziert.

endSession: Kann von einem Client gesendet werden um eine Session zu schließen. Dabei muss eine gültige `session-id` angegeben werden. Beendet der Client seine Session nicht und ist über eine bestimmte Zeit inaktiv, verfällt die Session automatisch. Der Minimalwert für diesen Session-Timeout liegt bei 180 Sekunden.

renewSession: Erneuert die Session eines Clients um den automatischen Session-Timeout zu umgehen. Auch hier muss eine gültige `session-id` angegeben werden.

Daten veröffentlichen

Der `publish`-Request wird verwendet um Metadaten auf dem MAP-Server zu verändern. Es wird zwischen 3 Unteroperationen unterschieden, die in beliebiger Anzahl und beliebiger Reihenfolge in einem `publish`-Request enthalten sein können.

update: Mit dieser Operation ist es möglich neue Metadaten im MAP-Server zu speichern. Dabei wird immer mindestens ein Identifier (maximal zwei Identifier) pro `update` angegeben, sowie eine Liste von Metadaten die mit diesen Identifiern verbunden sind.

notify: Genau wie beim `update` ist es auch mit dieser Operation möglich Metadaten an den MAP-Server zu übertragen. Der Unterschied liegt darin, dass die Daten, die mit dieser Operation veröffentlicht wurden, nicht im MAP-Server gespeichert werden, sondern direkt an alle registrierten Subscriber gesendet werden.

delete: Mit dieser Operation können Metadaten, die zuvor per `update` veröffentlicht wurden, wieder gelöscht werden.

Daten suchen

Mit der `search`-Operation können Daten im MAP-Server gesucht werden. Dafür wird ein Identifier als Startpunkt angegeben. Durch die Kombination von verschiedenen Filtern kann die Suche genauer eingegrenzt werden. Als Ergebnis werden alle verbundenen Identifier und Metadaten zurück geliefert, welche nicht durch die Filter aussortiert wurden.

Daten abonnieren

Um Daten zu abonnieren sind zwei Schritte nötig. Als erstes muss sich der IF-MAP-Client beim MAP-Server für bestimmte Daten registrieren. Danach öffnet er eine weitere Verbindung auf welcher er auf Neuigkeiten, für seine abonnierten Daten, wartet. Sobald der MAP-Server (von anderen Clients) neue Informationen erhält sendet er diese an alle angemeldeten Clients.

Ein IF-MAP-Client hat zwei Operationen zur Verfügung um seine Abonnement auf dem MAP-Server zu verwalten. Beide werden innerhalb eines `subscribe`-Request eingeschlossen.

update: Registriert den Client für neue Informationen bezüglich eines Identifiers. Analog zum `search`-Request (3.2.4) können auch hier Filter angegeben werden um die Ergebnismenge einzuschränken. Zusätzlich wird zur Identifikation ein Name für jedes `update` angegeben.

delete: Mit dieser Operation kann ein Abonnement wieder gelöscht werden.

Nachdem ein Client sich für Informationen angemeldet hat, kann er auf einer anderen asynchronen Verbindung passende Daten empfangen.

poll: Sendet über einen ARC die Anforderung nach neuen Informationen an den MAP-Server. Sind gerade keine neuen Informationen vorhanden blockiert diese Verbindung bis der MAP-Server neue Informationen erhält.

Weitere Operationen

Neben den bereits erwähnten Operationen gibt es noch weitere, welche hier nur kurz erwähnt werden.

purgePublisher: Mit dieser Operation können alle Metadaten, die von einem bestimmten Publisher stammen, gelöscht werden.

dump: Bei dieser Operation handelt es sich um eine nicht standardisierte Erweiterung die von Tobias Ruhe im Rahmen der Entwicklung der irongui [Ruh10] entstanden ist. Mit der `dump`-Operation ist es möglich alle Identifier, die sich momentan in einem MAP-Server befinden, abzufragen.

3.2.5 Transport

In der Spezifikation [Tru10a] ist festgelegt, dass alle IF-MAP-Operationen in eine SOAP-Nachricht eingebettet werden müssen. Dabei wird die SOAP Version 1.2 [ML07] eingesetzt. SOAP kann prinzipiell verschiedene Anwendungsprotokolle (z.B. HTTP oder SMTP) zum Transport verwenden, jedoch wird HTTP als Standard festgelegt. Außerdem wird festgelegt, dass die gesamte Kommunikation verschlüsselt durchgeführt werden muss. Das bedeutet, dass in der Regel IF-MAP-Kommunikation über eine, durch SSL/TLS verschlüsselte, HTTPS-Verbindung abläuft.

3.2.6 Authentifizierung

Für die Authentifizierung von IF-MAP-Clients stehen zwei Möglichkeiten zur Wahl. Zum einen können sich Clients mit der HTTP Basic Authentication [FHBH+99] beim Server authentifizieren, zum anderen wird die zertifikatsbasierte Authentifizierung unterstützt. Bei der Basic Authentication müssen auf dem MAP-Server ein Benutzername und Passwort für jeden Client hinterlegt werden. Der Client bildet dann bei einer Verbindungsanfrage einen String, welcher Benutzername und Passwort getrennt durch einen Doppelpunkt enthält. Dieser String wird dann Base64 kodiert und im HTTP-Header an den Server gesendet. Bei der zertifikatsbasierten Authentisierung wird während des SSL-Handshakes nicht nur vom MAP-Server ein Zertifikat an den Client gesendet, sondern der Client sendet auch sein Zertifikat an den Server.

3.2.7 Beispiel

Um das Zusammenspiel der verschiedenen Operationen zu verdeutlichen wird nun kurz eine beispielhafte Kommunikation zwischen zwei IF-MAP-Clients und einem MAP-Server beschrieben.

An der Kommunikation nehmen zwei IF-MAP-Clients und ein MAP-Server teil. Die Clients verfolgen dabei verschiedene Ziele, der Client „Publisher“ veröffentlicht Daten auf dem MAP-Server. Der Client „Subscriber“ registriert sich beim MAP-Server für bestimmte Informationen.

In Abbildung 3.6 ist der zeitliche Ablauf der einzelnen Nachrichten dargestellt.

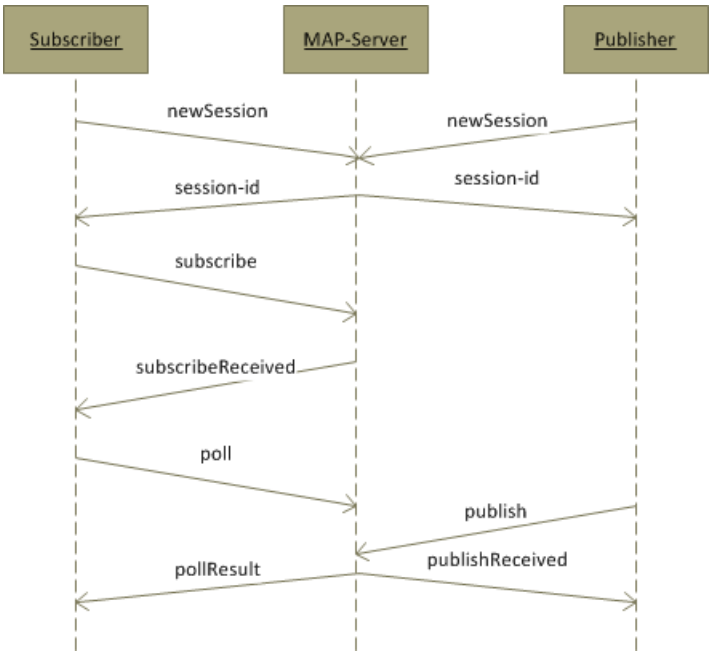


Abbildung 3.6: IF-MAP Beispiel-Kommunikation

Die erste Operation die von beiden Clients ausgeführt wird, ist das Eröffnen einer neuen Verbindung zum MAP-Server. Dafür sende beide Clients die Anfrage, welche in Listing 3.1 dargestellt ist, an den MAP-Server.

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:ifmap="http://www.trustedcomputinggroup.org/2010/IFMAP/2">
        <soap:Body>
                <ifmap:newSession/>
        </soap:Body>
</soap:Envelope>
```

Listing 3.1: IF-MAP Verbindungsanfrage

Falls die Clients über die nötige Berechtigung verfügen und vom MAP-Server wie in Abschnitt 3.2.6 beschrieben authentifiziert wurden, antwortet der Server jedem der Clients mit einer individuellen Sessionkennung (`session-id`) wie in den Listings 3.2 und 3.3 dargestellt.

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:ifmap="http://www.trustedcomputinggroup.org/2010/IFMAP/2">
  <soap:Body>
    <ifmap:response>
          <newSessionResult ifmap-publisher-id="publisher42" session-id="222...222"/>
    </ifmap:response>
  </soap:Body>
</soap:Envelope>
```

Listing 3.2: Neue Verbindung für Client „Publisher“

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:ifmap="http://www.trustedcomputinggroup.org/2010/IFMAP/2">
  <soap:Body>
    <ifmap:response>
          <newSessionResult ifmap-publisher-id="subscriber21" session-id="333...333"/>
        </ifmap:response>
  </soap:Body>
</soap:Envelope>
```

Listing 3.3: Neue Verbindung für Client „Subscriber“

Beide Clients können nun, da sie über eine gültige Session verfügen, alle IF-MAP-Operationen ausführen. Der Client „Subscriber“ registriert sich beim MAP-Server für alle Metadaten, die für die IP-Adresse `10.0.0.1` veröffentlicht werden (3.4). Zusätzlich wird noch eine Filteroption benutzt, welche die Ergebnismenge auf Identifier eingrenzt die maximal 4 Links von der IP-Adresse entfernt sind.

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:ifmap="http://www.trustedcomputinggroup.org/2010/IFMAP/2"
  xmlns:meta="http://www.trustedcomputinggroup.org/2010/IFMAP-METADATA/2">
        <soap:Body>
                <ifmap:subscribe session-id="333...333">
                        <update name="4711" max-depth="4">
                                <ip-address value="10.0.0.1" type="IPv4"/>
                        </update>
                </ifmap:subscribe>
        </soap:Body>
</soap:Envelope>
```

Listing 3.4: Anmeldung für Informationen

Die Antwort in Listing 3.5 vom MAP-Server signalisiert dem Client, dass seine Registrierung erfolgreich war. Der Subscriber sendet danach direkt ein `poll`-Request an den MAP-Server wie in Listing 3.6 dargestellt.

Nun sendet der Client Publisher neue Informationen an den MAP-Server wie in Listing

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:ifmap="http://www.trustedcomputinggroup.org/2010/IFMAP/2">
        <soap:Body>
                <ifmap:response>
                        <subscribeReceived/>
                </ifmap:response>
        </soap:Body>
</soap:Envelope>
```

Listing 3.5: Antwort auf Subscribe

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:ifmap="http://www.trustedcomputinggroup.org/2010/IFMAP/2">
        <soap:Body>
                <ifmap:poll session-id="333...333"/>
        </soap:Body>
</soap:Envelope>
```

Listing 3.6: Anfordern von neuen Informationen

3.7 gezeigt. Dabei könnte es sich um Daten von einem DHCP Server handeln, welcher anderen Clients die Information zur Verfügung stellen möchte, dass die MAC-Adresse `01:23:45:67:89:ab` zu der IP-Adresse `10.0.0.1` gehört. Als Antwort erhält er die Bestätigung aus Listing 3.8. Ein möglicher Graph der nach dem Veröffentlichen dieser Metadaten im MAP-Server entstanden ist, wird in Abbildung 3.7 gezeigt. Dabei ist zu beachten, dass zwischen den neu veröffentlichten Metadaten (rechts) und den bereits vorher bestehenden (links) keine Verbindung existiert.

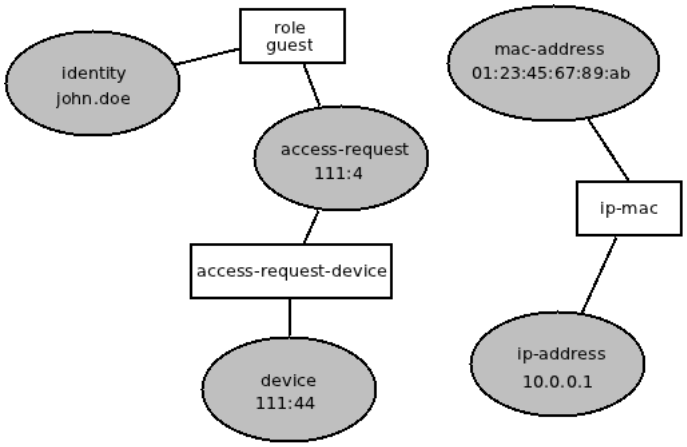


Abbildung 3.7: Beispiel-Graph nach Publish

Als Reaktion, auf das Veröffentlichen dieser neuen Daten, erhält der Client Subscriber

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:ifmap="http://www.trustedcomputinggroup.org/2010/IFMAP/2"
  xmlns:meta="http://www.trustedcomputinggroup.org/2010/IFMAP-METADATA/2">
        <soap:Body>
                <ifmap:publish session-id="222...222">
                        <update>
                                <ip-address value="10.0.0.1" type="IPv4"/>
                                <mac-address value="01:23:45:67:89:ab" />
                                <metadata>
                                        <meta:ip-mac ifmap-cardinality="multiValue" />
                                </metadata>
                        </update>
                </ifmap:publish>
        </soap:Body>
</soap:Envelope>
```

Listing 3.7: Veröffentlichen von neuen Informationen

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:ifmap="http://www.trustedcomputinggroup.org/2010/IFMAP/2">
        <soap:Body>
                <ifmap:response>
                        <publishReceived/>
                </ifmap:response>
        </soap:Body>
</soap:Envelope>
```

Listing 3.8: Bestätigung vom MAP-Server

nun die asynchrone Antwort auf den Poll-Request.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:ifmap="http://www.trustedcomputinggroup.org/2010/IFMAP/2">
  <soap:Body>
    <ifmap:response>
      <pollResult>
        <updateResult name="4711">
          <resultItem>
            <ip-address type="IPv4" value="10.0.0.1"/>
            <mac-address value="01:23:45:67:89:ab"/>
            <metadata>
              <meta:ip-mac
                xmlns:meta="http://www.trustedcomputinggroup.org/2010/IFMAP-METADATA/2"
                ifmap-cardinality="multiValue"
                ifmap-publisher-id="test7650336901"
                ifmap-timestamp="2011-07-11T14:53:50+02:00"/>
            </metadata>
          </resultItem>
        </updateResult>
      </pollResult>
    </ifmap:response>
  </soap:Body>
</soap:Envelope>
```

Listing 3.9: Antwort auf den Poll-Request

4 Analyse von OpenVAS

In diesem Kapitel werden die Grundlagen zum Open Vulnerability Assessment System (OpenVAS) beschrieben. Alle Angaben ohne weitere Hinweise beziehen sich auf die aktuelle Entwicklungslinie OpenVAS-4. Die Online Dokumentation auf der OpenVAS-Webseite [ope] beschreibt die veraltete Version 2.x, ausführliche Informationen zur Administration von OpenVAS 3.x und ein Ausblick auf Version 4 können aus [Rei10] entnommen werden.

Bei OpenVAS handelt es sich um ein Softwaresystem mit dem Netzwerkgeräte auf Sicherheitslücken und Schwachstellen untersucht werden können.

Das OpenVAS-Projekt ist aus dem Sicherheitsscanner Nessus hervorgegangen. Nachdem das Nessus-Projekt 2005 von der GNU General Public Licence zu einer proprietären Lizenz gewechselt hat, wurde basierend auf der letzten freien Nessus Version 2.2 OpenVAS ins Leben gerufen.

Da Nessus die Grundlage für OpenVAS bildet, sind sich viele der Konzepte ähnlich. Eine umfassende Einführung in die Arbeit mit der freien Nessus Version kann in [Der05] gefunden werden. Allerdings hat sich OpenVAS seit der Abspaltung von Nessus auch weiter entwickelt, so dass nicht mehr alle Bestandteile in beiden System gleich existieren. Beispiele für die Neuentwicklungen im OpenVAS-Projekt sind der Greenbone Security Assistant (4.2.5) oder das OpenVAS Management Protocol (4.3.3).

Zu den Konzepten, die gleich geblieben sind, gehört die Organisation im Client-Server-Modell. Dabei bildet der OpenVAS-Server die zentrale Instanz die sämtliche Scanaufträge verwaltet, ausführt und die Berichte aufarbeitet. Gesteuert wird dieser zentrale OpenVAS-Server von Clients, die über mehrere Wege auf den Server zugreifen können. Zu diesen Zugriffswegen gehören eine Weboberfläche (4.2.5), eine Desktopanwendung (4.2.6) und ein für OpenVAS neu entwickeltes Management Protokoll (4.3.3).

Für die Sicherheitsüberprüfung steht eine große Anzahl (im Juli 2011 ca. 21.000 Stück) von Angriffsszenarien (4.1.1) zur Verfügung. Jedes dieser Szenarien kann zur Überprüfung einer Schwachstelle verwendet werden. Vom Administrator können beliebig viele dieser Überprüfungen zusammengefasst werden. Diese Zusammenstellung kombiniert mit weiteren Angaben, wie z.B. ein oder mehrere Ziele (4.1.4) bilden einen Scanauftrag (4.1.5). Dieser Scanauftrag wird dann von OpenVAS automatisiert (ggf. auch wiederholt zu bestimmten Zeitpunkten) ausgeführt. Nach der Überprüfung werden von OpenVAS die Erkenntnisse zu einem Bericht zusammengefasst.

4.1 Begriffe

Im weiteren Verlauf werden alle wichtigen Begriffe für den Umgang mit OpenVAS genauer betrachtet.

4.1.1 Plug-ins

Die Plug-ins sind einer der wichtigsten Bestandteile von OpenVAS. Jedes Plug-in ist für die Überprüfung einer Schwachstelle zuständig. Plug-ins werden auch als Network Vulnerability Tests (NVT) bezeichnet.

Aufgrund der enorm großen Anzahl von Plug-ins sind diese in verschiedenen Kategorien organisiert. Für ein Scanauftrag können einzelne Plug-ins oder ganze Kategorien ausgewählt werden. Außerdem gibt es zwei übergreifende Kategorien von Plug-ins:

- Sichere Scans, die das Zielsystem nicht beeinflussen.
- Scans, die das Zielsystem mit allen Mitteln angreifen und unter Umständen einen Absturz verursachen können.

Um nur Plug-ins der ersten Kategorie zu verwenden muss in der Konfiguration die Option `safe_checks` aktiviert werden.

Details zu den einzelnen Plug-in Kategorien können aus [Rei10] oder den Beschreibungen die jedem Plug-in zugeordnet sind entnommen werden. Im folgenden sei nur eine kleine Auswahl aus den verfügbaren Plug-in Kategorien aufgezeigt:

- IT-Grundschutz,
- Buffer overflow,
- Databases,
- Red Hat Local Security Checks,
- ...

Jede dieser Kategorien (aktuell 47 Kategorien) enthält einige wenige, bis mehrere tausend, Plug-ins.

Da täglich neue Sicherheitslücken entdeckt werden gibt es sogenannte Network Vulnerability Tests Feeds. Über diese Feeds können aktuelle Plug-ins bezogen werden. Das OpenVAS-Projekt stellt dafür einen eigenen Feed zur Verfügung, welche mit dem Kommandozeilentool `openvas-nvt-sync` abgefragt werden kann.

4.1.2 Nessus Attack Scripting Language

Bei der Nessus Attack Scripting Language (NASL) handelt es sich um eine Skriptsprache, die speziell für die Formulierung von Plug-ins entworfen wurde. Von der Syntax her orientiert sich NASL sehr stark an C. Jedoch ist der Anwendungsbereich nahezu vollständig auf das Erstellen von Sicherheitstests eingeschränkt. Zur Steigerung der Sicherheit des OpenVAS-Servers sind aus einem NASL-Skript heraus keine lokalen Systemaufrufe oder das Versenden von Nachrichten an andere Rechner als dem Zielrechner erlaubt. [Der05]

4.1.3 Local Security Checks

Bei der Ausführung von Sicherheitsüberprüfungen gibt es grundsätzlich zwei verschiedene Strategien. Die normale Ausführung ohne weitere Konfiguration testet das Ziel von außen, indem nach offenen Ports und verwundbaren Diensten gesucht wird. Ein anderer Ansatz den OpenVAS bietet, sind die Local Securtiy Checks. Bei dieser Art der Überprüfung werden im OpenVAS-Server Anmeldeinformationen, in Form von SSH oder SMB Login-Informationen, hinterlegt. Plug-ins, die für Local Security Checks ausgelegt sind, verbinden sich dann mit dem Ziel wahlweise per SSH oder SMB und führen direkt auf dem Ziel ihre Überprüfungen durch. Mit dieser Art der Überprüfung können natürlich eine Vielzahl an Schwachstellen aufgedeckt werden, welche über einen Test von Außen nicht entdeckt werden können, weil sie keine aktiven Dienste darstellen. Ein Beispiel dafür ist das Aufspüren von Anwendungsprogrammen (z.B. PDF-Reader), die in bestimmten Versionen gefährliche Sicherheitslücken besitzen.

4.1.4 Targets

Als Target wird das Ziel eines Sicherheitsscans bezeichnet. Dabei kann die Angabe eines Targets auch mehrere Ziele umfassen. Es können ganze IP-Adressbereiche als Ziele definiert werden oder eine Liste von IP-Adressen festgelegt werden, die Angabe von Hostnamen ist auch erlaubt.

4.1.5 Tasks

Tasks sind die eigentlichen Scanaufträge die von OpenVAS ausgeführt werden. Eine Task besteht im Wesentlichen aus folgenden Bestandteilen:

- Einer Auswahl von Plug-ins und andere Parametern, z.B. `safe_checks`.
- Einem Trigger der z.B. nach jedem Scandurchlauf eine Benachrichtigung an den Administrator versendet.
- Eine Regel für die wiederholte Ausführung von Scans.

- Der Angabe eines Targets.

4.1.6 Reports

Die Reports, auch Berichte genannt, enthalten die Ergebnisse eines oder mehrerer Sicherheitsscans bezüglich einer Task. Dabei wird, sobald mehrere Scandurchläufe für eine Task durchgeführt wurden, in den Berichten ein Sicherheitstrend angegeben. Dieser Trend sagt aus, ob sich die Sicherheit im letzten Scandurchlauf zu den vorherigen Durchläufen eher verschlechtert, verbessert oder nicht verändert hat. Die weiteren Bestandteile eines Reports sind die gefundenen Sicherheitslücken, zusammen mit einer Beschreibung der Sicherheitslücke und einer Einschätzung der Gefahr die von dieser ausgeht.

4.2 Komponenten

Das Open Vulnerability Assessment System besteht aus mehreren Komponenten. Für den eigentlichen „klassischen“ [1] Scan-Betrieb der Version OpenVAS 3.x sind nur die Kernkomponenten Libraries, Scanner und Client nötig. Zusätzlich zu diesen Systembestandteilen gibt es noch weitere Komponenten, die den Funktionsumfang steigern und das Administrieren eines OpenVAS-Servers erleichtern.

Im Folgenden werden die Aufgabenbereiche und das Zusammenspiel der einzelnen Komponenten beschrieben. Dabei wird wie oben bereits erwähnt OpenVAS-4 als Grundlage verwendet. Alle Bestandteile von OpenVAS sind in Abbildung 4.1 abgebildet. Die OpenVAS-Architektur ist in drei Schichten aufgeteilt. In der oberen Schicht befinden sich die Clients, die sowohl auf den OpenVAS Manager als auch auf den OpenVAS Administrator zugreifen. Zusätzlich zu OpenVAS Manager und Administrator befindet sich noch der OpenVAS Scanner in der mittleren Serviceschicht. In der unteren (Daten-)Schicht befindet sich eine Datenbank mit den Konfigurationen und Scanberichten sowie die Sammlung an NVTs.

4.2.1 OpenVAS Libraries

Im Libraries Paket sind alle OpenVAS-Funktionen enthalten die von mehreren Komponenten benutzt werden. Dazu gehört unter anderem die Funktionalität, die für das Interpretieren der NASL-Skripte vorgesehen ist. Die Komponente Libraries taucht in Abbildung 4.1 nicht auf, da es eine Funktionssammlung ist, die von allen Komponenten aus der Serviceschicht verwendet wird.

[1] http://www.openvas.org/about-software.html

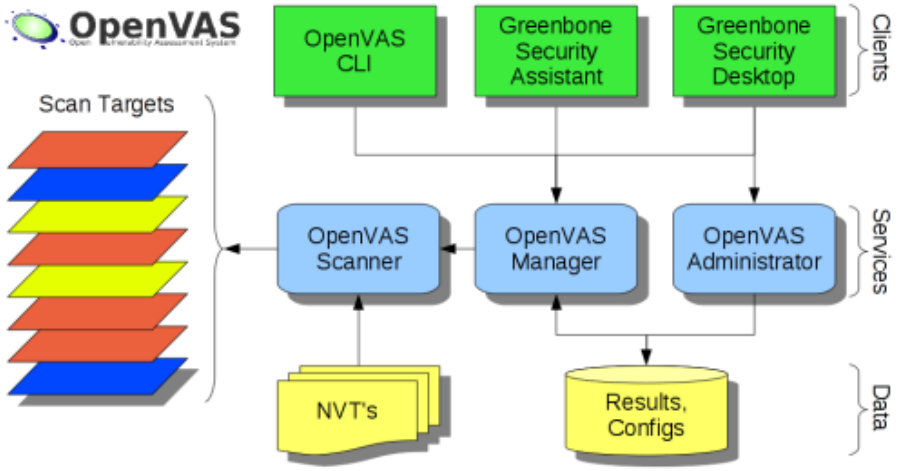


Abbildung 4.1: Komponenten von OpenVAS Quelle: http://www.openvas.org/index.html

4.2.2 OpenVAS Scanner

Der Scanner ist für das eigentliche Ausführen der Sicherheitsscans verantwortlich, gesteuert wird der Scanner vom OpenVAS Manager. Der Scanner benutzt die Sammlung von NVTs um die Ziele (links in Abbildung 4.1) zu überprüfen.

4.2.3 OpenVAS Manager

Der Manager ist die Verwaltungsinstanz von OpenVAS und übernimmt alle Aufgaben im Bezug auf Verwaltung von Scanaufträgen, Scankonfiguration und Scanberichten. Dabei kommuniziert der Manager über das OpenVAS Transfer Protocol (OTP) mit dem Scanner und initiiert so die eigentlichen Scans. Der Manager selbst bietet eine XML basierte Schnittstelle, die als OpenVAS Management Protocol (OMP) bezeichnet wird (4.3.3).

4.2.4 OpenVAS Administrator

Der Administrator übernimmt die Verwaltung von Benutzerkonten innerhalb von OpenVAS. Zur Benutzerverwaltung gehört auch das Management von Zugriffskontrollen um bestimmte Scankonfigurationen nur für spezielle Benutzer verwendbar zu machen.

4.2.5 Greenbone Security Assistant

Beim Greenbone Security Assistant handelt es sich um ein Web-Front-End zum Administrieren eines OpenVAS-Servers. Der Zugriff auf das Front-End findet per HTTPS

statt. Auf der Serverseite verwendet der Greenbone Security Assistant microhttpd und XSLT zum Generieren der dynamischen Webseiten. In Abbildung 4.2 ist das Webinterface abgebildet, der Screenshot zeigt den Dialog zur Auswahl von Plug-in Kategorien.

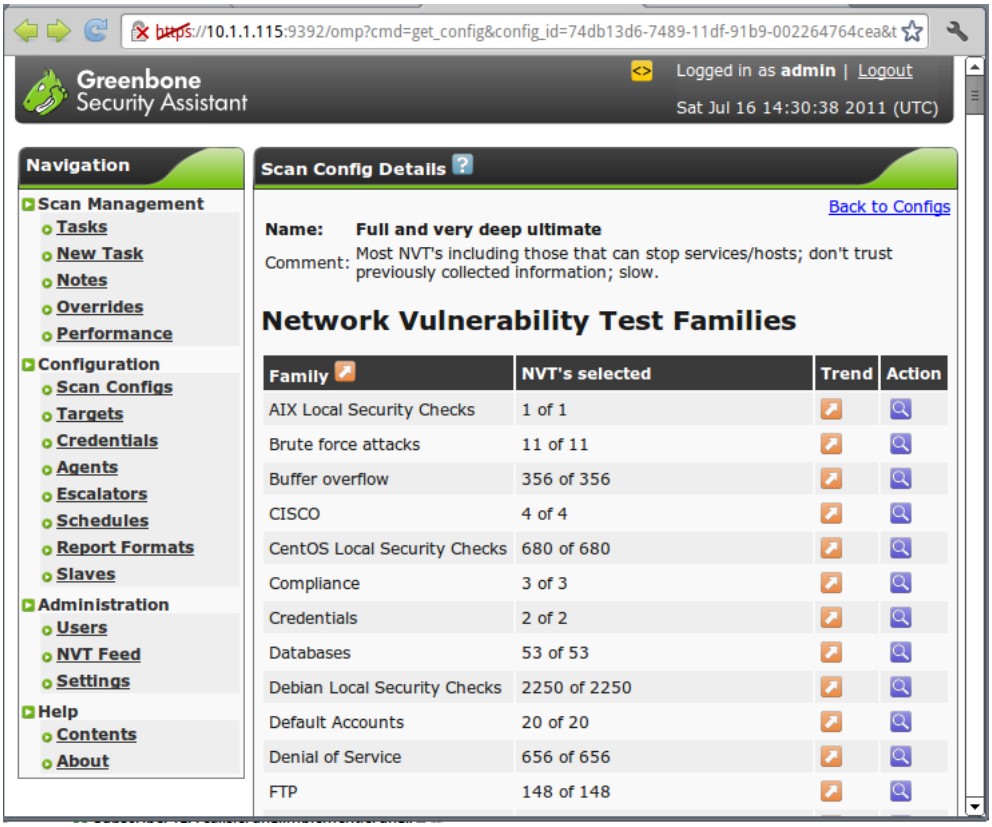


Abbildung 4.2: Greenbone Security Assistant

4.2.6 Greenbone Security Desktop

Analog zum Greenbone Security Assistant bietet der Greenbone Security Desktop die Möglichkeit zur Verwaltung eines OpenVAS-Servers. Anstatt eine Webinterface als Schnittstelle zu verwenden stellt der Greenbone Security Desktop eine eigenständige Desktopanwendung, basierend auf dem Qt-Framework, dar. Die Kommunikation mit dem OpenVAS-Server findet über das OpenVAS Management Protocol statt.

4.2.7 OpenVAS CLI

Mit dem OpenVAS CLI besteht die Möglichkeit einen OpenVAS-Server über die Kommandozeile zu steuern. Die Kommunikation findet dabei ebenfalls über das OpenVAS

Management Protocol statt.

4.3 Protokolle

Die einzelnen OpenVAS Komponenten kommunizieren untereinander und mit der Außenwelt über Protokolle, diese Protokolle werden in diesem Abschnitt kurz vorgestellt. In Abbildung 4.3 ist dargestellt welche Protokolle zwischen den Komponenten verwendet werden.

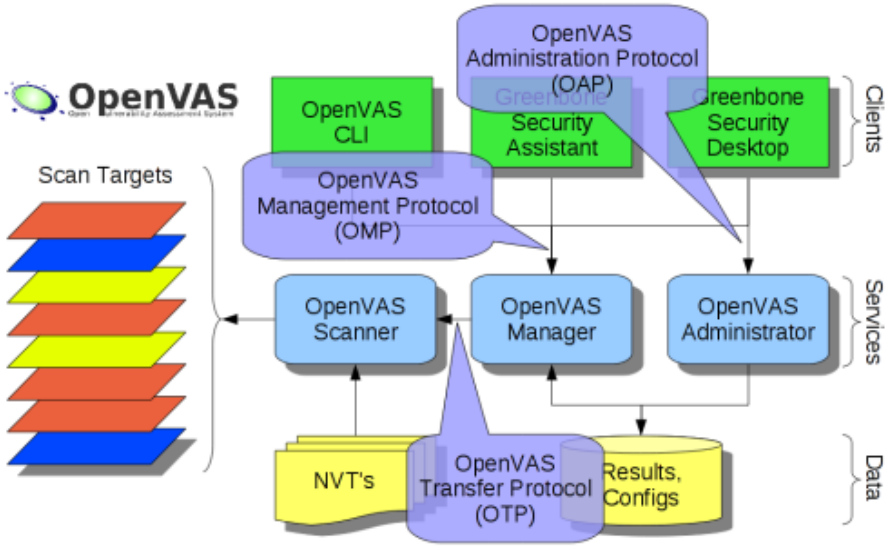


Abbildung 4.3: OpenVAS-Protokolle Quelle: http://www.openvas.org/about-software.html

4.3.1 OpenVAS Transfer Protocol

Das OpenVAS Transfer Protocol (OTP) ist aus dem Nessus Transfer Protocol (NTP) hervorgegangen. In der Version 2.x wurde OTP noch zur Kommunikation von OpenVAS Desktopclient und OpenVAS-Server eingesetzt. In der aktuellen Version 4.x wird OTP nur noch intern zur Steuerung des OpenVAS Scanners durch den OpenVAS Manager eingesetzt. Auf Empfehlung des OpenVAS-Projektes sollten keine neuen OTP Clients mehr entwickelt werden, da dieses Protokoll in späteren Versionen von OpenVAS eventuell nicht mehr zur Verfügung stehen wird.

4.3.2 OpenVAS Administration Protocol

Das OpenVAS Administration Protocol (OAP) ist für die Kommunikation mit dem OpenVAS Administrator vorgesehen.

4.3.3 OpenVAS Management Protocol

Über das OpenVAS Management Protocol ist eine einfacher Zugriff auf den OpenVAS Manager möglich. Das Protokoll basiert auf XML und ist durch ein RELAX NG Schema (2.1.2) spezifiziert. Die Kommunikation findet dabei über eine TLS/SSL-Verbindung statt. Das Protokoll selbst ist weitestgehend (bis auf den Authentisierungsstatus) zustandslos. OMP-Nachrichten bestehen nur aus dem XML-Dokument welches ein Kommando repräsentiert, es werden keine sonstigen Daten (z.B. Protokollheader) übertragen. Das Kommunikationsmodell ist ein einfaches Request-Response-Modell in dem die gesamte Kommunikation vom Client ausgeht und der Server immer direkt auf eine Anfrage antwortet.

Mit OMP können alle Konfigurationsschritte die für das Anlegen, Bearbeiten und Löschen von Sicherheitsscans nötig sind durchgeführt werden. Im folgenden wird eine kleine Auswahl von OMP-Kommandos vorgestellt eine vollständige Liste aller Kommandos kann auf der Webseite (`http://www.openvas.org/omp-2-0.html`) des OpenVAS-Projektes gefunden werden.

authenticate: Mit diesem Kommando kann sich ein Client beim OpenVAS-Server authentifizieren. Aktuell geschieht dies durch die Übertragung von Benutzername und Passwort, weitere Authentisierungsmechanismen sind aber für die Zukunft geplant.

get_reports: Mit diesem Kommando werden alle im OpenVAS-Server gespeicherten Scanberichte abgerufen.

create_target: Diese Kommando erstellt ein neues Target in der OpenVAS-Konfiguration.

start_task: Eine bereits im OpenVAS-Server bestehende Task kann mit diesem Kommando gestartet werden.

Beispiel

Aufgrund der Einfachheit von OMP kann das Protokoll mit Tools wie OpenSSL benutzt werden. Der folgende Auszug zeigt eine manuelle Verbindung mit dem OpenSSL Kommandozeilen-Tool, bei der die Version des OpenVAS Managers abgefragt wird. Alle anderen Kommandos können auf die selbe Weise ausgeführt werden.

```
$ openssl s_client -connect 10.1.1.115:9390

[...] SSL-Handshake Informationen [...]
```

```
---
<authenticate>
<credentials>
<username>admin</username>
<password>pazzw0rd</password>
</credentials>
</authenticate>
<authenticate_response status="200" status_text="OK"/><get_version/>
<get_version_response status="200" status_text="OK">
<version>2.0</version></get_version_response>DONE
$
```

4.4 Scanablauf

Beim Überprüfen eines Ziels führt OpenVAS immer die selben Schritte aus. Die Anordnung dieser einzelnen Schritte soll sicherstellen, dass die Überprüfung so effizient wie möglich ist und keine Ressourcen, z.B. für ausgeschaltete Rechner, verschwendet werden. Im Folgenden wird jede dieser Phasen kurz erläutert.

Host-Detection: In diesem Schritt wird festgestellt ob das Ziel erreichbar ist. Für diese Phase stehen OpenVAS die Möglichkeit einfacher ICMP-Echo-Request zur Verfügung, es können aber auch Tools wie Nmap von OpenVAS verwendet werden.

Service-Detection: Im zweiten Schritt wird überprüft welche Dienste auf dem Ziel laufen und welche Ports geöffnet sind. Für diese Überprüfung stehen OpenVAS mehrere Methoden zur Verfügung. Die wohl am häufigsten eingesetzte Variante ist auch hier der Portscanner Nmap, aber auch andere Portscanner wie PortBunny stehen zur Wahl.

Informationen sammeln: In dieser Phase wird versucht so viel Informationen über das Ziel zu sammeln wie möglich. Zu diesen Informationen gehören vor allem die Versionen der laufenden Dienste und im Fall von Local Security Checks die Versionen der vorhandenen Anwendungen.

Verwundbarkeitstests: Nachdem nun bekannt ist welche Dienste auf dem Ziel laufen und welche Anwendungen installiert sind, können in dieser Phase die geeigneten Plug-ins zur eigentlichen Sicherheitsüberprüfung ausgewählt und angewendet werden.

Denial-of-Service-Tests: Nachdem die normalen Sicherheitstests, welche das Zielsystem nicht beeinflussen sollen, durchgeführt wurden, können in dieser Phase noch zusätzliche Denial-of-Service (DoS) Attacken ausgeführt werden. Dabei wird versucht das Ziel entweder mit speziellen Anfragen oder einer großen Anzahl von Anfragen

zum Absturz zu bringen.

Bericht: In der letzten Phase werden alle Erkenntnisse der vorherigen Phasen zusammengefasst und zu einem Bericht aufgearbeitet.

5 Anforderungsanalyse

In diesem Kapitel wird noch einmal kurz das Ziel dieser Arbeit dargestellt. Danach werden dann die Anforderungen, welche an das Konzept und die Implementierung gestellt werden festgehalten.

5.1 Ziele

Nachdem die Grundlagen für den Umgang mit den beiden Basistechnologien OpenVAS und IF-MAP nun gelegt wurden, soll in diesem Abschnitt die eigentliche Zielsetzung dieser Arbeit dargestellt werden.

OpenVAS trägt durch die Aufdeckung von Sicherheitslücken, bevor diese aktiv ausgenutzt werden, maßgeblich zur Steigerung der Sicherheit und Integrität von Netzen bei. IF-MAP wurde entwickelt um sicherheitsrelevante Ereignisse innerhalb eines Netzwerkes zentral zu sammeln, zu strukturieren und für andere Netzwerkgeräte zur Verfügung zu stellen. Aufgrund des Anwendungsbereiches „Netzwerksicherheit“ beider Bestandteile bietet es sich an, eine Integration von OpenVAS in eine MAP-Infrastruktur zu untersuchen.

Das Ziel dieser Arbeit ist es, eine Möglichkeit zu entwickeln, die es OpenVAS ermöglicht die gewonnen Informationen an einen MAP-Server zu senden. Die zentrale Verfügbarkeit dieser Daten und der einfache standardisierte Zugriff ermöglicht es z.B. einer IF-MAP fähigen Firewall auf diese Daten zuzugreifen und basierend darauf Entscheidungen zu treffen.

Ein weiteres Ziel dieser Arbeit besteht darin OpenVAS in die Lage zu versetzen Metadaten aus dem MAP-Server zu abonnieren und aufgrund dieser Daten dynamisch zu reagieren.

5.2 Kommunikationsschnittstelle von OpenVAS

An die verwendete Kommunikationsschnittstelle zu OpenVAS ergeben sich mehrere Anforderungen.

Die wichtigste Anforderung ist, dass es sich um eine stabile Schnittstelle handelt, die möglichst wenig Änderungen erfährt und über einen langen Zeitraum unverändert bleibt.

Da es sich bei den Informationen, die aus OpenVAS gewonnen werden können, um potentiell noch offene Sicherheitslücken, auf gegebenenfalls sehr kritischen und sensiblen Systemen handelt, ist die Vertraulichkeit dieser Informationen eine weitere extrem wichtige Anforderung. Innerhalb des OpenVAS-Systems sind diese Informationen durch die Sicherheitsmaßnahmen die vom OpenVAS getroffen werden geschützt. Falls für die Anbindung an eine MAP-Infrastruktur der Transport dieser Informationen nötig sein sollte, muss dieser Transport unbedingt über eine verschlüsselte Verbindung geschehen. Die Notwendigkeit der Verschlüsselung bezieht sich nicht auf die eigentliche IF-MAP-Verbindung, diese muss nach Spezifikation immer verschlüsselt sein. Diese Anforderung bezieht sich auf den Transport von Schwachstelleninformationen aus OpenVAS heraus über eine dritte Instanz.

5.3 IF-MAP

Aus den in der IF-MAP-Spezifikation definierten Eigenschaften des Protokolls ergeben sich mehrere Anforderungen.

Da der MAP-Server nur den aktuelle Zustand der Metadaten speichert, die von den Clients veröffentlicht werden, können aus Sicht des MAP-Servers keine Aussagen zur Korrektheit oder Aktualität der Metadaten gemacht werden. Daraus folgt, dass der Client sicherstellen muss, dass Metadaten die von ihm veröffentlicht wurden, auch wieder von ihm gelöscht werden, sobald sie nicht mehr aktuell sind oder die Aktualität nicht mehr sicher gewährleistet werden kann.

Außerdem muss vom Client darauf geachtet werden, dass keine Metadaten doppelt veröffentlicht werden. Bei Metadaten, die das Attribut `singleValue` tragen ist dies weniger von Bedeutung. Bei diesem Metadatentyp würden immer die aktuellsten Metadaten ihre bereits veröffentlichten Vorgänger ersetzen. Bei Metadaten mit dem Attribut `multiValue` hingegen muss vom Client beachtet werden, dass diese nicht redundant an den MAP-Server gesendet werden oder dass ein übermäßiges Fluten mit Metadaten den MAP-Server überlastet.

Eine weitere wichtige Anforderung an den Client ist, dass auf Verbindungsabbrüche zum MAP-Server angemessen reagiert werden muss. Ein spezifikationskonformer MAP-Server löscht sämtliche Metadaten, die von einem Client veröffentlicht wurden, wenn dieser eine neue Verbindung aufbaut. Das heißt der Client ist dafür verantwortlich nach einem erneutem Verbindungsaufbau die Metadaten, die er bereits in einer vorherigen Sitzung veröffentlicht hat wieder an den MAP-Server zu senden, falls sie aktuell noch gültig sind.

5.4 Abbildung von OpenVAS-Daten in IF-MAP

In der IF-MAP-Spezifikation [Tru10b] wird eine Menge von Standard-Metadaten beschrieben. Zu dieser Beschreibung gehört neben dem Namen, der Attribute und dem Verwendungszweck auch eine Empfehlung mit welchem Identifiertypen der jeweilige Metadatentyp zu verwenden ist.

Aus diesen Metadaten muss zunächst eine Teilauswahl getroffen werden, die in der Semantik auf die selben Anwendungsbereiche abzielt wie die Informationen die OpenVAS erzeugt. Wichtig hierbei ist auch, dass die gewählten Metadatentypen möglichst viele der Schwachstelleninformationen korrekt und möglichst vollständig abbilden können.

5.5 Reaktionen von OpenVAS auf IF-MAP-Daten

Damit OpenVAS auf Metadaten, die von anderen Netzwerkgeräten veröffentlicht wurde, reagieren kann, muss auch hier zunächst eine Vorauswahl getroffen werden. Der wichtigste Punkt für die Auswahl hierbei ist, dass die Metadaten und die verbundenen Identifier genügend Informationen liefern müssen, um OpenVAS korrekt konfigurieren zu können. Auch hier gilt, dass zu dem Zeitpunkt an dem die Metadaten, die als Grundlage für die dynamische OpenVAS-Konfiguration dienten, nicht mehr aktuell sind eine Reaktion erfolgen muss. Diese Reaktion stellt z.B. das Zurücksetzen der Konfiguration dar.

5.6 Zusammenfassung der Anforderungen

Im Folgendem werden noch einmal alle Anforderungen zusammengefasst.

5.6.1 Funktionale Anforderungen

Veröffentlichen von OpenVAS-Daten

- Korrekte Abbildung der OpenVAS-Informationen auf IF-MAP-Metadaten.
- Möglichst vollständige Abbildung der OpenVAS-Informationen auf IF-MAP-Metadaten.
- Korrekte Aktualisierung von veralteten Metadaten.
- Löschen von nicht mehr aktuellen Metadaten.
- Erneutes Senden von noch aktuellen Metadaten nach einem Verbindungsabbruch.

Reaktion von OpenVAS auf IF-MAP-Daten

- Möglichst vollständige Konfiguration von OpenVAS.
- Reaktion auf das Löschen von Metadaten.

5.6.2 Nicht-funktionale Anforderungen

- Verwendung einer möglichst stabilen Schnittstelle.
- Gewährleistung der Schutzziele Integrität, Authentizität und Vertraulichkeit für die übertragenen Daten.
- Robustheit gegen Verbindungsabbrüche.
- Effiziente Verarbeitung und Weiterleitung von Informationen.

6 Konzept der Anbindung von OpenVAS

In diesem Kapitel wird das Konzept, welches für die Implementierung verwendet wurde näher erläutert.

Zusätzlich zu der gewählten Möglichkeit für den Zugriff auf OpenVAS, sollen hier kurz noch die Alternativen aufgezeigt werden. Danach wird untersucht welche Daten aus OpenVAS im MAP-Server veröffentlicht werden sollen, wie diese auf IF-MAP-Datentypen abgebildet werden und auf welche Weise sie dann schließlich veröffentlicht werden. Außerdem wird ein Szenario beschrieben in dem OpenVAS als Subscriber agiert und selbst Metadaten aus einem MAP-Server abonniert. Zum Abschluss dieses Kapitels wird noch kurz die Auswahl von Technologien für die Implementierung beschrieben.

6.1 OpenVAS Zugriff

Zur Datengewinnung aus OpenVAS heraus ergeben sich mehrere Möglichkeiten. Von diese Möglichkeiten werden drei in diesem Abschnitt vorgestellt. Jeder dieser Vorschläge wird dabei noch auf die jeweiligen Vor- und Nachteile untersucht.

Möglichkeit 1: OpenVAS Patch

Bei dieser Variante wird direkt in den Quellcode von OpenVAS eingegriffen. Dabei wird die Funktionalität so erweitert und angepasst, dass eine OpenVAS-Instanz selbstständig in der Lage ist mit einem MAP-Server zu kommunizieren.

Zu den Vorteilen dieser Variante gehören zum einen, dass die Kommunikation über keine dritte Instanz erfolgen muss und somit die Vertraulichkeit der Informationen durch die OpenVAS-Komponenten sichergestellt wird.

Ein großer Nachteil dieser Variante besteht darin, dass dieser Patch die Gesamtstabilität von OpenVAS gefährden könnte. Aufgrund der großen Codebasis von OpenVAS ist für diese Lösung eine große Einarbeitungszeit in den Quelltext nötig. Außerdem ist die direkte Einbindung in den Quelltext von OpenVAS mit einer starken Koppelung an OpenVAS verbunden. Sollten sich Implementierungsdetails innerhalb des OpenVAS-Projektes ändern ist es eventuell nötig den eingebrachte Patch auch anzupassen. Dies bedeutet unter Umständen einen großen Wartungsaufwand, der nicht selbst kontrollierbar ist, sondern viel mehr vom OpenVAS-Projekt abhängt.

Zusammenfassung der Vorteile (+) und Nachteile (-):

+ Direkte Kommunikation von OpenVAS mit dem MAP-Server.
- Gefährdung der OpenVAS-Stabilität.
- Unter Umständen sehr großer Wartungsaufwand.

Möglichkeit 2: Logfile Analyse

Diese Variante setzt auch sehr dicht an OpenVAS an, aber ohne in den Quellcode einzugreifen. Die Hauptidee besteht darin eine unabhängige Anwendung zu entwickeln, die kontinuierlich die Logfiles von OpenVAS analysiert und aus diesen Logfiles dann relevanten Informationen extrahiert und an einen MAP-Server sendet.

Der Vorteil dieser Lösung liegt darin, dass die eigentliche OpenVAS-Funktionalität nicht beeinträchtigt wird.

Diese Lösung hat allerdings auch Nachteile. Einer der größten Nachteile ist, dass ein Logfile sehr stark mit dem System, welches die Logfiles erstellt gekoppelt ist. Dies beinhaltet vor allem, dass das Format der Logfiles nicht standardisiert ist und sich oft ändern kann.

Weiterhin sind die Logfiles auch nur direkt im Dateisystem auf dem System verfügbar welches die Logs erstellt. Natürlich gibt es auch Wege um diesen Nachteil abzuschwächen, z.B. der Einsatz von syslog. Jedoch wird gerade durch IF-MAP versucht alte Systeme wie syslog abzulösen und durch eine einheitliche Gesamtlösung zu ersetzen.

Zusammenfassung der Vorteile (+) und Nachteile (-):

+ Keine Beeinträchtigung von OpenVAS.
- Starke Koppelung an das Logfile-Layout.
- Bindung an das Dateisystem des OpenVAS-Servers.

Möglichkeit 3: OMP-Client

Eine weitere Möglichkeit besteht in der Entwicklung eines OMP-Clients. Diese selbstständige Anwendung verwendet das OpenVAS Management Protocol um Informationen aus einem OpenVAS-Server auszulesen, aufzubereiten und an einen MAP-Server zu senden.

Der große Vorteil dieser Lösung besteht in der klaren Schnittstelle. Das OpenVAS Management Protocol wurde entworfen um externen Anwendungen den einheitlichen Zugriff auf einen OpenVAS-Server zu ermöglichen, deshalb ist es sehr wahrscheinlich, dass diese

Schnittstelle über einen langen Zeitraum keine Änderungen erfährt, die inkompatibel zu der bisherigen Schnittstelle wären.

Ein nicht zu unterschätzender Nachteil dieser Lösungsmöglichkeit ist die Entwicklung einer eigenständigen Anwendung, was einen initialen Entwicklungsaufwand bedeutet, aber auch später Wartungsaufwand nach sich zieht.

Zusammenfassung der Vorteile (+) und Nachteile (-):

+ Klar definierte Schnittstelle.
- Großer initialer Entwicklungsaufwand.
- Wartungsaufwand.

Begründung der Auswahl

Für die exemplarische Realisierung fällt die Wahl auf Möglichkeit 3. Alle drei Möglichkeiten sind nicht völlig frei von nachträglichem Wartungsaufwand. Bei Möglichkeit 1 ist der Wartungsaufwand allerdings nicht absehbar, da sich OpenVAS Implementierungsdetails zu jeder Zeit ändern können. Eine positive Auswirkung auf die Wartbarkeit wird bei Möglichkeit 3, wegen der klaren Schnittstelle zur Datengewinnung, erwartet.

Ein weiterer Grund, welcher für Möglichkeit 3 spricht, ist die Definition beider Protokolle in XML. In vielen aktuellen Programmiersprachen ist die Verarbeitung von XML sehr gut durch Bibliotheken unterstützt oder sogar Teil der Sprache selbst. Außerdem wird durch die Kommunikation mit XML nicht festgelegt in welcher Programmiersprache die Umsetzung realisiert werden muss.

6.2 Veröffentlichen von OpenVAS Daten

In diesem Abschnitt soll erläutert werden welche Schritte für das Veröffentlichen von Metadaten nötig sind. Dafür wird zunächst eine Auswahl von Daten getroffen die dann auf passende IF-MAP-Typen abgebildet werden.

6.2.1 Auswahl von Daten aus OpenVAS

Die Berichte von OpenVAS enthalten eine große Anzahl von Informationen. Die folgende Auflistung zeigt die Wichtigsten dieser Informationen, welche in der Implementierung verarbeitet werden sollen.

UUID: Eine ID, welche die gefundene Schwachstelle im OpenVAS-System eindeutig identifiziert.

Subnet: Die IP-Adresse des Netzwerkes in dem sich der Rechner mit der gefundenen Schwachstelle befindet.

Host: Der Name oder die IP-Adresse des Rechners auf dem die Schwachstelle gefunden wurde.

Port: Der Port auf dem der Dienst mit der Schwachstelle aktiv ist.

Name: Der Name der Schwachstelle.

CVE: Eine eindeutige Kennung für die gefundene Schwachstelle (siehe dazu Abschnitt 2.5).

CVSS-BASE: Eine Einschätzung wie schwerwiegend diese Schwachstelle ist, ähnlich zur Richterskala.

Risk-Factor: Eine Einschätzung (in einem kleineren Wertebereich als CVSS-BASE) wie schwerwiegend diese Schwachstelle ist, der Wertebereich reicht von NONE bis CRITICAL.

Description: Eine Beschreibung der Schwachstelle.

6.2.2 Abbildung von OpenVAS-Daten in IF-MAP

Die im vorherigen Abschnitt ausgewählten Daten aus OpenVAS müssen nun noch sinnvoll auf einen IF-MAP-Datentyp abgebildet werden. Für diese Abbildung gibt es zwei verschiedene Ansätze die nun vorgestellt werden.

Standard IF-MAP

Eine große Menge an Standard-Metadatentypen aus [Tru10b] kann für das Veröffentlichen von OpenVAS-Daten nicht verwendet werden, da diese eine andere Semantik haben und andere Informationen transportieren sollen. Außerdem sind in der Spezifikation für einige Metadatentypen explizite Regeln enthalten, an welche Identifier diese assoziiert werden *dürfen*. Für andere Metadatentypen gibt es Empfehlungen an welche Identifier sie assoziiert werden *sollten*. Zu den aus diesen Gründen bereits ausgeschlossenen Metadatentypen gehören:

- `access-request-device`
- `access-request-ip`
- `access-request-mac`
- `authenticated-as`
- `authenticated-by`
- `device-ip`

- `discovered-by`
- `enforcement-report`
- `ip-mac`
- `layer2-information`
- `location`
- `requset-for-investigation`
- `role`
- `wlan-information`
- `unexpected-behavior`

Es verbleiben folgende Metadaten und ihre in der Spezifikation definierten Verwendung:

capability: Wird für `access-request`-Identifier empfohlen und repräsentiert bestimmte Rechte eines Endgerätes, welche anwendungsabhängig definiert werden können. Diese Rechte oder Privilegien können dann von anderen Endgeräten genutzt werden um über Zugriffsanfragen des verbundenen Identifiers zu entscheiden.

device-attribute: Wird für `access-request`- und `device`-Identifier empfohlen. Diese Metadaten stellen (anwendungsabhängig) bestimmte Eigenschaften der verbundenen Identifier dar. In der Spezifikation werden als Beispiel „aktiver Virenscanner“ und „aktueller Patchstand“ genannt.

device-characteristic: Wird für `device`-Identifier kombiniert mit `access-request`-, `ip-address`- oder `mac-address`-Identifier empfohlen und stellt im Gegensatz zu `device-attribute`-Metadaten die Eigenschaften eines Endgerätes dar, die anwendungsunabhängig sind. Dazu gehören z.B. der Hersteller des Endgerätes oder das eingesetzte Betriebssystem.

event: Empfohlen für `access-request`-, `identity`-, `ip-address`, oder `mac-address`-Identifier. Dieser Metadatentyp repräsentiert ein Ereignis innerhalb des Netzwerkes welches aus Sicherheitssicht von Bedeutung ist. Unter anderem ist vorgesehen mit diesen Metadaten Schwachstellen eine Endgerätes zu signalisieren.

Es zeigt sich also, dass es mit `event`-Metadaten genau einen Metadatentyp gibt der benötigte Semantik besitzt um die von OpenVAS gefundenen Schwachstellen abbilden zu können. Das XML Schema definiert die im Folgenden gezeigten Elemente und Attribute für `event`.

name: Dieses Element enthält den Namen des Ereignisses. In der Implementierung wird der Name der Schwachstelle direkt in dieses Element abgebildet.

discovered-time: Dieses Element enthält den Zeitpunkt an dem das Ereignis zum ersten mal aufgetreten ist. Für dieses Element kann der Zeitstempel des Scanberichts

verwendet werden.

discoverer-id: Dieses Element enthält eine eindeutige ID, die den Entdecker des Ereignisses identifiziert. In diese Element kann Wahlweise der Hostname oder die IP-Adresse des OpenVAS-Servers eingetragen werden.

magnitude: Dieses Element stellt den Einfluss des Ereignisses als Zahlenwert von 0 (keinen Einfluss) bis 100 (sehr hoher Einfluss) dar. Der mit 10 multiplizierte Wert von CVSS-BASE erfüllt die Semantik dieses Elementes.

confidence: Dieses Element beschreibt, wie sicher sich der Client darüber ist, dass dieses Ereignis von Bedeutung und vor allem noch aktuell ist. Für dieses Element gibt es keine direkte Entsprechung, außerdem bestehen beim Einschätzen der Aktualität weitere Probleme die in Abschnitt 9.2 erläutert werden.

significance: Dieses Element beschreibt, wie wichtig oder dringlich das Ereignis ist. In der Implementierung wird der Risk-Factor Wert direkt in dieses Element abgebildet.

type: Dieses Element beschreibt den Typ des Ereignisses. Für dieses Element sind die Werte `p2p`, `cve`, `botnet infection`, `worm infection`, `excessive flows`, `behavioral change`, `policy violation` und `other` vorgesehen. Für die Implementierung wird nur der Wert `cve` verwendet. Laut Spezifikation muss bei der Verwendung von `cve` auch das Element `vulnerability-uri` gesetzt sein.

other-type-definition: Wird für das Element `type` der Werte `other` verwendet, muss in diesem Element definiert werden um was für einen speziellen Typen es sich handelt.

information: Dieses Element enthält eine menschenlesbare Repräsentation des Events. In der Implementierung wird in diesem Element die Beschreibung der Schwachstelle abgebildet.

vulnerability-uri: Dieses Element enthält die eindeutige CVE-Kennung falls als Typ `cve` angegeben wurde.

Nachdem die Wahl auf `event`-Metadaten gefallen ist, muss nun noch festgelegt werden mit welche Identifiern diese Daten verbunden werden. Da für `event`-Metadaten unter anderem der Identifier `ip-address` empfohlen wird und der OpenVAS-Bericht für jede Schwachstelle eine IP-Adresse enthält werden alle Schwachstellen mit `ip-address`-Identifiern veröffentlicht. Ein Beispiel wie ein Graph nach dem Veröffentlichen von OpenVAS-Daten aussehen könnte ist in Abbildung 6.1 dargestellt.

Erweitertes XML Schema

Eine andere Möglichkeit um den Verlust von Informationen zu umgehen und eine vollständige Abbildung der Daten zu ermöglichen, ist das Entwerfen eines eigenen XML Schemas für diesen Anwendungsbereich. IF-MAP wurde so entworfen, dass zusätzlich zu

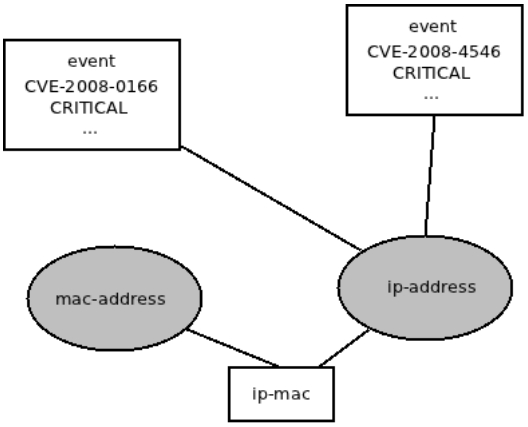


Abbildung 6.1: Event-Metadaten im Graphen

den bestehende Metadatentypen beliebige spezifische Erweiterungen eingebracht werden können. Die Vorteile eines speziellen XML Schemas werden in Abschnitt 9.2 erläutert.

6.2.3 Veröffentlichen der Metadaten

Die aus OpenVAS gewonnenen und dann zu IF-MAP-Daten konvertierten Schwachstellenberichte müssen anschließend auf dem MAP-Server veröffentlicht werden. Für die Veröffentlichung stehen potentiell zwei IF-MAP-Operationen zur Verfügung, die Daten können mit der `update`-Operation im MAP-Server gespeichert werden oder mit der `notify`-Operation nur den aktuell registrierten Subscribern gesendet werden.

In der aktuellen Version der IF-MAP-Spezifikation (Version 2.0) ist vorgesehen, dass `event`-Metadaten ausschließlich mit der `notify`-Operation veröffentlicht werden. Clients, die sich nicht nach Version 2.0 verhalten, müssen selbstständig nicht mehr aktuelle `event`-Metadaten, die mit der `update`-Operation veröffentlicht wurden, aus dem MAP-Server entfernen.

Für die Implementierung wird der Ansatz gewählt, dass frei konfiguriert werden kann ob eine Schwachstelle über die `update`- oder `notify`-Operation veröffentlicht wird. Die Entscheidungsvariable für eine der beiden Operationen ist die von OpenVAS vergebene Gefährdungsstufe der Schwachstelle. Durch diese Konfigurationsmöglichkeit ist es zum einen möglich die Implementierung als IF-MAP-Version 2.0 kompatibel zu betreiben. Zum anderen ermöglicht es aber auch hoch kritische Schwachstellen im MAP-Server abzulegen und so für Clients verfügbar zu machen, die sich erst später für diese `event`-Metadaten anmelden.

6.3 Dynamische Konfiguration von OpenVAS

Für die dynamische Konfiguration von OpenVAS gibt es eine Vielzahl von Anwendungsbereichen und Möglichkeiten. In der Implementierung wird nur eine dieser Möglichkeiten exemplarisch umgesetzt. Einen Ausblick in die zukünftige Entwicklung und welche anderen Anwendungsbereiche es noch gibt ist in Kapitel 9 zu finden.

Der Anwendungsfall der umgesetzt wird bezieht sich auf die Verwendung des Metadatentyps `request-for-investigation`. Dieser Metadatentyp wird von einem Policy Decision Point (PDP) (siehe Kapitel 3.1) an einen anderen Identifier assoziiert. Der PDP signalisiert damit, dass für eine sicherheitsrelevante Entscheidung weitere Informationen über diesen anderen Identifier benötigt werden. Die Aufgabe der Implementierung besteht nun darin sich beim MAP-Server für diesen Metadatentyp anzumelden. Sobald eine Poll-Operation neue Identifier mit `request-for-investigation` Metadaten liefert, soll für diesen neuen Identifier ein Scanauftrag bei OpenVAS eingereicht werden.

Um diese Anwendungsfall zu verdeutlichen ist in Abbildung 6.2 der Graph innerhalb des MAP-Servers dargestellt, der eine mögliche Ausgangssituation darstellt.

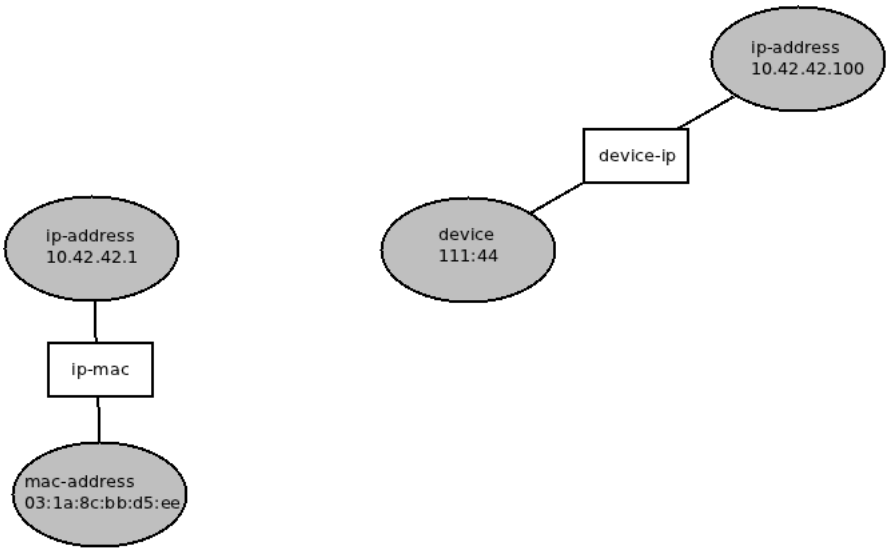


Abbildung 6.2: Anwendungsfall Subscriber - Ausgangsgraph

Der Identifier `device: 111:44` stellt dabei einen Policy Decision Point (PDP) dar und der Identifier `ip-address: 10.42.42.1` ein beliebiges Endgerät. Der Subscriber registriert sich nun für alle Metadaten vom Typ `request-for-investigation` die an den Identifier `device: 111:44` assoziiert werden. Der PDP selbst erfährt von der neuen IP-Adresse `10.42.42.1`, z.B. weil er Subscriber für alle Metadaten `ip-mac` ist, die ein DHCP veröffentlicht. Da der PDP keine sicherheitsrelevanten Kenntnisse über die IP-Adresse besitzt, veröffentlicht er selbst die Metadaten `request-for-investigation` zwischen der ihm

unbekannten IP-Adresse und sich selbst. Der nun entstandene Graph ist in Abbildung 6.3 dargestellt.

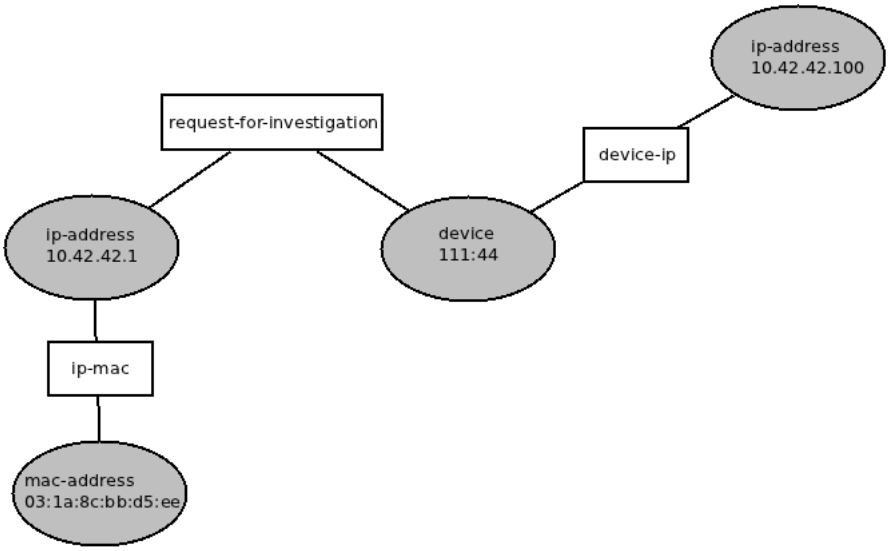


Abbildung 6.3: Anwendungsfall Subscriber - Metadaten veröffentlicht

Der OpenVAS-Client erhält nun diese neuen Informationen vom MAP-Server und kann für die IP-Adresse `10.42.42.1` einen neuen Scanauftrag in der OpenVAS-Konfiguration anlegen.

Nachdem OpenVAS dann dieses neue Ziel gescannt hat, werden alle gefundenen Schwachstellen wie in Abschnitt 6.2 beschrieben im MAP-Server veröffentlicht. Der PDP der zuvor die `request-for-investigation` Metadaten veröffentlicht hat, kann nun an den von OpenVAS veröffentlichten `event`-Metadaten erkennen welche Sicherheitslücken der ihm zuvor völlig unbekannte Rechner besitzt. Mit diesen Informationen können dann weitere Entscheidungen gefällt werden, z.B. das isolieren des Rechners mit gravierenden Schwachstellen durch einen PEP.

6.4 Plattform

In diesem Abschnitt werden die Technologien, die für die Umsetzung verwendet wurden erwähnt und ihre Auswahl kurz begründet werden.

Die Implementierung wird in der Programmiersprache Java realisiert. Der große Vorteil beim Einsatz von Java besteht darin, dass die XML-Verarbeitung durch eine Vielzahl von Komponenten in der Standardbibliothek hervorragend unterstützt wird.

Weiterhin gliedert sich eine weitere Java-Implementierung sehr gut in die bisher an der

Fachhochschule Hannover entstandenen Komponenten des IRON-Projektes [1] ein.

Als Implementierung eines MAP-Servers wird irond verwendet. Dieser MAP-Server wurde an der Fachhochschule Hannover im Rahmen des IRON-Projektes entwickelt. Neben omapd [oma] stellt irond die einzige Verfügbare Open Source Implementierung eines MAP-Servers dar.

6.5 Zusammenfassung

Abschließend wird nun noch einmal skizziert, was konkret umgesetzt wird. Es werden zwei Anwendungsfälle implementiert. Der erste Anwendungsfall ist für das Veröffentlichen von Daten aus OpenVAS heraus zuständig. Dabei wird auf OpenVAS mit Hilfe des OpenVAS Management Protocols zugegriffen. Veröffentlicht werden nur Metadaten vom Typ `event`. Diese Metadaten enthalten die wesentlichen Informationen die aus OpenVAS gewonnen werden können. Im weiteren Verlauf wird dieser Anwendungsfall auch als *Publisher* bezeichnet.

Der zweite Anwendungsfall setzt die dynamische Konfiguration von OpenVAS mit Hilfe von Metadaten aus einem MAP-Server um. Dabei werden nur `request-for-investigation`-Metadaten beachtet wie in Abschnitt 6.3 beschrieben. Dieser Anwendungsfall wird im weiteren Verlauf auch als *Subscriber* bezeichnet.

In Kapitel 8 wird unter anderem untersucht, ob das aufgestellte Konzept die Anforderungen aus Kapitel 5 erfüllt.

[1] https://trust.inform.fh-hannover.de/joomla/index.php/projects/iron

7 Realisierung der Anbindung von OpenVAS

In diesem Kapitel wird die entstandene Implementierung einer experimentellen Anbindung von OpenVAS vorgestellt. Dabei wird hauptsächlich auf das Design der Anwendung eingegangen. Für die wichtigsten Bereiche der Anwendungen wird zusätzlich die umgesetzte Implementierung des Designs in Abschnitt 7.2 besprochen und mit Codeausschnitten verdeutlicht.

In Kapitel 8 wird untersucht, ob die Anforderungen, welche in Kapitel 5 aufgestellt wurden, von der im Folgenden vorgestellten Realisierung erfüllt werden.

Die Abbildung der Funktionalität für den Anwendungsfall Subscriber und den Anwendungsfall Publisher findet in zwei separaten Programmen statt. Diese beiden Programme teilen sich allerdings, wegen der großen Menge an gleicher Funktionalität, eine gemeinsame Basis, deshalb werden beide Anwendungen gemeinsam in diesem Kapitel erläutert.

7.1 Design

Beide Anwendungen sind in zwei logisch voneinander getrennte Schichten aufgeteilt. Eine Schicht, die Kommunikationsschicht, übernimmt dabei alle notwendigen Operationen, die für die Netzwerkkommunikation notwendig sind. Über der Kommunikationsschicht ist die Anwendungsschicht angeordnet. In der Anwendungsschicht befindet sich die eigentliche Anwendungslogik welche die Anwendungsfälle umsetzt. Diese Aufteilung ermöglicht eine lose Koppelung zwischen den beiden Schichten und damit eine unabhängige Weiterentwicklung oder Modifikation einer der Schichten ohne die andere zu beeinflussen.

Der genaue Aufbau der einzelnen Schichten und die Spezifikation der Dienste, die diese Schichten anbieten oder nutzen, wird in den folgenden Abschnitten beschrieben.

Da beide Anwendungsfälle abhängig von einer Verbindung zu einem OpenVAS und zu einem MAP-Server sind, wurde diese Funktionalität in der Oberklasse `VasmapComponent` implementiert. Diese abstrakte Klasse ist in der Lage Verbindungen zu diesen beiden Servern aufzubauen und im Falle eines gescheiterten Verbindungsaufbaus diese Aktion zu wiederholen. Die Architektur der Implementierung ist in Abbildung 7.1 dargestellt.

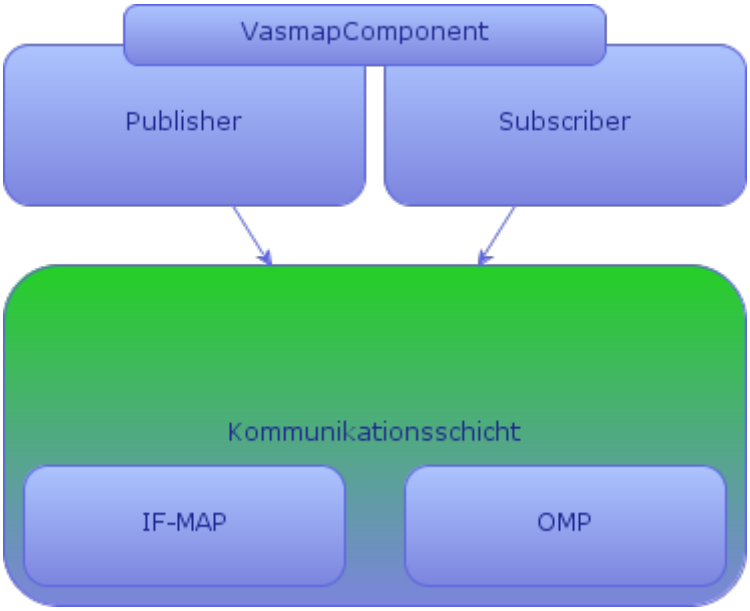


Abbildung 7.1: Aufbau der Implementierung

7.1.1 Publisher

Alle wichtigen Klassen der Anwendungsschicht des Publisher-Paketes sind in Abbildung 7.2 dargestellt. Im Folgenden werden alle Klassen und Interfaces in diesem Paket vorgestellt und ihr Aufgabenbereich erläutert.

Vulnerability: Diese Klasse repräsentiert eine Schwachstelle, die von OpenVAS gefunden wurde. Es sind Attribute für alle in Abschnitt 6.2.1 ausgewählten Daten vorhanden. Diese Klasse wird hauptsächlich in der Anwendungsschicht verwendet, um Entscheidungen für die weitere Verarbeitung von gefundenen Schwachstellen zu treffen.

VulnerabilityUpdate: Diese Klasse enthält eine Instanz der Klasse `Vulnerability`. Außerdem besitzt sie weitere Attribute, die beschreiben wie die referenzierte `Vulnerability`-Instanz auf IF-MAP-Metadaten und IF-MAP-Operationen abgebildet werden soll. Das Attribut `updateType` beschreibt, ob die referenzierte `Vulnerability`-Instanz mit der `notify`- oder `update`-Operation veröffentlicht werden soll. Die Attribute `identifierType`, `connectedToOtherIdentifier` und `connectedIdentifierType` bestimmt mit welchen Typen von Identifiern und mit wie vielen (einem oder zwei) Identifiern die Metadaten verbunden sind. Das Attribut `metadataType` bestimmt in welchen Metadatentyp die `Vulnerability`-Instanz abgebildet werden soll.

RiskFactor: Dieses Enum stellt die Werte der Gefahreneinschätzung aus OpenVAS für eine Schwachstelle dar.

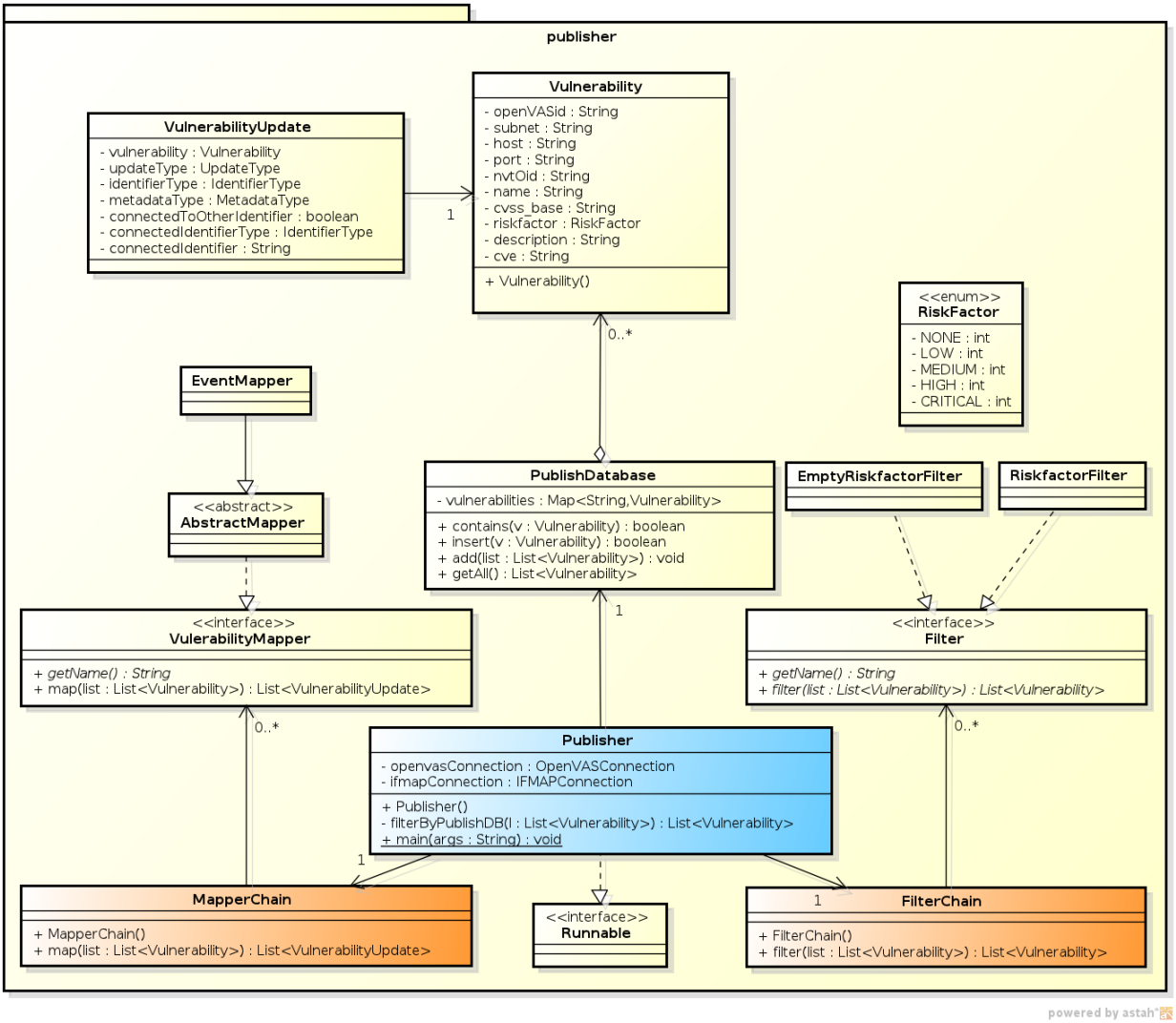


Abbildung 7.2: Publisher

PublishDatabase: Diese Klasse verwaltet eine Menge an `Vulnerability`-Instanzen die bereits im MAP-Server veröffentlicht wurden.

Filter: Diese Interface bietet eine Methode um aus einer Liste von `Vulnerability`-Objekte unerwünschte heraus zu filtern.

FilterChain: Dieses Klasse bietet eine Methode um aus einer Liste von `Vulnerability`-Instanzen, nach bestimmten Regeln, nicht erwünschte Elemente zu entfernen. Für das Entfernen der Elementen werden eine oder mehrere Objekte, die das Interface `Filter` implementieren, verwendet.

EmptyRiskfacorFilter: Diese Klasse implementiert das Interface `Filter` und ist in der Lage `Vulnerability`-Objekte, welche keinen gültigen Wert für das Attribut `Vulnerability.riskfactor` enthalten, heraus zu filtern.

RiskfactorFilter: Auch diese Klasse erfüllt das `Filter`-Interface und filtert `Vulnerability`-Objekte mit einem Wert für `Vulnerability.riskfactor` der niedriger ist als ein Vorgabewert.

VulnerabilityMapper: Diese Interface definiert eine Methode, die eine übergebene Liste von `Vulnerability`-Objekten auf eine Liste von `VulnerabilityUpdate`-Objekten abbildet.

MapperChain: Diese Klasse kann mehrere Objekte vom Typ `VulnerabilityMapper` verwenden um `Vulnerability`-Objekte umzuwandeln.

AbstractMapper: In dieser Klasse werden alle Schritte die für die meisten Implementierungen des `VulnerabilityMapper`-Interfaces gleich sind durchgeführt.

EventMapper: Diese Klasse bildet `Vulnerability`-Objekte so in `VulnerabilityUpdate` Objekte ab, dass sie als `event`-Metadaten weiter verarbeitet werden können.

Publisher: Diese Klasse bildet die eigentliche Logik ab und verwendet verschiedene andere Klassen. Diese Klasse selbst ist eine Unterklasse von `VasmapComponent` und verfügt somit über die Fähigkeit mit einem IF-MAP- und einem OpenVAS-Server zu kommunizieren. Außerdem implementiert diese Klasse das Interface `Runnable` um in beliebigen Threads ausgeführt werden zu können. Wird der Thread in dem eine Instanz dieser Klasse gestartet wird nicht mit der Methode `Thread.interrupt()` zur Beendigung aufgefordert, werden in einem bestimmten Intervall immer die selben Schritte ausgeführt. Diese Schritte sind in Abbildung 7.3 näher erläutert.

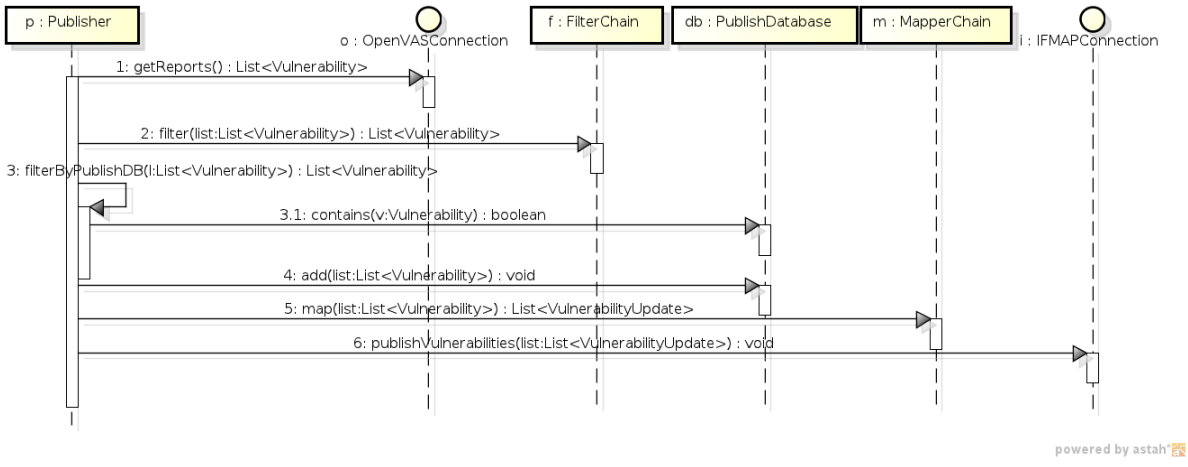


Abbildung 7.3: Publisher-Ausführung

Abbildung 7.3 stellt den oben erwähnten Ablauf dar, welcher vom Publisher in regelmäßigen Abständen wiederholt wird. Nachdem die initiale Verbindung sowohl zum OpenVAS- als auch zum MAP-Server hergestellt wurde, werden zunächst alle Berichte aus OpenVAS abgerufen. Danach werden dann mit Hilfe der `FilterChain` alle `Vulnerability`-Objekte die unerwünscht sind herausgefiltert. Im nächsten Schritt findet ein Abgleich mit der `PublishDatabase` statt. `Vulnerability`-Objekte, die bereits in der `PublishDatabase` vorhanden sind, werden verworfen. Bleiben dann noch `Vulnerability`-Objekte über werden diese zur `PublishDatabase` hinzugefügt und an die `MapperChain` gesendet um abschließend auf dem MAP-Server veröffentlicht zu werden. Sind nach allen Filtervorgängen keine `Vulnerability`-Objekte mehr vorhanden wird ein `renewSession`-Request an den MAP-Server gesendet. Im letzten Schritt wird für einen bestimmten Zeitabschnitt gewartet und dann wieder beim abrufen der Berichte aus OpenVAS begonnen.

Im Design des Publishers wurde der Verarbeitungsschritt „herausfiltern von unerwünschten Bestandteilen“ durch die Klasse `FilterChain` eingeführt, um auf möglichst einfache Art und Weise anpassen zu können, welche `Vulnerability`-Objekte für die weiteren Schritte verwendet werden. Es können also beliebig viele `Filter`-Instanzen zu der Klasse `FilterChain` hinzugefügt werden, die nach und nach das Gesamtergebnis auf die gewünschte Größe reduzieren.

Gleiches gilt für die Einführung der Klasse `MapperChain`. Auch diese Zwischenschicht der Verarbeitung von `Vulnerability`-Objekten wurde eingeführt um nachträglich die Abbildungsweise auf `VulnerabilityUpdate`-Objekte, und damit auch letztendlich die Abbildung auf IF-MAP-Metadaten, möglichst flexibel und erweiterbar zu gestalten. Es könnte also im späteren Verlauf eine Klasse implementiert werden, welche das Interface `VulnerabilityMapper` erfüllt und `Vulnerability`-Objekte so auf `VulnerabilityUpdate`-Objekte abbildet, dass diese von der Kommunikationsschicht in ein spezielles OpenVAS-IF-MAP-Schema transformiert werden können.

7.1.2 Subscriber

In Abbildung 7.4 ist das Klassenmodell des Subscriber-Designs abgebildet. Dieses Modell setzt nur den in Abschnitt 6.3 vorgestellten Anwendungsfall um. Der Subscriber registriert sich also mit Hilfe der Schnittstelle `IFMAPConnection` beim MAP-Server für Metadaten von Typ `request-for-investigation`. Sobald vom MAP-Server neue Informationen zu diesen Metadaten gesendet werden, erstellt der Subscriber in der OpenVAS-Konfiguration ein neues Target und eine neue Task für diese.

Im Folgenden sollen kurz die Komponenten des Subscribers erläutert werden.

IPTarget: Diese Klasse repräsentiert eine IP-Adresse, die vom Subscriber verarbeitet wird. Es sind Attribute für die eigentliche IP-Adresse, sowie Konfigurationsinformationen bezüglich OpenVAS vorhanden. Außerdem gibt es ein Attribut das signalisiert, ob diese IP-Adresse noch im MAP-Server vorhanden ist oder schon aus dem MAP-Server gelöscht wurde.

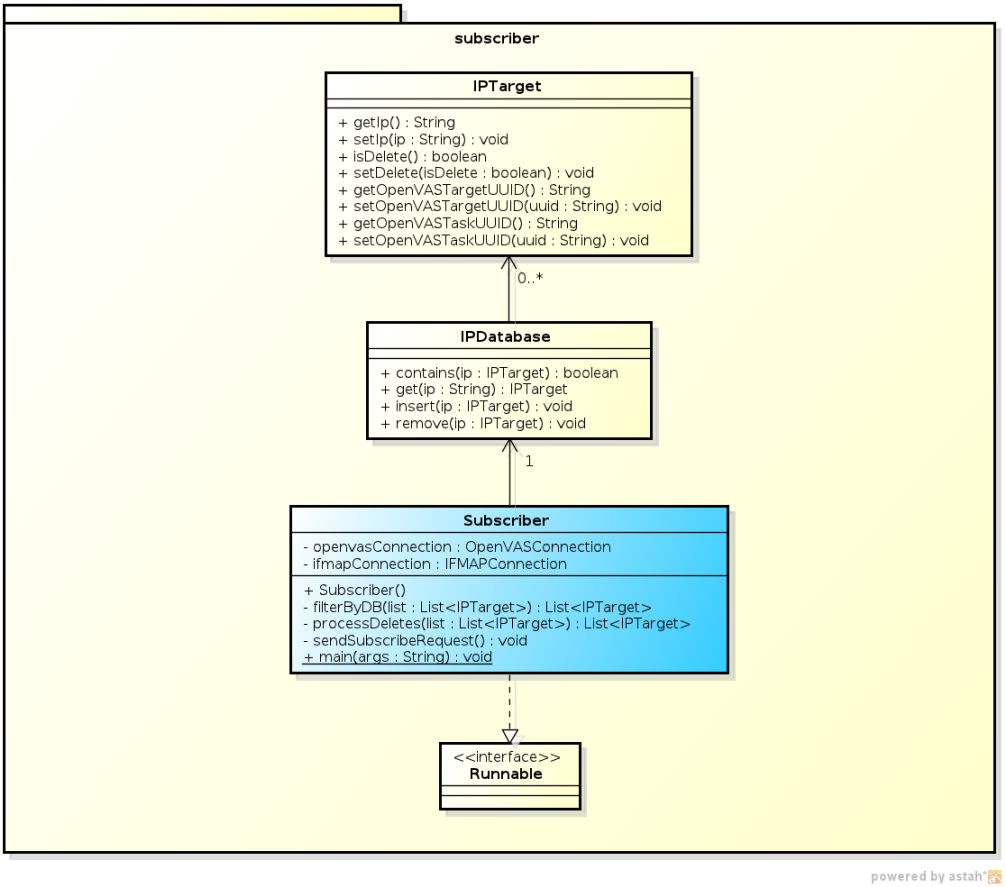


Abbildung 7.4: Subscriber

IPDatabase: Diese Klasse verwaltet alle Objekte vom Typ `IPTarget`.

Subscriber: Diese Klasse beinhaltet die Hauptanwendungslogik des Subscribers. Diese Klasse selbst ist eine Unterklasse von `VasmapComponent` und verfügt somit über die Fähigkeit mit einem MAP- und einem OpenVAS-Server zu kommunizieren. Zusätzlich implementiert diese Klasse das Interface `Runnable` um in beliebigen Threads ausgeführt werden zu können. Diese Klasse registriert sich nach dem Starten einmalig für `request-for-investigation` Metadaten beim MAP-Server. Wird der Thread, in dem diese Klasse läuft, nicht unterbrochen, führt diese Klasse immer wieder einen `poll`-Request aus. Sobald neue Metadaten vom MAP-Server übermittelt werden, erstellt diese Klasse mit Hilfe des Interfaces `OpenVASConnection` ein neues Target und eine neue Task im OpenVAS-Server und startet die Task. Wird durch ein `poll`-Request signalisiert, dass `request-for-investigation` Metadaten gelöscht wurden, werden auch das zugehörige Target und die Task aus OpenVAS entfernt.

7.1.3 Kommunikationsschicht

Die Kommunikationsschicht kapselt die gesamte Funktionalität, die sowohl für den Datenaustausch mit dem OpenVAS-Server, als auch für die Kommunikation mit dem MAP-Server notwendig ist.

Zu dieser Funktionalität gehören folgende Kernbereiche:

- Herstellen der Socketverbindung zum OpenVAS- oder MAP-Server.
- Marshalling und Demarshalling von XML-Nachrichten.

Die gesamte Kommunikationsschicht ist in den Abbildungen 7.5 und 7.6 dargestellt.

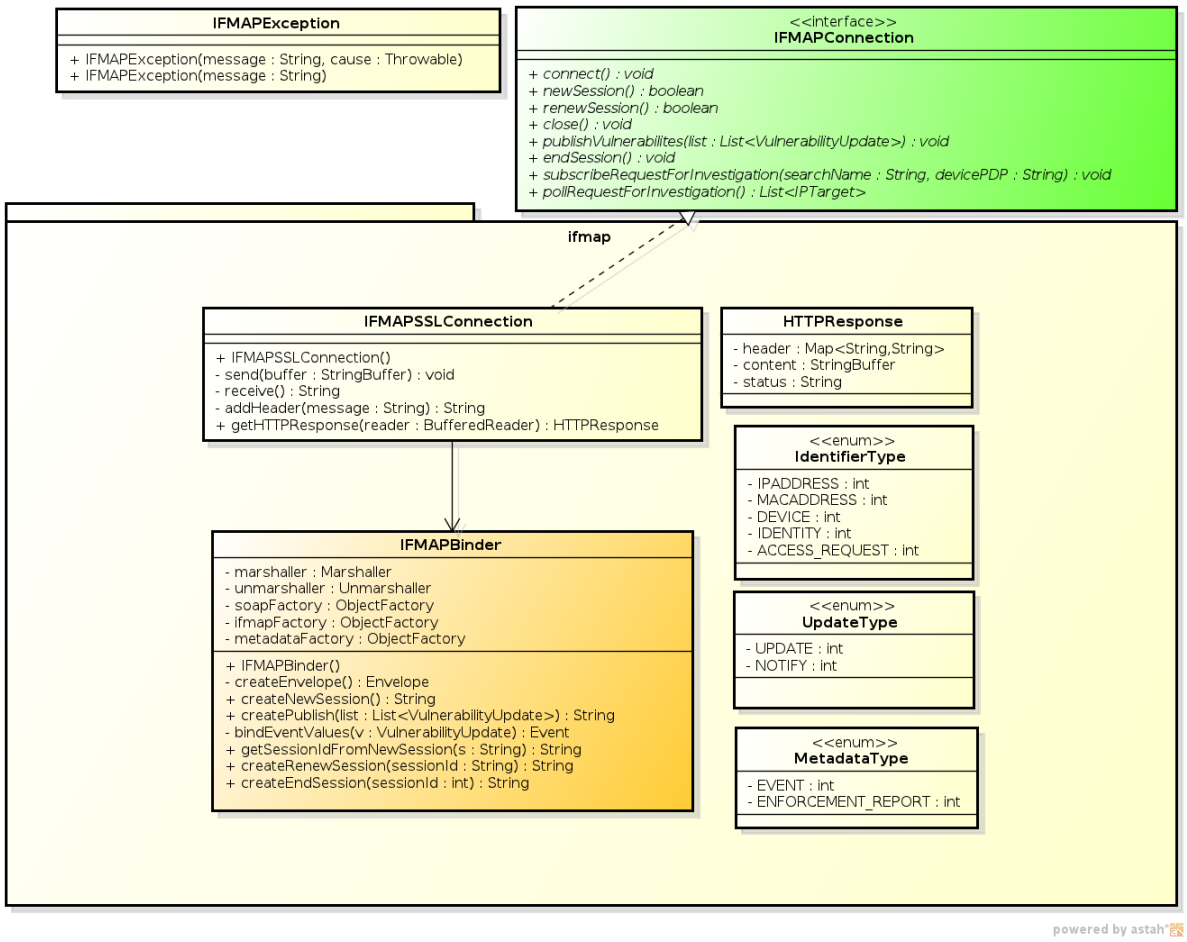


Abbildung 7.5: Kommunikationsschicht (IF-MAP)

Die Dienste die von der Kommunikationsschicht bereit gestellt werden sind durch zwei Java-Interfaces definiert. Die Verbindung und der Datenaustausch mit einem MAP-Server ist durch das Interface `IFMAPConnection` festgelegt. Der Datenaustausch mit einem

OpenVAS-Server erfolgt über ein Objekt welches das Interface `OpenVASConnection` erfüllt. Beim Interface `OpenVASConnection` wurde bewusst jeglicher Bezug zum OpenVAS Management Protocol vermieden, deswegen könnte diese Interface auch für andere Verbindungsarten (ohne den Einsatz von OMP) zu OpenVAS genutzt werden.

Zunächst werden alle Methode des Interfaces `IFMAPConnection` vorgestellt.

connect(): Stellt eine Verbindung mit dem MAP-Server her. Bei diesem Verbindungsaufbau wird nur die grundlegende SSL-Verbindung zum MAP-Server aufgebaut, es werden keine weiteren IF-MAP-Nachrichten gesendet.

newSession(): Sendet einen `newSession`-Request an den MAP-Server. Falls die Anfrage für eine neue IF-MAP-Session Erfolg hat gibt diese Methode `true` zurück, bei gescheitertem Verbindungsaufbau wird `false` zurückgegeben.

renewSession(): Sendet einen `renewSession`-Request an den MAP-Server.

close(): Beendet wie Verbindung mit dem MAP-Server.

publishVulnerabilities(List<VulnerabilityUpdate>): Sendet die übergebene Liste von `VulnerabilityUpdate` an den MAP-Server.

endSession(): Sendet einen `endSession`-Request an den MAP-Server.

subscribeRequestForInvestigation(String, String): Sendet einen `subscribe`-Request an den MAP-Server. Der erste Parameter bestimmt dabei den Namen, unter welchem diese Anforderung im MAP-Server gespeichert wird. Der zweite Parameter enthält den Namen des `device`-Identifiers für den dieser Request bestimmt ist. Die Tiefe dieser Suche im MAP-Server Graphen beträgt 1. Diese Methode ist für den kombinierten Einsatz mit der Methode `pollRequestForInvestigation()` entworfen worden und sollte nur mit dieser eingesetzt werden.

pollRequestForInvestigation(): Sendet einen `poll`-Request an den MAP-Server. Diese Methode blockiert bis eine Antwort vom MAP-Server eingetroffen ist. Der Rückgabewert dieser Methode stellt eine Liste von IP-Adressen dar. Vorbedingung für diese Methode ist, dass mit der Methode `subscribeRequestForInvestigation(String, String)` ein `subscribe`-Request auf `request-for-investigation` Metadaten gesendet wurde.

Zu diesem Interface ist noch zu erwähnen, dass die beiden Methoden `subscribeRequestForInvestigation(String, String)` und `pollRequestForInvestigation()` ausschließlich für den Anwendungsfall Subscriber entworfen wurden und wegen starker Nebeneffekt nur für diesen verwendet werden sollen.

Das zweite wichtige Interface für die Kommunikation mit einem OpenVAS-Server wird durch das Interface `OpenVASConnection` definiert.

connect(): Stellt eine Verbindung mit einem OpenVAS-Server her.

close(): Beendet die Verbindung mit dem OpenVAS-Server.

authenticate(): Sendet Authentifizierungsinformationen an den OpenVAS-Server. Wenn die Authentifizierung erfolgreich war gibt diese Methode `true` zurück, bei einer ungültigen Authentifizierung wird `false` zurück gegeben.

getReports(): Sendet eine Anfrage nach allen verfügbaren Scanberichten an den OpenVAS-Server. Der Rückgabewert dieser Methode ist eine Liste von `Vulnerability`, die alle von OpenVAS-Server gefundenen Schwachstellen darstellt.

createTarget(String, String): Erzeugt ein neues Target in der Konfiguration des OpenVAS-Servers. Der erste Parameter legt den Namen des Targets fest, der zweite Parameter muss die zum Target gehörende IP-Adresse sein.

getConfigUUID(String): Fragt die Scankonfiguration mit dem übergebenem Namen auf dem OpenVAS-Server ab. Der Rückgabewert dieser Methode stellt die UUID der Scankonfiguration dar.

createTask(String, String, String): Mit dieser Methode wird eine neue Task im OpenVAS-Server angelegt. Die Parameter bestimmen dabei den Namen der Task, die UUID mit der zugehörigen Scankonfiguration und die UUID des Ziels dieses Scanauftrages.

startTask(String): Startet den Scanauftrag der zu der übergebenen UUID gehört.

deleteTask(String): Löscht den Scanauftrag der zu der übergebenen UUID gehört aus der OpenVAS-Konfiguration.

deleteTarget(String): Löscht das Target welches zu der übergebenen UUID gehört aus der OpenVAS-Konfiguration.

Die beiden Interfaces stellen somit die Dienste der Kommunikationsschicht dar. Mit ihnen ist die Anwendungsschicht in der Lage sowohl IF-MAP- als auch OMP-Operationen durchzuführen, ohne konkrete Implementierungsdetails dieser Kommunikation kennen zu müssen.

Im Folgenden werden noch weitere Klassen der Kommunikationsschicht und ihre Aufgaben erläutert.

IFMAPException und OpenVASException: Diese beiden Exceptions erweitern `java.lang.Exception` und signalisieren unerwartete Fehler in der Kommunikationsschicht.

IFMAPBinder und OMPBinder: Diese beiden Klassen sind für das Erstellen von XML-Dokumenten zuständig. Sie nehmen Nachrichten in Form von normalen Datentypen entgegen und konstruieren aus diesen XML-Dokumente, passend für das jeweilige Protokoll. Außerdem sind sie in der Lage bestimmte XML-Dokumente zu verarbeiten und aus ihnen Informationen zu extrahieren.

IdentifierType, UpdateType und MetadataType: Sind Enum-Deklarationen die für das Abbilden auf IF-MAP-Typen verwendet werden.

HTTPResponse: Diese Hilfsklasse speichert HTTP-Header, HTTP-Body und den HTTP-Statuscode.

OMPHandler: Diese Klasse erweitert `org.xml.sax.helpers.DefaultHandler` für OMP-Zwecke.

ReportHandler und AuthenticateHandler: Diese Klassen erben von `OMPHandler` und sind für das Verarbeiten von OMP-Nachrichten zuständig.
`ReportHandler` ist z.B. in der Lage aus den OpenVAS-Berichten `Vulnerability`-Objekte zu extrahieren. Neben den in Abbildung 7.6 gezeigten und hier aufgelisteten, gibt es in der Implementierung noch weitere Klassen, die `OMPHandler` erweitern und für die Verarbeitung bestimmter Nachrichten zuständig sind.

EndOfResponseException: Diese Exception wird nur innerhalb der (OMP-)Kommunikationsschicht verwendet und signalisiert das Ende einer OMP-Nachricht. Weitere Details für die Notwendigkeit dieser Exception sind in Abschnitt 7.2.3 zu finden.

IFMAPSSLConnection und OMPSSLConnection: Diese beiden Klassen erfüllen das jeweilige Interface der Kommunikationsschicht. Wichtige Details zur Implementierung werden in Abschnitt 7.2.3 erwähnt.

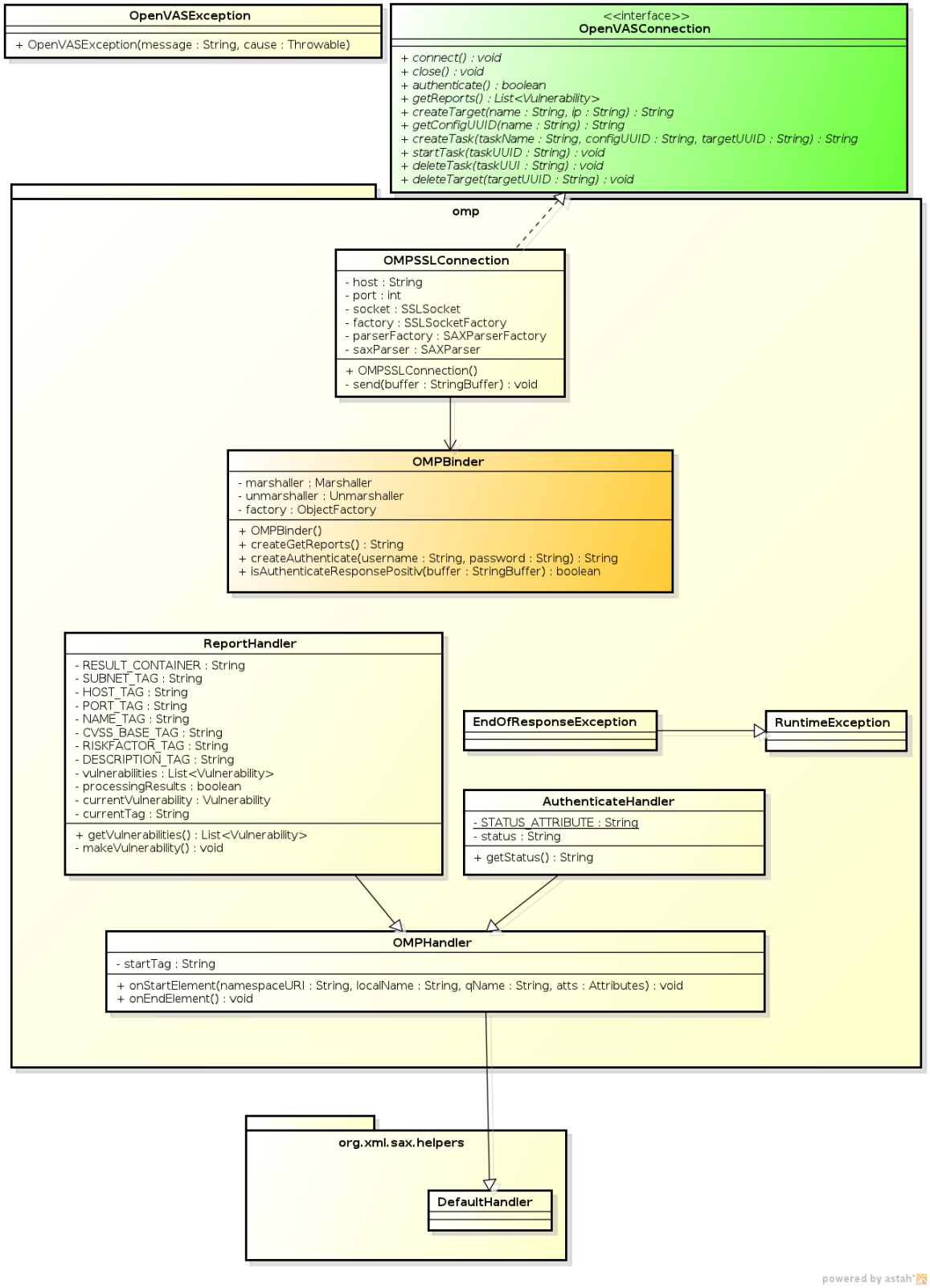


Abbildung 7.6: Kommunikationsschicht (OMP)

7.2 Implementierung

In diesem Abschnitt wird zunächst die Realisierung der beiden Anwendungsfälle erläutert. Danach folgt die Implementierung der Kommunikationsschicht sowie einige grundlegenden Eigenschaften der Umsetzung, die für die Anwendungsfälle nicht relevant sind, sondern die technische Basis und Hilfs- sowie Zusatzfunktionen beschreiben. Zum Abschluss wird der Aufbau der Testumgebung erläutert.

7.2.1 Publisher

Filter

Die Klasse `FilterChain` kann dynamisch Objekte vom Typ `Filter` laden. Dazu wird der Reflection-Mechanismus von Java verwendet. Für das dynamische Laden der Klassen müssen folgende Voraussetzungen erfüllt sein:

- Die Klasse muss das Interface `Filter` implementieren,
- die Klasse muss sich in dem Paket `de.fhhannover.publisher.filter` befinden,
- die Klasse muss der Namenskonvention `<beliebigerName>Filter` genügen und
- in der Datei `configuration.properties` muss der Klassenname ohne Paket und ohne das abschließende `Filter` eingetragen werden.

Mapper

Die Klasse MapperChain ist analog zu der Klasse `FilterChain` in der Lage Instanzen, die das Interface `VulnerabilityMapper` implementieren dynamisch zu laden. Dafür müssen folgende Voraussetzungen erfüllt sein:

- Die Klasse muss das Interface `VulnerabilityMapper` implementieren oder die abstrakte Klasse `AbstractMapper` erweitern,
- die Klasse muss sich in dem Paket `de.fhhannover.publisher.mapper` befinden,
- die Klasse muss der Namenskonvention `<beliebigerName>Mapper` genügen und
- in der Datei `configuration.properties` muss der Klassenname ohne Paket und ohne das abschließende `Mapper` eingetragen werden.

Die aktuelle Implementierung der Klasse `MapperChain` sieht vor, dass jede `VulnerabilityMapper`-Instanz die selbe Liste von `Vulnerability`-Objekten erhält und dass dann die Ergebnisliste von `VulnerabilityUpdate`-Objekten die Vereinigung aller dieser einzelnen Ausführungen

darstellt. Auf diese Weise ist es möglich, dass die Daten zu einer Schwachstelle aus OpenVAS in verschiedene Metadatentypen abgebildet werden können, z.B. in `event`-Metadaten und gleichzeitig in ein spezielles XML Schema.

7.2.2 Subscriber

Da der Subscriber aus den abonnierten IF-MAP-Metadaten ausschließlich Informationen gewinnen kann, die eine Identifizierung des Ziels auf IP-Adressebene ermöglicht, müssen alle anderen Konfigurationsparameter bereits im OpenVAS-Server hinterlegt sein. Durch den Wert des Schlüssels `vasmap.subscriber.openvas.config.id` aus der Konfigurationsdatei `configuration.properties` (siehe auch Abschnitt 7.2.3) wird die Scankonfiguration festgelegt, welche für die neuen Ziele verwendet wird. Außerdem bestimmt der Wert des Schlüssels `vasmap.subscriber.startIdentifier.requestForInvestigation` von welchem `device`-Identifier `request-for-investigation`-Metadaten abonniert werden.

7.2.3 Kommunikationsschicht

Erzeugen von XML-Nachrichten

Für das Erzeugen von OMP- und IF-MAP-Nachrichten wird JAXB (2.1.4) eingesetzt. Da beide Protokolle durch einen Schema definiert sind können aus diesen Vorgaben leicht Java-Klassen generiert werden.

Für diese automatische Generierung wird der im Standard JDK enthaltene Compiler `xjc` verwendet. Durch die Aufrufe

```
$ xjc <ifmap-binding-for-soap>.xsd
$ xjc <ifmap-metadata-for-network-security>.xsd
$ xjc <soap>.xsd
```

können alle Klassen, die für die IF-MAP-Kommunikation nötig sind aus den IF-MAP XML Schema Dateien erzeugt werden.

Da OMP nicht als XML Schema definiert, sondern als RELAX NG Schema, und die RELAX NG Unterstützung von `xjc` noch als experimentell eingestuft wird, muss das OMP Schema erst in ein gleichwertiges XML Schema überführt werden. Für diese Konvertierung kann das Tool Trang[1] verwendet werden. Da im Original OMP Schema allerdings die Definition

```
type_name = xsd:string - xsd:whitespace
```

erscheint, muss dieses vor der Konvertierung angepasst werden. Dies muss geschehen, da in XML Schema kein Datentyp `whitespace` vorgesehen ist und der Konvertierungsversuch mit Trang in der Fehlermeldung

[1] http://www.thaiopensource.com/relaxng/trang.html

```
$ java -jar trang.jar omp.rnc omp.xsd
./omp.rnc:152:26: error: datatype library
"http://www.w3.org/2001/XMLSchema-datatypes"
does not have a datatype "whitespace"
```

abbrechen würde. Nach der Änderung dieser Zeile auf

```
type_name = xsd:string
```

kann mit dem Aufruf

```
$ java -jar trang.jar omp.rnc omp.xsd
```

ein passendes XML Schema für das OpenVAS Management Protocol in der Datei `omp.xsd` erzeugt werden. Die ursprüngliche Definition sagte aus, dass der Typ `type_name` aus einem String (`xsd.string`) besteht, aber keine Leerzeichen (`- xsd:whitespace`) enthalten darf. Das Entfernen von `- xsd:whitespace` hatte während der Entwicklung und den anschließenden Tests keine Auswirkungen, da keine Typname mit Leerzeichen vorhanden waren. Aus dem OMP XML Schema können nun durch den Aufruf

```
$ xjc omp.xsd
```

auch alle für die OMP-Kommunikation benötigten Klassen erzeugt werden.

Als Beispiel für die Verwendung der erzeugten Klassen soll hier kurz die Authentifizierung beim OpenVAS-Server gezeigt werden.

Die Authentisierung findet über das OMP-Kommando `authenticate` statt. Um die notwendige XML-Nachricht zu erstellen besitzt die Klasse `OMPBinder` die Methode `createAuthenticate(String, String)` welche aus dem übergebenen Benutzernamen und Passwort das Authentisierungskommando erstellt und als String zurück gibt.

Ein Auszug der Klasse `OMPBinder` ist in Listing 7.1 abgebildet. Aus Gründen der Übersichtlichkeit wurde die Fehlerbehandlung sowie andere Bestandteile der Klasse entfernt.

Für das Umwandeln der Laufzeitobjekte in XML-Nachrichten wird eine Instanz von `Marshaller` benötigt. Diese wird aus dem `JAXBContext` erzeugt, welche zuvor dafür konfiguriert wurde Objekte der Klasse `AuthenticateCredentials` zu verarbeiten. Die Klasse `AuthenticateCredentials` stellt eine der zuvor mit `xjc` generierten Klassen dar. Die zusätzlichen Eigenschaften, die für den Marshaller konfiguriert werden sind:

- Keine Formatierung der XML-Nachricht mit Zeilenumbrüchen durch `Marshaller.JAXB_FORMATTED_OUTPUT`
- Deaktivieren der XML-Deklaration `<?xml ... ?>` durch `Marshaller.JAXB_FRAGMENT`

Für das eigentliche Erstellen der XML-Nachricht wird zunächst mit Hilfe einer Fabrikklasse, die auch von `xjc` generiert wurde, ein Objekt der Klasse `AuthenticateCredentials` und ein Objekt der Klasse `Credentials` erzeugt. Danach wird der übergebene Benutzername und das Passwort eingetragen und die korrekte Objektbeziehungen hergestellt. Im

```java
import javax.xml.bind.Marshaller;
// ... weitere Imports ...

public class OMPBinder {

  private ObjectFactory factory;
  private Marshaller marshaller;
  // ... weitere Attribute ...

  public OMPBinder() {
    factory = new ObjectFactory();
    JAXBContext context = JAXBContext.newInstance(
      AuthenticateCredentials.class);

    marshaller = context.createMarshaller();
    marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.FALSE);
    marshaller.setProperty(Marshaller.JAXB_FRAGMENT, Boolean.TRUE);
  }

  public String createAuthenticate(String username, String password) {
    StringWriter writer = new StringWriter();
    AuthenticateCredentials auth = factory.createAuthenticateCredentials();
    Credentials c = factory.createCredentials();

    c.getUsernameOrPassword().add(
      new JAXBElement<String>(
        new QName("username"), String.class, username));
    c.getUsernameOrPassword().add(
      new JAXBElement<String>(
        new QName("password"), String.class, password));

    auth.setCredentials(c);
    marshaller.marshal(factory.createAuthenticate(auth), writer);
    return writer.toString();
  }

  // ... weitere Methoden ...
}
```

Listing 7.1: Erstellen einer OMP-Nachricht

Anschluss wandelt der Marshaller die Objektstruktur in die XML-Nachricht aus Listing 7.2 um.

```xml
<authenticate>
   <credentials>
     <username>benutzername</username>
     <password>passwort</password>
   </credentials>
 </authenticate>
```

Listing 7.2: XML Nachricht

Das Erzeugen von IF-MAP-Nachrichten geschieht analog zu diesem Beispiel in der Klasse `IFMAPBinder`.

Verarbeiten von IF-MAP-Nachrichten

Zusätzlich zum Erzeugen von IF-MAP-Nachrichten wird die Klasse `IFMAPBinder` auch für den umgekehrten Verarbeitungsschritt — das Erzeugen von Objekten aus XML-Nachrichten (demarshalling) — verwendet.

Verarbeitung von OMP-Nachrichten

Für die Verarbeitung von OMP-Nachrichten wird ein JAX-Parser eingesetzt. Aufgrund der potentiell enormen Größe von z.B. OpenVAS-Berichten werden diese nicht durch JAXB verarbeitet, sondern durch einen JAX-Parser sequentiell eingelesen. SAX-Parser haben den Vorteil, dass immer nur der aktuell bearbeitete Teil des XML-Dokuments vorhanden sein muss. Wäre das Verarbeiten der OpenVAS-Berichte mit der JAXB-Technologien umgesetzt, würde das gesamte Dokument eingelesen und dann komplett in Form eines Objektbaumes zur Laufzeit vorgehalten, was einen unnötigen Mehraufwand im Vergleich zu einem SAX-Parser bedeuten würde.

Ein weiterer Vorteil bei der Verwendung eines SAX-Parsers ist die volle Kontrolle über den Vorgang der Verarbeitung. So kann dynamisch beim erkennen von bestimmten XML-Tags nur diese Werte ausgelesen werden und andere nicht relevante XML-Tags ignoriert werden.

Der Vorteil der dynamischen Reaktion auf XML-Tags ist bei der Verarbeitung von OMP-Nachrichten extrem wichtig. Da OMP keinerlei Protokoll-Header definiert, ist beim lesen einer OMP-Nachricht unbekannt wie lang diese Nachricht ist. Im Gegensatz dazu enthält z.B. ein HTTP-Header immer die Angabe `Content-Length`, die die Länge der Nutzdaten in Byte darstellt. Da OMP nicht über diese Angabe verfügt muss ein anderer Weg gefunden werden um das Ende einer Nachricht zu erkennen. Dieses Problem ist mit einem SAX-Parser zu lösen. Der Parser speichert das erste XML-Tag welches verarbeitet wird. Erkennt der Parser das schließende Tag der gesamten Nachricht wird durch das Werfen einer `EndOfResponseException` an den Aufrufer signalisiert, dass das Ende der OMP-Nachricht erreicht wurde. Würde auf dem Datenstrom (dem Socket) der die OMP-Nachricht enthält immer wieder die `read`-Methode aufgerufen, würde diese am Ende der Nachricht blockieren und es könnte keine weitere Verarbeitung oder eine Rückgabe der Ergebnisse statt finden. Deshalb ist eine Verarbeitung von OMP-Nachrichten mit JAXB auch nicht möglich. Die JAXB Implementierung würde auf dem Datenstrom der OMP-Nachricht immer wieder eine Leseoperation (z.B. blockierende `read`-Methode) ausführen und dann am Ende der OMP-Nachricht, diese nicht erkennen und weiterhin blockieren ohne ein Ergebnis an den Aufrufer zurück zu geben.

Für die Verarbeitung und das Erkennen des letzten Tags einer OMP-Nachricht ist die Klasse `OMPHandler` zuständig. Diese Klasse selbst stellt eine Erweiterung der Klasse `org.xml.sax.helpers.DefaultHandler` dar und ist in Listing 7.3 abgebildet.

```
import org.xml.sax.helpers.DefaultHandler;
// ... weitere imports ...

public class OMPHandler extends DefaultHandler {

  private String startTag = null;

  protected void onStartElement(String namespaceURI, String localName,
                                String qName, Attributes atts) {
    if (this.startTag == null) {
      this.startTag = localName;
    }
  }

  protected void onEndElement(String namespaceURI, String localName,
                              String qName) throws EndOfResponseException {
    if (this.startTag != null && localName.equals(this.startTag)) {
      throw new EndOfResponseException();
    }
  }
}
```

Listing 7.3: OMPHandler

Die Klasse speichert das erste Tag einer OMP-Nachricht und wirft sobald das passende schließende Tag entdeckt wird eine `EndOfResponseException` an den Aufrufer. Alle Unterklassen von `OMPHandler` rufen vor der Verarbeitung eines Tags immer die Methode `OMPHandler.onStartElement` und vor dem Verarbeiten des schließenden Tags immer `OMPHandler.onEndElement` auf. So wird gewährleistet, dass die Verarbeitung der OMP-Nachrichten niemals am Ende blockiert.

TLS/SSL Socket

Eine wichtige Anforderung an die Kommunikation ist die Wahrung der Vertraulichkeit der Daten. Deshalb werden für die gesamte Kommunikation sowohl zum OpenVAS- als auch zum MAP-Server verschlüsselte Verbindungen verwendet. Die beiden Klassen `OMPSSLConnection` und `IFMAPSSLConnection` besitzen dafür ein Attribut vom Typ `javax.net.ssl.SSLSocket` welches alle normalen Socketoperationen bietet diese aber durch TLS/SSL gesichert werden.

Da für die verschlüsselten Verbindungen X.509 Zertifikate verwendet werden, müssen für beide Anwendungen ein Trust- und ein Key-Store konfiguriert werden. Der Truststore enthält dabei üblicherweise alle Zertifikate anderer Anwendungen mit denen kommuniziert werden soll. Der Keystore enthält den eigenen privaten Schlüssel und den zugehörigen öffentlichen Schlüssel/Zertifikat. Trust- und Key-Store können auch durch eine einzige Datei dargestellt werden, die alle benötigten Zertifikate und Schlüssel enthält. Es werden folgende Zertifikate und Schlüssel benötigt:

- Das Zertifikat des MAP-Servers,

- das Client-Zertifikat, welches den Zugang zum MAP-Server ermöglicht und
- das Zertifikat des OpenVAS-Servers.

Diese Bestandteile werden zusammen in einer Datei abgelegt, die vom Typ Java Key Store (JKS) ist und durch ein Passwort zugriffsgeschützt ist. Innerhalb der Anwendung kann dann, durch das Aufrufen der Methode `System.setProperty` mit den Schlüsseln `javax.net.ssl.keyStore` und `javax.net.ssl.keyStorePassword` (analog dazu `trustStore`), diese JKS-Datei verwendet werden.

Logging

Die gesamte Implementierung benutzt für das Logging die Standard Logging API von Java, die sich im Paket `java.util.logging` befindet. Die Verwendung dieser Standard API hat den Vorteil, dass keine zusätzlichen Abhängigkeiten mit der Software verteilt werden müssen, jedoch ist zu beachten, dass diese Logging API im Vergleich zu z.B. log4j etwas weniger Funktionalität bietet.

Um ein einfaches Anpassen der Logausgaben zu ermöglichen, kann in der Datei `logging.properties` die Anzahl der Logausgaben für bestimmte Pakete oder Klassen verändert werden. In Listing 7.4 ist ein Auszug aus dieser Konfigurationsdatei abgebildet.

```
handlers= java.util.logging.ConsoleHandler
.level= INFO

de.fhhannover.inform.vasmap.publisher.Publisher.level = INFO

java.util.logging.ConsoleHandler.level = ALL
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter
```

Listing 7.4: Auszug Logging Konfigurationsdatei

Konfiguration

Die beiden Anwendungen hängen von einer Vielzahl von Konfigurationsparametern ab. Um diese leicht ändern zu können, wird die Konfigurationsdatei `configuration.properties` beim Starten der Anwendung eingelesen und ist dann über die Klasse `Configuration` für lesenden Zugriff erreichbar. In der Klasse `Configuration` gibt es für jeden Eintrag in der Konfigurationsdatei eine öffentlichen Schlüssel in Form eines `public static final String` über den mit Hilfe der statischen Methode `Configuration.get` der Wert des Schlüssels in der Konfigurationsdatei erfragt werden kann. Durch die Verwendung dieser Schlüssel ist das Ändern von Schlüsselnamen leichter möglich, da die Schlüssel nicht in der gesamten Anwendung verteilt als eigenständige Strings verwendet werden.

7.3 Testumgebung

Der Aufbau der Testumgebung ist in Abbildung 7.7 dargestellt.

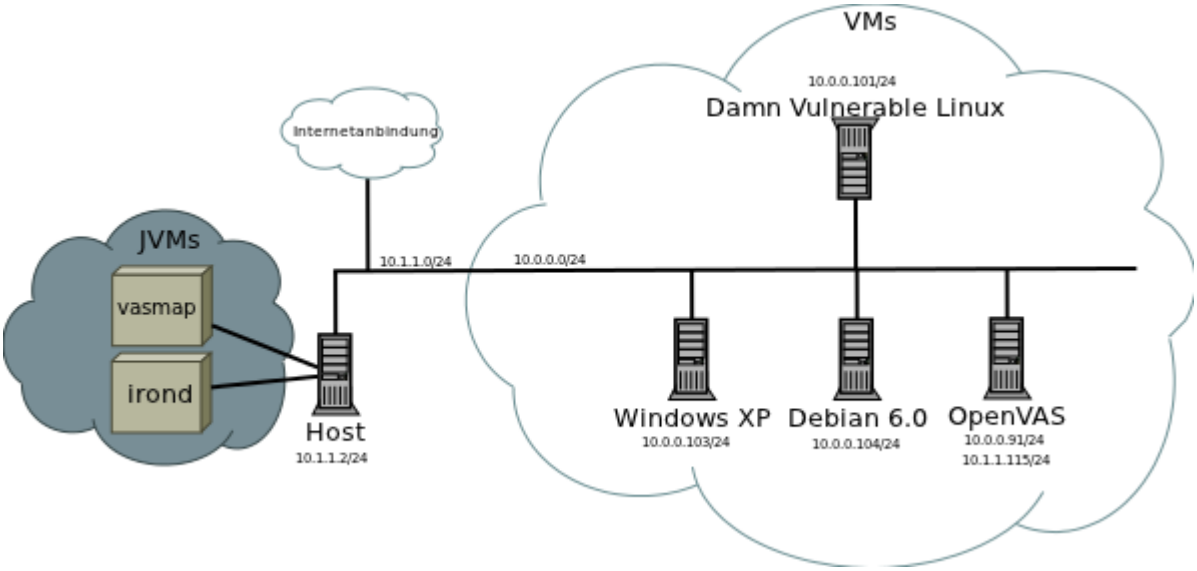


Abbildung 7.7: Testumgebung

Alle Rechner außer „Host“ werden mit Hilfe von Oracle VM VirtualBox[2] simuliert. Als MAP-Server Implementierung wurde irond (Version 0.2.2) verwendet welche direkt auf dem Rechner „Host“ ausgeführt wird. Außerdem wird auch die Implementierung (vasmap) direkt auf dem Rechner „Host“ ausgeführt. Die Rechner „OpenVAS“, „Damn Vulnerable Linux“, „Windows XP“ und „Debian 6.0“ stellen alle eigenständige virtuelle Maschinen dar. Alle virtuellen Maschinen befinden sich in einem eigenen IP-Subnetz ohne Internetverbindung. Von dieser Abtrennung ausgenommen ist der Rechner „OpenVAS“, welcher über eine Netzwerkbrücke zum Rechner „Host“ auf das Internet zugreifen kann und so in der Lage ist NVT-Feeds abzurufen.

Auf den Rechnern werden die folgende Betriebssysteme und Softwareversionen eingesetzt. Alle aufgezählten Dienste waren in der Standardkonfiguration aktiv.

- Host
  - Ubuntu 11.04
  - java version „1.6.0_22“
    OpenJDK Runtime Environment (IcedTea6 1.10.2) (6b22-1.10.2-0ubuntu1 11.04.1)
    OpenJDK Server VM (build 20.0-b11, mixed mode)

[2] http://www.virtualbox.org

  - java version „1.6.0_24“
    Java(TM) SE Runtime Environment (build 1.6.0_24-b07)
    Java HotSpot(TM) Server VM (build 19.1-b02, mixed mode)

- OpenVAS
  - openSUSE 11.4
  - OpenVAS Scanner 3.2.3
  - OpenVAS Manager 2.0.2
  - OpenVAS Administrator 1.1.1
- Damn Vulnerable Linux
  - Damn Vulnerable Linux Version 1.5 (Januar 2009)
  - Apache HTTP Server Version 1.3.37
  - MySQL 5.0.24a
  - OpenSSH Server Version 4.4p1
- Windows XP
  - Windows XP Service Pack 3
  - keine weitere Software
- Debian 6.0
  - Debian 6.0
  - OpenSSH Server Version 5.5p1-6

OpenVAS Installation

Die OpenVAS Installation wurde aus dem openSUSE Build Service (OBS) vollzogen und nicht selbst kompiliert. Dazu muss als erstes OBS in der Paketverwaltung von openSUSE eingerichtet werden. Dies geschieht mit dem Befehl:

```
# zypper ar
"http://download.opensuse.org/repositories/security:/OpenVAS:/\
STABLE:/v4/openSUSE_11.3/security:OpenVAS:STABLE:v4.repo"
```

Danach können mit dem Kommando:

```
# zypper --non-interactive --gpg-auto-import-keys in \
greenbone-security-assistant gsd openvas-cli openvas-scanner \
openvas-manager openvas-administrator
```

alle benötigten OpenVAS-Komponenten installiert werden. Zum Starten von OpenVAS wird das Shell-Skript in Anhang 2 verwendet. In dieser Grundkonfiguration ist der Greenbone Security Assistant auf Port `9392` und der OpenVAS Manager auf Port `9390` erreichbar. Eventuell muss noch in den Dateien `/etc/sysconfig/openvas-manager` und `/etc/sysconfig/greenbone-security-assistant` der Wert für `MANAGER_ADDRESS` und `GSA_ADDRESS` auf `0.0.0.0` geändert werden, um einen Zugriff von außen zu ermöglichen.

Keystores

Damit sich die Komponenten gegenseitig authentifizieren können, müssen einige Zertifikate ausgetauscht werden. Für Publisher und Subscriber gibt es getrennte Keystores in denen der private Schlüssel sowie alle vertrauenswürdigen Zertifikate enthalten sein müssen. Das von OpenVAS benötigte Zertifikat befindet sich in dem Verzeichnis `/var/lib/openvas/CA/` auf dem OpenVAS-Server. Sowohl das OpenVAS-Zertifikat als auch das irond Zertifikat muss in beide Keystores importiert werden. Außerdem müssen die Zertifikate von Publisher und Subscriber in den Keystore von irond importiert werden.

Übersetzen und Ausführen der Implementierung

Um das Übersetzen der Implementierung zu erleichtern wurde Apache Ant[3] eingesetzt. Mit dem Befehl:

```
$ ant jar
```

im Hauptverzeichnis des Quellcodes kann im Verzeichnis `build/dist/bin/vasmap-$version-bin` eine JAR-Datei erstellt werden, die sowohl Publisher als auch Subscriber enthält. Publisher und Subscriber können im JAR-Verzeichnis mit den Befehlen

```
$ java -classpath .:vasmap.jar \
de.fhhannover.inform.vasmap.publisher.Publisher
```

und

```
$ java -classpath .:vasmap.jar \
de.fhhannover.inform.vasmap.subscriber.Subscriber
```

ausgeführt werden.

Testdurchführung

Um die Systeme nicht nur von außen zu scannen, sondern auch Local Security Chechs durchführen zu können, werden zunächst die benötigten SSH-Logindaten wie in Abbildung 7.8 gezeigt in OpenVAS hinterlegt.

[3] `http://ant.apache.org/`

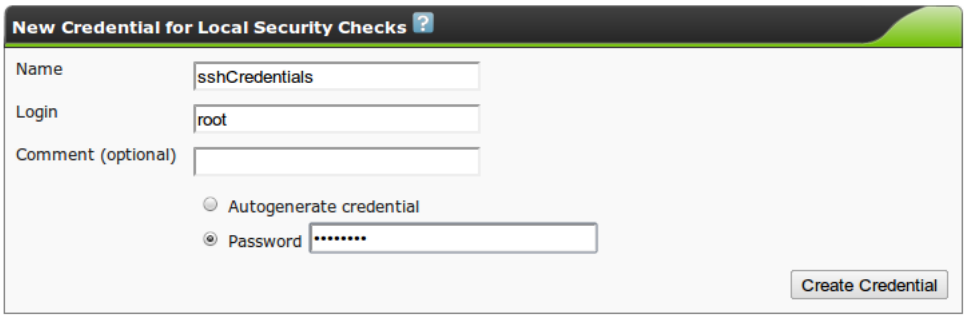


Abbildung 7.8: Einrichten der Zugangsdaten

Danach wird ein neues Target angelegt, welches alle IP-Adressen aus dem Subnetz der virtuellen Maschinen enthält, es werden außerdem die zuvor konfigurierten Zugangsdaten angegeben 7.9.

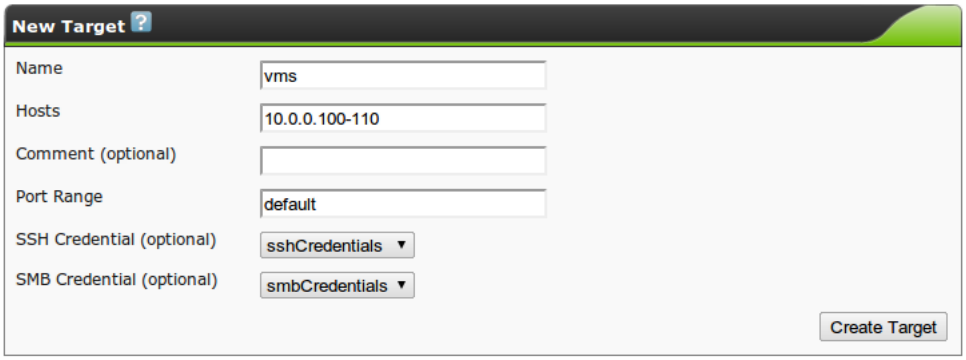


Abbildung 7.9: Einrichten des Targets

Die benötigten Plugins werden wie schon in Abbildung 4.2 gezeigt ausgewählt und in einer Scankonfiguration gespeichert. Als letztes wird für die Kombination aus Target und Konfiguration eine Task angelegt wie in Abbildung 7.10 dargestellt.

Nachdem die Task über das Webinterface gestartet wurde, kann bis zum wirklichen Scanbeginn noch Zeit vergehen, da OpenVAS die Scanaufträge nicht direkt sondern abhängig von internen Parametern (z.B. Systemlast) ausführt.

Nachdem der Scanauftrag beendet wurde, kann im Webinterface ein Bericht 7.11 abgerufen werden.

Publisher

Der Publisher sendet dem OpenVAS-Server in regelmäßigen Abständen Anfragen nach neuen Scanberichten. Sind neue Berichte eingetroffen beginnt der Publisher diese mit

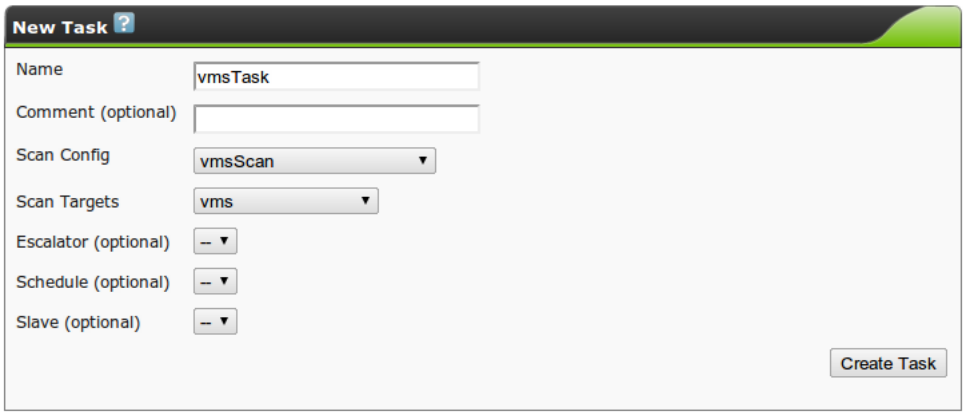


Abbildung 7.10: Einrichten der Task

Hilfe der Filter- und Mapper-Architektur zu bearbeiten und auf dem MAP-Server zu veröffentlichen. Ein Beispiel für die Veröffentlichung von Scanberichten mit Local Security Checks auf dem Host „Damn Vulnerable Linux“ und sehr niedrigen Filtereinstellungen zeigt Abbildung 7.12.

Werden hingegen die Filtereinstellungen restriktiver gesetzt entstehen Graphen wie in Abbildung 7.13 gezeigt.

Subscriber

Nach dem Starten des Subscribers registriert dieser sich sofort beim MAP-Server für `request-for-investigation`-Metadaten. In der Testumgebung simuliert ein Python-Skript (siehe Anhang 3) einen PDP, der diese Metadaten veröffentlicht. Direkt nachdem die Metadaten im MAP-Server veröffentlicht wurden kann in der OpenVAS-Konfiguration ein neues Target, wie in Abbildung 7.14 gezeigt, gefunden werden.

Außerdem wurde eine neue Task angelegt und die Ausführung beauftragt. Werden die Metadaten von dem Python-Skript wieder gelöscht, werden auch alle in OpenVAS veränderten Konfigurationen zurück genommen.

Security Issues reported for 10.0.0.103

Medium netbios-ns (137/udp)
NVT: Using NetBIOS to retrieve information from a Windows host (OID: 1.3.6.1.4.1.25623.1.0.10150)

```
The following 6 NetBIOS names have been gathered :
 WINXPVM         = This is the computer name registered for workstation services by a WINS↵
 client.
 CORE            = Workgroup / Domain name
 WINXPVM         = Computer name
 CORE            = Workgroup / Domain name (part of the Browser elections)
 CORE
   __MSBROWSE__
The remote host has the following MAC address on its adapter :
   08:00:27:e3:27:d2
If you do not want to allow everyone to find the NetBios name
of your computer, you should filter incoming traffic to this port.
Risk factor : Medium
CVE : CAN-1999-0621
```

Back to summary

Port summary for host "10.0.0.104"

Service (Port)	Threat
general/tcp	High

Security Issues reported for 10.0.0.104

High (CVSS: 7.8) general/tcp
NVT: Debian Security Advisory DSA 2184-1 (isc-dhcp) (OID: 1.3.6.1.4.1.25623.1.0.69119)

```
The remote host is missing an update to isc-dhcp
announced via advisory DSA 2184-1.
It was discovered that the ISC DHCPv6 server does not correctly
process requests which come from unexpected source addresses, leading
to an assertion failure and a daemon crash.
Solution:
The oldstable distribution (lenny) is not affected by this problem.
For the stable distribution (squeeze), this problem has been fixed in
version 4.1.1-P1-15+squeeze1.
For the testing distribution (wheezy) and the unstable distribution
```

Abbildung 7.11: Scanbericht

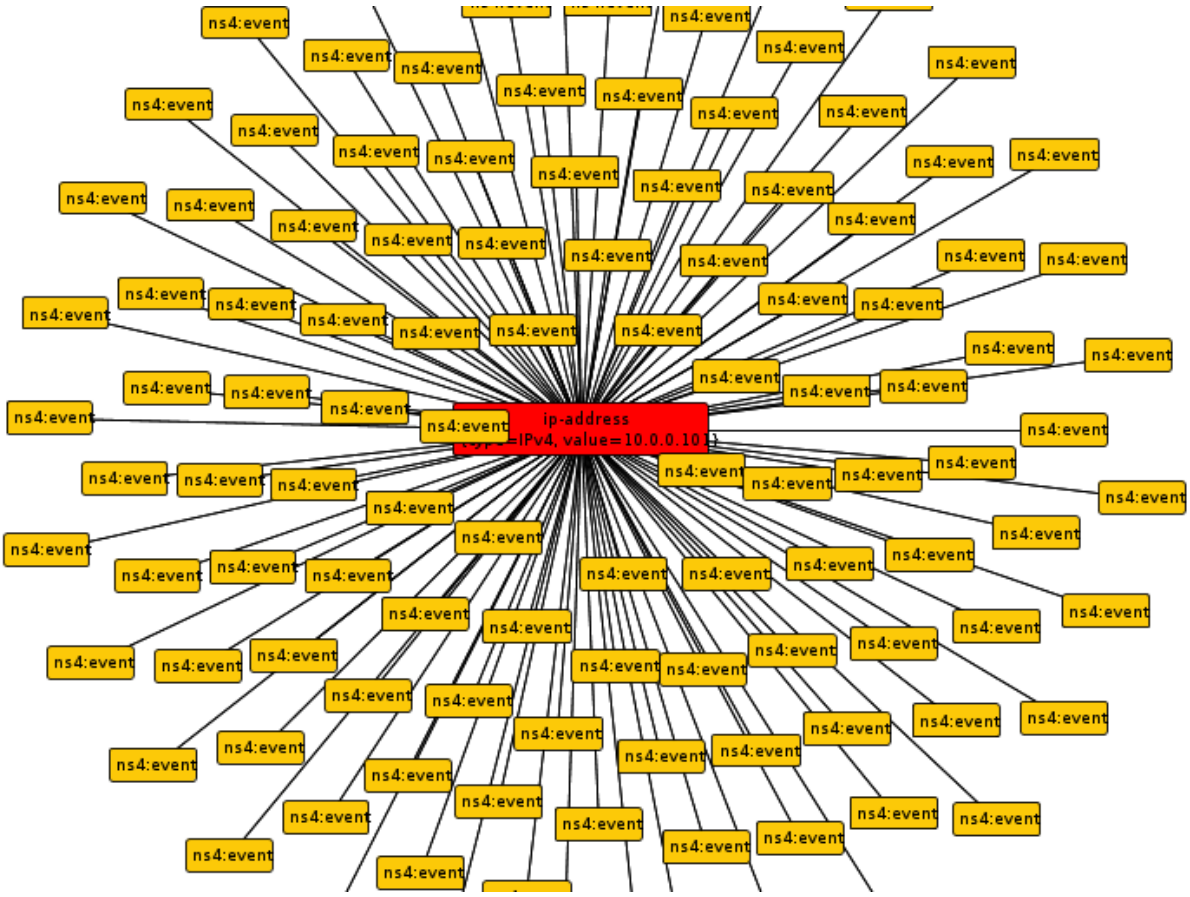
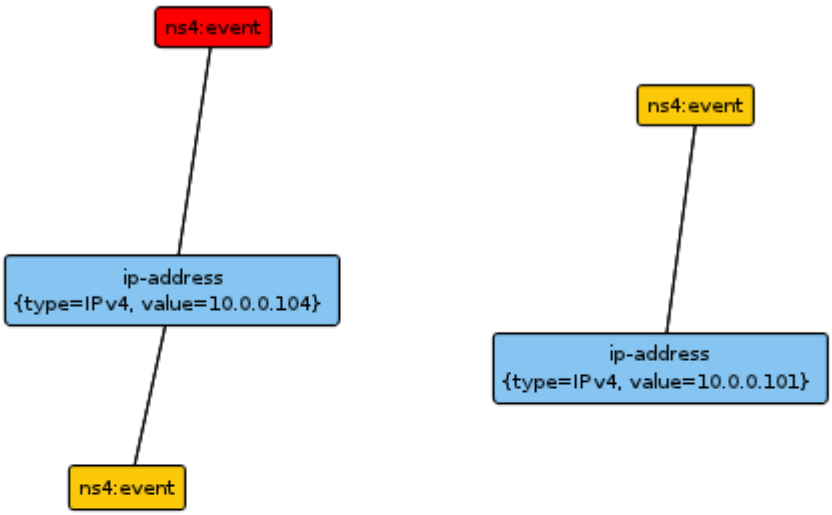


Abbildung 7.12: Niedrige Filtereinstellungen (irongui)



ifmap-publisher-id=vasmapPublisher1313817211, xmlns:ns4=http://www.trustedcomputi

Value
HIGH
0
Debian Security Advisory DSA 2184-1 (isc-dhcp)
CVE-2011-0413

Abbildung 7.13: Hohe Filtereinstellungen (irongui)

Targets

Name	Hosts	IPs	Port Range	SSH Credential	SMB Credential	Actions
10.42.42.1_vasmap	10.42.42.1	1	default			
Localhost	localhost	1				
vms	10.0.0.100-110	11	default	sshCredentials	smbCredentials	

Abbildung 7.14: Vom Subscriber angelegtes Target

8 Reflexion der Anforderungen

In diesem Kapitel soll kritisch untersucht werden, ob durch das Konzept und die Implementierung alle Anforderungen aus Kapitel 5 erfüllt werden.

8.1 Auswahl der Schnittstelle

Die Anforderungen, die sich auf die Schnittstelle zu OpenVAS bezogen, hatten ihren Schwerpunkt in der Auswahl einer möglichst stabilen Schnittstelle mit wenig Änderungen. Diese Anforderung wurde durch die Auswahl von OMP zum Zugriff erfüllt. Es ist sehr wahrscheinlich, dass OMP in Zukunft nur minimale Änderungen erfährt, da es explizit als Schnittstelle für externe Anwendungen definiert ist. Da sich OpenVAS in aktiver Entwicklung befindet sind Änderungen zwar nicht ausgeschlossen allerdings werden diese höchstwahrscheinlich abwärtskompatibel zu vorherigen Protokollversionen sein.

8.2 Schutz der Daten

Eine weitere wichtige Anforderung, die sich auf den Transport der Daten bezog, war die Notwendigkeit die transportierten Daten auf dem gesamten Übertragungsweg von OpenVAS bis zum MAP-Server vertraulich zu behandeln. Dies ist unbedingt notwendig, da es sich um sehr sensible Daten handelt, die von einem Angreifer ausgenutzt werden könnten. Diese Anforderung wurde dadurch erfüllt, dass die gesamte Kommunikation verschlüsselt durch Verbindungen statt findet, die durch das TLS/SSL Protokoll gesichert sind. Diese Verschlüsselung wird für den Transportweg von OpenVAS zum Publisher oder Subscriber verwendet und von dort weiter bis zum MAP-Server. Es gibt also drei Punkte an denen die Daten im Klartext vorliegen:

- Innerhalb des OpenVAS-Servers,
- innerhalb des MAP-Servers und
- innerhalb des Publishers oder Subscribers.

Die Stationen OpenVAS-Server und MAP-Server schützen die Daten durch eigene Zugriffskontrollen. Der Publisher und der Subscriber selbst bieten keine Dienste an, sondern öffnen immer selbstständig eine Verbindung, deshalb ist eine Zugriffskontrolle hier nicht nötig. Natürlich kann ein Angreifer, sobald er Zugriff auf den Rechner hat, auf dem der

Publisher oder Subscriber läuft, den Speicher untersuchen in dem die Daten unverschlüsselt liegen, gleiches gilt allerdings auch für den OpenVAS-Server und den MAP-Server.

8.3 Reaktion auf Verbindungsabbrüche

Da die Kommunikation über Netzwerkverbindungen abgewickelt wird, bestand eine Anforderung darin, dass die Anwendungen angemessen auf Verbindungsabbrüche reagieren. Für die IF-MAP-Verbindung wurde daher vorgesehen, dass im Fall eines Verbindungsabbruches alle IF-MAP-Operationen, die nötig sind um den vorherigen Zustand wieder herzustellen, erneut ausgeführt werden. Das bedeutet, dass der Publisher aus seiner eigenen Datenbank alle Metadaten wieder auf den MAP-Server überträgt und der Subscriber sich erneut für Metadaten registriert.

8.4 Management der Metadaten

Anforderungen, die sich aus der Protokollspezifikation für IF-MAP ergeben haben, bezogen sich überwiegend auf das selbstständige Management von Metadaten durch die Clients. Zum einen war verlangt, dass keine Metadaten doppelt oder in zu großen Mengen veröffentlicht werden, zum anderen bestand die Anforderung nicht mehr aktuelle Metadaten aus dem MAP-Server zu löschen. Die Veröffentlichung von redundanten Metadaten wird im Publisher durch den Einsatz einer Datenbank (7.1.1) gelöst, welche alle bereits veröffentlichten Metadaten enthält. Auch das übermäßige Überschwemmen des MAP-Servers mit Metadaten kann durch beliebige Filter-Kombinationen (7.1.1) verhindert und je nach Aufkommen von Metadaten angepasst werden.

Das Löschen von veralteten Metadaten gestaltet sich komplizierter und ist mit einigen Problemen verbunden, da sich der Publisher nur auf die Informationen aus OpenVAS stützen kann. OpenVAS speichert wie in Abschnitt 4.1.6 beschrieben sämtliche Berichte um z.B. eine Einschätzung der Sicherheitsentwicklung geben zu können. Bei jeder Abfrage werden also alle jemals gefundenen Sicherheitslücken übertragen, also auch solche die eventuell gar nicht mehr bestehen. Über OMP ist es nicht direkt möglich (siehe dazu auch Abschnitt 9.2) nur die aktuellsten Sicherheitslücken abzufragen, außerdem kann OpenVAS auch keine Aussage über die Aktualität der Sicherheitslücken machen, da die Scanberichte im Vergleich zur aktuellen Situation unter Umständen schon veraltet sind. Das Löschen von eventuell veralteten Metadaten mit der `delete`-Operation wurde also in dieser experimentellen Umsetzung nicht realisiert, da der Fokus hier zunächst auf der Untersuchung einer einfachen Anbindung und dem Veröffentlichen lag. Eine mögliche Lösung für die hier erwähnten Probleme der Entscheidung welche Metadaten nicht mehr aktuell sind und gelöscht werden müssen wird in Abschnitt 9.2 aufgezeigt.

Die Implementierung des Subscribers beachtet hingegen das Löschen von Metadaten und nimmt alle zuvor, aufgrund dieser Metadaten, gemachten Konfigurationsschritte zurück.

8.5 Abbildung auf Metadaten

Die Abbildung der OpenVAS-Daten in IF-MAP-Metadaten erfüllt auch alle Anforderungen, da durch die Wahl von `event` alle wichtigen Informationen, ohne Informationsverlust, abgebildet werden können. Ein Nachteil der verwendeten Abbildung ist die nicht einheitliche Terminologie für einige Eigenschaften der Schwachstellen. Eine Lösung, die diesen Nachteil vermeidet ist die Definition einer IF-MAP-Erweiterung welche in Abschnitt 9.2 vorgestellt wird. Die Auswahl des Szenarios für den Subscriber erfüllt alle notwendigen Anforderungen an die Datenabbildung, da eine IP-Adresse zur Konfiguration von OpenVAS für dieses einfache Szenario ausreicht.

8.6 Zusammenfassung

Im folgenden wird noch einmal eine Übersicht über die Umsetzung der Anforderungen gegeben und ob sie erfüllt (✓) wurden:

Funktionale Anforderungen

Veröffentlichen von OpenVAS-Daten

- ✓ Korrekte Abbildung der OpenVAS-Informationen auf IF-MAP-Metadaten.
- ✓ Möglichst vollständige Abbildung der OpenVAS-Informationen auf IF-MAP-Metadaten.
- Korrekte Aktualisierung von veralteten Metadaten.
- Löschen von nicht mehr aktuellen Metadaten.
- ✓ Erneutes Senden von noch aktuellen Metadaten nach einem Verbindungsabbruch.

Reaktion von OpenVAS auf IF-MAP-Daten

- ✓ Möglichst vollständige Konfiguration von OpenVAS.
- ✓ Reaktion auf das Löschen von Metadaten.

Nicht-funktionale Anforderungen

- ✓ Verwendung einer möglichst stabilen Schnittstelle.

- ✓ Gewährleistung der Schutzziele Integrität, Authentizität und Vertraulichkeit für die übertragenen Daten.
- ✓ Robustheit gegen Verbindungsabbrüche.
- ✓ Effiziente Verarbeitung und Weiterleitung von Informationen.

9 Fazit und Ausblick

In diesem Kapitel wird erläutert, ob und in welchem Maß die Anbindung von OpenVAS an eine MAP-Infrastruktur sinnvoll ist. Zum Abschluss wird ein Ausblick in eine mögliche zukünftige Entwicklung der Anbindung gegeben, sowie das Erweiterungspotential der entstandenen Implementierung aufgezeigt.

9.1 Beurteilung

In diesem Abschnitt soll besprochen werden, ob und in welchem Ausprägungen die Anbindung von OpenVAS an eine MAP-Infrastruktur sinnvoll ist. Dabei wird zwischen den beiden Anwendungsfällen Publisher und Subscriber differenziert.

9.1.1 Publisher

Die Veröffentlichung von Sicherheitsrelevanten Daten aus OpenVAS ist grundsätzlich sehr sinnvoll, wenn es um die automatisierte Zusammenarbeit von verschiedenen Netzwerkkomponenten, ohne den Eingriff von Administratoren, geht. Damit gemeint sind z.B. Anwendungsfälle wie das automatische Ändern von Firewallregeln aufgrund von gefundenen Schwachstellen oder die automatisierte Sicherheitsüberprüfung von Endgeräten die ein Netzwerk neu betreten.

Der Vorteil der zentralen Verfügbarkeit aller relevanten Schwachstellen, um im laufenden Betrieb von Sicherheitspersonal ausgewertet werden zu können, wird immer kleiner mit wachsender Anzahl von Metadaten, Schwachstellen und Endgeräten, solange keine Toolunterstützung für das Personal gegeben ist. In den Tests zu dieser Arbeit sind (abhängig von den Filtereinstellungen) Metadaten im dreistelligen Bereich pro Identifier angefallen (7.12). Dies macht deutlich, dass in größeren Netzen die Anzahl von Metadaten für Menschen nicht mehr sinnvoll erfassbar ist und die eigentlichen Zugriffswege zu den Informationen in OpenVAS per Webinterface oder Desktopanwendung weiterhin zur Schwachstellenanalyse durch Menschen unverzichtbar sind.

Jedoch ist die Veröffentlichung der Daten in einem MAP-Server unerlässlich, wenn die Daten von OpenVAS mit den veröffentlichten Daten anderer Komponenten in Zusammenhang gebracht und für z.B. Korrelationsanalyse genutzt werden sollen.

Insgesamt ist das Veröffentlichen von OpenVAS-Daten in einem MAP-Server also ein Sicherheitsgewinn, da alle gewohnten Wege zur Sicherheitsanalyse weiterhin zur Verfügung stehen. Durch die Koordination mit einer Vielzahl an weiteren Metadaten werden neue Möglichkeiten geschaffen Netzwerke dynamisch und ohne Eingriffe sicherheitsrelevante Entscheidungen treffen zu lassen.

9.1.2 Subscriber

Wie im Szenario aufgezeigt ist schon mit den Standard Metadatentypen eine dynamische Konfiguration möglich. Zwar unterliegt diese einige Einschränkungen (z.B. muss der Großteil der Konfiguration schon auf dem OpenVAS-Server vorhanden sein), aber es ist offensichtlich welches Potential diese dynamische Konfigurationsmöglichkeiten bieten. In einer zukünftigen Entwicklung könnten nicht nur die Ziele dynamisch konfiguriert werden, sondern auch die eigentliche Scankonfiguration durch IF-MAP-Metadaten erfolgen (siehe dazu auch 9.2).

Neben den Vorteilen und Möglichkeiten bestehen allerdings auch Gefahren. Sobald ein Angreifer (oder ein fehlerhafter Client) in einem Szenario, in dem die gesamte OpenVAS-Konfiguration über IF-MAP-Metadaten abgewickelt wird, beginnt Metadaten zu veröffentlichen, die Scans für wichtige interne Systeme konfigurieren und Plug-ins aus der Kategorie „aggressive Scans“ (4.1.1) auswählt, entspricht das einem Denial of Service Angriff, für den der Angreifer kaum eigene Ressource einsetzen musste.

9.2 Erweiterungspotential

Publisher

Wie schon geschildert ist der Publisher aktuell nicht in der Lage aus den Informationen aus OpenVAS eine Entscheidung abzuleiten, ob die Metadaten noch aktuell sind. Aktuell werden mit dem OMP-Kommando `get_reports` sämtliche Schwachstellen abgerufen. Für die Lösung dieses Problems gibt es drei Alternativen, die sich recht ähnlich sind und einige Änderungen an der Implementierung erfordern würden.

Die erste Möglichkeit ist mit einem manuellem Eingriff verbunden und vermindert dadurch wieder einen Teil der Automatisierung. Die Vorgehensweise würde folgenden Schritte beinhalten:

1. Ein Administrator löscht manuell die Berichte aus OpenVAS, wenn alle enthaltenen Sicherheitslücken geschlossen sind.
2. Der Publisher registriert beim Abrufen der Berichte, dass bestimmte Schwachstellen nicht mehr in den Berichten auftauchen und löscht diese Schwachstellen aus dem MAP-Server.

Da in OMP kein direkter Weg vorgesehen ist die zuletzt gefundenen Schwachstellen abzurufen, müssen für die zweite Lösungsmöglichkeit verschiedene OMP-Aufrufe kombiniert werden, um diese Informationen zu erlangen. Dieser Ablauf von OMP-Aufrufen und Verarbeitungsschritten ist in Abbildung 9.1 dargestellt.

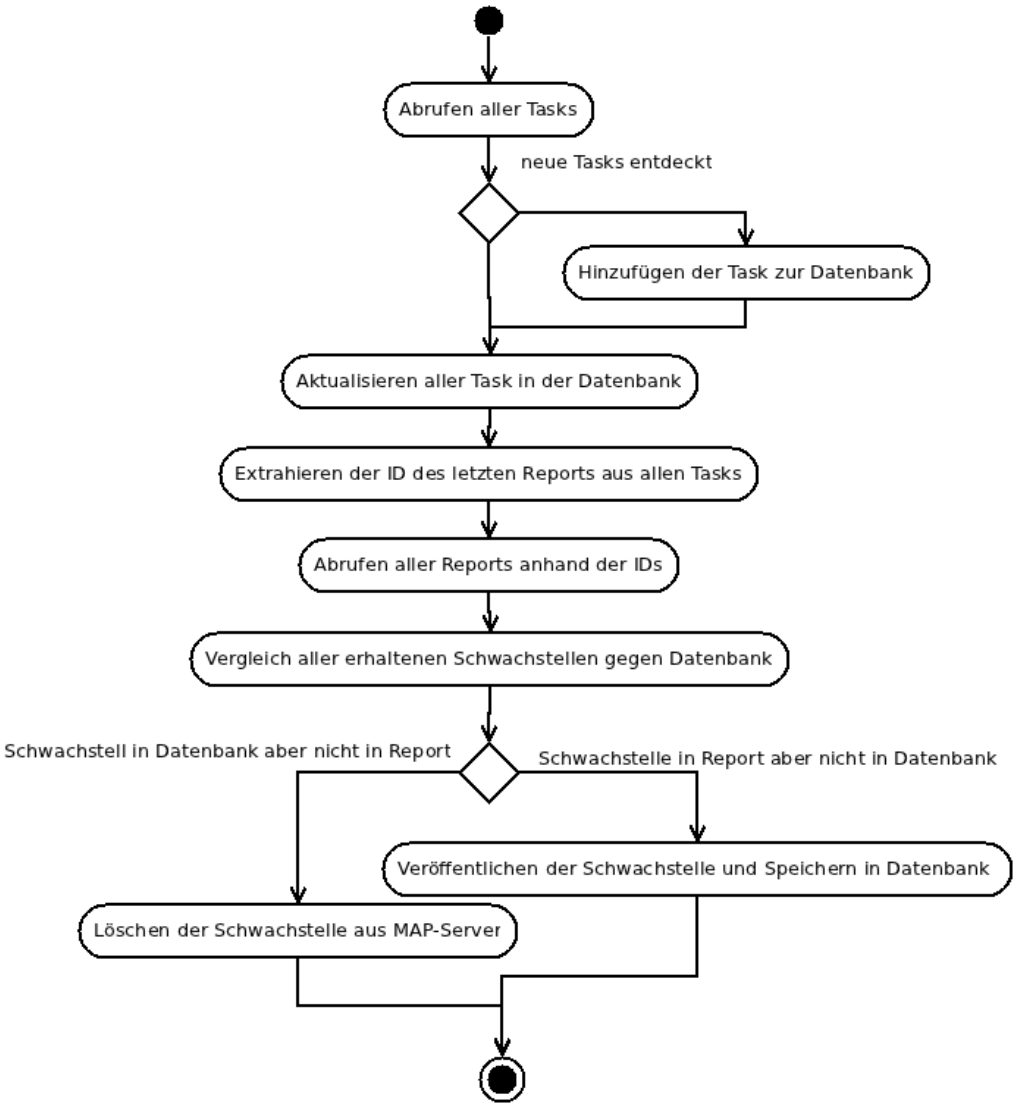


Abbildung 9.1: Löschen von Metadaten

Auch bei dieser Lösung besteht weiterhin das bereits angesprochene Problem, dass die Informationen aus OpenVAS nicht den aktuellen Zustand wiedergeben müssen, selbst wenn diese erst einige Minuten alt sind. Außerdem werden so unter sehr ungünstigen Umständen Metadaten gelöscht, die immer noch aktuell sind. Dies geschieht, wenn ein Host oder eine Schwachstelle nicht im neusten OpenVAS-Bericht auftaucht, weil er in

den Phasen Host-Detection oder Service-Detection (4.4) nicht erreichbar war, die Sicherheitslücke aber immer noch besteht.

Eine dritte Lösung, die aber kaum geeignet ist, wäre das Löschen der Schwachstellen-Metadaten einem anderen Client zu überlassen. Laut Spezifikation kann ein MAP-Server so konfiguriert werden, dass es anderen Clients untersagt ist Metadaten zu löschen, die sie nicht selbst veröffentlicht haben. Außerdem müsste dieser Client auch wieder von einem Administrator bedient werden, der sichergestellt hat, dass die Lücken geschlossen sind.

Subscriber

Auch für den Anwendungsfall des Subscribers gibt es noch weitere Szenarien, die in einer zukünftigen Implementierung umgesetzt werden können. Ein Beispiel für ein solches Szenario wäre das Abonnieren von `ip-mac`-Metadaten die von einem DHCP-Server veröffentlicht werden. Mit diesen Informationen könnten dann Sicherheitsscans in einem dynamischen IP-Netzwerk umgesetzt werden.

XML Schema

Ein weiterer Aspekt der in einer zukünftigen Implementierung umgesetzt werden kann ist die Entwicklung eines eigenständigen XML Schemas für die Metadaten aus OpenVAS. In Abschnitt 6.2.2 wurde gezeigt, dass alle wichtigen Informationen aus OpenVAS in den Metadatentyp `event` übersetzt werden können. Die Entwicklung eines XML Schema für OpenVAS-Metadaten ermöglicht es allerdings immer die selbe Terminologie zu verwenden. Ein minimales Beispiel wie ein solches XML Schema aufgebaut sein könnte, ist in Listing 9.1 gezeigt.

Neben der Verwendung einer einheitlichen Terminologie können in diesem XML Schema auch die in Abschnitt 9.1.2 angesprochenen Konfigurationsparameter für OpenVAS abgebildet werden. Im dargestellten XML Schema ist dies durch das Element `openvasConfiguration` angedeutet. Dieses Element enthält alle für eine vollständige OpenVAS-Konfiguration benötigten Elemente.

9.3 Schlussbemerkung

Die Vorteile einer Anbindung von OpenVAS an eine MAP-Infrastruktur sind offensichtlich. Durch die Verfügbarkeit der Informationen für andere Clients wird eine dynamische Reaktion erst ermöglicht. Gleichzeitig ist es durch OMP relativ einfach möglich, dass OpenVAS auf Ereignisse im Netzwerk reagiert. Dabei dürfen aber die erwähnten Nachteile auch nicht außer Acht gelassen werden.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ifmap="http://www.trustedcomputinggroup.org/2010/IFMAP/2"
  targetNamespace="http://inform.fh-hannover.de/~vasmap"
  xmlns="hhttp://inform.fh-hannover.de/~vasmap"
  elementFormDefault="qualified"
>

  <!-- Publisher -->
  <xsd:element name="vulnerability">
    <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="uuid" />
            <xsd:element name="name" />
            <xsd:element name="subnet" />
            <xsd:element name="host" />
            <xsd:element name="port" />
                <xsd:element name="cve" />
                <xsd:element name="cvss-base" />
                <xsd:element name="risk-factor" />
                <xsd:element name="description" />
          </xsd:sequence>
        </xsd:complexType>
  </xsd:element>

  <xsd:simpleType name="riskFactor">
      <xsd:restriction base="xsd:string">
          <xsd:enumeration value="NONE"/>
          <xsd:enumeration value="LOW"/>
          <xsd:enumeration value="MEDIUM"/>
          <xsd:enumeration value="HIGH"/>
          <xsd:enumeration value="CRITICAL"/>
      </xsd:restriction>
  </xsd:simpleType>

  <!-- Subscriber -->
  <xsd:element name="openvasConfiguration">
    <xsd:sequence>
      <xsd:element name="target" type="targetType" />
      <xsd:element name="config" type="configType" />

      <!-- ... weitere notwendige Elemente ... -->

    </xsd:sequence>
  </xsd:element>

  <xsd:complexType name="targetType">
        <!-- ... alle Target-Informationen -->
  </xsd:complexType>

  <xsd:complexType name="configType">
        <!-- ... alle Config-Informationen -->
  </xsd:complexType>

</xsd:schema>
```

Listing 9.1: OpenVAS XML Schema

Dadurch, dass es sich bei IF-MAP noch um ein sehr junges Protokoll handelt, sind viele Dinge noch nicht erprobt. Dies betrifft z.B. die Absicherung des MAP-Servers sowohl im Bezug auf Angriffe, als auch die Frage nach einem Rechtemanagement für verschiedene IF-MAP-Clients auf der Seite des MAP-Servers. Außerdem sind für viele Konzepte, die durch IF-MAP ermöglicht werden, noch keine Implementierungen vorhanden (z.B. Korrelationsanalyse der Metadaten), so dass erst mit der Verfügbarkeit solcher Tools genauere Abschätzungen gemacht werden können, die über die hier erwähnten Vor- und Nachteile hinaus gehen.

Literaturverzeichnis

[BPM+08] Tim Bray, Jean Paoli, Eve Maler, François Yergeau, and C. M. Sperberg-McQueen. Extensible markup language (XML) 1.0 (fifth edition). W3C recommendation, W3C, November 2008. http://www.w3.org/TR/2008/REC-xml-20081126/.

[Bru11] Ralf Bruns. Software engineering 2. Fachhochschule Hannover, Ricklinger Stadtweg 120, 30459 Hannover, Wintersemester 2010/2011. Vorlesungsskript.

[CKC05] E. Cole, R.L. Krutz, and J.W. Conley. *Network security bible*. Bible Series. Wiley Pub., 2005.

[Cla02] James Clark. Relax ng compact syntax. Technical report, OASIS, November 2002. http://relaxng.org/compact-20021121.html.

[CM01] James Clark and MURATA Makoto. Relax ng specification. Technical report, OASIS, December 2001. http://relaxng.org/spec-20011203.html.

[Der05] Renaud Deraison. *Nessus : [Installation, Konfiguration; Insiderwissen vom Gründer des Nessus-Projekts; auf CD: Snort, Ethereal, Nessus]*. IT-Sicherheit. mitp-Verl., Bonn, 1. aufl., deutsche ausg. edition, 2005.

[DR08] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176.

[Fac10] Fachhochschule Hannover, Fakultät IV - Abteilung Informatik, Ricklinger Stadtweg 118, 30459 Hannover. *IRON Abschlussbericht*, 2010.

[FGM+97] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2068 (Proposed Standard), January 1997. Obsoleted by RFC 2616.

[FHBH+99] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617 (Draft Standard), June 1999.

[IEE04] IEEE. Ieee 802.1x. `http://standards.ieee.org/getieee802/download/802.1X-2004.pdf`, 2004.

[Jur11] Nico Jurran. Angriff auf playstation network: Persönliche daten von millionen kunden gestohlen. http://heise.de/-1233136, 27.04.2011. heise Security.

[Kre02] Akitoshi Kredel, Heinz ; Yoshida. *Thread- und Netzwerk-Programmierung mit Java : Praktikum für die parallele Programmierung.* dpunkt-Verl., Heidelberg, 2., aktualisierte und erw. aufl. edition, 2002.

[McL02] Brett McLaughlin. *Java & XML.* OReilly, Beijing, 2. aufl. edition, 2002.

[McL07] Justin McLaughlin, Brett ; Edelson. *Java & XML : [solutions to real-world problems].* OReilly, Beijing, 3. ed. edition, 2007.

[MCP+06] Eve Maler, John Cowan, Jean Paoli, François Yergeau, Tim Bray, and C. M. Sperberg-McQueen. Extensible markup language (XML) 1.1 (second edition). W3C recommendation, W3C, August 2006. http://www.w3.org/TR/2006/REC-xml11-20060816.

[ML07] Nilo Mitra and Yves Lafon. SOAP version 1.2 part 0: Primer (second edition). Technical report, W3C, April 2007. http://www.w3.org/TR/2007/REC-soap12-part0-20070427/.

[oma] omapd - open source if-map server. https://code.google.com/p/omapd/.

[ope] Openvas - openvas - open vulnerability assessment system community site. http://www.openvas.org/.

[ore11] ore. Anonymous bestraft gema für youtube-streit. http://www.spiegel.de/netzwelt/netzpolitik/0,1518,769347,00.html, 20.06.2011. Spiegel Online.

[PGB+09] David Peterson, Shudi (Sandy) Gao, Paul V. Biron, Ashok Malhotra, Henry S. Thompson, Ashok Malhotra, and C. M. Sperberg-McQueen. W3C xml schema definition language (XSD) 1.1 part 2: Datatypes. Last call WD, W3C, December 2009. http://www.w3.org/TR/2009/WD-xmlschema11-2-20091203/.

[Plö05] Steffen Plötner, Johannes ; Wendzel. *Netzwerk-Sicherheit : Praxisbuch; [Risikoanalyse, Methoden und Umsetzung; für Unix/Linux und Windows; VPN, WLAN, Intrusion Detection, Disaster Recovery, Kryptologie].* Galileo computing. Galileo-Press, Bonn, 1. aufl. edition, 2005.

[Rei10] Holger Reibold. *OpenVAS kompakt.* Professionelles Linux- und Open-Source-Know-How. Brain Media.de (bomots.de), Saarbrücken, 2010.

[Rei11] Ole Reißmann. Jugend hackt. http://www.spiegel.de/netzwelt/web/0,1518,770307,00.html, 28.06.2011. Spiegel Online.

[rrz10] *XML - Version 1.1 : Grundlagen.* RRZN, Hannover, 2010. 8., veränderte Auflage.

[Ruh10] Tobias Ruhe. Visualisierung von Informationen einer zentralen Netzwerk-Datenbank, 2010. Fachhochschule Hannover, Ricklinger Stadtweg 118, 30459 Hannover.

[SMTM+09] C. M. Sperberg-McQueen, Henry S. Thompson, Murray Maloney, Henry S. Thompson, David Beech, Noah Mendelsohn, and Shudi (Sandy) Gao. W3C xml schema definition language (XSD) 1.1 part 1: Structures. Last call WD, W3C, December 2009. http://www.w3.org/TR/2009/WD-xmlschema11-1-20091203/.

[SWB08] Trusted computing systeme : Konzepte und anforderungen, 2008.

[tcg] Trusted computing group. `http://www.trustedcomputinggroup.org/`.

[tcg09] Overview of trusted network connect (tnc) if-map. `http://www.trustedcomputinggroup.org/resources/overview_of_trusted_network_connect_tnc_ifmap`, 2009.

[Tru10a] Trusted Computing Group. *TNC IF-MAP Binding for SOAP*, Juli 2010. Revision 36.

[Tru10b] Trusted Computing Group. *TNC IF-MAP Metadata for Network Security*, September 2010. Revision 25.

[wcc] World wide web consortium (w3c). `http://www.w3.org/`.

Anhang

Skripte

```
#!/bin/sh

echo "fetching openvas nvt ..."
openvas-nvt-sync

echo "checking openvas user 'om' ..."
test -e /var/lib/openvas/users/om || openvas-mkcert-client -n om -i


/etc/init.d/openvas-manager stop
/etc/init.d/openvas-scanner stop

echo "starting openvas-scanner ..."
openvassd

echo "migrating ..."
openvasmd --migrate

echo "rebuilding ..."
openvasmd --rebuild

echo "killing openvas-scanner ..."
killall openvassd

sleep 15

/etc/init.d/openvas-scanner start
/etc/init.d/openvas-manager start
/etc/init.d/openvas-administrator restart
/etc/init.d/greenbone-security-assistant restart

echo "checking openvas user 'admin' ..."
test -e /var/lib/openvas/users/admin || openvasad -c add_user -n admin -r Admin


echo "... done!"
exit 0
```

Listing 2: OpenVAS Startskript

```
import urllib2
import sys
import re
import base64
from urlparse import urlparse
import xml.sax as sax
```

```
theUrl      = 'https://localhost:8443'
username    = 'test'
passwort    = 'test'

sessionID = ""
publisherID = ""

class SessionExtractor(sax.handler.ContentHandler):

        def __init__(self):
                self.sessionID = None
                self.publisherID = None

        def startElement(self, name, attrs):
                if name == "newSessionResult":
                        self.sessionID = attrs["session-id"]
                        self.publisherID = attrs["ifmap-publisher-id"]

def extractSessionInfo(newSessionResponse):
        global sessionID
        global publisherID

        sessionExtractor = SessionExtractor()
        sax.parseString(newSessionResponse, sessionExtractor)
        sessionID = sessionExtractor.sessionID
        publisherID = sessionExtractor.publisherID

def sendSOAPmessageAndShowResponse(message):
        print "sending:"
        print ""
        print message
        print ""

        req = urllib2.Request(theUrl)
        base64string = base64.encodestring('%s:%s' % (username, passwort))[:-1]
        authheader = "Basic %s" % base64string

        req.add_header("Authorization", authheader)
        handle = urllib2.urlopen(req, message)

        response = handle.read()
        print "response:"
        print ""
        print response
        print
        print ""
        return response

def newConnection():
        message='''
     <s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
      xmlns:ifmap="http://www.trustedcomputinggroup.org/2010/IFMAP/2">
          <s:Body>
        <ifmap:newSession />
          </s:Body>
     </s:Envelope>
        '''
        response = sendSOAPmessageAndShowResponse(message)
        extractSessionInfo(response)

def sendDelete():
        message='''
```

```
     <s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
      xmlns:ifmap="http://www.trustedcomputinggroup.org/2010/IFMAP/2"
      xmlns:meta="http://www.trustedcomputinggroup.org/2010/IFMAP-METADATA/2">
          <s:Body>
            <ifmap:publish session-id="%s">
              <delete
                 filter="meta:request-for-investigation[@ifmap-publisher-id='%s']">
                    <device>
                          <name>111:44</name>
                    </device>
                        <ip-address value="10.42.42.1" type="IPv4" />
                  </delete>
            </ifmap:publish>
      </s:Body>
    </s:Envelope>
    ''' % (sessionID, publisherID)
        sendSOAPmessageAndShowResponse(message)

def sendUpdate():
        message='''
     <s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
      xmlns:ifmap="http://www.trustedcomputinggroup.org/2010/IFMAP/2"
      xmlns:meta="http://www.trustedcomputinggroup.org/2010/IFMAP-METADATA/2">
           <s:Body>
                <ifmap:publish session-id="%s">
                  <update>
                   <device>
                        <name>111:44</name>
                   </device>
                   <ip-address value="10.42.42.1" type="IPv4" />
                   <metadata>
                        <meta:request-for-investigation
                           ifmap-cardinality="multiValue" />
                   </metadata>
                  </update>
                </ifmap:publish>
          </s:Body>
    </s:Envelope>
    ''' % (sessionID)
        sendSOAPmessageAndShowResponse(message)

if __name__ == "__main__":
        userinput = ""
        while userinput != "q":
                print "1 = new connection"
                print "2 = send update"
                print "3 = send delete"
                print "q = quit"
                userinput = raw_input("> ")

                if userinput == "1":
                        newConnection()
                        print ""
                        print "sessionID = " + sessionID
                        print "publisherID = " + publisherID
                        print ""
                elif userinput == "2":
                        sendUpdate()
                elif userinput == "3":
                        sendDelete()
                elif userinput == "q":
                        break
```

```
        print "... bye"
```

Listing 3: Simulierter Publisher

CD-ROM

Inhalt der CD-ROM:

- Quellcode der Implementierung
- Diese Bachelorarbeit im Portable Document Format (PDF)
- Python-Hilfs-Skript (3)