Inhaltsverzeichnis

1 Einleitung

Stepping Stones sind Computer, die in einem Netzwerk erreichbar sind und dazu benutzt werden, um von dort aus Dienste eines anderen Computers in Anspruch zu nehmen. Man möchte Stepping Stones erkennen, um Schritte in Bezug auf die Sicherheit der Computersysteme und möglicherweise auch eine Strafverfolgung einleiten zu können. Ziel dieser Diplomarbeit ist es, einen Überblick über die möglichen Erkennungsmethoden für Stepping Stones zu geben, die einzelnen Methoden miteinander zu vergleichen und eine Abschätzung über die Einsetzbarkeit in breitbandigem Netzwerkverkehr zu geben.

Zuerst wird eine kurze Zusammenfassung der zu Grunde liegenden Sachverhalte gegeben. Dabei werden grundsätzliche Themen der Intrusion Detection behandelt und ein Eindruck vermittelt, welche Probleme bei der Erkennung auftreten können. Weiter wird erläutert, was man unter Stepping Stones versteht und welche Ziele mit der Verwendung eines oder mehrerer Stepping Stones erreicht werden sollen.

Im weiteren Verlauf wird ein umfassender Literaturüberblick von Algorithmen zur Erkennung vorgestellt und hinsichtlich ihrer Verwendbarkeit analysiert. Dabei gilt besonderes Augenmerk zum einen der Geschwindigkeit der Methode, da ein langsamer Algorithmus den Netzwerkverkehr durch die Erkennung beeinträchtigen würde. Zum anderen wird darauf geachtet, daß kein aktiver Eingriff in den Ablauf des Netzwerkverkehrs nötig ist, da dieser Eingriff sowohl eine Erkennung durch den Benutzer des Stepping Stones ermöglicht als auch zu einer massiven Beeinträchtigung der normalen Nutzung des Netzwerklinks führen kann. Die Erkennung durch den Benutzer kann bei einer illegalen Nutzung dazu führen, daß die Verbindung unterbrochen wird, bevor man in der Lage ist weitere Ermittlungen einzuleiten.

Anschließend wird eine Analyse der ausgewählten Algorithmen auf realem Verkehr des Münchner Wissenschaftsnetzes (MWN) durchgeführt und erläutert, was dabei gefunden worden ist bzw. zu welchen Problemen es im realen Einsatz gekommen ist. Das MWN stellt für die gesamten Münchner Hochschulen und weitere Institutionen die Verbindung zum Internet her und bietet damit die ideale Grundlage, um auf einer breitbandigen Verbindung die Algorithmen zu testen.

Abschließend wird diese Arbeit einen Ausblick auf den realen Einsatz der Erkennungsmöglichkeiten geben und auf Risiken oder Erweiterungsmöglichkeiten eingehen. Außerdem werden Szenarien vorgestellt, die es einem potentiellen Angreifer ermöglichen die Erkennung zu umgehen.

2 Intrusion Detection

Bevor man sich einer näheren Betrachtung eines besonderen Bereiches der Intrusion Detection, der Erkennung von Stepping Stones, widmet, sollte man einen Eindruck von der Thematik der Intrusion Detection im Allgemeinen haben. Es ist darüber hinaus wichtig sich Gedanken zu machen, welche Szenarien von der Intrusion Detection aufgedeckt werden können und welche Schlußfolgerungen aus den Ergebnissen der Intrusion Detection gezogen werden können. Dieses Kapitel versucht den Bogen von den allgemeinen Gegebenheiten in der Intrusion Detection über die konkrete Fragestellungen der Reaktion auf eine Intrusion bis hin zur Beschreibung des konkreten Bedrohungsszenarios eines Stepping Stones zu spannen und auf die wesentlichen Gesichtspunkte kurz einzugehen.

2.1 Intrusion

Bei der Beschäftigung mit der Thematik Intrusion Detection ist es zuerst einmal wichtig zu definieren, was eigentlich unter dem Begriff der Intrusion zu verstehen ist. Bei der Übersetzung ins Deutsche erhält man für den Begriff „Intrusion“ Begriffe wie „Eindringen“ und „Störung“. Insgesamt ist damit die Summe aller „unerlaubten, nicht autorisierten Eingriffe in die Informationssysteme“ [1] gemeint. Diese Eingriffe lassen sich nach Spenneberg grob in drei verschiedene Gebiete unterteilen: den Einbruch, den Mißbrauch und die Anomalie. In der Regel versteht man hier unter einem Einbruch einen, möglicherweise erfolgreichen, Angriff auf ein geschütztes System von Außen, wohingegen der Mißbrauch den selben Tatbestand von einem internen System bezeichnet. Eine Anomalie bezeichnet einen Vorgang der „nicht normal“ ist. Dazu muß man, um eine Anomalie diagnostizieren zu können, zuerst klar definieren, was als normal erachtet wird. Die Erkennung von Anomalien kann einem Administrator auf verschiedene Weisen helfen. Sie kann auf der einen Seite Hinweise auf Fehlkonfigurationen, z.B. im Netzwerkbereich, geben und auf der anderen Seite zeigt sich eine Anomalie, im Vergleich zum normalen Betrieb der Informationssysteme, oft als Folge eines oder als Hinweis auf einen Einbruch/Mißbrauch. Damit ist es möglich Einbrüche zu erkennen, die vom IDS nicht erkannt wurden, aber trotzdem einen nicht normalen Ablauf erzeugt haben.

2.2 Notwendigkeit

Wie die letzten Monate und Jahre immer deutlicher zeigen, ist das Gefährdungspotential für die Daten eines Unternehmens oder auch von Privatpersonen extrem angestiegen. Dabei reicht die Bandbreite der kostspieligen Bedrohung von Würmern wie Sasser [2] o.ä. über akuten Virenbefall bis hin zu unternehmenskritischen Varianten, denn welches Unternehmen kann es sich schon leisten, daß z.B. sein Webserver gehackt wird und dort möglicherweise diskriminierende Inhalte präsentiert werden.

Diese Art des Einbruches gehört aber noch zu den leichteren Varianten einer Intrusion, denn es entstehen durch die globale Vernetzung auch immer mehr Möglichkeiten von außen an geheime Dokumente eines Unternehmens zu gelangen oder gar gezielt diese Dokumente zu verändern, um der Konkurrenz zu schaden. Außerdem besteht hier sicherlich auch die Möglichkeit sich selbst, z.B. durch gezielte Manipulationen von Kontobewegungen, zu bereichern.

Heutzutage stellt es kein großes Problem dar unter Ausnutzung von Sicherheitslücken in diversen Programmen auf einfachste Weise in fremde Rechner einzudringen oder an ihnen sonstigen Schaden anzurichten. Ausgenutzt wird dabei oft die Unaufmerksamkeit der Benutzer der Informationssysteme. Die Fehler der Benutzer reichen dabei von viel zu einfachen Paßwörtern (oder gar ungeschützten Systemen), über das mangelhafte Einspielen von Sicherheitsupdates durch Fehleinschätzung der Lage oder schlicht Unwissenheit bis hin zu falsch konfigurierten Sicherheitssystemen.

Es ist einfach derartige Sicherheitslücken auszunutzen, weil man beim Durchstöbern des Internets ohne lange suchen zu müssen schon auf fertige Programme trifft, die die eine oder andere Sicherheitslücke ausnutzen, um denjenigen Code einzuschleusen/ auszuführen, den der Angreifer möchte. Zu den bekannteren Varianten dieser Tools gehören sicherlich Netbus und BackOrrifice [3]. Dabei ist es nur noch nötig mit Hilfe eines Portscanners potentiell verwundbare Systeme ausfindig zu machen und dann der Reihe nach, automatisiert, das fertige Tool auf diese Systeme anzusetzen.

Es gibt hier drei Strömungen zu beobachten, die alle nicht minder gefährlich sind. Auf der einen Seite gibt es eine ganze Menge Script-Kiddies die einfach ausprobieren wollen, wo sie überall eindringen können. In der Regel führen sie die Angriffe einfach nur zum Spaß aus und richten vergleichsweise geringen Schaden an. In letzter Zeit ist hier aber auch immer wieder eine Gefährdung dadurch aufgetreten, daß andere Angreifer durch Fehler in diesen Erst-Einbrüchen quasi Huckepack mit in die Systeme eindringen konnten und dort weiteren Schaden anrichteten. Bestes Beispiel hierfür ist Phatbot [4], der über mehrere verschiedene Wege, unter anderem im Gefolge von Sasser oder MyDoom, einen Computer infizieren kann. Auf der anderen Seite gibt es gezielte Angriffe auf einzelne Systeme, hinter denen eine kriminelle Intention steckt. Diese Angriffe reichen von Spionage, meist industrieller Natur, über Sabotage bis hin zu Betrug, o.ä. Die letzte Strömung liegt in der Mitte zwischen diesen beiden Extremen und ist sicherlich als äußerst gefährlich einzustufen. Hierbei handelt es sich um Angreifer, die eine bestimmte Weltanschauung mit ihren Aktionen zu Schau stellen wollen. Egal ob es ihnen darum geht zu beweisen, daß Windows-Systeme unsicher sind, oder eine andere Glaubensrichtung die vermeintlich bessere ist, ist es Ziel dieser Gruppe mit möglichst geringen Mitteln möglichst großen Schaden anzurichten.

Zieht man all diese Entwicklungen in Betracht, ist es klar, daß ein adäquater Schutz von Nöten ist, um der Bedrohung entgegen zu wirken. In vielen Unternehmen und Institutionen wird in diesem Bereich bereits mit Firewalls und Virenscannern ein großer Teil der Bedrohung abgewehrt. Zusätzlich dazu ist es aber, als weiteres Standbein der Sicherheit, dringend nötig mit Intrusion Detection darauf zu achten, ob neue Angriffe stattfinden, die durch die bereits bestehenden Systeme nicht oder nur ungenügend abgebildet sind.

So gesehen besteht also die Sicherheit aus zwei Bereichen, denen die Aufmerksamkeit der Administratoren gewidmet sein muß. Die Vermeidung von Angriffen ist der eine wichtige Punkt von beiden, denn es ist klar, daß durch alle möglichen Maßnahmen versucht werden muß, einen Einbruch, aber auch den Mißbrauch, schon im Vorfeld zu verhindern. Ohne die Intrusion Detection ist die Vermeidung aber nur ein stumpfes Schwert, da erst durch die Erkennung von Schwachstellen diese Sicherheitslücken gestopft werden können. Daher muß der Intrusion Detection, gerade im laufenden Betrieb, eine gehörige Menge an Aufmerksamkeit gewidmet werden, damit neu auftretende oder bisher nicht beachtete Probleme frühzeitig erkannt und behoben werden können.

Die Intrusion Detection muß darüber hinaus im laufenden Betrieb aber noch einen weiteren Aspekt verfolgen. Dieser wichtige Aspekt ist die Aufzeichnung und der Nachweis eines Einbruches zu späteren Verfolgung durch gesetzliche Behörden. Sollte ein Einbruch festgestellt werden, soll auf der einen Seite erst mal der Schuldige gefunden werden und auf der anderen Seite soll dieser Schuldige dann durch die gefundenen Beweise zur Rechenschaft gezogen werden können.

2.3 Intrusion Detection Systeme

Im Bereich der Intrusion Detection Systeme (IDS) unterscheidet man zwischen zwei großen Gruppen, die sich über kommerzielle aber auch frei verfügbar Systeme erstrecken. Auf der einen Seite stehen die sogenannten Network Intrusion Detection Systeme (NIDS), die versuchen anhand des Netzwerkverkehrs, der an ihnen vorbei oder durch sie hindurch fließt, zu erkennen, ob und wo eine Intrusion stattfindet und diese nötigenfalls zu protokollieren bzw. zu unterbinden. Auf der anderen Seite wird von den Host Intrusion Detection Systemen (HIDS) der Ansatz verfolgt, einen Angriff auf einen speziellen Computer zu erkennen und zu verhindern.

Die HIDS versuchen ihr Ziel durch Analysen von Logdateien, Integritätstest bzw. Echtzeitanalysen von System- und Dateizugriffen zu erreichen. Bei der Analyse der Logdateien können Positiv- oder Negativlisten definiert werden. Die Positivlisten dienen hierbei dazu, eine Liste der Ereignisse festzulegen, deren Eintreten einen Alarm auslösen soll. Im Gegensatz dazu bieten die Negativlisten eine Art Filter, welche Ereignisse keinen Alarm auslösen sollen. Alle Ereignisse, die nach der Filterung übrig bleiben, werden dann als Grund für einen Alarm gesehen. Integritätstest sind ebenfalls ein wichtiger Anhaltspunkt einer Kompromittierung eines Computers, da bei einem solchen Angriff meist neben dem eigentlichen Ziel das zusätzliche Ziel verfolgt wird diesen Computer erneut als Stepping Stone benutzen zu können oder auf dem Computer gar administrative Rechte zu erhalten, um die volle Kontrolle zu erhalten. Dafür werden oft Systemdateien verändert, so daß sie zusätzliche Funktionen erfüllen oder sogenannte Root-Kits installiert, die durch weitere Lücken die volle Kontrolle über den Computer bescheren.

Die Funktionsweise eine NIDS auf der anderen Seite setzt, wie bereits gesagt, auf der Analyse des Netzwerkverkehrs auf. Dabei werden meist verschiedene Technologien eingesetzt um aus dem Verkehr mögliche Angriffe zu extrahieren. Die einfachste Art und Weise ist die Signatur-Erkennung, bei der in einer Datenbank alle möglichen bekannten Angriffe mit ihrer Paket-Signatur hinterlegt sind. Tritt im Verkehr eine derartige Sequenz auf, dann kann auf einen Angriff geschlossen werden. Hat der Angreifer Kenntnis vom Einsatz eines Signaturvergleiches kann er allerdings eine Strategie zur Umgehung der Erkennung anwenden, die das NIDS überlastet, bzw. den eigentlichen Angriff verschleiert. Als Konsequenz daraus gibt es die zustandsorientierte Signatur-Erkennung die neben Signaturvergleichen auch den Zustand der jeweiligen Verbindungen berücksichtigt. Zusätzlich dazu ist es in einigen Punkten nötig vorher eine Protokolldekodierung durchzuführen, um an die eigentlichen Daten, unabhängig von ihrer Kodierung, zu kommen und so eine legale von einer illegalen Nutzung zu unterscheiden. Da diese Technologien recht aufwendig sind und beispielsweise auch nicht in der Lage sind eine Häufung von Portscans zu erkennen kommen zusätzlich zu den genannten Technologien noch statistische und heuristische Verfahren zum Einsatz um aus dem Netzwerkverkehr Rückschlüsse auf unerwünschte Tätigkeiten zu ziehen.

Bei allen IDS ist die Notwendigkeit der Reaktion gegeben, wenn eine Intrusion festgestellt worden ist. Die Reaktionen bilden ein breites Spektrum der Möglichkeiten ab, die je nach dem als wie schwerwiegend die Intrusion betrachtet wird variieren können. Gemeinsame Reaktionen sind beispielsweise Einträge in speziellen Logdateien, Versand von Benachrichtigungen via Email oder SMS, sowie das Auslösen von SNMP-Traps an einer weiteren Monitoring-Stelle. Ein NIDS bietet darüber hinaus weitere Möglichkeiten um einen größeren Schaden zu verhindern. Hierfür ist es bei einigen NIDS möglich zur Laufzeit Firewall-Regeln so zu verändern, daß entweder der spezifische Rechner des Angreifers für eine gewisse Zeit geblockt wird, oder gar den Zugriff auf den angegriffenen Rechner komplett zu sperren. Außerdem kann das NIDS gezielt Verbindungen beenden, die als Intrusion erkannt wurden, um so die weitere Durchführung des Angriffes zu unterbinden..

2.4 Fehlertypen der Erkennung

Allen Intrusion Detection System ist gemein, daß ihre Erkennung natürlich nicht perfekt ist. Zwar wird versucht eine möglichst perfekte Erkennung von Angriffen zu erreichen, aber es kann, gerade beim Einsatz von statistischen und heuristischen Methoden, nie ganz ausgeschlossen werden, daß nicht doch Angriffe unerkannt bleiben, oder eine korrekte Nutzung als Angriff gewertet wird.

Als False-Positive wird in der Literatur ein, im Sinne der Erkennung, fälschlicherweise als positiv bewerteter Fakt bezeichnet. Auf die Intrusion Detection angewendet bedeutet das, daß ein Alarm ausgelöst wurde, obwohl keine Intrusion vorlag. Im Gegenzug dazu bezeichnet ein False-Negative die Tatsache, daß ein Fakt nicht als positiv bewertet wurde, obwohl er als positiv bewertet hätte werden müssen. Es wurde also bei einer tatsächlichen Intrusion kein Alarm ausgelöst.

In der Regel kann man aber bei der Erkennung nicht die Rate beider auf Null senken, da sie in gewisser Weise voneinander abhängen. Je mehr False-Positives man durch Veränderungen ausmerzt, um so höher ist das Risiko False-Negatives zu bekommen, die man nicht bemerken kann. Auf der anderen Seite wäre es jetzt naheliegend zu sagen, daß man einfach die Schwelle der Erkennung niedrig hält, um möglichst keine False-Negatives zu bekommen. Dies führt aber dazu, daß sehr viele Alarme ausgelöst werden, die von Hand nachgeprüft werden müssen. Die Flut der Alarme sorgt dann dafür, daß hohe Kosten für die Nachbearbeitung entstehen, bzw. die Nachbearbeitung der Alarme schlampig durchgeführt wird oder gar unterbleibt. Somit würden in dieser Flut auch alle tatsächlichen Alarme untergehen und übersehen werden.

Die Bewertung wie schwerwiegend diese beiden Fehlertypen sind ist nicht trivial, da eine Verschiebung hin zu beiden Extremen dazu führt, daß die Intrusion Detection wirkungslos wird. Daher gilt es das fragile Gleichgewicht möglichst ideal zu finden, was nur durch Erfahrung und eine fortwährende Überwachung und Anpassung des IDS zu erreichen ist.

2.5 Reale Intrusion Detection

In der bisherigen Arbeit wurde beschrieben, warum Intrusion Detection betrieben werden sollte und welche Vor-/Nachteile die eine oder andere Möglichkeit bietet. Nun stellt sich die Frage wie man die Intrusion Detection in der realen Welt vorfindet oder sie auch selbst einsetzen kann. Es gibt in diesem Bereich kommerzielle Produkte, deren Kosten schnell den fünfstelligen Dollarbereich erreichen und so gerade für kleinere Einrichtungen finanziell nicht relevant sind. Die OpenSource-Gemeinde bietet aber

auch eine gute Anlaufstelle für kostenfreie Alternativen. Eine Frage beim Einsatz von OpenSource-Tools ist, in wie weit diese Alternativen die Qualität der kommerziellen Produkte erreichen. Allein deren Verbreitung und die Aktualisierungszyklen der Signaturen sprechen allerdings nicht dafür, daß die freien Alternativen schlechter sind als kommerzielle Varianten. Ein weiterer Kritikpunkt an den OpenSource-Tools ist, daß der Angreifer durch den Zugriff auf den Sourcecode der Tools die Möglichkeit hat, das System auf Schwachstellen zu untersuchen und sein Handeln dahingehend zu optimieren, daß er beim Angriff diese Schwachstellen ausnutzen kann. Dem kann man sicherlich mit dem Standardargument der OpenSource-Gemeinde entgegentreten, daß diese Schwachstellen bei der Menge der beteiligten Entwickler allerdings jedoch schneller gefunden und behoben werden, als das bei kommerzieller Software üblich ist.

Darüber hinaus steht und fällt aber die Möglichkeit eines Angreifers eine Schwachstelle ausnutzen zu können mit der Methodik des Erkennungsalgorithmus, die sowohl bei den OpenSource-Tools als auch bei den kommerziellen Produkten ähnlich ist.

Wie Ralf Spenneberg in seinem Buch schreibt sind Ziele eines IDS-Systems [5]:

- Verfügbarkeit der bereitgestellten Dienste
- Integrität und Verfügbarkeit der bereitgestellten Daten
- Unterstützung in der Reaktion auf einen Angriff
- Fehlertoleranz
- Kontinuierlicher Lauf ohne wesentliche Administration
- Erzeugung minimalen Overheads
- Leichte Integration in die vorhandene Struktur
- Hohe Intelligenz, die ein Umgehen möglichst unmöglich macht

Bei den IDS stellt momentan noch keines der Produkte alle Punkte perfekt zur Verfügung. Vielmehr konzentrieren sich die einzelnen Produkte auf wenige Punkte, die sie sehr gut machen, während sie andere Punkte sehr vernachlässigen. Jedes der verfügbaren Systeme, egal ob frei oder nicht, hat also seine Stärken und Schwächen, die man kennen sollte, um den zu der Situation am Besten passenden Kompromiß zwischen den aufgeführten Kriterien zu finden.

Im Zusammenhang mit der Intrusion Detection gilt es zwei rechtliche Aspekte zu beachten, die hier lediglich kurz beschrieben werden sollen, damit beim Einsatz der Intrusion Detection klar wird, auf was geachtet werden sollte. Ein, in unserer Gesetzgebung deutlich formulierter, Aspekt ist der Schutz der Privatsphäre. Diese Privatsphäre wird durch diverse Gesetze zum Datenschutz derart geregelt, daß die gewonnenen Daten lediglich in einer vollständigen Form gespeichert werden dürfen, wenn die Person (in diesem Falle begrenzt auf natürliche Personen) ihre Einwilligung dazu gegeben hat. Andernfalls dürfen die Daten nur anonymisiert gespeichert werden, so daß eine Rückverfolgung zu der natürlichen Person nur unter großen Aufwand möglich ist. Diesen Fakt sollte man beim Einsatz eines IDS immer im Hinterkopf behalten, weil durch das IDS möglicherweise sensible Daten aufgezeichnet werden, die in dieser Form nicht gespeichert werden dürfen und somit eine andere Art der Speicherung gefunden werden muß. Der zweite wichtige Punkt ist die Tatsache, daß die gesammelten Daten eines IDS möglicherweise als Beweismittel verwendet werden

sollen. Wurde durch das IDS eine Straftat aufgezeichnet, so müssen die Daten so gespeichert werden, daß sichergestellt ist, daß die Daten nicht modifiziert worden sind.

Im realen Einsatz ist es darüber hinaus wichtig, sich schon im Vorfeld darüber Gedanken zu machen, wie im Falle eines Angriffes zu verfahren ist. Dieser gefaßte Notfallplan muß einen detaillierten Ablauf enthalten, wer zu welchem Zeitpunkt welche Konsequenzen ergreift. Der Notfallplan sollte damit beginnen, daß formuliert wird, welche Gefahrenstufen vom IDS erkannt werden und wie das IDS auf die einzelnen Stufen reagieren soll, d.h. es muß festgehalten werden, ab welcher Gefahrenstufe welcher Mitarbeiter auf welchem Wege davon Kenntnis erhält und welche Sofortmaßnahmen durch das IDS und die Mitarbeiter erfolgen müssen. Die nächsten Schritte, die der Notfallplan enthalten muß, sind Regeln zur „forensischen Analyse“ des Angriffes, Wiederherstellungsanweisungen für das angegriffene System und weitere Maßnahmen wie Benachrichtigung der Entscheidungsträger oder die Strafverfolgung. Darüber hinaus sollte der Notfallplan auch enthalten, wie bei der Erkennung eines neuen Angriffes verfahren werden soll, damit bei der Wiederinbetriebnahme des Computers nicht ein erneuter gleicherartiger Angriff möglich ist.

Eine der wichtigsten Entscheidungen ob ein IDS zum Einsatz kommen soll, ist sicherlich die Frage, ob es sich finanziell überhaupt lohnt. Oft werden hier ausschließlich die Kosten für die Implementierung und die Wartung eines IDS gesehen, ohne sich Gedanken darüber zu machen ob und wie viel Geld dadurch gespart werden, daß das IDS einen, zwar zeitlich begrenzten, Totalausfall eines, oder mehrer, Computer verhindert.

2.6 Stepping Stones

Nachdem in der bisherigen Arbeit zunächst einige allgemeine Informationen zur Intrusion Detection gegeben worden sind, soll im Folgenden eine spezielle Art der Intrusion beleuchtet werden, nämlich die Benutzung eines Computers als Stepping Stone. Im weiteren Verlauf werden dann Methoden zur Erkennung von Stepping Stones vorgestellt und bewertet.

Im Internet ist es für einen Angreifer ein leichtes seine Identität zu verschleiern oder zumindest einer Strafverfolgung weitestgehend zu entgehen, in dem er dafür sorgt, daß der Angriff nicht zu ihm zurückverfolgt werden kann. Jetzt stellt sich die Frage, wie ein Angreifer es anstellt, daß er nicht für seinen Angriff verantwortlich gemacht werden kann, wenn, zumindest an den neuralgischen Stellen, jeder Zugriff aufgezeichnet oder zumindest mit seiner Herkunft verzeichnet wird. Genau hier kommt der Begriff des Stepping Stones ins Spiel, der es einem potentiellen Angreifer ermöglicht als jemand anderes zu erscheinen als er tatsächlich ist.

Man spricht von einem Computer als Stepping Stone, wenn er von einer Person dazu benutzt wird, um von dort aus Dienste eines anderen Computers in Anspruch zu nehmen. In der Regel werden Stepping Stones im Bereich von interaktiven Benutzersessions verwendet, um eben auf einem weiteren Computer beliebige Verarbeitungen durchzuführen. An dieser Stelle ist es sinnvoll, noch wenig tiefer ins Detail zu gehen. Genau genommen ist ein Computer ein Stepping Stone wenn er zwei Verbindungen, eine kommende und eine abgehende, aufweist, deren Inhalt gleich ist. Also deren Daten genauso wieder abgesendet werden, wie sie angekommen sind. Im Rahmen dieser Arbeit wird nur auf die Verwendung im interaktiven Verkehr eingegangen, weil dort ein großes Gefährdungspotential zu sehen ist und es dafür

Algorithmen zur Erkennung gibt. Daraus resultiert auch die Tatsache, daß im weiteren nur TCP-Verbindungen betrachtet werden.

Computer werden täglich von vielen Menschen auf ganz normale und auch legale Weise als Stepping Stone benutzt. Oft ist es so, daß lediglich ein oder zwei Computer eines Unternehmens oder einer Organisation im Internet zu erreichen sind. Diese Beschränkung dient im Allgemeinen dem Schutz des Unternehmens, da so nur wenige Computer auf Angriffe hin überwacht werden müssen. Muß nun ein Mitarbeiter auf einen Computer im internen Netzwerk des Unternehmens zugreifen, wird er, wenn die Möglichkeit dazu besteht, einen der erreichbaren Computer als Stepping Stone verwenden, um von dort aus auf einen der internen Computer zuzugreifen.

Bei der illegalen Benutzung eines Computers als Stepping Stone wird nun dieser dazwischen liegende Computer als Schutzmechanismus des Angreifers benutzt. Für den eigentlichen Zielrechner des Angriffes sieht es so aus, als würde der Angriff vom dazwischen liegenden Computer kommen, anstatt vom Computer des Angreifers. Diese Art der Verschleierung der Herkunft kann natürlich beliebig kaskadiert werden, um einen immer größeren Abstand zwischen der Quelle des Angriffes und dem eigentlichen Ziel zu schaffen. Insbesondere, wenn der Angreifer über Computer in verschiedenen Ländern geht, wird es für einen Geschädigten extrem schwierig, auch nur die IP-Adresse des Angreifers zu ermitteln, geschweige denn rechtliche Schritte gegen den Angreifer einzuleiten und das mögliche Urteil auch durchzusetzen.

Die Wirkung der Verschleierung wird natürlich noch dadurch erhöht, daß der Angreifer in der Regel bei seinem Angriff nicht seine offiziellen Logins auf den verschiedenen, als Stepping Stone genutzten, Maschinen benutzen wird, sondern sich vorher auf anderen Wegen Logins von realen oder imaginären Benutzern beschaffen wird. Dadurch wird die tatsächliche Ermittlung einer Person noch weiter erschwert.

Oft bauen sich Angreifer so ein immer größeres Netzwerk von Computern auf, die ihnen dazu dienen, ihre Person zu verschleiern. Aus diesem Netzwerk wählen sie immer neue Kombinationen von Stepping-Stone-Ketten aus, damit die Rückverfolgung wesentlich erschwert wird. Jeder neu hinzugekommene, kompromittierte Computer potenziert so die Möglichkeiten der Verschleierung.

Durch dieses Gefährdungspotential ist es wichtig, die Nutzung von Stepping Stones frühzeitig zu erkennen und möglicherweise dagegen vorzugehen. Im Bereich der IDS wird versucht die Erkennung von Stepping Stones durch verschiedene Ansätze zu erreichen, die im weiteren genauer vorgestellt und bewertet werden. Die große Schwierigkeit liegt aber eher, darin die legalen von den illegalen Stepping Stones zu unterscheiden, da ja wie beschrieben täglich viele Benutzer auf die Verwendung verschiedener Computer Stepping Stone angewiesen sind, um ihre Arbeit zu erledigen. Hierfür muß jede Stelle, die eine solche Erkennung einsetzen möchte, ihre eigenen Regeln aufstellen, nach denen unterschieden werden kann, was gewünscht und was unerwünscht ist.

3 Algorithmen zur Erkennung

Im folgenden Kapitel werden die Algorithmen kurz erklärt, die momentan in der Literatur zu finden sind. Die vorliegenden Algorithmen versuchen auf unterschiedlichsten Wegen eine Erkennung von Stepping Stones möglich zu machen. Dabei werden von trivialen Methoden wie der Zählung von Paketen oder der Analyse von Paketinhalten bis hin zu komplexen Methoden wie der Wavelet Analyse Gebrauch gemacht. Eine genauere Beschreibung der ausgewählten Algorithmen erfolgt dann in Kapitel 4.

3.1 Vergleichskriterien

Damit die einzelnen Algorithmen miteinander verglichen werden können, gilt es eine Reihe von Kriterien zu definieren, mit denen eine Vorauswahl getroffen werden kann. Diese gewählten Kriterien sind implementierungsspezifische Details, die auf einen Algorithmus entweder zutreffen oder nicht zutreffen. Dadurch können bereits im Vorfeld einige Algorithmen ausgeschlossen werden, da sie sich nicht für den Einsatz im breitbandigen Umfeld eignen. Weiterhin gibt es einige Kriterien, die später einen qualitativen Vergleich der getesteten Algorithmen erlauben.

3.1.1 Unterscheidungsmerkmale

INHALTS- UND TIMINGBASIERTE ANALYSE

Ein Hauptunterschied in den Analyseformen besteht hinsichtlich der zur Erkennung benötigten Daten. Zur Erkennung kann einerseits der Inhalt der Pakete als Vergleichswerkzeug eingesetzt werden, andererseits kann man die Analyse auch auf dem zeitlichen Ablauf der Verbindung aufbauen. Dieser auf den ersten Blick triviale Ansatz den Inhalt zu vergleichen, spielt allerdings in den beschriebenen Algorithmen keine Rolle. Er spielt hier hauptsächlich keine Rolle, weil er, beispielsweise durch verschlüsselte Verbindungen, leicht auszuhebeln ist und außerdem im breitbandigen Umfeld die Wahrscheinlichkeit ähnlicher Paketsequenzen im unverschlüsselten Verkehr zu einer hohen Rate an false-positives führen würde. Daher wird bei den Algorithmen versucht, die Korrelation zwischen zwei Strömen über die zeitliche Abfolge ihrer Pakete nachzuweisen. Die timingbasierte Analyse bietet auch den Vorteil, daß weit weniger Daten analysiert werden müssen, da man für die timingbasierte Analyse mit den Paketheadern auskommt und nicht die kompletten Pakete benötigt.

PASSIVE UND AKTIVE ANALYSE

Gerade wenn man breitbandigen Netzwerkverkehr untersuchen will, ist es von großer Relevanz, ob man aktiv in den Netzwerkverkehr eingreifen muß, um einen Angriff feststellen zu können oder ob es reicht, wenn der vorhandene Verkehr analysiert wird. Ein aktiver Eingriff hat gleich mehrere Nachteile. Zum einen erhöht der Eingriff in den Verkehr, z.B. in das Timing der Pakete, die Möglichkeit, daß die Analyse durch den Angreifer entdeckt wird und er Gegenmaßnahmen ergreift. Zum anderen kann der Eingriff auch die legale Nutzung der Dienste negativ beeinflussen oder gar unnutzbar machen, weil beispielsweise zusätzliche Pakete eingefügt werden oder der Durchfluß der Pakete durch Timingveränderungen gehemmt wird.

REALTIME- UND OFFLINEANALYSE

Ein wichtiger Unterschied zwischen den einzelnen Algorithmen ist, in welchem zeitlichen Umfeld die Analyse erfolgt. Hierbei gilt zwischen einer Realtimeanalyse und eine Offlineanalyse zu unterscheiden. Bei einer Realtimeanalyse kann, oder muß, die Analyse des Netzwerkverkehrs passieren, während der Zugriff stattfindet. Möchte man eine aktive Analyse einsetzen, gibt es keine andere Möglichkeit als die Analyse in Realtime durchzuführen. Ein Vorteil einer Realtimeanalyse ist aber auch eine zeitnahe Erkennung von Angriffsszenarien und einer daraus resultierenden schnelleren Reaktionsmöglichkeit auf das Ereignis.

ANZAHL DER MEßPUNKTE

Bei der Gewinnung der Analysedaten kann es je nach Netzwerktopologie einen oder mehrere Meßpunkte geben, an denen der Netzwerkverkehr aufgenommen bzw. analysiert wird. Generell ist es problematisch, wie man mit der Analyse an mehreren Punkten umgehen soll, weil hierbei auf unterschiedliche Delays verschiedener Verbindungen zu achten ist und es das Problem der Synchronisation der Uhren der aufzeichnenden Stellen zu meistern gilt.

3.1.2 Qualitative Merkmale

ERKENNUNGSRATE

Ein wichtiges Vergleichskriterium ist sicherlich die Erkennungsrate der Algorithmen. Die wirkliche Erkennungsrate kann durch zwei Faktoren negativ beeinflußt werden. Einerseits gilt es die sogenannten False-Positives zu vermeiden. Unter False-Positives versteht man allgemein Vorfälle, die als unerwünscht erkannt werden, obwohl sie nicht unerwünscht sind. Treten hiervon zu viele auf, ist der administrative Aufwand der Überprüfung unerwünschter Vorfälle zu groß, was gemeinhin dazu führt, daß die Überprüfung schlampig bis gar nicht stattfindet. Andererseits gilt es ebenso die sogenannten False-Negatives zu vermeiden. False-Negatives sind Vorfälle, die unerwünscht sind, allerdings nicht als solche erkannt werden.

GESCHWINDIGKEIT

Gerade bei der Analyse von großen Mengen Netzwerkverkehrs ist die Geschwindigkeit der Algorithmen von großer Bedeutung. Idealerweise wird der Vorfall bereits während er stattfindet, oder zumindest kurz danach entdeckt, um sowohl eine Sicherstellung etwaiger Logdateien als auch eine Minimierung des Schadens zu erreichen. Beispielsweise nützt es wenig, wenn ein unerwünschter Vorfall erst nach 3-wöchiger Analyse als solcher erkannt wird.

SPEICHERVERBRAUCH

Ein elementares Kriterium für den Vergleich ist der Speicherverbrauch. Belegt hier ein Algorithmus zu viel Speicher für die benötigten Daten, so kann das sehr schnell dazu führen, daß der auf dem Erkennungscomputer verfügbare Hauptspeicher volläuft und so das Programm zum Absturz gebracht wird.

3.2 Umgehung der Erkennung

Da im folgenden Methoden beschrieben werden, mit denen die Nutzung als Stepping Stone erkannt werden kann, scheint es sinnvoll, im Vorfeld schon darüber

nachzudenken, welche Möglichkeiten ein Angreifer hat, um der Erkennung zu entgehen. Bei den einzelnen Algorithmen wird näher darauf eingegangen, warum eine Umgehung möglich wird. Im Wesentlichen gibt es zwei Varianten die hier beschriebenen timingbasierten Algorithmen zu umgehen. Die beiden Varianten verändern beide den zeitlichen Ablauf einer der beiden Verbindungen, die an einem Stepping Stone beteiligt sind. Die erste Möglichkeit besteht darin, bei einer der beiden Verbindung die ankommenden und abgehenden Pakete für einen gewissen Zeitraum zurückzuhalten, bevor sie versandt werden. Diese Methode funktioniert um so besser, wenn dieses Zurückhalten für zufällige Zeiträume geschieht. Die zweite Variante besteht darin, daß in einen der Ströme sogenannte Chaff-Pakete als Tarnung eingeführt werden. Diese Pakete sind so beschaffen, daß sie die gewünschte Benutzung nicht beeinträchtigen, weil sie automatisch ausgefiltert werden. Auch hier wird eine bessere Tarnung erreicht, wenn die Pakete zufällig eingestreut werden. Dabei wird man auch eine sichere Methode erreichen können, eine Erkennung komplett zu umgehen, wenn man die Nutzdaten komplett in einen Strom einbettet, der eine gleichförmige Struktur hat. Werden die Nutzpakete so eingebettet, daß sie statt der dauerhaften Chaff-Pakete übertragen werden, fällt jegliche Timingcharakteristik des einen Stromes weg, und es wird unmöglich die Korrelation zu erkennen.

3.3 Algorithmen

Im folgenden Abschnitt werden die einzelnen Algorithmen mit ihren Funktionsprinzipien erläutert. Es wird für die Algorithmen eine Einordnung in die Kriterien gegeben und eine Abschätzung getroffen, wie sich die Algorithmen im Einsatz verhalten werden.

3.3.1 ON/OFF-Algorithmus

Der hier beschriebene On/Off-Algorithmus wurde von Yin Zhang und Vern Paxson in ihrem Paper „Detecting Stepping Stones“ [6] vorgestellt. Im Paper wird die Grundidee verfolgt, daß anhand der Nutzung der Verbindung eine Charakteristik hergestellt werden kann, die für jede Verbindung annähernd einzigartig ist.

Der Algorithmus basiert auf einer Analyse des Timings der übertragenen Netzwerkpakete. Damit ist der Algorithmus unabhängig vom Inhalt der Pakete und kann daher auch verschlüsselte Verbindungen erkennen. Für diesen Algorithmus besteht keine Einschränkung hinsichtlich einer Realtime- oder Offlineanalyse. Er ist in beiden Varianten gleichermaßen zur Erkennung einsetzbar. In Bezug auf die Anzahl der Meßpunkte ist der Algorithmus insofern kritisch, als daß er sich auf die exakten Paketankunftszeiten verläßt. Dies bedeutet, daß eine Erkennung mit mehreren Meßpunkten ausschließlich dann zuverlässig funktionieren kann, wenn die Uhren der Meßpunkte gleich laufen und die aufgezeichneten Pakete dann in der richtigen zeitlichen Reihenfolge an den Algorithmus übergeben werden.

Der Algorithmus ist aus Sicht des Ressourcenverbrauches recht genügsam, da er weder große Datenmengen im Speicher vorhalten muß, um seine Analyse durchzuführen, noch in den einzelnen Schritten sehr komplizierte Berechnungen durchführen muß. Es muß lediglich eine Liste der aktiven Verbindungen verwaltet werden in der der Beginn der letzten OFF-Phase vermerkt wird und ein Zähler für das gemeinsame Auftreten von OFF-Phasen zweier Verbindungen sowie je Verbindung ein Zähler für die Anzahl der OFF-Phasen geführt werden. Diese Daten können nach Beendigung der Verbindung freigegeben werden.

Basierend auf der Beobachtung, daß bei interaktivem Verkehr (hierzu zählen beispielsweise Telnet-Session oder SSH-Sessions) charakteristisch unterschiedliche Phasen der Aktivität zu verzeichnen sind, wurde der Algorithmus so entworfen, daß er daraus eine Korrelation herstellen kann. Ist in einer Netzwerkverbindung für einen gewissen Zeitraum T_{idle} kein Paket zu verzeichnen sprechen wir von einer OFF-Phase. Dabei kommt ein Paket für diese Betrachtung nur in Frage, wenn es sich tatsächlich um ein Paket handelt, das noch nicht bearbeitet wurde. Noch nicht bearbeitet bedeutet in diesem Zusammenhang, daß das Paket weder neu übertragen wurde (erkennbar an der Sequenznummer des Pakets), noch daß es sich um ein keep-alive Paket handelt. Sobald in einem Strom wieder ein Paket, das diese Vorgabe erfüllt, auftritt beginnt für diesen Strom eine ON-Phase.

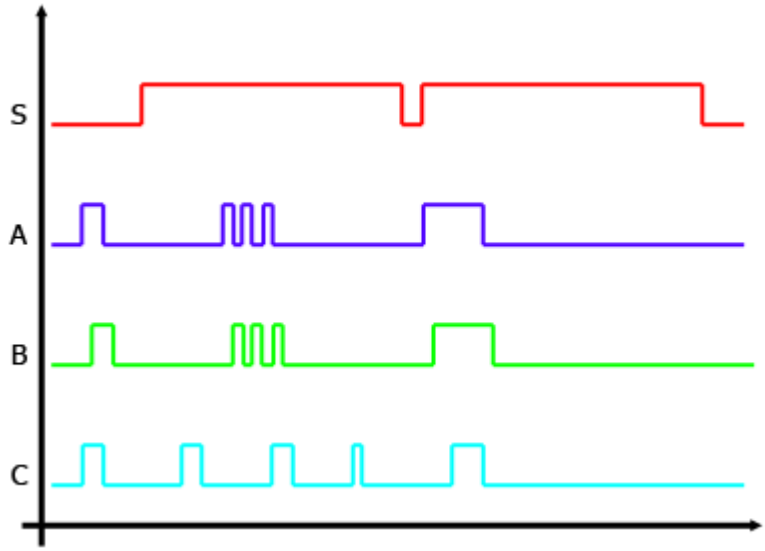

Abbildung 1: Exemplarische Netzwerkströme

In der Abbildung 1 wurden exemplarisch zeitliche Verläufe verschiedener Ausprägung dargestellt. Hierbei stellt der Strom S einen Serverprozess dar, der keine wirklich prägnanten Merkmale enthält. Die beiden Ströme A und B zeigen eine deutlich übereinstimmende Charakteristik einer interaktiven Session. Der Strom C beginnt zwar nahezu gleichzeitig mit den Prozessen A und B und hört auch zur selben Zeit auf, aber im Mittelteil ist eine deutliche Abweichung zu erkennen, aus der sich schließen läßt, daß die Ströme nicht miteinander korrelieren.

Diese Charakteristik der ON/OFF Phasen ist, wie zu erkennen, insbesondere bei interaktivem Verkehr sehr ausgeprägt, da sich in ihr deutlich die Abstände zwischen zwei Zeichen beim Tippen und auch „Denkpausen“ des Nutzers widerspiegeln. In Internet-Studien wurde hierzu festgestellt (zitiert nach [6]), daß 25% aller Tastenanschläge mit einem zeitlichen Abstand von 500 Millisekunden oder mehr eintreffen. 15% der Tastenanschläge sogar mehr als 1 Sekunde auseinander. Im Gegensatz dazu ist Verkehr, der von zwei Serverprozessen ausgetauscht wird nicht dieser Charakteristik unterworfen, da hier meist auf eine kurze Anfrage der eine Seite eine schnelle und gebündelte Antwort der anderen Seite folgt, bevor die Verbindung wieder abgebaut wird.

Daraus läßt sich die Annahme herleiten, daß zusammengehörige interaktive Netzwerkströme in etwa zu ähnlicher Zeit ihre OFF-Phase verlassen, wenn beispielsweise eine Taste gedrückt wurde, oder ein Programm fertig gelaufen ist, und seine Antwort schickt. Die sich daraus ergebenden Gemeinsamkeiten müssen jetzt lediglich noch quantifiziert werden, um dies in einem Algorithmus implementieren zu können.

Dazu werden zwei Kontrollparameter δ und γ eingeführt, die einen direkten Einfluß auf die Erkennungsrate haben. Der Parameter δ steht hierbei für den Zeitunterschied zwischen dem Anfang einer OFF-Phase zweier Ströme. Die OFF-Phasen werden als korrelierend betrachtet, wenn dieser Zeitunterschied kleiner als δ ist. Mit dem Parameter δ können nun korrelierende OFF-Phasen erkannt und gezählt werden.

Um aus einer Reihe, möglicherweise zufälliger, Gemeinsamkeiten zweier Ströme diejenigen herauszufiltern, die wirklich miteinander in Verbindung stehen, wird der Prozentsatz der korrelierenden OFF-Phasen in bezug auf die aufgetretenen OFF-Phasen gebildet, in dem die Anzahl der gemeinsamen OFF-Phasen ($OFF_{1,2}$) durch die minimale Zahl der OFF-Phasen in den einzelnen Strömen (OFF_1, OFF_2) dividiert wird.

$$\frac{OFF_{1,2}}{\min(OFF_1, OFF_2)} \geq \gamma$$

Übersteigt dieser Wert den variablen Schwellwert γ, dann werden die beiden Ströme als Stepping Stone gewertet.

Um die Rate der False-Positives zu minimieren wurde noch ein daraus abgeleitetes Kriterium hinzugefügt. Hierzu wurde der Algorithmus so angepaßt, daß bei der Berechnung zusätzlich aufeinanderfolgende Gemeinsamkeiten zweier Ströme betrachtet werden. Es werden also die Vorkommen von Gemeinsamkeiten gezählt ($OFF^*_{1,2}$), bei denen mindestens min_{csc} OFF-Phasen hintereinander miteinander korrelieren. Diese Verfeinerung führt dann zu folgender Formel.

$$\frac{OFF^*_{1,2}}{\min(OFF_1, OFF_2)} \geq \gamma'$$

Der deutlich niedrigere Schwellwert γ' muß für die Markierung als Stepping Stone ebenfalls überschritten werden. Damit wurde das Problem behoben, daß lange Verbindungen die Anzahl der zufälligen Gemeinsamkeiten, die über die lange Zeit aufsummiert wurden, False-Positives erzeugt haben.

Der Algorithmus wurde von Paxson/Zhang im Rahmen ihrer Arbeit implementiert und mit von ihnen gesammelten kleinen Netzwerktraces verifiziert. Die Traces stammten vom Lawrence Berkeley National Laboratory (LBNL) und der University of Columbia at Berkeley (UCB) enthielten zwischen 3831 Telnet/Rlogin-Verbindungen (120MB, 1.5 Millionen Pakete) und 7319 Telnet/Rlogin-Verbindungen (390MB, 5 Millionen Pakete). Bei der Verifikation wurden für die Parameter geeignete Werte gefunden, die die Anzahl der False-Positives auf Null gesenkt haben, bei gleichzeitiger Minimierung der False-Negatives.

Eine Erkennung durch diesen Algorithmus läßt sich von Seiten des Angreifers mit relativ leichten Mitteln entgegenwirken. Der Angreifer muß dafür lediglich das Timing einer der beiden beteiligten Verbindungen verändern, in dem er beispielsweise Chaff-

Pakete in die Verbindung einstreut oder Pakete für einen gewissen Zeitraum zurückhält, damit die OFF-Phasen nicht mehr korrelieren.

Da dieser Algorithmus bereits implementiert und verifiziert worden ist, wird er im Vergleich als Referenz dienen. Der Algorithmus hat einige Schwachstellen in Bezug auf die Umgehbarkeit der Erkennung, die von den im weiteren vorgestellten Algorithmen verbessert werden sollen. Die weiteren Algorithmen müssen allerdings erst unter Beweis stellen, daß sie alle Vorfälle erkennen, die auch vom ON/OFF Algorithmus erkannt werden und darüber hinaus auch Vorfälle korrekt erkennen, die diesen Algorithmus durch seine Schwachstellen umgehen können.

3.3.2 Watermarking Algorithmus

Der Watermarking Algorithmus wurde von Xinyuan Wang und Douglas S. Reeves in ihrem Paper „Robust Correlation of Encrypted Attack Traffic Through Stepping Stones by Manipulation of Inter-Packet Delays" [7] vorgestellt. Der Algorithmus versucht einer Verbindung eine eindeutig wiedererkennbare Signatur anzuhängen, die später zur Erkennung des Stepping Stones führen soll.

Zur Erkennung verwendet der Algorithmus eine Methode mit aktivem Eingriff in das Paket Timing des Netzwerkverkehrs. Wegen dieser aktiven Charakteristik des Algorithmus kann er ausschließlich als Realtimeanalyse durchgeführt werden. Problemlos ist es möglich, das Einbetten des Wasserzeichens und die anschließende Erkennung an zwei unterschiedlichen Meßpunkten durchzuführen, sofern die erkennende Station weiß, welche Wasserzeichen verwendet worden sind oder wie die Wasserzeichen logisch aufgebaut sind. Nachdem für die Einbettung und die Erkennung kaum Daten vorgehalten werden müssen, ist der Speicherverbrauch dieses Algorithmus minimal. Die Auslastung der CPU ist für diesen Algorithmus bei steigender Anzahl der gleichzeitigen Verbindungen nicht zu unterschätzen, allerdings steigt sie nur linear an.

Bei der Erkennung wird versucht in den Abstand zwischen den einzelnen Paketen eines Stroms (Inter-Packet Delay) ein Wasserzeichen, bestehend aus einer variablen Anzahl an Bits, einzubetten, welches auch nach mehreren dazwischen liegenden Stepping Stones und Abwehrmaßnahmen seitens des Angreifers immer noch zu erkennen ist. Dazu werden als erstes die gemessenen Abstände zwischen den Paketen mittels einer gegebenen Schrittweite *s* quantifiziert, so daß der Abstand eindeutig einer ganzen Zahl zugeordnet werden kann. Ein gesetztes Bit (in der Abbildung 2 das Bit *w*) wird jetzt in den Abstand zwischen den zwei Paketen eingebettet, in dem der Abstand so weit verlängert wird, daß bei der anschließenden Dekodierung die o.g. Quantifizierung einen ungeraden Zahlenwert ergibt. Ist der Zahlenwert bereits ungerade wird der Abstand nicht verändert. Ein nicht gesetztes Bit wird dementsprechend so eingebettet, daß bei der Dekodierung eine gerade Zahl herauskommt.

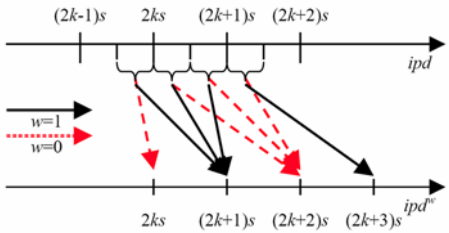

Abbildung 2: Einbettung eines Bits w [8]

Um eine größere Robustheit gegenüber einer zeitlichen Veränderung durch den Angreifer zu erlangen, wird ein Bit nicht nur in einen Abstand eingebettet, sondern in den Mittelwert mehrerer Pakete. Durch die Mittelwertbildung wird erreicht, daß eine Veränderung der Zeit um mehr als *s/2* bei wenigen Paketen nicht dazu führt, daß die Erkennung verhindert wird, weil davon auszugehen ist, daß ein Angreifer eher wenige Pakete lange verzögern wird, um einer Erkennung zu entgehen, anstatt alle Pakete lange zu verzögern.

Diese Einbettung kann nun auf einfache Weise wiederholt werden, in dem an verschiedenen Stellen im Strom jeweils Bits mit der oben beschriebenen Methode eingebettet werden. So ist es möglich, eine signifikante Anzahl an Bits in die aufgefangenen Ströme einzubetten, so daß eine eindeutige Zuordnung möglich wird. Die Auswahl der Abstände, in die die Wasserzeichen eingebettet werden sollen, hängen von der Implementierung ab und kann beliebig erfolgen, solange der Dekodierer die Lage der eingebetteten Wasserzeichenbits kennt.

In der Arbeit wurden Tests durchgeführt, bei denen für die Einbettung eines Bits der Durchschnitt jeweils über 12 Paketabstände gebildet wurde. Bei den Tests wurde eine Schrittweite von 400ms verwendet. Dabei wurden bis zu einer maximalen zufälligen Veränderung des Timings von 800ms alle Wasserzeichen wieder korrekt erkannt. Selbst bei einer zufällig verteilten Veränderung von 1 Sekunde werden noch zwischen 84% und 97% wiedererkannt.

Im Rahmen ihrer Arbeiten haben Wang und Reeves den Algorithmus implementiert und mit einigen Netzwerktraces verifiziert. Hierzu haben sie einen realen Trace mit 121 SSH-Verbindungen und einen virtuellen Trace mit 1000 Telnet-Verbindungen, den sie anhand einer empirischen Verteilung erzeugt haben, verifiziert und getestet.

Für einen Angreifer besteht bei der Analyse durch diesen Algorithmus eine größere Schwierigkeit die Erkennung zu umgehen, denn er muß hierfür den Ablauf seiner interaktiven Session soweit modifizieren, daß eine der beiden Verbindungen eine große Verzögerung der Pakete bekommt. Diese Verzögerung muß aber jenseits von 1 Sekunde liegen, damit die Möglichkeit der Erkennung, wie sie in der Arbeit vorgestellt worden ist, auf 50-60% abfällt. Bessere Abwehrmaßnahmen kann der Angreifer allerdings durch genaue Kenntnis der Implementierung erreichen, wenn er weiß, zwischen welchen Paketen die Wasserzeichen eingebettet werden und mit welcher Schrittweite die Quantifizierung stattfindet.

3.3.3 Abstandsalgorithmus

Der von Kunikazu Yoda und Hiroaki Etoh vorgestellte Abstandsalgorithmus aus ihrer Arbeit „Finding a Connection Chain for Tracing Intruders“ [9] geht einen etwas anderen Weg als die anderen hier beschriebenen Algorithmen. Er versucht aus den zeitlichen Ablaufgraphen zweier Verbindungen den minimalen Abstand der Graphen zu finden und daraus eine Aussage über die Korrelation zu treffen.

Wie alle anderen Algorithmen wird auch bei diesem Algorithmus die Analyse auf dem Timing der Pakete durchgeführt. Aus der Systematik des Algorithmus ergibt sich keine Einschränkung in Bezug auf Realtime- oder Offlineanalyse. Allerdings ist eine effiziente Implementierung nur als Offlineanalyse möglich, da andernfalls zu viele Daten vorgehalten werden müssen. Für die Implementierung des Algorithmus wird kein aktiver Eingriff in den Netzwerkverkehr benötigt. Der Algorithmus ist an sich für die Erkennung an einem Meßpunkt ausgelegt, und kann daher nur mit den selben Einschränkungen wie bereits beim ON/OFF-Algorithmus erwähnt dazu benutzt werden,

auf Daten mehrerer Meßpunkte zu operieren. Hinsichtlich des Speicherverbrauches wird bei diesem Algorithmus für das Vorhalten der benötigten Daten eine größere Menge verbraucht, weil für jeden Strom die kompletten Paketankunftszeiten benötigt werden. Um für einen gegebenen Strom alle möglichen Stepping Stones in einem Trace zu finden, wird „CPU-Zeit in der Größenordnung von O(nN) benötigt, wobei n die Anzahl der Pakete im Vergleichsstrom und N die Gesamtanzahl der Pakete im Trace ist“ [7]. Daher wird wesentlich mehr CPU-Zeit benötigt, wenn für alle Ströme eines Traces mögliche Stepping Stones gefunden werden sollen.

Der Algorithmus basiert auf der Analyse der Graphen, die aus dem zeitlichen Ablauf der Paketankünfte basieren. Diese Graphen werden aus den TCP Sequenznummern über der Zeit gebildet und zeigen damit graphisch einen Verlauf der Aktivität in diesem Strom. Bei der Bildung dieser Graphen werden Pakete die, z.B. auf Grund von Fehlern, erneut übermittelt werden müssen ignoriert. D.h. es werden lediglich Pakete herangezogen, deren Sequenznummer größer ist als die höchste empfangene Sequenznummer in diesem Strom.

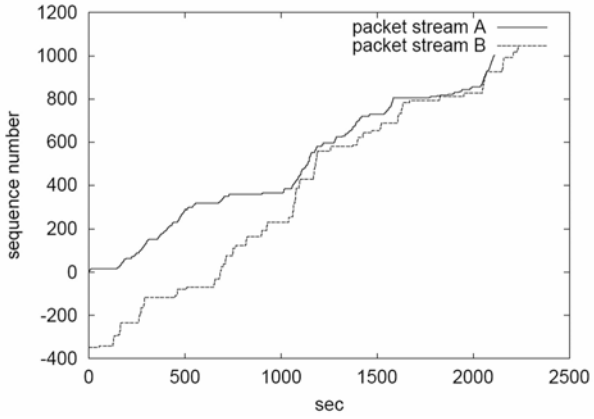

Abbildung 3: Darstellung zweier Netzwerkströme [10]

Der Entscheidung ob ein Stepping Stone vorliegt wird nun aus dem minimalen Abstand zwischen den Graphen zweier Ströme gezogen. Dieser minimale Abstand wird durch Verschieben des Vergleichsgraphen sowohl entlang der Sequenznummernachse als auch entlang der Zeitachse berechnet. Dabei darf der Vergleichsgraph den Ausgangsgraphen, wie auch in Abbildung 3 zu sehen, nicht schneiden.

Die Experimente in der Arbeit von Etoh/Yoda haben gezeigt, daß unabhängige Ströme in aller Regel einen minimalen Abstand von mindesten 3 Sekunden ausweisen. Für Ströme, die zur selben Verbindungskette gehören, finden sich hier minimale Abstände, die zwischen 1 und 3 Sekunden liegen. Eben diese Verbindungen werden von dem Algorithmus als Stepping Stone markiert.

Möchte man der Erkennung durch diesen Algorithmus entgehen, reicht es aus, eine seiner Verbindungen mit einer zufälligen Folge von Chaff-Paketen zu überlagern, um diese Abstandsbestimmung soweit zu verfälschen, daß die Verbindungen nicht mehr als Stepping Stone markiert werden.

3.3.4 Interpacket-Delay Algorithmus

Vorgestellt wurde der IPD-Algorithmus von Xinyuan Wang, Douglas S. Reeves und S. Felix Wu in ihrer Arbeit mit dem Thema „Inter-Packet Delay Based Correlation for

Tracing Encrypted Connections through Stepping Stones“ [11]. Es wird in dem Algorithmus versucht, Stepping Stones über eine Ähnlichkeit in der zeitlichen Länge der Abstände zwischen zwei Paketen nachzuweisen.

Der IPD-Algorithmus ist ebenfalls ein Algorithmus, der ausschließlich auf dem Timing der Pakete arbeitet, und setzt auch keinen aktiven Eingriff in den Netzwerkverkehr voraus. Der Algorithmus ist gleichermaßen zur Realtimeanalyse als auch zur Offlineanalyse einsetzbar und ist durch seine Charakteristik auch mit Datenpaketen, die an zwei oder mehr Meßpunkten aufgefangen wurden problemlos einsetzbar.

Hinsichtlich des Speicherverbrauches stellt er höhere Ansprüche an den Erkennungscomputer, da für jeden Strom der komplette Paketverlauf für die Analyse vorgehalten werden muß. Auch von der CPU-Zeit ist der Algorithmus sehr verschwenderisch, da er bei einem Analyselauf für jedes Paar Ströme eine große Zahl Vergleiche benötigt.

Die Analyse basiert auf dem Versuch, eine Korrelation zwischen zwei Strömen zu finden, in dem versucht wird Ähnlichkeiten im Ablauf der Pakete zu finden. Dabei wird davon ausgegangen, daß zwei Ströme die einen Stepping Stone bilden starke Ähnlichkeiten in der zeitlichen Abfolge ihrer Pakete haben. Diese Ähnlichkeiten lassen sich in den relativen Zeitabständen der Pakete gut sichtbar machen. Man kann beobachten, daß in solchen Strömen die Zeitabstände zwischen den empfangenen Paketen übereinstimmen. Das Finden dieser Übereinstimmung wird lediglich durch eine gewisse Verschiebung über die Pakete erschwert. Idealerweise ergibt sich bei der Analyse ein konstanter Offset zwischen den zwei Strömen, der durch die Verarbeitung auf dem Stepping Stone entsteht.

Um diese Korrelationspunkte zu finden, iteriert der Algorithmus über die Vektoren, die sich aus den Zeitabständen zwischen den Paketen bilden lassen und findet mit Hilfe einer Funktion Korrelationspunkte. Diese Funktion (CPF = Correlation-Point-Function) liefert einen Wert zwischen 0 und 1, der angibt wie „gut“ die Übereinstimmung ab diesem Punkt ist. Die Übereinstimmung ab einem Punkt wird nur bezüglich eines bestimmten Paket-Fensters berechnet. Liegt der Wert der CPF für zwei Startpunkte in den Vektoren über einem Schwellwert δ_{cp}, dann werden diese Punkte als Korrelationspunkt betrachtet und in den Ergebnisvektoren gespeichert. Eine Anforderung an eine solche Funktion ist natürlich, daß die Funktion für keine zwei Ströme, die nicht zu einem Stepping Stone gehören, den Wert 1 liefert. Weiterhin soll sowohl die Rate der False-Positives als auch die Rate der False-Negatives minimal sein.

Es gibt für die CPF mehrere geeignete Funktionen, die zu einer unterschiedlichen Güte der gewonnenen Korrelationspunkte führt. Die Arbeit von Wang, Reeves und Wu stellt vier verschiedene Funktionen vor, und vergleicht die Funktionen, in dem für alle Funktionen die gleichen Parameter und Ausgangsdaten verwendet werden. Dabei schneidet die Min/Max Summenfunktion deutlich am besten ab, und ihr wird von den Autoren im weiteren auch eine Erkennungsrate von 100% True-Positives und 0% False-Positives bescheinigt.

Die Funktionen sind jetzt im folgenden aufgeführt, wobei X und Y für die beiden Ströme stehen, j die aktuelle Vergleichsposition in X bezeichnet, k für den aktuell zu vergleichenden Offset steht und s die Größe des Vergleichsfensters steht.

MIN/MAX SUM RATIO (MMS)

$$CPF(X,Y,j,k,s)_{MMS} = \frac{\sum_{i=j}^{j+s+1} \min(x_i, y_{i+k})}{\sum_{i=j}^{j+s+1} \max(x_i, y_{i+k})}$$

STATISTICAL CORRELATION (STAT)

$$CPF(X,Y,j,k,s)_{STAT} = \begin{cases} \rho(X,Y,j,k,s) & , \rho(X,Y,j,k,s) \geq 0 \\ 0 & , \rho(X,Y,j,k,s) < 0 \end{cases}$$

$$\rho(X,Y,j,k,s) = \frac{\sum_{i=j}^{j+s+1} (x_i - E(X)) \times (y_{i+k} - E(Y))}{\sqrt{\left[\sum_{i=j}^{j+s+1} (x_i - E(X))^2\right] \times \left[\sum_{i=j}^{j+s+1} (y_{i+k} - E(Y))^2\right]}}$$

NORMALIZED DOT PRODUCT 1(NPD1)

$$CPF(X,Y,j,k,s)_{NDP1} = \frac{\sum_{i=j}^{j+s+1} x_i \times y_{i+k}}{\max(\sum_{i=j}^{j+s+1} x_i^2, \sum y_{i+k}^2)}$$

NORMALIZED DOT PRODUCT 2(NPD2)

$$CPF(X,Y,j,k,s)_{NDP2} = \frac{\sum_{i=j}^{j+s+1} x_i \times y_{i+k}}{\sum_{i=j}^{j+s+1} \left[\max(x_i, y_{i+k})\right]^2}$$

Diese so gewonnenen Korrelationspunkte werden dann mit einer weiteren Funktion, der Correlation-Value-Function (CVF), miteinander verglichen. Die CVF berechnet dabei wie konstant sich der Offset zwischen den Korrelationspunkten verhält. Diese Funktion ergibt sich aus statistischen Überlegungen und gibt wieder einen Wert zwischen 0 und 1 an, der über einem weiteren Schwellwert δ liegen muß, damit dieses Verbindungspaar als Stepping Stone markiert wird.

$$CVF(C_x, C_y) = \begin{cases} 0 & , n = 0 \\ \rho(C_x, C_y) & , n > 1 \\ 1 & , n = 1 \end{cases}$$

$$\rho(C_x, C_y) = \frac{\sum_{i=1}^{n} (j_i - E(C_x)) \times (j_i + k_i - E(C_y))}{\sqrt{\left[\sum_{i=1}^{n} (j_i - E(C_x))^2\right] \times \left[\sum_{i=1}^{n} (j_i + k_i - E(C_y))^2\right]}}$$

Der Algorithmus ist, genauso wie die vorhergehenden, anfällig gegenüber zufällig eingestreuten Chaff-Paketen sowie gegenüber einer, ebenfalls zufälligen, Veränderung des Timings im Paketablauf. Bei gleichförmiger Veränderung durch einen konstanten Strom an Chaff-Paketen ist zumindest noch eine gewisse Möglichkeit der Erkennung gegeben.

Im Rahmen ihrer Arbeit wurde der Algorithmus von Wang, Reeves und Wu implementiert und verifiziert. Sie setzten hierzu 5 verschiedene Traces ein, die sowohl Telnet- als auch SSH-Sessions enthielten. In ihren Tests waren zwischen 15 Sessions

(5111 Pakete) und 400 Sessions (364158 Pakete) enthalten, die aus größeren Traces herausgefiltert wurden.

3.3.5 Wavelet Algorithmus

Der Wavelet-Algorithmus wurde von eine großen Gruppe vorgestellt, die schon Erfahrungen im Bereich der Stepping Stone Erkennung gesammelt haben. Die Autoren sind David L. Donoho, Ana Georgina Flesia, Umesh Shankar, Vern Paxson, Jason Coit und Stuart Staniford. Der in ihrer Arbeit „Multiscale Stepping-Stone Detection: Detecting Pairs of Jittered Interactive Streams by Exploiting Maximum Tolerable Delay“ [12] vorgestellte Algorithmus basiert auf einer Analyse der Ankunftszeiten der Paketströme. Dabei wird geprüft, in wie weit zwei korrelierende Ströme ähnliche Zeitcharakteristika aufweisen, wenn man ihre Ankunftszeiten analysiert.

Um den im folgenden beschriebenen Algorithmus verstehen zu können soll an dieser Stelle ein kleiner Exkurs erfolgen, der die allgemeinen Konzepte der Wavelet-Transformation erläutert. Anschließend wird der Algorithmus näher beschrieben.

WAVELET-TRANSFORMATION

Dieser Abschnitt stellt lediglich eine Art Crashkurs dar, der die nötigen Kenntnisse vermittelt, um einen Eindruck zu bekommen wie der Algorithmus funktioniert. Eine tiefere Auseinandersetzung mit der Thematik der Waveletanalyse würde hier zu weit führen.

Die Wavelet-Transformation wird in vielen Bereichen eingesetzt. Beispielsweise findet sie ihren Einsatz im Bereich der digitalen Signalverarbeitung in der Elektrotechnik oder im Bereich der Bildkompression um nur zwei, auf den ersten Blick nicht miteinander verwandte, Bereiche zu nennen. Da die Berechnung einer solchen Transformation mit einem nicht unerheblichen Aufwand verbunden ist, stellt sich die Frage welche Vorteile diese Umwandlung mit sich bringt. Ein Vorteil der Daten nach einer Wavelet-Transformation ist „eine verbesserte Robustheit des Signals gegenüber Störungen auf bestimmten Übertragungskanälen“ [13]. Ein weiterer positiver Aspekt, der in der Bildkompression eine Rolle spielt, ist eine Fokussierung der Daten auf wichtigere Bestandteile und damit verbunden der Möglichkeit unwichtigere Teile wegzulassen. Darüber hinaus kann im Bereich der Bildkompression eine bessere Komprimierbarkeit mit herkömmlichen, verlustfreien Kompressionsverfahren erreicht werden, wenn die transformierten Daten komprimiert werden, als wenn die Ausgangsdaten verwendet werden.

Die Wavelet-Transformation versucht ein gegebenes Ausgangssignal als Komposition einer einfachen vorgegebenen Funktion, dem Wavelet, darzustellen. Um dieses möglich zu machen, muß das Wavelet bei der Komposition verschoben und skaliert werden. Die dafür benötigten Koeffizienten sind dann das Ergebnis der Analyse.

Um die Bedeutung der Koeffizienten zu verdeutlichen, soll hier ein kurzes Beispiel [14] zeigen welchen Einfluß die einzelnen Teile bei der Rekonstruktion des Signals haben. Ausgehend von einer stückweise konstanten Funktion die durch die Zahlenfolge (9, 7, 3, 5) repräsentiert wird, kann man daraus mit dem Haar-Wavelet eine Transformation berechnen deren Koeffizienten (12, 4, $\sqrt{2}$, $-\sqrt{2}$) betragen. „Das Haar-Wavelet ist das erste in der Literatur bekannt gewordene Wavelet und wurde 1909 von Alfred Haar vorgeschlagen. Es ist außerdem das einfachste bekannte Wavelet.“ [15] Abbildung 6 zeigt das Haar-Wavelet.

Ziel des Beispieles ist es, zu zeigen wie aus den Werten der Transformation wieder das oben abgebildete Originalsignal (Abbildung 4) rekonstruiert werden kann. Dabei

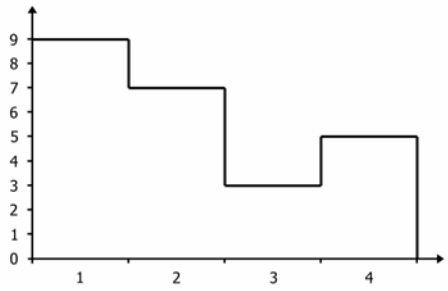

Abbildung 4: Darstellung des Orginalsignals

wird auch deutlich, wie die Koeffizienten der Wavelet-Transfomartion eine schichtweise immer genauere Form der Rekonstruktion ermöglicht.

Aus den vorgegebenen Funktionen, der abgebildeten Skalierungs- und Waveletfunktion (Abbildung 6), läßt sich mit den ersten beiden Koeffizienten zumindest eine grobe Form des Signals rekonstruieren. Dabei wird die Skalierungsfunktion mit der ersten Waveletkoeffizienten multipliziert. Daraufhin wird die Waveletfunktion mit dem zweiten Koeffizienten multipliziert und zu der vorherigen Funktion addiert.

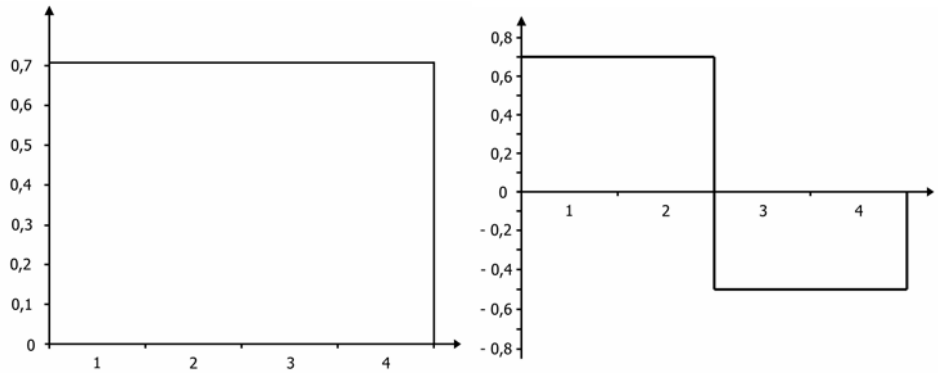

Abbildung 6: Skalierungsfunktion und Haar-Wavelet

Die in Abbildung 5 blau eingefärbten Bereiche stellen die nach oben verschobene und skalierte Version des Wavelets dar. Die Bereiche wurden zu der Grundfunktion addiert und ergeben dementsprechend einen höheren Wert im linken Bereich und einen niedrigeren Wert im Rechten. Daraus ergibt sich die fett markierte Grundform des Signals.

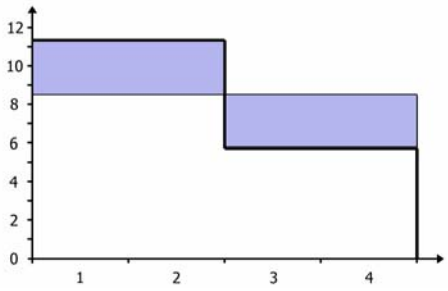

Abbildung 5: Erster Schritt der Rekonstruktion

Für den zweiten Schritt wird diese Grundform nur wiederum mit der Skalierungsfunktion multipliziert, um den Signalpegel zu erhalten, und anschließend mit den beiden genaueren Wavelets, die mit den zweiten beiden Koeffizienten der Wavelet-Transformation skaliert wurden überlagert.

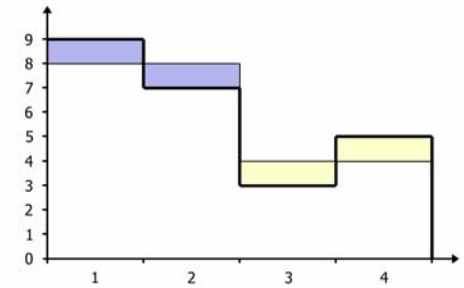

Abbildung 7: Zweiter Schritt der Rekonstruktion

Wie in Abbildung 7 schön zu erkennen ist, erzeugt das Wavelet im unteren Bereich aus der Grobform die gewünschten Werte. Im oberen Bereich des Signals wird das Wavelet durch das negative Vorzeichen des Koeffizienten umgekehrt und sorgt so auch hier für die korrekten Werte.

Wie das Beispiel hier zeigt ist mit dem Haar-Wavelet, eine perfekte Rekonstruktion des Ausgangssignals möglich. Es gibt eine ganze Reihe von Wavelets die hier zum Einsatz kommen können, um die Transformation durchzuführen, die je nach Einsatzgebiet und Zielsetzung der Transformation ihre Stärken und Schwächen haben.

Die hier eingesetzte Multi-Skalen-Analyse der Diskreten Wavelet Transformation (DWT) iteriert bei der Implementierung als Filterbank (man spricht hier auch von der Fast Wavelet Transformation, FWT) schrittweise über den Ausgangsvektor und spaltet in jedem Schritt immer niederfrequente Daten ab, die durch die Systematik dann mit einer niedrigeren Genauigkeit abgetastet worden sind, als die verbleibenden höherfrequenten Anteile. Diese Iterationen werden immer auf der Hälfte der Daten des vorrausgehenden Schrittes vollzogen. So werden für eine vollständige Analyse von 2^n Werten n Schritte (also $\log_2(2^n)$) benötigt.

Die später beschrieben Wavelet Bibliothek implementiert die DWT eben als die hier beschriebene Filterbanklösung. „Diese Filterbank Implementierung gewährleistet eine schnelle numerische Realisierung der Diskreten Wavelet Transformation. Dadurch lassen sich ähnlich wie bei der FFT schnelle Algorithmen für deren Berechnungen schreiben.“ [16]

Am häufigsten wird die Waveletanalyse zur Kompression von Bilddaten verwendet. Grundsätzlich ist es damit möglich aus vorhandenen Bilddaten (u.a. im JPEG-Format) stärker komprimierte Bilder ohne Qualitätsverlust zu erzeugen. Da die Waveletanalyse im Wesentlichen ein Signal in stark maßgebliche und weniger wichtige Bestandteile zerlegen kann, ist sie hierfür bestens geeignet. Die Waveletanalyse ist eine Form der Vereinfachung eines Signals, wie es auch die Fouriertransformation ist. Ein Signal, in diesem Falle eine Zeile eines Bildes, wird dabei als eine Kombination aus einem anderen, kleineren Signal, dem Wavelet, aufgefaßt. Die Transformation wird dabei sowohl auf die Zeilen, als auch auf Spalten des Bildes angewendet und liefert als Ergebnis ein Bild wie es in der Abbildung 8 auf der nächsten Seite zu sehen ist. Die daraus errechneten Bildkoeffizienten können dann mit normalen Kompressions-methoden weiter komprimiert werden, da sie im überwiegenden Teil gleichförmig und

null sind. Darüber hinaus können auch unrelevante Teile einfach auf Null gesetzt werden, um so eine noch bessere Kompressionsrate zu erreichen.

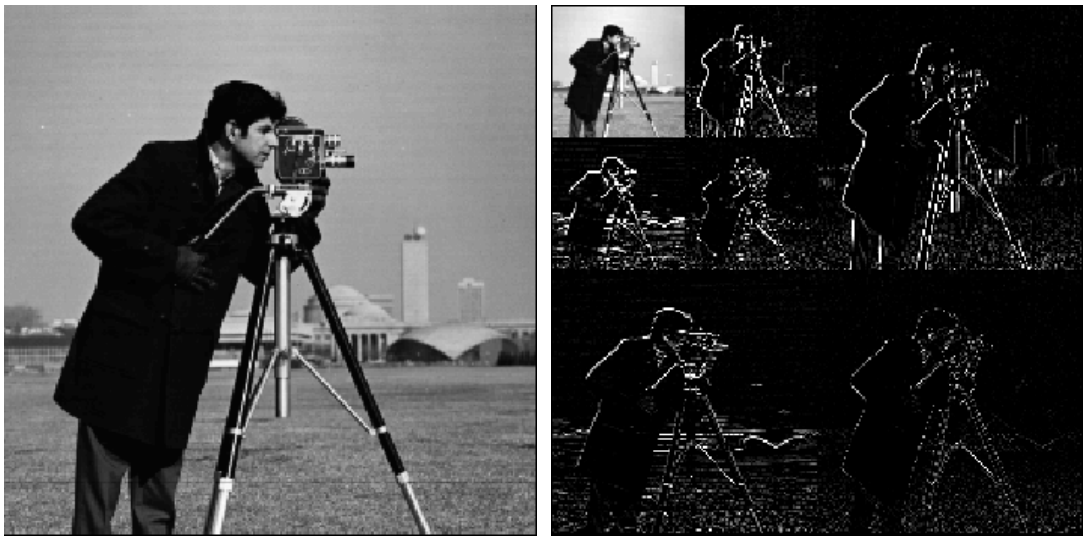

Abbildung 8: Beispiel einer mehrdimensionalen Waveletanalyse eines Bildes [17]

ALGORITHMUS

Wie bereits beschrieben ist die Grundlage dieses Algorithmus eine Timinganalyse der einzelnen Ströme, die sowohl Online als auch Offline geschehen kann. Der Algorithmus agiert rein passiv auf den gewonnen Daten und läßt sich nur mit den üblichen Problemen auf mehreren Meßpunkten gleichzeitig betreiben.

Der Speicherverbrauch dieses Algorithmus ist dadurch, daß für jeden Strom der komplette Verlauf benötigt wird ebenfalls hoch anzusiedeln. Bei der CPU-Auslastung ist die Wavelet-Transformation zwar eine teure Operation, aber da sie im immer nur zur bei der Beendigung einer Verbindung berechnet werden muß ist unter normalen Bedingungen kein größerer Engpaß zu erwarten.

Um eine Analyse durchführen zu können wird der für jeden Paketstrom einzeln erfaßte Vektor der Ankunftszeiten zunächst in eine festgelegte Anzahl an gleichlange Zeitintervalle unterteilt und die jeweilige Anzahl der Pakete je Zeiteinheit gezählt. Auf den daraus resultierenden Vektoren wird anschließend eine vollständige Waveletanalyse durchgeführt. Die Waveletanalyse liefert zu einem gegebenen Vektor eine Reihe von Koeffizienten, die darstellen, wie aus dem vorgegebenen Wavelet das „Ausgangssignal“ rekonstruiert werden kann. Die Koeffizienten geben dabei eine Wertigkeit vor, wie stark ein gewisser Anteil des Wavelets im Ausgangssignal repräsentiert ist. Der eigentliche Vergleich findet nun auf diesen Koeffizienten statt. Die Koeffizienten werden mittels einer statistischen Formel auf ihre Ähnlichkeit überprüft und der daraus resultierende Wert dient zur Wertung, ob ein Stepping Stone vorliegt oder nicht.

Der Algorithmus ist auch in der Lage Stepping Stones zu erkennen, auch wenn der Angreifer versucht durch eine Änderung im Timing der Pakete, oder durch einfügen von Chaff-Paketen seine Erkennung zu verschleiern. Dies wird eben durch die Fähigkeit der „Konzentration“ auf wesentliche Bestandteile der Waveletanalyse möglich. Durch diese Fähigkeit geraten die unwesentlicheren Bestandteile des Signals, wie ein konstanter Fluß von Chaff-Paketen oder eine kleine Veränderung des Pakettimings, im

Vergleich zu den wichtigen Bestandteilen, wie die wiedererkennbaren Muster des Tippens oder „Denkpausen“, in den Hintergrund und ermöglichen so trotzdem eine Erkennung.

In Bezug auf die Möglichkeiten einer Erkennung zu entkommen stellt der Wavelet-Algorithmus eine große Robustheit zur Verfügung. Bis zu einem gewissen Maß an, selbst zufälliger, Verzögerung und Chaff-Paketen, überwiegt der Anteil der der allgemeinen interaktiven Charakteristik gegenüber den herausfilterbaren Ablenkungsmanövern.

Der Algorithmus wurde in der genannten Arbeit nur theoretisch vorgestellt. Daher gab es bisher keine Beispielimplementierung, sondern er wurde erstmalig im Rahmen dieser Diplomarbeit implementiert.

3.3.6 Paket-Zähler Algorithmus

Eine sehr neue Arbeit „Detection of Interactive Stepping Stones with Maximum Delay Bound: Algorithms and Confidence Bound“ [19] stellt den Paket-Zähler Algorithmus vor. Diese Arbeit stammt von dem Autoren Avrim Blum, Dawn Song und Shoba Venkataraman. Dieser Algorithmus versucht mit wesentlich schwächeren Annahmen und Einschränkungen auszukommen, als die vorher veröffentlichten Algorithmen. Außerdem wird in der Arbeit versucht, beweisbare Grenzen für die Fehlerrate und die minimal zur Erkennung benötigte Paketzahl zu finden. Grundlage des Algorithmus ist die Annahme, daß bei zwei Verbindungen, die zu einem Stepping Stone gehören, innerhalb einer bestimmten Anzahl von Paketen nur ein geringer Unterschied in der Anzahl der Pakete auftritt.

Im weitesten gesehen basiert auch dieser Algorithmus auf einer Timinganalyse der beteiligten Ströme, wobei er eigentlich im klassischen Sinne weder eine Timinganalyse noch eine Inhaltsanalyse durchführt. Er ist problemlos als Realtime- oder als Offlinealgorithmus einsetzbar. Die Analyse wird auf den Paketheadern rein passiv durchgeführt. Einem Einsatz an mehreren Meßpunkten steht nichts im Wege, da er lediglich auf die richtige Reihenfolge der Pakete, nicht aber auf deren exakten Ankunftszeitpunkt angewiesen ist.

Die Rechenzeit, die der Algorithmus für seine Analysen benötigt ist eher gering. Zwar muß eine gewisse Auswertung bei jedem Paket durchgeführt werden, aber diese besteht nur aus wenigen Vergleichen. Der Speicherverbrauch ist an sich auch gering, da nicht viele Daten vorgehalten werden müssen, doch wird die Auswertung zeigen, daß der Speicherverbrauch im breitbandigen Umfeld zum Ausschlußkriterium werden kann.

Als Vorraussetzungen für den Algorithmus wird angenommen, daß es für einen Angreifer ein maximale Verzögerung Δ gibt, die im interaktiven Verkehr erträglich ist. Dies läßt den Schluß zu, daß alle Pakete aus Strom A in Strom B spätestens nach der Zeit Δ ebenfalls erscheinen müssen. Die Verzögerung, die durch die Leitung o.ä. verursacht wird wird hierbei nicht in die Berechnung einbezogen, da sie vergleichsweise sehr gering gegenüber der Verzögerung durch das interaktive Verhalten des Angreifers ist. Aus der Leitungscharakteristik ergibt sich die maximale Zahl an Paketen $p_{\Delta,}$ die der Angreifer in dieser Zeit senden kann.

Dieser Algorithmus kommt mir sehr wenig Aufwand aus. Es muß lediglich für jedes Strom-Paar ein gemeinsamer Paketzähler geführt werden. Sind in beiden Strömen zusammen mehr als n Pakete aufgezeichnet worden, so wird eine Überprüfung durchgeführt, ob sich die Paketzähler der beiden Einzelströme um mehr als p_{Δ}

unterscheiden. Ist dies der Fall, so kann davon ausgegangen werden, daß die beiden Ströme nicht zu einem Stepping Stone gehören. Andernfalls wird der Zähler für eine mögliche Korrelation erhöht und die Beobachtung wieder von vorne begonnen. Überschreitet der Zähler einen Schwellwert m, dann kann mit Sicherheit δ davon ausgegangen werden, daß die beiden Ströme zu einem Stepping Stone gehören. Wenn der Wert p_Δ bekannt ist, dann müssen maximal M Pakete beobachtet werden, um die False-Positive Rate δ sicherstellen zu können. Selbst wenn p_Δ nicht bekannt ist, kann eine Abschätzung der maximalen Paketzahl getroffen werden.

Der Algorithmus kann auch durch kleine Änderungen so modifiziert werden, daß der Stepping Stone trotz Abwehrmaßnahmen des Angreifers erkannt werden kann. Hier gilt jedoch auch, daß die Erkennung des Stepping Stones mit Abwehrmaßnahmen nur bis zu einer Obergrenze an hinzugefügten Chaff-Paketen möglich ist.

Eine Implementierung und Evaluierung des Algorithmus wurde im Rahmen ihrer Arbeit nicht durchgeführt. Die Richtigkeit des Algorithmus wurde auf theoretischer Grundlage bewiesen. Die Implementierung wurde für diese Diplomarbeit erstellt.

3.3.7 Hop-Count Algorithmus

Yung stellt in seinem Paper „Detecting Long Connection Chains of Interactive Terminal Sessions“ [20] eine weitere spezialisierte Möglichkeit, den Hop-Count Algorithmus, vor, um Stepping Stones anhand ihrer zeitlichen Charakteristik zu erkennen. Die Idee des Algorithmus ist es, die Anzahl der weiteren Computer in einer Stepping-Stone-Kette zu bestimmen und daraus zu schließen, ob diese Kette legitim oder als Angriff zu sehen ist.

Die Analyse, die hier durchgeführt wird, basiert auf dem zeitlichen Verhalten interaktiver Sessions. Er ist online, während eine Session läuft, genauso gut einsetzbar, wie offline auf einem Trace. Um einen Stepping Stone zu erkennen, ist für den Hop-Count Algorithmus kein aktiver Eingriff in den Netzverkehr nötig. An mehreren Meßpunkten ist er allerdings nur unter den bekannten Problemen einsetzbar.

Für eine Implementierung des Algorithmus ist weder ein hoher Speicherverbrauch, noch eine hohe CPU-Auslastung zu erwarten, da für die einzelnen Verbindungen nur wenige Werte vorgehalten und später miteinander verglichen werden. Außerdem findet der Vergleich ausschließlich innerhalb einer Verbindung statt, und nicht wie bei den anderen Algorithmen zwischen Verbindungspaaren.

Der Algorithmus geht von Implementierungsdetails von interaktiven Sessions aus. Diese Details werden sowohl von verschlüsselten, als auch unverschlüsselten interaktiven Sessiondiensten durchgeführt. Es geht hier um das Feature von „delayed acknowledgements“. Dabei ist allerdings nicht das „delayed aknowledgement“ des TCP Protokolls gemeint, sondern eine vergleichbare Eigenschaft von interaktiven Serverdiensten wie SSH. Bei einer interaktiven Session wird normalerweise jedes vom Client übertragene Zeichen vom Server bestätigt, genauso wie es im umgekehrten Fall ist. Diese Bestätigung wird im interaktiven Verkehr einfach mit dem nächsten Zeichen oder der nächsten Antwort mitgeschickt. Steht hierfür keine Übermittlung zur Verfügung, bei der die Bestätigung angehängt werden kann, so wird nach Ablauf einer gewissen Zeitspanne ein delayed acknowledgement ausgeführt. Dies ist ein leeres Paket, das lediglich die Bestätigung übermittelt. Bei verschlüsseltem Verkehr ist die Erkennung der delayed acknowledgement Pakete nicht ganz so trivial wie bei unverschlüsseltem Verkehr, aber trotzdem problemlos aus dem Kontext der Verbindung möglich.

Bei langen Verbindungsketten kommt es nun zu dem gut meßbaren Phänomen, daß von dem direkt nächsten Hop in der Verbindungskette immer ein delayed acknowledgement zurückgeschickt wird, weil das Paket, welches das Acknowledgement verursacht hat noch auf dem Weg zum eigentlichen Zielhost ist und somit das Antwortpaket noch nicht bei dem nächsten Hop in der Verbindungskette wieder angekommen ist.

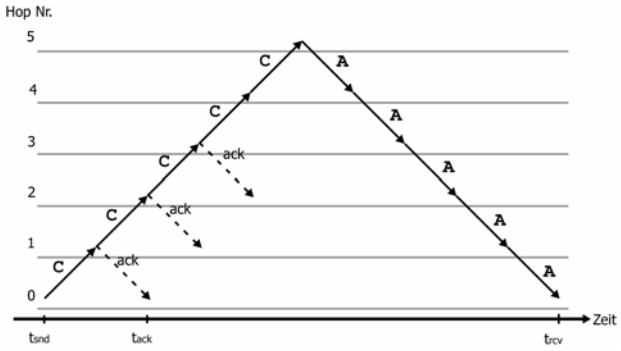

Abbildung 9: Delayed Aknowledgement [21]

Wie in der Abbildung 9 zu erkennen wird ein Zeichen C welches zum Zeitpunkt t_{snd} von Rechner 0 (der Rechner für den die Erkennung durchgeführt wird) abgeschickt wird vom nächsten Rechner in der Kette nach dem abgelaufenen Timeout mit einem delayed acknowledgement bestätigt. Das delayed acknowledgement wird vom überwachten Rechner dann zum Zeitpunkt t_{ack} empfangen. Nachdem das Zeichen nun die Kette weitergewandert ist, wandert auf dem umgekehrten Weg die endgültige Bestätigung, meist in Form einer Anforderung das Zeichen auf dem Bildschirm darzustellen, zurück. Die Ankunft dieser Bestätigung wird dann zum Zeitpunkt t_{rcv} verzeichnet.

Aus dieser Zeitdifferenz zwischen dem Absenden des Zeichens, dem delayed acknowledgement und der eigentlichen Bestätigung des Zeichens läßt sich gut eine Abschätzung treffen, wie viele Computer noch auf dem Weg zum eigentlichen Ziel in dieser Kette hängen. Sind das mehr als 2, so kann man meist davon ausgehen, daß es sich hierbei um eine nicht erlaubte Nutzung eines Computers als Stepping Stone handelt

Mit den bekannten Möglichkeiten ist dieser Algorithmus nicht zu umgehen, da er auf völlig anderen Daten arbeitet. Allerdings ist es hier durch den Einsatz individueller Serverprozesse für die interaktiven Sessions möglich die delayed aknowledgments zu unterbinden und somit eine Erkennung unmöglich zu machen.

3.4 Vorauswahl

Die hier vorgestellten Algorithmen bieten die Basis, aus der eine Auswahl getroffen werden mußte, welche Algorithmen wirklich zum Einsatz kommen sollen.

Bei der Auswahl wurde bedingt durch die Aufgabenstellung wert darauf gelegt, daß kein aktiver Eingriff in den Netzwerkverkehr stattfindet. Diese Entscheidung liegt in zwei Tatsachen begründet, die es im breitbandigen Umfeld verbieten aktiv einzugreifen. Die erste Tatsache ist schlicht technischer Natur. Im breitbandigen Umfeld wird in der Regel der Verkehr über eine Glasfaserleitung übermittelt, die dann für den Sniffer durch einen Replikator angezapft wird. Der Sniffer zeichnet also eine Kopie des Verkehrs auf, und ist somit nicht in der Lage Veränderungen am vorbeifließenden Verkehr vorzunehmen. Die zweite Tatsache liegt in der Art und Weise des Watermarking-

Algorithmus begründet. Der Algorithmus verändert das Timing aller vorbeifließenden interaktiven Sessions, in dem er die ausgewählten Pakete um bis zu 200ms verzögert. Diese Verzögerung ist aber in einer interaktiven Session auf einer breitbandigen Leitung durchaus spürbar. Diese spürbare Verzögerung führt einerseits zu einer Beeinträchtigung aller Benutzer, deren legale interaktive Sessions dadurch verlangsamt werden und andererseits zur Möglichkeit der Entdeckung durch den Angreifer.

Die ausgewählten Algorithmen sollten darüber hinaus auf jeden Fall die Möglichkeiten bieten offline eingesetzt zu werden. Ein Algorithmus, der beide Möglichkeiten unterstützt, ist klar vorzuziehen, doch scheint es durch den im breitbandigen Umfeld zum Teil sehr hohen Speicherverbrauch und die Geschwindigkeit erforderlich, daß eine Offlineanalyse möglich ist.

Inhaltsbasierte Algorithmen wurden auf Grund ihrer Trivialität nicht näher beschrieben und finden auch keine weitere Betrachtung, da davon ausgegangen wird, daß bei illegaler Nutzung von Stepping Stones überwiegend verschlüsselter Verkehr verwendet wird. Daher können lediglich Algorithmen zum Einsatz kommen, die auf dem Timing der Verbindung arbeiten. Ein weiterer Vorteil von timingbasierten Algorithmen im Breitbandeinsatz ist, daß bei der Aufzeichnung lediglich die Paketheader auf aufgezeichnet werden müssen und nicht die kompletten Pakete, was zu einer immensen Verringerung des aufgezeichneten Volumens führt.

Die folgende Tabelle 1 faßt die, Kriterien die auf die Algorithmen anwendbar sind, noch einmal zusammen und ermöglichst so einen Überblick über die Algorithmen. Referenzimplementierung beschreibt hierbei, ob es eine Implementierung seitens der Autoren gab, mit der Analysen durchgeführt worden sind. Da eine Integration in das vorhandene IDS Bro, das in Kapitel 4.2 näher beschrieben wird, erreicht werden sollte, wurde für die Algorithmen ebenfalls angegeben, ob die vorhandene Referenz-implementierung bereits in das IDS integriert war.

	Aktiv/ Passiv	**Online/ Offline**	**Inhalt/ Timing**	**Referenz-implement.**
On/Off Algorithmus	*Passiv*	*On-/Offline*	*Timing*	*Ja (Bro)*
Watermarking Algorithmus	*Aktiv*	*Online*	*Timing*	*Ja*
Abstandsalgorithmus	*Passiv*	*On-/Offline*	*Timing*	*Ja*
Interpacket-Delay Algorithmus	*Passiv*	*On-/Offline*	*Timing*	*Ja*
Wavelet Algorithmus	*Passiv*	*On-/Offline*	*Timing*	*Nein*
Paket-Zähler Algorithmus	*Passiv*	*On-/Offline*	*(Timing)*	*Nein*
Hop-Count Algorithmus	*Passiv*	*On-/Offline*	*Timing*	*Ja*

Tabelle 1: Überblick der Algorithmen

Die Kriterien Geschwindigkeit, Speicherverbrauch und Erkennungsrate wurden bei der Vorauswahl nicht weiter berücksichtigt, sondern werden erst in der Analyse der ausgewählten Algorithmen zum Vergleich herangezogen.

Anhand der vorliegenden Kriterien wurden zur näheren Betrachtung die folgenden Algorithmen ausgewählt:

- On/Off Algorithmus
- Interpacket-Delay Algorithmus
- Wavelet Algorithmus
- Paket-Zähler Algorithmus

Diese Algorithmen machen den besten Eindruck, um im breitbandigen Umfeld sinnvoll einsetzbar zu sein. Der On/Off Algorithmus dient wie oben schon erwähnt als Referenzimplementierung, da er der am besten getestete Algorithmus ist und zugleich auch im eingesetzten Softwareumfeld (Bro) bereits implementiert ist. Die anderen Algorithmen wurden im Rahmen dieser Arbeit ebenfalls für Bro implementiert. Der IPD Algorithmus wirkt vielversprechend, da er bereits zur Laufzeit in der Lage ist Stepping Stones zu erkennen, noch während die Session aktiv ist. Die Entscheidung für den Wavelet Algorithmus ist klar, weil dieser Algorithmus die Möglichkeit bietet, Stepping Stones zu erkennen auch wenn der Angreifer aktiv in den zeitlichen Ablauf eingreift um sein Tun zu verschleiern. Schlußendlich rundet der Paket-Zähler Algorithmus die Auswahl ab, da er eine effiziente und schnelle Möglichkeit bietet eine Analyse durchzuführen.

4 Vergleich der Algorithmen

Im nun folgenden Kapitel werden die Rahmenbedingungen erklärt, unter denen sich die Algorithmen dem Vergleich stellen müssen. Dabei kommen sowohl die Herkunft der Daten auf denen die Analyse durchgeführt wird, als auch spezifische Implementierungsdetails, sowie Details zu der eingesetzten Bibliothek zur Sprache. Anschließend werden zu den einzelnen Algorithmen genauere Details ihrer Implementierung und Designentscheidungen besprochen.

4.1 Datenumfeld

Der Vergleich der Algorithmen wird auf Daten durchgeführt, die im Umfeld des Münchner Wissenschaftsnetzes (MWN) gesammelt wurden. Die Herkunft der Daten und deren Inhalte werden im folgenden näher beschrieben werden.

4.1.1 MWN

Um einen schnellen und umfassenden Überblick darüber zu bekommen, was das Münchner Wissenschaftsnetz ist und welche Bereiche es umfaßt, gibt es keine kürzere und bessere Beschreibung, als die von den Verantwortlichen selbst verfaßte. Deshalb soll der folgenden Absatz aus dem MHN-Überblick des Leibnitz-Rechenzentrums hier ungekürzt wiedergegeben werden.

„Das Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaften (LRZ) betreibt das Münchner Hochschulnetz (MHN). An dieses Netz sind der größte Teil der Standorte der Münchner Hochschulen und Fachhochschulen im Münchner Raum sowie viele Studentenwohnheime angeschlossen. Wir sprechen auch vom Münchner Wissenschaftsnetz (MWN), da auch außer-universitäre Einrichtungen wie z.B. die Bayerische Staatsbibliothek, einige Museen und mehrere Institute der Max-Planck-Gesellschaft über das Netz versorgt werden. Das MWN besteht aus einem Backbonenetz, an dem über Router und Switches die Netze der Einrichtungen an den verschiedenen Standorten angeschlossen sind. Die Router und Switches sind untereinander je nach Bandbreitenbedarf des einzelnen Standorts mittels Gigabit-oder Fast-Ethernet (1000 / 100 Mbit/s) verbunden. Das physische Medium besteht in der Regel aus Glasfaserstrecken, die von der Deutschen Telekom AG, den Stadtwerken München bzw. der Firma M"net langfristig angemietet worden sind.“ [21]

Die Daten, auf denen im Rahmen dieser Diplomarbeit die Analysen durchgeführt wurden, sind an der Nahtstelle des hier beschriebenen MWN zum deutschen Gigabit-Wissenschaftsnetz [22] gesammelt worden. Um eine Vorstellung über die Menge des hier auftretenden Verkehrs zu bekommen, gibt das LRZ die transferierte Datenmenge auf diesem Uplink mit „zurzeit 24 TByte an eingehenden und 30 TByte an ausgehenden Daten“ [21] an. Die verwendeten Sniffer bekommen ihre Daten an dieser Stelle (siehe Abbildung 10: am CSRWAN-Router) über den Monitorport des Routers als Kopie des realen Verkehrs zur Aufzeichnung zugeteilt. Über Replikatoren in der Glasfaser werden diese Daten dann auf mehrere Sniffer zur Weiterverarbeitung verteilt. Bei der Replikation des Netzwerkverkehrs wird, vereinfacht ausgedrückt, ein Teil des Lichtes, das die Daten durch die Glasfaser transportiert, über ein Prisma aus der Glasfaser ausgelenkt und anschließend verstärkt, während der Großteil des Lichtes unverändert weiter durch die Glasfaser läuft. Der ausgelenkte Teil trifft dann über entsprechende Netzwerkkarten auf den Sniffern ein und kann dort live beobachtet und verarbeitet, oder auch zur späteren Bearbeitung mitgeschnitten werden. Die folgende Abbildung zeigt im

Überblick die MWN-Backbone-Infrastruktur und verdeutlicht so, wie viele Computer und Netzwerke über dieses Backbone angeschlossen sind.

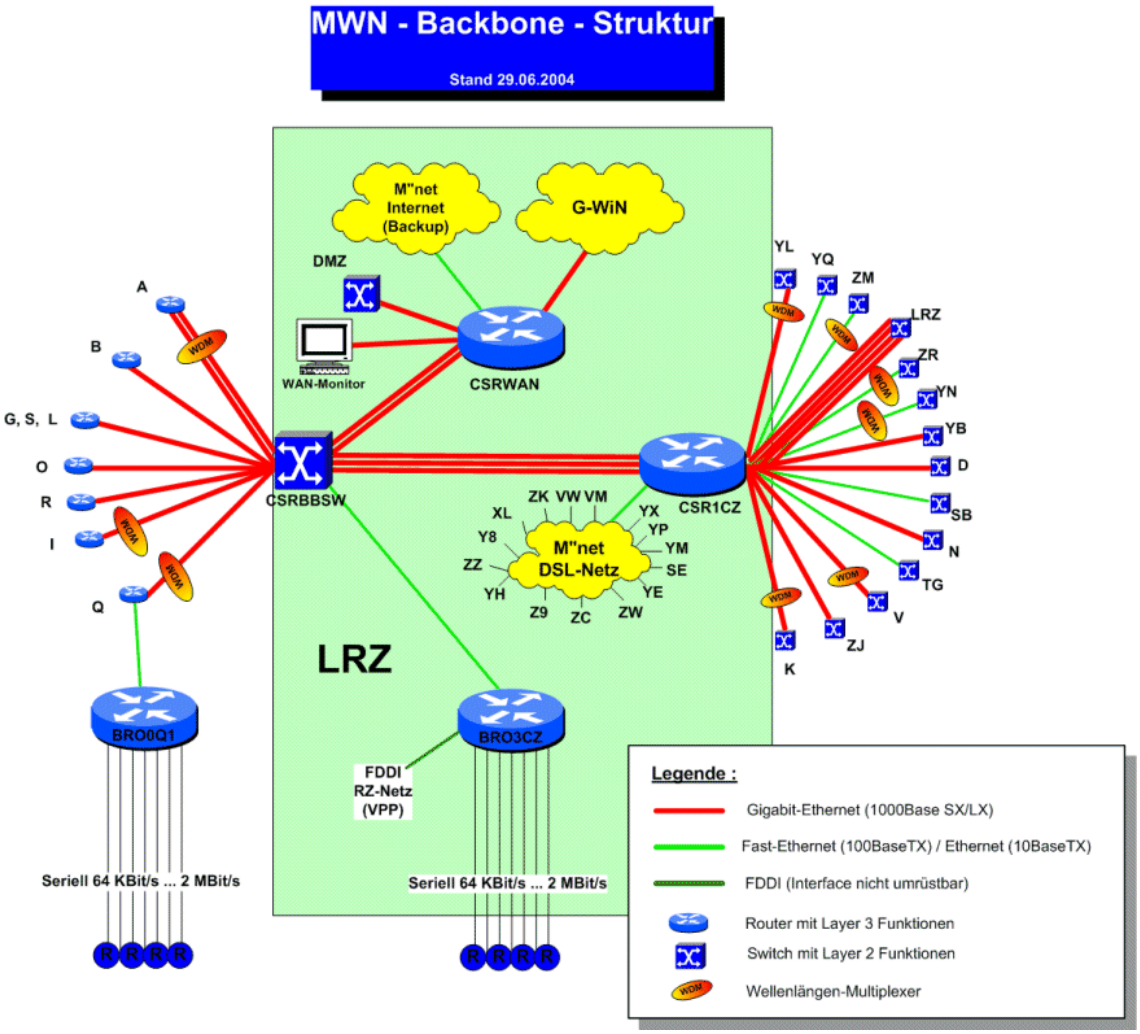

Abbildung 10: Backbone-Struktur des MWN [21]

Für das Auffangen der Traces wurde der Rechner *flamingo* eingesetzt. Dieser bekommt seine Daten wie oben beschrieben aus dem optischen Splitter, der vom Monitoring-Port gefüttert wird. *Flamingo* ist ein Computer mit zwei Intel Xeon 3.06GHz CPUs die jeweils Hyperthreading unterstützen, also insgesamt vier virtuelle CPUs zur Verfügung stellen. Er verfügt über 2 GB Hauptspeicher und ist mit einem Hardwareraid (ca. 600 GB) ausgestattet. Als Betriebssystem kommt auf flamingo FreeBSD in der Version 5.2.1 zum Einsatz. Die eigentliche Verarbeitung erfolgte dann auf dem nur für derartige Verarbeitungen vorgesehenen Rechner *condor*. *Condor* ist ein Sun Fire v880 Server mit 8 CPUs (sparcv9+vis2) und 32 GB Hauptspeicher. Dadurch war es problemlos möglich mehrere Algorithmen parallel zu testen. Als Betriebssystem war Sun Solaris 9 im Einsatz. Die Festplattenkapazität von *condor* liegt jenseits der 1TB-Marke.

4.1.2 Traces

Für den Vergleich der Algorithmen in dieser Arbeit wurden vier große Traces an der oben erwähnten Stelle erzeugt. Die Traces umfaßten jeweils 10GByte Daten, mit je ca.

140 Millionen Paketen. Der erste Trace *cs-full* beinhaltet die kompletten Daten, die an das G-WiN übergeben wurden, den international beliebte Web- und FTP-Server *leo.org* eingeschlossen, ohne jegliche Einschränkung bezüglich der beteiligten Ports. Zwei weitere Traces *cs-1* und *cs-2* enthalten ebenfalls komplette Daten aus dem Netzwerk der TU München, allerdings schließen sie *leo.org* wegen seines großen Verkehrsaufkommens aus. Der vierte Trace *login* enthält dann lediglich interaktiven Verkehr aus dem Netzwerk der TU München, also nur Pakete an denen, an einem der beiden Endpunkte, die Ports 22 (SSH), 23 (Telnet) oder 513 (Rlogin) beteiligt sind. Daraus ergeben sich, bei der gleichen Größe der Traces, unterschiedlich lange Zeiträume die überwacht wurden. Zusätzlich dazu wurden dann aus den vorhandenen Traces noch weitere kleinere Subsets entnommen, die nur spezielle Bereiche des TU-Netzwerkes umfassen. Dabei wurden die Traces auf zwei Subnetze unterschiedlicher Größe *net* (Lehrstuhl VIII) und *halle* (Computerraum der TU München) beschränkt. Daraus ergeben sich die für den Vergleich eingesetzten vier kleinen Traces *net-cs* und *net-login*, sowie *halle-cs* und *halle-login.*

Im folgenden wird zu jedem dieser Traces dargelegt, wie viele Pakete enthalten sind, wie viele TCP-Verbindung der Trace enthält und wie viele analysierte Verbindungen darin zu finden sind. Darüber hinaus wird eine Statistik der häufigsten benutzen Ports in den Traces gezeigt, damit ein Eindruck entsteht, welche Dienste von dem Trace überwacht worden sind. Der Trace *net-cs* enthielt in seinen 122MB ca. 1,6 Millionen Pakete. Von den 20351 im Trace enthaltenen TCP Verbindungen wurden 4445 durch die Algorithmen analysiert. Der Trace enthielt sehr viel Traffic auf hohen Ports (*32808* 59,2%, bzw. *19218* 59,2%). Von den bekannten Diensten sind im Trace vor allem http (12,1%) und SSH (2,5%) enthalten. Im zweiten Trace *net-login*, der aus 2,6 Millionen Paketen in 193MB besteht, wurden alle 638 Verbindungen analysiert, da es sich hier um vorgefilterte Daten interaktiver Sessions handelt. Im Trace sind ausschließlich SSH-Verbindungen zu finden. Im Trace *halle-cs* sind in seinen 623MB 8,6 Millionen Pakete enthalten. Von den 15.941 Verbindungen wurden 2122 analysiert. Die in diesem Trace am häufigsten genutzten Dienste sind SSH (33,1%) und http (19,0%). Im vierten und mit 881MB größten Trace *halle-login* sind ca. 12,2 Millionen Pakete enthalten. Nachdem es sich auch hier um vorgefilterte Daten interaktiver Sessions handelt, wurden wieder alle 1914 Verbindungen analysiert. Ähnlich wie beim *net-login* Trace sind auch in diesem Trace, bis auf 4 Telnet-Verbindungen nur verschlüsselte SSH-Sessions zu finden.

Im Trace *net-cs* wurde zur Kontrolle ein Stepping Stone eingebracht, der von den Algorithmen erkannt werden sollte. Bei diesem Stepping Stone wurde eine SSH-Verbindung von einem Computer, der durch eine DSL-Einwahl mit dem Internet verbunden war, zu dem Lehrstuhlcomputer *sora.net.informatik.tu-münchen.de* aufgebaut. Nach einer gewissen Zeit wurde von diesem Computer eine weitere SSH-Verbindung zum Server *www.jason.de* aufgebaut und dort einige Kommandos ausgeführt. Anschließend wurde diese Verbindung beendet und nach einer weiteren Zeit auch die ursprüngliche Verbindung zu *sora.net.informatik.tu-muenchen.de* geschlossen.

4.2 Implementierungsumfeld

Die Implementierung der Algorithmen wurde in C++ und der Scriptsprache, die das eingesetzte IDS zur Verfügung stellt, durchgeführt. Für die Waveletanalyse wurde auf eine C-Bibliothek zurückgegriffen, die die benötigten Funktionen zur Verfügung stellt. Lediglich einer der Algorithmen lag bereits in einer zum Vergleich geeigneten Form vor. Die verbleibenden drei Algorithmen wurden mit einem gemeinsamen Teil

programmiert, der dann nur die spezifischen Anteile der Algorithmen zur gegebenen Zeit aufruft.

4.2.1 Bro

Für die Implementierung wurde eine Integration in das Intrusion Detection System Bro [23] gewählt, da in Bro bereits eine vollständige TCP-State-Engine sowie ein Reporting- und Messagesystem implementiert ist und somit eine ideale Grundlage für diese zusätzlichen Module bildet.

Bro selbst ist in C++ geschrieben und bietet viele Schnittstellen, an die weitere Plugins angedockt werden können. Darüber hinaus bietet Bro eine eigene Scriptsprache, in der Teile der Algorithmen implementiert werden können. Bro bietet ein Framework aus Objekten, die die unterschiedlichen Bereiche der Netzwerkkommunikation abbilden. Diese Objekte bilden sowohl protokollspezifische als auch eher allgemeine Bereiche wie das Eventhandling oder die Übergabe von Parametern aus der Scriptebene in den C++ Bereich, und umgekehrt, ab.

Bro versucht mit seiner Architektur die Gegebenheiten der Intrusion Detection im breitbandigen Umfeld möglichst gut zu meistern und die auftretenden Datenmengen auf sinnvolle Art und Weise aufzubereiten. Dazu ist die Architektur von Bro in ein Schichtenmodell untergliedert, das das Informationslevel soweit reduzieren kann, daß es auch bei viel Netzwerkverkehr möglich ist Analysen durchzuführen.

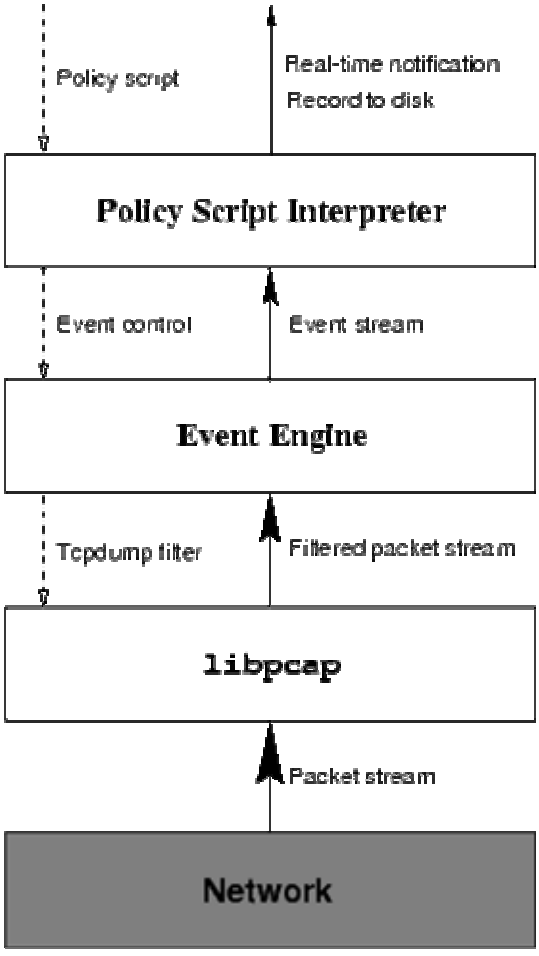

Abbildung 11: Schichtenmodell von Bro [24]

Die Datenmenge, die auf der Netzwerkebene eintrifft wird, wie in der Abbildung 11 schön zu erkennen ist, nach oben durch die Schichten gereicht. Dabei wird der Paketstrom in der ersten Ebene durch die Paketbehandlungs-bibliothek *libpcap* bearbeitet. Mit ihr ist es möglich den Verkehr anhand typischer Netzwerkkriterien, wie zum Beispiel Quell- oder Zieladresse, zu filtern. Der so entstehende gefilterte Netzwerkstrom erreicht nun die zweite Schicht, in der aus den Verbindungen und Paketen Events für die höher liegende API erzeugt werden. Diese Events werden dann auf die Scriptebene weitergereicht, wo dann nur noch die Events bearbeitet werden, die für den jeweiligen Algorithmus relevant sind. Die drei neu implementierten Algorithmen setzen auf der Ebene der Event Engine an und erzeugen lediglich Events, wenn sie Statistiken generieren, oder einen Stepping Stone

gefunden haben. Der On/Off Algorithmus dagegen setzt nur zu einem geringen Teil auf der Event Engine auf und führt den Hauptteil seiner Analyse auf Scriptebene durch.

4.2.2 Wavelet Bibliothek

Für die Implementierung der Algorithmen wurde eine Wavelet Bibliothek benötigt. Diese Bibliothek sollte mehrere Aspekte für eine gute Integration erfüllen. Die Bibliothek sollte möglichst schlank sein, da von den Routinen der Bibliothek nur ein sehr geringer Teil benutzt wird, der die Waveletanalyse auf einem eindimensionalen Vektor durchführt. Viele der existierenden Bibliotheken stellen ein großes Spektrum von Funktionen zur Verfügung, die auf der einen Seite, bei den eher grafisch motivierten Bibliotheken, Funktionen zur Kompression und Dekompression von verschiedenen Bildformaten enthalten. Auf der anderen Seite, bei den eher mathematisch motivierten Bibliotheken, sind oft noch komplette Implementierungen der Fouriertransformation und anderer Transformationen enthalten, die nicht benötigt werden. Darüber hinaus sollte die Bibliothek eine gut dokumentierte API haben, und sich problemlos in die C++-Umgebung von Bro integrieren lassen. In der Grundauswahl sollte die Bibliothek vor allem aber auch performant sein. Da sie hier im Rahmen der Diplomarbeit zum Einsatz kommt, war eine wichtiger Aspekt auch, daß sie aus dem OpenSource-Bereich stammt und somit kostenfrei einsetzbar ist.

Die genannten Aspekte wurden von keiner der gefundenen Bibliotheken vollständig erfüllt, so daß ein bestmöglicher Kompromiß gefunden werden mußte. Die jetzt eingesetzte „Wavelet Bibliothek“ [25] bietet bis auf die Fokussierung auf die Geschwindigkeit alle geforderten Aspekte. Es konnte in der Implementierung nicht festgestellt werden, daß die Performance der Bibliothek einen negativen Einfluß auf die gesamte Laufzeit hat, sondern vielmehr, daß die Laufzeit innerhalb der Bibliothek weit weniger in die Laufzeit eingeht, als die der restlichen Teile des Algorithmus.

4.2.3 Gemeinsamkeiten der Implementierung

Für alle Algorithmen gilt gleichermaßen, daß in Bro eine spezialisierte Instanz eines *TCP_EndpointAnalyzers* abgeleitet wurde, in deren Methoden die Analyse durchgeführt wird. Dieser Schritt ist nötig, da bei jedem ankommenden Paket, je nach eingesetztem Algorithmus, teils komplexe Arbeiten ausgeführt werden müssen. Wie schon aus den Algorithmen erkennbar, wird hier lediglich eine Analyse von TCP-Verbindungen durchgeführt, da zum einen interaktive Sessions über eine UDP-Verbindung wenig Sinn machen, da ja keine Sicherheit für den Empfang der Pakete gewährleistet werden kann. Zum anderen basieren die Algorithmen zum Teil auf TCP-spezifischen Vorraussetzungen ohne die ein Funktionieren der Algorithmen nicht möglich ist.

Im Wesentlichen wurden bei dem TCPEndpointAnalyzer-Objekt die Methoden *DataSent()* und *Done()* überschrieben, welche beim Erhalt eines neuen Pakets bzw. bei der Beendigung einer Verbindung aufgerufen werden. Dieser abgeleitete Analyzer wurde dann der Bro internen Liste hinzugefügt, so daß bei allen neu ankommenden Verbindungen und Paketen zusätzlich zu den sonst gewählten Analyzern auch eine Verarbeitung in Hinblick auf die Stepping Stone Analyse durchgeführt wird. Mit dem in der Bro Scriptsprache geschriebenen Teil der Analyse kann nun ausgewählt werden welche der Algorithmen zum Einsatz kommen sollen und auch welche Werte die Parameter der Algorithmen annehmen sollen. So ist es möglich zu überprüfen, bei welchen Parametern gute oder schlechte Ergebnisse in Bezug auf die Erkennungsrate erzeugt werden. Weiterhin ist es möglich über die Scriptebene zu filtern welche Pakete

aus dem Trace, oder auch im Livebetrieb, überhaupt zur Analyse herangezogen werden sollen.

Bei der Neuimplementierung der drei Algorithmen im Rahmen dieser Diplomarbeit wurde eine von den restlichen Analysen in Bro unabhängige Lösung angestrebt. Hierfür wurde ein weiterer Analyzer abgeleitet und in die globale Liste von Bro eingehängt. Zusätzlich wurde ein weiteres globales Objekt erzeugt, das für administrative Zwecke herangezogen wird und den Hauptteil der Analysen durchführt. Dieses globale Objekt, der *SteppingDetector*, verwaltet die benötigten Listen von Objekten die zur Datensammlung herangezogen werden. Für die Listen wurde auf Objekete der STL zurückgegriffen, da hier von einer sauberen und effizienten Implementierung ausgegangen werden kann. Dabei werden schon zur Laufzeit nur diejenigen Daten gespeichert bzw. verarbeitet, die auch zur Analyse herangezogen werden sollen. Die Auswahl, welche der folgenden Algorithmen verwendet werden sollen, wird auf Scriptebene getroffen. Standardmäßig werden die Daten für alle Algorithmen gesammelt und jeder Algorithmus zu gegebener Zeit ausgeführt. Darüber hinaus wird eine Liste verwaltet, die zur späteren Erzeugung von *stepping_found* Events benötigt werden, damit eine Referenz auf die beteiligten *TCP_Connections* übergeben werden kann.

Das *stepping_found* Event wird von den einzelnen Erkennungsalgorithmen erzeugt, wenn ein Stepping Stone erkannt worden ist. Dieser Event bekommt die beteiligten Verbindungen, einen beschreibenden Text, sowie eine Trefferwahrscheinlichkeit übergeben. Auf Scriptebene wird dann eine Meldung ausgegeben und die Meldung auch in eine Logdatei geschrieben, in der alle gefundenen Stepping Stones vermerkt werden. Darüber hinaus wird noch eine Meldung in einer allgemeineren Datei im CSV-Format erzeugt, die in beliebige Tabellenkalkulationsprogramme oder Datenbanken importiert werden kann, um dort weitere Auswertungen zu machen. Die so erzeugten CSV-Dateien werden im späteren Verlauf der Arbeit in eine Tabellenkalkulation importiert, um die Analyse der Verwendbarkeit der Algorithmen zu unterstützen und eine vergleichende Auswertung machen zu können.

Vom *SteppingDetector* werden auch in regelmäßigen, konfigurierbaren Abständen statistische Daten über die benötigte Laufzeit und den aktuellen Speicherverbrauch der Algorithmen mittels eines *stepping_stats* Events auf die Scriptebene durchgereicht. Die so generierten statistischen Daten werden wiederum formatiert auf dem Bildschirm ausgegeben, in eine Logdatei geschrieben und als Rohdaten im CSV-Format in eine Datei zur weiteren Aufbereitung gespeichert. Diese Statistikgenerierung kann auch von der Scriptebene aus deaktiviert werden, so daß hierdurch weder Ressourcen verschwendet werden, noch unnötige Bildschirmausgaben gemacht werden.

4.2.4 Testparcours

Zur Überprüfung der Funktionalität der Algorithmen, die im Rahmen dieser Diplomarbeit implementiert wurden, diente ein selbst erzeugter Trace von 1,8 MB Größe (18892 Pakete), der neben 8 SSH-Verbindungen auch normalen Netzwerkverkehr wie http-Verbindungen oder Emailverkehr enthält. Unter Zuhilfenahme dieses Traces wurde hier die Plausibilität der Algorithmen untersucht.

Diese 8 SSH-Verbindungen bestehen aus zwei gleichzeitigen Stepping-Stone-Ketten von verschiedenen Computern auf den zentralen Computer auf dem der Trace erzeugt wurde und von diesem wieder weg zu anderen Computern. Gleichzeitig dazu sind in dem Trace noch zwei SSH-Verbindungen, die als Gegenprobe dienen und nicht zu einer

dieser Ketten gehören, aber die selben Ziele beinhalten. In diesen beiden SSH-Verbindungen wurden ähnliche Kommandos zu ähnlichen Zeitpunkten ausgeführt, wie in den anderen, zu den Stepping Stones gehörenden, Verbindungen. Schematisch ist der Testparcours in der Abbildung zu erkennen.

Die Algorithmen mußten aus diesem Trace die zwei Stepping-Stone-Ketten mit ihren jeweils drei beteiligten SSH-Verbindungen erkennen, ohne dabei einen der zwei zur Ablenkung dienenden Verbindungen als Stepping Stone zu markieren.

4.2.5 Vorfilterung

Im hier betrachteten Umfeld von breitbandigem Netzwerkverkehr kann es bei allen vorgestellten Algorithmen leicht dazu kommen, daß sowohl die CPU-Belastung als auch der Speicherverbrauch die Grenzen der eingesetzten Maschinen sprengt oder zumindest jenseits sinnvoller Werte befördert, da zum Teil gewaltige Datenmengen vorgehalten werden müssen. Gerade wenn auf Netzwerkverkehr gearbeitet wird, der große Mengen FTP- oder HTTP-Verkehr beinhaltet, d.h. große Mengen von Verkehr enthält, der aus langen Verbindungen mit vielen Paketen besteht, müssen u.U. für jedes dieser Pakete seine Ankunftszeit, o.ä. gespeichert werden. Die angesprochene hohe CPU Belastung liegt meist an vielen langen Verbindungen, unabhängig von der Auslastung der Verbindungen. Da bei den Analysevorgängen in der Regel zu einem gegebenen Strom jeder andere gerade aktive Strom in Relation gesetzt werden kann, und somit eine Analyse vorgenommen werden muß, wird die CPU Belastung zwangsläufig höher, je mehr Verbindungen überwacht werden müssen, zu denen noch keine Aussage getroffen werden kann.

Diese Überlegungen führen zu einem zweistufigen Filtersystem wie es auch in der vorliegenden Implementierung im Einsatz ist. Die erste Stufe des Filters setzt bereits auf Scriptebene an und schränkt hier global die Menge der Pakete ein, die tatsächlich von Bro verarbeitet werden. Durch diesen Filter kann das Augenmerk auf bestimmte Verbindungscharakteristika beschränkt werden. So ist es zum Beispiel möglich hier über den Filter gezielt nur Pakete zu verarbeiten, die im Bereich der interaktiven Session (Telnet/SSH/Rlogin) liegen oder gezielt nur gewisse Subnetze zu analysieren. Die zweite Stufe der Filterung passiert dann in der eigentlichen Implementierung und versucht die Anzahl der nötigen Vergleiche zu reduzieren. Eine sehr simple Regel in diesem Bereich filtert alle Vergleiche heraus, bei dem zwei Ströme einer Verbindung miteinander verglichen werden sollen, da es hier keine Korrelation geben kann. Weiterhin werden keine zwei Verbindungen miteinander verglichen, die sich auf einem gemeinsamen Computer einen gemeinsamen Port teilen. Bei diesen Verbindungen ist davon auszugehen, daß beide Verbindungen unabhängig voneinander einen Dienst (beispielsweise SMTP) dieses Computers nutzen. Im Offlinebetrieb ist darüber hinaus auch ein dreistufiges System denkbar, wobei hier allerdings die zusätzliche Stufe die selben Funktionen bietet wie die erste Stufe der Implementierung. D.h. diese Stufe bietet lediglich die zusätzliche Möglichkeit, die aufzuzeichnende Menge der Daten zu reduzieren, um sich so ein Bild über einen längeren Zeitraum machen zu können ohne die Festplattenkapazitäten der aufzeichnenden Stelle überzustrapazieren.

4.3 Kandidaten für den Vergleich

Im folgenden werden noch ein mal kurz spezielle Details der Implementierung vorgestellt. Dabei werden sowohl die beteiligten Klassen und ihre Funktion erläutert, als auch spezielle Entscheidungen der Implementierung der Algorithmen näher beschrieben.

4.3.1 On/Off Algorithmus

Der On/Off Algorithmus wurde bereits in Bro implementiert als das Paper von Paxson und Zhang vorgelegt worden ist. Bei dieser Implementierung wurde ein zweistufiges System eingesetzt. Der erste Ansatzpunkt des Algorithmus stellt wie bereits beschrieben die Objekteebene von Bro dar. Hier wurden auf der C++-Ebene Klassen des Typ *TCP_EndpointAnalyzer* und *TCP_Analyzer* abgeleitet, die dazu dienen gewisse Vorarbeiten zu leisten, bevor dann die Hauptarbeit auf Scriptebene realisiert wird. Darüber hinaus wird ein Objekt *SteppingStoneManager* definiert, welches administrative Aufgaben für das Verbindungshandling übernimmt.

Der *SteppingStoneManager* wird, ähnlich wie der *SteppingDetector* der anderen Algorithmen, als globales Objekt instanziiert und als Referenz an die Objekte *SteppingStoneAnalyzer* und *SteppingStoneEndpoint* übergeben. In diesem Objekt wird zum einen ein, für einen Programmablauf eindeutiger, Integer-Schlüssel generiert, mit dem dann auf Scriptebene weitergearbeitet wird. Darüber hinaus verwaltet das Objekt eine geordnete Liste aller *SteppingStoneEndpoints* die anhand der Zeiten sortiert ist, wann dieser *SteppingStoneEndpoint* seine letzte ON-Periode begonnen hat. Bei Ankunft jedes Paketes wird diese Liste dann soweit durchlaufen, bis ein *SteppingStoneEndpoint* gefunden wird, dessen Anfang der ON-Periode innerhalb eines gewissen Grenzwertes zur aktuellen Zeit liegt. Alle vorher durchlaufenen *SteppingStoneEndpoints* werden als abgeschlossen betrachtet, da die Abweichung zwischen den ON-Perioden so groß ist, daß es unwahrscheinlich ist, daß diese *SteppingStoneEndpoints* wirklich zu einem Stepping Stone gehören. Auf der Scriptebene werden schließlich die Verbindungen miteinander verglichen und festgestellt, ob ein Verbindungspaar einen Stepping Stone bildet. Dieser Vergleich erfolgt über eine Tabelle, in der die Anzahl der aufgetretenen Übereinstimmung der ON-Perioden gezählt werden und eine Tabelle, die die Anzahl der aufeinanderfolgenden aufgetretenen Übereinstimmung der ON-Perioden beinhaltet. Schlußendlich wird auf Scriptebene auch die Ausgabe der Verbindungsdaten vollzogen, wenn ein Stepping Stone erkannt wurde.

4.3.2 IPD Algorithmus

Bei der Implementierung des IPD-Algorithmus wird die Hauptarbeit auf der C++-Ebene durchgeführt, da gerade bei der Analyse viel CPU-Zeit verbraucht wird und bei einer Implementierung in C++ eine höhere Geschwindigkeit zu erwarten ist. Der IPD-Algorithmus muß für seine Analyse die zeitliche Distanz zwischen den Paketankunftszeiten heranziehen. Die Daten werden für jeden Netzwerkstrom einzeln erfaßt. Die Erfassung der Differenz erfolgt bereits zum Zeitpunkt der Paketankunft. Diese Vorgehensweise ist zwar aus Sicht der Speicherperfomance nicht die beste Wahl, weil hier u.U. unnötigerweise Daten im Zusammenspiel mit dem Waveletalgorithmus, der die exakten Zeiten der Paketankunft benötigt, vorgehalten werden. Aus diesen exakten Zeiten würden sich beim Analyselauf des Algorithmus die Zeitdifferenzen errechnen lassen. Allerdings müßte dann bei jedem Analyselauf der komplette Zeitdifferenzenvektor für den IPD-Algorithmus berechnet werden, was eine wesentlich höhere CPU-Belastung bedeuten würde im Vergleich zu einer geringen Speichermehrbelastung. Daher wurde die Entscheidung zugunsten einer speicherungüstigeren Programmierung gegenüber der performaceungüstigeren Programmierung gewählt.

Die theoretische Beschreibung des Algorithmus liefert leider keine Anhaltspunkte darüber, zu welchem Zeitpunkt die Analyse auf den Zeitdifferenzenvektoren durchgeführt werden sollen. Da die Analyse eine eher zeitintensive Angelegenheit ist,

mußten hier geeignete Werte gefunden werden, wann die Analyse durchgeführt wird. Der triviale Ansatz, die Analyse bei jedem Paket zu machen, scheitert an der viel zu großen Rechenzeit, die hierfür aufgewendet werden muß. Der zweite, ebenfalls triviale Ansatz, die Analyse bei der Beendigung der Verbindung durchzuführen, verschenkt leider den Vorteil des Algorithmus, daß bereits zur Laufzeit der Verbindung eine Aussage getroffen werden kann. Daher wurde eine Variable *stepping_ipd_interval* eingeführt, die angibt, nach wie vielen Paketen eine Analyse durchgeführt werden soll. Am Ende einer Verbindung wird auf jeden Fall die Analyse durchgeführt, so daß ein Wert von -1 für *stepping_ipd_interval* dazu führt, daß die Analyse nur beim Verbindungsende durchgeführt wird. Die Auswertung der Algorithmen wird aber zeigen, daß selbst die Analyse am Ende der Verbindung immer noch Probleme bereitet. Bei der Analyse müssen, wie bereits beschrieben, Korrelationspunkte zwischen zwei Strömen gefunden werden. Das Finden dieser Punkte benötigt eine Laufzeit im Rahmen O(nN) mit der Anzahl n der Pakete in Strom 1 und der Anzahl N der Pakete in Strom 2, weil eben zu jedem möglichen Startpunkt einer Sequenz ein Startpunkt im zweiten Strom gefunden werden muß. Zusätzlich dazu muß aber bei der Analyse zu einem bestimmten Strom auch über alle Ströme m iteriert werden, um einen möglicherweise passenden Strom zu finden. Damit erhöht sich die Komplexität auf O(mnN). Um das ganze im erträglichen Rahmen zu halten ist dringend die erwähnte Vorfilterung nötig, um nur Ströme miteinander zu vergleichen, bei denen es überhaupt möglich ist, daß sie zu einem Stepping Stone gehören.

Der *SteppingDetector* holt sich bei der Analyse zusätzlich zum gegebenen Ausgangsstrom der Reihe nach jeweils einen anderen Strom aus der Liste und vergleicht dann jeweils ein Fenster der Größe *stepping_ipd_window_size* des einen mit den möglichen Fenstern des anderen Stroms. Der so für jede mögliche Kombination von Fenstern errechnete Vergleichswert wird nun mit dem Wert *stepping_ipd_cpf_delta* verglichen und die beiden Startpunkte der Fenster bei Überschreiten dieses Schwellwertes als Korrelationspunkte gespeichert.

Aus den sich daraus ergebenden Korrelationspunkten wird dann ein Übereinstimmungswert berechnet, der bei Überschreiten des Schwellwertes *stepping_ipd_cvf_delta* das *stepping_found* Event auslöst, der diesen Übereinstimmungswert als Trefferwahrscheinlichkeit erhält.

4.3.3 Wavelet Algorithmus

Für die Implementierung des Wavelet-Algorithmus gab es keine andere Wahl, als ihn komplett, bis auf die Ausgabefunktionen, in C++ zu realisieren, weil auf eine vorgefertigte Wavelet-Bibliothek zurückgegriffen werden mußte. Die Wavelet-Transformation selbst auf der Scriptebene zu implementieren, ist alleine aus Sicht der Geschwindigkeit unsinnig. Für die Analyse durch den Wavelet Algorithmus müssen zu jedem untersuchten Strom alle Ankunftszeiten der Pakete vorgehalten werden. Gerade bei langen und stark ausgelasteten Verbindungen kann dies zu einem hohen Speicherverbrauch führen. Wenn auch Serverprozesse oder Dateitransfer in den Bereich der Analyse fallen, werden hier große Datenmengen im Speicher gehalten, was durch den Algorithmus nicht zu vermeiden ist. Dem kann nur durch geeignete Vorfilterung entgegen getreten werden.

Die wirkliche Analyse wird für einen gegebenen Datenstrom in seinem Lebenszyklus nur ein Mal durchlaufen, und zwar zum Zeitpunkt an dem die Verbindung beendet wird. Dies passiert sowohl wenn die Verbindung normal und korrekt beendet wird, als auch wenn sie abgebrochen wird und anschließend durch einen Timeout als beendet

betrachtet wird. Bei dieser Analyse wird aus dem Vektor der Paketankunftszeiten zunächst die Start- und Endzeit bestimmt, aus der sich dann mit der Anzahl der Bereiche *stepping_wavelet_num_areas* die Schrittweite errechnet. Dabei wird der Zeitenvektor einmalig durchlaufen und für jedes Paket aus der Startzeit und der Schrittweite der Bereich bestimmt, in dem das Paket zu zählen ist.

Für diesen so berechneten Bereichsvektor wird nun mit einem Haar-Wavelet eine Wavelet-Transformation durchgeführt. Die selbe Prozedur wird nun mit jedem aktiven Strom, der nicht von der Vorfilterung ausgeschlossen wurde, durchgeführt und die so berechneten Koeffizienten mit den Koeffizienten des Ausgangsstromes verglichen. Überschreitet dieser Vergleichswert beim Vergleich den Schwellwert *stepping_wavelet_delta*, wird wieder ein *stepping_found* Event ausgelöst, der den berechneten Vergleichswert als Trefferwahrscheinlichkeit erhält.

Die effizientere Programmierung, die Wavelet-Transformation für den Ausgangsvektor einmalig zu berechnen und anschließend nur mit den Transformationen der einzelnen Ströme zu vergleichen, mußte fallen gelassen werden, da dadurch ein großes Speicherloch entstanden wäre. Dieses Speicherloch kommt daher, daß durch die Anforderungen der eingesetzten Bibliothek an dieser Stelle zwei Pseudobilder, die ein Pixel hoch und *stepping_wavelet_num_areas* breit ist, erzeugt werden müssen, die die Ausgangsdaten und die transformierten Daten aufnehmen. Auf diese Pseudobilder verweist dann jeweils ein Zeilenvektor, auf denen die eigentliche Transformation stattfindet. Es wäre jetzt nur möglich diese Zeilenvektoren als Referenz zu speichern, um anschließend die Vergleich durchführen zu können. Diese Zeilenvektoren haben allerdings nur eine interne Referenz auf die ursprünglichen Pseudobilder, so daß der Speicher dieser Pseudobilder nach Abschluß der Analyse nicht wieder freigegeben werden kann. Daher muß hier der, für die Performance nachteilige, Fakt in Kauf genommen werden, daß für jeden Vergleich zweier Ströme jeweils beide Transformationen berechnet werden müssen, obwohl theoretisch eine Transformation zwischengespeichert werden könnte.

4.3.4 Paket-Zähler Algorithmus

Die Implementierung dieses Algorithmus ist, wie die Einfachheit des Algorithmus nahelegt, nicht weiter komplex. Herzstück der Implementierung ist die Klasse *SteppingPair*, die das Zählen der Pakete übernimmt und einen von drei möglichen Zuständen für die Analyse annimmt. Bei jedem Paket, das auf einem Strom eintrifft, werden in der entsprechenden Methode des *SteppingDetectors* die *SteppingPairs* herausgesucht, an denen dieser Strom beteiligt ist und der Paketzähler des SteppingPairs für diesen Strom erhöht. Ist das Objekt bereits im Zustand „Stepping Stone“, passiert keine weitere Aktion. Überschreitet die gemeinsame Paketzahl der beteiligten Ströme den Wert *stepping_counter_pmax*, dann wird für dieses Paar überprüft, ob die Differenz der Paketzähler den Wert *stepping_counter_dmax* unterschreitet und gegebenenfalls der Zähler für das Vorkommen von „verdächtigen Vorfällen“ erhöht. Anschließend wird noch dieser Vorkommenszähler gegen den Wert *stepping_counter_omax* verglichen. Überschreitet der Vorkommenszähler den Schwellwert, dann wird das Objekt in den Zustand „Stepping Stone“ versetzt. Überschreitet die Paketdifferenz den Schwellwert *stepping_counter_dmax* kann davon ausgegangen werden, daß die beiden Ströme „kein Stepping Stone“ sind. Sonst bleibt das Objekt im Zustand „noch nicht feststellbar“.

Für den Algorithmus wird vom *SteppingDetector* eine Liste von *SteppingPairs* verwaltet, die auf Anforderung durchlaufen und überprüft wird. Bei dieser Überprüfung wird von allen, zu einem gegebenen Strom passenden, *SteppingPairs* abgefragt in

welchem Zustand sie sich befinden. Mögliche Zustände sind „Stepping Stone“, „kein Stepping Stone“ und „noch nicht feststellbar“. Wenn der Zustand „Stepping Stone“ gemeldet wird, wird wie gehabt ein *stepping_found* Event ausgelöst, welches bei diesem Algorithmus als Trefferwahrscheinlichkeit 100% bekommt, da hier, schon durch den Algorithmus bedingt, keine Abstufung getroffen werden kann. Alle SteppingPairs werden anschließend aus der Liste entfernt und ebenfalls aus den zu den jeweiligen Verbindungen gehörenden *SteppingStreams* gelöscht. Dieses Entfernen aus den Listen geschieht ebenfalls, wenn der Zustand „kein Stepping Stone“ zurückgeliefert wird. So lange der Zustand „noch nicht feststellbar“ geliefert wird, passiert an dieser Stelle nichts.

4.4 Ergebnis des Vergleichs

Nachdem die bisherige Arbeit einen Eindruck über die möglichen Algorithmen und ihre Implementierung gegeben hat, ist es jetzt an der Zeit die einzelnen Implementierungen miteinander zu vergleichen. Dabei gibt es eine Reihe von Gemeinsamkeiten, die alle Algorithmen aufweisen. Es gibt jedoch auch einige spezielle Erkenntnisse zu den einzelnen Algorithmen, die im Anschluß erläutert werden.

4.4.1 Gemeinsame Erkenntnisse

Bei der Auswertung der einzelnen Algorithmen ist zuallererst ein großes Problem mit den Traces aufgetreten. Keiner der Algorithmen hat es auf Anhieb geschafft, die großen Traces zu durchlaufen. Ein Teil der Algorithmen hatte bei der Fülle der Daten ein Problem mit der zeitlichen Verarbeitung, während ein anderer Teil die Speicherobergrenze von 4GB gesprengt hat. Auch eine Einschränkung an dieser Stelle, nur den interaktiven Verkehr (Ports 22,23 und 513) zu überwachen, hat keinen Erfolg gebracht. Der Grund dafür, daß auch nach dieser Einschränkung immer noch kein Algorithmus durchlief, konnte dann aus der näheren Betrachtung der gesammelten Daten erkannt werden. In allen Traces waren an unterschiedlichen Stellen große Mengen von kurzen SSH-Verbindungen. Diese Verbindungen sehen bei näherer Betrachtung so aus, als ob hier von einem (immer wieder unterschiedlichen) Rechner aus versucht wurde, ganze Subnetze im Netz der TU München zu kompromittieren. Dazu wurde von diesem Computer aus gleichzeitig, oder kurz zeitversetzt, zu unterschiedlichen Computern in der TU München eine SSH-Verbindung aufgebaut, die nach nur 5-10 Paketen wieder getrennt worden ist, also bei der die Authentifizierung gescheitert zu sein scheint. Da aber durch diesen nahezu gleichzeitigen Verbindungsaufbau schnell 50.000-60.000 gleichzeitige Verbindungen offen und zu überwachen waren, ist es nicht weiter verwunderlich, daß die Algorithmen wegen ihrer quadratischen Laufzeitcharakteristik an dieser Stelle zum Aufgeben gezwungen waren.

An der quadratischen Laufzeit, bzw. bei einigen Algorithmen auch dem quadratischen Speicherverbrauch, sind keine großen Verbesserungsmöglichkeiten zu erkennen, da eben jede Verbindung mit jeder anderen Verbindung ein mögliches Stepping Stone Paar ist. Hier stellt man auch bei den Referenzimplementierungen der einzelnen Algorithmen fest, daß oft eine effiziente Implementierung gefunden wurde, um zu einem gegebenen Strom aus einem Trace alle möglichen anderen Ströme zu finden, die zusammen einen Stepping Stone ergeben. Dieses Szenario ist für den realen Einsatz in einem IDS nicht ausreichend. Beim realen Einsatz, egal ob dieser in Realtime oder Offline durchgeführt wird, ist im Gegensatz dazu die Situation so, daß man zu allen Verbindungen alle möglichen Stepping Stones finden will. Daraus ergibt sich schon der zusätzliche Faktor, der die Algorithmen schwer belastet. Erst die Auswertung

von Tests auf den verkleinerten Traces (*net* und *halle*) haben dann brauchbare Ergebnisse liefern können.

Dabei fiel auf, daß die Laufzeit der Algorithmen, bis auf einen, relativ ähnlich ist. Lediglich im Speicherverbrauch lassen sich deutliche Unterschiede feststellen. Die oben erwähnte quadratische Charakteristik wirkt sich, je nach Algorithmus, zum Teil schon ab ca 4.000 gleichzeitigen Verbindungen so negativ aus, daß die Algorithmen stehen zu bleiben scheinen oder abstürzen, weil sie keinen Speicher mehr belegen können. Näheres dazu wird aber bei den einzelnen Algorithmen erläutert, da dort auch exakte Zahlen über den Speicher- und CPU-Verbrauch genannt werden können.

Auch in der Erkennungsrate schwanken die Ergebnisse sehr deutlich zwischen einer riesigen Menge, die auch viele unsinnige Verbindungen enthält und einer viel zu geringen Menge an erkannten Stepping Stones. Darin läßt sich, wie schon am Anfang dieser Arbeit erwähnt, die Problematik für den realen Einsatz erkennen, da hier beide Extreme dazu führen, daß man sich nicht wirklich sicher sein kann, ob Stepping Stones aufgetreten sind oder nicht.

Ein sehr interessanter Fakt der nebenbei aus der Auswertung der Tests gezogen werden konnte, ist die Tatsache, daß bei allen erkannten Stepping Stones keine unverschlüsselte Verbindung zu finden war. Es wurden, wie gesagt, zwar zum Teil auch eine Menge False-Positives gefunden, aber selbst unter diesen war keine Verbindung, die nicht zu einer SSH-Verbindung gehört. Das läßt den Schluß zu, daß schon jetzt eine Konzentration der Bemühungen auf timingbasierte Algorithmen wichtig ist. Da in Zukunft noch mehr Menschen auf die Sicherheit einer verschlüsselten Verbindung setzen werden, wird es immer schwerer werden, überhaupt Ergebnisse mit einem inhaltsbasierten Ansatz finden zu können. Zusätzlich steht zu vermuten, daß Personen, die Stepping Stones zu illegalen Zwecken benutzen, verstärkt auf die Verschlüsselung setzen, um von vorne herein nicht durch inhaltsbasierte Signaturerkennung aufzufliegen.

Leider ist es bei der nun folgenden Betrachtung der einzelnen Algorithmen schwierig, einen Vergleich zwischen dem Referenzalgorithmus und den Algorithmen, die im Rahmen dieser Arbeit erstellt wurden, zu ziehen, da die statistischen Daten des On/Off-Algorithmus nicht die Detailstufe aufweisen, die die neuen Algorithmen bieten können. Beim On/Off-Algorithmus muß hier auf die von Bro selbst zur Verfügung gestellten Daten zurückgegriffen werden, die auf der verwendeten Testmaschine leider nicht vollständig erzeugt worden sind, da gewisse Funktionen der C++-Laufzeitumgebung nicht wie erwünscht funktioniert haben. Bei den anderen Algorithmen hingegen wurden detaillierte Statistiken über das Verhalten der einzelnen Algorithmen gesammelt, die den tatsächlichen Verbrauch durch die Algorithmen zeigen, ohne die im Hintergrund ebenfalls benötigte Bro-Umgebung mit einzubeziehen. Umgekehrt hat allerdings die Bro-Umgebung in ihren Statistiken, wegen der oben beschriebenen Problematik, den Verbrauch der Algorithmen auch nicht in Betracht gezogen, so daß ein direkter Vergleich auf dieser Ebene auch nicht möglich war. Trotzdem sind die hier gewonnen Informationen insoweit miteinander vergleichbar, als das man eine Abschätzung finden kann, wie sich die einzelnen Algorithmen beim Betrieb auf breitbandigen Daten verhalten werden.

Nachdem, wie bereits beschrieben, ein Vergleich auf den großen Traces nicht aussagekräftig war, wurden die Daten, mit denen die einzelnen Algorithmen analysiert werden, auf vier der kleinen Traces beschränkt. Beim Vergleich wurden die Algorithmen nach einer maximalen Laufzeit von ca. 60 Minuten abgebrochen und der

bis dahin erreichte Zustand bewertet. Die kleinen Traces die hier zum Vergleich heran gezogen werden sind *halle-cs*, *halle-login* sowie *net-cs* und *net-login* wie sie in Kapitel 4.1.1 beschrieben worden sind.

4.4.2 On/Off Algorithmus

Bei den Tests des On/Off-Algorithmus ist vordergründig aufgefallen, daß der On/Off-Algorithmus in der vorliegenden Implementierung wegen seiner langen Entwicklungszeit soweit optimiert ist, daß er auf den großen Traces noch am ehesten, neben dem Paket-Zähler-Algorithmus, Chancen hat seine Arbeit erfolgreich zu verrichten. Zum Teil können die anderen Algorithmen sicherlich noch dahingehend optimiert werden, daß die Zahl der benötigten Vergleiche geringer ausfällt, aber das allein wird es nicht allen ermöglichen, auf den großen Traces zu laufen. Weiterhin muß man hier einschränken, daß bei der Implementierung des On/Off-Algorithmus von vorne herein eine Beschränkung auf die interaktiven Verbindungen eingesetzt wurde.

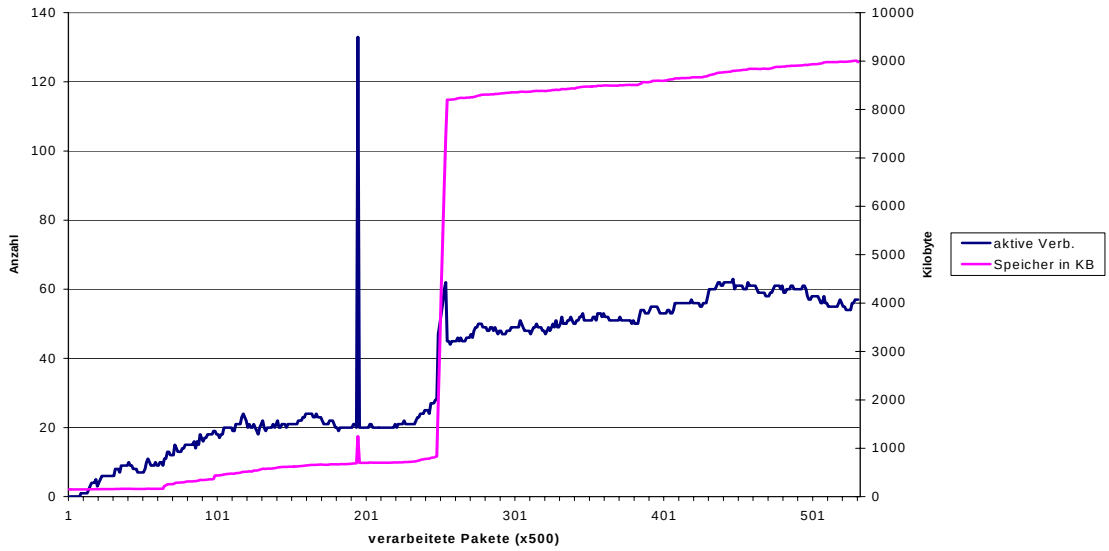

Abbildung 12: Speicherverbrauch des On/Off Algorithmus

Ein darüber hinaus wesentlich größeres Problem ist bei den Test auf den großen Traces zu Tage getreten, das auch in der Abbildung 12 des Speicherverbrauches deutlich sichtbar wird. Durch einen Fehler in der Programmierung, der leider in der zur Verfügung stehenden Zeit nicht auffindbar war, kommt es dazu, daß Verbindungen unter bestimmten Vorraussetzungen nicht aus dem Speicher entfernt werden. Deswegen erreichte der Algorithmus auch bei den Tests auf den großen Traces das 4GB Speicherlimit und wurde beendet. Allerdings war die Laufzeit des Algorithmus zu diesem Zeitpunkt auch schon bei ca. 42 Stunden angelangt, womit, wie bei allen anderen Algorithmen auch, der sinnvolle Zeitrahmen für die Analysen gesprengt war.

Im Trace *halle-cs* wurde vom On/Off Algorithmus kein Stepping Stone gefunden. Für diesen Trace benötigte der Algorithmus 73,7 Sekunden. Der maximale Speicherverbrauch lag bei 9MB. Der Verlauf ist aus der Grafik ersichtlich. Die Analyse auf dem Trace *halle-login* benötigte 7 Minuten und 11,1 Sekunde um 17 Stepping Stones zu finden. Der maximale Speicherverbrauch lag bei 5,7MB. Bei der Analyse des Traces *net-cs* wurden wiederum keine Stepping Stones gefunden, womit hier zumindest ein False-Negative zu verbuchen ist, da für die Diplomarbeit hier ein Stepping Stone

eingebracht worden ist. Für die Analyse benötigte der Algorithmus 3 Minuten und 3,2 Sekunden, bei einer maximalen Speicherauslastung von 30MB. Für die Verarbeitung des vierten Traces *net-login*, der bei allen Algorithmen zum Vergleich näher beschrieben wird, betrug die benötigte Zeit 100,6 Sekunden. Bei der Analyse konnten keine Stepping Stones gefunden werden. Der maximale Speicherverbrauch lag bei 5,6MB. Aus der Grafik ist gut ersichtlich, daß sich der Algorithmus hinsichtlich des Speicherverbrauches linear zu der Anzahl der aktiven Verbindungen verhält.

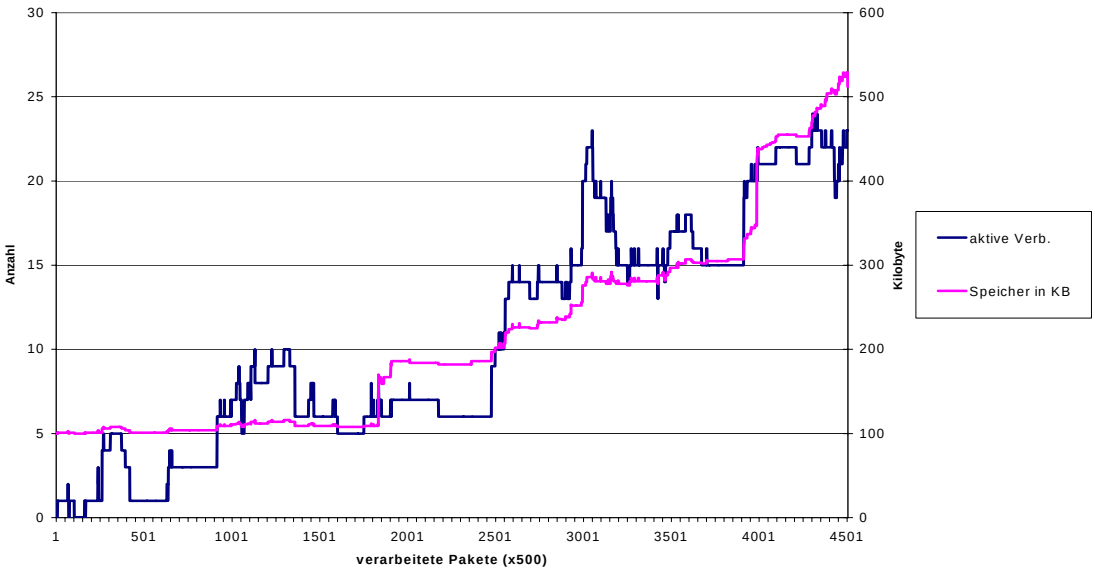

Abbildung 13: Speicherverbrauch des On/Off Algorithmus

Über den Verbrauch an CPU-Zeit kann, mangels der Möglichkeit zur Erfassung der entsprechenden Daten, keine genaue Aussage getroffen werden, aber anhand der oben genannten Zahlen ist auch hier ein linearer Zeitbedarf festzustellen. Dieser lineare Zeitbedarf zeigte sich auch bei dem Versuch auf den großen Traces, die allerdings durch das Speicherleck nicht zu Ende gebracht werden konnten.

4.4.3 IPD Algorithmus

Der IPD-Algorithmus schneidet hier im Vergleich deutlich schlechter ab, als die anderen Algorithmen, was aber durch seinen internen Aufbau zu erklären ist. Es hat sich gezeigt, daß die Art und Weise der Suche nach Korrelationspunkten einen sehr großen negativen Einfluß auf die Laufzeit des Algorithmus hat. Durch die Tatsache, daß für zwei gegebene Ströme der Längen n und m schon $n*m$ Anfangspunkte berechnet werden müssen, ist eine für das IDS nötige Suche über alle Ströme derart ressourcenraubend, daß der Algorithmus selbst bei den kleinen Traces jenseits aller realen Anwendbarkeit liegt. Bereits wenige Verbindungen mittlerer Länge (ca. 1500-2000 Pakete) bringen den Algorithmus zum Erliegen, da die CPF bei zwei dieser Verbindungen schon für 2,25 Millionen Startpunktkombinationen berechnet werden muß. Stehen jetzt mehrere dieser Kombination zur Überprüfung an, so wird ein solcher Analyselauf extrem lange dauern. Die hier angesprochene Problematik führte auch dazu, daß selbst die Versuche auf den kleinen Traces nicht alle bis zum Ende durchgeführt werden konnten. In einigen Fällen mußte der Algorithmus nach den oben erwähnten 60 Minuten abgebrochen werden. Darüber hinaus wurde in diesen Fällen

noch versucht ein Durchlaufen über einen längeren Zeitraum durchzuführen, um einen Abschluß erreichen zu können. Diese Versuche wurden dann aber auch nach zum Teil 74 Stunden ohne Ergebnis abgebrochen, weil damit der Kosten/Nutzen ad absurdum geführt wird.

Der Test auf dem *halle-cs* Trace konnte nach 38 Minuten und 11,4 Sekunden abgeschlossen werden. In dieser Zeit lag der maximale Speicherverbrauch zwischenzeitlich bei 281KB. Die längste Verbindung umfaßte 18765 Pakete. In dem Trace wurden vom IPD-Algorithmus 120 Stepping Stones gefunden. Bei den gefundenen Stepping Stones scheint es sich in allen Fällen um False-Positives handeln, die einfach zufällig ähnliche Schemata aufwiesen. Die Verbindungen gehen alle von einem Subnetz auf Server in einem anderen Subnetz und sind eher kurz, allerdings sind sie nicht so kurz, als daß es sich um fehlgeschlagene Loginversuche handeln kann. Alle Verbindungen sind hier SSH-Verbindungen und die meisten gehen in die selbe Richtung. Der Versuch mit dem Trace *halle-login* wurde nach 1 Stunde und drei Minuten abgebrochen. Zu diesem Zeitpunkt waren erst 56 Verbindungen analysiert worden und davon noch 20 aktiv. Die längste Verbindung zu diesem Zeitpunkt betrug 9411 Pakete. Der Speicherverbrauch war bei maximal 186KB angekommen. Aus den Zahlen sieht es so aus, daß im Anfang dieses Traces einige Verbindungen mit vielen Paketen enthalten sind. Daher müssen bei jedem Vergleich von zwei Strömen, wie oben beschrieben, zu viele Korrelationspunkte berechnet werden. Der Algorithmus hatte zu diesem Zeitpunkt 23 Stepping Stones gefunden, die nicht mit denen des On/Off Algorithmus übereinstimmen, aber wieder ähnliche Muster wie im ersten Fall aufweisen. Im weiteren wurde nun noch der Trace *net-cs* mit dem Algorithmus überprüft. Dieser Trace wurde in 10 Minuten und 57,2 Sekunden durchlaufen. Dabei wurden 6 Stepping Stones gefunden, von denen zumindest einer verifiziert werden kann, da er extra zu diesem Zweck durchgeführt worden ist. Auch die anderen Stepping Stones wirken wie tatsächliche Stepping Stones, konnten aber wegen der Verschlüsselung nicht verifiziert werden. Bei einer maximalen Länge eines Stromes von 14811 Paketen wurden maximal 143KB Speicher verbraucht.

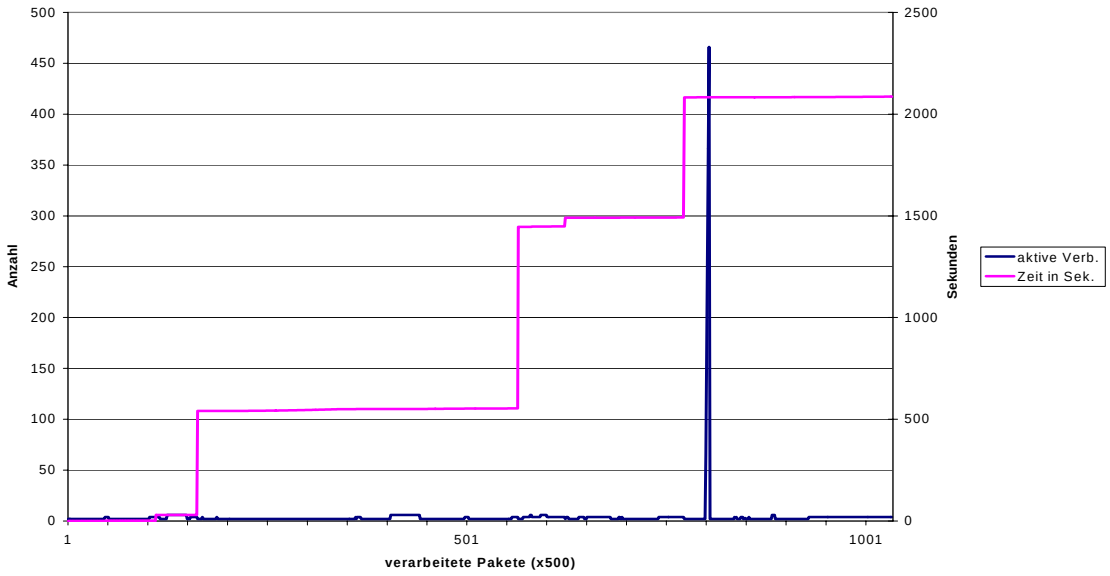

Abbildung 14: Zeitverlauf der IPD Algorithmus

Abschließend wurde noch der Trace *net-login* untersucht. Leider mußte auch hier der Algorithmus wieder nach 1 Stunde und 2 Minuten abgebrochen werden. Zu diesem Zeitpunkt waren 523 Verbindungen analysiert und 10 Stepping Stones gefunden, für die es wahrscheinlich ist, daß es sich tatsächlich um Stepping Stones handelt. Vom Hauptspeicher waren 118KB belegt. In der Grafik des Speicherverlaufs sieht man an den Spitzen, wie sich länger laufende Verbindungen auswirken. Eine Abhängigkeit von der Menge der aktiven Verbindungen ist dagegen nicht relevant. Betrachtet man die Grafik des Zeitverlaufes fallen einem sofort die Sprünge in der Zeit auf. Diese sind ein deutliches Zeichen dafür, daß an diesen Stellen eine Analyse auf mindestens zwei langen Strömen passiert ist. Dies tritt in der Regel dann auf, wenn eine langer Strom beendet wird, da er dann mit allen anderen Strömen verglichen werden muß.

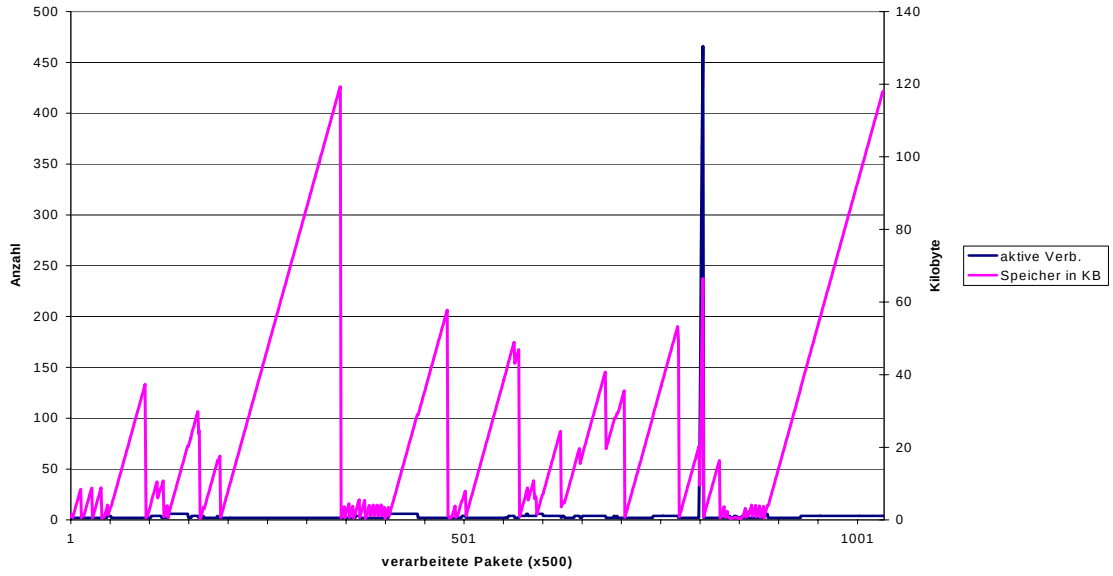

Abbildung 15: Speicherverlauf des IPD Algorithmus

4.4.4 Wavelet Algorithmus

Beim Durchführen der Tests mit dem Wavelet Algorithmus kam es zu sehr unterschiedlichen Ergebnissen, deren Schwankungen durch die Länge der zu untersuchenden Verbindungen zustande kommt. Die Erkennungsrate lag bei den Versuchen jeweils gleichbleibend hoch. Bei der Laufzeit hingegen kam es bei der Existenz langer Verbindungen zu einer hohen Verzögerung der Laufzeit. Die Verlängerung der Laufzeit ist beim Wavelet Algorithmus jedoch bei weitem nicht so extrem, wie beim IPD Algorithmus. Der untersuchte *halle-login* Trace enthielt eine SSH-Verbindung mit annähernd 350.000 Paketen. Allein das Vorhandensein dieser Verbindung im Trace führte dazu, daß die Laufzeit des Algorithmus explodiert ist. Auch hier ist es klar, daß es zu diesem Verhalten kommen muß, denn jede Verbindung, die beendet wird, muß sich selbst mit allen noch aktiven Verbindungen vergleichen. Das führt dazu, daß bei jeder Verbindung die beendet wird zumindest ein mal die kompletten 350.000 Pakete durchlaufen werden, um sie den Zeitintervallen zuzuordnen. Die anschließende Wavelet Transformation hat hierbei kaum einen Einfluß auf die Laufzeit, weil sie immer auf einem Vektor konstanter Länge arbeitet. Es hat sich dadurch auch gezeigt, daß die Verarbeitungsgeschwindigkeit der Wavelet Bibliothek

kaum ein Rolle spielt, im Vergleich zu der Zeit, die für das Einsortieren in die Zeitintervalle aufgewendet werden muß.

Wie bei den anderen Algorithmen auch, wurden die vier Traces der Reihe nach durchgeführt und ausgewertet. Beim Trace *halle-cs* waren nach 1 Minute und 18,4 Sekunden alle Verbindungen bearbeitet. Der maximale Speicherverbrauch während dieser Zeit war 281KB, bei einer maximalen Verbindungslänge von 18765 Paketen. Vom Wavelet-Algorithmus wurden in diesem Trace 10 Stepping Stones gefunden. Dabei gilt zu beachten, das der Schwellwert bei nur 70% gelegen hat, was sich im Vortest als ein brauchbarer Wert herausgestellt hatte, um die gegebene Aufgabe zu lösen. Von diesen 10 erkannten Stepping Stones liegen nur 2 über dem Schwellwert von 85%. Bei den verbleibenden zwei Stepping Stones könnte es sich tatsächlich um Stepping Stones handeln, was aber wieder wegen der Verschlüsselung nicht exakt nachprüfbar ist. Bei der Analyse der *halle-login* Traces mußte der Algorithmus nach 1 Stunde und 2 Minuten beendet werden. In dieser Zeit konnte der Algorithmus 413 Verbindungen analysieren. Bei deren Verarbeitung wurden 2473KB Speicher verbraucht. Die längste im Trace bis dahin verarbeitete Verbindung hatte 209280 Pakete. Diese immense Zahl führte auch dazu, daß der Algorithmus seine Arbeit nicht zu Ende bringen konnte, weil die Berechnung der Transformation für diese Verbindung eine lange Zeit in Anspruch nimmt. Nachdem die Transformation aber immer dann berechnet werden muß, wenn eine Verbindung beendet wird, führt das zu einer sehr langen Laufzeit. Es wurde versucht diesen Trace noch einmal separat zum Durchlaufen zu bringen, aber der Versuch wurde ebenfalls nach ca 70 Stunden beendet (Eine maximalen Länge von ca. 350.000 Paketen war erreicht). In dem Trace wurden 14 Stepping Stones gefunden, die sich mit den gefundenen Stepping Stones des IPD-Algorithmus decken. Der Trace *net-cs* war beim Test nun wieder nach 78,7 Sekunden mit der Analyse fertig. Der maximale Speicherverbrauch lag bei 143KB. Die längste bearbeitete Verbindung umfaßte 14811 Pakete. Es wurden keine Stepping Stones in diesem Trace erkannt. Somit ist auch hier mindestens der eine beabsichtigte Stepping Stone nicht erkannt worden.

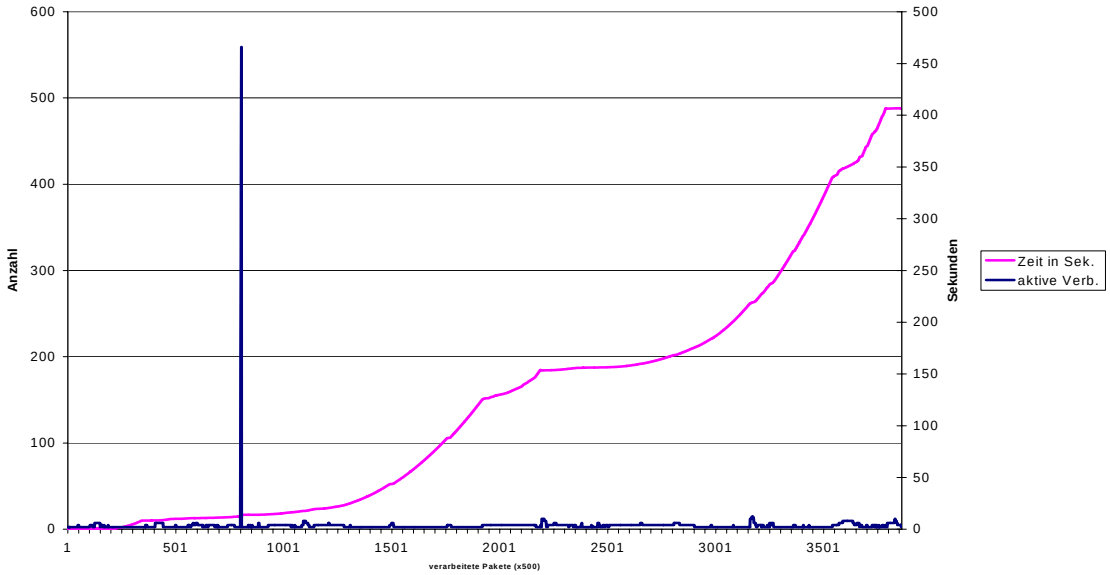

Abbildung 16: Zeitverlauf des Wavelet Algorithmus

Der abschließende Vergleichstest mit dem Trace *net-login* war nach 9 Minuten und 10,3 Sekunden beendet. Maximal wurden 805KB Hauptspeicher für die Analyse verbraucht. Die maximale Länge eines bearbeiteten Stromes betrug 82261 Pakete. In den analysierten Strömen wurden 3 Stepping Stones entdeckt, die alle plausibel wirken.

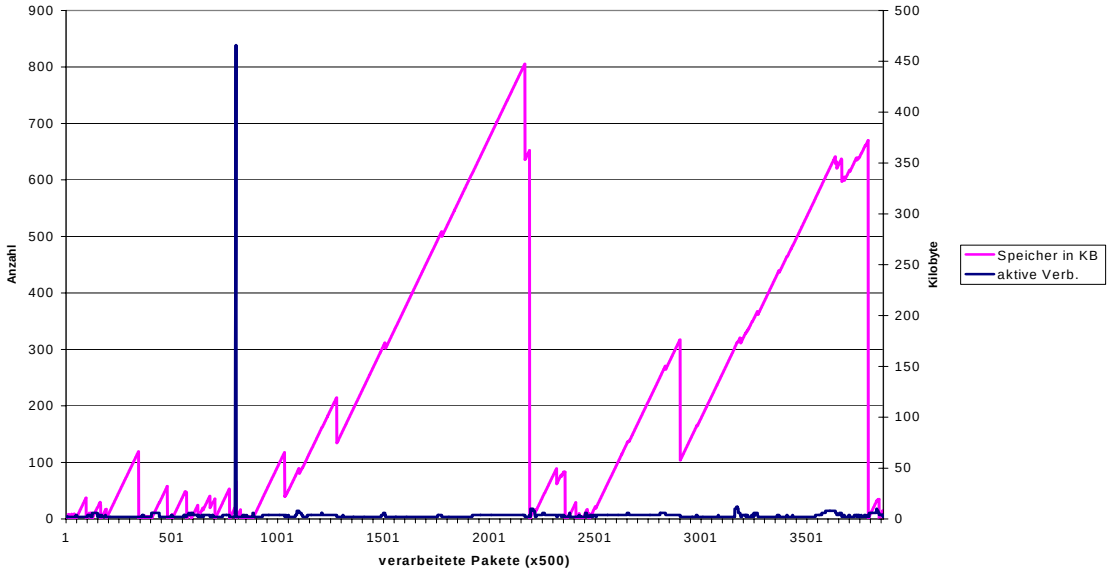

Abbildung 17: Speicherverlauf des Wavelet Algorithmus

Wiederum waren die Verbindungen verschlüsselt gewesen. In der Kombination von beiden Grafiken kann man sehen, wie die Zeit, die für die Analyse benötigt wird, nahezu exponentiell ansteigt, wenn die Verbindungen länger werden. Der Speicherverbrauch zeigt wieder die typischen Merkmale langer Verbindungen mit vielen Paketen. Daher kommen die deutlich sichtbaren Sägezähne in der Grafik.

4.4.5 Paket-Zähler Algorithmus

Die Tests mit dem Paket-Zähler Algorithmus haben eine wesentlich stärkere Abhängigkeit der Ergebnisse von den gewählten Parametern gezeigt, als es bei den anderen Algorithmen der Fall war. Insbesondere die Wahl des Parameters p_Δ (in der Implementierung ergeben sich daraus die Parameter *stepping_counter_pmax* und *stepping_counter_dmax*) entscheidet darüber, ob überhaupt Verbindungen gefunden werden. Im Testparcours war mit den Werten *stepping_counter_dmax* = 10 und *stepping_counter_pmax* = 250 alle Stepping Stones gefunden worden, wohingegen in den kleinen Traces damit nicht ein einziges Verbindungspaar als Stepping Stone markiert worden ist.

Die zweite Erkenntnis zum Paket-Zähler Algorithmus ist nicht minder schlimm für den tatsächlichen Einsatz. Der Algorithmus ist extrem anfällig, durch eine Vielzahl geöffneter Verbindungen zum Absturz wegen Speicherüberlastung gebracht zu werden. Prinzipiell gab es zwei Möglichkeiten der Implementierung, die während der Evaluierungsphase der Algorithmen beide getestet worden sind. Die erste Möglichkeit war, je Verbindung einen Paketzähler zu führen und dann bei der Paketankunft den gemeinsamen Zustand zu prüfen und die möglichen Vorkommen in beiden Verbindungen zu speichern, bis eine Entscheidung getroffen werden kann. Dieser Ansatz birgt aber das große Problem, daß dadurch Stepping Stones nicht erkannt

werden, wenn in einer der beiden Verbindungen schon eine gewisse Zeit gearbeitet worden ist. Da beide Verbindungen ihre Paketzähler unabhängig voneinander führen, hat die länger existierende Verbindung bereits weit mehr Pakete erhalten und gezählt und die Kombination beider Verbindungen wird als „kein Stepping Stone“ markiert. Diese Markierung muß dann in beiden Verbindungen gespeichert werden, damit anschließend kein Vergleich mehr stattfinden muß. Die zweite Möglichkeit, die in der abschließenden Implementierung enthalten ist, umgeht dieses Problem, indem sie die Paketzähler der Verbindungen miteinander koppelt. Es wird also bei jeder neu hinzukommenden Verbindung ein neues Paketzählerobjekt erzeugt, das die Zahl der Pakete ab diesem Zeitpunkt erfaßt. Allerdings sind es gerade diese, zwar kleinen, aber sehr vielen Objekte, die den Hauptspeicher der IDS-Computer zum überlaufen bringen. Um dies zu verdeutlichen wird eine Beispielrechnung zeigen, ab wie vielen gleichzeitigen, unentschiedenen Verbindungen es zum Kollaps kommt. Ausgehend von einer normalen 32-Bit-Architektur ist es einer Anwendung möglich bis zu 4GB Hauptspeicher zu belegen. Eine Erweiterung auf die 64-Bit-Architektur würde hier auch nur die Symptome bekämpfen, nicht aber das eigentlich Problem. Die minimale Menge an Daten die für ein solches Objekt belegt werden muß, sind 10 Byte. Die 10 Byte sind jeweils ein Long-Wert a 4 Byte für jede der beiden beteiligten Verbindungen und zwei Short-Werte à 1 Byte für Zustand des Objekts und den Zähler des Auftretens von verdächtigen Abständen. In der vorliegenden Implementierung sind es jeweils 20 Byte, da noch zusätzlich statistische Daten gespeichert werden. Ausgehend von der minimalen Menge von 10 Byte sind es dann also ca. 429 Millionen Objekte, die maximal im Speicher existieren können. Wenn jetzt noch in Betracht gezogen wird, daß jede Verbindung aus zwei Strömen besteht, die unabhängig voneinander korreliert werden sollen und jeden dieser Ströme mit jedem anderen ein Objekt erhalten soll, so kommt man auf 14655 Verbindungen, die gleichzeitig aktiv sein dürfen. Diese Menge an Verbindungen wird im breitbandigen Umfeld jedoch möglicherweise erreicht werden. Ein potentieller Angreifer kann sich aber dieses Wissen leicht zunutze machen, in dem er einfach sehr viele Verbindungen öffnet und so die IDS-Software zum Absturz bringt.

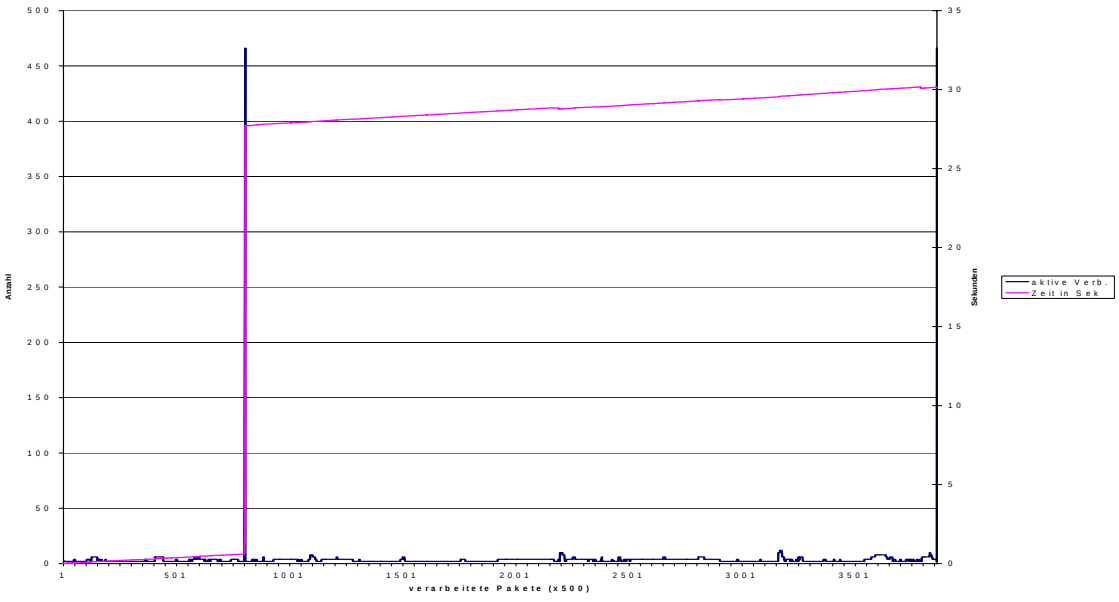

Abbildung 18: Zeitverlauf des Paket-Zähler Algorithmus

Ebenso wie die anderen Algortihmen mußte auch der Paket-Zähler Algorithmus die vier kleinen Traces bearbeiten. Mit dem ersten der Testtraces *halle-cs* war der Algorithmus bereits nach 72,1 Sekunden fertig. Dabei hatte er sein Speichermaximum bei 15853KB erreicht. Die Länge der Verbindungen spielt für den Paket-Zähler Algorithmus keine Rolle, daher soll hier lieber die maximale Zahl der gleichzeitigen Verbindungen genannt werden. Sie betrug 252. Darus ergeben sich 504 Ströme und somit 254016 Verarbeitungs-Objekte. Von dem Algorithmus wurde ein Stepping Stone erkannt, der auch plausibel erscheint. Auch der zweite Trace *halle-login* wurde in einer guten Zeit von 6 Minuten und 52,2 Sekunden bearbeitet. Dabei wurde maximal 12921KB Speicher belegt. Die maximale Anzahl gleichzeitiger Verbindungen lag bei 227, also 206116 Verarbeitungs-Objekte. In diesem Trace wurden keine Stepping Stones gefunden. Durch Modifikationen der Parameter konnten hier keine besseren Ergebnisse erzielt werden, da die Rate der Erkennung (und damit aber auch der False-Positives) exponentiell angestiegen ist und so die Qualität des Ergebnisses nicht verbessert hat. Der dritte Trace *net-cs* konnte vom Algorithmus in 2 Minuten und 43,2 Sekunden verabeitet und analysiert werden. Dabei benötigte er 14599KB Hauptspeicher bei einer Spitze von 242 gleichzeitig aktiven Verbindungen. Wiederum wurde kein Stepping Stone erkannt, also auch nicht der Stepping Stone der eigentlich vorhanden hätte sein müssen. Durch Modifikationen konnte hier zumindest der Stepping Stone erkannt werden, allerdings würde er im realen Einsatz in der Flut der False-Positives untergehen.

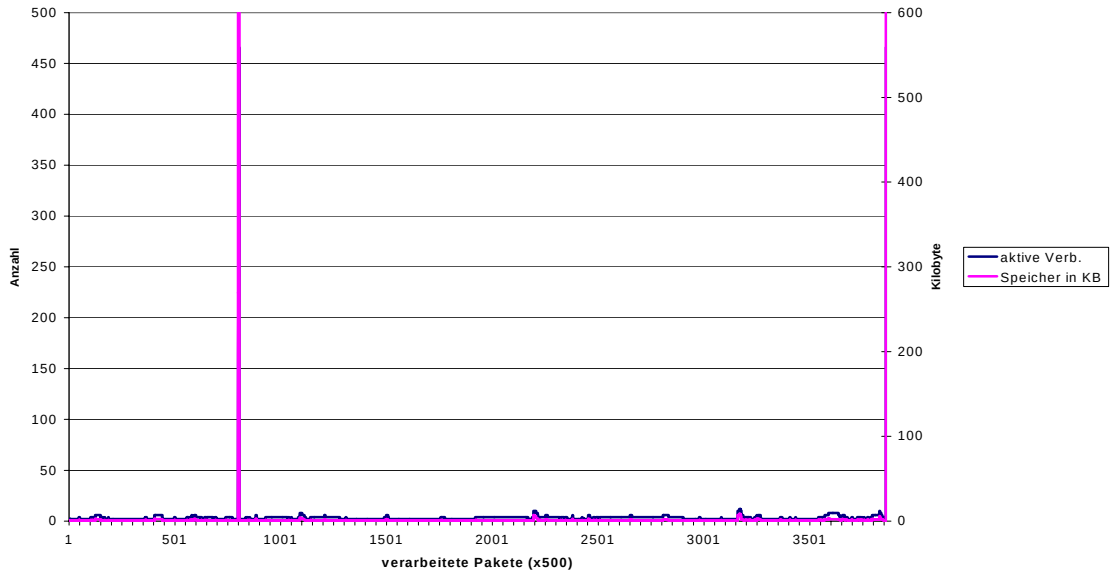

Abbildung 19: Speicherverlauf des Paket-Zähler Algorithmus

Beim abschließenden Vergleich mit dem Trace *net-login* war der Paket-Zähler Algorithmus nach 1 Minute und 26,6 Sekunden fertig. Wieder lag der maximale Speicherverbrauch in ähnlicher Höhe, bei 13551KB. Daraus ersichtlich ist auch, daß die Maximalzahl der gleichzeitig aktiven Verbindungen auf einem vergleichebaren Niveau, bei 233 Verbindungen. Die Grafiken zeigen, daß beim Paketzähler sowohl der Speicherverbrauch, als auch der Zeitverlauf eng an die Anzahl der gleichzeitigen Verbindungen gebunden ist. Gerade beim Verlauf der Speichernutzung wird deutlich, daß die Spitzen der gleichzeitigen Verbindungen auch zu noch wesentlich höheren Spitzen im Speicherverbrauch führen. Die Grafik schneidet hier auch die beiden

extremen Spitzen ab, da bei einem anderen Maßstab die Ähnlichkeiten im Bereich der kleinen Werte nicht mehr zum Ausdruck kämen.

4.5 Zusammenfassung

Die Ergebnisse die in der Auswertung der Algorithmen gefunden wurden, sollen im folgenden noch einmal kurz zusammengefaßt werden. Allgemein war festzustellen, daß die Algorithmen im breitbandigen Umfeld mit der Menge der Daten überfordert sind. Selbst wenn der Verkehr, der untersucht wird, auf die interaktive Dienste eingeschränkt wird, kommt es oft dazu, daß trotzdem entweder die Laufzeit jenseits nutzbarer Werte liegt oder der Speicher volläuft. Gerade diese Einschränkung ermöglicht es aber einem Angreifer, der einen Stepping Stone unerkannt nutzen möchte, der Erkennung zu entkommen, indem er seinen Server auf einen Port hören läßt, der nicht den Standardports für interaktive Dienste entspricht. Auf kleineren Datenmengen funktionieren die Algorithmen recht gut, sind jedoch zum Teil sehr empfindlich, was die Wahl der Parameter anbetrifft. Dabei macht es dann auch kaum einen Unterschied, ob die Algorithmen den Verkehr überwachen, während er über das Netzwerk fließt oder ob eine nachträgliche Analyse eines Traces durchgeführt wird. Im Überblick der Algorithmen scheint es, daß der On/Off Algorithmus immer noch eine gute Wahl ist, wenn er auch möglicherweise nicht alle Stepping Stones findet und gegenüber der Umgehung empfindlicher ist als die anderen Algorithmen. Der IPD-Algorithmus hat wegen seiner großen Probleme mit der Laufzeit die Erwartungen nicht erfüllen können. Um ihn überhaupt nutzen zu können, mußte der größte Vorteil des Algorithmus, sein Operieren auf unvollständigen Daten, geopfert werden. Leider hat auch der Wavelet Algorithmus Probleme mit der Laufzeit bei langen Verbindungen. Davon abgesehen bietet er eine gute Robustheit bezüglich der Umgehung. Vom realen Einsatz ist der Paket-Zähler Algorithmus leider noch ein gutes Stück entfernt, weil er zu empfindlich auf die Parameter reagiert. Bei den Algorithmen läßt sich feststellen, daß sie alle gute Ideen enthalten, aber die beschriebenen Methoden eher theoretischer Natur sind. Für die Implementierung und den Einsatz ergeben sich daraus zum Teil Probleme, die in den Arbeiten nicht angesprochen wurden. Außerdem wird dabei nicht umfassend auf den Einsatz mit großen Datenmengen eingegangen. Gerade dieser Mangel trat nun auch deutlich in den Ergebnissen und der Implementierung zu Tage.

Eine weitere Erkenntnis, die bei den Analysen nebenbei gewonnen werden konnte, ist, daß die Verwendung von unverschlüsselten interaktiven Diensten in der Zwischenzeit nahezu keine Rolle mehr spielt. Fast jeder setzt auf die verschlüsselten und komprimierbaren Dienste einer SSH-Verbindung. Darüber hinaus sind alle gefundenen Stepping Stones tatsächlich verschlüsselt gewesen und es ist zu erwarten, daß diese Tatsache bei den illegalen Stepping Stones noch mit einer wesentlich höheren Gewißheit zu finden ist.

5 Fazit und Ausblick

Im Verlauf dieser Diplomarbeit wurden viele Fakten rund um die Erkennung von Stepping Stones in breitbandigem Netzwerkvekehr dargestellt. Stepping Stones sind Computer, die in einem Netzwerk erreichbar sind und dazu benutzt werden, um von dort aus Dienste eines anderen Computers in Anspruch zu nehmen. Mit dieser Arbeit wurde das Ziel verfolgt die Algorithmen zur Erkennung von Stepping Stones miteinander zu vergleichen und sie bezüglich ihrer Einsetzbarkeit auf breitbandigen Netzwerkverbindungen zu überprüfen. Es wurden für den Vergleich breitbandige Verbindungen gewählt, da in der Literatur bisher keine Vergleiche in diesem Umfeld durchgeführt wurden. Insbesondere in den einzelnen Arbeiten zu den Algorithmen wurden die Tests immer nur auf kleinen Datenmengen ausgeführt.

Die Diplomarbeit wurde mit einer eher allgemeinen Einführung in die Intrusion Detection begonnen, um im Verlauf dieser Einführung zu der speziellen Art des Stepping Stones zu kommen. Im darauf folgenden Abschnitt wurde ein Literaturüberblick gegeben, der die verschiedenen Algorithmen näher erklärt. Dabei wurde das Augenmerk auf eine Reihe Vergleichskriterien gelenkt, die später zur Auswahl der vier Algorithmen geführt hat, die miteinander verglichen wurden. Die Literatur weist hier nur timingbasierte Algorithmen auf. Die Algorithmen wurden so ausgewählt, daß sie keinen aktiven Eingriff in das Netzwerk erfordern. Darüber hinaus sollten sie sowohl online als auch offline einsetzbar sein. Bei der Implementierung der Algorithmen wurden einige Schwachstellen in den Beschreibungen der Algorithmen gefunden. Hier fehlten zum Teil Vorschläge für die Werte der Parameter oder die Angabe zu welchen Zeitpunkten die Analyse durchgeführt werden soll. Die Algorithmen wurden in das existierende IDS Bro integriert um so eine gute Basis für den Vergleich zu erhalten. Beim anschließenden Vergleich der Algorithmen konnte festgestellt werden, daß sie für den Einsatz im breitbandigen Umfeld nicht geeignet sind. Bei der Überwachung einer Breitbandleitung führt selbst eine Einschränkung auf interaktive Dienste nicht zum Erfolg. Für den Einsatz auf kleineren Subnetzen hingegen sind die Algorithmen geeignet. Dabei ist allerdings, wie auch bei den meisten anderen Erkennungsalgorithmen der Intrusion Detection, eine feine Justierung der Parameter und eine ständige Plausibilitätskontrolle der Alarme unumgänglich. Durch die Systematik der Algorithmen läßt sich in allen Fällen eine Möglichkeit der Umgehung finden. Der Wavelet Algorithmus bietet hierbei die beste Alternative, da er nicht so sehr auf kleine Details der Verbindungen eingeht, sonder vielmehr das Gesamtbild der Verbindung in Betracht zieht. Die anderen Algorithmen haben zum Teil die gesetzten Erwartungen nicht erfüllen können.

Für die Zukunft scheint es wegen der ausgeklügelten Umgehungsstrategien der Angreifer für die Intrusion Detection immer schwieriger zu werden, die Angriffe zu erkennen. Auch die Erkennung von Stepping Stones wird in diesem Bereich deutlich schwieriger werden. Gerade dann, wenn die Erkennung auf einer breitbandigen Leitung stattfinden soll, muß hier das Problem der quadratischen Laufzeit gelöst werden. Eine weitere Beschäftigung mit der angesprochenen Thematik der Vorfilterung wird sicherlich zu besseren Ergebnissen führen. Die Vorfilterung ist aber gleichzeitig ein schwieriges Feld, da dadurch auch Schlupflöcher entstehen können, wenn Verbindungen möglicher Stepping Stones nicht miteinander verglichen werden. Darüber hinaus lassen sich sicherlich auch noch, durch kleinere Veränderungen in der Systematik der Algorithmen, ihre Implementierungen optimieren, aber das Potential ist eher gering einzuschätzen, solange die quadratische Laufzeit nicht verändert werden

kann. Einen guten Ansatz in Bezug auf die Laufzeit bietet sicherlich der Paket-Zähleralgorithmus, der aber leider das Problem auf eine quadratische Speichernutzung transferiert.

Im wesentlichen erscheint es vor allem aber als nahezu unlösbar, die legale von der illegalen Nutzung zu unterscheiden. Nachdem die gefundenen Stepping Stones nicht auf ihren Inhalt hin untersucht werden können, ist es dem IDS, und seinem Administrator, an dieser Stelle unmöglich eine Entscheidung zu fällen. Dies führt dazu, daß es fraglich ist, inwieweit der Einsatz ressourcenverschlingender Algorithmen zur Erkennung von Stepping Stones überhaupt sinnvoll ist.

6 Literaturverzeichnis

[1] Ralf Spenneberg, Intrusion Detection für Linux-Server, Seite 24, Markt+Technik Verlag, München, 2003

[2] Symantec Security Response, W32.Sasser.Worm, Stand 27.07.2004 http://securityresponse.symantec.com/avcenter/venc/data/w32.sasser.worm.html

[3] http://www.subseven.de/s2/trojans.htm

[4] „Superwurm mit öffentlichem Quelltext“, Heise Verlag, 19.04.2004 http://www.heise.de/newsticker/meldung/46634

[5] Ralf Spenneberg, Intrusion Detection für Linux-Server, Seite 44, Markt+Technik Verlag, München, 2003

[6] Yin Zhang, Vern Paxson, „Detecting Stepping Stones“, Proc. 9th USENIX Security Symposium, August 2000

[7] Xinyuan Wang, Douglas S. Reeves, “Robust Correlation of Encrypted Attack Traffic through Stepping Stones by Manipulation of Interpacket Delays”, Proc. 10th ACM Conference on Computer and Communications Security (CCS 2003), October, 2003

[8] Xinyuan Wang, Douglas S. Reeves, “Robust Correlation of Encrypted Attack Traffic through Stepping Stones by Manipulation of Interpacket Delays”, Seite 3, Proc. 10th ACM Conference on Computer and Communications Security (CCS 2003), October, 2003

[9] Kunikazu Yoda, Hirosaki Etoh, “Finding a Connection Chain for Tracing Intruders”, In: F. Guppens, Y. Deswarte, D. Gollmann and M. Waidner, editors, 6th European Symposium on Research in Computer Security -- ESORICS 2000 LNCS-1895, Toulouse, France (October 2000)

[10] Kunikazu Yoda, Hirosaki Etoh, “Finding a Connection Chain for Tracing Intruders”, Seite 8, In: F. Guppens, Y. Deswarte, D. Gollmann and M. Waidner, editors, 6th European Symposium on Research in Computer Security -- ESORICS 2000 LNCS-1895, Toulouse, France (October 2000)

[11] Xinyuan Wang, Douglas S. Reeves, and S. Felix Wu. “Inter-Packet Delay Based Correlation for Tracing Encrypted Connections through Stepping Stones”. In Proceedings of the 7th European Symposium on Research in Computer Security (ESORICS 2002), LNCS 2502, October, 2002.

[12] David L. Donoho, Ana G. Flesia, Umesh Shankar, Vern Paxson, Jason Coit, and Stuart Staniford, “Multiscale Stepping-Stone Detection: Detecting Pairs of Jittered Interactive Streams by Exploiting Maximum Tolerable Delay”, Proc. RAID 2002

[13] Michael Clemens, „Wavelet Tutorial“ , Stand 21.10.2004, http://nt.eit.uni-kl.de/wavelet/transformation.html

[14] In Anlehnung an Michael Clemens, „Wavelet Tutorial“, Stand 21.10.2004, http://nt.eit.uni-kl.de/wavelet/index.html

[15] Haar-Wavelet, Stand 11.11.2004, http://www.uni-protokolle.de/Lexikon/Haar-Wavelet.html

[16] Michael Clemens, „Wavelet Tutorial“, Stand 21.10.2004, http://nt.eit.uni-kl.de/wavelet/filterbank.html

[17] Michael Clemens, „Wavelet Tutorial“, Stand 21.10.2004, http://nt.eit.uni-kl.de/wavelet/dwt_2d.html

[18] Avrim Blum,Dawn Song, Shohba Venkataraman, “Detection of Interactive Stepping Stones wirth Maximum Delay Bound: Algorithms and Confidence Bounds”, Proc. RAID 2004, Seventh International Symposium on Recent Advances in Intrusion Detection, September 15-17, 2004

[19] Kwong H. Yung, “Detecting Long Connection Chains of Interactive Terminal Sessions”, Proc. RAID 2002

[20] Kwong H. Yung, “Detecting Long Connection Chains of Interactive Terminal Sessions”, Seite 7, Proc. RAID 2002

[21] „Überblick über das Münchner Wissenschaftsnetz (MWN)“, Stand 16.08.2004 http://www.lrz-muenchen.de/services/netz/mhn-ueberblick/

[22] „Gigabit-Wissenschaftsnetz (G-WiN)“, Stand: 01.10.2004 http://www.dfn.de/index.jsp?path=/gigabitwissenschaftsnetz/

[23] „Bro Intrusion Detection System“, Stand 11.11.2004 http://bro-ids.org/

[24] Vern Paxson, Bro: A System for Detecting Network Intruders in Real-Time, Computer Networks, 31(23-24), pp. 2435-2463, 14.12.1999

[25] “Wavelet Bibliothek” von Martin Andreas Dietze http://herbert.the-little-red-haired-girl.org/en/software/wavelet/