

Verteilte Angriffserkennung

Christoph Riesinger

26. Januar 2007

1 Einleitung und Überblick

1.1 Einleitung

Die Wurmattaken der letzten Jahre haben gezeigt, dass vom Einzelanwender bis hin zu großen Anbietern von Netzwerkdiensten jeder durch Gefahren aus dem Internet bedroht ist. Je mehr vertrauliche Daten über das globale Netz ausgetauscht werden, desto größer ist auch die Gefahr, dass sich Unbefugte Zugriff auf diese verschaffen wollen. Ein Schutz vor derart böswilligen Aktivitäten ist also die logische Konsequenz.

Darüber hinaus ist es immer das Ziel von Diensten, hochverfügbar zu sein, unabhängig von der Anzahl der Klienten, die sie bedienen. Dies ist bei Netzwerkdiensten natürlich nicht anders.

Um heutzutage größtmöglichen Schutz vor böartigen Aktivitäten aus dem Internet zu bieten, werden komplexe und ausgeklügelte Sicherheitssysteme eingesetzt. Teil dieser Sicherheitssysteme sind sogenannte *Intrusion Detection Systeme* (kurz: IDS). Die Aufgabe solcher IDS besteht darin, Zugriffe auf Computersysteme oder Computernetze zu analysieren und zu entscheiden, ob es sich um reguläre Anfragen oder um böswillige Zugriffe handelt. Man spricht hierbei von Angriffserkennung.

Natürlich müssen solche IDS rund um die Uhr funktionsfähig sein, denn sobald diese ausfallen, entfällt eine wichtige Komponente des Sicherheitssystems. Sie müssen Netzwerke und Angebote überwachen, auf die Millionen von Nutzern mit Milliarden von Anfragen zugreifen können. Mit dieser Herausforderung muss ein IDS zurecht kommen, trotzdem zuverlässig arbeiten und noch dazu möglichst ressourcenschonend sein. Klassische, zentral lokalisierte Angriffserkennung, wie sie aktuell zum großen Teil eingesetzt wird, stößt hierbei schnell an ihre Belastungsgrenzen.

1.2 Überblick

Ein verteilter Ansatz der Angriffserkennung kann eine Lösung für dieses Problem sein. Die Angriffserkennung findet nicht zentral an einem Punkt statt, sondern verteilt sich auf viele einzelne Punkte. Diejenigen, die sich an solch

einer verteilten Angriffserkennung beteiligen, werden im Folgenden als Teilnehmer bezeichnet. Diese Teilnehmer können Prozesse auf einem einzelnen Rechner sein, mehrere Rechner in einem Netzwerk oder eine Mischform aus beidem. Dieser Text beschäftigt sich mit verteilter Angriffserkennung. Er beleuchtet die Nachteile bestehender, zentralisierter Angriffserkennung und zeigt Lösungen durch verteilte Ansätze auf. Zwei Ideen von verteilter Angriffserkennung werden vorgestellt. Zuerst präsentiert dieser Text eine hierarchisch organisierte Idee unter Verwendung von Software-Agenten. Anschließend wird eine dezentral organisierte Idee vorgestellt, welche auf *Peer-to-Peer* (kurz: P2P) Netzwerken aufbaut. Dabei liegt immer ein besonderes Augenmerk auf den Lösungen für diejenigen Probleme, die durch verteilte Angriffserkennung entstehen. Solche verteilten IDS werden auch als *Distributed Intrusion Detection Systeme* (kurz: DIDS) bezeichnet.

2 Probleme und Lösungen zentraler Angriffserkennung

In vielen bestehenden IDS übernimmt eine einzige Komponente die Angriffserkennung. Diese ist folglich zentral lokalisiert. Zwar gibt es Varianten, in denen die Datenerfassung dezentral, also an mehreren Orten geschieht, die Auswertung der gesammelten Daten findet jedoch weiterhin zentral statt.

2.1 Probleme klassischer, monolithischer Angriffserkennung

Nun werden die wesentlichen Nachteile zentraler Angriffserkennung aufgeführt.

2.1.1 Single Point of Failure

Ist die komplette Algorithmik zur Angriffserkennung in einem einzigen Punkt konzentriert, führt ein Ausfall oder die Manipulation dieses einzelnen Punktes dazu, dass die Angriffserkennung nicht mehr zuverlässig arbeitet oder komplett ausfällt. Für das zu schützende System bedeutet dies, dass man keinerlei Aussagen über dessen Integrität mehr machen kann. Im schlimmsten Fall ist sogar ein Totalausfall des Systems die Folge. Einem Angreifer wird es besonders einfach gemacht, denn für ihn reicht es aus, eine einzige Komponente zu kompromittieren um die komplette Angriffserkennung auszuhebeln.

2.1.2 Schlechte Skalierbarkeit

Einem klassischen IDS stehen nur die Ressourcen zur Verfügung, welche lokal vorhanden sind. Jedem ist bekannt, dass sich in einem solchen Szenario die Leistungsfähigkeit nicht unbegrenzt steigern lässt. Auch kann sich jeder vorstellen, dass auf diese Weise der Verwaltungsaufwand für die Ressourcen überproportional rasch ansteigt. Doppelter Speicher und doppelte Rechenleistung führen also nicht unbedingt zu doppelter Effizienz bei der Angriffserken-

nung. Skalierbarkeit ist folglich nicht gewährleistet, was der in der Regel steigenden Benutzerzahl eines Dienstes nicht Rechnung trägt.

2.1.3 Mangelnde Konfigurier- und Wartbarkeit

Wird die Konfiguration eines IDS geändert, steht dieses für die Zeit der Änderung im Allgemeinen nicht zur Verfügung. Oft muss es neu gestartet werden, damit neue Einstellungen übernommen werden. Selbiges gilt auch für die Wartbarkeit, denn bei einer Aktualisierung der IDS-Software werden typischerweise Dateien ausgetauscht. Kommt es dabei zu Komplikationen, arbeitet das IDS nur noch unzureichend oder überhaupt nicht mehr.

2.2 Vorteile verteilter Angriffserkennung

Alle der oben genannten Probleme lassen sich durch verteilte Angriffserkennung lösen. Außerdem bietet die verteilte Angriffserkennung zusätzliche Vorteile, die nun vorgestellt werden.

2.2.1 Höhere Ausfallsicherheit

Bei einem DIDS geht man davon aus, dass die Angriffserkennung an vielen verteilten Stellen durchgeführt wird. Fällt eine dieser Erkennungsstellen aus, beispielsweise verursacht durch einen Softwarefehler, einen Angriff oder Überlastung, hat das auf das Gesamtsystem der Angriffserkennung nur wenig bis gar keine Auswirkungen. Die noch funktionsfähigen Teilnehmer am DIDS können ihre Aufgabe weiterhin einwandfrei erfüllen. Man kann sogar soweit gehen, dass die noch funktionsfähigen Teilnehmer diejenigen Aufgaben erfüllen, welche bisher vom jetzt ausgefallenen Teilnehmer erledigt wurden.

2.2.2 Bessere Skalierbarkeit

Mehrere Teilnehmer in einem DIDS bedeuten auch erhöhte Rechenleistung. Schon das Prinzip des *Divide & Conquer* besagt, ein großes Gesamtproblem in mehrere kleine Teilprobleme aufzuteilen. Diese Teilprobleme können nun leichter gelöst und zur Gesamtlösung für das Ausgangsproblem zusammengefügt werden. Bei einem verteilten Ansatz können die Teilprobleme auf die Teilnehmer des Systems verteilt werden, welche diese dann lösen. Darüber hinaus ist es wesentlich einfacher, mehrere Rechner in einem Verbund zusammen zu schalten, als ein monolithisches System immer weiter aufzurüsten.

2.2.3 Modularität erhöht Wartbarkeit

Da nun mehrere Teilnehmer die Angriffserkennung durchführen, können Aktualisierungsarbeiten bei den einzelnen Teilnehmern durchgeführt werden, ohne die Grundfunktionalität des gesamten DIDS zu gefährden. Tritt bei der Aktualisierung eines Teilnehmers ein Fehler auf, kann der Rest des DIDS uneingeschränkt weiterarbeiten, da er kaum vom ausgefallenen Teilnehmer abhängt. Auf diese

Weise lässt sich Schritt für Schritt ein DIDS aktualisieren, ohne eine einzige Sekunde der Nichtverfügbarkeit des DIDS. Selbiges gilt natürlich auch für die Konfigurierbarkeit des Systems.

2.2.4 Flexible Arbeitsteilung und Heterogenität

Der verteilte Ansatz bietet nun die Möglichkeit, jedem Teilnehmer genau die Aufgaben zuzuteilen, die er auch im Stande ist zu lösen. So können leistungsfähigere Knoten mehrere und komplexere Teilaufgaben übernehmen als weniger leistungsfähige. Heterogenität des Gesamtsystems ist die Folge, welche es Angreifern erschwert, Sicherheitslücken zu finden, die bei jedem Teilnehmer ausgenutzt werden können.

Ein weiteres Beispiel dieser Heterogenität ist die Tatsache, dass es keinerlei Festlegung gibt, mit welcher Programmiersprache die Funktionalität eines Teilnehmers realisiert wird. Lediglich Schnittstellen für die Kommunikation müssen erfüllt werden.

2.2.5 Umfangreicheres Analysematerial

Da die gesammelten Informationen nun von verschiedenen Teilnehmern stammen, können nun auch Arten von Bedrohungen erkannt werden, die sonst überhaupt nicht oder nur sehr schwer detektiert worden wären. Als beispielsweise der Computerwurm Slammer [14] im Sommer 2003 eine Schwachstelle im SQL-Server von Microsoft [16] ausnutzte um diesen zum Absturz zu bringen, fluteten von dem Wurm infizierte Zombierechner das Internet mit präparierten Datenpaketen. Sobald ein MSSQL-Server eines dieser Pakete empfangt, stürzt dieser ab. Eine solch große Zahl von ungewöhnlichen Paketen konnte während des Ausbruchs von vielen IDS im Internet beobachtet werden. Fügt man nun diese Information von mehreren Teilnehmern eines DIDS zusammen, kann man viel präziser die Arbeitsweise des Slammerwurms erkennen und effektivere Gegenmaßnahmen ergreifen.

2.3 Herausforderungen verteilter Angriffserkennung

Das höhere Maß an Flexibilität und der Wegfall einer zentralen Kontrollinstanz bei verteilter Angriffserkennung ist mit neuen Herausforderungen verknüpft, die man von klassischen Ansätzen her nicht kennt. In den später vorgestellten Projekten verteilter Angriffserkennung wird explizit auf die dortige Behandlung eben dieser Herausforderungen eingegangen.

2.3.1 Kommunikation

Bei zentraler Angriffserkennung müssen Informationen höchstens zwischen Prozessen oder Threads ausgetauscht werden, sobald das zu analysierende Material vorhanden ist. Verteilte Angriffserkennung erfordert nun eine wesentlich komplexere Regelung der Kommunikation. Aufgrund der Heterogenität kann es vorkommen, dass Informationen innerhalb eines Rechners aber auch innerhalb eines

Netzwerks ausgetauscht werden müssen. Eine solche Kommunikation führt zu einem erhöhten Datenaufkommen.

2.3.2 Authentifizierung von DIDS-Partnern

Vertrauen und Integrität spielen in allen Systemen, bei denen es mehrere Teilnehmer gibt, eine wichtige Rolle. Dies trifft vor allem bei einem DIDS zu. Es ist äußerst wichtig sicher zu stellen, dass alle Teilnehmer vertrauenswürdig sind. Befindet sich ein Teilnehmer im Verbund, der sich unter Kontrolle eines Angreifers befindet, könnte dieser Teilnehmer Fehlinformationen so im System verteilen, dass die von den Administratoren gewünschte Angriffserkennung im DIDS nicht mehr korrekt ausgeführt werden kann.

2.3.3 Auslösen von Alarmen

Wird ein Angriff erkannt, so muss dies auch bekannt gemacht werden, damit eventuelle Gegenmaßnahmen eingeleitet werden können. Ein IDS löst einen solchen Alarm aus, sobald ein gewisser Schwellwert überschritten wird. Bei einem verteilten System stehen jedoch in der Regel nicht allen Teilnehmern die gleichen Datensätze zur Analyse zur Verfügung. Darüber hinaus können die Schwellwerte für das Auslösen eines Alarms aufgrund der Heterogenität von den Teilnehmern unterschiedlich gewählt sein. Dies führt nun zu der Frage, wann und wo ein Alarm für das Gesamtsystem ausgelöst werden soll.

2.3.4 Analyse der Effizienz

Hauptaufgabe sowohl eines IDS als auch eines DIDS ist die korrekte Klassifizierung von Angriffen. Dabei sollen die Anzahl der Fehler erster Art (ein Zugriff wird als böswillig klassifiziert, obwohl dieser zulässig ist) und die Fehler zweiter Art (ein Zugriff wird nicht als Angriff klassifiziert, obwohl es sich nicht um einen Angriff handelt) so gering wie möglich gehalten werden. Eine Analyse der Qualität der Angriffserkennung ist deshalb unverzichtbar. Ist dies bei zentraler Angriffserkennung mit einem gewissen Aufwand noch gut realisierbar, stellt uns das bei verteilter Angriffserkennung vor ein großes Problem.

3 Hierarchische Idee verteilter Angriffserkennung

In diesem Abschnitt wird nun eine hierarchisch organisierte Idee eines verteilten Angriffserkennungssystems vorgestellt. Darin übernehmen sogenannte Software-Agenten verschiedene Aufgaben wie beispielsweise die Angriffserkennung oder das Verwalten der Teilnehmer. Zwei später in diesem Kapitel vorgestellte Konzepte (AAFID [2] und DOMINO [3]) greifen die Idee eines Software-Agenten gestützten, hierarchisch organisierten DIDS auf.

3.1 Software-Agenten

Unter einem *Software-Agenten* [10] in der Informatik versteht man ganz allgemein ein Softwareprogramm, welches eine oder mehrere der folgenden Eigenschaften erfüllt. Das Programm...

- ...läuft, soweit möglich, ohne externe Steuerung durch einen Benutzer oder ein anderes Programm, ist also **autonom**.
- ...reagiert auf Änderungen in seiner Umwelt, ist also **reaktiv**.
- ...trifft Vorkehrungen auf Ereignisse, bevor diese geschehen, ist also **proaktiv**.
- ...tauscht Daten mit anderen Software-Agenten aus, ist also **sozial**.
- ...passt sich basierend auf zuvor getroffenen Entscheidungen an, ist also **lernfähig**.

Dabei können ein oder mehrere Software-Agenten auf einem konkreten Rechner laufen, in der Praxis jedoch sind sie auf mehrere Rechner in einem Netzwerk verteilt. Sie sind zumeist als Prozesse oder Threads implementiert.

Speziell auf das Szenario der verteilten Angriffserkennung angewandt bedeutet dies, dass jeder Software-Agent im Wesentlichen für sich selbst arbeitet und nicht zwingend von anderen abhängig ist. Dies führt zu der gewünschten Robustheit eines DIDS. Zusätzlich tauschen die Komponenten eines DIDS bisher noch nicht konkret definierte Information aus, was der sozialen Komponente eines Software-Agenten entspricht. Die Reaktivität eines Software-Agenten in einem DIDS zeigt sich in der Analyse von Zugriffen, sobald diese erfolgen und eine eventuelle Reaktion auf eine Bedrohung, wie beispielsweise das Auslösen eines Alarms.

3.2 Hierarchische Organisation

Hierarchische Organisation bedeutet, dass Ergebnisse, die auf unterster Ebene gewonnen werden an eine höhere Instanz zur weiteren Analyse weitergereicht werden. Dies kann über mehrere Ebenen erfolgen, so dass Teilauswertungen wieder an höhere Instanzen weitergeleitet werden bis hin zu einer obersten Ebene, die dann ein Gesamtergebnis liefert. Die Teilnehmer aller Ebenen können durch Software-Agenten implementiert werden.

Auf diese Weise entsteht eine Baumstruktur, wie man sie oft in der Informatik findet. Die Wurzel dieses Baums stellt der Software-Agent der höchsten Ebene dar, der ein Gesamtergebnis liefert. Die Blätter sind diejenigen Software-Agenten, die Zugriffe auf das System aufzeichnen und auswerten. Grafik 1 soll dies veranschaulichen.

Eine nicht ganz so starre Interpretation einer hierarchischen Organisation erlaubt, dass ein Teilnehmer am DIDS seine Ergebnisse an nicht nur einen einzigen Software-Agenten in einer höheren Ebene übermitteln muss, sondern diese durchaus an mehrere höhergelegene Software-Agenten weitergeben kann.

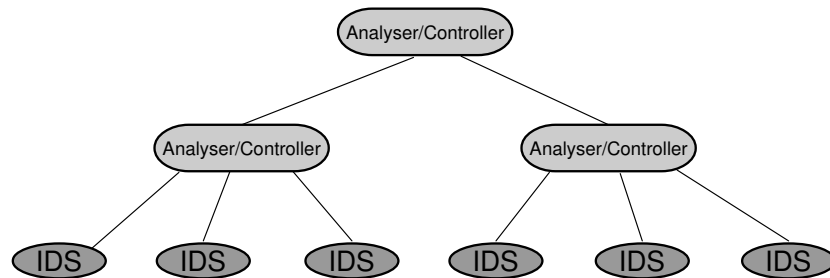


Abbildung 1: Diese Grafik zeigt eine abstrakte hierarchische Struktur eines DIDS. [2]

Bei einer solch hierarchischen Struktur hat man aber nun wieder einen Single Point of Failure erzeugt, nämlich die Wurzel der Hierarchie. Fällt dieser aus, gibt es keine Instanz mehr, die ein Gesamtergebnis bestimmt und die Teilergebnisse der anderen Teilnehmer können wertlos sein. Um dies zu vermeiden hebt man die Limitierung auf, dass sich in einer höheren Ebene weniger Software-Agenten befinden als in einer niedrigeren. Auf diese Weise können mehrere Wurzeln erzeugt werden, die ein aussagekräftiges Gesamtergebnis ermitteln, wie Grafik 2 zeigt.

Dabei handelt es sich nun aber nicht mehr um eine klassische hierarchische Struktur, sondern viel mehr um ein Multi-Level-System, in dem Nachrichten von unten nach oben gereicht werden.

Es bleibt zu sagen, dass eine solch hierarchische Organisation zwar dem Anspruch der Verteilung der Angriffserkennung gerecht wird, jedoch ist diese Art der Strukturierung nicht flexibel sondern starr, da Software-Agenten immer Informationen von unteren Ebenen erhalten, diese verarbeiten und anschließend an höhere Ebenen weiterleiten.

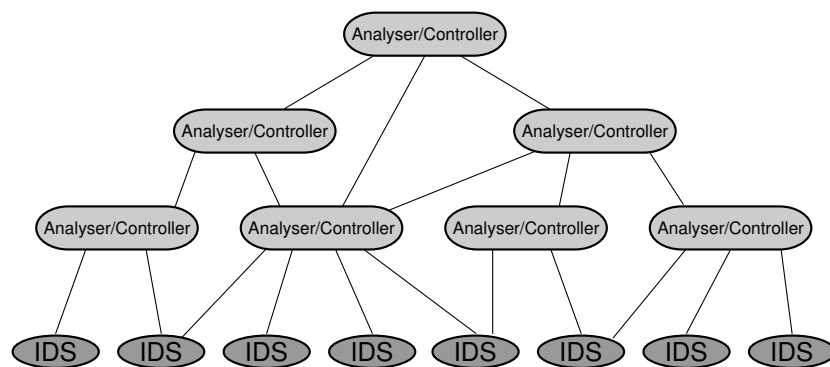


Abbildung 2: Diese Grafik zeigt eine abstrakte hierarchische Struktur eines DIDS bei der es nicht nur eine Wurzel gibt. Darüber hinaus kann jeder Teilnehmer Nachrichten an mehrere Teilnehmer einer höheren Hierarchieebene schicken. [2]

3.3 AAFID: Ein erstes konkretes Konzept eines hierarchisch organisierten DIDS

Bisher wurde nur die Idee erläutert, die hinter dem Ansatz eines hierarchisch organisierten DIDS, basierend auf Software-Agenten, steht. Diese wurde bisher bewusst abstrakt gehalten. Nun aber sollen zwei konkrete Konzepte veranschaulichen, wie ein solch hierarchisch organisiertes DIDS in der Realität aussehen könnte. Dabei wird jeweils kurz der Aufbau des Konzeptes skizziert und dann auf Lösungsansätze der in 2.3 erwähnten Probleme verteilter Angriffserkennung eingegangen.

Als erstes soll *Autonomous Agents for Intrusion Detection* (kurz: AAFID) [2] vorgestellt werden.

3.3.1 Agenten, Transceiver, Monitore

Grafik 3 soll veranschaulichen, wie die Netzwerktopologie von AAFID aussieht. Wie dort zu erkennen ist, besteht die AAFID Architektur aus drei essentiellen Komponenten: *Agenten*, *Transceiver* und *Monitore*. Agenten können auf vielen Rechnern im AAFID System installiert sein. Dabei können auch mehrere Agenten auf einem Rechner ihre Arbeit verrichten. Sie zeichnen Ereignisse auf, die auf dem entsprechenden Rechner geschehen und melden diese an einen einzelnen Transceiver. Transceiver wiederum melden ihre Ergebnisse an Monitore, welche nun einen Überblick über das zu schützende System haben. Die hierarchische Struktur ist klar zu erkennen. Diese drei Komponenten sollen nun genauer beleuchtet werden.

- Im Wesentlichen erfüllt ein Agent im AAFID System genau die Eigenschaften, die auch ganz allgemein ein Software-Agent besitzt. Er ist also reaktiv, sozial und so weiter. Er meldet anomales oder interessantes Verhalten auf dem zu überwachenden Rechner an seinen Transceiver. Die Möglichkeit der Kommunikation mit anderen AAFID Agenten besteht jedoch nicht. Auch das Auslösen eines globalen Alarms bleibt den Transceivern und Monitoren vorbehalten.
- Transceiver stellen die Kommunikationsschnittstelle eines Rechners nach außen dar. Dementsprechend kann ein Rechner zwar mehrere Agenten, jedoch nur einen Transceiver besitzen. Daneben gehört die Kontrolle der Agenten zu den Hauptaufgaben eines Transceivers. Dieser kann Agenten starten oder beenden und erhält die bereits erwähnten Berichte über anomales oder interessantes Verhalten. Letztendlich leitet ein Transceiver seine verarbeiteten Daten weiter, nämlich an die Monitore, von denen jeder Transceiver mehrere beliefern kann.
- Die höchste Ebene der Hierarchie bilden die Monitore. Diese kontrollieren die Transceiver. Hauptunterschied zu den Transceivern ist, dass ein Monitor Kontrolle über mehrere Rechner hat, während ein Transceiver nur die

Agenten eines einzigen Rechners kontrolliert. Deshalb können Monitore Angriffe erkennen, die mehrere Rechner betreffen.

An dieser Stelle sollte noch erwähnt sein, dass ein Agent in AAFID zwar ein Software-Agent ist, wie er anfänglich definiert wurde, ein Software-Agent jedoch allgemeiner betrachtet werden muss. So kann man auch einen Transceiver oder einen Monitor als Software-Agent sehen, denn auch diese sind unter anderem sozial.

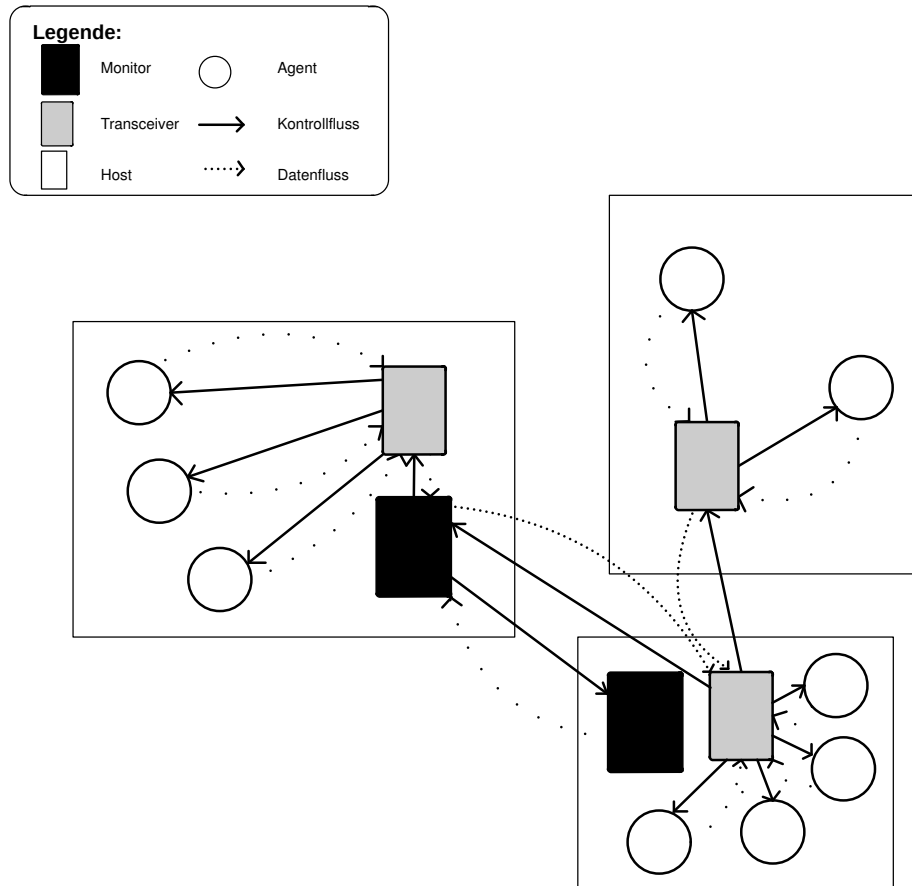


Abbildung 3: Diese Grafik zeigt das physische Layout des AAFID Systems. Die Schlüsselkomponenten Agent, Transceiver und Monitor werden verwendet. Darüber hinaus sind die Kommunikations- und Kontrollflüsse zwischen den Schlüsselkomponenten eingezeichnet. [2]

3.3.2 Kommunikation bei AAFID

Das Problem der Kommunikation bei DIDS wurde bereits angesprochen. AAFID setzt sich mit dieser Problematik auseinander und untersucht verschiedene Lösungsansätze. Zwar wird nicht zwingend vorgegeben, welcher Kommunikationsmechanismus letztendlich zum Einsatz kommen muss, jedoch werden für verschiedene Anforderungen entsprechende Lösungen vorgestellt.

Grundsätzlich unterscheidet AAFID zwei Arten des Informationsaustausches. Zum einen wäre das die *Intra-Host Communication*, also das Übermitteln von Nachrichten zwischen Komponenten auf einem einzelnen Rechner, zum anderen die *Inter-Host Communication*, also die Verständigung mehrerer Rechner über ein Netzwerk.

Bei der Intra-Host Communication unterscheidet man wiederum zwischen *One-to-Many Communication*, wie sie bei der Kontrolle eines Transceivers über seine Agenten vorkommt, und *Many-to-One Communication*, die bei der Übertragung von Berichten der Agenten an einen Transceiver verwendet wird.

Intra-Host Communication könnte wie folgt realisiert werden:

- **Message Queues** bieten den Vorteil der Priorisierung von Nachrichten, sind aber recht ressourcenintensiv und deshalb nicht skalierbar. Aus diesem Grund laufen sie Gefahr, Opfer von *Denial of Service* (kurz: DoS) Attacken zu werden.
- **Shared Memory** bietet eine effiziente Möglichkeit, Informationen zwischen Prozessen auszutauschen, da kein Kopieren von Informationen notwendig ist. Jedoch existiert auch hier die Problematik der Limitierung der Speichergröße. Nichts desto trotz bietet sich Shared Memory besonders für One-to-Many Communication an.
- **Pipes**, wie sie von Unix her bekannt sind, sind bidirektionale, unstrukturierte Datenströme, die nach dem Prinzip first-in first-out arbeiten. Daten werden auf das eine Ende der Pipe geschrieben und am anderen Ende gelesen. Wurde ein Datum gelesen, wird es aus der Pipe entfernt. Nach dem selben Prinzip sind auch first-in first-out Dateien denkbar.

An die Inter-Host Communication werden generell die selben Anforderungen gestellt, wie an jede beliebige andere Kommunikation in einem Netzwerk. Die Zeit, die eine Nachricht zu ihrem Empfänger benötigt, sollte möglichst kurz sein. Dafür sollten möglichst viele Nachrichten in kurzer Zeit verschickt werden können. Natürlich soll eine Nachricht nur vom Empfänger und von niemand anderem lesbar sein, also sicher. Zu guter letzt wird von dem Nachrichtenmechanismus hohe Ausfallsicherheit erwartet.

Die Entwickler von AAFID schlagen vor, ein bereits bestehendes Transportprotokoll wie UDP oder TCP zu verwenden, oder ein neues Protokoll zu entwerfen, das die eben aufgezählte Leistungsfähigkeit bietet.

3.4 DOMINO: Ein zweites konkretes Konzept eines hierarchisch organisierten DIDS

Als zweites sei *Distributed Overlay for Monitoring Internet Outbreaks* (kurz: DOMINO) [3] für hierarchisch organisierte verteilte Angriffserkennung erwähnt, welches an der University of Wisconsin, Madison entwickelt wurde.

In vielen Punkten ähnelt es AAFID, nicht zuletzt aufgrund seiner hierarchischen Struktur und seiner Verwendung von Software-Agenten. Jedoch wurde DOMINO bereits mit aus der Praxis stammenden Daten getestet. Dabei analysierte DOMINO Portscanlogs von 1600 konventionellen Firewalls.

DOMINO soll eingesetzt werden, um globale Angriffe zu erkennen, wie sie bei Wurmausbrüchen vorkommen. Verteilte Angriffserkennung ist dafür prädestiniert aufgrund der Tatsache, dass die Teilnehmer über das ganze Internet verteilt sein können, und sie so gleiches bösartiges Verhalten an verschiedenen Stellen detektieren können.

3.4.1 Axis Nodes, Satellite Nodes, Terrestrial Nodes

Grafik 4 zeigt den abstrahierten Aufbau eines DOMINO Netzes.

- Die zentrale Komponente sind die so genannten *Axis Nodes*. Sie sind verantwortlich für einen Großteil des Datenaustausches. Zu ihren Aufgaben gehört ein effizientes Reagieren auf Änderungen in der Netzwerkstruktur der Teilnehmer am DIDS. Ebenso fungieren sie als Punkte, an denen Informationen über erkannte Angriffe gesammelt werden. Diese Informationen können von den Axis Nodes oder von den Satellite Nodes stammen.
- *Satellite Communities* sind kleine Netzwerke bestehend aus *Satellite Nodes*. Sie sind hierarchisch organisiert. Dies äußert sich darin, dass gesammelte Informationen an höhere Satellite Nodes oder einen Axis Node als höchste Komponente weitergeleitet werden. Dabei werden von Satellite Nodes gesammelte Daten als weniger vertrauenswürdig eingestuft, als die von Axis Nodes gesammelten.
- Als eine Art „Freie Mitarbeiter“ kann man die *Terrestrial Nodes* verstehen. Sie sammeln ebenfalls Daten, jedoch werden diese als wenig vertrauenswürdig eingestuft. Über so genannte *Access Points* werden die gesammelten Daten auf die Ebene der Axis Nodes eingebracht. Terrestrial Nodes sollen die Abdeckung im zu untersuchenden Netzwerk erhöhen. Allerdings gehören sie selbst nicht wirklich zur DOMINO Infrastruktur, was sich darin äußert, dass sie nicht den DOMINO Kommunikationsmechanismus implementieren und über keine Active-Sinks verfügen, auf die im nächsten Abschnitt eingegangen wird.

All diese Komponenten können wieder durch Software-Agenten realisiert werden.

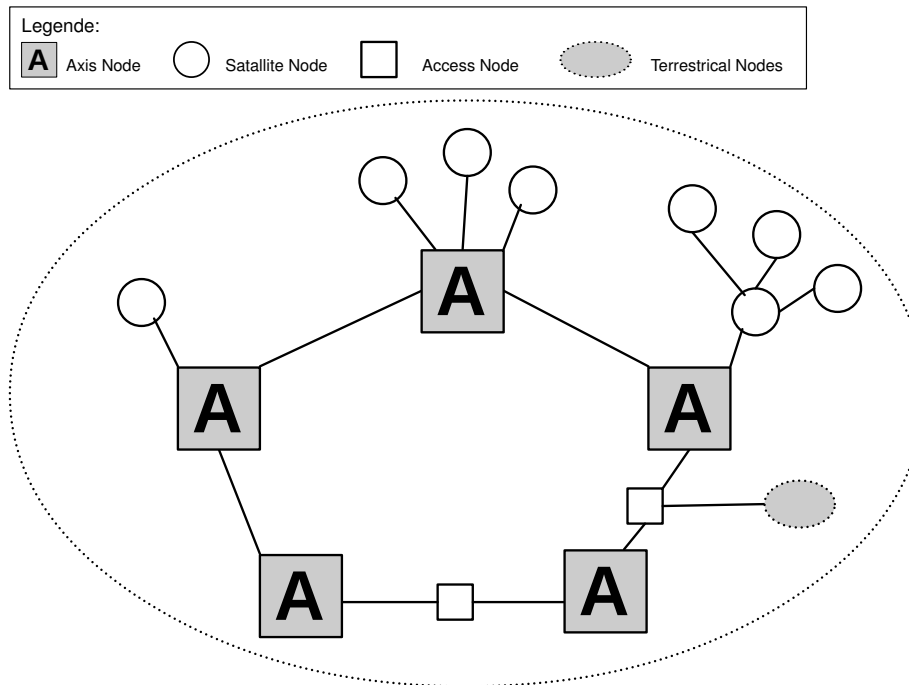


Abbildung 4: Diese Grafik zeigt das physische Layout eines beispielhaften DOMINO Systems. Es zeigt die Beziehungen zwischen Axis Nodes, Satellites, Terrestrial Nodes und Access Points. [3]

3.4.2 Active-Sinks

DOMINO soll zur Angriffserkennung von globalen Bedrohungen verwendet werden. Aus diesem Grund ist eine große Zahl von Punkten wünschenswert, an denen Informationen gesammelt werden. Um eine große Anzahl solcher Punkte zu erhalten, verwendet DOMINO neben den Teilnehmern selbst auch noch sogenannte *Active-Sinks*.

Internet Providern stehen Bereiche von IP-Adressen zur Verfügung, die diese an ihre Kunden verteilen. In der Regel sind zu einem konkreten Zeitpunkt nicht alle dieser IP-Adressen vergeben. Das heißt also, dass es IP-Adressen gibt, die zwar einem Internetprovider zugeteilt sind, der Internetprovider jedoch eine solche IP-Adresse im Moment nicht an einen Kunden vergeben hat.

Viele Angriffe beginnen damit, sequentiell IP-Adressen auszuprobieren, ob diese erreichbar sind und ob das mit der Adresse assoziierte Gerät angreifbar ist. Eine Active-Sink simuliert nun einen Rechner im Netzwerk mit einer im Moment ungenutzten IP-Adresse, in dem sie ein geringes Maß an Interaktion unterstützt. Die Active-Sinks sind bei DOMINO in den Axis Nodes lokalisiert.

Haupteinsatzgebiet von solchen Active-Sinks ist eigentlich das Verlangsamen von Wurmausbrüchen wie der von Code Red. Sie stellen einen sogenannten

Honeypot da, eine Art Falle für einen Angreifer. Dieser geht davon aus, im Honeypot ein potenzielles Opfer gefunden zu haben, verschwendet aber Zeit bei dem erfolglosen Versuch, den Honeypot zu infizieren.

Bei DOMINO dienen Active-Sinks mehr der Identifizierung von bösartigen oder fehlskonfigurierten Quellen, da diese einer Active-Sink gegenüber ein spezielles, entlarvendes Verhalten zeigen. Solche entlarvten Quellen werden in Blacklisten verwaltet, die nun kurz vorgestellt werden.

Datenschutzrechtlich ist dies jedoch äußerst bedenklich. Verlässt ein Benutzer das Internet, geht also offline, kann es sein, dass die eben von ihm verwendete IP-Adresse nun für eine Active-Sink verwendet wird. Nachrichten, die an die IP-Adresse des offline gegangenen Benutzers gesendet werden und für diesen bestimmt sind, können nun von der Active-Sink empfangen und ausgewertet werden, obwohl die Nachrichten nicht für die Active-Sink bestimmt sind.

3.4.3 Blacklisten

Bei den *Blacklisten* von DOMINO handelt es sich um Blacklisten im klassischen Sinne. In ihnen werden als anomal erkannte Quellen eingetragen und Anfragen von diesen Quellen in Zukunft einfach ignoriert.

Blacklisten werden von den Axis Nodes erstellt, da dort auch die Active-Sinks den Netzwerkverkehr überwachen. Aber auch aus den Ergebnissen der Hierarchie der Satellites lassen sich Einträge für die Blacklisten gewinnen. Da jeder Axis Node aufgrund seiner Unabhängigkeit von den anderen Axis Nodes andere Blacklisteinträge gewinnen kann, müssen die Blacklistinformationen zwischen den Axis Nodes in regelmäßigen Abständen ausgetauscht werden. Dabei hat jeder Axis Node lokale Blacklisten, die er allein erstellt und die er an die anderen Axis Nodes sendet. Aus den einzelnen lokalen Blacklisten können nun globale Blacklisten gewonnen werden.

Die später vorgestellten Ergebnisse zeigen, dass es ratsam ist, für verschiedene Zeitspannen verschiedene Blacklisten anzulegen. Zeitspannen von einer Stunde, einem Tag, einer Woche oder sogar einem Monats sind denkbar.

Nachdem nun ein paar der Charakteristika von DOMINO erläutert wurden, wird nun auf zwei Lösungsansätze der bei einem DIDS entstehenden Probleme eingegangen.

3.4.4 Sicherheit

Da es sich bei einem DIDS um eine hoch sicherheitsrelevante Anwendung handelt, muss auch garantiert sein, dass die Kommunikation zwischen den Teilnehmern sicher ist und man sich auf die Integrität der Teilnehmer verlassen kann. Aus diesem Grund wird bei DOMINO der komplette Austausch von Nachrichten und Berichten verschlüsselt. Besonders bei den Axis Nodes ist dies wichtig, da deren gesammelte Informationen höchste Vertrauenswürdigkeit genießen. Die Verschlüsselung wird mit einem öffentlichen Schlüssel vollzogen, der Einsatz des RSA-Algorithmus wäre hier denkbar. Allerdings könnte man auch andere Ver-

schlüsselungsalgorithmen einsetzen. Es ist auch möglich, spezielle *Certificate Authorities* (kurz: CA) für DOMINO einzuführen, mit denen sich neue Teilnehmer an DOMINO über ein spezielles Schlüsselaustauschprotokoll anmelden können.

Neben der Verschlüsselung der Nachrichten wird deren Integrität auch noch mittels Prüfsummen sichergestellt. Als Hashalgorithmus könnte SHA-1 zum Einsatz kommen, da dieser momentan in Bezug auf Kollisionen als sicher gilt.

3.4.5 Kommunikation bei DOMINO

Für die Kommunikation zwischen den Teilnehmern findet das *DOMINO Summary Exchange Protocol* Verwendung. Es basiert auf XML und kennt sieben Nachrichtenkategorien: Alerts, Summary Messages, Heartbeats, Topology Messages, Queries, DB Updates und Triggers. Das genaue Aussehen der XML-Nachrichten kann dem Anhang von [3] entnommen werden.

Das Protokoll wird für die Kommunikation zwischen den Axis Nodes verwendet. Eine modifizierte Variante kommt beim Informationsaustausch mit den Satellites zum Einsatz. Die Terrestrial Nodes verwenden in der Regel nicht dieses Protokoll.

3.4.6 Auswertung des Praxiseinsatzes

Es existieren bereits Praxiserfahrungen mit DOMINO. Zu diesem Zweck wurden Daten von über 1600 Portscanlogs konventioneller Firewalls über einen Zeitraum von vier Monaten verwendet, um Erfahrungen mit DOMINO zu sammeln.

Zwei der Ergebnisse werden hier vorgestellt. Zum einen erkannte man, dass es sinnvoll ist, verschiedene Blacklisten für unterschiedliche Zeiträume anzulegen. Zum anderen wird gezeigt, wie viele Einträge eine Blacklist haben soll, um einen guten Kompromiss zwischen Größe und Effektivität der Blacklist zu erhalten. Betrachtet man Quellen von Portscans, stellt man fest, dass von jeder der Top10-Quellen (damit sind die 10 Quellen gemeint, von denen die meisten Portscans ausgehen) der erste Scan bereits innerhalb einer Stunde erfolgt. Das bedeutet, dass egal zu welchem Zeitpunkt „ x “ eine Analyse der Portscans beginnt, zum Zeitpunkt „ $x + \text{eine Stunde}$ “ jede der Top10-Quellen den Rechner, dessen Logs gerade untersucht werden, gescant hat. Mit einer Blacklist, die nur diejenigen Quellen enthält, die innerhalb der letzten Stunde einen Scan durchgeführt haben, kann man also bereits ein Groß der Scans abfangen. Weitert man aber nun das Suchfeld auf die 60 häufigsten Quellen aus, von denen ein Portscan ausgeht sieht man, dass es sinnvoll ist, auch eine Blacklist für Aktivitäten über einen ganzen Monat zu haben. Aus diesem Grund verwendet DOMINO mehrere Blacklisten mit unterschiedlichen Granularitäten.

Blacklisten müssen den Spagat aus Größe und Effektivität schaffen, denn die lokalen Blacklisten eines Axis Nodes werden verwendet, um bei den anderen Axis Nodes globale Blacklisten anzulegen. Eine geringe Größe der lokalen Blacklisten ist deshalb aus Gründen der Bandbreite wünschenswert. Die Praxis hat gezeigt, dass zu jeder beliebigen Stunde 90% aller Scans von gerade einmal etwa 1024

Quellen ausgingen. 60% aller Scans gingen von gerade einmal 16 Quellen aus, was noch erstaunlicher ist. Das zeigt, dass die Blacklisten nicht all zu groß sein müssen.

4 Dezentrale Idee verteilter Angriffserkennung

Im letzten Kapitel 3 wurde ausführlich die Idee der verteilten Angriffserkennung in einem hierarchisch organisierten Netz gestützt von Software-Agenten vorgestellt.

Nun soll eine andere Idee zeigen, dass die Unabhängigkeit der Teilnehmer in einem DIDS wesentlich besser realisiert werden kann, als das bisher gezeigt wurde. Verteilte Angriffserkennung basierend auf P2P-Netzwerken (Abschnitt 4.1) macht dies möglich. Diese Idee erlaubt es, den Teilnehmerverbund wesentlich lockerer zu strukturieren.

Allerdings treten auch hierbei wieder die bereits aus 2.3 bekannten Probleme verteilter Angriffserkennung auf, die sich bei P2P-Netzen noch weiter intensivieren. Neue Lösungen bezüglich Teilnehmerverwaltung und Kommunikation müssen gefunden werden.

Wieder wird zuerst auf die grundlegende Technik dieser dezentral organisierten Idee eingegangen, bevor mit INDRA [4] ein konkretes Konzept vorgestellt wird.

4.1 Peer-to-Peer

Bei einem P2P-Netz handelt es sich um eine Art von Netzwerkstruktur, die dezentral organisiert ist. Das bedeutet, dass die Teilnehmer direkt miteinander kommunizieren und es somit im idealen Fall keinerlei zentrale Punkte gibt, die das Netzwerk verwalten. Das Gegenteil eines P2P-Netzes ist die klassische Server-Client-Struktur, in der ein Server Informationen zur Verfügung stellt und Clients diese in Anspruch nehmen. Folglich sind alle Teilnehmer an einem P2P-Netz gleichberechtigt. Sie bieten gleichzeitig Dienstleistungen an und nehmen Dienstleistungen von anderen Teilnehmern in Anspruch.

Alle Entscheidungen, die in einem P2P-Netz getroffen werden, basieren auf lokalen Informationen. Alle Teilnehmer verfügen in der Regel über ähnliche Fähigkeiten, Aufgaben und Rechte.

Die Teilnehmer an einem P2P-Netz werden *Peers* genannt. Da aber die partizipierenden Objekte an einem DIDS bisher in diesem Text immer als Teilnehmer bezeichnet wurden, wird dies nun auch so fortgesetzt. Darüber hinaus wird der Begriff *Node* als äquivalenter Terminus für Peer bzw. Teilnehmer verwendet.

Eine solche Netzwerkstruktur kann man nun als grundlegendes Overlay-Netzwerk [11] für ein System verteilter Angriffserkennung verwenden. Unter einem Overlay-Netzwerk versteht man ein Netzwerk, das auf einem bereits bestehenden Netzwerk aufbaut. Dabei abstrahiert man im Overlay-Netzwerk vom eigentlich zu Grunde liegenden Netzwerk um spezielle Eigenschaften eines Netzwerks besser realisieren zu können. Dabei entsteht eine neue Netzwerktopologie, die sich

logisch oder strukturell von der zu Grunde liegenden Netzwerkstruktur unterscheiden kann.

4.2 INDRA: Ein konkretes Konzept eines DIDS basierend auf einem P2P-Netz

Intrusion Detection and Rapid Action (kurz: INDRA) [4] ist ein Ansatz verteilter Angriffserkennung basierend auf einem P2P-Netz.

4.2.1 Organisation der Teilnehmer an INDRA: Pastry

Tabelle 1: So könnte eine Pastry Routing Table aussehen. Sie gehört dem Knoten mit der NodeID **37105264**. Sie ist für ein Netzwerk mit maximal 2^{24} Teilnehmern ausgelegt. Konfigurationsparameter $b = 3$. Sie besitzt 7 Zeilen. Die IP-Adressen, mit denen die NodeIDs assoziiert sind, sind ebenfalls eingetragen.

NodeID des jetzigen Knoten: 37105264				
Muster		$2^b - 1 = 2^3 - 1 = 7$ Einträge pro Zeile.		
$\underbrace{x_0}_{\text{verschieden}} \quad x_1 x_2 x_3 x_4 x_5 x_6 x_7$		$\underbrace{0}_{\text{verschieden}} \quad \underbrace{0025712}_{\text{beliebig}}$ → 128.118.124.123	...	$\underbrace{7}_{\text{verschieden}} \quad \underbrace{1054313}_{\text{beliebig}}$ → 42.22.225.145
$\underbrace{x_0}_{\text{gleich}} \quad \underbrace{x_1}_{\text{verschieden}} \quad x_2 x_3 x_4 x_5 x_6 x_7$		$\underbrace{3}_{\text{gleich}} \quad \underbrace{0}_{\text{verschieden}} \quad \underbrace{123033}_{\text{beliebig}}$ → 116.14.228.222	...	$\underbrace{3}_{\text{gleich}} \quad \underbrace{6}_{\text{verschieden}} \quad \underbrace{451001}_{\text{beliebig}}$ → 40.141.221.60
$\underbrace{x_0 x_1}_{\text{gleich}} \quad \underbrace{x_2}_{\text{verschieden}} \quad x_3 x_4 x_5 x_6 x_7$		$\underbrace{37}_{\text{gleich}} \quad \underbrace{0}_{\text{verschieden}} \quad \underbrace{71264}_{\text{beliebig}}$ → 174.48.83.206	...	$\underbrace{37}_{\text{gleich}} \quad \underbrace{7}_{\text{verschieden}} \quad \underbrace{35530}_{\text{beliebig}}$ → 20.163.81.159
$\underbrace{x_0 x_1 x_2}_{\text{gleich}} \quad \underbrace{x_3}_{\text{verschieden}} \quad x_4 x_5 x_6 x_7$		$\underbrace{371}_{\text{gleich}} \quad \underbrace{1}_{\text{verschieden}} \quad \underbrace{0334}_{\text{beliebig}}$ → 85.236.146.191	...	$\underbrace{371}_{\text{gleich}} \quad \underbrace{7}_{\text{verschieden}} \quad \underbrace{3123}_{\text{beliebig}}$ → 158.86.20.79
$\vdots$		$\vdots$	$\vdots$	$\vdots$
$\underbrace{x_0 x_1 x_2 x_3 x_4 x_5}_{\text{gleich}} \quad \underbrace{x_6}_{\text{verschieden}} \quad x_7$		$\underbrace{371052}_{\text{gleich}} \quad \underbrace{0}_{\text{verschieden}} \quad \underbrace{0}_{\text{beliebig}}$ → 108.227.203.72	...	$\underbrace{371052}_{\text{gleich}} \quad \underbrace{7}_{\text{verschieden}} \quad \underbrace{1}_{\text{beliebig}}$ → 15.9.50.1
$\underbrace{x_0 x_1 x_2 x_3 x_4 x_5 x_6}_{\text{gleich}} \quad \underbrace{x_7}_{\text{verschieden}}$		$\underbrace{3710526}_{\text{gleich}} \quad \underbrace{4}_{\text{verschieden}}$ → 120.9.246.129	...	$\underbrace{3710526}_{\text{gleich}} \quad \underbrace{7}_{\text{verschieden}}$ → 228.165.17.58

Da in P2P-Netzen alle Teilnehmer unabhängig von einander sind, muss eine Lösung gefunden werden, welche das effektive gegenseitige Finden von Teilnehmern im Netzwerk erlaubt. Es wäre denkbar, dass die Teilnehmer ständig Listen derjenigen Knoten austauschen, die sie kennen. Dies würde allerdings zu einer hohen Last für das Netzwerk führen, denn man müsste es schaffen, dass jeder

Teilnehmer alle anderen Teilnehmer am Netzwerk kennt. Hierbei handelt es sich nicht nur um ein Problem von INDRA sondern um ein Problem bei P2P-Netzen ganz allgemein.

Zur Lösung dieses Problems verwendet INDRA *Pastry* [7], ein sicheres, robustes und selbstorganisierendes Netzwerk-Overlay, insbesondere für das Internet. Es übernimmt die effektive Verwaltung der Teilnehmer und ermöglicht über diesen Austausch von Informationen in einem P2P-Netz. Es folgt eine grobe Skizzierung von Pastry. Für genauere Informationen siehe Quelle [7].

Jedem Teilnehmer an Pastry wird eine *NodeID* zugewiesen. Da diese einmalig im (INDRA-) Netzwerk sein soll, muss der Raum der NodeIDs entsprechend groß gewählt werden. Pastry legt bei jedem Teilnehmer eine Tabelle an, die zu einer gegebenen NodeID die IP-Adresse des Teilnehmers enthält. Diese *Routing Table* besitzt $\lceil \log_{2^b} N \rceil$ Zeilen und $2^b - 1$ Einträge pro Zeile, wobei N die Anzahl der Knoten im Netzwerk und b ein Konfigurationsparameter mit typischen Wert 4 ist.

Für den hiesigen Zweck werden die NodeIDs nicht binär sondern als eine Folge von Stellen zur Basis 2^b notiert. Derjenige Knoten, über dessen Routing Table im folgenden gesprochen wird, wird als „jetziger Knoten“ bezeichnet. Die $2^b - 1$ Einträge in der $n + 1$ -ten Zeile der Routing Table verweisen jeweils auf einen Knoten, dessen NodeID zu der NodeID des jetzigen Knoten in den ersten n Stellen übereinstimmt. Die $n + 1$ -te Stelle muss sich jedoch von der $n + 1$ -ten Stelle des jetzigen Knoten unterscheiden. Dafür gibt es $2^b - 1$ Möglichkeiten. Ferner wird verlangt, dass sich die $2^b - 1$ Einträge in der $n + 1$ -ten Zeile an ihrer $n + 1$ -ten Stelle untereinander unterscheiden. Das führt dazu, dass jede Ziffer $a \in [0, \dots, 2^b - 2]$ an der $n + 1$ -ten Stelle der NodeID in der $n + 1$ -ten Zeile vorkommt. Jede NodeID darf in der Routing Table maximal einmal auftreten. Tabelle 1 zeigt ein Szenario, in dem $b = 3$ ist und es theoretisch $N = 8^8 = 2^{24}$ Knoten im Netzwerk geben kann.

Soll nun die IP-Adresse eines bestimmten Teilnehmers ermittelt werden, so wird dessen NodeID benötigt. Man vergleicht nun die NodeID des Empfängers mit der des jetzigen Knoten und ermittelt die Anzahl der führenden Stellen, die bei beiden NodeIDs übereinstimmen. Dies entspricht der oben erwähnten Zahl n . Ist $n = \lceil \log_{2^b} N \rceil - 1$, so tritt in der letzten Zeile der Routing Table garantiert die Empfänger-NodeID auf. Eine Verknüpfung mit der IP-Adresse des Empfängers ist also vorhanden.

Meist ist jedoch $n < \lceil \log_{2^b} N \rceil - 1$, weshalb ein Routing über mehrere Stationen, sogenannte *Hops*, notwendig wird. Man schlägt in der $n + 1$ -ten Zeile der Routing Table nach. In ihr befinden sich $2^b - 1$ Einträge. Man bestimmt nun denjenigen dieser Einträge, dessen NodeID zur Empfänger-NodeID minimalen Abstand hat, die Differenz also möglichst klein ist. An diesen Eintrag sendet man nun den „NodeID-IP-Adresse-Zuordnungsauftrag“. Dort angekommen wird der Vorgang wiederholt (mit dem Bestimmen der n gemeinsamen führenden Stellen der beiden NodeIDs).

Mit diesem Mechanismus kann man nun garantieren, dass das erfolgreiche Zuordnen einer NodeID (insofern sich diese im Netzwerk befindet) zur entsprechenden IP-Adresse nach maximal $\lceil \log_{2^b} N \rceil$ Hops vollzogen werden kann. Dies

ist die Anzahl der Stellen der NodeIDs. Die Anzahl der Hops ist also logarithmisch zu der Anzahl der Teilnehmer. Nach jedem Hop stimmen die NodeID des Zielknotens und die NodeID des jetzigen Knotens in mindestens einer führenden Stelle mehr überein als beim letzten Hop. Es werden solange Hops ausgeführt, bis der Suchauftrag beim eigentlichen Empfänger angekommen ist.

4.2.2 Bekanntmachung von Ereignissen bei INDRA: Scribe

Nachdem gezeigt wurde, wie Teilnehmer an einem P2P-Netz mittels Pastry verwaltet und zu einer gegebenen NodeID effizient die zugehörige IP-Adresse gefunden werden kann, folgt nun die Skizzierung eines *Event Notification Systems* das es ermöglicht, eine Vielzahl von Ereignissen einer großen Menge von Abnehmern dieser Ereignisse zur Verfügung zu stellen. Diese Art der Bekanntmachung von Ereignissen ist ideal für ein P2P-Netz. Anhand von Scribe [6] wird gezeigt, was man sich unter einem Event Notification System vorzustellen hat. Jedem Teilnehmer an Scribe ist es erlaubt, ein *Topic* zu eröffnen. Andere Teilnehmer können sich bei Interesse an einem solchen Topic registrieren und werden zu *Subscribers* dieses Topic. Jeder berechtigte Subscriber eines Topic kann nun Ereignisse oder Informationen bekannt geben, über die jeder Subscriber an der gleichen Topic informiert wird. Ein solches System muss also vier Funktionen erfüllen:

- Es muss möglich sein, neue Topics zu erstellen (**create**).
- An diesem Topics muss man sich anmelden können (**subscribe**).
- Aber auch das Abmelden muss möglich sein (**unsubscribe**).
- Gibt ein Subscriber ein konkretes Ereignis in einer Topic bekannt, so soll dies den anderen Subscribern an der Topic mitgeteilt werden (**publish**).

Scribe verwendet das Overlay von Pastry um das Erstellen von Topics, das Anmelden bei einer Topic und das Bekanntmachen von Ereignissen innerhalb einer Topic zu ermöglichen.

Nachdem nun zwei Lösungen für typische Probleme in einem P2P-Netz präsentiert wurden, wird folgend auf Besonderheiten und Charakteristika von INDRA eingegangen.

4.2.3 INDRA Daemons

Sämtliche Funktionalität von INDRA wird von sogenannten *Daemons* erfüllt. Diese Daemons entsprechen bei INDRA Java Threads. Jeder Daemon gehört zu einer der folgenden vier Klassen:

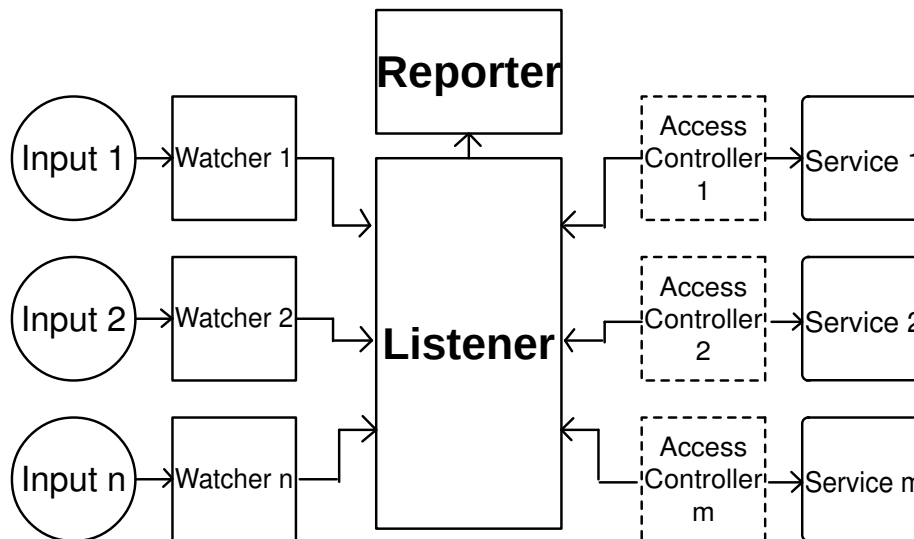


Abbildung 5: Diese Grafik zeigt das Zusammenspiel der verschiedenen INDRA Daemons. [4]

- **Watchers:** Sie stellen das erste Level der Erkennung verdächtiger Aktivität dar. Dazu gehören beispielsweise mehrere fehlgeschlagene Anmeldeversuche oder Portscans. Zum Bereich der Überwachung zählt sowohl der lokale Rechner als auch das komplette Netzwerk.
- **Listeners:** Zu ihren Aufgaben zählt das Sammeln von Warnungen, die sie von den Watchers erhalten und erstellen je nach Konfiguration durch den Administrator oder die aktuelle Sicherheitsstufe entsprechende Warnungen für die Access Controllers. Somit stehen sie zwischen Watchers und Access Controllers. Wären Watchers Sinnesorgane und Access Controllers Gliedmaßen, so wären die Listeners die zentrale Kontrollinstanz, die aufgrund von Sinneswahrnehmungen die entsprechenden motorischen Aktivitäten ausführt.
- **Access Controllers:** Diese Daemons stellen den kontrollierten Zugriff auf Ressourcen wie Festplatte, Netzwerk, Arbeitsspeicher usw. zu Verfügung. Diese Zugriffskontrolle erfolgt automatisch und wird von den Listeners ausgelöst. Bekommt ein Access Controller beispielsweise von einem Listener die Information, dass von einem Rechner im Netz aus versucht wird, einen Benutzeraccount zu knacken, so filtert der Access Controller alle Kombinationen aus Angreifer und angegriffenen Account heraus.
- **Reporters:** Die Kommunikation mit der Außenwelt wird durch sie ermöglicht. Diese Kommunikation kann einerseits das Empfangen von Warnungen anderer Teilnehmer an INDRA sein, die an die lokalen Listener weitergeleitet

werden, oder andererseits das Versenden von gesammelten Warnungen der lokalen Listener an andere Netzwerkteilnehmer.

Jeder Daemon kann wieder als Software-Agent, wie er bereits in 3.1 vorgestellt wurde, realisiert werden.

Grafik 5 veranschaulicht das Zusammenspiel der einzelnen INDRA Daemons.

4.2.4 Neighborhood Watch

Bei INDRA wird nicht nur Wert darauf gelegt, lokale Angriffe und Bedrohungen auf einen Rechner festzustellen. Da es sich um verteilte Angriffserkennung handelt, jedoch keine Hierarchie mehr vorhanden ist, deren oberste Instanz einen globalen Alarm auslösen könnte, wird ein Verfahren benötigt, dass die Informationen partizipierender Teilnehmer an INDRA mit einbezieht. Bei INDRA ist dies die *Neighborhood Watch*.

Teilnehmer von INDRA sind in der Lage über Angriffe auf Nachbarn informiert zu werden. Dies ist effizient möglich, falls es sich bei dem Netzwerk um ein *Shared Network* handelt, denn nur so ist *Broadcasting* [12] möglich. Darüber hinaus erzielt man den selben Effekt, wenn man INDRA auf einem *Internet Gateway* oder einem *Internet Switch* installiert.

Auf diese Weise lassen sich Angriffe erkennen, die wie folgt aussehen könnten:

- Ein Angreifer **A** startet Angriffe auf ein vertrauenswürdigen Netzwerk.
- Auf allen Rechnern im vertrauenswürdigen Netzwerk läuft INDRA. Einer dieser Rechner ist **B**.
- Da auch **B** von **A** angegriffen wird, erkennt **B** den Angriff und sendet eine Sicherheitswarnmeldung an seine vertrauenswürdigen Nachbarn, dass von **A** ein Angriff ausgeht.
- Die Nachbarn erhalten die Sicherheitswarnung von **B**, verifizieren diese (die Sicherheitswarnung kann wie bei DOMINO verschlüsselt sein und der Inhalt kann mittels eines Hashwertes geprüft werden) und setzen **A** auf eine Blacklist.
- Nachdem der Angriff von **A** auf **B** gescheitert ist, versucht **A** Angriffe auf andere Teilnehmer im vertrauenswürdigen Netz zu starten. Da diese aber bereits von **B** gewarnt wurden, misslingt der Angriff.

Die Kommunikation, die für die Neighborhood Watch benötigt wird, wird über Scribe realisiert.

4.2.5 Vertrauenssystem

Auch INDRA beschäftigt sich mit der Problematik der zuverlässigen Authentifizierung von Teilnehmern, die für ein vertrauenswürdigen Netzwerk erforderlich ist. INDRA verwendet hierfür ein *Web-of-Trust*, bei dem anders als bei DOMINO, keine digitalen Zertifikate von einer zentralen CA geholt werden, sondern

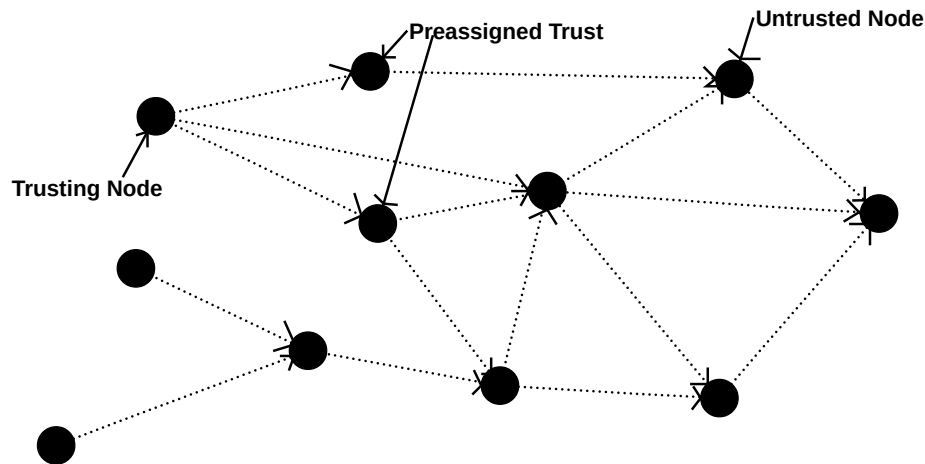


Abbildung 6: Diese Grafik zeigt ein Web-of-Trust. Die Kanten zwischen den Knoten geben den Grad der Vertrauenswürdigkeit zwischen den Knoten wieder. [4]

Vertrauenswürdigkeit entlang der Knoten bestimmt wird. Dies wird auch eher der Struktur eines P2P-Netzes gerecht.

Wie Grafik 6 zeigt, sind Knoten über sogenannte *Trust Relationships* miteinander verbunden. Sie werden als gewichtete Kanten (wie die Gewichte der Kanten bestimmt werden, ist nicht festgelegt) im Graphen gezeichnet. Der Wert der Gewichtung gibt den Grad der Vertrauenswürdigkeit zwischen den anhängenden Knoten wieder.

Ein paar der Knoten haben bereits von Anfang an einen konkreten Wert der Vertrauenswürdigkeit, während bei den restlichen Knoten deren Vertrauenswürdigkeit anhand deren *Trust Relationships* zu anderen Teilnehmern erst noch berechnet werden muss.

Jedoch handelt es sich bei dieser Art von Vertrauensgewichtung in einem Web-Of-Trust noch um ein sehr junges Konzept, weshalb der Nutzen und die Effektivität eines Web-Of-Trust noch nicht genau erfasst werden kann. Auch gibt es noch keine Regeln, wie die Gewichtung der Kanten zwischen den Knoten für die gegenseitige Vertrauenswürdigkeit bestimmt werden soll.

4.2.6 INDRA PlugIn

INDRA bietet einen Mechanismus an, über den zur Laufzeit zusätzliche Daemons zugeschaltet werden können. Dies ist zum Beispiel beim Bekanntwerden neuer Sicherheitslücken notwendig. Der Administrator kann neuen Java-Code schreiben, der die benötigte neue Funktionalität liefert und diesen an andere Teilnehmer an INDRA weitergeben. Über einen speziellen *PlugIn*-Manager lassen sich diese Erweiterungen nach erfolgreicher Gültigkeitsprüfung dann dynamisch nachladen.

Dabei können über den PlugIn-Manager alle Arten von Daemons nachgeladen werden, so dass auch für die neuen Funktionen auch neue Systemressourcen angefordert werden können (geschieht durch Access Controllers).

5 Diskussion: Einsatz der vorgestellten Ideen in der Praxis

Bei den bisher vorgestellten Ideen verteilter Angriffserkennung (hierarchisch organisiert (3) und dezentral organisiert (4)) und deren möglichen Konzepten der Implementierung (AAFID 3.3, DOMINO 3.4 und INDRA 4.2), handelt es sich bisher eher um Theorie als um Praxis.

Eine wichtige praktische Frage ist, wie und wo man die vorgestellten Konzepte verteilter Angriffserkennung in der Realität einsetzen kann. Schließlich muss die Software, die die geforderte Funktionalität bietet, irgendwo lokalisiert sein.

Die naheliegendste Vermutung wäre, Clients zu schreiben, die die Funktionalität der einzelnen Konzepte bieten. Soll nun ein Rechner Teilnehmer an einem DIDS werden, so installiert man einfach den Client auf dem entsprechenden Rechner. Dies ist allerdings eher umständlich und lässt viel vorhandenes Potential ungenutzt (zum Beispiel könnten Log-Dateien bereits vorhandener Sicherheitssysteme zur Analyse verwendet werden).

Eine Verbesserung wäre, Grundfunktionalität eines DIDS in den Kernel vom Betriebssystem zu integrieren. Auf diese Weise können Implementierungen, die diese Grundfunktionalität nutzen, besonders effektiv ausgeführt werden.

PlugIns sind eine andere Möglichkeit. Es existieren ja bereits umfangreiche Sicherheitssysteme nach klassischen Mustern. Anstatt diese auszutauschen, könnte man sie um die Funktionalität verteilter Angriffserkennung erweitern. Die Heterogenität von DIDS kommt hierbei zu Hilfe. Im privaten Endanwenderbereich beispielsweise dominieren wenige Sicherheitssuiten von wenigen Herstellern wie Symantec [17], G DATA [18] oder McAfee [19] den Markt. Es wäre also denkbar, für diese gängigen Produkte PlugIns zu entwickeln, die die von den Konzepten verteilter Angriffserkennung geforderten Schnittstellen erfüllen. Auf diese Weise könnte man eine enorme Teilnehmerzahl für ein DIDS rekrutieren.

Leicht modifiziert könnte man PlugIns auch bei Hardware verwenden. In Deutschland wird von Privatanwendern hauptsächlich ADSL [13] in verschiedenen Geschwindigkeiten für den Zugang ins Internet genutzt. Dabei werden von den Internet Providern Hardwarerouter [15] für den Zugang angeboten, die meist auch ein IDS beinhalten. Gäbe es Erweiterungen der Firmwares dieser Router, die die Funktionalität für die Teilnehmer an einem DIDS nachrüsten, ließen sich auch so große Teilnehmerzahlen gewinnen.

6 Fazit

Dieser Text beschäftigt sich mit verteilter Angriffserkennung. Er beleuchtet die Nachteile klassischer, zentral erfasster bzw. ausgewerteter Angriffserken-

nung (Abschnitt 2.1) und zeigt Lösungen, die die verteilte Angriffserkennung dafür bietet (Abschnitt 2.2). Zwei Ideen verteilter Angriffserkennung werden vorgestellt. Hierarchisch organisierte Angriffserkennung unter Verwendung von Software-Agenten (Abschnitt 3) und dezentral organisierte Angriffserkennung in P2P-Netzen (Abschnitt 4). Es werden Konzepte präsentiert (AAFID 3.3, DOMINO 3.4 und INDRA 4.2), wie diese Ideen in der Realität umgesetzt werden können. Dabei wird besonders auf Lösungen für Probleme, die bei verteilter Angriffserkennung gegenüber zentraler Angriffserkennung entstehen können, Wert gelegt.

Hierarchisch organisierte Angriffserkennung ist in eine starre, relativ unflexible Struktur gepresst, weshalb sie nur als erster Schritt auf dem Weg zu einem optimalen DIDS gesehen werden kann. Die dezentrale Idee mit einem P2P-Netz kommt dieser erwünschten Optimum schon recht nahe.

Es zeigt sich, dass es sich bei diesen Konzepten größtenteils um Theorie handelt und deren Praxistauglichkeit sich noch bewähren muss.

Literatur

- [1] Ajith Abraham, Johnson Thomas: Distributed Intrusion Detection Systems - A Computational Intelligence Approach.
- [2] Jai Balasubramaniyan, Jose Garcia-Fernandez, David Isacoff, Eugene Spafford, Diego Zamboni: An Architecture for Intrusion Detection using Autonomous Agents. Technical Report 98/05, Purdue University, 1998.
- [3] Vinod Yegneswaran, Paul Barford, Somesh Jha: Global Intrusion Detection in the DOMINO Overlay System. Computer Sciences Department, University of Wisconsin, Madison.
- [4] Ramaprabhu Janakiraman, Marcel Waldvogel, Qi Zhang: INDRA - A peer-to-peer approach to network intrusion detection and prevention.
- [5] Phillip Porras, Peter Neumann: EMERALD - Event Monitoring Enabling Responses to Anomalous Live Disturbances. In Proceedings of the 20th National Information System Security Conference, National Institute of Standards and Technology, 1997.
- [6] Antony Rowstron, Anne-Marie Kermarrec, Miguel Castro, Peter Druschel: SCRIBE - The Design of a large-scale event notification infrastructure. In *Network Group Communication*, 2001.
- [7] Antony Rowstron, Peter Druschel: Pastry - Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed System Platforms (Middleware)*, Heidelberg, 2001.

- [8] Sonja Buchegger, Jean-Yves Le Boudec: Performance Analysis of the CONFIDANT Protocol. In *Proceedings of IEEE/ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC)*, Lausanne, 2002.
- [9] Hermann de Meer: Skript zur Vorlesung „Peer-to-Peer Networking“ im SS 2005, Passau, 2005.
- [10] Wikipedia über Software-Agenten:
<http://de.wikipedia.org/w/index.php?title=Software-Agent&oldid=25055645>
 Version vom 13.12.2006.
- [11] Wikipedia über Overlay-Netzwerk:
<http://de.wikipedia.org/w/index.php?title=Overlay-Netzwerk&oldid=25236201>
 Version vom 17.12.2006.
- [12] Wikipedia über Broadcast:
<http://de.wikipedia.org/w/index.php?title=Broadcast&oldid=25640145>
 Version vom 28.12.2006.
- [13] Wikipedia über ADSL:
http://de.wikipedia.org/w/index.php?title=Asymmetric_Digital_Subscriber_Line&oldid=26349237
 Version vom 13.01.2007.
- [14] Wikipedia über SQL Slammer:
http://de.wikipedia.org/w/index.php?title=SQL_Slammer&oldid=25803018
 Version vom 01.01.2007.
- [15] Wikipedia über Router:
<http://de.wikipedia.org/w/index.php?title=Router&oldid=26405796>
 Version vom 14.01.2007.
- [16] Produkthomepage des MSSQL-Servers von Microsoft:
<http://www.microsoft.com/sql/default.msp>
- [17] Homepage von Symantec Deutschland:
<http://www.symantec.com/de/de/index.jsp>
- [18] Homepage von G DATA:
<http://www.gdata.de/portal/DE/>
- [19] Homepage von McAfee Deutschland:
<http://de.mcafee.com/>