Intrusion Detection Management mit „unIDS“

Bericht

Department für Informatik

22.02.06

Dipl.-Informatiker
Rüdiger Busch

Inhaltsverzeichnis

1 Einleitung....4
1.1 Was ist unIDS?....4
1.2 Was ist unIDS nicht?....5
2 Ziele von Intrusion Detection Management....5
2.1 Erweiterbarkeit....5
2.2 Installation....6
2.3 Bedienbarkeit....6
2.4 Plattformunabhängigkeit....6
3 Bestehende IDM Systeme....6
3.1 Frei verfügbare Systeme....6
3.1.1 ACID....6
3.1.2 Scotty....6
3.2 Kommerzielle Systeme....7
3.2.1 ArcSight....7
Architektur....7
3.2.2 neuSECURE....9
Die grafische Schnittstelle....10
3.2.3 nFX OSP....11
Funktionen von nFX OSP....11
3.3 Bewertung und Vergleich....13
4 Der Aufbau von unIDS....15
4.1 Die unIDS Architektur....15
4.2 Die Kommunikation von unIDS....16
4.3 Die Agenten....17
4.3.1 Zielsetzungen und Anforderungen....17
4.3.2 Anwendungsfälle....17
Anmelden in der Mittelschicht....17
Abmelden aus der Mittelschicht....18
Laden und Starten von einzelnen Plugins....18
Änderung der Konfiguration von einzelnen Plugins....18
Senden von gesammelten Informationen....18
4.4 Die Mittelschicht....18
4.4.1 Aufgaben der Mittelschicht....19
Verwaltung der unIDS-Komponenten....19
Nachrichtenaustausch....19
Verwaltung des Systemzustands....19
Verarbeitung und Analyse....20
Anforderungen an die Mittelschicht....20
4.4.2 Entwurfsentscheidungen....20
Die Programmiersprache....20
Verwendete Bibliotheken und Frameworks....21
Nichtfunktionale Bestandteile....21
4.4.3 Architektur der Mittelschicht....22
Architekturmodell der Mittelschicht....22
4.4.4 Schnittstelle zu Agent und Proxy....23
4.4.5 Schnittstelle zur grafischen Benutzeroberfläche....24
Zusammenspiel von GuiInterface und GuiManager....24
GuiInterface....25
4.5 Die GUI....25
4.5.1 Veränderungen des Prototypen....25
Kommunikation....25
Datenhaltung....26

XML Konzept....26
Ergonomische Aspekte....26
Internationalisierung....26
4.5.2 JNLP....26
4.5.3 Grundsätze der Dialoggestaltung (Teil 10)....27
4.5.4 Gebrauchstauglichkeit (Teil 11)....27
4.5.5 Informationsdarstellung (Teil 12)....28
4.5.6 Benutzerführung (Teil 13)....28
4.5.7 Dialogführung (Teil 14-17)....28
Dialogführung mittels Menüs....28
Dialogführung mittels Kommandosprache....29
Dialogführung mittels direkter Manipulation....29
Dialogführung mittels Bildschirmformularen....29
5 Zusammenfassung....29
5.1 Ausblick....30
6 Anhang....31
6.1 Anleitung zur Schnellinstallation....31
6.2 Offene Merkmale....33

1 Einleitung

Intrusion Detection bezeichnet den Prozess des Erkennens von Angriffen bzw. Einbruchsversuchen auf einen Computer bzw. in ein Netzwerk. Umgangssprachlich werden zum Teil auch Gegenmaßnahmen zum Bereich Intrusion Detection gezählt, wobei dies korrekter Weise dem Intrusion Response zugeordnet ist.

Mittlerweile gibt es eine Vielzahl von Werkzeugen, die zum Erkennen von Angriffen eingesetzt werden können. Dies sind z.B. Intrusion Detection Systeme, Virenscanner und bedingt auch Firewalls. Um möglichst alle sicherheitsrelevanten Bereiche abzudecken, kommen meist verschiedene Schutzmaßnahmen zusammen und es bildet sich eine heterogene Sicherheitsstruktur. Dies stellt für die Sicherheit des Netzwerkes zwar einen gewissen Vorteil dar, da ein gleichzeitiger Angriff auf alle Sicherungssysteme unwahrscheinlich ist, führt aber auch zu einer Situation, die zwei Probleme beinhaltet. Zum einen sind die Sicherungsmechanismen häufig nicht untereinander kompatibel und bieten zudem keine einheitliche Schnittstelle für den Benutzer, so dass ein hohes Maß an Kenntnissen im Umgang mit den Systemen erforderlich ist, zum anderen werden durch die Systeme große Mengen an Daten erzeugt, die es ständig zu überwachen gilt.

An dieser Stelle setzt das Intrusion Detection Management an. Mit Hilfe einer Agentenstruktur ermöglicht es eine Anpassung an unterschiedlichste Datenquellen und gestattet dem Benutzer durch eine einheitliche Oberfläche, das gesamte Sicherungssystem im Auge zu behalten. Mit zusätzlichen Filtern kann zudem eine Minimierung der relevanten Daten durchgeführt werden. Im Zuge dessen ist an der Universität Oldenburg das Projekt *unIDS*[1] (unified Intrusion Detection Management System) ins Leben gerufen worden.

1.1 Was ist unIDS?

unIDS ist der Prototyp eines Intrusion Detection Management Systems (IDM), welches wie bereits oben erwähnt, ein System ist, das ein Computernetzwerk auf Sicherheitsverletzungen hin überwachet, indem es sicherheitsrelevante Daten über das Netzwerk und dessen Rechner sammelt und Benutzern zur Verfügung stellt. Diese sollen mithilfe der Daten den Sicherheitszustand des Netzwerks erkennen können, um gegebenenfalls auf Sicherheitsverletzungen zu reagieren.

Um eine möglichst großflächige Datenbasis zu bieten, sind einige bewährte Sicherheitswerkzeuge und Quellen sicherheitsrelevanter Daten als Kern in der Basisversion von unIDS integriert. Dazu gehören 'Snort', ein Network Intrusion Detection System (NIDS), das Sicherheitsverstöße innerhalb des Netzwerkverkehrs im eingesetzten System feststellt, meldet und speichert, und ein *Logchecker,* der das Systemlog auf sicherheitsrelevante Einträge hin überprüft.

Um die Vorgänge auf einzelnen Rechnern zu überwachen, werden deren Systemstatistiken wie die Stärke des Netzverkehrs über ihre Interfaces, die CPU-Auslastung oder die Anzahl eingeloggter Benutzer an das unIDS-System übertragen.

Um die Datenbasis erweitern zu können und auf das konkret zu überwachende Netzwerk abstimmen zu können, ist ein wichtiger Gesichtspunkt bei der Entwicklung von unIDS eine einfache Erweiterbarkeit für neue Sicherheitswerkzeuge.

Ein wesentlicher Aspekt eines IDM ist zudem seine Benutzbarkeit. Ein IDM sollte daher intuitiv nutzbar sein, indem es die Vorgänge im überwachten Netzwerk graphisch anschaulich darstellt. Weiterhin sollte es eine leicht zu benutzende graphische Oberfläche bieten. Für unIDS wird daher eine graphische Oberfläche entwickelt, die diese Ansprüche erfüllen soll.

1 sprich: „junids“

1.2 Was ist unIDS nicht?

unIDS ist kein Intrusion Detection System, sucht also nicht selber nach Sicherheitsverletzungen. Daher hängt die Stärke von unIDS vor allem von den eingebundenen Sicherheitswerkzeugen ab.

Desweiteren ist unIDS kein Intrusion Response System, das heißt, es bietet keine integrierten Reaktionsmöglichkeiten auf eine etwaige Sicherheitsverletzung an. Die Entscheidung, welche Schritte unternommen werden sollen und deren Ausführung, liegt beim Benutzer.[PG0304]

2 Ziele von Intrusion Detection Management

Allgemein können die Ziele eines IDM wie folgt definiert werden:

- Es genügt softwareergonomischen Aspekten und ist auf menschliche Analysefähigkeiten angepasst.
- Ein IDM schließt die Lücken zwischen den unterschiedlichen Sicherheitswerkzeugen und den Nutzern.
- Es unterstützt technisch die Organisation in großen Umgebungen und ermöglicht so eine schnelle und effektive Reaktion auf einen Einbruch oder einen Versuch.
- Es stellt eine einheitliche Schnittstelle für Sicherheitswerkzeuge dar.
- Es ermöglicht eine sichere Verwaltung aller beteiligten Komponenten.
- Es bietet dem Benutzer eine grafische Schnittstelle, welche sicherheitsrelevante Vorgänge darstellt und effektives Eingreifen ermöglicht.

Daran angelehnt sind die Ziele von unIDS. Hier beginnt der softwareergonomische Ansatz schon bei der Installation. Eine Verfeinerung der Zielsetzung von unIDS wird in diesem Kapitel beschrieben.

2.1 Erweiterbarkeit

Ein IDM muss sich leicht in eine bestehende Sicherheitsstruktur einfügen lassen. Eine Voraussetzung dafür ist eine flexible Erweiterbarkeit des Systems, um mit allen notwendigen Komponenten kommunizieren zu können und später zu integrierende Komponenten einbinden zu können. Die Einbindung der Komponenten wird von unIDS durch Agenten realisiert. Ein unIDS-Agent ist ein Programm auf einem Rechner, dass Daten von Sicherheitswerkzeugen, die auf dem Rechner laufen, einsammelt und an unIDS weiterleitet. Um die Daten eines Sicherheitswerkzeugs zu sammeln, besitzt ein Agent ein sogenanntes Plugin für dieses Werkzeug, eine Erweiterung des Agenten, die Kenntnis darüber besitzt, wo sich die sicherheitsrelevanten Daten befinden und wie diese zu interpretieren sind.

Die zentral gesammelten Daten werden durch den unIDS Analyser aufbereitet und dann der graphischen Schnittstelle zur Verfügung gestellt. Um komponentenabhängige Daten korrekt darstellen zu können, ist für die GUI ebenfalls eine Erweiterungsmöglichkeit nach dem Plugin Prinzip vorgesehen.

Um ein neues Sicherheitswerkzeug einzubinden, muss ein Plugin für den Agenten entwickelt werden, der die Daten des Werkzeugs sammelt und an unIDS sendet, sowie ein Plugin für den Analyser, der die gelieferten Daten verarbeiten kann. Optional kann eine eigene Darstellungsform dieser Daten für die GUI entwickelt werden. Die GUI bietet aber eine Standarddarstellung für Daten ohne eigenes Plugin an. Diese Standarddarstellung gibt die Daten in tabellarischer Form aus.

2.2 Installation

Die Installation von unIDS soll auch von einem wenig erfahrenen Administrator durchgeführt werden können. Dafür stellt das System Installationskripte zur Verfügung, die alle notwendigen Konfigurationen übernehmen. Lediglich die Sicherheitsstruktur von unIDS selber muss der Administrator dem System mitteilen.

2.3 Bedienbarkeit

Auch ein unerfahrener Benutzer soll sich schnell in unIDS einarbeiten können. Daher wird für unIDS eine Graphische Benutzeroberfläche (GUI) entwickelt, die übersichtlich strukturiert und intuitiv verständlich sein soll. Über diese GUI soll der Benutzer sämtliche Funktionen des unIDS-Systems ausführen können, so dass er sich mit der dahinterliegenden Struktur nicht befassen muss.

Für einige Bereiche wird allerdings ein Benutzer benötigt, der tiefere Kenntnisse über das Netzwerk besitzt. Dazu gehört das Erstellen des Netzwerks für die GUI des unIDS-Systems: unIDS besitzt nur Informationen über diejenigen Rechner im Netz, auf denen Komponenten von unIDS installiert sind und die somit Daten an die Zentrale schicken.

2.4 Plattformunabhängigkeit

Die Plattformunabhängigkeit seitens des Systems wird durch die verwendeten Sprachen garantiert. Für die grafische Schnittstelle wird auf JAVA zurück gegriffen, der gesamte Unterbau wurde mit der Interpretersprache Python realisiert, welches ebenso wie JAVA für diverse Betriebssysteme kostenlos zur Verfügung steht. Zudem können die Agenten prinzipiell in jeder beliebigen Sprache auf einem beliebigen Betriebssystem installiert werden, sofern sie die standardisierte Kommunikation von unIDS unterstützen.

3 Bestehende IDM Systeme

Bevor weiter auf die Details von unIDS eingegangen wird, soll zunächst der Stand der Technik anhand auf dem Markt befindlicher Systeme dargestellt werden. Dazu werden diese im folgenden kurz beschrieben und anschließend bezüglich der im vorigen Kapitel aufgeführten Kriterien bewertet. Eine ausführlichere Darstellung findet sich in der Diplomarbeit [JS05].

3.1 Frei verfügbare Systeme

Aus dem Bereich OpenSource stehen bisher nur wenige Systeme zur Verfügung, die sich dem IDM zuordnen lassen. Zwei davon sind *ACID* (Analysis Console for Intrusion Databases) und *Scotty*.

3.1.1 ACID

ACID[2] ist ein Web-Frontend, welches in PHP geschrieben wurde und der Datenaufbereitung von *Snort* und *Logsnorter* dient. Es erlaubt sicherheitsrelevante Daten und Alarmmeldungen tabellarische und z.T. grafisch darzustellen und nach verschiedenen Kriterien zu sortieren. Weiterhin kann nach bestimmten Eigenschaften, z.B. Signatur oder Alarmzeit gesucht werden. Auf aufwendige grafische Darstellungen zur Datenanalyse wurde verzichtet.

3.1.2 Scotty

Das Programm[3] wird seit 1993 an der TU Braunschweig von J. Schönwälder und H. Langendörfer für den Bereich Netzwerkmanagement entwickelt und stellt eine Erweiterung der Sprache *Tcl* (Tool Command Language) um netzwerkmanagementspezifische Kommandos dar.

Um die Ergebnisse grafisch anzeigen zu können, existiert zusätzlich eine einfache Benutzerschnittstelle namens Tkined, mit der Netzwerkansichten erstellt und gepflegt werden können (siehe Abbildung 1). Vorraussetzungen für die Nutzung von Scotty/Tkined ist die

2 http://www.andrew.cmu.edu/user/rdanyliw/snort/snortacid.html
3 http://www.ibr.cs.tu-bs.de/projects/scotty/

Installation der Tcl- und Tk-Pakete.

Der Schwerpunkt liegt wie oben erwähnt im Netzwerkmanagement, aufgrund verschiedener Erweiterungen, z.B. IP Troubleshooter, Event Filter, SNMP Troubleshooter, lassen sich aber auch sicherheitsrelevante Ereignisse verarbeiten und darstellen.

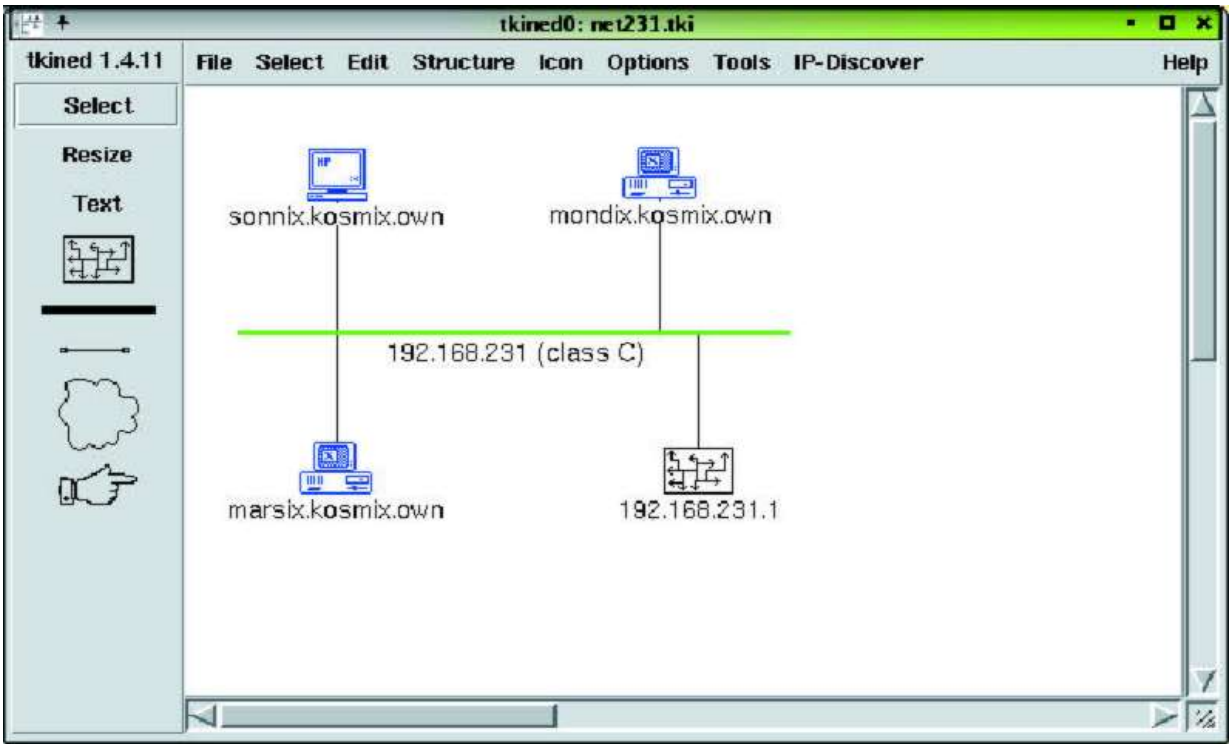

Abbildung 1 Scotty/Tkined

3.2 Kommerzielle Systeme

Im kommerziellen Bereich werden mittlerweile Lösungen angeboten, die sich selbst für die Überwachung großer Netze eignen. Exemplarisch sollen an dieser Stelle *ArcSight* des gleichnamigen Herstellers, *nFX OSP* der Firma netForensics sowie *neuSecure*, welches mittlerweile von Micromuse Inc. vertrieben wird, betrachtet werden.

3.2.1 ArcSight

ArcSight umfasst wichtige Funktionen für das Sicherheitsmanagement wie Aggregation und Archivierung von Ereignissen, Echtzeitanalysen, Untersuchung von Zwischenfällen, Reaktion auf Angriffe und Berichterstellung. Durch diese Techniken ist es möglich eine große Anzahl von Logdaten unterschiedlichster Geräte zu überwachen und zu verarbeiten.

Architektur

Die ArcSight Architektur, wie in Abbildung 2 zu sehen, beinhaltet ein System zur Sammlung und Speicherung von Daten, um die netzwerkseitigen Alarmmeldungen und Warnungen zusammen zu führen. Diese können in einer Konsole dargestellt und überwacht werden. Die Architektur von ArcSight beinhaltet zusätzlich zu der Konsole *SmartAgents* und *Manager*, die im folgenden näher beschrieben werden.

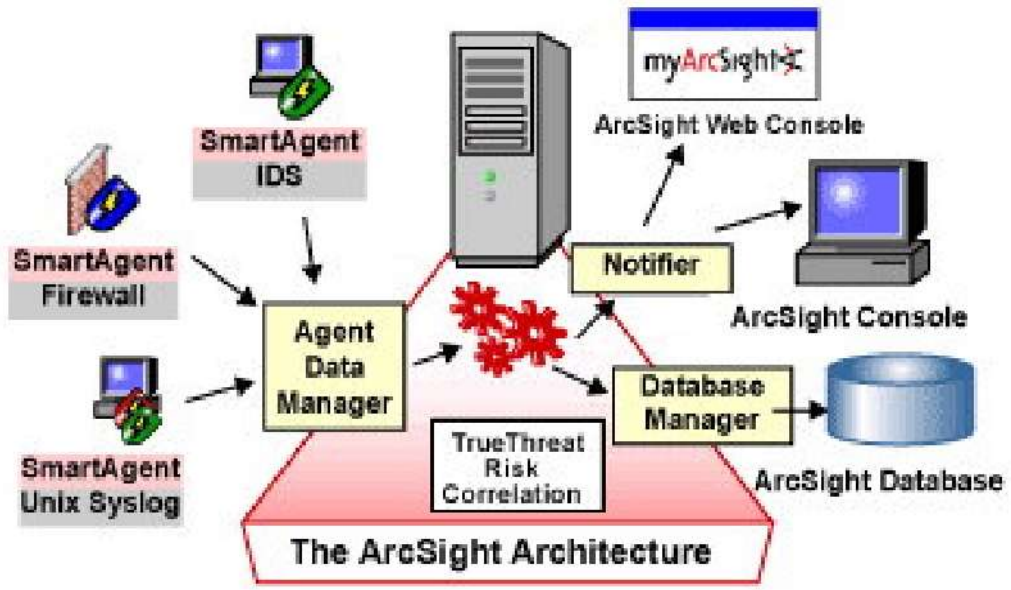

Abbildung 2 ArcSight Architektur

- Smart Agents
  Dies sind kleine Programme, die Daten (u.a. von Firewalls und Intrusion Detection Systemen) sammeln und an die Datenbank weiterleiten. Zur Reduzierung des Datenverkehrs, kann der Administrator bereits hier Bedingungen festlegen, inwieweit unwichtige Daten herausgefiltert werden sollen. Außerdem werden vom Agenten doppelte Alarmmeldungen aufgespürt.

- Manager
  Manager sind serverseitige Komponenten, die Daten weiter zusammenführen, filtern und die Ereignisse mittels einer *Rule-Engine* und in einer zentralen Datenbank korrelieren. Eine Hauptaufgabe der ArcSight Manager, schematisch in Abbildung 3 dargestellt, ist die Sammlung und Speicherung von historischen und aktuellen Ereignissen, um ein komplettes Bild über die sicherheitsrelevanten Aktivitäten zu erstellen. Die Manager liefern zudem eine zentrale Administration (Kontrollpunkt), eine Wissensdatenbank und ein sogenannten *case management workflow*, in dem Informationen zu einem Vorfall gesammelt werden können.

- Konsole
  Eine GUI (siehe Abbildung 4) mit der sich Ereignisse anzeigen und untersuchen lassen sowie Regeln und Berichte erstellt werden können. Diese Anwendung steht am zentralen System zur Verfügung, mit *myArcSight* lässt sich die Anzeige aber auch per Webbrowser darstellen. Die beiden wesentlichen Funktionen der GUI sind dabei:

  - Link Analysis Maps
    Identifiziert Beziehungen zwischen scheinbar nicht zusammengehörigen Alarmmeldungen, durch die geschickte grafische Aufzeichnung von Daten durch Icons und Linien. Das Ergebnis ist ein Schema des Angriffs, der die Quelle und das Ziel aufzeigt. Durch diese Echtzeit-Momentaufnahme von Aktivitäten können Administratoren erkennen, wenn ein Ziel von mehreren Quellen angegriffen wird oder eine Quelle versucht einen verteilten Angriff zu starten.

  - Geo-Spatial Maps
    Erlaubt den Analysten die Ereignisse nach Land und Städten aufzuspüren, markiert verdächtigen Verkehr von bestimmten Ländern und zeichnet verdächtige Quellen auf.

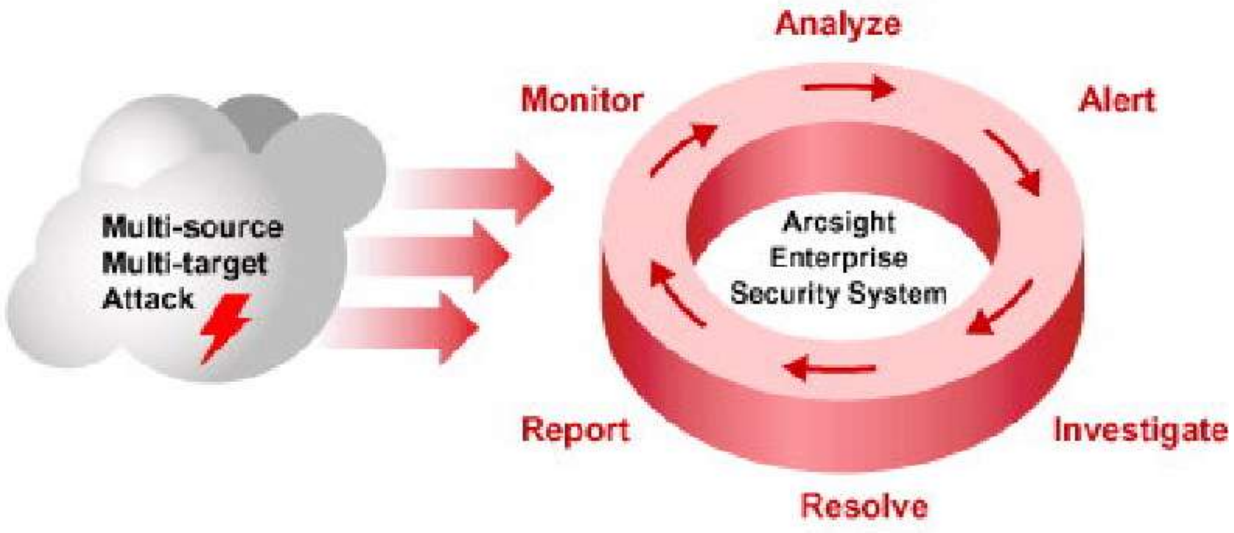

Abbildung 3 ArcSight Manager

Abbildung 4 ArcSight Konsole

3.2.2 neuSECURE

neuSECURE steuert Überwachung, Korrelation und Durchführung von Sicherheitsanalysen. Die Analyse und Korrelation von Logdateien in Echtzeit ermöglicht es Administratoren, bei Sicherheitsverletzungen unverzüglich eingreifen zu können. Die Entwicklung von neuSECURE wurde durch die steigende Anzahl von Beschädigungen durch Würmer voran getrieben. Beim Versuch das Problem mit den Würmern in den Griff zu bekommen, mussten sich die Entwickler

mit Informationen auseinandersetzen, die von vielen Firewalls und IDS aufgefangen wurden. neuSECURE unterstützt daher eine zentrale Sammlung von Logdateien diverser Sicherheitsumgebungen. Weitergehende Funktionen sind z.B.

- Aggregation
  neuSECURE fasst rohe Logdaten der Geräte zusammen, um Administratoren bei der Erkennung von Angriffen, bei kritischen Fehlkonfigurationen und internem Missbrauch zu unterstützen.
- Korrelation
  Es können Angriffe aufgedeckt und priorisiert werden, damit sie zur weiteren Betrachtung zur Verfügung stehen. Die *Deterministic Threat Analysis* besteht aus verschiedenen Korrelationstechniken. Der Vorteil liegt darin, dass verschiedene Methoden unterschiedliche Ereignisse aufspüren können. Die statistische Korrelation ist gut geeignet für unbekannte Angriffe und anormales Verhalten. Die regelbasierte Korrelation lässt sich gut für die Aufdeckung von internem Missbrauch und zur Durchführung von Sicherheitsrichtlinien einsetzen.
- Untersuchung und Reaktion
  neuSECURE integriert diverse Untersuchungs- und Reaktionswerkzeuge, die helfen sollen, unerwünschte Ereignisse zeitsparend zu beheben.

Die grafische Schnittstelle

Durch die grafische Aufbereitung der Daten (siehe Abbildung 5), können Administratoren die Sicherheitsaspekte schnell erkennen und über die Konsole mit anderen Personen in Kontakt treten. Dies wird ermöglicht durch die Anfertigung von Berichten und der Möglichkeit diese Dokumente in ein bestimmtes Format zu exportieren. neuSECURE hierfür unterstützt folgende Funktionen:

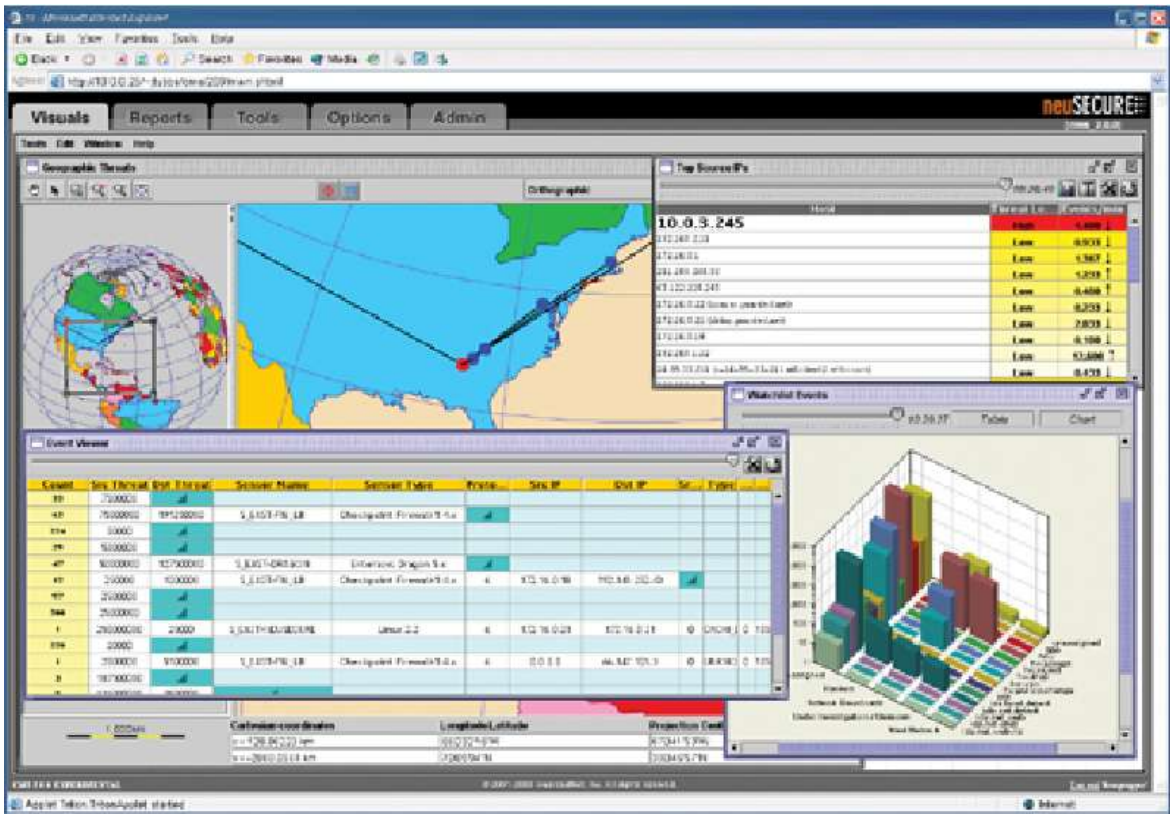

Abbildung 5 Die GUI von GuardedNet

- PowerGrid Event Viewer
  Daten können in vielen unterschiedlichen Fenstern angezeigt werden. Diese Daten können grafisch dargestellt, ein Report erzeugt oder in verschiedene Formate umgewandelt werden.

- Geographical Threat Manager
  Dieser erkennt die Quelle der Ereignisse und deren Ziele, welches in einer 3D-Sicht angezeigt werden können.
- TimeSlider
  Mittels dieser Funktion ist es möglich vorangegangene Ereignisse in der Reihenfolge ihres Auftretens anzusehen.

3.2.3 nFX OSP

NetForensic entwickelte 1999 ein Security Information Management System namens *nFX Open Security Platform* (nFX OSP). Diese bietet eine automatisierte Lösung für zentrales Log- und Event Management in Echtzeit sowie für forensische Analyse. Gleichzeitig beinhaltet es umfangreiche Möglichkeiten für statische und dynamische Berichterstellung. Eine flexible Architektur ermöglicht die Anpassung an technische, organisatorische und geographische Gegebenheiten.

- Geräte Integration
  nFX OSP integriert verschiedene Netzwerk- und Sicherheitsgeräte wie Intrusion Detection Systeme, Firewalls und Antivirus Systeme.
- Architektur
  Die Architektur ist ausgelegt für hohe Leistungsfähigkeit und Skalierbarkeit des Netzwerkes.
- Administration
  nFX OSP kann den Netzwerkverkehr kontinuierlich überwachen und Berichte anfertigen, wenn Probleme entstanden sind. Desweiteren können automatisch Modifikationen vorgenommen werden, um die Leistungsfähigkeit des Systems zu verbessern.

Funktionen von nFX OSP

Im folgenden werden die Funktionen, die von nFX OSP angeboten werden, näher beschrieben.

- Zusammenführen von Informationen (Consolidation)
  Für das Zusammenführen der Informationen sind Agenten für verschiedene Produkte und Anwendungen verfügbar. Sie ermöglichen das dezentrale Sammeln und Aufbereiten der Ereignisse und bieten eine sichere, verschlüsselte Verbindung zum Manager an.
- Übersetzen von Datenformaten und Ereignis Information (Normalisation)
  Die Normalisierung ist das Herzstück des Systems. Ereignisse von diversen Datenquellen wie Firewalls, Router, IDS-Systeme, Viren-Scanner, Logdateien, Datenbanken oder Web-Server werden in ein einheitliches Format (XML-Format) übersetzt und können somit korreliert werden.
- Verdichten von Informationen (Aggregation)
  Für die Optimierung der Systemleistung müssen die vielen Daten, die gesammelt wurden, reduziert werden. Diese Aufgabe übernimmt ein Filter, damit nur relevante Information Verwendung finden.
- Erkennen von Beziehungen und Abhängigkeiten (Correlation)
  Durch die Normalisierung der Informationen wird eine Korrelation mit herstellerunabhängigen Daten erst möglich. Unterstützt werden statistische und regelbasierende Korrelationen. Mit dem *Vulnerability Correlation Module* können sicherheitsrelevante Ereignisse automatisch erkannt werden. Für die Erstellung von Regeln existiert ein grafischer *Rule Builder*. Die statistische Korrelation bietet eine rein mathematische Sicht auf die Netzwerkaktivitäten.
- Visuelle Aufbereitung von Informationen (Visualizing)
  Um sich einen Überblick zu verschaffen und Problembereiche schnell zu erkennen, werden mit einem Zusatz-Modul Informationen in der Datenbank grafisch aufbereitet und mit hoher Aussagekraft dargestellt, z.B. Standort, Abteilung, Zeitpunkt, Art des Angriffs. Einige Beispiele zur Visialisierung sind (siehe dazu Abbildungen 6-8):

- Expert Assistent
  Führt den Benutzer durch die verschiedenen Sichten, um schnell an die Informationen zu gelangen die zur Identifizierung und Analyse benötigt werden.
- Link Map
  Erlaubt dem Administrator den Weg zu sehen, wie sich der Angriff ausgebreitet hat.
- Geo Map
  Erlaubt dem Analysten die Ereignisse nach Land und Städten aufzuspüren, markiert verdächtigen Verkehr von bestimmten Ländern und zeichnet verdächtige Quellen auf.

- Automatisierung von Aufgaben (Automate)
  Dies beinhaltet insbesondere die Bereiche der *Notification* und *Reporting*. Im Bereich Notification wird definiert, wann und wie bei einem spezifischen Ereignis alarmiert wird. Unterstützt wird zudem das zeitgesteuerte Reporting und die automatisierte Verteilung in verschiedenen Formaten (PDF, XLS, HTML, JPG) über E-Mail.

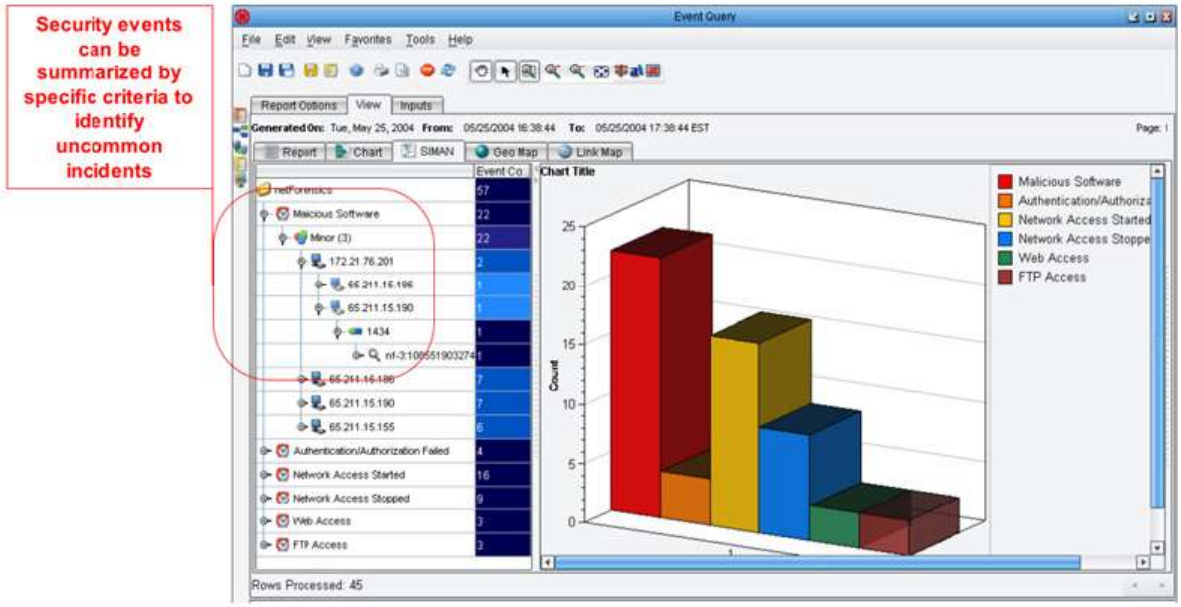

Abbildung 6 Identifizierung und Analyse

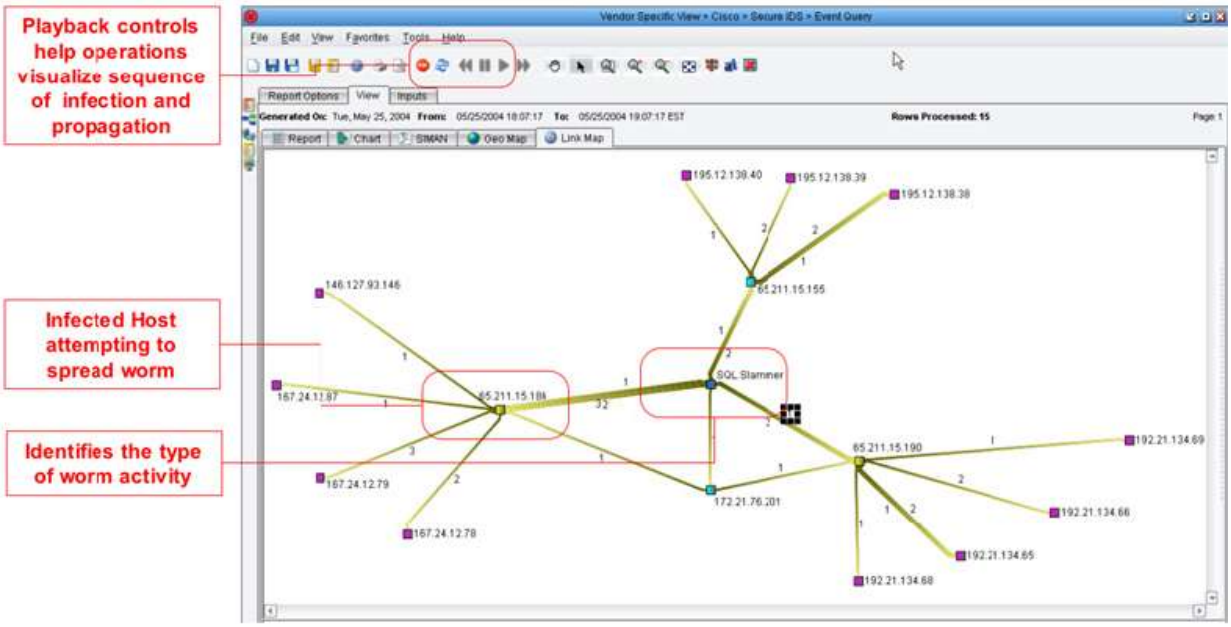

Abbildung 7Ausbreitung von Angriffen

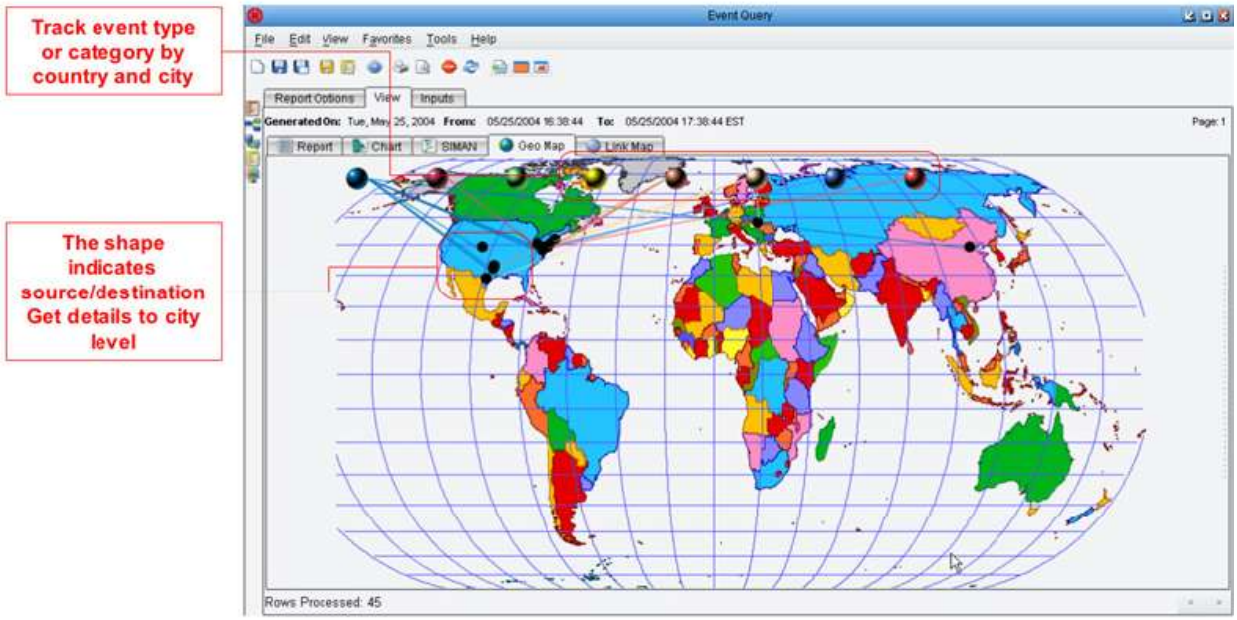

Abbildung 8 Geologische Karte

3.3 Bewertung und Vergleich

In Tabelle 1 ist die Bewertung der Intrusion Detection Management Software zu den im folgenden beschriebenen Kriterien dargestellt. Dabei werden die Kriterien nach unten stehenden Gesichtspunkten beurteilt:

	ACID	Scotty	nFX OSP	ArcSight	neuSecure
Funktionsumfang	+	+	+	+	+
Handhabbarkeit	+	0	+	+	?
Individualisierbarkeit	-	0	?	+	?
Dokumentation	0	+	+	+	0
Installation/Konfiguration	0	+	?	?	?
Erweiterbarkeit	+	0	+	+	+

+/-: Produkt fiel positiv/negativ auf
0: Neutrale Wertung
?: Keine eindeutige Bewertung möglich, z.B. weil Angaben des Herstellers fehlten

Tabelle 1 Bewertung der IDM Systeme

- Funktionsumfang
  Wie viele und welche Funktionen stellt die Software bereit? Enthält die Software die wichtigsten Funktionen, um das Netzwerk überwachen zu können?
- Handhabbarkeit
  Ist die graphische Oberfläche (GUI) benutzerfreundlich? Sind die verwendeten Icons intuitiv verständlich?
- Individualisierbarkeit
  Kann die GUI bzw. das System den Bedürfnissen unterschiedlicher Benutzer angepasst werden?
- Dokumentation
  Gibt es ein Handbuch zu der Software? Ist eine Hilfefunktion vorhanden?

- Installation/Konfiguration
  Wie aufwendig ist die Installation? Gibt es eine Anleitung, die eine einfache Installation bzw. Konfiguration ermöglicht?
- Erweiterbarkeit
  Lassen sich weitere Module z.B. für neue Geräte und Funktionen hinzufügen?

ACID zeichnet sich durch eine Vielzahl von Funktionen aus und bietet verschiedene Such- und Sortierfunktionen. Nachteilig ist die fehlende Individualisierbarkeit und die nicht vorhandene Möglichkeit, das Netzwerk grafisch darzustellen. Scotty dagegen erlaubt die Erstellung von Netzwerkansichten, wobei das Verschieben von Komponenten und die Menüführung ungewöhnlich, aber leicht zu erlernen ist. Positiv erweist sich zudem der große Funktionsumfang und die Erweiterbarkeit durch selbst geschriebene Skripte.

Die Beurteilung der kommerziellen Systeme konnte allein aufgrund der Informationen der vorhandenen Literatur erfolgen, da Testversionen nicht zur Verfügung standen. Bei allen Produkten wurde der große Funktionsumfang angepriesen. Durch die Verwendung von Normalisierung, Aggregation und Korrelationstechniken bieten die Managementsysteme eine effiziente Informationsaufbereitung an. neuSecure und ArcSight stellen zudem Untersuchungs- und Reaktionswerkzeuge zur Verfügung.

Bei der freien Software hat sich gezeigt, dass es zwar gute Ansätze, aber bisher nur recht unvollständige Lösungen gibt. Im kommerziellen Bereich wird dagegen eine große Zahl von Funktionalitäten angeboten. Letztere sollten z.T. auch bei der Entwicklung von unIDS beachtet werden.

4 Der Aufbau von unIDS

Der allgemeine Aufbau einer mit einem IDM versehenen Sicherheitsstruktur ist in Abbildung 9 zu sehen. Für alle Sicherheitswerkzeuge wird ein Agent zur Verfügung gestellt, der dem IDM Daten übermittelt und in der Gegenrichtung Steueranweisungen erhält. Prinzipiell ist auch eine Steuerung der Sicherheitswerkzeuge über die Agenten möglich, aber zunächst nicht vorgesehen. Das IDM selber bereitet die eingehenden Daten auf und stellt sie dem Administrator in geeigneter Weise dar. Konkrete Reaktionen im Falle eines Angriffes werden nicht unterstützt. Ist dieses gewünscht, müssen entsprechende Signale an externe Prozesse übermittelt werden.

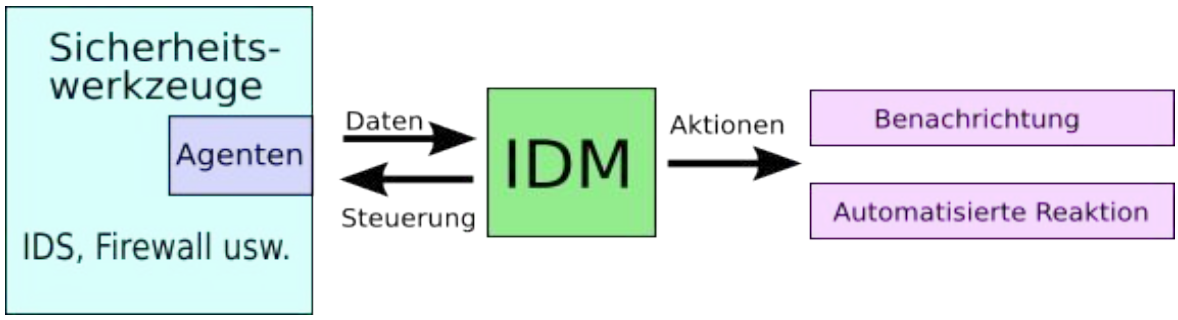

Abbildung 9 Sicherheitsstruktur mit IDM

4.1 Die unIDS Architektur

Abbildung 10 zeigt den detailierteren Aufbau von unIDS. Entsprechend der Anordnung in der Grafik, werden die einzelnen Aufgaben in Schichten aufgeteilt, mit denen sie im weiteren Verlauf identifiziert werden. So befindet sich die GUI im oberen Teil und wird dementsprechend mit Oberschicht bezeichnet. Analyser, Datenbank usw. sind in der Mitte angeordnet und werden wie bereits in der Grafik geschehen als Komponenten der Mittelschicht gesehen. Die unten angesiedelten Agenten sind demnach in der Unterschicht. Bisher noch nicht erwähnt wurde der Proxy, welcher in Abbildung 10 einem Subnetz zugeordnet ist. Ein Proxy dient im Wesentlichen der Skalierbarkeit des Systems und gestattet einen hierarchischen Aufbau der Agentenstruktur.

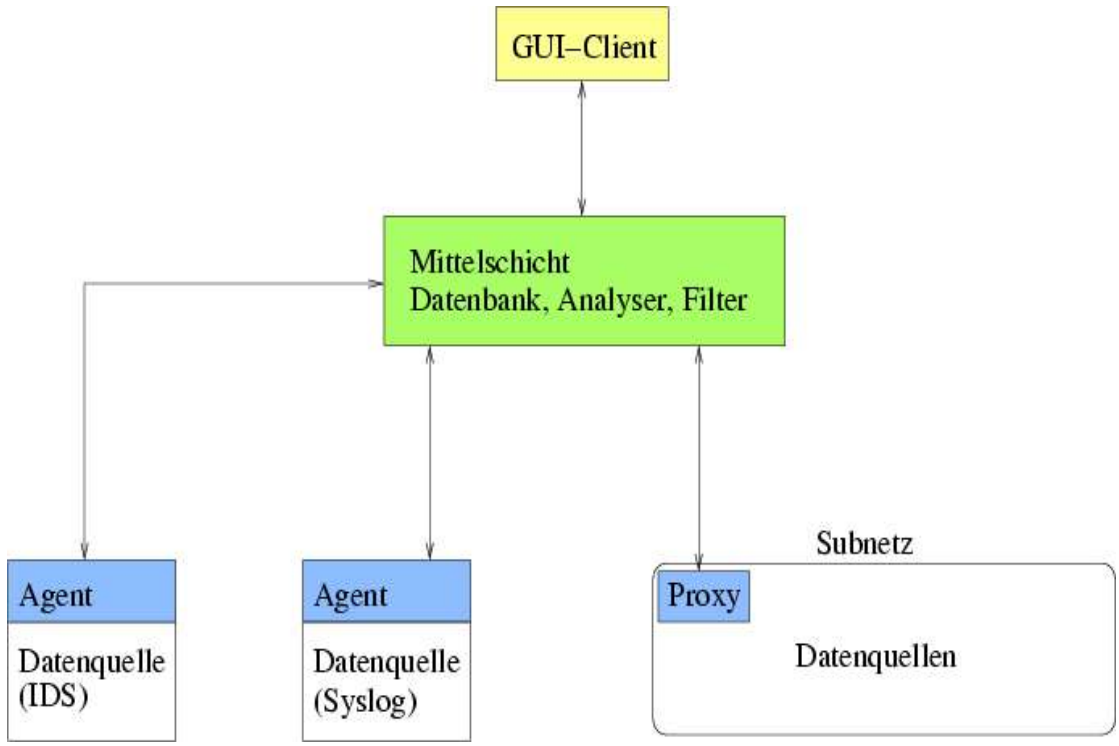

Abbildung 10 Aufbau von unIDS

Prinzipiell kann jeder Agent seine Daten direkt an die Mittelschicht senden, bei größeren Netzwerken mit unterschiedlichen Subnetzen, die ggf. mit eigenen Firewalls gesichert sind, kann eine Änderung im System dann allerdings zu einem großen Rekonfigurationsaufwand führen. Aus diesem Grund ist es möglich, Proxys zwischen zu schalten. Mehrere Agenten werden einem Proxy zugeordnet und kommunizieren nur noch mit diesem. Ist ein Subnetz durch eine Firewall getrennt, wird zusätzlich das Durchtunneln über den Proxy realisiert. Für eine noch weiter gehenden Strukturierung ist es zudem möglich, mehrere Proxys zu gruppieren, die ihrerseits nur mit einem übergeordneten Proxy kommunizieren. Zusätzlich kann ein Proxy auch noch über eine Vorfilterfunktion verfügen, die Nachrichten von geringem Interesse abfängt und somit den administrativen Datenverkehr vermindert.

4.2 Die Kommunikation von unIDS

Zum Austausch der Nachrichten baut unIDS auf standardisierte Formate und Übertragungsprotokolle auf. Dieses sind:

- Das Intrusion Detection Message Exchange Format (IDMEF)
  Der Standard wurde[4] entwickelt, um zu ermöglichen, dass verschiedene Systemen, egal ob kommerziell oder frei verfügbar, ihre Daten untereinander austauschen können. Es soll es den Benutzern leichter gestatten, verschiedene Systeme miteinander einzusetzen. Für genauere Informationen siehe http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-14.txt.

- Das Intrusion Detection Exchange Protocol (IDXP)
  Einem Zitat laut Robert Krauz[5] ist IDXP folgendermaßen definiert:
  „IDXP ist ein verbindungsorientiertes Protokoll der Anwendungsschicht. Es wurde von der IDWG für den Austausch von Daten zwischen verschiedenen IDS ausgearbeitet. Dieser Datenaustausch findet hauptsächlich zwischen Analysierer und Manager des IDS statt. Primär realisiert es den Austausch von IDMEF-Nachrichten, kann aber auch einfachen Text oder binäre Daten übertragen. Das IDXP spezifiziert ein BEEP - Profil und hat dadurch die gleichen Eigenschaften und Vorteile, wie das BEEP-Framework generell. Authentifikation, Integrität, Zuverlässigkeit und andere wichtige Punkte des Protokoll-Designs müssen nicht neu implementiert werden, sondern können ohne großen Aufwand einfach, durch das Einbinden vorhandener BEEP-Profile (z.B. TLS oder SASL), übernommen werden. Das IDXP-Profil ist bei der IANA unter http://iana.org/beep/transient/idwg/idxp registriert. Es muss also nur in der Start-Nachricht, während des Kanalaufbaus zwischen dem Client und dem Server, vereinbart werden.“

- Das Blocks Extensible Exchange Protocol (BEEP)
  Aus der gleichen Quelle stammt das Zitat zu BEEP:
  „BEEP ist ein verbindungsorientiertes, asynchrones Protokoll-Framework der Anwendungsschicht. Es wurde entworfen, um den Aufbau von Netzwerk-Anwendungs-Protokollen zu vereinfachen und zu verbessern. BEEP liefert einen Mechanismus, der verschiedene Problemfelder des Protokoll-Designs, in eine Menge von wiederbenutzbaren Modulen integriert. Diese Module werden Profile (profiles) genannt. Immer wieder auftauchende Probleme des Protokoll-Designs sind z.B. Authentifikation, Verschlüsselung, Kompression oder die Fehlerbehandlung. Diese Elemente werden im BEEP in die wiederbenutzbaren Module gepackt und müssen dadurch nicht immer wieder neu implementiert werden. Das ist auch der entscheidende Vorteil gegenüber herkömmlichen Protokollen. Das Protokoll setzt auf ein vorhandenes Transportprotokoll auf, was im RFC 3081 (Mapping the BEEP Core onto TCP) am Beispiel TCP veranschaulicht wird. BEEP dient als Grundlage, des von der IDWG beschriebenen Intrusion Detection Exchange Protocols (IDXP), ist beschrieben im RFC 3080 und wurde im März 2001 von der BEEP Working Group vorgestellt.“

- Twisted
  Twisted ist ein *event-driven networking framework* für Python, das im wesentlichen bedeutet,

4 Genaugenommen wird er noch weiterentwickelt.
5 http://www-rnks.informatik.tu-cottbus.de/de/materials/ws2002seRecent/krauz.pdf

dass Twisted ereignisorientiert arbeitet. Für den Benutzer bedeutet dies, dass er so genannte *callbacks* schreibt, welche vom Framework aufgerufen werden. Die verschiedenen Twisted-Projekte unterstützen im Moment TCP, UDP, SSL/TLS, multicast, Unix-Sockets sowie eine große Anzahl von Anwendungsprotokollen (HTTP, NNTP, IMAP, SSH, IRC, FTP, usw.). Weitere Informationen zu Twisted sind auf der Originalwebseite: www.twistedmatrix.com, sowie wikipedia http://en.wikipedia.org/wiki/Twisted zu finden.

4.3 Die Agenten

Ein Agent stellt in unIDS die unterste Schicht dar und ist ein sogenannter *Monitoring-Client*, der auf einem Host ausgeführt wird, um sicherheitsrelevante Informationen zu sammeln und an die Mittelschicht weiter zu leiten. Der Zugriff auf diese Informationen geschieht dabei über Plugins, die eine Schnittselle zwischen dem Agenten und den eingesetzten IDS bilden.

4.3.1 Zielsetzungen und Anforderungen

Aus Anwendersicht ist das gewünschte Ziel, dass der Agent auf einem Rechner ausgeführt wird, um auf diesem eine möglichst effiziente und konstruktive Überwachung von sicherheitsrelevanten Informationen durchzuführen. Hierzu verlässt sich der Agent u.a. auf die Vorgehensweise installierter IDS-Programme, die dem Agenten Informationen über stattfindende Angriffe bzw. Missbrauchsversuche liefern. Die gesammelten Information werden dann an die Mittelschicht übergeben, bevor sie dem Anwender aufbereitet in der GUI zur Verfügung gestellt werden. Als Schnittstelle zu den unterschiedlichen IDS ist der Einsatz von Plugins vorgesehen. Die Plugins sind dafür zuständig, dass die Informationen eines IDS gesammelt oder Logdateien mit sicherheitsrelevanten Daten analysiert werden. Die Verwendung von Plugins gestattet es, dass der Agent nachträglich um unterschiedliche Funktionalitäten erweitert werden kann. Damit nun eine nachträgliche Integration von Plugins realisierbar ist, der Agent modular aufgebaut werden.

Ein vorrangiges Anliegen bei der Entwicklung des Agenten besteht darin, ein *Monitoring* von sicherheitsrelevanten Informationen zu realisieren und dabei ein anwendbares Rahmenwerk für den Einsatz von Plugins bereitzustellen. Zur Erfüllung dieser Anforderungen muss der Agent in der Lage sein, folgende Aufgaben zu erfüllen:

- Auf- und Abbau einer Verbindung zur Mittelschicht
- Realisierung eines clientseitigen BEEP-Protokoll
  (Senden und Empfangen von BEEP-Nachrichten)
- Verarbeitung von XML-basierten BEEP-Nachrichten
- Aufrechterhaltung und Überwachung der Verbindung zur Mittelschicht
- Bereitstellung und Verwaltung von allokierten BEEP-Kanälen
- Laden von verfügbaren Plugins
- Aktivieren und Deaktivieren von einzelnen Plugins

4.3.2 Anwendungsfälle

Aus den oben beschriebenen Zielsetzungen lassen sich unterschiedliche Anwendungsfälle für den Agenten ableiten. In den folgenden Abschnitten sollen diese näher erläutert werden.

Anmelden in der Mittelschicht

Nachdem das Management einen Kanal zur Mittelschicht aufgebaut hat, meldet der Agent seine Bereitschaft dort an. Bei der Anmeldung sind zwei Fälle zu unterscheiden:

- Verfügt der Agent über keine eindeutige ID, so liegt eine erstmalige Anmeldung vor, die mit einer ID-Anforderung an die Mittelschicht verbunden ist. Die Anmeldung erfolgt dabei mit der Aussendung einer *GreetingMessage*, die das Attribut *request_id* beinhaltet. Die Mittelschicht quittiert diese Anforderung ebenfalls mit einer GreetingMessage, die neben dem Attribut *set_id*

eine eindeutige ID für den Agenten enthält.

- Verfügt der Agent bereits über eine eindeutige ID, die bei der Initialisierung des Agenten aus einer lokalen Datei eingelesen wird, so erfolgt die Anmeldung in der Mittelschicht ohne eine ID-Anforderung, d.h. die GreetingMessage wird ohne das Attribut request_id an die Mittelschicht gesendet. Die Mittelschicht quittiert diese Anmeldung ebenfalls mit einer GreetingMessage, die neben dem Attribut set_id die ID des Agenten enthält. Die ID dient dem Agenten als Absender-ID bei künftigen Nachrichtenübertragungen.

Nach der Anmeldung in der Mittelschicht initiiert der Agent den Aufbau von zusätzlichen BEEP-Kanälen für die Profile *Data* und *IDXP*, über welche letztendlich die gesammelten Information in Form von spezifizierten XML-Nachrichten an die Mittelschicht gesendet werden können.

Abmelden aus der Mittelschicht

Der Agent meldet sich aus der Mittelschicht ab, wenn das Programm des Agenten beendet wird. Hierzu sendet der Agent die XML-Nachricht *SignOffMessage*. Nach einer ordnungsgemäßen Abmeldung des Agenten aus der Mittelschicht ist eine erneute Anmeldung des Agenten in der Mittelschicht möglich.

Laden und Starten von einzelnen Plugins

Nach der erfolgreichen Anmeldung des Agenten an der Mittelschicht und dem Aufbau der drei logischen BEEP-Kanäle *Managemant, Data* und *IDXP* können im Agenten alle Plugins, die in einer Konfigurationsdatei aufgeführt sind, geladen werden. Die Konfigurationsdatei legt auch fest, welche von den verfügbaren Plugins unmittelbar nach dem Laden gestartet werden sollen.

Änderung der Konfiguration von einzelnen Plugins

Über die GUI hat der Anwender die Möglichkeit, die Konfiguration von einzelnen im Agenten verfügbaren Plugins zu beeinflussen. Die GUI sendet hierzu in einer XML-Nachricht eine entsprechende Konfigurationsänderung an die Mittelschicht. Die Mittelschicht verarbeitet diese Nachricht und leitet die Konfigurationsänderung in der XML-Nachricht *ConfigurationMessage* an den Agenten weiter. Der Agent vergleicht den Inhalt mit der aktuellen Konfiguration und ändert diese gegebenenfalls entsprechend der neu angeforderten Konfiguration. Über eine *ConfigurationMessage* können beispielsweise einzelne Plugins ein- bzw. abgeschaltet oder bestimmte Plugin-Parameter nachträglich verändert werden.

Senden von gesammelten Informationen

Um die Aussendung der gesammelten Informationen zu ermöglichen, stellt die Software des Agenten ein Rahmenwerk zu Verfügung, das die erforderlichen BEEP-Kanäle *Data* und *IDXP* unterhält und eine Methode für das Senden von gesammelten Informationen implementiert. Sobald ein Plugin über entsprechende Informationen verfügt, sendet er diese in Form von XML-Nachrichten über den korrespondierenden BEEP-Kanal an die Mittelschicht.

Die Implementierung der Agenten und Plugins soll an dieser Stelle nicht vertieft werden. Weitergehende Details zu der Entwicklung von Agenten sind in der technischen Referenz[PG0405] ab Abschnitt 5.3.1 und in der Diplomarbeit [JS05] zu finden.

4.4 Die Mittelschicht

Die Mittelschicht ist die zentrale Komponente von unIDS. Sie ist für die Verwaltung aller anderen Komponenten, den Nachrichtenaustausch und die Nachrichtenanalyse verantwortlich. Im Folgenden wird zunächst ein Überblick über die Mittelschicht gegeben. Daran anschließend werden die Entwurfsentscheidungen sowie die Architektur der Mittelschicht erläutert.

Eine detailierte Beschreibung des Datenmodells, der Nachrichtenanalyse und weiterer Komponenten der Mittelschicht sind in der technischen Referenz[PG0405] des unIDS Projektes zu finden.

4.4.1 Aufgaben der Mittelschicht

In der Mittelschicht als zentrale Komponente sind unterschiedliche Aufgaben zusammengefasst. Diese reichen von der Verwaltung der verschiedenen unIDS-Komponenten über die Organisation des Systemzustandes bis hin zur Nachrichtenverarbeitung und -analyse.

In diesem Abschnitt wird zunächst ein Überblick über die verschiedenen grundsätzlichen Aufgaben der Mittelschicht gegeben. Detailliertere Beschreibungen sind in den daran anschließenden Abschnitten enthalten.

Verwaltung der unIDS-Komponenten

Eine wesentliche Aufgabe der Mittelschicht als zentraler Komponente von unIDS ist die Verwaltung der anderen am System beteiligten Komponenten. Hierzu zählen auf der einen Seite die in Abschnitt 4.3 vorgestellten Agenten sowie auf der anderen Seite die in Abschnitt 4.5 beschriebene Benutzerschnittstelle.

Allen Komponenten innerhalb von unIDS wird dabei ein systemweit eindeutiger Identifikator zugeordnet. Über diesen Identifikator kann die Komponente dann innerhalb des Systems angesprochen werden. Die Mittelschicht stellt dabei sicher, dass jeder neu vergebene Identifikator systemweit eindeutig ist und dass an ihn adressierte Nachrichten über die Mittelschicht ihr Ziel erreichen können.

Nachrichtenaustausch

Neben der Verwaltung der Komponenten ist die Mittelschicht für den Datenaustausch zwischen den Komponenten zuständig. Hierbei sind generell zwei Nachrichtenklassen zu unterscheiden:

- Nachrichten mit einem speziellen Umschlag, der den Empfänger und Absender in Form des zuvor beschriebenen Identifikators enthält. Die zwischen Agent beziehungsweise Proxy und Mittelschicht ausgetauschten Nachrichten gehören grundsätzlich dieser Klasse an[6]
- Nachrichten zwischen der grafischen Benutzerschnittstelle und der Mittelschicht; diese benötigen den Umschlag nicht, da sich der Empfänger implizit aus der Verbindung ergibt.

Die Mittelschicht muss neben dem reinen Austausch der Nachrichten auch in der Lage sein, diese zu verarbeiten und gegebenenfalls zugehörige Folgenachrichten oder Statusänderungen auszulösen. Dies bedeutet zum Beispiel, dass bei eingehenden IDMEF-Nachrichten diese dem richtigen Gerät-Eintrag zugeordnet werden und der Status für diesen Eintrag entsprechend verändert wird. Zusätzlich sind IDMEF-Nachrichten direkt an jede verbundene Benutzerschnittstelle sofort weiterzuleiten.

Verwaltung des Systemzustands

unIDS besitzt einen systemweiten Zustand, der über die Mittelschicht verwaltet und auch aktualisiert wird. Andere Komponenten können den systemweiten Zustand durch von ihnen stammende Nachrichten beeinflussen oder Teile davon abfragen. Der Zustand teilt sich in einen persistenten Teil und einen Laufzeitteil, der bei jedem Neustart verloren geht.

Im Systemzustand sind die Informationen über alle bekannten Agenten, die Benutzerdaten sowie die letzten Statusinformationen zu den verschiedenen Geräten enthalten.

Im Allgemeinen bilden die Agenten in Bezug auf den Systemzustand die Informationslieferanten, damit sind sie in der Regel diejenigen, die den Systemzustand beeinflussen. Die grafische Benutzerschnittstelle auf der anderen Seite benötigt für die Visualisierung Informationen über den Systemzustand und bildet daher eine Senke.

---

6 Ausgenommen IDMEF-Nachrichten, da dies nicht mit den entsprechenden RFCs vereinbar wäre

Verarbeitung und Analyse

Die vor allem von den Agenten gelieferten Informationen müssen zunächst den richten Datensätzen innerhalb des Systemzustandes zugeordnet werden. Dazu muss die Mittelschicht die in den Nachrichten enthaltenen Informationen analysieren und in den Systemzustand einpflegen. Den Regelfall bilden dabei die Statusinformationen der verschiedenen Plugins der Agenten. Die in diesen Nachrichten enthaltenen Informationen müssen im Wesentlichen unverändert in den Systemzustand übernommen werden. Daneben werden durch die Agenten auch IDMEF-Nachrichten versandt, die als solche zwar direkt an alle angemeldeten Benutzerschnittstellen weitergeleitet werden, darüber hinaus aber auch zu Veränderungen im Systemzustand führen müssen. Auch hier muss zunächst eine korrekte Zuordnung zu dem entsprechenden Datensatz im Systemzustand hergestellt werden. Für diesen Datensatz wird dann der „Status-Parameter“ aktualisiert.

Anforderungen an die Mittelschicht

Im Folgenden werden die grundsätzlichen Anforderungen an die Mittelschicht in einer Aufzählung dargestellt.

- Verbindung zu vielen Komponenten
  Sowohl zu den Agenten als auch zu den grafischen Benutzerschnittstellen müssen unter Umständen eine Vielzahl an Verbindungen unterhalten werden können.
- Robustheit
  Da ein Ausfall der Mittelschicht unIDS komplett zum Stillstand bringt, muss die Mittelschicht in besonderem Maße robust sein. Dies bedeutet, dass auftretende Fehler möglichst nicht zu einem kompletten Ausfall der Mittelschicht führen sollten.
- Schnelligkeit
  Die Nachrichten sollten möglichst so schnell verarbeitet werden können, dass für den Benutzer an der grafischen Benutzerschnittstelle keine unangenehmen Verzögerungen auftreten. Dies heißt für die Mittelschicht, dass insbesondere bei Nachrichten, die direkt an die grafischen Benutzerschnittstellen weiterzuleiten sind, die Verzögerungszeiten möglichst gering zu halten sind.
- Persistenz
  Der Systemzustand muss auch nach einem Neustart der Mittelschicht erhalten bleiben. Bestimmte Teile des Zustandes, die nur zur Laufzeit relevant sind dürfen dabei verloren gehen.
- Portabilität
  Die Mittelschicht *sollte* auf möglichst vielen Betriebssystemen nutzbar sein. Sie *muss* mindestens auf Linux-kompatiblen Betriebssystemen nutzbar sein.

4.4.2 Entwurfsentscheidungen

In diesem Abschnitt werden zunächst die für die Mittelschicht getroffenen Entscheidungen zum Vorgehensmodell beschrieben. Daran anschließend werden die Entscheidungen zur verwendeten Programmiersprache sowie zu den genutzten Bibliotheken und Frameworks dargestellt und begründet.

Die Programmiersprache

Zur Auswahl für die Programmiersprache standen zum einen Java, wodurch sich Synergien zwischen der Entwicklung der Mittelschicht und der grafischen Benutzeroberfläche ergeben hätten und Python, einer interpretierten, objektorientierten Sprache, die ihre Stärken vor allem in der schnellen Entwicklung von Prototypen und einer umfangreichen Systembibliothek hat. Beide Sprachen sind ähnlich portabel.

Da für die Entwicklung zunächst kurzfristig ein lauffähiger Prototyp entstehen sollte, an dem weitere Entscheidungsmerkmale durchgetestet werden können, fiel die Wahl auf Python.

Die guten Erfahrungen bei der Erstellung des Prototypen und die umfangreichen zur Verfügung stehenden Bibliotheken führten im Laufe der Entwicklung zu der Entscheidung, auch die weitere Implementierung der Mittelschicht in Python zu realisieren. Speziell auch die Möglichkeit in Interaktion mit dem Interpreter neue Entwicklungen oder auch Bibliotheken in „Echtzeit“ zu testen hat sich als großer Vorteil im Vergleich zur Java-basierten Entwicklung gezeigt.

Im Vergleich zu Java stellen sich aber auch einige Nachteile insbesondere bei der Typsicherheit dar. Hier ist in hohem Maße ein diszipliniertes Vorgehen seitens der Entwickler erforderlich.

Verwendete Bibliotheken und Frameworks

Für die Entwicklung der unIDS-Mittelschicht wurden verschiedene Bibliotheken beziehungsweise Frameworks eingesetzt, die entweder bei der Entwicklung unterstützen sollten oder direkter Bestandteil der Funktionalität der Mittelschicht sind.

Als Basis für die Entwicklung der Mittelschicht wurde das in Abschnitt 4.2 beschriebene Framework Twisted verwendet. Die Entscheidung erfolgte im Wesentlichen aus zwei Gründen:

- Solides Serverframework
  Twisted bildet ein solides Serverframework. Es bringt eine Ereignissteuerung, die Verwaltung von Thread-Pools, robuste Netzwerkzugriffe sowie alle nötigen Funktionen, um schnell stabile Programme zu entwickeln, die als Serverprozess unter definierten Benutzerrechten laufen. Damit konnte die Entwicklung der Mittelschicht um wesentliche Arbeitsschritte erleichtert werden.
- Komponentenkonzept
  Twisted bringt ein umfangreiches Konzept zur Aufteilung der Anwendung in verschiedene lose gekoppelte Komponenten mit. Insbesondere in Bezug auf die unterschiedlichen Aufgaben der Mittelschicht sollte dies für eine übersichtlich gestaltete Architektur genutzt werden.

Die Grundlage für das Komponentenkonzept von Twisted bilden die im Rahmen des ZOPE-Projekts entwickelten Python-Erweiterungen für Interfaces, so dass auch in Python das Konzept von Schnittstellen verwendet werden kann.

Als Basis für die Kommunikation mittels des BEEP-Frameworks wurde die Implementierung *BEEPy* verwendet. Die Bibliothek ist zum einen auf der Basis von Twisted erstellt und wird zum anderen auch für die Implementierung des Agenten verwendet. Während der Entwicklung der Mittelschicht hat sich die relativ dünne Entwicklergemeinde um das BEEPy-Projekt als nachteilig erwiesen, da zum Teil erhebliche Fehler innerhalb der BEEPy-Implementierung mühsam selbst beseitigt werden mussten[7].

Für den persistenten Systemzustand war zunächst eine Eigenimplementierung mit einem *Relationalen Datenbank-Management-System* (RDBMS) im Hintergrund angedacht. Diese Entscheidung musste allerdings während der Entwicklung revidiert werden, da mit dieser Lösung nicht genug Flexibilität zur Reaktion auf geänderte Anforderungen durch die Umgebung der Mittelschicht geblieben war. Als wesentlich flexiblere Alternative wurde die Objektdatenbank *ZODB* des ZOPE-Projekts verwendet.

Nichtfunktionale Bestandteile

Im Zuge der Revidierung der Entwurfsentscheidungen zum persistenten Systemzustand und durch sich weiter ändernde Anforderungen an die Struktur des Systemzustands wurde auch die Struktur der Daten umgestellt. Zur Verwaltung der verschiedenen Strukturen über den vorhandenen Datensätzen wurden mehrere gerichtete Graphen eingesetzt, um auch bei weiteren Änderungen flexibel reagieren zu können. Als Basis für die Verwaltung der Datenstruktur wurde die Bibliothek *pygraphlib*[8] eingesetzt.

---

7 Die während der Entwicklung der Mittelschicht beim BEEPy-Projekt eingestellten BUG-Reports sind im Wesentlichen zum Ende dieses Projektes noch offen gewesen.

8 http://pygraphlib.sourceforge.net

Da die Mittelschicht nicht über eine direkte Benutzerschnittstelle verfügt, wurde konsequent der Logging-Mechanismus von Python eingesetzt. Dieser stellt eine hierarchische Logging-Umgebung zur Verfügung. Das Ziel dieser Entscheidung ist die Visualisierung der Vorgänge innerhalb der Mittelschicht für die beteiligten Entwickler. Der Umfang der auszugebenden Meldungen sollte für die verschiedenen Bereiche der Mittelschicht einzeln konfigurierbar sein. Zur Steigerung der Qualität bei der Entwicklung der Mittelschicht und zur Überprüfung der Spezifikation wurde vor allem bei der Entwicklung der Nachrichtenverarbeitung für die grafische Benutzerschnittstelle auf das in Python vorhandene Framework für Unit-Tests zurückgegriffen.

4.4.3 Architektur der Mittelschicht

In diesem Abschnitt wird die Architektur der Mittelschicht beleuchtet. Zunächst wird hierzu ein Überlick über das entwickelte Architekturmodell gegeben. Anschließend wird das für die Mittelschicht entwickelte Komponentenmodell beschrieben. Darauf folgend wird die Komponente *ServiceManager* beschrieben, die für die Verwaltung der Mittelschicht an sich und ihrer Bestandteile verantwortlich ist. Abschließend wird das Format der Konfigurationsdatei beschrieben, welches sich stark an dem Architekturmodell der Mittelschicht orientiert.

Architekturmodell der Mittelschicht

Wie schon in Abschnitt 4.4.1 angedeutet, kommen in der Mittelschicht ganz unterschiedliche Aufgabenbereiche zusammen. Um die Mittelschicht möglichst übersichtlich zu halten und um die einzelnen Bestandteile der Mittelschicht voneinander weitestgehend unabhängig zu halten, wurde die Architektur sehr stark komponentenbasiert aufgebaut, wie in Abbildung 11 dargestellt.

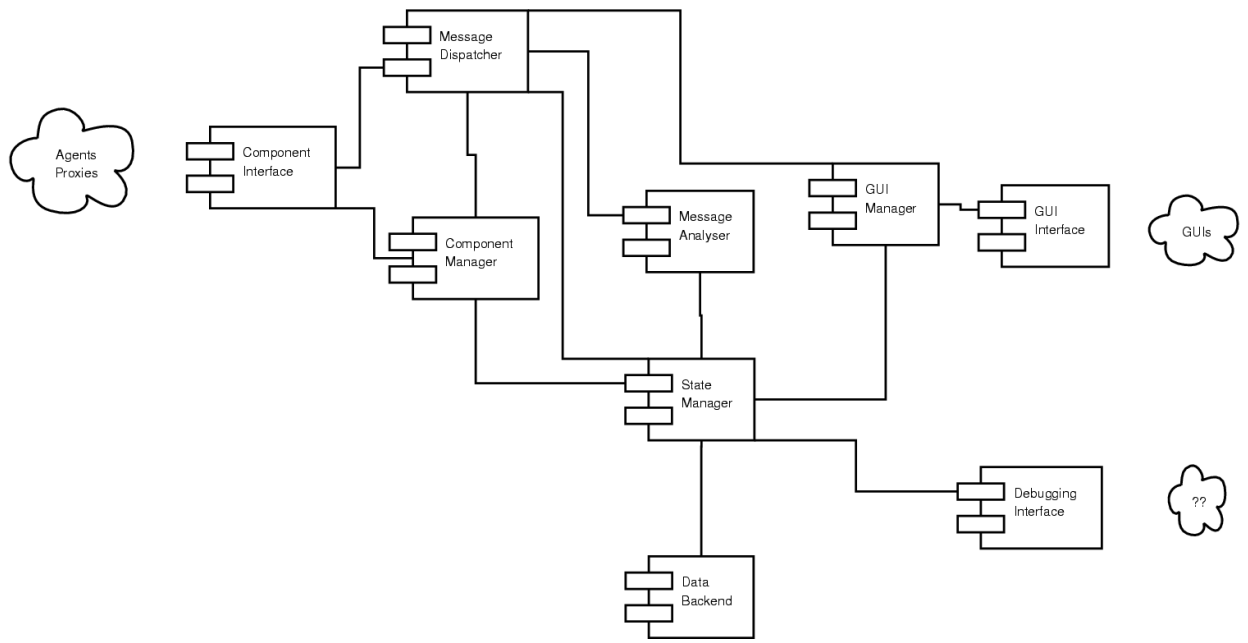

Abbildung 11 Architektur der Mittelschicht

- Das *ComponentInterface* ist für die Verbindung zu den einzelnen Komponenten (Agenten und Proxys) verantwortlich.
- Der *ComponentManager* verwaltet die verschiedenen unIDS-Komponenten, die über das ganze Netzwerk verteilt sein können. Eine der wichtigsten Aufgaben ist die Sicherstellung der eindeutigen Identifizierbarkeit jeder unIDS-Komponente.
- Das *DataBackend* ist für die Datenhaltung verantwortlich. Der vom *StateManager* festgehaltene Zustand muss passend auf eine Datenbank abgebildet werden.
- Das *DebuggingInterface* Diese Komponente ist für Debugging Zwecke vorgesehen. Sie wird bei

Bedarf implementiert. Ihr Vorhandensein darf das Verhalten von unIDS genauso wenig ändern, wie ihr Fehlen[9].

- Das *GuiInterface* ist mit dem ComponentInterface vergleichbar. Es ist für die Verbindungen zu den aktiven Benutzerschnittstellen verantwortlich. Auf Grund der besonderen Eigenarten der Verbindung zur grafischen Benutzerschnittstelle wurde diese Funktion nicht mit im ComponentInterface implementiert.
- Der *GuiManager* ist dem ComponentManager sehr ähnlich. Die Verwaltung der grafischen Benutzerschnittstellen unterscheidet sich jedoch erheblich von der Verwaltung der anderen Komponenten, so dass hierfür eine eigene Komponente geschaffen wurde.
- Der *MessageAnalyser* ist verantwortlich für die Analyse der eingehenden Nachrichten im Kontext des aktuellen Systemzustands.[10]
- Der *MessageDispatcher* ist innerhalb der Mittelschicht für die richtige Zustellung der Nachrichten verantwortlich. Dabei können vom ComponentInterface kommende Nachrichten entweder direkt für die grafische Benutzerschnittstelle oder für die Mittelschicht bestimmt sein.
- Der *StateManager* verwaltet den unIDS-Zustand. Wie schon in den Anforderungen in Abschnitt 4.4.1 beschrieben, besteht der Zustand aus einem persistenten und aus einem nicht persistenten Teil.

Eine ausführliche Beschreibung des Komonentenmodells befindet sich in Abschnitt 4.4.2 der technischen[PG0405] Dokumentation von unIDS.

4.4.4 Schnittstelle zu Agent und Proxy

Die Mittelschicht ist mit sämtlichen Agenten und Proxys direkt oder indirekt verbunden. In diesem Abschnitt wird beschrieben, wie innerhalb der Mittelschicht die Verbindung zu Agent und Proxys sowie die Agenten und Proxys selbst verwaltet werden.

Die Schnittstelle zu Agent und Proxy wird innerhalb der Mittelschicht im Wesentlichen durch die Komponenten ComponentInterface und ComponentManager realisiert. Dabei ist das ComponentInterface für die konkreten Verbindungen zu den verschiedenen unIDS-Komponenten verantwortlich, während der ComponentManager für das Verwalten der unIDS-Komponenten an sich zuständig ist.

- ComponentInterface:
    - Agent / Proxy meldet sich an
      Der Agent / Proxy muss durch das ComponentInterface beim ComponentManager registriert werden.
    - Agent / Proxy senden Nachrichten
      Sofern der Absender der Nachricht dem System bekannt ist, wird die Nachricht weiter verarbeitet. Dies bedeutet in der Regel, dass sie an den MessageDispatcher weitergeleitet wird.
    - Die Verbindung bricht ab
      Sobald ein Zusammenbruch der Verbindung bemerkt wird, muss das ComponentInterface die unIDS-Komponente beim ComponentManager abmelden.
    - Nachrichten an Agent / Proxy senden
      Falls Nachrichten an einen Agenten oder Proxy gesandt werden sollen, ist das ComponentInterface für die Übertragung der Nachricht verantwortlich
- ComponentManager

9 Diese Komponente wurde tatsächlich nicht implementiert, da über verschiedene andere Möglichkeiten ausreichend Einblicke in die Abläufe der Mittelschicht gewonnen werden konnten.

10 Im Rahmen der Implementierung wurde die Funktionalität der Analyse in den Handlern zu den einzelnen Nachrichtenklassen untergebracht, so dass bisher keine eigene Komponente zur Analyse vorhanden ist.

- Session-Identifikator erzeugen
  Jede pysikalische BEEP-Verbindung zur Mittelschicht ist eine Session. Der ComponentManager ordnet jeder Session einen eindeutigen Idenfikator zu, über den die Session erreichbar ist. Über diesen Identifikator wird der Session dann ein Kontextobjekt zugeordnet, welches eine Liste mit den über diese Session erreichbaren unIDS-Komponenten enthält.
- Session-Identifikator löschen
  Wird eine Verbindung beendet, so muss auch der zugehörige Session-Identifikator gelöscht werden.
- Anmeldung verarbeiten
  unIDS-Komponenten, die sich erstmalig oder aber mit einem schon vorhandenen Identifikator an der Mittelschicht anmelden, werden durch den ComponentManager als verbunden in den System-Zustand eingetragen. Analog hierzu muss das Abmelden verarbeitet werden.
- Zugriff auf Kontexte
  Nach der Verarbeitung der An- und Abmeldevorgänge stellt der ComponentManager den anderen Komponenten der Mittelschicht eine Möglichkeit bereit, die Kontextobjekte der angemeldeten unIDS-Komponenten zuzugreifen. Über diese Kontextobjekte können andere Komponenten dann zum Beispiel Nachrichten an die unIDS-Komponenten senden.

Der ComponentManager ist also insbesondere für die Verwaltung des Teils des Zustands der Mittelschicht verantwortlich, der Informationen über die vorhandenen Proxys und Agenten[11] enthält.

4.4.5 Schnittstelle zur grafischen Benutzeroberfläche

Die Schnittstelle zur grafischen Benutzerschnittstelle ist von den Komponenten her ähnlich aufgebaut, wie die Schnittstelle zu Agent und Proxy. Allerdings sind die Komponenten selbst anders realisiert, da zwischen der grafischen Benutzerschnittstelle und der Mittelschicht wesentlich mehr verschiedene Nachrichtentypen ausgetauscht werden als zwischen der Mittelschicht und dem Agent/Proxy.

Zusammenspiel von GuiInterface und GuiManager

Bei einer Anmeldung der grafischen Benutzerschnittstelle durch den Benutzer, wird vom GuiInterface über den GuiManager eine spezielle Sicht auf den Systemzustand der Mittelschicht angefordert. Diese Sicht ist schon personalisiert, das heißt, sie hat eine Referenz auf das ihr zugeordnete Benutzerobjekt. Sämtliche Zugriffe der grafischen Benutzerschnittstelle auf den Systemzustand der Mittelschicht erfolgen über diese personalisierte Sicht. An diesem Punkt würde auch die Zugriffssteuerung entsprechend der Rechte des aktuellen Benutzers aktiv werden.[12]

Der GuiManager ist dabei ausschließlich für die Authorisierung des Benutzers verantwortlich. Hierzu stellt er die Eigenschaft *portal* bereit, über die ein Benutzer am System angemeldet werden kann. Das Ergebnis einer solchen Anmeldung ist ein Objekt, dass die Schnittstelle *IAvatar*[13] bereitstellt und darüber den Zugriff auf das in der technischen Referenz[PG0405] beschriebene Datenmodell ermöglicht. Nach erfolgreicher Anmeldung eines Benutzers ist der GuiManager bis zum Ende der Sitzung des Benutzers nicht mehr aktiv.

Direkt nach der erfolgreichen Anmeldung übernimmt das GuiInterface alle weiteren Aufgaben, die sich im Wesentlichen auf die korrekte Verarbeitung der eingehenden Nachrichten beschränken. Dabei werden Nachrichten als XML-Zeichenketten von der grafischen Benutzerschnittstelle (Gui)

---

11 Proxys sind im Datenmodell zwar vorgesehen, wurden aber im Rahmen des Projektes nur als transparente Proxys realisiert, so dass sie bisher nicht im Systemzustand auftauchen können.
12 Derzeit nicht vollständig realisiert
13 Diese Schnittstelle ist im Paket unids.middlelayer.statemodel zu finden.

über das Netzwerk an die Mittelschicht übertragen. Dort werden sie innerhalb der Komponente GuiInterface weiterverarbeitet. Nach der Umwandlung der Nachricht in ein passendes Nachrichten-Objekt wird die Anfrage über passende Zugriffe auf die Geschäftslogik ausgeführt. Das Ergebnis wird dann wieder in eine XML-Zeichenkette verpackt und an die grafische Benutzerschnittstelle zurückgesandt.

GuiInterface

Das GuiInterface wickelt die Kommunikation mit der grafischen Benutzerschnittstelle ab. Dabei gilt es die eintreffenden XML-Zeichenketten in die richtigen Nachrichten-Objekte zu verpacken und danach zu verarbeiten.

Für die Verarbeitung der Nachrichten sind im GuiInterface zwei Bereiche für die Nachrichtenverarbeitung vorgesehen. Zunächst müssen die XML-Zeichenketten in die richtigen Nachrichten-Objekte umgewandelt werden. Hierzu dient die *Message-Registry*. Sie verwaltet eine Liste von Implementierungen der Schnittstelle *IMessageBuilder*, die in der Lage sind, bestimmte XML-Zeichenketten zu Verarbeiten und in die entsprechenden Nachrichten-Objekte umzuwandeln. Daneben ist die *Message-Listener-Registry* vorhanden, die eine Menge von Implementierungen der Schnittstelle IMessageListener verwaltet, die wiederum für bestimmte Nachrichtentypen registriert wurden und in der Lage sind genau diese Nachrichtentypen weiterzuverarbeiten.

Die Schnittstelle *IGuiInterface* beschreibt, wie die anderen Komponenten der Mittelschicht mit dem GuiInterface kommunizieren können. Die beiden Methoden dienen dazu, weitere Nachrichtentypen im GuiInterface zu registrieren.[14]

4.5 Die GUI

Die Ergebnisse im Bereich der Oberflächenprogrammierung aus der Projektgruppe 03/04[PG0304], sollten in der überarbeiteten Version von unIDS berücksichtigt werden. Um die bereits entwickelten Elemente einbinden zu können, wurde erneut die Programmiersprache Java verwendet.

Während im Anfangsstadium eine Implementierung der Software als Java-Applet realistisch erschien, zeigte sich mit zunehmenden Entwicklungsfortschritt, dass eine andere Java-Technologie besser für ein solches Vorhaben geeignet ist. Deshalb wurde die grafische Benutzer-Schnittstelle nicht mehr als Applet realisiert, sondern als eigenständige Applikation, die über das Internet mit Hilfe des *Java Network Launch Protocol* (JNLP) angesprochen werden kann.[JE2006]

Abschließend werden in diesem Abschnitt einige Vorgaben der ISO 9241[ISO9241] aufgeführt und erklärt, da die Benutzeroberfläche des unID-Systems entsprechend gestaltet werden soll. Dabei liegt der Schwerpunkt auf den Teilen 10-17.

4.5.1 Veränderungen des Prototypen

Bei der Überarbeitung des unIDS-Prototypen, der im Zuge der Projektgruppe 03/04[PG0304] entwickelt wurde, haben sich Optimierungen und Verbesserungen ergeben, die hier in Kürze dargestellt werden. Eine detaillierte Darstellung, in der auch die Arbeitsweise des ersten Prototypen beschriebe wird, befindet sich in der technischen Referenz[PG0405].

Kommunikation

Die ursprünglich eigenständig entwickelte Kommunikation zwischen GUI und Datenbank wird durch eine BEEP Implementierung ersetzt und somit standardisiert.

---

14 Weitere Informationen sind in Abschnitt 4.12.3 in der technischen Referenz[PG0405] zu finden.

Datenhaltung

Die Datenhaltung wird zur Vereinfachung der Struktur zentral gespeichert. Statt auf einen GUI-Proxys zurückzugreifen werden jetzt alle Daten direkt von der Datenhaltungs- und Analyseschicht (Mittelschicht) verwaltet.

XML Konzept

Die ursprünglichen weitreichenden Abhängigkeiten wurden durch eine Kapselung der einzelnen Schichten voneinander, sowie einer exakten Festlegung der einzelnen XML-Schemata aufgetrennt. Weiterhin wird jetzt die Mittelschicht direkt über die Änderungen informiert, damit eine parallele Arbeit sichergestellt werden kann. Eine Reduzierung der redundanten Informationen bei einer Speicherung wird durch ausschließliche Sendung von veränderten Daten realisiert. Ein ähnliches Schema gilt für die Abfrage von Daten. Hierdurch wird durch eine explizite Anfrage der Aufruf spezifischer Daten ermöglicht.

Ergonomische Aspekte

Die in der ISO 9241[ISO9241] beschrieben Richtlinien wurden bei der Entwicklung des ersten Prototypen nicht beachtet. Dementsprechend mussten verschiedene Eigenschaften der GUI verändert werden. Eine ausführliche Beschreibung findet sich in [JE2006].

Internationalisierung

Im ersten Prototyp wurde die Internationalisierung nur bedingt unterstützt. Es gab im Installationsverzeichnis eine Java Datei, die eine Übersetzung beinhaltete, der fest einprogrammierte, durchnummerierte Zahlen zugeordnet waren. Diese Zahlen wurden als Referenz für die Übersetzungsschemata verwendet. Eine sinnvolle Erweiterung um andere, zu beginn nicht eingeplante Sprachen wurde dadurch erschwert. Die Internationalisierung des neuen Prototypen wurde gemäß des von Java zur Verfügung gestellten Standards implementiert.

4.5.2 JNLP

Das Java Network Launch Protocol definiert ein plattformunabhängiges Protokoll für die Verteilung, Installation, das Starten und dasUpdaten von Java-Programmen über das Netzwerk. Die Referenzimplementierung zu Sun's JNLP ist Java Web Start.

Die Vorteile von Applikationen gegenüber Applets sind die folgenden:

- Der Benutzer kann sich den Start intuitiver gestalten, er kann wie er es von anderen Anwendungen auch gewöhnt ist, eine Verknüpfung zu dem Programm auf seinen Desktop setzen. Er muss nicht wie bei einem Applet den Browser starten, die Seite aufrufen und das Programm starten.
- Die Klassen müssen nicht bei jedem Start neu geladen werden, wodurch das Programm schneller starten kann.
- Die Applikationen kann auch offline gestartet werden. Allerdings kommt diese Konstellation in einem komplett aufgesetzten unID-System eher selten vor. Einzige Ausnahme, die Mittelschicht und die grafische Benutzer-Schnittstelle laufen auf dem gleichen Rechner.
- Bei der Realisierung kann eine normale Applikation geschrieben werden, es müssen keine Konventionen für Applets berücksichtigt werden.

Im Gegensatz zu Applets die jedesmal neu geladen werden, werden bei Java WebStart nur die Daten neugeladen die seit dem letzten Start verändert worden sind. Dies ermöglicht auch einen schnelleren Start. Desweiteren sind Applets nicht für zu große Programmkonstrukte gedacht.

Folgende Voraussetzungen müssen getroffen werden, damit eine Applikation als Java Web Start angeboten werden kann:

- Das Programm muss öffentlich auf einem Server zur Verfügung gestellt werden.

- Der Webserver muss das JNLP-Protokoll unterstützen.
- Auf dem Rechner des Anwenders müssen, sowohl eine JRE als auch Java Web Start installiert sein.
- Der Kern der JNLP-Technik sind die JNLP-Dateien. Es müssen eine oder eventuell mehrere JNLP-Dateien erstellt werden, wobei eine dieser Dateien die Startdatei sein muss.
- Sämtliche Programmteile müssen in ein oder mehrere JARs verpackt werden.

- Alle von der Applikation verwendeten JARs müssen signiert werden. Dies ist nötig um ein Programm, das bestimmte Rechte benötigt, für diese freizuschalten. Dazu ist als erstes die Erstellung eines Keys nötig "`keytool -genkey -alias unIDS2'". Die JARs werden dann mit Hilfe des Befehls "`jarsigner unIDS-GUI.jar unIDS2'" signiert.

4.5.3 Grundsätze der Dialoggestaltung (Teil 10)

- Aufgabenangemessenheit
  Ein Produkt ist aufgabenangemessen, wenn der Benutzer mit Hilfe des Produkts schnell und zuverlässig seine Aufgabe erfüllen kann. So sollten unnötige Eingaben durch ihn vermieden werden. Der Benutzer sollte nicht unnötig belastet werden, um so mentale und psychische Überlastungen zu vermeiden. Zusätzlich sollte das System den Benutzer bei der Arbeit unterstützen und ihm hilfreich zur Seite stehen.
- Selbstbeschreibungsfähigkeit
  Im Sinne der Selbstbeschreibungsfähigkeit sollte unmittelbar ersichtlich sein, welche Möglichkeiten bestehen, den Dialog mit einem System fortzusetzen. Es sollte also ohne auszuprobieren erkennbar sein, wie zu handeln ist, um in einem Programm zu navigieren. Hierzu müssen die einzelnen Navigationselemente, aber auch Rückmeldungen des Systems unmittelbar erkennbar und verständlich sein.
- Steuerbarkeit
  Das Dialogsystem sollte dem Benutzer Möglichkeiten der Interaktion bieten, so dass er jederzeit den Ablauf bestimmen kann. Zusätzlich sollte das Dialogsystem auch die Arbeitsgeschwindigkeit des Benutzers berücksichtigen.
- Erwartungskonformität
  Ein Dialog sollte bei gleicher Eingabe immer auf die gleiche Weise reagieren. Außerdem sollte er an den Kenntnisstand des Benutzers angepasst sein.
- Fehlertoleranz
  Bei fehlerhafter Eingabe sollte der Benutzer durch das Dialogsystem auf seinen Fehler aufmerksam gemacht werden, so dass er ihn korrigieren kann.
- Individualisierbarkeit
  Der Benutzer sollte das Dialogsystem an seine Anforderungen anpassen können, so dass seine individuellen Fähigkeiten besser genutzt werden können.
- Lernförderlichkeit
  Das Dialogsystem sollte dem Benutzer Hilfsmittel zum Erlernen von Abläufen zur Verfügung stellen. So wird der Benutzer beim Arbeiten durch das System geschult.

4.5.4 Gebrauchstauglichkeit (Teil 11)

Die ISO 9241[ISO9241] definiert Gebrauchstauglichkeit als das

„Ausmaß, in dem ein Produkt durch bestimmte Benutzer in einem bestimmten Nutzungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und zufriedenstellend zu erreichen.“

Effektivität bedeutet, dass die Software den Benutzer beim Erledigen seiner Aufgaben unterstützt, und die Ergebnisse korrekt sind. Die Effizienz gibt den Aufwand an, den der Benutzer zum

Erreichen seiner Arbeitsziele betreiben muss.

Die Zufriedenstellung des Benutzers wird erreicht, wenn er die Aufgabe möglichst ohne Beeinträchtigung durch das Produkt durchführen kann, und der Benutzer das Produkt gerne benutzt.

Ein Software-Produkt ist also gebrauchstauglich, wenn der Benutzer bei seinen Aufgaben durch das System unterstützt wird. Ist bei der Entwicklung eines Produkts auf benötigte Funktionalität verzichtet worden und der Benutzer braucht zur Erledigung seiner Arbeitsaufgabe länger, dann ist ein Produkt nicht ergonomisch.

4.5.5 Informationsdarstellung (Teil 12)

Entsprechend der sieben Grundsätze der Dialoggestaltung existieren auch sieben Grundsätze der statischen Informationsdarstellung. Diese werden im zwölften Teil der ISO 9241 vorgestellt. Die folgenden charakteristischen Eigenschaften sollten beachtet werden, um den Benutzer in die Lage zu versetzen, Informationen effektiv, effizient und mit Zufriedenheit aufzunehmen.

- Klarheit
  der Informationsinhalt wird schnell und genau vermittelt
- Unterscheidbarkeit
  die angezeigte Information kann genau unterschieden werden
- Kompaktheit
  den Benutzern wird nur jene Information gegeben, die für das Erledigen der Aufgabe notwendig ist
- Konsistenz
  gleiche Information wird innerhalb der Anwendung entsprechend den Erwartungen des Benutzers stets auf gleiche Art dargestellt
- Erkennbarkeit
  die Aufmerksamkeit des Benutzers wird zur benötigten Information gelenkt
- Lesbarkeit
  die Information ist leicht zu lesen
- Verständlichkeit
  die Bedeutung ist leicht verständlich, eindeutig, interpretierbar und erkennbar

4.5.6 Benutzerführung (Teil 13)

Die Benutzerführung soll zu einer effizienteren Benutzung des Systems führen, dazu soll eine übermäßige psychische Arbeitsbelastung vermieden werden. Um dies zu erreichen, sollen dem Benutzer zusätzliche Informationen auf Anfrage zur Verfügung gestellt werden. Dies kann ein Handbuch sein, aber auch eine Online-Hilfe, die der Benutzer jederzeit aus dem Programm heraus aufrufen kann.

Ein weitere Möglichkeit der Benutzerführung liefert das Fehlermanagement, das den Benutzer bei der Entdeckung, Erklärung und Behebung von Fehlern. Dies soll zum Beispiel verhindern, das der Benutzer in Situationen gelangt, aus denen er ohne fremde Hilfe nicht mehr herauskommen kann.

4.5.7 Dialogführung (Teil 14-17)

Um in Interaktion mit den Benutzer zu treten, stellt die ISO 9241 vier unterschiedliche Arten der Dialogführung vor. Diese können je nach Kenntnis- und Erfahrungsstand des Benutzers zur Anwendung kommen.

Dialogführung mittels Menüs

Dem Benutzer wird die Steuerung des Programms mit Hilfe von Menüs erleichtert. Das können

sowohl die bekannten Menüs im oberen Bereich eines Fensters (Pull-Down-Menüs) als auch aufklappbare Menüs (Pop-Up-Menüs) sein. Bei dieser Art der Dialogführung werden dem Benutzer eine oder mehrere Optionen zu Verfügung gestellt, aus denen er sich eine auswählen kann.

Dialogführung mittels Kommandosprache

Die Dialogführung mittels Kommandosprache gestattet es dem Benutzer vollständige oder abgekürzte Kommandos einzugeben. Diese Kommandos entsprechen der Syntax der zugehörigen Kommandosprache. Nach der Eingabe eines Kommandos mit den entsprechenden Parametern führt der Computer diese aus.

Dialogführung mittels direkter Manipulation

Wird die reale Welt auf dem Rechner abgebildet, spricht man von Dialogführung mittels direkter Manipulation. Ein Beispiel hierfür ist die Desktop-Oberfläche moderner Rechnersysteme, der Desktop enthält die gleichen Elemente wie ein Arbeitsplatz (zum Beispiel Papierkorb, Aktenordner). Die angezeigten Objekte können wie in der realen Welt behandelt werden.

Dialogführung mittels Bildschirmformularen

In Bildschirmformularen wird dem Benutzer ermöglicht Eintragungen vorzunehmen, wie bei realen Dokumenten. Er kann Felder auswählen und die dort stehenden Daten verändern.

5 Zusammenfassung

Der zweite Prototyp weist eine klare Struktur auf, die einzelnen Bereiche sind deutlich voneinander getrennt und weisen keine unnötigen Verstrickungen im Bereich der GUI mehr auf. Die XML Kommunikation wurde von der Implementierung anderer Bereiche getrennt. Es wird in den meisten Fällen keine Überschreibung der Daten mehr vollzogen sondern eine direkte Auswertung getätigt.

Die Internationalisierung wurde verbessert. Die Kommunikation auf Netzwerkebene wurde durch den BEEP-Standard ersetzt und ermöglicht so ein leichteres Verständnis der Kommunikationsprozesse. Das Plugin-System wurde ergonomischer gestaltet, die Schnittstelle wurde verkleinert und allgemeiner gehalten. Dadurch können nahezu alle Bereiche durch Plugins erweitert werden. Außerdem melden sich die Plugins nun automatisch an, sofern sie im Installationsverzeichnis stehen.

Das Sammeln von Systeminformationen wird durch ein effektives Zusammenspiel von Oberfläche und Mittelschicht realisiert. Informationen kommen in Echtzeit zum Benutzer und haben keine Latenzzeit mehr durch das Pollingverfahren. Die Anzeige der Informationen selbst wurde durch ein bewährtes Paket zur Visualisierung von statistischen Daten ersetzt. Damit ergeben sich eine Vielzahl von neuen Möglichkeiten der Darstellung.

Das Aufgabenspektrum der Mittelschicht hat sich im Laufe der Entwicklung des unIDS-Systems als wesentlich komplexer herausgestellt als zunächst bei der Planung des unIDS-Projekts angenommen. Positiv zu bewerten ist, dass sich das Architekturmodell, welches letztendlich entwickelt wurde, bewährt hat. Es bietet komfortable Möglichkeiten um die Mittelschicht modular zu erweitern. Über den Rand der Mittelschicht hinaus hat sich zwischen den verschiedenen unIDS-Komponenten ein sehr flexibles und gut erweiterbares Nachrichtenmodell etabliert.

Insgesamt bildet die Mittelschicht eine robuste Basis, die im Vergleich zur alten unIDS-Version wesentlich stabiler, aufgeräumter und klarer ist, auch wenn einige der geplanten Eigenschaften mit den vorhandenen Mitteln nicht realisiert werden konnten. Somit wurde zwar nicht jedes einzelne Ziel erreicht, aber die wesentliche Zielsetzung im Sinne einer Weiter- bzw. Neuentwicklung des unIDS-Systems auch im Bereich der Mittelschicht erfolgreich umgesetzt.

Die in Abschnitt 4.3.1 gesetzten Ziele und Anforderungen für die Agenten konnten erfolgreich umgesetzt werden. In den durchgeführten Tests haben der Agent und die jeweiligen Plugins bereits gezeigt, dass sie in der Lage sind, sowohl Informationen von dem IDS *Snort* zu sammeln als auch

zahlreiche sicherheitsrelevante Informationen über den zu überwachenden Rechner selbst zu generieren. Nach den anfänglichen Problemen bzgl. der Aussendung von BEEP-Nachrichten, konnte letztendlich eine zuverlässige Schnittstelle zur Mittelschicht durch den Einsatz von *BEEPy* und *Twisted* entwickelt werden. Die BEEP-Nachrichten können somit problemlos zwischen der Mittelschicht und dem Agenten übertragen werden. Das Plugin-Konzept erwies sich auch während der Implementierungsphase als sehr vorteilhaft, weil es eine unabhängige Implementierung der einzelnen Plugins durch mehrere Entwickler ermöglichte und eine effektivere und schnellere Entwicklung zur Folge hatte.

5.1 Ausblick

Einige Anforderungen seitens der GUI wurden bisher noch nicht erfüllt. So konnte die ursprüngliche 2D Ansicht nicht in einen 2D Editor umgewandelt werden. In diesem sollte es möglich sein, interaktiv mit einem visuellen Feedback ein Netzwerk zu modellieren. Weitere Ansichten, die eine große Menge sicherheitsrelevanter Daten leicht verständlich darstellen, wurden ebenfalls bisher nicht geplant und entworfen.

Bezüglich der Mittelschicht sind konzeptionell zwei wesentliche Punkte weiter zu entwickeln. Dies ist zunächst die Einbindung von vollständigen Proxys in das unIDS-Modell. Hierzu fehlen neben der dann geänderten Aufgabenverteilung zwischen Agent, Proxy und Mittelschicht auch die dazu nötigen Nachrichtenklassen. Strukturell ist das Datenmodell der Mittelschicht bereits so ausgelegt, dass auch Proxys mit untergebracht werden könnten.

Weiterhin ist der Aufgabenbereich der Analyse in weiten Bereichen noch offen. Benötigt wird eine klare Zielsetzung auf der aufbauend die erforderlichen Eigenschaften der Mittelschicht definiert und realisiert werden können.

Abschließend bleibt noch ein Blick auf die Unterschicht von unIDS. Es wurde je ein Agent für das Syslog eines Rechners und für Snort entwickelt. Für den sinnvollen Einsatz ist allerdings ein breiteres Spektrum von Agenten notwendig. Ebenso steht eine Weiterentwicklung der Proxys bezüglich Datenfilterung und Unterstützung der Skalierbarkeit des Systems noch aus.

6 Anhang

6.1 Anleitung zur Schnellinstallation

Dieser Abschnitt beschreibt die einfache Installation von unIDS. Eine ausführliche Anleitung zur Installation, Konfiguration und Benutzung befindet sich im unIDS Benutzerhandbuch[PG0405-2].

Die Funktion der vollständigen Installation wurde auf folgendem System verifiziert:

- Debian Installation der Knoppix CD 4.0 mit den zur Distribution gehörenden Paketen
    - MySQL
    - kaffe 1.1.6 (java)
    - ant
    - python 2.3

Getestet wurde der current Zweig der unIDS CD[15]

Zusätzlich wurden die Programme

- elementtree-1.2.6
- TwistedSumo
- ZODB3-3.4.1
- ZopeInterface-3.0.1
- pygraphlib-0.6.0.1

aus dem current Unterverzeichnis „lib/dependencies“ mit der jeweiligen Setup Routine installiert.[16] Die Werkzeuge aus dem Unterverzeichnis „lib/src“ wurden in das Python Verzeichnis

```
/usr/lib/python2.3/site-packages
```

kopiert.

Die eigentliche unIDS Installation erfolgt aus dem „current“ Verzeichnis. Hier sollten folgende Dateien vorliegen:

```
agent
beepy-agent_setup.py
gui
middlelayer
proxy
agent_setup.py
ez_setup.py
lib
middlelayer_setup.py
proxy_setup.py
setup.py
```

Mit

```
python setup.py install --prefix=${HOME}/unids_tmp
```

wird die Installation initiiert. Für alle weiteren gewünschten Komponenten, z.B. der Mittelschicht, muss die jeweilige Installationsroutine explizit aufgerufen werden. Es ist erlaubt, alle Komponenten auf einem Rechner zu installieren.

Wurde wie oben ein Prefix verwendet, muss der Pythonpfad angepasst werden. Dies solle in der

15 Backup unter joule:/home/archiv/projektgruppen/Projektgruppe_IDM/Teil2/software/current.tbz (Interner Vermerk!)

16 python setup.py install

Regel wie folgt aussehen:

```
export PYTHONPATH=${HOME}/unids_tmp/lib/python2.4/site-packages
```

Unter Umständen ist ein komplexerer Ausdruck erforderlich. Wenn z.B. eine amd64 - Architektur verwendet wird, muss vermutlich folgendes verwendet werden:

```
export  PYTHONPATH=${HOME}/unids_tmp/lib/python2.4/site-packages:${HOME}/
unids_tmp/lib64/python2.4/site-packages
```

Gegebenenfalls einfach nachsehen, welche Unterverzeichnisse vorhanden sind.

Jetzt müssen einige Konfigurationsverzeichnisse erstellt werden. Prinzipiell können sie an beliebiger Stelle erzeugt werden, der Übersicht halber sollten sie aber leicht zu finden sein, z.B. in dem oben angelegten Verzeichnis „unids_tmp“.

Hier wird jetzt für jede Komponente (Mittelschicht, Agent usw.) ein Verzeichnis angelegt.

`mkdir middlelayer`
`mkdir agent` usw.

In das jeweilige Verzeichnis wechseln und dort entsprechend den Befehl

```
mktap unids.middlelayer -c ${HOME}/unids_tmp/etc/unids/middlelayer.xconf
```

ausführen. (middlelayer entsprechend durch agent ersetzen.)

Das gleiche gilt für das Starten der Komponenten, welches mittels

```
twisd -nof unids.middlelayer
```

geschieht.

Die GUI lässt sich anschließend aus dem Verzeichnis „gui“ mit

```
ant run
```

starten. Der voreingestellte Zugang lautet admin/test.

6.2 Offene Merkmale

Die hier zitierte Liste spiegelt den Stand vom 03.10.2005 der technischen Referenz[PG0405] wider. Eine neuere Liste ist u.a. auf der unIDS CD zu finden.

- Antworten der grafischen Schnittstelle verarbeiten
  Antworten der GUI werden von der Mittelschicht bisher nicht verarbeitet. Ein möglicher Ansatz wäre eine Art Event-Log in der Datenbank.
- Maximalwerte für die Statusanzeige
  Sollen Angaben, z.B. der Speicher- oder Prozessorauslastung prozentual dargestellt werden, muss zunächst das absolute Maximum dieser Werte bestimmt werden. Hierfür stehen noch keine allgemeingültigen Regelungen zur Verfügung.
- Schema der Benutzerverwaltung
  Die Benutzerverwaltung ist bisher nur rudimentär implementiert.
- Verbesserte Auswertung der IDMEF-Nachrichten
  Die Nachrichten könnten nach bekannten Angriffen durchsucht und ausgewertet werden. Aus diesen Daten ließen sich Statistiken erstellen, die für jedes Gerät Art und Häufigkeit der Angriffe visualisieren. Hierfür liegt bisher kein Konzept vor.
- Zugriffsberechtigungen für alle Entitäten entwerfen
  Bisher existiert kein Zugriffsschutz für die einzelnen Entitäten des Systems, dieser ist aus Sicherheitsgründen aber notwendig. Ein möglicher Ansatz wäre eine rollenbasierte Lösung.
- Nachrichten exakt auf aktuellen Benutzer anpassen
  In der Nachricht über die Netzwerkstruktur (Subnetze, Geräte usw.) werden bisher alle vorhandenen Objekte übermittelt, unabhängig davon, ob der aktuelle Benutzer diese sehen darf oder nicht. Die Nachricht muss so zusammengestellt werden, dass nur noch die Objekte enthalten sind, die der aktuell angemeldete Benutzer sehen darf. Weiterhin ist festzulegen, wie mit Fällen umgegangen wird, in denen z.B. ein Subnetz nicht sichtbar ist, aber einzelne der darin enthaltenen Geräte für den entsprechenden Benutzer freigegeben sind.
- Netzwerkschnittstellen den Geräten zuordnen
  Die Netzwerkstruktur wird um Schnittstellen erweitert, das heißt, ein Gerät kann mehrere Schnittstellen besitzen. Diese sind in der Strukturnachricht unterzubringen und für die GUI verfügbar zu machen. Hierzu muss der Agent zusätzliche Daten über die Hardware und deren Konfiguration liefern, damit die Mittelschicht darüber informiert ist, welche Netzwerkschnittstellen auf einem überwachten System zur Verfügung stehen.
- Beep Kanal Autostart
  Die Mittelschicht wird so erweitert, dass sie versucht, falls kein passender Kanal zur Verfügung steht, diesen selbst zu öffnen und erst nach einem erfolglosen Versuch der Nachrichtenversand scheitert.

Literaturverzeichnis

ISO9241: DIN EN ISO 9241-X: Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten
JE2006: Jens Eschke, Entwicklung einer grafischen Benutzer-Schnittstelle für das Intrusion Management System unIDS, Rechnernetze und Telekommunikation, Universität Oldenburg,2006
JS05: Jan Schoof, Untersuchung und Entwicklung von IDM Komponenten im Rahmen des unIDS Projektes, Rechnernetze und Telekommunikation, Universität Oldenburg,2005
PG0304: Projektgruppe 03/04, unIDS, Implementierung eines Intrusion Management Systems, Rechnernetze und Telekommunikation, Universität Oldenburg,2004
PG0405: Projektgruppe 04/05, unIDS - Technische Referenz, Rechnernetze und Telekommunikation, Universität Oldenburg,2005
PG0405-2: Projektgruppe 04/05, unIDS Benutzerhandbuch, Rechnernetze und Telekommunikation, Universität Oldenburg,2005