



Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Johannes Martens

Entwicklung eines Netzwerkmonitors als "Bridge-
Lösung" auf Basis von Kleinstrechnern

Johannes Martens

Entwicklung eines Netzwerkmonitors als "Bridge- Lösung" auf Basis von Kleinstrechnern

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Klaus-Peter Kossakowski
Zweitgutachter : Prof. Dr.-Ing. Martin Hübner

Abgegeben am 03.03.2016

Johannes Martens

Thema der Arbeit

Entwicklung eines Netzwerkmonitors als "Bridge-Lösung" auf Basis von Kleinstrechnern

Stichworte

Netzwerk, Sicherheit, Sniffer, Kleinstrechner, Brücke

Kurzzusammenfassung

Diese Arbeit beschreibt die Entwicklung eines Netzwerkmonitors. Es wird die Architektur und verwendete Technologie vorgestellt. Anschließend wird mithilfe der Technologien die Architektur umgesetzt und deren Fähigkeiten anhand dreier Use Cases aufgezeigt.

Johannes Martens

Title of the paper

Development of a network monitor as a "bridge solution" based on a single board computer

Keywords

network, security, single board computer, intrusion detection, bridge,

Abstract

This paper describes the development of a network monitor. It presents the architecture and technologies that were used. The architecture is implemented and its capabilities are tested with the help of three use cases.

Inhaltsverzeichnis

1	Einleitung	6
1.1	Motivation	6
1.2	Ziel / Abgrenzung	8
1.3	Use Cases	9
2	Überblick	10
2.1	Technologische Grundlagen	10
2.1.1	Intrusion Detection	10
2.1.2	Kleinstrechner (Einplatinencomputer)	11
2.1.3	Netzwerkbrücke	12
2.2	Rechtliche Grundlagen	12
3	Architektur und verwendete Technologien	13
3.1	Architektur	13
3.2	Phytec Board	19
3.3	Software	20
3.3.1	Yocto Project	20
3.3.2	LibPcap/JnetPcap	20
3.3.3	ZeroMQ Messagequeue	21
3.3.4	Apache Webserver	21
4	Entwicklung und Einrichtung	23
4.1	Plattform	25
4.1.1	Linux	25
4.1.2	Bridge	26
4.1.3	LEDs	28
4.2	Anwendung	29

Einleitung	5
4.2.1 Sniffer	29
4.2.2 Paketverteiler	31
4.2.3 SYN-Flood Detektor.....	33
4.2.4 Teamviewer Detektor	36
4.2.5 Warner	37
4.2.6 E-Mailer	38
4.2.7 Wireshark Anbindung	39
4.2.8 Statusseite.....	41
5 Einsatz und Evaluation des Konzeptes.....	42
5.1 UseCase 1 : SYN-Flood Detektion.....	42
5.2 UseCase 2 : Unerlaubter Teamviewer-Zugriff.....	42
5.3 UseCase 3 : Kombiniertes Angriff.....	43
5.4 Weitere Funktionen	43
5.4.1 Wireshark.....	43
5.4.2 Statusseite.....	44
5.5 Performance der Netzwerkbrücke.....	44
5.6 Funktionalität	45
6 Fazit	49
6.1 Bewertung des Konzeptes.....	49
6.2 Ausblick / Mögliche Weiterentwicklung	49
7 Anhang.....	51
7.1 Literaturverzeichnis.....	51
7.2 Abbildungsverzeichnis.....	54
7.3 Technische Daten des Phyttec phyBOARD®-Wega	55

1 Einleitung

In der Einleitung wird zuerst dargestellt welche Gründe die Entwicklung des Netzwerkmonitors auf Basis eines Kleinstrechners motivieren. Zweitens wird erklärt welche Funktionen der Netzwerkmonitor erfüllen soll, und welche nicht. Im letzten Teil der Einleitung wird anhand dreier Use Cases der Nutzen des Netzwerkmonitors aufgezeigt.

1.1 Motivation

Fast täglich finden Social Engineering Angriffe statt, die gezielt die „Schwachstelle Mensch“ ausnutzen um Zugang zu geschützten Daten zu erhalten. Ein Beispiel hierfür sind E-Mails, die den Nutzer auffordern einen Anhang zu öffnen. Diese Angriffsart wird oft im Rahmen eines gezielten Angriffs (Advanced Persistent Threat (APT)) genutzt. Der Ablauf eines APT-Angriffes wird in Abbildung 1 verdeutlicht. Besonders APTs können Unternehmen großen finanziellen Schaden zufügen. Zum einen durch zusätzliche Kosten, die durch Entfernung der Angriffswerkzeuge und notwendige Säuberungsarbeiten anfallen, zum anderen geschäftliche Einbußen und weitere Kosten, die durch gestohlene Daten entstehen können.

Erleichternd für die Angreifer kommt noch dazu, dass viele Unternehmen nicht einmal grundlegende IT-Sicherheitsmaßnahmen umgesetzt haben. [vgl. Bund15a S.24-26,S.31-32] Aber auch Privatpersonen werden immer öfter Ziel von Angriffen und besitzen zunehmend mehr angreifbare Geräte, wie Smartphones und –watches, oder Geräte aus dem Internet der Dinge (Internet of things).

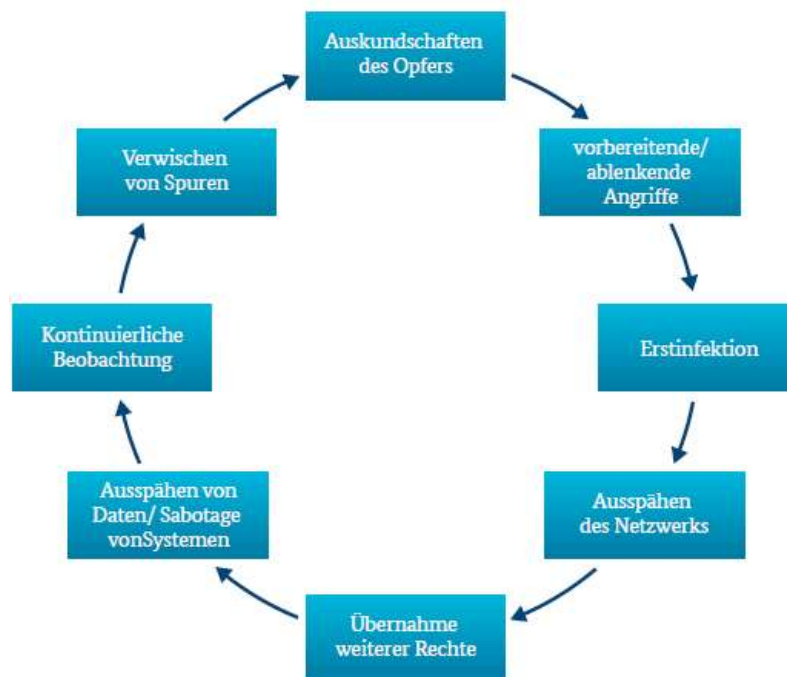


Abbildung 1 : Vorgehensweise bei einem APT-Angriff [Bund15a S.26 Abbildung 6]

Intrusion Detection Systeme helfen die Auswirkungen von Angriffen zu verringern oder ganz zu verhindern, indem sie frühzeitig auf einen Angriff hinweisen. Unternehmen und Privatpersonen können sich aber oftmals keine modernen Intrusion Detection Systeme leisten, da diese viele Tausend Euro kosten können, wie zum Beispiel der „Cisco Firepower 7010“ eine Intrusion Detection Lösung von Cisco. [vgl. Cisc16] Der hohe Preis liegt zum Teil daran, dass die meisten Intrusion Detection Systeme Teil eines Intrusion Detection and Prevention Systems sind.

Am anderen Ende des Spektrums der Intrusion Detection Systeme befinden sich Anti-Viren-Programme wie sie auf den allermeisten PCs installiert sind. Diese schützen aber nur einen einzelnen Computer. Viele Personen besitzen aber mehrere elektronische Geräte, wie Tablets, Smartphones, Spielekonsolen, und andere internetfähige Geräte. Selbst Fernseher und Musik-Anlagen verfügen oftmals über einen Internetzugang. Nicht all diese Geräte lassen sich aber durch Anti-Viren-Software schützen, der Nutzer hat z.B. gar keine Möglichkeit eine solche Software auf seiner Spielekonsole zu installieren. Selbst wenn ein Nutzer auf den Geräten auf denen die Möglichkeit besteht, Intrusion Detection Software installiert, fällt es schwer den Überblick über die verschiedensten verwendeten Programme zu behalten. Dazu kommt noch, dass viele Programme die Nutzer mit Informationen geradezu überfluten, sodass es schwerfällt die relevanten Informationen aus der Masse herauszupicken.

Oftmals würde es aber reichen, gerade bei kleineren Unternehmen und Privathaushalten, wenn überhaupt vor einem Angriff gewarnt wird, ähnlich wie heutzutage in den meisten Räumen Rauchmelder installiert sind, aber keine Sprinkler- oder andere Feuerlöschanlagen. In vielen Bundesländern ist der Einsatz von Rauchmeldern sogar gesetzlich vorgeschrieben. Vielen Menschen sind aber ihre digitalen Besitztümer genauso wichtig wie reale Gegenstände. Genau die Aufgabe eines digitalen Rauchmelders soll der Netzwerkmonitor erfüllen. So wie ein Rauchmelder vor einem Brand warnt, warnt der Netzwerkmonitor auf Basis eines Kleinstrechners vor Angriffen.

1.2 Ziel / Abgrenzung

Ziel dieser Arbeit ist die Entwicklung eines Netzwerkmonitors, der als Intrusion Detection System in kleinen Unternehmen und Privathaushalten zum Einsatz kommen soll. Dabei soll er als Netzwerkbrücke leicht in eine bestehende Netzwerk-Infrastruktur integrierbar sein. Eine mögliche Platzierung des Monitors in einem bestehenden Netzwerk ist in Abbildung 2 zu erkennen.

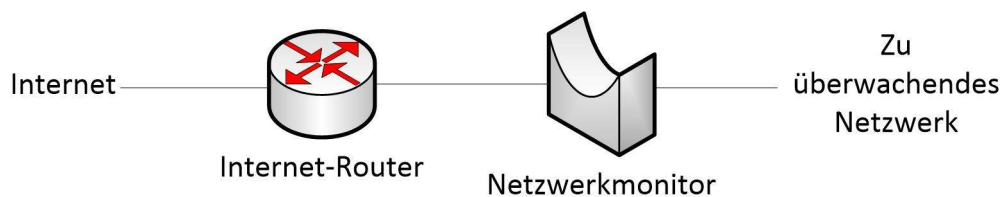


Abbildung 2 : Beispielhafte Platzierung des Netzwerkmonitors in einem Netzwerk

Als Hardware-Basis soll ein Kleinstrechner genutzt werden.

Der Monitor soll leicht um weitere Komponenten erweiterbar sein, z.B. die Erkennung weiterer Angriffe.

Er soll in der Lage sein verschiedene, auch mehrstufige, wie z.B. APTs, oder wiederholte Angriffe, während sie stattfinden zu entdecken, und Nutzer über Auftreten eines Angriffes informieren.

Er kennt dabei zwei unterschiedliche Warnungstypen, einen Alarm, der vor besonders schweren Angriffen warnt, und eine normale Warnung, die auf alle anderen Vorfälle hinweist.

Der Netzwerkmonitor soll Angriffe nur entdecken, nicht bekämpfen oder verhindern, wie z.B. eine Firewall oder Intrusion Prevention Systeme.

1.3 Use Cases

Anhand dreier Use Cases soll gezeigt werden, dass der Netzwerkmonitor die genannten Ziele erfüllt.

Dabei handeln die ersten beiden Use Cases von jeweils einem einzelnen Angriff, während der dritte Use Case einen zusammengesetzten Angriff aus den Angriffen der ersten beiden Use Cases behandelt.

Der erste Use Case behandelt das Erkennen eines Denial of Service (DoS) Angriffes auf ein Ziel, das sich im vom Netzwerkmonitor überwachten Netz befindet. Hierbei ist es nicht relevant, ob der Angriff von nur einer Quelle ausgeht, oder mehrere Verursacher beteiligt sind, z.B. durch Verwendung eines Botnetzes. Solche DoS-Angriffe können vor allem Privatpersonen beeinträchtigen, verhindern sie doch die Nutzung des Internets und mit Umstellung des Telefonnetzes der Telekom auf VOIP auch des Telefons, während sie stattfinden. Weiterhin nutzen viele Personen Services wie Netflix, und nicht mehr das klassische TV, um Filme zu schauen. Allein im ersten Halbjahr 2015 fanden über 29000 Denial of Service (DoS) Angriffe in Deutschland statt, stellen also eine ernst zu nehmende Bedrohung dar. [vgl. Bund15a S.30-31] Wenn nun ein solcher Angriff erkannt wird, meldet dies der Netzwerkmonitor an den Nutzer.

Im zweiten Use Case wird betrachtet, wie die Teamviewer Fernwartungs-Software genutzt werden kann, um einen Angriff vorzubereiten oder durchzuführen.

Gerade Privatpersonen oder kleinere Unternehmen wie Arztpraxen nutzen Teamviewer, um sich bei Problemen von einem externen IT-Dienstleister, oder im Falle von Privatpersonen auch durch Bekannte, Kinder, Enkel, helfen zu lassen. Sie bekommen es aber nicht mit, wenn mithilfe von Teamviewer ein Angreifer auf ihr System zugreift. Bei unerlaubter Nutzung von Teamviewer meldet der Netzwerkmonitor dies in Form einer Warnung an den Nutzer.

Der dritte Use Case handelt nun von einem kombinierten Angriff.

Der Angreifer nutzt seinen Zugang zu Teamviewer, um das anvisierte System auf weitere Angriffe vorzubereiten. Zugangsdaten zu Teamviewer könnte er z.B. über Social Engineering von einem Mitarbeiter erhalten haben, indem er sich in einer Email als Mitarbeiter des externen Unternehmens, das mit der Wartung der Systeme beauftragt ist, ausgibt.

Das Passwort, das zu den Teamviewer-Zugangsdaten gehört, ist standardmäßig nur vier Stellen lang, und somit relativ leicht mithilfe von Brute-Force Methoden herauszufinden.

Nun kann der Teamviewer-Zugriff genutzt werden, um weitere Angriffe vorzubereiten oder durchzuführen. Der Angreifer kann Ports öffnen, um nun eine SYN-Flood Denial-of-Service-Angriff zu ermöglichen. Dieser mehrstufige Angriff wird vom Netzwerkmonitor erkannt, und dem Nutzer in Form eines Alarms mitgeteilt.

2 Überblick

Dieses Kapitel stellt zuerst für die Entwicklung des Netzwerkmonitors relevante technologische Grundlagen vor. Darauf folgt ein Einblick in wichtige Forschungen, Projekte und Produkte im Bereich der Network Intrusion Detection and Prevention, z.B. snort, Wireshark und andere Endknoten-Lösungen, Hardware-Firewalls, Lösungen für Großunternehmen/-Netze.

Der letzte Teil des Kapitels handelt von aktueller deutscher Rechtsprechung und welche Probleme die Verwendung eines Netzwerkmonitors aus rechtlicher Sicht haben kann.

2.1 Technologische Grundlagen

In diesem Kapitel werden verschiedene technologische Grundlagen, die für die Entwicklung eines Netzwerkmonitors auf Basis eines Kleinstrechners relevant sind, vorgestellt.

2.1.1 Intrusion Detection

„Intrusion detection is the process of monitoring the events occurring in a computer system or network and analyzing them for signs of possible incidents[..]“ [ScMe07 S.15]

So definiert das National Institute of Standards and Technology, das U.S. Amerikanische Gegenstück zum deutschen Institut für Normung (DIN) die Intrusion Detection, die Hauptaufgabe des Netzwerkmonitors. Mögliche Vorfälle (incidents) sind z.B. Angreifer die sich unautorisiert Zugriff auf Systeme verschaffen oder Nutzer, die bestehende Rechte ausnutzen um sich Zugriff zu ihnen nicht erlaubten Bereichen zu verschaffen.

Es existieren drei verschiedene Arten der Intrusion Detection, signaturbasiert, Anomalie-basiert und die Analyse von stateful Protocols.

Die signaturbasierte Intrusion Detection vergleicht beobachtete Ereignisse mit abgespeicherten Signaturen bekannter Angriffe oder anderer negativer Vorkommnisse. Dies entspricht dem zweiten Use Case, in dem die Signatur der Zugriff auf Teamviewer außerhalb der erlaubten Zeiten ist. Ein anderes Beispiel für eine Signatur ist eine E-Mail mit einem bestimmten Betreff oder Anhang, von der bekannt ist, dass sie Malware enthält.

Anomalie-basierte Intrusion Detection vergleicht Profile normalen Verhaltens mit dem tatsächlichen Verhalten eines Systems. Tritt eine Abweichung zwischen Profil und tatsächlichem Verhalten auf ist dies eine Anomalie und weist auf einen Angriff hin. Ein Beispiel für eine Anomalie, der durch ein solches Intrusion Detection System entdeckt werden kann ist ein DOS-(Denial of Service) Angriff, der sich durch eine deutlich höhere Anzahl aufgebauter Verbindungen zu einem Server, als dies normalerweise zu erwarten ist, bemerkbar macht. Diese Angriffsart wird im ersten UseCase behandelt.

Die Analyse von stateful Protocols beobachtet das Verhalten verschiedenster Netzwerkprotokolle, wie z.B. FTP, TCP, etc., und warnt, wenn das beobachtete Verhalten vom im Protokoll festgelegten abweicht. Beispielhaft hierfür ist das FTP-Protokoll, ein Nutzer sollte zu Beginn einer FTP-Session nur einige wenige Befehle ausführen, wie Aufrufen der Hilfe oder Angabe von Nutzernamen und Kennwort um sich zu authentifizieren. Wenn sich eine Session nun aber über längeren Zeitraum in diesem Anfangsstadium befindet, verstößt dies gegen die Definition des FTP-Protokolls und wird vom Intrusion Detection System gemeldet. [vgl. ScMe07 S.15-18] Ein anderes Beispiel hierfür ist der erste Use Case, in dem gegen das definierte Verhalten des TCP-Protokolls verstoßen wird.

2.1.2 Kleinstrechner (Einplatinencomputer)

"Hinter dem Begriff Single-Board-Computer (SBC) verbirgt sich ein Prozessorboard, das alle zum Betrieb des Systems erforderlichen Komponenten wie den Mikroprozessor, die Arbeitsspeicher, Caches, die Chipsätze usw. enthält. Weitergehende periphere Funktionen für die Aus- und Eingabe, die Grafikdarstellung oder die Kommunikationssteuerung können über weitere Aufsteckmodule hinzugefügt werden." [Lipi16]

Oftmals verfügen moderne Einplatinencomputer, anders als in der Definition dargestellt, bereits über einige Ein- und Ausgabe-Schnittstellen, wie z.B. USB oder Ethernet. [vgl. Isik15 S.55 - 64]

2.1.3 Netzwerkbrücke

Netzwerkbrücken verbinden mehrere LANs auf der Sicherungsschicht des OSI-Modells. Sie verfügen über mehrere Netzwerkanschlüsse und leiten eingehende Pakete anhand der im Rahmen-Header enthaltenen Adresse in Richtung ihrer Ziele, die die Bridge aus einer Tabelle entnimmt, weiter. Die einzelnen Anschlüsse einer Brücke können unterschiedliche Geschwindigkeiten unterstützen, z.B. könnte ein Anschluss über eine 100 Mbit/s Leitung verfügen, ein anderer über eine 1000 Mbit/s.

Für ihren ursprünglichen Verwendungszweck, zur Verbindung unterschiedlicher LAN-Arten, z.B. Ethernet und Token-Ring-LAN, werden Bridges kaum noch eingesetzt. Vielmehr sind Netzwerkschwitches nichts anderes als Brücken. [vgl. TaWe12 S.395-396] Diese Eigenschaft wird von vielen Intrusion-Detection-Systemen ausgenutzt. Wenn ein solches System als Brücke oder Switch in einem Netzwerk integriert ist, muss der Netzwerkverkehr das IDS passieren um weiter in das Netzwerk vorzudringen. So können Angriffe das IDS nicht umgehen. Aufgrund dieser Eigenschaft soll auch der Netzwerkmonitor als Netzwerkbrücke fungieren.

2.2 Rechtliche Grundlagen

Nach geltendem deutschem Recht ist es nicht erlaubt „sich oder einem anderen unter Anwendung von technischen Mitteln nicht für ihn bestimmte Daten (§ 202a Abs. 2) aus einer nichtöffentlichen Datenübermittlung oder aus der elektromagnetischen Abstrahlung einer Datenverarbeitungsanlage“ zu verschaffen. [StGB §202b Abfangen von Daten]

Der Netzwerkmonitor ermöglicht Zugriff auf nichtöffentliche Datenübermittlungen, wie sie z.B. im heimischen Netzwerk einer Privatperson, oder dem internen Netzwerk eines Unternehmens vorliegen.

Verwendung des Monitors sollte kein Problem darstellen, solange Nutzer des durch den Netzwerkmonitors überwachten Netzwerkes darüber informiert wurden, dass eine Überwachung des Netzwerkverkehrs stattfindet, ähnlich wie auch auf die Überwachung durch Videokameras hingewiesen wird.

Eine weitere Möglichkeit ist, dass Nutzer vor Verbindung in das Netzwerk eine Einwilligung unterschreiben müssen, in der sie der Überwachung zustimmen.

Über weitere rechtliche Probleme, wie sie sich unter anderem aus StGB §202c Vorbereiten des Ausspähsens und Abfangens von Daten ergeben, soll hier nicht eingegangen werden.

3 Architektur und verwendete Technologien

In diesem Kapitel wird zuerst die Architektur des Netzwerkmonitors vorgestellt. Darauf folgt, wie die Anforderungen der Architektur, die Auswahl der verwendeten Technologien beeinflusst haben.

Als erste Technologie wird die Hardwarekomponente des Netzwerkmonitors vorgestellt, der Kleinstrechner Phytec phyBOARD-Wega AM335x 5V. Weiterhin werden die verschiedensten Software-Technologien vorgestellt, das Betriebssystem des phyBOARD-Wegas, die verwendete Sniffing-Bibliothek, das Message-Queue-System, sowie der Apache Webserver.

3.1 Architektur

Die Anforderungen an die Architektur des Netzwerkmonitors ergeben sich aus den in Kapitel 1.2 genannten Zielen.

Die Hardware, die die Basis des Monitors bildet, ist ein Kleinstrechner, der über mehrere steuerbare LEDs verfügt, um den Nutzer über Warnungen und Alarmer zu informieren.

Durch geringen Stromverbrauch und Größe des Kleinstrechners ist der Netzwerkmonitor einfach in bestehende Infrastruktur integrierbar. Um als Netzwerkbrücke dienen zu können, muss der Kleinstrechner mindestens zwei Netzwerkanschlüsse haben. Diese Netzwerkbrücke soll mithilfe des Betriebssystems konfiguriert werden, um eine möglichst hohe Übertragungsgeschwindigkeit zu erreichen. Also muss das Betriebssystem des Kleinstrechners dies ermöglichen.

Die Software-Architektur des Netzwerkmonitors lässt sich in drei Bereiche einteilen, die jeweils über mehrere Komponenten verfügen. Der erste Bereich beinhaltet das Übertragen von Netzwerkpaketen mithilfe der Brücke und die Bereitstellung dieser Pakete an andere Komponenten. Deshalb besteht der erste Bereich aus drei Komponenten, der Netzwerkbrücke, einem Sniffer, der die Pakete von der Netzwerkbrücke ausliest, und einer Übertragungskomponente, die die Pakete an Komponenten im zweiten Bereich überträgt.

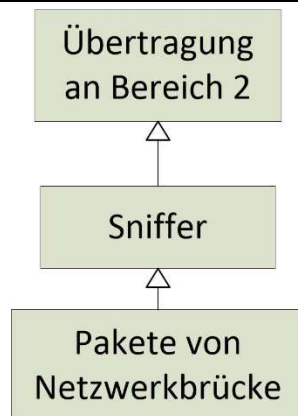


Abbildung 3 : Bereich 1

Der zweite Bereich umfasst die Analyse der erfassten Netzwerkpakete durch verschiedene Komponenten. Hierbei ist jede Komponente auf die Erkennung eines spezifischen Angriffs spezialisiert, z.B. SYN-Floods. Diese Komponenten generieren Warnungen, wenn ein Angriff entdeckt wird und übergeben diese Warnung an den dritten Bereich des Netzwerkmonitors.

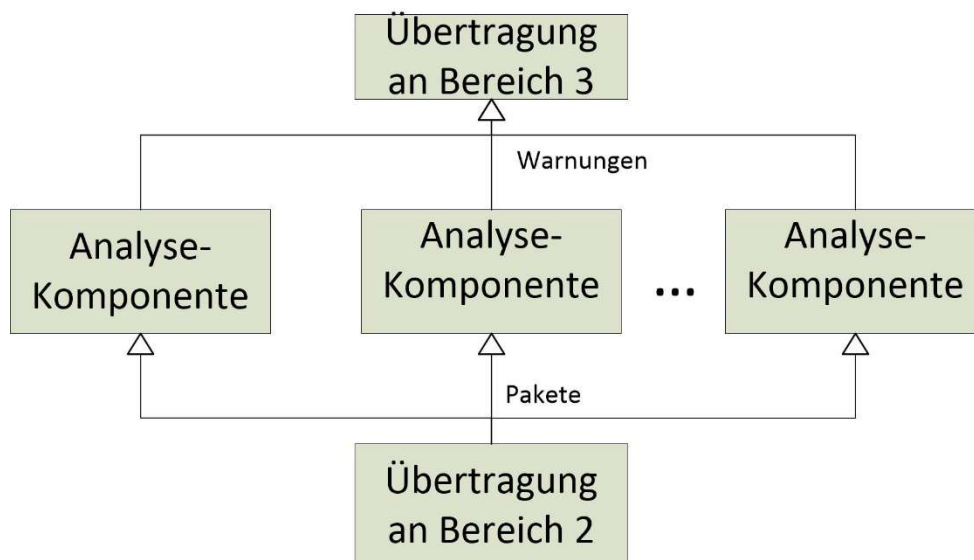


Abbildung 4 : Bereich 2

Der dritte Bereich enthält eine Warner-Komponente, dessen Aufgabe es ist, die einzelnen Warnungen zu analysieren ob sie Teil eines mehrstufigen oder wiederholten Angriffs sind.

Nach Durchführung dieser Analyse wird die Entdeckung eines Angriffes dem Nutzer mitgeteilt. Die geschieht mithilfe der LEDs des Kleinstrechners und E-Mails die an den Nutzer geschickt werden.

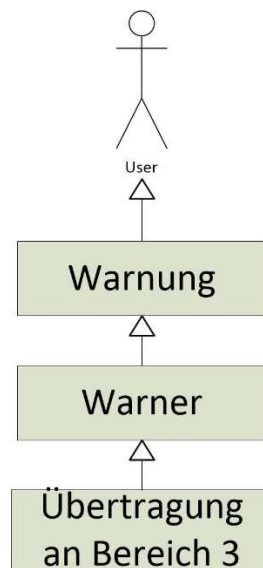


Abbildung 5 : Bereich 3

Die einzelnen Bereiche des Netzwerkmonitors sollen möglichst so miteinander kommunizieren, dass einzelne Komponenten hinzugefügt oder ausgetauscht werden können, ohne dass die anderen Komponenten verändert werden müssen. Es soll zum Beispiel für den ersten Bereich nicht relevant sein, wie viele Analysekomponenten die Pakete erhalten und auswerten wollen. In Hinblick auf die Erweiterung des Netzwerkmonitors soll es für den ersten Bereich keine Rolle spielen, ob sich einige Komponenten des zweiten Bereiches auf dem Kleinstrechner oder auf einem externen Rechner befinden. Daraus folgt auch, dass es den externen Komponenten, die sich im zweiten Bereich befinden, möglich sein muss Warnungen an den dritten Bereich zu senden. Auch soll es möglich sein Komponenten, die in unterschiedlichen Programmiersprachen geschrieben wurden, zu verwenden.

Eine Möglichkeit die Kommunikation zu gestalten, sind simple Funktionsaufrufe. Hierbei würde die Komponente, die für die Kommunikation zwischen zwei Bereichen zuständig ist, einfach eine Funktion einer Komponente im nächsten Bereich ansprechen, und die benötigten Daten als Parameter übergeben. Dies führt aber zu einer engen Kopplung der einzelnen Bereiche, bei Hinzufügen einer weiteren Komponente im zweiten Bereich müsste, die Komponente, die im ersten Bereich die Paketdaten für den zweiten Bereich bereitstellt, angepasst werden. Auch ist es oftmals schwierig Funktionen, die in einer anderen Sprache geschrieben sind, oder sich in einer externen Komponente befinden, aufzurufen.

Vorteil einer solchen Lösung ist die hohe Performance, es müssen keine I/O-Operationen, z.B. das Schreiben von Daten auf die Festplatte, vorgenommen werden um Informationen zwischen den Bereichen auszutauschen.

Aufgrund der engen Kopplung, und der Schwierigkeit bei Verwendung mehrerer Programmiersprachen sind Funktionsaufrufe keine gute Lösung.

Eine andere Möglichkeit ist die Verwendung einer Datenbank, mit deren Hilfe die einzelnen Bereiche miteinander kommunizieren. Hierbei würden die Pakete, die im ersten Bereich erfasst wurden in eine Datenbank gespeichert, und von dort für Komponenten aus dem zweiten Bereich abrufbar sein. Genauso mit Warnungen, die von Bereich 2 an Bereich 3 übertragen werden sollen.

Dieses Vorgehen hat einige Vorteile. So sind die einzelnen Bereiche voneinander getrennt, Bereich 1 muss kein Wissen über die Komponenten des zweiten Bereiches haben, um die Pakete zu übergeben. Auch Nutzung unterschiedliche Programmiersprachen und externer Komponenten ist kein Problem, verfügen doch viele Programmiersprachen über Möglichkeiten um mit Datenbanken, unabhängig von ihrer Lokation, zu kommunizieren. Problematisch ist aber, dass das Lesen und Schreiben von Daten in einer Datenbank mit einem relativ hohen Aufwand verbunden ist, verglichen mit dem ersten Lösungsansatz.

Eine dritte Möglichkeit ist die Verwendung von Messagequeues zur Kommunikation zwischen den Bereichen. Hier würde der erste Bereich die Pakete an eine Messagequeue übergeben, die von Komponenten des zweiten Bereiches ausgelesen wird. Die Warnungen des zweiten Bereiches würden über eine weitere Messagequeue an den dritten Bereich übertragen werden.

Diese Lösung bietet die gleichen Vorteile wie die Datenbank-Lösung. So sind die einzelnen Bereiche unabhängig voneinander, sie müssen nur die Messagequeue kennen, nicht die Komponenten der anderen Bereiche. Auch die Verwendung unterschiedlicher Programmiersprachen ist möglich, solange die jeweilige Programmiersprache mit der Messagequeue kommunizieren kann. Weiterhin sind Messagequeues, genau wie Datenbanken auch von externen Komponenten erreichbar. Ein Nachteil der Messagequeue-Lösung ist der im Vergleich zur Funktionsaufrufs-Lösung höherer Aufwand, der zur Übertragung einer Nachricht nötig ist. Allerdings ist dieser Aufwand geringer als bei der zweiten Lösung.

Aufgrund der geringen Kopplung der Bereiche und relativ guten Leistung werden Messagequeues als Mittel der Kommunikation zwischen den Bereichen genutzt.

Als Datenformat der ersten Messagequeue wurde das LibPcap-Format gewählt, da dieses im Bereich der IT-Sicherheit weit verbreitet und von vielen Programmen genutzt wird. Dabei wird der Global Header des Datenformates weggelassen. Die Messagequeue enthält demnach eine stetige Folge von Paketen, wie auf Abbildung 7, nur ohne den Global Header, zu sehen ist. Dies führt dazu, dass sich jederzeit weitere Komponenten bei der Messagequeue anmelden können und nicht nur zu Beginn der Übertragung.

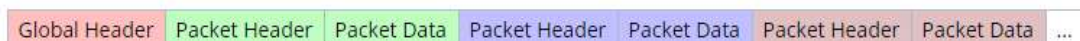


Abbildung 6 : LibPcap-Datenformat [Wire15a]

Die einzelnen Pakete im LibPcap-Datenformat werden über eine Publisher-Subscriber-Messagequeue den einzelnen Analyse-Komponenten des zweiten Bereiches zur Verfügung gestellt, wobei der erste Bereich als Publisher auftritt. Im Publish-Subscribe-Pattern sendet ein oder mehrere Publisher Daten an eine beliebige Anzahl von Subscribern ohne zu wissen ob und wie viele Subscriber es gibt. Subscriber erhalten dabei nicht alle Daten der Publisher sondern nur solche, die zu Ihrer Subscription passen. Ein Beispiel hierfür ist ein Publisher, der Wetterdaten für alle Postleitzahlen der USA überträgt. Ein Subscriber ist aber nur an den Daten für einen Ort interessiert. Er setzt also eine Subscription um nur Daten für diesen Ort zu erhalten. Andere Daten werden nicht an diesen Subscriber gesendet. [vgl. Jaco09] und [Hint13 S.9-14]

Die Verwendung dieses Patterns führt dazu, dass sich beliebig weitere Komponenten bei der Messagequeue als Subscriber registrieren können, und so Zugriff auf die Paketdaten haben. Die Messagequeue ist auch für Subscriber, die nicht auf dem Kleinstrechner installiert sind zu erreichen. So ist das Auswerten der Paketdaten auch auf anderen Rechnern möglich, z.B. um die Daten, die der Netzwerkmonitor erfasst hat mit denen anderer Intrusion Detection Systeme zu verknüpfen.

Analyse-Komponenten im zweiten Bereich werten die Daten der einzelnen Netzwerkpakete im LibPcap-Format aus und können über eine weitere Messagequeue Warnungen an den dritten Bereich weiterleiten, falls die Auswertung Hinweise auf einen Angriff vorfindet. Die Messagequeue, die Warnungen an den Warnungsauswerter liefert, tut dies im Round-Robin-Verfahren. Es wird also reihum von jeder Analyse-Komponente eine Warnung verarbeitet, sodass nicht eine Komponente mit besonders vielen Warnungen alle anderen „übertönt“.

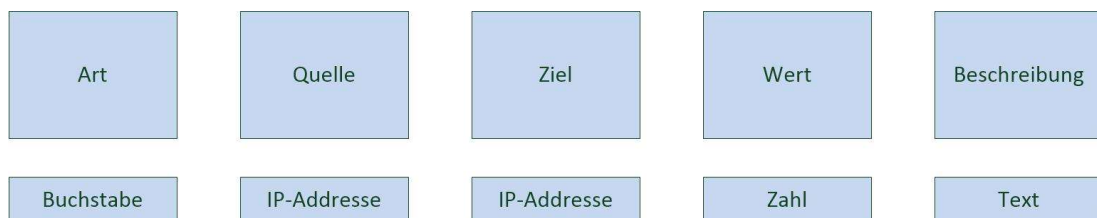


Abbildung 7 : Format einer Nachricht der zweiten Messagequeue

Das Format einer Warnung ist auf Abbildung 7 einzusehen. Die einzelnen Teile der Warnung sind mit einem „|“ Zeichen getrennt. Die Art beschreibt in einem Buchstaben ob es sich bei der Nachricht um eine Warnung oder einen Alarm handelt. Die Quelle ist die Adresse des Verursachers des Angriffs, das Ziel, die des Opfers. Der Wert ermöglicht die Einordnung des Vorfalls in einen mehrstufigen Angriff, je höher der Wert, desto später in einer Angriffskette taucht er auf. Die Beschreibung beschreibt den Vorfall, z.B. „DOS-Attack caused by Synflood“. Auch externen Komponenten des zweiten Bereiches können ihre Warnungen an diese Messagequeue übergeben.

Die Aufgabe des Warnungsauswerters ist es, die einzelnen Warnungen darauf zu überprüfen, ob sie Teil eines mehrstufigen oder wiederholten Angriffes sind. Ein mehrstufiger Angriff ist als dritter Use Case in Kapitel 1.3 beschrieben. Bei einem wiederholten Angriff handelt es sich um einen Angreifer der in kurzer Zeit eines oder mehrere Ziele im Netzwerk mehrmals angreift.

Nach der Auswertung der Warnung teilt der Auswerter das Ergebnis dem Nutzer mit. Dies geschieht über eine E-Mail an den Nutzer, Anzeige des Warnungs- und Alarmzustandes auf einer Webseite und Aufleuchten verschiedener LEDs am Netzwerkmonitor.

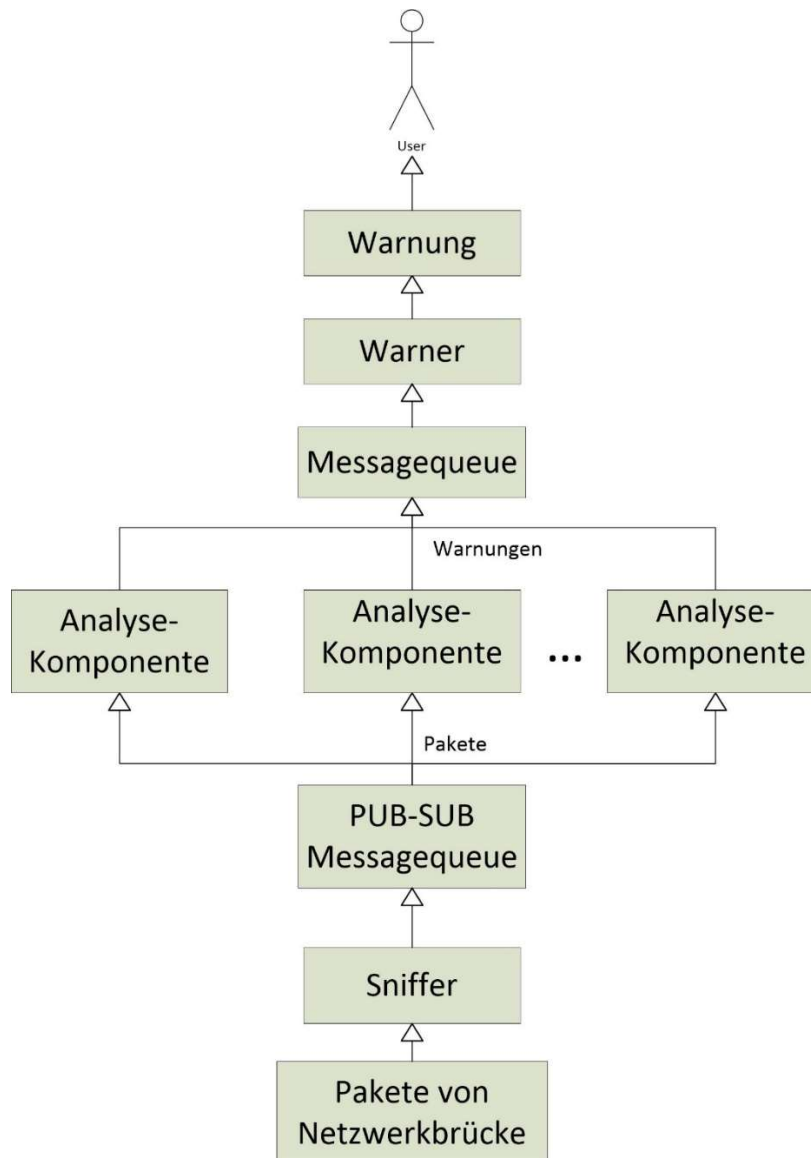


Abbildung 8 : Architektur des Netzwerkmonitors

3.2 Phytex Board



Abbildung 9 : Phytex phyBOARD®-Wega Lieferumfang

Das Phytex phyBOARD®-Wega ist der Kleinstrechner, der die Basis des Netzwerkmonitors bildet. Phytex ist ein deutscher Entwickler von Mikroprozessormodulen und Kleinstrechnern (Single Board Computer). Dies bietet den Vorteil eines deutschsprachigen Supports, zur Lösung eventuell auftretender Probleme.

Das Board verfügt über zwei Netzwerkanschlüsse und drei steuerbare LEDs, weitere Technische Daten befinden sich im Anhang. Andere Kleinstrechner, wie Raspberry Pi, BeagleBone Black oder Intels Galileo und Edison, verfügen zwar über ähnliche Leistung wie das Wega, aber nicht über mehrere Netzwerkanschlüsse. [vgl. Isik15 S.43-53]

Teil des Lieferumfangs des Kleinstrechners ist eine virtuelle Maschine, die auf einem beliebigen Laptop- oder Desktop-Computer eingerichtet werden kann. Auf der virtuellen Maschine, die auf Ubuntu-Basis arbeitet, ist bereits die Software, die zur Arbeit mit dem phyBOARD®-Wega benötigt wird, installiert. Dadurch fällt der Einstieg in die Arbeit mit dem

Wega leicht. Darunter befindet sich auch das Yocto Projekt, mit dessen Hilfe das Betriebssystem des Wega, eine auf Linux basierte Distribution namens Phyttec Yogurt erstellt worden ist. [vgl. Phyt16]

3.3 Software

In den folgenden Unterkapiteln wird die verwendete Software vorgestellt.

3.3.1 Yocto Project

Das Yocto Projekt ist ein Open Source Projekt, basierend auf dem OpenEmbedded Projekt, das Templates, Werkzeuge und Methoden zur Erstellung von Linux-basierten Systemen für Embedded-Produkte unabhängig der Hardware-Architektur bereitstellt. [vgl. Yoct16]

Es ermöglicht die Anpassung des Betriebssystems des Wega, indem durch Verwendung verschiedener Tools und Subprojekte des Yocto Projektes eine neue Version des Phyttec Yogurt Betriebssystems erstellt und anschließend auf das Wega aufgespielt werden kann. Eine so angepasste Variante des Betriebssystems bildet die Basis des Kleinstrechners.

Einzelne Features des Betriebssystems befinden sich hierbei als Rezepte in verschiedenen Layern des OpenEmbedded Projektes. Ein Rezept enthält Informationen über ein einzelnes Software Projekt, z.B. LibPcap oder ZeroMQ. Einige Informationen sind die Abhängigkeiten von anderen Rezepten, Quelle des Source-Codes, und wie dieser abzurufen, konfigurieren und kompilieren ist. Layers sind Sammlungen von Rezepten, die als Git-Repositories verwaltet werden. Wichtigstes Tool ist bitbake, die Build Engine. Sie verarbeitet die Informationen der Rezepte und erstellt aus ihnen ein neues Linux-basiertes Betriebssystem. [vgl. Phyt16b S.1-3]

3.3.2 LibPcap/JnetPcap

Zur Entwicklung der Sniffer-Komponente wird eine Möglichkeit benötigt, um auf den Netzwerkverkehr zuzugreifen. Hierfür soll die LibPcap Bibliothek genutzt werden.

LibPcap ist eine in C/C++ geschriebene Bibliothek zum Erfassen von Netzwerkverkehr. [vgl. Tcps15] Sie wird unter anderem von Wireshark verwendet. [vgl. Wire08]

Die Sniffer-Komponente soll in Java entwickelt werden, deshalb wird ein Wrapper benötigt, der Zugriff auf LibPcap in Java erleichtert. Verwendet wird hierfür JnetPcap, ein Java Wrapper für LibPcap. JnetPcap erlaubt außerdem in Echtzeit das Decodieren der erfassten

Pakete aus dem LibPcap-Datenformat. [vgl. Slyt14] Deshalb wird JnetPcap auch zur Analyse der Pakete in verschiedenen Analysekomponenten verwendet.

Andere Java-Bibliotheken, wie z.B. JPCap [vgl. Char13], die über einen ähnlichen Funktionsumfang verfügt, wird bereits seit mehreren Jahren nicht mehr weiterentwickelt, während bei JnetPcap seit 2015 an einer neuen Version gearbeitet wird. [vgl. Bedn15]

Durch Installation von LibPcap auf dem Netzwerkmonitor kann die vorhandene Sniffer-Komponente einfach durch eine Andere ausgetauscht werden; der Zugriff auf die Pakete ist bereits vorhanden.

3.3.3 ZeroMQ Messagequeue

„ZeroMQ (also known as ØMQ, 0MQ, or zmq) looks like an embeddable networking library but acts like a concurrency framework. It gives you sockets that carry atomic messages across various transports like in-process, inter-process, TCP, and multicast. You can connect sockets N-to-N with patterns like fan-out, pub-sub, task distribution, and request-reply. It's fast enough to be the fabric for clustered products. Its asynchronous I/O model gives you scalable multicore applications, built as asynchronous message-processing tasks. It has a score of language APIs and runs on most operating systems.” [Hint13 S.ix]

ZeroMQ ist die Messagequeue, die genutzt wird um die zwei Messagequeues des Netzwerkmonitors zu implementieren.

Ein Vorteil von ZeroMQ ist die Unterstützung von 51 verschiedenen Programmiersprachen. [vgl. Imat14] Dadurch können die einzelnen Komponenten in all diesen unterschiedlichen Sprachen geschrieben sein, und können trotzdem miteinander kommunizieren.

Weiterer Vorteil von ZeroMQ ist die hohe Performance. So ist ein Laptop mit einem Intel i5 Prozessor aus dem Jahre 2011 in der Lage über eine Pub-Sub-Messagequeue 10 Millionen Nachrichten innerhalb von weniger als 5 Sekunden zu filtern. [vgl. Hint13 S.14]

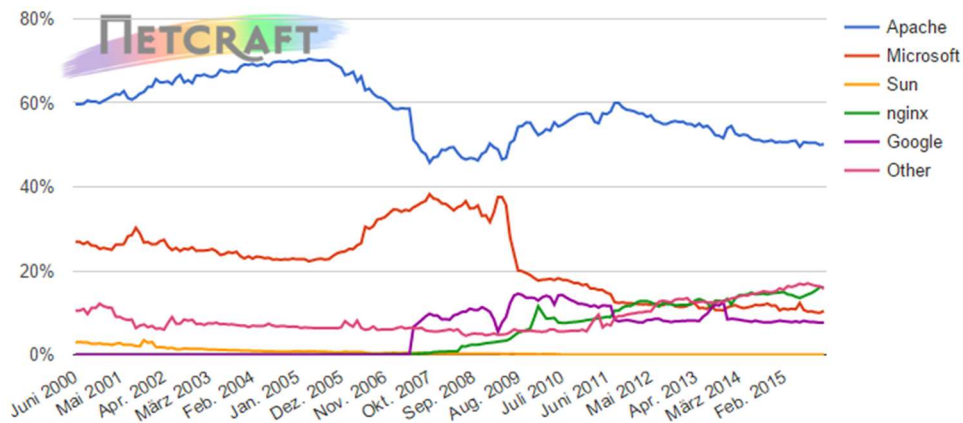
Auch benutzt ZeroMQ anders als andere Messagequeue-Systeme, wie z.B. RabbitMQ keinen zentralen Server als Messagebroker. Solch ein Messagebroker hat negative Auswirkungen auf die Performance von Messagequeues. So kann ZeroMQ mehrere Hunderttausend Nachrichten pro Sekunde verarbeiten, Messagequeues mit einem Broker je nach Messagequeue-System nur wenige Zehntausend. [vgl. Trea14] Aufgrund dieser hohen Performance wurde ZeroMQ als Messagequeue gewählt.

3.3.4 Apache Webserver

Um die Nutzung der Statusseite im Webbrowser zu ermöglichen, wird ein Webserver benötigt. Der Webserver muss auf dem Betriebssystem des phyBOARD Wega lauffähig sein. Zudem sollte der Webserver verschiedenste Programmiersprachen unterstützen um die Weiterentwicklung des Netzwerkmonitors zu vereinfachen

Diese Eigenschaften erfüllt der Apache HTTP Server mit der Möglichkeit durch Installation von Modulen verschiedenste Programmiersprachen zu unterstützen.

Auch ist der Apache der meistverbreitete Webserver, wie auf Abbildung 10 zu erkennen ist. So ist bei Auftreten von Problemen die Wahrscheinlichkeit groß ist, dass ein anderer Entwickler bereits eine Lösung für das Problem gefunden und im Internet veröffentlicht hat. Auch ist bereits Erfahrung mit Apache und PHP, nicht aber mit anderen Webservern und Sprachen vorhanden. Ein weiterer Vorteil des Apache Webservers ist neben seiner großen Verbreitung, sein offener Quellcode.[vgl. Apac16] Dies führt dazu, dass schnell auf auftretende Sicherheitslücken oder andere Fehler des Servers reagiert werden kann. Der Fakt, dass der Apache Webserver bereits seit 1995 entwickelt wird, ist ein weiteres Argument für den Server, da anzunehmen ist, dass der offene Quellcode, der seit 20 Jahren stetig weiterentwickelt wird, kaum noch Fehler enthält.



Developer	November 2015	Percent	December 2015	Percent	Change
Apache	86,528,264	49.99%	86,135,302	50.14%	0.14
nginx	27,855,455	16.09%	27,480,550	16.00%	-0.10
Microsoft	17,289,855	9.99%	17,887,532	10.41%	0.42
Google	13,182,501	7.62%	13,196,505	7.68%	0.06

Abbildung 10 : web server developers: Market share of active sites [Netc15]

4 Entwicklung und Einrichtung

Dieses Kapitel stellt die Einrichtung der Plattform und die Entwicklung der Software vor. Zuerst wird aufgezeigt wie das Betriebssystem des phyBOARD-Wegas angepasst und die Netzwerkbrücke eingerichtet wurde. Anschließend werden die einzelnen Software-Komponenten vorgestellt.

Der Aufbau der Software folgt hierbei der in Kapitel 3.1 vorgestellten Architektur. Die Netzwerkbrücke leitet Pakete weiter, während die Sniffer-Komponente die Pakete erfasst. Die Pakete werden anschließend von der Sniffer-Komponente, mit Hilfe der PacketDistributor-Komponente und einer Publish-Subscribe Messagequeue an verschiedenste andere Komponenten weitergeleitet. Abnehmer sind verschiedene Analyse-Komponenten, wie Synflood- und Teamviewer-Detektions-Komponente, sowie die Wireshark-Anbindung.

Von der Synflood- oder Teamviewer-Komponente erzeugte Warnungen werden mithilfe einer weiteren Messagequeue an die Warner-Komponente weitergeleitet.

Diese analysiert die Warnungen, ob sie Teil eines mehrstufigen oder wiederholten Angriffes sind, und erzeugt anschließend für den User sichtbare Warnungen, mithilfe der Email-Komponente, dem Ledcontrol-Shellskript und der Statusseite.

Alle Komponenten, mit Ausnahme der Statusseite, Ledcontrol und Netzwerkbrücke sind in Java geschrieben.

Viele der Komponenten verfügen über eine Konfigurationsdatei, mit deren Hilfe die einzelnen Komponenten angepasst werden können.

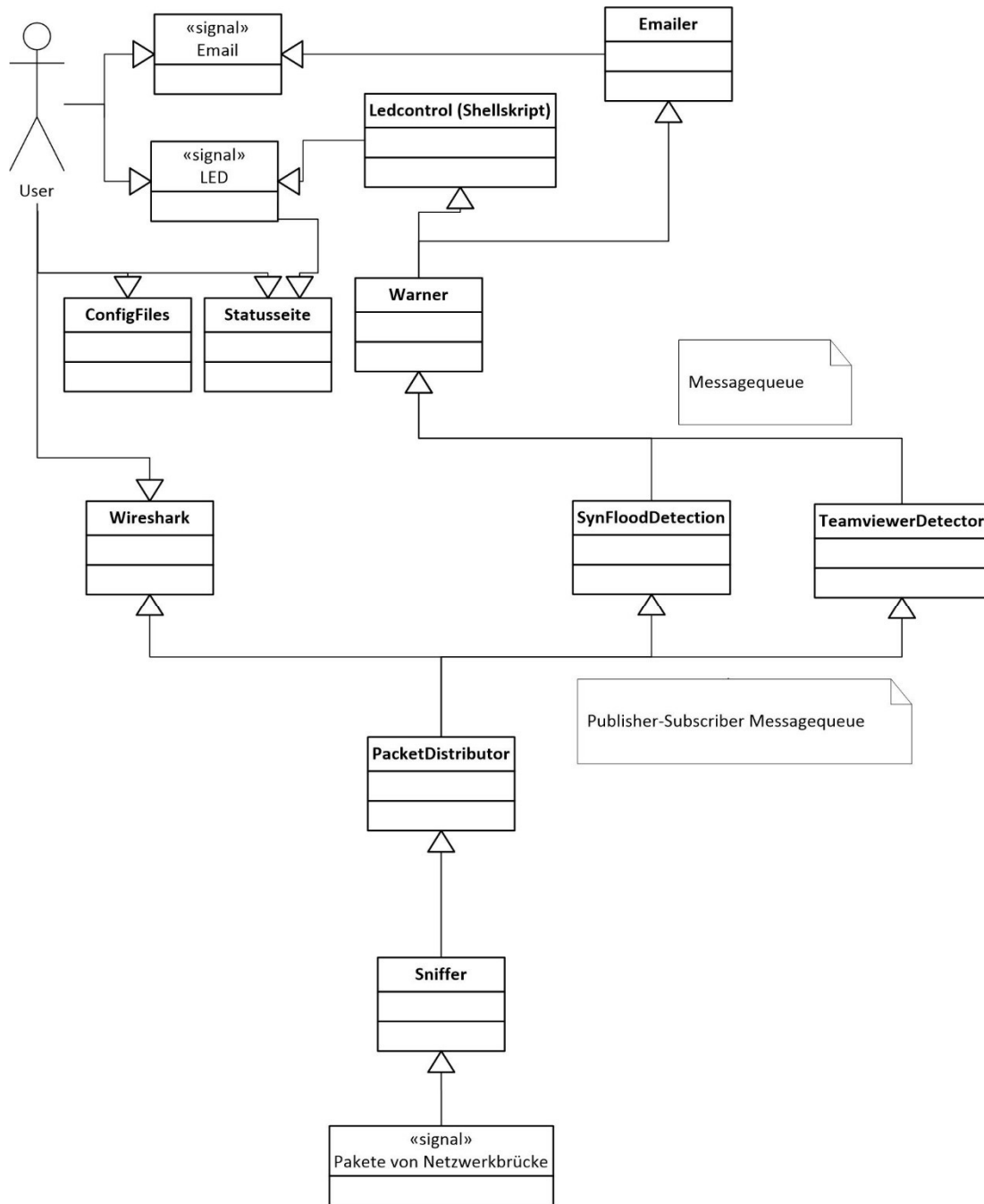


Abbildung 11 : Überblick über die Komponenten des Netzwerkmonitors

4.1 Plattform

Die Plattform umfasst die Software, die benötigt wird um die Hardware des Netzwerkmonitors zu betreiben. Dies umfasst hauptsächlich das Betriebssystem, eine auf Yocto 1.7 basierende Linux Distribution. Im ersten Kapitel wird dargestellt wie durch Verwendung des Yocto Projects und der von Phyttec zur Verfügung gestellten virtuellen Maschine, das Betriebssystem des phyBOARD Wegas angepasst wurde. Weiterhin wird die Einrichtung verschiedenster Funktionen des Betriebssystems beschrieben. Das zweite Kapitel dient der Beschreibung der Konfiguration der Netzwerkbrücke. Weiterhin wird die Ansteuerung der LEDs erklärt.

4.1.1 Linux

Das Phyttec Yogurt Betriebssystem ist in drei verschiedenen Varianten verfügbar.

Einmal das phytec-headless-image für nicht-graphische Anwendungen, dann das phytec-qt5demo-image für Qt5 und andere graphische Anwendungen, die dann durch den auf dem Board vorhandenen HDMI-Anschluss dargestellt werden können. Die dritte Variante ist das phytec-headless-image-sdk, eine Variante des ersten Images auf dem bereits zusätzliche Entwicklungs-Tools, u.a. C-Compiler, make und anderen, die zum Installieren von Linux-Programmen ohne Nutzung eines Paketmanagers nötig sind, installiert sind.

Das Image wurde mit Hilfe verschiedener Tools des Yocto Projektes zudem um LibPcap und ZeroMQ erweitert und anschließend auf das Wega aufgespielt.

Weitere benötigte Software (Apache2, PHP, JnetPcap, Zero MQ Language Bindings für Java) wurde dann per Hand, also mithilfe des Quellcodes, dem Make-Befehl, etc., auf das Wega installiert. Gesteuert wird das Wega dabei entweder über eine serielle Schnittstelle, die mithilfe eines USB-Adapters an einen anderen Rechner angeschlossen wird, oder über das Netzwerk via SSH.

Phyttec Yogurt nutzt systemd zum Verwalten des Systems und seiner Services.

Es wird auch genutzt um die verschiedenen Komponenten des Netzwerkmonitors beim Hochfahren des Wega zu starten und während des Betriebes ihren Status zu überwachen.

Einzelne Komponenten sind hierbei jeweils ein eigener Service. Neue Komponenten können leicht als weitere Services hinzugefügt werden und sind dadurch unabhängig von bestehenden Komponenten.

Zum Erstellen eines neuen Services wird eine .service-Datei benötigt.

Diese Datei enthält Informationen darüber von welchen anderen Services der neue Service abhängig ist, in diesem Fall soll der Service gestartet werden, nachdem die Netzwerkbrücke br0 gestartet wurde (Requires / After). Weiterhin wird angegeben welches Programm vom Service gestartet werden soll, hier ein Shellskript, das die .Jar-Datei des Synflood-Detektors ausführt.

Systemd verfügt über verschiedene Ziele (Targets), je nach gewünschter Umgebung des Betriebssystems, multi-user.target, z.B. ist für ein nicht-graphisches Mehr-User-System zuständig, graphical.target für ein Mehr-User-System mit graphischer Oberfläche.[Free15]



```
synflood-detector.service x
1 [Unit]
2 Description=Starts the synflood detector
3 Requires=sys-subsystem-net-devices-br0.device
4 After=sys-subsystem-net-devices-br0.device
5
6 [Service]
7 Type=simple
8 ExecStart=/usr/bin/synflood-detector
9
10 [Install]
11 WantedBy=multi-user.target
```

Abbildung 12 : .service-Datei zum Einrichten eines neuen systemd-Services

Nach Einrichtung einer .service-Datei muss der Service noch mit dem Shell-Befehl:

```
Systemctl enable <Pfad zur .service-Datei>
```

aktiviert werden. Nun wird der Service bei jedem Start des Wega ausgeführt.

4.1.2 Bridge

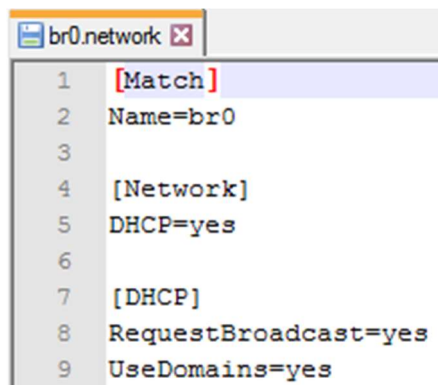
Die Netzwerkbrücke ist mithilfe des Systemd-networkd, ein Service der Teil von Systemd ist, eingerichtet worden. Durch Einrichtung und Verwaltung der Brücke durch einen Teil des Betriebssystems ist die Brücke von den restlichen Komponenten des Netzwerkmonitors getrennt und kann auch ohne Sie funktionieren. Selbst wenn also z.B. der Sniffer abstürzen sollte, wird davon die Bridge nicht beeinflusst und funktioniert normal weiter. Die Einrichtung der Brücke erfolgt hierbei durch mehrerer Konfigurationsdateien, die networkd mitteilen, welche Eigenschaften die Brücke haben soll. Zur Einrichtung der Brücke werden 4 verschiedene Dateien benötigt:



```
1 [NetDev]
2 Name=br0
3 Kind=bridge
```

Abbildung 13 : Erstellung des Brücken-Netzwerk-Interfaces

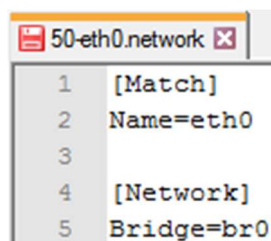
Die br0.netdev Datei erstellt ein virtuelles Netzwerkinterface, welches durch eine .network Datei konfiguriert werden kann.



```
1 [Match]
2 Name=br0
3
4 [Network]
5 DHCP=yes
6
7 [DHCP]
8 RequestBroadcast=yes
9 UseDomains=yes
```

Abbildung 14 : Konfiguration des Brücken-Netzwerk-Interfaces

Die br0.network Datei konfiguriert das virtuelle Netzwerkinterface der Netzwerkbrücke. Zur Verwendung im Heimnetzwerk des Autors ist sie für DHCP konfiguriert um das Weg von einem anderen Rechner erreichen zu können. Dies ist notwendig um z.B. auf die Statusseite zuzugreifen oder die Wireshark-Anbindung zu nutzen. Ohne Einrichtung einer IP-Adresse in dieser Datei ist der Netzwerkmonitor nicht über das Netzwerk zu erreichen. Weitere Konfiguration muss dann über die serielle Schnittstelle stattfinden. RequestBroadcast für DHCP-Nachrichten ist notwendig, da normale DHCP-Nachrichten den Netzwerkmonitor nicht erreichen.



```
1 [Match]
2 Name=eth0
3
4 [Network]
5 Bridge=br0
```

Abbildung 15 : Einrichtung einer der beiden Netzwerkinterfaces

In den eth0 und eth1 Dateien werden die beiden realen Netzwerkinterfaces des Wegs für den Brückenbetrieb eingerichtet.

4.1.3 LEDs

Das PhyBOARD WEGA verfügt über drei steuerbare LEDs, jeweils eine rote, gelbe, und grüne. Steuerung der LEDs erfolgt über mehrere Dateien, deren Inhalt das Verhalten der LEDs kontrolliert. So verfügt jede LED über eine brightness-Datei, die eine Zahl zwischen 0 und 1 enthält, wobei 0 für aus und 1 für maximale Helligkeit steht. Um die Steuerung der LEDs zu vereinfachen kann die Helligkeit, unter Verwendung des Shellskriptes, welches in Abbildung 16 sehen ist, gesteuert werden.

```
#!/bin/bash

//This Script allows to control the LEDs of the
//Phytec Phyboard WEGA

//no arguments
if [ $# == 0 ]
then
    echo "need at least one argument"
    exit 1
fi

//show the help
if [ $1 == "--help" ]
then
    echo "led-control <color> <state>"
    echo "available colors are : red, green, yellow"
    echo "available states are 1 (on), 0 (off)"
    exit 1
fi

//not the right number of arguments
if [ $# != 2 ]
then
    echo "Wrong number of arguments"
    exit 1
fi

//unknown LED-Color selected
if [ $1 != red -a $1 != "yellow" -a $1 != "green" ]
then
    echo "Wrong LED-Color. Only available colors are red/yellow/green"
    exit 1
fi

//Wrong LED-State selected
if [ $2 != 1 -a $2 != 0 ]
then
    echo "Wrong state. Only available states are 1 (on), 0 (off)"
    exit 1
fi

//Set State of selected LED
cd /sys/devices/user_leds.7/leds/user_led_$1
echo $2 > brightness
```

Abbildung 16 : Shellskript zur Steuerung der LEDs

4.2 Anwendung

In diesem Kapitel werden die verschiedenen Implementationen der Softwarekomponenten, die in Kapitel 3.1. erläutert wurden, vorgestellt.

Die Reihenfolge, in der die Komponenten vorgestellt werden, orientiert sich dabei am Weg, die ein Paket zurücklegt, vom sniffen an der Netzwerkbrücke, über Auswertung durch verschiedene Intrusion-Detection-Systeme, bis zur Meldung eines Vorfalls an den Nutzer.

Alle Java-Komponenten werden als ausführbare JAR-Dateien auf dem Netzwerkmonitor installiert, und wie in Kapitel 4.1.1 beschrieben, als Service durch Systemd gestartet.

Durch Bereitstellung der Komponenten als JARs können diese leicht auf einen anderen Kleinstrechner übertragen werden, vorausgesetzt weitere benötigte Software, wie JnetPcap / LibPcap, ZeroMQ, sind vorhanden.

4.2.1 Sniffer

Die Aufgabe der Sniffer-Komponente ist es, die von der Brücke übertragenen Pakete zu erfassen und mit Hilfe des Paketverteilers, den anderen Komponenten zur Verfügung zu stellen. Der Sniffer arbeitet als ein Thread, der die Netzwerkpakete von der Netzwerkbrücke ausliest und durch Verwendung der Paketverteiler-Komponente an die Publish-Subscribe Messagequeue übergibt. Der Thread wird von der SnifferStarter Klasse erzeugt, und bei Erhalt eines Shutdown-Signales auch wieder beendet, wie in Abbildung 17 zu sehen ist. Dieses Verfahren wird auch zum Starten und Beenden der anderen Java-Komponenten verwendet.

```
8 public class SnifferStarter {
9     static Sniffer sniffer;
10
11     public static void main(String[] args) throws IOException {
12         //Starting a sniffer + packetDistributor
13         PacketDistributor packetDistributor = new packetDistributorImpl();
14         sniffer = new Sniffer(packetDistributor);
15
16         sniffer.start();
17
18         //Interrupting the sniffer when shutting down
19         Runtime.getRuntime().addShutdownHook(new Thread() {
20             public void run() {
21                 sniffer.interrupt();
22             }
23         });
24     }
25 }
```

Abbildung 17 : Starter des Sniffers

Im Konstruktor des Sniffers werden die Informationen aus der Konfigurationsdatei geladen. Diese umfassen den Namen der Brücke und die Länge des ReadTimeouts. Der Sniffer wartet nach Starten des Threads solange, bis die Netzwerkbrücke über eine IP-Adresse verfügt, um

sicherzugehen, dass sie bereit ist Pakete zu übertragen. Sollte die Brücke nicht bereit sein, würde der Sniffer abstürzen.

Anschließend werden einige weitere Einstellungen vorgenommen und ein LibPcap-Filter, der verhindert, dass vom Netzwerkmonitor ausgehende, oder an den Monitor adressierte Pakete erfasst werden, konfiguriert. Ein solcher Filter ist deutlich performanter, arbeitet er doch auf dem Kernel-Level, als jedes Paket einzeln im Java-Code auf seine Herkunft oder Zielort zu überprüfen.

Wichtigste Einstellungen sind die Länge des ReadTimeouts, und die Anzahl der Bytes, die von jedem Paket erfasst werden sollen. Das ReadTimeout verhindert, dass jedes Paket einzeln vom Betriebssystem an den Sniffer übertragen wird, sondern sorgt dafür, dass das Betriebssystem entweder solange wartet bis die Zeit des Timeouts abgelaufen ist, oder der Packetbuffer des Betriebssystems voll ist, bevor es die Pakete an den Sniffer übergibt. Durch Einstellung der Byte-Anzahl kann die Anzahl der Pakete, die in den Packetbuffer passen erhöht werden. Außerdem erhöht sich durch kleinere Pakete die Performance aller Komponenten des ersten und zweiten Bereiches.

Auch ist mit einem niedrigen Timeout (<10 Sekunden) zu beobachten das schnell Pakete verloren gehen, da der Sniffer noch nicht alle Pakete abgearbeitet hat, bevor das Betriebssystem die nächsten Pakete bereitstellt. [vgl. Tcpd15b]

Daraufhin beginnt der Sniffer Pakete zu erfassen, in das LibPcap-Format umzuwandeln und durch den Paketverteiler an die Messagequeue zu übertragen. Dies geschieht durch eine while-Schleife, die solange die von JnetPcap bereitgestellte loop-Funktion aufruft, bis der Thread interrupted wird. Diese loop-Funktion würde normalerweise ohne Unterbrechung weitere Pakete bearbeiten, wurde aber so erweitert, dass sie alle 0.5 Sekunden, die JnetPcap-Loop verlässt, um die While-Bedingung überprüfen zu können. Dies ist notwendig, um bei Beendigung des Threads reservierte Ressourcen freizugeben.

```

126 //Run until thread is interrupted
127 while (!interrupted()) {
128
129     JBufferHandler<String> jpacketHandler = new JBufferHandler<String>() {
130
131         //function used by pcap.loop
132         @Override
133         public void nextPacket(PcapHeader header, JBuffer buffer, String user) {
134
135             //Prepare to write captured packet to messagequeue
136             ByteBuffer bb = ByteBuffer.allocate(16 + buffer.size());
137             bb.order(ByteOrder.nativeOrder());
138             bb.clear();
139             bb.putInt((int) header.hdr_sec());
140             bb.putInt(header.hdr_usec());
141             bb.putInt(header.hdr_len());
142             bb.putInt(header.hdr_wirelen());
143             bb.put(buffer.getByteArray(0, buffer.size()));
144             bb.flip();
145
146             //write to messagequeue
147             packetDistributor.distribute(bb.array());
148
149             //check for interrupts periodically
150             if (System.currentTimeMillis() > interval) {
151                 interval = System.currentTimeMillis() + INTERVAL;
152                 pcap.breakloop();
153             }
154         }
155     };
156     //Loop calls nextPacket in a loop
157     pcap.loop(0, jpacketHandler, "sniffer");
158
159 }
160 //Shutdown
161 System.out.println("Shutting down");
162 pcap.close();
163 packetDistributor.shutdown();
164 }
165 }

```

Abbildung 18 : Schleifen des Sniffers

4.2.2 Paketverteiler

Der Paketverteiler stellt die vom Sniffer erfassten Paketdaten anderen Komponenten des Netzwerkmonitors zur Verfügung.

Dies geschieht mit Hilfe der ZeroMQ Messagequeue und dessen Publisher-Subscriber-Pattern, wobei der Paketverteiler als Publisher und die anderen Komponenten als Subscriber agieren. Der Verteiler kann durch eine Konfigurationsdatei angepasst werden.

Sie ermöglicht die Einstellung des Netzwerkports, der von der Messagequeue genutzt werden soll. Sie arbeitet hierbei mit dem TCP-Protokoll um die Pakete auch Anwendungen zur Verfügung zu stellen, die nicht auf dem Netzwerkmonitor laufen.

```
public class packetDistributorImpl implements PacketDistributor {

    private Properties properties;
    private Context context;
    private Socket publisher;

    public packetDistributorImpl() throws IOException {

        //Loading of Config File
        this.properties = new Properties();
        InputStream input = new FileInputStream("/usr/bin/jars/packet_distributor.conf");
        properties.load(input);
        input.close();

        //Initialising ZeroMQ
        this.context = ZMQ.context(1);
        this.publisher = context.socket(ZMQ.PUB);
        String port = properties.getProperty("zmq_port");
        publisher.bind("tcp://*:" + port);
    }

    @Override
    //Sending a Packet to all subscribers
    public void distribute(byte[] packet) {
        publisher.send(packet, 0);
    }

    @Override
    //Shutting down ZeroMQ
    public boolean shutdown() {
        publisher.close();
        context.term();
        return true;
    }
}
```

Abbildung 19 : Teil des Codes des Paketverteilers

Die einzelnen Pakete werden als Java-Byte-Array vom Sniffer an den Paketverteiler übergeben, wobei das Array ein Paket im Libpcap-Format enthält. [vgl. Wire15] Dieses Format wird von vielen Tools, wie Wireshark, Snort, TcpDump und anderen unterstützt.

Die Messagequeue verteilt hierbei jede Nachricht des Publishers an alle Subscriber, die zum Zeitpunkt des Aussendens mit der Messagequeue verbunden sind und deren Filter auf die Nachricht zutreffen.

Filter sind eine Option, die von einem Subscriber eingestellt werden kann, um zu regulieren, welche Nachrichten er erhält. Es sind Strings oder Bytes, die mit dem ersten Element einer Nachricht verglichen werden. Nur bei Übereinstimmung erhält ein Subscriber die Nachricht. Das erste Element einer Nachricht des Paketverteilers ist der Packet-Header des Libpcap-Dateiformats. Dieser enthält Informationen über den Zeitpunkt, an dem das Paket vom Sniffer erfasst wurde, und die Länge des Pakets. Dadurch kann die Filterfunktion der

Messagequeue nur dazu genutzt werden, Pakete zu erhalten, die zu einem gewissen Zeitpunkt mit einer gewissen Länge gesendet wurden.

4.2.3 SYN-Flood Detektor

Die Aufgabe des SYN-Flood Detektors ist es die Netzwerkdaten, die der Detektor mithilfe der Pub-Sub Messagequeue vom Sniffer erhält, zu analysieren ob sie Hinweise auf einen DOS-Angriff, der auf Basis einer SYN-Flood funktioniert, enthalten. Zur Analyse der Pakete wird der CUSUM-basierte Algorithmus verwendet, der von Wang, Zhang und Shin in „Detecting SYN Flooding Attacks“ vorgestellt wurde. [vgl. WaZS02]

Gewählt wurde dieser Algorithmus aufgrund seiner Einfachheit. Dies wird u.a. daran deutlich, dass nicht wie bei anderen Algorithmen Informationen über jede einzelne TCP-Verbindung gespeichert wird. Dieser Algorithmus nutzt den normalen Ablauf eines Auf-/Abbaus einer TCP-Verbindung, wie er in Abbildung 17 zu sehen ist, aus, um eine SYN-Flood zu erkennen, indem die Anzahl der empfangenen SYN-Pakete mit der Summe der empfangenen FIN und RST-Pakete verglichen wird.

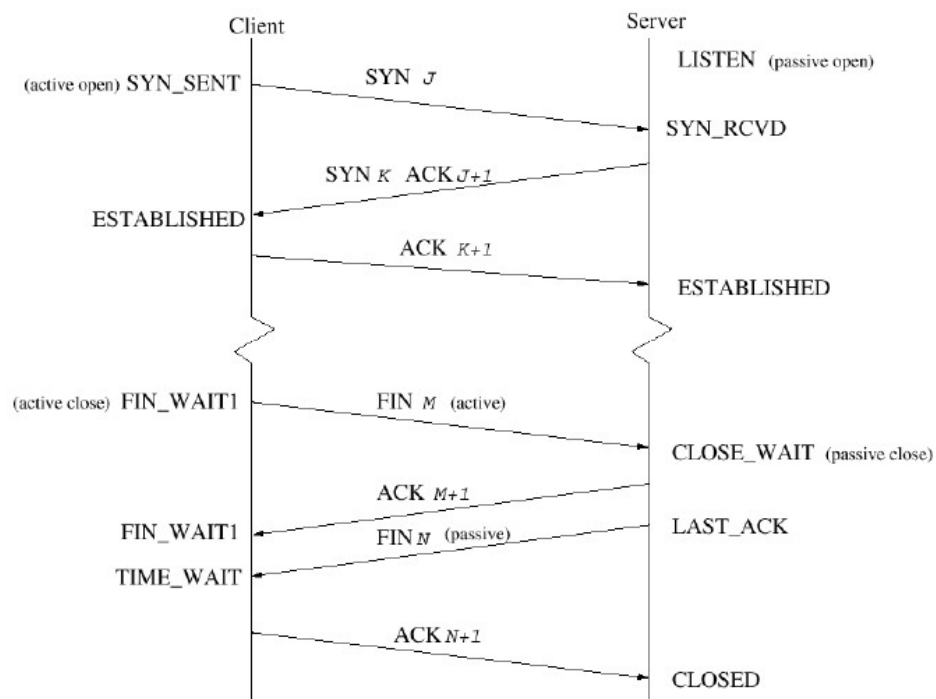


Abbildung 20 : Auf-/Abbau einer TCP-Verbindung [WaZS02 Fig. 1]

Sowohl FIN, als auch RST-Pakete können genutzt werden um eine TCP-Verbindung zu beenden. Alle SYN-, FIN- und RST-Pakete werden durchgehend von der Sniffer-Anbindung des Detektors erfasst und in einem PacketCounter gespeichert. Dabei werden auch Quelle und Ziel der einzelnen SYN-Pakete gespeichert, um bei Auswertung der Daten erkennen zu können, welche Adresse Quelle und Ziel des Angriffs sind. Da aber eine gewisse Zeit zwischen Auf- und Abbau einer TCP-Verbindung vergeht, können die SYN-Pakete nicht zum gleichen Zeitpunkt wie die FIN- und RST-Pakete gezählt werden. Deshalb beginnt und endet der Zeitraum in dem FIN- und RST-Pakete erfasst werden 10 Sekunden nach dem der SYN-Pakete, wobei 10 Sekunden der durchschnittlichen Dauer einer TCP-Verbindung entsprechen.

Daraus folgt, dass es sich bei der Differenz zwischen SYN- und FIN-/RST-Paketen um die Anzahl von Paketen handelt, die entweder eine langlebige TCP-Verbindung geöffnet haben, oder Teil eines SYN-Flood-Angriffes sind. Diese Differenz wird alle 20 Sekunden berechnet und lässt sich zusammen mit vielen weiteren Parametern des Algorithmus in einer Konfigurationsdatei anpassen. Unter Verwendung dieser Differenz und weiteren Formeln, die dem Artikel von Wang, Zhang und Shin entnommen werden können berechnet der Algorithmus einen neuen Entscheidungswert.[vgl. WaZS02 Kapitel III.b] Diese Abläufe werden in Abbildung 21 und 22 noch einmal verdeutlicht.

Wenn dieser Entscheidungswert nun einen Grenzwert überschreitet, wird eine Warnmeldung durch die zweite Messagequeue an den Warner geschickt. Die Warnung hat folgende Form:

„a|Quell-IP|Ziel-IP|25|DOS-Attack caused by Synflood“

Quell- und Ziel-IP sind hierbei einfach die Adressen, die am meisten SYN-Pakete gesendet, bzw. empfangen haben.

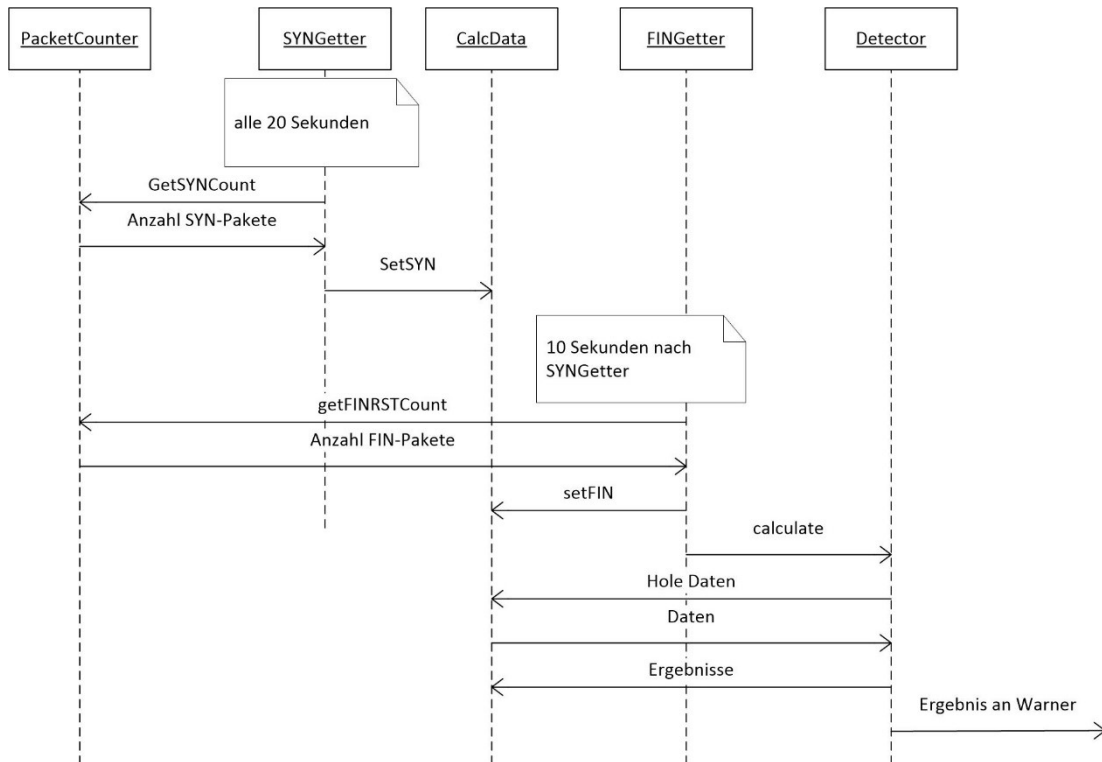


Abbildung 21 : Ablauf SYN-Flood-Detektor

```

15 public float calculate(CalcData calcData) {
16     //Algorithm based upon "Detecting SYN Flooding Attacks" by Wang, Zhang, Shin
17     //and is based upon CUSUM
18     int fins = calcData.getFIN();
19     int syms = calcData.getSYN();
20     //Average amount of FINS in last Sampling Period
21     float lastMeanFINS = calcData.getLastMeanFINS();
22     //Decision of the last Sampling Period
23     float lastDecision = calcData.getLastDecision();
24     // Average amount of recieved FIN-Packets
25     float meanFINS = memory * lastMeanFINS + (1 - memory) * fins;
26     if (meanFINS < 1f) {
27         meanFINS++; //Smaller FINS cause problems
28     }
29     calcData.setLastMeanFINS(meanFINS);
30     int delta = syms - fins;
31     float xn = delta / meanFINS; //Normalize delta
32     //To cause xn to drift towards the negative during normal operation
33     float tildeXn = xn - normal;
34     float decision = Java.Math.max(0, lastDecision + tildeXn); //New Decision
35
36     calcData.setLastDecision(decision);
37     return decision;
38 }
  
```

Abbildung 22 : Berechnung des Entscheidungswertes

4.2.4 Teamviewer Detektor

Die Aufgabe des Teamviewer Detektors ist es die Nutzung von der Teamviewer-Fernwartungs-Software außerhalb festgelegter Zeiten zu erkennen. Hierbei sind einige Daten wichtig. So nutzt Teamviewer Netzwerkport 5938 für den Netzwerkverkehr. Auch findet auch ohne aktive Fernwartungsverbindung ein gewisses Grundrauschen an Netzwerkverkehr, von Teamviewer ausgehend, auf diesem Port statt. Dieser Verkehr könnte verschiedene Ursachen haben. Es ist möglich dass die Teamviewer-Software nach Updates auf den Teamviewer-Servern sucht. Eine andere Möglichkeit ist, dass die lokale Teamviewer-Software den Teamviewer-Servern ein Signal sendet, dass sie aktiv und zur Fernwartung bereit ist. Aufgrund dieses Grundrauschens ist es notwendig, erst eine Warnung zu senden, wenn ein gewisses Paketvolumen erreicht wurde. Als angemessener Wert haben sich hierbei 100Pakete in 20 Sekunden herausgestellt Diese Werte, aber auch der erlaubte Zeitraum in dem Teamviewer genutzt werden darf, und einige andere Parameter lassen sich in einer Konfigurationsdatei einstellen.

Der Algorithmus der vom Teamviewer Detektor verwendet wird ist folgender (in Pseudocode):

```
If( IstTCPPaket & ZielPort = TeamviewerPort & aktueller Zeitpunkt ist nicht erlaubt )
    TeamviewerPaketAnzahl + 1
    If(TeamviewerPaketAnzahl > PaketGrenze)
        If(Es wurde bereits gewarnt)
            Timer zum Senden des „All Clear“ auf volle Laufzeit zurücksetzen
        Else
            Warnung an Warner senden
            Timer zum Senden des „All Clear“ starten
            Warnungszustand= Es wurde eine Warnung gesendet

    If(Kein Messintervall gestartet)
        IntervallBeginn = Aktueller Zeitpunkt
    Else
        If(Ende des Messintervalls erreicht)
            InterVallbeginn = Aktueller Zeitpunkt
            TeamviewerPaketAnzahl = 1
```

Der Algorithmus zählt also die Anzahl der TeamviewerPakete, die über einen gewissen Zeitraum anfallen und sendet bei Erreichen einer gewissen Anzahl eine Warnung. Wenn über einen gewissen Zeitraum keine Warnung mehr ausgelöst wird, wird durch einen Timer eine Nachricht über das Ende der Warnung an den Warner geschickt.

4.2.5 Warner

Aufgabe des Warners ist es, die Warnungen, die von den verschiedenen Analysekomponenten erzeugt werden, auf Zusammenhänge zwischen verschiedenen Warnungen zu überprüfen, ob also z.B. ein Angreifer mit verschiedenen Angriffsmethoden, die von jeweils verschiedenen Komponenten erkannt werden, versucht ein System zu kompromittieren.

Diese Zusammenhänge werden von Methoden die von Valeur, Vigna, Kruegel und Kemmerer im Artikel „A Comprehensive Approach to Intrusion Detection Alert Correlation“ [VVKK04] vorgestellt werden, insbesondere denen aus den Kapiteln über „Attack Thread Reconstruction“ und „Multistep Correlation“, erkannt.

Die einzelnen Warnungen erhält der Warner über eine ZeroMQ Push-Pull Messagequeue, wobei der Warner pullt, die Analysekomponenten ihre Warnungen pushen. Diese Variante der ZeroMQ Messagequeue zieht einzelne Nachrichten, die an sie geschickt wurden im Round Robin Verfahren. Wenn also mehrere Analysekomponenten Warnungen an den Warner schicken, wird reihum von jeder Komponente eine Warnung verarbeitet.



Abbildung 23 : Format der Warnungen

Bei Verarbeitung der einzelnen Warnungen werden zuerst durch die Art der Warnung die LEDs gesteuert. Art „a“ für Alert löst das rote LED aus, „w“ für Warnung das gelbe. Ein „z“ gefolgt von einem „a“ oder „w“ deaktivieren das jeweilige LED wieder. Um zu verhindern, dass die LEDs deaktiviert werden, nachdem mehrere Warnungen sie aktiviert haben, zählt der Warner die Anzahl der erhaltenen Aktivierungen und Deaktivierungen. Erst wenn die Anzahl der Deaktivierungen mit denen der Aktivierungen übereinstimmt, werden die LEDs deaktiviert.

Nach Ansteuern der LEDs werden die Warnungen analysiert, ob sie Teil eines wiederholten oder mehrstufigen Angriffes sind. Ein wiederholter Angriff ist hierbei, wie in [VVKK04 Kapitel "Attack Thread Reconstruction"] beschrieben mehrere Angriffe, die vom gleichen Angreifer ausgehend das gleiche Ziel angreifen.

Ein mehrstufiger Angriff, wie z.B. ein APT, besteht aus unterschiedlichen Angriffen, wobei Angriffe in den ersten Stufen genutzt werden, um die späteren Angriffe vorzubereiten. Ein Beispiel hierfür ist der dritte Use Case, in dem sich der Angreifer mit Hilfe von Teamviewer Zugriff auf ein System verschafft, um dort Schutzmechanismen zu deaktivieren und es dadurch verwundbar für einen SYN-Flood Angriff zu machen. Um Angriffe als Teil eines mehrstufigen Angriffes zu erkennen, wird der Wert des Angriffes aus der Warnung genutzt.

Dabei gilt, je höher der Wert, desto später in einem mehrstufigen Angriff wird der spezifische Angriff wahrscheinlich verwendet.

Je nach Ergebnis dieser Auswertung versendet der Warner nun eine andere Mail an den Nutzer, um ihn über die Vorfälle zu informieren. Die Mails beinhalten Informationen darüber, ob der Angriff wiederholt / mehrstufig / alleinstehend ist, welche Quelle und Ziel des Angriffes sind und eine kurze Beschreibung des Angriffes.

4.2.6 E-Mailer

Der E-Mailer wird vom Warner verwendet um Warnungen an den Nutzer zu verschicken. Hierfür wird JavaMail, eine Bibliothek die das Versenden von Emails in Java ermöglicht, verwendet. [vgl. Orac14] Zum Versenden der E-Mails wird ein für diese Arbeit eingerichtetes Gmail-Konto verwendet.

Zuerst werden die Daten von Gmails SMTP-Server eingestellt und anschließend die Mail versendet.

Da der Mailer oftmals beim ersten Versenden einer Mail von Googles SMTP-Server abgelehnt wird, wird 5 Mal versucht die Mail zu versenden. Dies ist beim Versenden von insgesamt mehreren 100 Warnungen nicht einmal fehlgeschlagen, einer der 5 Versuche hat also immer funktioniert.

Das Problem hat mit dem SSL-Handshake zu tun, konnte aber nicht gelöst werden.

```
25 Properties props = new Properties();
26 props.put("mail.smtp.auth", "true");
27 props.put("mail.smtp.host", "smtp.gmail.com");
28 props.put("mail.smtp.ssl.enable", "true");
29 props.put("mail.smtp.port", "465");
30 props.put("mail.smtp.ssl.trust", "*");
31
32 Session session = Session.getInstance(props, new javax.mail.Authenticator() {
33     protected PasswordAuthentication getPasswordAuthentication() {
34         return new PasswordAuthentication(username, password);
35     }
36 });
37
38 Message message = new MimeMessage(session);
39
40 int count = 0;
41 int maxTries = 5;
42
43 while(true) {
44     try {
45         message.setFrom(new InternetAddress("jomartensbsctestmail@gmail.com"));
46         message.setRecipients(Message.RecipientType.TO, InternetAddress.parse(address));
47         message.setSubject("Network-Monitor-Warning");
48         message.setText(text);
49         Transport.send(message);
50         break;
51     } catch (MessagingException e) {
52         if (++count == maxTries) throw e;
```

Abbildung 24 : Code des Mailers

4.2.7 Wireshark Anbindung

Die Wireshark-Anbindung ermöglicht es, die vom Sniffer erfassten Pakete in Wireshark zu analysieren. Wireshark sollte hierfür auf einem Linux-Rechner, der den Netzwerkmonitor über das Netzwerk erreichen kann, installiert sein. Linux wird benötigt, da die Wireshark Anbindung zur Kommunikation mit Wireshark eine Named Pipe durch Verwendung von Befehlen der Linux-Shell erstellt. Auch ist Erstellung und Nutzung einer Named Pipe deutlich einfacher in Linux als in Windows. Der Netzwerkmonitor muss über das Netzwerk erreichbar sein, um die Messagequeue, mit der sich die Wireshark Anbindung, wie andere Komponenten, die Pakete analysieren, verbindet, erreichen zu können.

Die Netzwerkadresse des Monitors wird in einer Konfigurationsdatei der Anbindung mitgeteilt.

```
//Create Named Pipe
ArrayList<String> command = new ArrayList<String>();
command.add("mkfifo");
command.add(fifoLocation);
new ProcessBuilder(command).start();
out = new BufferedOutputStream(new FileOutputStream(fifoLocation));

//Initialise ZeroMQ
ZMQ.Context context = ZMQ.context(1);
ZMQ.Socket subscriber = context.socket(ZMQ.SUB);
subscriber.connect(monitorAddress);
byte[] empty = new byte[0];
subscriber.subscribe(empty);

//Global Header of libPcap format
ByteBuffer b = ByteBuffer.allocate(24); // 24 bytes
b.order(ByteOrder.nativeOrder());
b.clear();
b.putInt(0xa1b2c3d4); // magic_number
b.putShort((short) 2); // major version number
b.putShort((short) 4); // minor version number
b.putInt(TimeZone.getDefault().getRawOffset() / 1000); // timezone in seconds
b.putInt(0); // accuracy of timestamps
b.putInt(64 * 1024); // snaplen 64K
b.putInt(1); // data link type ethernet
b.flip();

//Write Global Header to Named Pipe
out.write(b.array());
out.flush();

//Write Packets to named Pipe
while (!this.isInterrupted()) {
    out.write(subscriber.recv());
    out.flush();
}
```

Abbildung 25 : Teil des Codes der Wireshark Anbindung

Wireshark erwartet, dass Daten im LibPcap Format auf der Named Pipe liegen, die einzelnen Pakete aus der Messagequeue befinden sich bereits in diesem Format. [vgl. Wire15b]
Es fehlt aber noch der „global header“, der sich zu Anfang jeder LibPcap-Datei befindet. Dieser wird von der Anbindung als erstes in die Named Pipe geschrieben.
Anschließend wird solange, bis die Anwendung beendet wird, jedes Paket in die Named Pipe geschickt und somit Wireshark zur Verfügung gestellt.

4.2.8 Statusseite

Die Statusseite ermöglicht es Nutzern auf einen Blick den aktuellen Zustand des Netzwerkmonitors herauszufinden. Sie ist in PHP geschrieben und wird auf einem Apache Webserver, der auf dem Netzwerkmonitor installiert ist, ausgeführt.

Wie auf Abbildung 5 zu sehen ist liefert die Statusseite Informationen über den Zustand der verschiedenen Komponenten und ob eine Warnung oder Alarm ausgelöst wurde.

Angaben zum Zustand der Komponenten erhält die Statusseite vom systemd System und Service Manager. Informationen über Alarm und Warnung erhält sie, indem der Status der einzelnen LEDs ausgelesen wird. Es ist zwar möglich in PHP auf die SUB-PUB ZeroMQ Messagequeue zuzugreifen, doch bietet dies keinen Mehrwert für den Nutzer, kann er doch die Pakete der SUB-PUB Messagequeue zwischen Sniffer und Auswertern bereits in Wireshark betrachten.

```
<?php

echo "Sniffer: ", check_process_status("start-sniffer"), "<br>";
echo "Warner: ", check_process_status("warner"), "<br>";
echo "Synflood Detector: ", check_process_status("synflood-detector"), "<br>";
echo "Teamviewer Detector: ", check_process_status("teamviewer-detector"), "<br>";
echo "<br>";
echo "Alert: ", check_led_status("red"), "<br>";
echo "Warning: " , check_led_status("yellow"), "<br>";

//Get Process state from systemd
function check_process_status($systemd_name) {
    $command = "systemctl status " . $systemd_name;
    if (strpos(shell_exec($command),"dead")) {
        return "dead";
    } else {
        return "alive";
    }
}

//Get status of led
function check_led_status($color) {
    $status_file = "/sys/devices/user_leds.7/leds/user_led_" . $color . "/brightness";
    if (strcmp(trim(file_get_contents($status_file)),"1") == 0 ) {
        return "yes";
    } else {
        return "no";
    }
}
}
```

Abbildung 26 : Code der Statusseite

5 Einsatz und Evaluation des Konzeptes

Im Rahmen dieses Kapitels werden zuerst die Use Cases vorgestellt. Anschließend werden die weiteren Funktionen des Netzwerkmonitors erklärt. In einem weiteren Unterkapitel wird die Auswirkung, die der Netzwerkmonitor auf die Geschwindigkeit des Netzwerkes hat, betrachtet. Als letztes wird die Funktionalität des Monitors überprüft.

5.1 UseCase 1 : SYN-Flood Detektion

Der erste Use Case handelt von einem Nutzer, der in seiner Freizeit gerne Online-Computerspiele spielt. In vielen Onlinespielen wird Peer-2-Peer-Technologie verwendet, um die einzelnen Mitspieler miteinander zu verbinden. Dadurch können die Mitspieler des Nutzers durch Betrachtung ihres Netzwerkverkehrs, z.B. mit Wireshark die IP-Adresse des Nutzers herausfinden. Mithilfe der IP-Adresse ist ein Mitspieler, der verhindern möchte, dass der Nutzer das Spiel gewinnt, in der Lage einen SYN-Flood Angriff auf den Nutzer zu starten. Hier kann nun der Netzwerkmonitor dem Nutzer helfen, indem der Nutzer vom Monitor nach Beginn des Angriffes eine Warnungs-Email mit u.a. der IP-Adresse des Angreifers erhält. Mithilfe dieser Adresse kann der Nutzer sich an den Betreiber des Spiels wenden und den Angreifer vom Spiel ausschließen lassen. Weiterhin könnte der Nutzer sein Modem, in der Hoffnung dadurch eine neue IP-Adresse von seinem Internet Service Provider zugeteilt zu bekommen, neu starten.

5.2 UseCase 2 : Unerlaubter Teamviewer-Zugriff

Der zweite Use Case beschreibt einen Nutzer, der sich nicht sehr gut mit Computern auskennt und sich deshalb bei auftretenden Problemen von einem Bekannten/Enkel/Kind helfen lässt. Diese „Fachleute“ haben aber nicht immer Zeit das Problem vor Ort zu lösen und nutzen deshalb die Teamviewer-Fernwartungs-Software, um das Problem zu bearbeiten. So hat also der Nutzer die Teamviewer-Software auf seinem Rechner installiert, und ermöglicht dadurch auch einem Angreifer Zugriff auf seinen Rechner. Hier hilft der Netzwerkmonitor, indem er dem Nutzer durch eine E-Mail und Aufleuchten einer LED mitteilt, dass gerade ein

Fernwartungs-Zugriff auf seinen Rechner stattfindet, obwohl der Nutzer dies gar nicht erlaubt hat. Auf diese Warnung kann der Nutzer z.B. mit dem Herunterfahren des Rechners reagieren und den Angriff so unterbinden.

5.3 UseCase 3 : Kombiniertes Angriff

Der dritte Use Case handelt von einem Nutzer der eine Webseite auf seinem eigenen Server betreibt und diesen Server durch einen externen IT-Dienstleister, durch Nutzung von Teamviewer warten lässt. Der Server verfügt über einen Schutzmechanismus der einen SYN-Flood Angriff verhindern kann. Nun verschafft sich der Angreifer die Teamviewer-Zugangsdaten für den Server, entweder über Social Engineering oder Brute Force. Durch Teamviewer greift der Angreifer nun auf den Server zu und deaktiviert den SYN-Flood Schutz des Servers um diesen anschließend mit SYN-Flood Angriffen außer Betrieb setzen zu können. Der Netzwerkmonitor erkennt aber das Muster unerlaubter Teamviewer Zugriff-> SYN-Flood-Angriff und warnt den Nutzer durch E-Mails und Aufleuchten der LEDs. So kann der Nutzer schnell seinen IT-Dienstleister über den Angriff informieren und so einen längeren Ausfall des Servers verhindern.

5.4 Weitere Funktionen

Der Netzwerkmonitor verfügt über zwei weitere Funktionen, die nicht im Zusammenhang mit den beiden Use Cases stehen.

Die Erste ist die Möglichkeit, die von der Sniffer-Komponente ausgelesenen Netzwerkpakete, in Wireshark, „the world’s foremost network protocol analyzer“ [Wire16], zu analysieren. Als zweite Funktion verfügt der Netzwerkmonitor außerdem über eine Statusseite, auf der der aktuelle Status der einzelnen Komponenten eingesehen werden kann.

5.4.1 Wireshark

Die Wireshark-Anbindung ermöglicht es Nutzern, die vom Netzwerkmonitor erfassten Pakete zu betrachten, bevor eine Analyse durch den Monitor erfolgt ist.

Dies ermöglicht eine tiefere Analyse des Netzwerkverkehrs, wie zum Beispiel die Entdeckung von Angriffen, die der Monitor nicht entdecken kann.

Dies kann entweder durch die Betrachtung der Pakete in Wireshark geschehen oder durch den Export der erfassten Paketdaten in viele verschiedene Dateiformate, welche von verschiedensten anderen Intrusion Detection Tools gelesen und analysiert werden können. [vgl. Wire16]

5.4.2 Statusseite

Die Statusseite bietet den Nutzern einen Überblick über den aktuellen Zustand des Netzwerkmonitors. Es wird der aktuelle Zustand der einzelnen Komponenten dargestellt. Dies ermöglicht den Nutzern auf einen Blick festzustellen ob einzelne Komponenten normal laufen oder abgestürzt sind. Auch wird die aktuelle Situation, ob also eine Warnung oder ein Alarm ausgelöst wurde, dargestellt.

So ist es Nutzern möglich sich einen Überblick über den Status des Netzwerkmonitors zu verschaffen, ohne die Hardware des Monitors in Sichtweite zu haben.

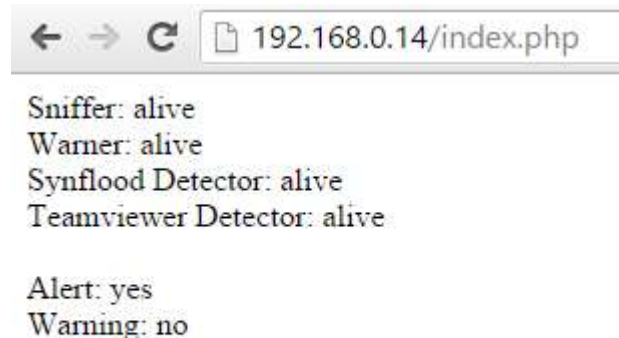


Abbildung 27 : Statusseite während ein Alarm ausgelöst wurde

5.5 Performance der Netzwerkbrücke

Die Performance des vom Netzwerkmonitors überwachten Netzwerkes wird durch die Überwachung nicht beeinflusst.

Zum einen hat der Autor der Arbeit den Netzwerkmonitor bereits seit mehreren Wochen im dauerhaften Einsatz und lässt sein gesamtes Netzwerk vom Monitor überwachen. Dabei sind weder beim Spielen von Onlinespielen, noch beim Anschauen von HD-Videos auf Youtube.com und Netflix.com Beeinträchtigungen aufgefallen.

Weiterhin wurde die Übertragungsgeschwindigkeit im lokalen Netzwerk getestet. Hierfür wurde eine 2GB große Datei übertragen. Einmal über den Netzwerkmonitor, dessen Netzwerkkarten über eine Datenrate von 100 Mbit/s verfügen und einmal über einen Switch dessen maximale Datenrate 100 Mbit/s beträgt. Dies ist notwendig, da die beiden Rechner, zwischen denen die Datei übertragen wurde jeweils über Netzwerkkarten mit einer maximalen Datenrate von 1 Gbit/s verfügen. Durch Einsatz des 100Mbit/s Switches verfügen aber sowohl der Netzwerkmonitor, als auch der Vergleichstest über die gleiche Maximalgeschwindigkeit.

Das Ergebnis ist, das bei beiden Versuchen die durchschnittliche Übertragungsrate bei 75 Mbit/s lag.

Um auszuschließen, dass die Übertragungen nicht durch andere Auswirkungen, wie z.B. eine langsame Festplatte limitiert wurden, wurde die Übertragung anschließend noch einmal ohne den 100 Mbit/s Switch durchgeführt. Hierbei betrug die durchschnittliche Übertragungsrate 420 Mbit/s. Also wurden die Versuche nicht von einer langsamen Festplatte oder anderen Problemen beeinflusst.

5.6 Funktionalität

Der Netzwerkmonitor erfüllt seine Aufgabe, das Warnen des Nutzers, wenn ein Angriff auftritt.

Dies wurde getestet indem ein Rechner im lokalen Netzwerk vom Monitor überwacht wurde, während simulierte Angriffe von einem anderen Rechner innerhalb des lokalen Netzwerkes ausgelöst wurden. Zur Simulation eines SYN-Flood Angriffes wurde Hping3 mit folgenden Einstellungen genutzt:

```
Sudo hping3 -S -i u10000 192.168.0.11
```

-S = Nur SYN-Pakete

-i u10000 = Sende ein Paket alle 10000 Mikrosekunden, also 100 Pakete pro Sekunde

192.168.0.11 = Die Adresse des Rechners, der vom Monitor überwacht wird

Dies führt innerhalb einer Minute dazu, dass vom Netzwerkmonitor ein SYN-Flood Angriff erkannt wird. Aufgrund des gewählten Algorithmus zur Entdeckung von SYN-Flood Angriffen kann es allerdings vorkommen, dass erst mehrere Minuten nach Ende des Angriffes, diese Beendigung auch vom Monitor erkannt wird.

Leider führt das Senden von mehr Paketen pro Sekunde dazu, dass die Funktionalität des Netzwerkmonitors beeinträchtigt wird. Ursache hierfür ist die geringe Leistung des Netzwerkmonitors und die geringe Performance von Java als Programmiersprache. Es wird also der Netzwerkmonitor Opfer eines DOS-Angriffes, den er eigentlich entdecken sollte.

Während eines SYN-Flood Angriffes ist die Netzwerkbrücke des Monitors mit der Übertragung der Pakete so ausgelastet, dass der Apache Webserver nicht mehr zu erreichen ist. Auch ist der Sniffer nicht in der Lage Pakete von der Brücke auszulesen, da ihm keine Rechenleistung zur Verfügung steht. Auch ein SSH-Zugriff auf den Netzwerkmonitor ist während des DOS-Angriffes nicht möglich.

Erst nach Ende des Angriffes ist der Sniffer wieder in der Lage Pakete auszulesen. Erst zu diesem Zeitpunkt findet eine Entdeckung des Angriffes statt, da sich einige der SYN-Flood Pakete noch im „Capture Buffer“ des Betriebssystems befinden.

Andere Angriffe, die nicht den ganzen Netzwerkmonitor lahmlegen, wie z.B. der unerlaubte Teamviewer Zugriff aus dem zweiten Use Case erkennt der Monitor aber sicher.

Dies wurde mit dem Aufbau aus dem SYN-Flood Versuch überprüft, nur das dieses Mal der nicht überwachte Rechner eine Teamviewer-Verbindung zum überwachten Rechner aufbaut. Der Teamviewer Detektor ist hierbei durch die Konfigurationsdatei so eingestellt worden, dass der aktuelle Zeitpunkt, außerhalb des Zeitraumes liegt, in dem Nutzung des Teamviewers keinen Alarm auslöst. In unter einer Minute erkennt der Netzwerkmonitor den unerlaubten Teamviewerzugriff und meldet ihn mit Hilfe von E-Mail und LED.

Diese beiden Versuche wurden jeweils nach einem Neustart des Netzwerkmonitors durchgeführt. Wenn sich der Monitor bereits über einen längeren Zeitraum im Betrieb befindet, kann sich die Entdeckung von Angriffen um einige Minuten verlängern, je nachdem wie viele Pakete sich bereits aufgestaut haben.

Auch das Erkennen eines mehrstufigen Angriffes, wie er im dritten Use Case dargestellt wird, wurde überprüft, indem die ersten beiden Versuche hintereinander durchgeführt wurden.

Hierbei ist zu beobachten, dass der erste Angriff über Teamviewer noch innerhalb einer Minute erkannt wird, sich die Entdeckung des zweiten Angriffes, dem SYN-Flood Angriff, aber um bis zu 5 Minuten verzögern kann. Ursache für die Verzögerung ist zum einen, dass sich aufgrund der geringen Leistung des Monitors bereits Pakete, die noch zum ersten Angriff gehören, im Capture Buffer aufgestaut haben und zuerst abgearbeitet werden müssen.

Weitere Ursache ist, dass je nach Stärke des SYN-Flood Angriffes, die Übertragung der Pakete durch die Netzwerkbrücke bereits einen Großteil der Kapazitäten des Kleinstrechners beansprucht, und dadurch die Bearbeitung der Pakete durch Sniffer und Analysekomponenten verlangsamt.

Die Performanceprobleme werden deutlich in Abbildung 25 dargestellt.

```
Mem: 171452K used, 76008K free, 0K shrd, 5472K buff, 62008K cached
CPU: 94% usr 3% sys 0% nic 0% idle 0% io 0% irq 2% sirq
Load average: 2.37 2.15 1.32 3/208 2465
```

PID	PPID	USER	STAT	VSZ	%VSZ	%CPU	COMMAND
1860	1858	root	S	176m	73%	47%	java -Djava.library.path=/usr/lib:/usr/local/lib -jar /usr/bin/jars/teamviewer.jar
1856	1852	root	S	191m	73%	44%	java -Djava.library.path=/usr/lib:/usr/local/lib -jar /usr/bin/jars/synflood.jar
2429	1	root	S	5624	2%	2%	sshd: root@pts/0
2465	2463	root	R	2924	1%	2%	top
1854	1850	root	S	177m	73%	1%	java -Djava.library.path=/usr/lib:/usr/local/lib -jar /usr/bin/jars/sniffer.jar
1864	1859	root	S	176m	73%	1%	java -Djava.library.path=/usr/lib:/usr/local/lib -jar /usr/bin/jars/warner.jar
9	2	root	SM	0	0%	1%	[rcu_sched]
2428	2	root	SM	0	0%	1%	[kworker/0:0]
1890	1886	daemon	S	235m	97%	0%	/usr/local/apache2/bin/httpd -k start
1893	1886	daemon	S	235m	97%	0%	/usr/local/apache2/bin/httpd -k start
1894	1886	daemon	S	235m	97%	0%	/usr/local/apache2/bin/httpd -k start

Abbildung 28 : Auslastung Kleinstrechner

Diese Abbildung enthält einen Auszug über den Zustand des Kleinstrechners, welcher durch den „top“-Befehl auf einer Linux-Shell erstellt wurde. Es wird deutlich, dass die beiden Analysekomponenten den Kleinstrechner komplett auslasten. Der Sniffer ist nur in der Lage weitere Pakete auszulesen, wenn die Analysekomponenten keine Pakete mehr verarbeiten. Dies führt dazu, dass evtl. nicht alle Pakete vom Sniffer erfasst werden können, da der Capture Buffer des Betriebssystems nur eine begrenzte Größe hat. Wenn der Buffer gefüllt ist werden weitere Pakete nicht dem Buffer hinzugefügt, sondern ignoriert.

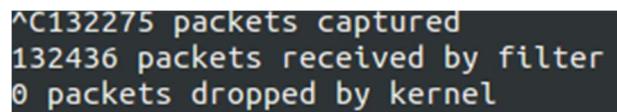
Versuchsweise wurde die Priorität des Sniffer-Prozesses erhöht. Dies führte dazu, dass zwar keine Pakete mehr verloren gingen, allerdings wurden auch keine Warnungen mehr ausgelöst, da die einzelnen Analysekomponenten kaum noch Rechenzeit zur Verfügung hatten. Die CPU wurde nämlich bereits unter normaler Auslastung der Netzwerkverbindung durch z.B. das Anschauen eines Youtube-Videos zu einem Großteil der Zeit durch den Sniffer ausgelastet.

Um die Leistung des Kleinstrechners zu überprüfen wurden die Java-Komponenten des Netzwerkmonitors deaktiviert und Netzwerkpakete mithilfe von TCPDump erfasst. [vgl. Tcpcd15a] Hierfür wurde TCPDump mit folgenden Befehlen ausgeführt:

```
tcpdump -i br0 -w /dev/null
```

-i br0 = das Netzwerkinterface auf dem TCPDump sniffen soll, hier die Brücke br0
-w /dev/null = Schreibe Paketdaten nach /dev/null

Während TCPDump nun mit diesen Einstellungen den Netzwerkverkehr überwacht, wurde ein Youtube-Video mit einer Länge von 2 Minuten angeschaut. Das Ergebnis ist in Abbildung 26 zu sehen.



```
^C132275 packets captured
132436 packets received by filter
0 packets dropped by kernel
```

Abbildung 29 : TCPDump-Daten nach Sniffing während eines Youtube-Videos

Pakete, die nicht von TCPDump erfasst wurden sind die „packets dropped by kernel“. Hierdurch wird deutlich, dass die Leistung des Kleinstrechners zum Erfassen der Pakete ausreicht. Selbst wenn unter Verwendung von hping3 und dem --flood Parameter ein SYN-Flood Angriff mit vielen Hunderttausend Paketen pro Sekunde ausgeführt wird, verliert TCPDump keine Pakete. Also lassen sich Probleme mit der Performance zu den Java-Komponenten zurückverfolgen.

Die Leistung der PUB-SUB Messagequeue wurde mithilfe der Wireshark-Anbindung getestet. Hierfür wurden die Analyse-Komponenten des Netzwerkmonitors deaktiviert. Um zu erkennen ob alle Pakete die von Sniffer erfasst wurden auch bei der Wireshark-Anbindung ankommen, wurde ein dauerhafter Ping mit folgendem Befehl gestartet:

```
Ping -t -4 google.de
```

-t = Kein Limit für die Anzahl an Pings
-4 = Nutze Ipv4
Google.de = Adresse die gepingt wird

Da der Ping Befehl die versendeten Pakete mit einer ansteigenden Nummer versieht, ist leicht zu erkennen wenn Pakete nicht von der Messagequeue übertragen wurden. Hierbei gehen die ersten Ping-Pakete verloren, wenn mit hping3 10000 SYN-Pakete pro Sekunde an einen vom Monitor überwachten Rechner verschickt werden. Gleichzeitig ist in der Ausgabe des „Top“-Befehles zu erkennen, dass der Sniffer-Prozess über den gesamten Zeitraum die CPU zu mehr als 95% auslastet. Hierbei bildet aber nicht die Messagequeue, sondern der Aufwand den der Sniffer betreiben muss, um ein Paket von der Brücke auszulesen und in das LibPcap-Format umzuwandeln, die Engstelle.

So braucht ZeroMQ 21 Sekunden um einen simplen „Hello“-String eine Million Mal über eine PUB-SUB Messagequeue zu übertragen, wobei der Publisher, wie der Sniffer auf dem Kleinstrechner und der Subscriber, wie die Wireshark-Anbindung auf einem anderen Rechner im Netzwerk liefen.

6 Fazit

Im Rahmen des Fazits wird zuerst das Konzept bewertet. Es wird aufgezeigt welche Ansätze gut funktioniert haben, und an welchen Stellen noch Möglichkeiten zur Verbesserung existieren.

Anschließend wird dargestellt wie der Netzwerkmonitor weiterentwickelt werden könnte.

6.1 Bewertung des Konzeptes

Probleme lassen sich hauptsächlich in der Implementation der Architektur finden, wie in Kapitel 5.6. Funktionalität deutlich wird. So lässt sich der Netzwerkmonitor zwar leicht um weitere Funktionen erweitern, dank der beiden Messagequeues, ist jedoch bereits durch die bereits vorhandenen Komponenten ausgelastet. Größter Faktor ist hier die Zeit, die die einzelnen Komponenten benötigen um die einzelnen Pakete zu bearbeiten, daran zu erkennen, das der Sniffer kaum 10000 Pakete in der Sekunde verarbeiten kann, die Messagequeue aber ca. 50000 Nachrichten pro Sekunde verschicken kann.

Auch vergeht einige Zeit, ungefähr eine Minute unter idealen Gegebenheiten, zwischen Beginn eines Angriffes und dessen Entdeckung. Dieser Zeitraum ist bei länger andauernden Angriffen, wie einem SYN-Flood Angriff vielleicht nicht so wichtig, könnte bei einem unerlaubten Teamviewer-Zugriff aber bedeuten, dass wichtige Daten bereits entwendet wurden, bevor der Angriff vom Monitor bemerkt wird.

Beide Probleme haben ihre Ursache in der Implementierung der Architektur, nicht der Architektur selbst, ließen sich also durch Verwendung eines schnelleren Kleinstrechners oder effizienter programmierten Komponenten beheben.

6.2 Ausblick / Mögliche Weiterentwicklung

Der Netzwerkmonitor kann in vielen Bereichen weiterentwickelt werden. Eine Erweiterung ist die Implementation weiterer Analysekomponenten, um weitere Angriffe zu erkennen. Diese neuen Komponenten lassen sich aufgrund der beiden Messagequeues leicht in das bereits vorhandene System integrieren, muss die neue Analysekomponente doch nur die Anbindungen an die beiden Messagequeues implementieren, was aufgrund der Verfügbarkeit von ZeroMQ in ca. 50 Sprachen leicht möglich sein sollte.

Eine weitere mögliche Weiterentwicklung ist die Vernetzung mehrerer Netzwerkmonitore um z.B. ein größeres Netzwerk zu überwachen. Jeder Monitor könnte hierbei die anderen Monitore über einen entdeckten Angriff informieren. Dies könnte über die Messagequeues des Warners geschehen, sodass Meldungen anderer Netzwerkmonitore genau wie Warnungen einer lokalen Analysekomponente verarbeitet werden. Durch Verwendung vorhandener Ressourcen muss nur noch ein Mechanismus implementiert werden mit dem die jeweiligen Netzwerkmonitore die Messagequeues der anderen Monitore herausfinden können. Dies könnte zum Beispiel mit einem zentralen Messagebroker geschehen.

Eine dritte Erweiterungsmöglichkeit ist die Entwicklung komplexerer Analysekomponenten, die mehr Rechenleistung benötigen, als ein Kleinstrechner zur Verfügung stellen kann. Genau wie die Wireshark-Anbindung kann diese komplexere Software auf einem leistungsstärkeren Desktop-Rechner oder einem Server laufen, und trotzdem Zugriff auf die vom Netzwerkmonitor erfassten Paketdaten erhalten.

7 Anhang

7.1 Literaturverzeichnis

- [Apac16] THE APACHE SOFTWARE FOUNDATION: *About the Apache HTTP Server Project - The Apache HTTP Server Project.* URL https://httpd.apache.org/ABOUT_APACHE.html. - abgerufen am 2016-01-12
- [Bedn15] BEDNARCZYK, MARK: *jNetPcap v2 discussion | jNetPcap OpenSource.* URL <http://jnetpcap.com/?q=node/1314>. - abgerufen am 2016-02-12
- [Bund15a] BUNDESAMT FÜR SICHERHEIT IN DER INFORMATIONSTECHNIK: *Die Lage der IT-Sicherheit in Deutschland 2015*, Bundesamt für Sicherheit in der Informationstechnik (BSI) (2015)
- [Bund15b] BUNDESMINISTERIUM DER JUSTIZ UND FÜR VERBRAUCHERSCHUTZ: *StGB - Strafgesetzbuch.* URL <http://www.gesetze-im-internet.de/stgb/BJNR001270871.html>. - abgerufen am 2016-01-07
- [Char13] CHARLES, P.: *Network Packet Capture Facility for Java download | SourceForge.net.* URL <https://sourceforge.net/projects/jpcap/>. - abgerufen am 2016-02-12
- [Cisc16] CISCOPRICE: *7010 GPL Price.* URL <http://ciscoprize.com/gpl/FP7010-BUN>. - abgerufen am 2016-02-03
- [Free15] FREEDESKTOP.ORG: *systemd.special.* URL <https://www.freedesktop.org/software/systemd/man/systemd.special.html>. - abgerufen am 2016-02-17
- [Hint13] HINTJENS, PIETER: *ZeroMQ: Messaging for Many Applications*: O'Reilly Media, Inc., 2013
- [Imat14] IMATIX CORPORATION: *ØMQ Language Bindings - zeromq.* URL http://zeromq.org/bindings:_start. - abgerufen am 2016-02-12

- [Isik15] ISIKDAG, UMIT: *Enhanced Building Information Models, SpringerBriefs in Computer Science*. Cham : Springer International Publishing, 2015 — ISBN 978-3-319-21824-3
- [Jaco09] JACOBSEN, HANS-ARNO: Publish/Subscribe. In: *Encyclopedia of Database Systems* (2009), S. 2208–2211
- [Lipi16] LIPINSKI, KLAUS: *Einplatinen-Computer :: single board computer :: SBC :: ITWissen.info*. URL <http://www.itwissen.info/definition/lexikon/Einplatinen-Computer-SBC-single-board-computer.html>. - abgerufen am 2016-01-27
- [Netc15] NETCRAFT: *December 2015 Web Server Survey | Netcraft*. URL <http://news.netcraft.com/archives/2015/12/31/december-2015-web-server-survey.html#more-22337>. - abgerufen am 2016-01-12
- [Orac14] ORACLE: *JavaMail API Reference Implementation: Wiki: Home — Project Kenai*. URL <https://java.net/projects/javamail/pages/Home>. - abgerufen am 2016-02-24
- [Phyt16a] PHYTEC MESSTECHNIK GMBH: *phyBOARD-Wega Single Board Computer - TI AM335x | PHYTEC*. URL <http://www.phytec.de/de/produkt/single-board-computer/phyboard-wega/>. - abgerufen am 2016-02-11
- [Phyt16b] PHYTEC MESSTECHNIK GMBH: *Yocto Reference Manual* (2016)
- [ScMe07] SCARFONE, KAREN ; MELL, PETER: Guide to intrusion detection and prevention systems (idps). In: *NIST special publication* Bd. 800 (2007), Nr. 2007, S. 94
- [Slyt14] SLY TECHNOLOGIES: *jNetPcap OpenSource | Protocol Analysis SDK*. URL <http://jnetpcap.com/>. - abgerufen am 2016-02-12
- [TaWe12] TANENBAUM, ANDREW S. ; WETHERALL, DAVID J.: *Computernetzwerke*. 13. Aufl. München : Pearson Deutschland GmbH, 2012 — ISBN 978-3-86894-137-1
- [Tcpd15a] TCPDUMP/LIBPCAP: *TCPDUMP/LIBPCAP public repository*. URL <http://www.tcpdump.org/>. - abgerufen am 2016-02-12
- [Tcpd15b] TCPDUMP/LIBPCAP: *Manpage of PCAP*. URL <http://www.tcpdump.org/manpages/pcap.3pcap.html>. - abgerufen am 2016-02-17
- [Trea14] TREAT, TYLER: *Dissecting Message Queues — Brave New Geek*. URL <http://bravenewgeek.com/dissecting-message-queues/>. - abgerufen am 2016-02-29

- [VVKK04] VALEUR, FREDRIK ; VIGNA, GIOVANNI ; KRUEGEL, CHRISTOPHER ; KEMMERER, RICHARD ; OTHERS: Comprehensive approach to intrusion detection alert correlation. In: *Dependable and Secure Computing, IEEE Transactions on* Bd. 1 (2004), Nr. 3, S. 146–169
- [WaZS02] WANG, HAINING ; ZHANG, DANLU ; SHIN, KANG G.: Detecting SYN flooding attacks. In: *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. Bd. 3 : IEEE, 2002, S. 1530–1539
- [Wire08] WIRESHARK FOUNDATION: *libpcap - The Wireshark Wiki*. URL <https://wiki.wireshark.org/libpcap>. - abgerufen am 2016-02-12
- [Wire15a] WIRESHARK FOUNDATION: *Development/LibpcapFileFormat - The Wireshark Wiki*. URL <https://wiki.wireshark.org/Development/LibpcapFileFormat>. - abgerufen am 2016-01-27
- [Wire15b] WIRESHARK FOUNDATION: *CaptureSetup/Pipes - The Wireshark Wiki*. URL https://wiki.wireshark.org/CaptureSetup/Pipes#Named_pipes. - abgerufen am 2016-02-03
- [Wire16] WIRESHARK FOUNDATION: *Wireshark · Go Deep*. URL <https://www.wireshark.org/>. - abgerufen am 2016-01-12
- [Yoct16] YOCTO PROJECT: *About | Yocto Project*. URL <https://www.yoctoproject.org/about>. - abgerufen am 2016-02-12

7.2 Abbildungsverzeichnis

Abbildung 1 : Vorgehensweise bei einem APT-Angriff [Bund15a S.26 Abbildung 6]	7
Abbildung 2 : Beispielhafte Platzierung des Netzwerkmonitors in einem Netzwerk	8
Abbildung 3 : Bereich 1	14
Abbildung 4 : Bereich 2	14
Abbildung 5 : Bereich 3	15
Abbildung 6 : LibPcap-Datenformat [Wire15a].....	16
Abbildung 7 : Format einer Nachricht der zweiten Messagequeue	17
Abbildung 8 : Architektur des Netzwerkmonitors	18
Abbildung 9 : Phyttec phyBOARD®-Wega Lieferumfang.....	19
Abbildung 10 : web server developers: Market share of active sites [Netc15]	22
Abbildung 11 : Überblick über die Komponenten des Netzwerkmonitors.....	24
Abbildung 12 : .service-Datei zum Einrichten eines neuen systemd-Services.....	26
Abbildung 13 : Erstellung des Brücken-Netzwerk-Interfaces	27
Abbildung 14 : Konfiguration des Brücken-Netzwerk-Interfaces	27
Abbildung 15 : Einrichtung einer der beiden Netzwerkinterfaces.....	27
Abbildung 16 : Shellskript zur Steuerung der LEDs	28
Abbildung 17 : Starter des Sniffers	29
Abbildung 18 : Schleifen des Sniffers.....	31
Abbildung 19 : Teil des Codes des Paketverteilers	32
Abbildung 20 : Auf-/Abbau einer TCP-Verbindung [WaZS02 Fig. 1]	33
Abbildung 21 : Ablauf SYN-Flood-Detektor.....	35
Abbildung 22 : Berechnung des Entscheidungswertes.....	35
Abbildung 23 : Format der Warnungen	37
Abbildung 24 : Code des Mailers	38
Abbildung 25 : Teil des Codes der Wireshark Anbindung.....	39
Abbildung 26 : Code der Statusseite.....	41
Abbildung 27 : Statusseite während ein Alarm ausgelöst wurde	44
Abbildung 28 : Auslastung Kleinstrechner	46
Abbildung 29 : TCPDump-Daten nach Sniffing während eines Youtube-Videos	47

7.3 Technische Daten des Phyttec phyBOARD®-Wega

phyBOARD-Wega AM335x	
Module	phyCORE-AM3354
Operating System	Linux (Kernel 3.2)
Distribution	Yocto 1.7 Based
SOM Mounting	Soldered (DSC)
CPU	AM3354
Power	800 MHz
Memory	128 MB NAND, Flash, 256 MB DDR3 RAM, 4 kByte EEPROM
Interfaces	
CAN	1x CAN
Serial	2x RS232
Ethernet	2x Ethernet 10/100 Mbit/s
USB	1x USB Host / 1x USB-OTG
Audio	1x Stereo Line In, 1x Stereo, Speaker Line-Out
Expansion & Configuration	
Mass storage	µSD Card Holder
Expansion Bus	UART 0, SPI 0, I ² C 0, JTAG, MMC 2, UART 2, UART 3, GPIOs, Interrupt, Reset, Analog Inputs
Control elements	1x Button Reset
Controls	Goldcap for on module RTC
Display & Touch	---
Display Connector	Parallel (on A/V Connector)
Resistive Touch	4-Wire (on A/V Connector)
Capacitive Touch	I ² C (on A/V Connector)
HDMI	on HDMI-Adapter
Supply Voltage	5 V
Temperature Range	0°C to 70°C
Dimensions SBC	100 x 72 mm (Pico-ITX format)

[Phyt16a]

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den _____