HAW HAMBURG

BACHELORTHESIS
Michael Deichen

Konzept zur Erkennung der Ausbreitung von Malware und internen Angriffen mittels Medium-Interaction Honeypots

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Michael Deichen

Konzept zur Erkennung der Ausbreitung von Malware und internen Angriffen mittels Medium-Interaction Honeypots

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science 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. Olaf Zukunft

Eingereicht am: 29. August 2020

Michael Deichen

Thema der Arbeit

Konzept zur Erkennung der Ausbreitung von Malware und internen Angriffen mittels Medium-Interaction Honeypots

Stichworte

Honeypot, Malware, Ausbreitung, Erkennung, Angriff, Analyse, Auswertung, Prototyp

Kurzzusammenfassung

Eine Bachelorthesis mit dem Konzept zur Erkennung der Ausbreitung von Malware und internen Angriffen mit der Planung und Realisierung eines experimentellen Prototyps in Form einer dockerisierten DMZ innerhalb einer virtuellen Maschine mit zusätzlichem Log-Server auf einer separaten Maschine zur konsistenten Aufzeichnung relevanter Daten zu definierten Ereignissen und Ermöglichung einer zeitnahen Analyse mittels einer per Internetbrowser aufrufbaren Oberfläche...

Michael Deichen

Title of Thesis

Concept for detecting the spread of malware and internal attacks using medium interaction honeypots

Keywords

Honeypot, Malware, Spread, Detection, Attack, Analysis, Evaluation, Prototype

Abstract

A bachelor's thesis with the concept of detecting the spread of malware and internal attacks with the planning and realization of an experimental prototype in the form of a dockerized DMZ within a virtual machine with additional log server on a separate machine for consistent recording of relevant data on defined events and enabling a real-time analysis via an interface accessible via internet browser...

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungen

apt Advanced Packaging Tool.

CLI Command Line Interface.

DMZ Demilitarisierte Zone.

DNS Domain Name System.

DoS Denial of Service.

ELK Elasticsearch, Logstash & Kibana.

IDS Intrusion Detection System.

IoT Internet of Things.

IP Internet Protocol.

IPS Intrusion Prevention System.

NAS Network Attached Storage.

NTP Network Time Protocol.

PAM Pluggable Authentication Module.

SMTP Simple Mail Transfer Protocol.

USV Unterbrechungsfreie Stromversorgung.

VM Virtuelle Maschine.

1 Einleitung

»Internet, das ist doch nur ein Hype.«

So soll es 1993 Bill Gates vor seinem Team bei Microsoft gesagt haben. [38]

Dieser »Hype« hat sich durchgesetzt und wächst rasant. Anfang letzten Jahres waren 22 Milliarden Geräte mit dem Internet verbunden [18] - und jede Sekunde kommen alleine an Internet of Things (IoT)-Gerätschaften durchschnittlich 127 dazu [36]. Diese Menge möchte administriert, gewartet und jede entdeckte Unsicherheit oder jeder Fehler beseitigt werden - möglichst komfortabel aus der Ferne, idealerweise noch automatisiert über standardisierte Schnittstellen.

Durch die Vielzahl verschiedener Systeme und neuer Entwicklungen wird es auch interessant Malware gegen diese zu entwickeln und einzusetzen. Die Absichten der Autoren oder Auftraggeber hinter Malware können unterschiedlich sein. Vom einfachen Wunsch nach Aufmerksamkeit des Entwicklers über Spionage und Sabotage bis hin zu gewerbsmäßiger Bandenkriminalität oder gar Cyberkriege auf politischer Ebene ist vieles denkbar - und zukünftiges noch unvorstellbar.

Doch interessiert in dieser Thesis die Abwehr beziehungsweise eine möglichst frühzeitigen Erkennung von Malware und Angriffen, welche durchaus auch von dem eigenen Personal oder anderen inneren Angreifern ausgehen kann. Den jeweiligen Motivationen der Angreifer sind keine Grenzen gesetzt.

Um Abwehrmaßnahmen ergreifen zu können, ist das Wissen der Ausbreitungswege beziehungsweise Vorgehensweisen von Angreifern sehr hilfreich. Diesen Erkenntnissen möchte ich mit dieser Thesis näher kommen.

Nebenbei sei erwähnt, dass zum besseren Verständnis hier lediglich der einfache Plural *Angreifer* Erwähnung findet - ich bitte zu beachten, dass *Angreiferinnen* damit ebenfalls gemeint sind. Gleichermaßen geht es mit den Begriffen *Autoren* zu *Autorinnen*, *Entwicklern* zu *Entwicklerinnen* und *Experten* zu *Expertinnen*.

1.1 Ziel und Aufbau der Thesis

Mit dieser Bachelorthesis verfolge ich das Ziel, die Verbreitung von Malware und Angriffe innerhalb eines Netzwerks erkennen und protokollieren zu können. Daher möchte ich hier ein Honeypot in Form eines virtuellen Netzwerks, bestehend aus Jumphost- und Webserver-Container, aufbauen. Für diesen Honeypot möchte ich Regeln definieren, welche die zu protokollierenden Events beschreiben. Aus den Logs soll ersichtlich werden, wie sich die Schadsoftware oder Angreifer vom Jumphost zum Webserver vorarbeiten oder welche Versuche unternommen werden. Dazu wird der Jumphost unzureichend abgesichert und leicht aufzufinden sein. Dies erfolgt mit offenen Ports an den Diensten *telnetd* und *sshd* sowie die Verwendung häufig verwendeter Benutzernamen und einfacher Passwörter sowohl für mehrere Benutzer als auch dem Administratorkonto *root*.

Die aufgezeichneten Herangehensweisen sollen von fachkundigem Personal analysiert werden können, um der Entwicklung neuer, oder Weiterentwicklung vorhandener, Verhaltensdefinitionen dienlich zu sein. Solche aufbereiteten Regelsätze sollen daraufhin unterstützen, die Ausbreitung gleicher oder ähnlicher Malware beziehungsweise Angriffen frühzeitig zu erkennen.

Zu Beginn der Planung eines Konzepts mit grober Architektur soll berücksichtigt werden, dass es mehrere Honeypot-Netzwerke, folgend *Honeynets* genannt, geben können soll, dessen anfallende Daten auf einen externen Logging-Server protokolliert werden. Der externe Logging-Server soll es ermöglichen, Aufzeichnungen schreibgeschützt vorzuhalten, aufzuarbeiten und externen Analysten zur Verfügung zu stellen. Die Umsetzung des Prototyps erfolgt auf Basis des Konzepts.

1.2 Abgrenzung

Primär geht es um die Konzeption und Aufbaumöglichkeiten eines Honeynets zur Protokollierung von Ausbreitungswegen der Angreifer und Malware. Ergänzend folgt die Entwicklung eines dazu passenden Prototypen, damit bei den Experimenten an diesem die Funktionalität des Konzepts demonstriert werden kann.

Es ist nicht die Absicht, eine vollständige Analysefunktion abzubilden oder Angriffe sofort zu erkennen und Gegenmaßnahmen einzuleiten.

1.3 Gliederung der Arbeit

Diese Bachelorthesis gliedert sich in eine Einleitung, verschiedenen Grundlagenerklärungen, der Beschreibung des Konzepts, einer Planung und Umsetzung eines Prototyps mit anschließenden Experimenten, einem Ausblick für weitere Entwicklungen und abschließendem Fazit.

In dem Grundlagenkapitel wird beginnend das Konstrukt des Honeypots mit wichtigen Typisierungen erklärt. Zusätzlich beinhaltet das Kapitel 2 Beschreibungen zur Virtualisierung und Containerisierung, da diese den Grundstein des Konzepts und Prototypen bilden werden.

Das anschließende Kapitel 3 beinhaltet die Beschreibung des Konzepts, woraufhin ein Prototyp im Kapitel 4 geplant und umgesetzt wird. Dazu gehören ein Komponentendiagramm sowie das Aufzeigen und Erklärungen wichtiger Code-Stellen und Konfigurationen.

Anschließende Experimente im Kapitel 5 belegen die Fähigkeit, Anforderungen des Konzepts erfüllen sowie erwartete Daten vom Prototypen erkennen, aufzeichnen und auswerten zu können.

Die Thesis wird letztendlich mit einem Ausblick und Fazit im Kapitel 6, der Zusammenfassung, abgeschlossen.

2 Grundlagen

In diesem Kapitel wird beschrieben, welche relevanten Technologien, Konzepte und Systeme dem Honeypot dieser Thesis zugrunde liegen.

2.1 Honeypot

Beginnend wird darauf eingegangen, worum es sich bei einem Honeypot handelt. Nach den Definitionen folgt ein Blick in den Einfluss auf die IT-Sicherheit und welche Schwierigkeiten unter anderem im Umgang bei der Anwendung von Honeypots auftreten können. Abschließend wird noch auf einige wichtige rechtliche Rahmenbedingungen hingewiesen, welche beim Betrieb von Honeypots beachtet werden müssen.

2.1.1 Eigenschaften und Typen von Honeypots

Die Eigenschaften sind von den jeweiligen Typen der Honeypots abhängig und werden in den Definitionen wiedergegeben.

Definitionen

> $ honey pot (N) A system (e.g., a web server) or system resource (e.g., a file on a server) that is designed to be attractive to potential crackers and intruders, like honey is attractive to bears. (See: entrapment.)

Lautet die formale Definition eines Honeypots in der RFC 4949 [25].

Fingierte oder gar absichtlich eingebaute Sicherheitslücken oder Schwachstellen bilden den Honig der Honeypots und sollen diesen für Bären, hier: Angriffen mittels Malware oder (internen) Angreifern, attraktiv wirken lassen. Diese Systeme unterscheiden sich

hauptsächlich in der Ausprägung möglicher Interaktion und Einsatzbereiche, welche folgend erörtert werden.

Low-Interaction womit keine oder nur sehr wenig Interaktion ermöglicht wird. Die angebotenen Dienste oder Protokolle werden lediglich simuliert mit dem Ziel, den Angreifer zuerst einmal zu täuschen und dabei möglichst viele Daten von ihm und der Netzwerkanfrage aufzuzeichnen.

Angreifer mit erweiterten Kenntnissen über das anzugreifende Protokoll oder Dienst könnten einen Honeypot dieser Klassifikation schnell erkennen, was jedoch stark von dem Umfang der Simulation abhängt.

Automatisierte Angriffe durch Malware können Low-Interaction Honeypots für reale Systeme halten und mit einem Angriff gegen das Protokoll oder Dienst beginnen, welcher mangels echter Interaktion jedoch nicht erfolgreich durchführbar sein kann. Diese Form des Honeypots ist dienlich, um zu ermitteln, ob oder woher dieser Honeypot erreicht und gefunden werden kann.

Medium-Interaction bietet einen definierten Funktionsumfang und diese Funktionen wurden speziell für den Einsatz als Honeypot programmiert. Wie auch die Low-Interaction Honeypots kann eine Menge an Diensten simuliert werden, doch ist eine gewisse Interaktion mit dem Angreifer oder der Malware notwendig. Es kann Beispielsweise ermöglicht werden, zusätzliche Programme auf das System kopieren zu lassen und die Ausführung dessen nicht zu ermöglichen [40].

High-Interaction lassen umfangreiche Aktivitäten des Angreifers oder Malware auf dem Honeypot zu. Anmeldungen sind, sofern sie das Protokoll vorsieht, möglich und dem Angriff wird eine große Bandbreite an Funktionen zugestanden. Oftmals bestehen solche Implementierungen aus Standardinstallationen von Betriebssystemen, welche für den Honeypot modifiziert wurden. Die Modifikationen beinhalten grundsätzlich Logging-Funktionalitäten, wie Beispielsweise einem Keylogger.

Dadurch, dass beim Angriff nahezu vollständig auf das System zugegriffen werden kann, sind Aufzeichnungen der Aktivitäten auf einem vernetzten System zu empfehlen, sodass nachträgliche Modifikationen der Logs durch den Angreifer nicht möglich sind [40]. Durch

die uneingeschränkten Möglichkeiten des Angreifers kann die Auswertung besonders umfangreich, interessant und aufschlussreich sein.

Research Honeypots dienen der Entdeckung neuer Angriffsverfahren und -Muster [40]. Diese Systeme erwarten nichts Spezielles, sie protokollieren erst einmal nahezu alles. Die protokollierten Ergebnisse werden nach einem entdeckten Angriff ausgewertet, um das Vorgehen identifizieren zu können. Dadurch lassen sich neue Regeln für fortan bekannte Angriffsmethoden definieren.

Production Honeypots sind im Produktionsumfeld eingesetzt und werden ohne Bekanntmachung eingerichtet. Dadurch ist zu erwarten, dass jede Verbindung dorthin nur durch einen Angriff oder einer Vorbereitung zu einem solchen, wie durch einer Netzwerkdurchsuchung, entstehen kann.

2.1.2 Einfluss auf die IT-Sicherheit

Erkenntnisse der durch Honeypots aufgezeichneten Daten können mehrfach positiv dienlich sein. Zum einen lässt sich ein Angriff auf die Infrastruktur der Honeypot-Umgebung frühzeitig erkennen, dies trägt auch die Bezeichnung Intrusion Detection System (IDS).

Weiterhin können die aus der Analyse gewonnen Daten weltweit ausgetauscht werden, wodurch andere Standorte die bislang unbekannten Angriffsverfahren kennenlernen und vorbeugende Maßnahmen ergreifen können. Der Austausch kann unterschiedlich und sollte bedarfsgerecht erfolgen. Zum einen durch fertige Regeldefinitionen, welche auf Basis der Analyse von protokollierten Vorgängen erstellt wurden, oder auch der Blick auf die rohen Logs um ein Review der Regeln zu ermöglichen. Global verteilte Honeypots können weitläufige Verbreitungswege aufzeigen, indem die Zeitstempel verglichen werden - sofern die gleichen Methoden von Malware oder Angreifern auf den verteilten Systemen aufgezeichnet und gefunden werden. Veränderungen am jeweiligen auffälligen Vorgehen zeigen die Anpassung der Angriffstaktiken oder Mutationen von Malware auf, welche durch entsprechende Fachkompetenz die Ziele der Schädlinge erahnen lassen kann.

So hatte es beispielsweise der prominente Wurm *Stuxnet* auf SCADA-Systeme von Siemens abgesehen, welche vielfach in Industrieanlagen Verwendung finden [41]. Mit einer

frühzeitigen Erkennung des Wurms und Ziels hätten Betreiber von solchen Anlagen gezielt informiert werden können. Die aus den Honeypot-Daten ermittelten Regeldefinitionen im Intrusion Prevention System (IPS) des jeweiligen Betreibers hätte eine weitere Ausbreitung des Wurms verhindern und die Systeme vor diesem schützen können.

2.1.3 Aktueller Stand zu Honeypots in der IT-Sicherheit

Honeypots verschiedenster Arten werden professionell von Sicherheitsanbietern betrieben. Die daraus gesammelten Informationen, wie Beispielsweise Angriffsmuster oder auffallende Adressen des Internet Protocol (IP), werden - meist gegen Bezahlung - von diesen Unternehmen in aufgearbeiteter Form angeboten.

Anwendungen wie ein IDS oder IPS arbeiten mit solchen Mustern, um bekannte Bedrohungen in den Netzwerken frühestmöglich zu erkennen und im Falle von IPS abzuwehren. Somit werden die am Netzwerk hinter einer IPS angeschlossenen Geräte vor diesen schädlich aufgefallenen Programmen und Vorgängen geschützt. Firewall-Systeme können auf bekannte IP-Adressen besonders reagieren. Zu den möglichen Aktionen können unter anderem ein direktes Blockieren, ein sofortiges Umleiten auf einen Honeypot oder eine Paketlimitierung der Netzwerkverbindung zur Unterbindung von Denial of Service (DoS)-Attacken gehören. Je nach technischer Möglichkeiten könnten Verbindungen, welche von oder zu den speziellen IP-Adressen aufgebaut werden oder bereits bestehen, aufgezeichnet und nachträglich auf Unregelmäßigkeiten hin analysiert werden.

Folgend werden aktuelle Lösungen und Systeme mit einer kleinen Zusammenfassung vorgestellt, welche in Bezug zu Honeypots, Gewinnung von Regeldefinitionen oder Anwendung dessen in IDS oder IPS stehen. Einige Systeme arbeiten bereits zusammen und verfolgen das gemeinsame Ziel, produktive Netzwerke vor Angriffen und Malware zu schützen.

Zwei sehr bekannte Anbieter von Sicherheitslösungen sind *Kaspersky* und *proofpoint*. *Kaspersky* betreibt neben Forschungs- und Entwicklungslaboren auch eigene Honeypots, um Bedrohungen aus dem Internet zu detektieren [35]. Ebenso unterhält *proofpoint* mit der Abteilung *Emerging Threats* ein weltweites Sensor-Netzwerk, um global Angriffsmuster frühestmöglich identifizieren und daraus Regeldefinitionen ableiten zu können [9].

Die Regeldefinitionen von *Emerging Threats* finden in dem IDS / IPS *Suricata* Verwendung [30]. Die aktuelle Version 6 (beta) dieser Software unterstützt bereits die Überwa-

chung von HTTP/2 und wird von der gemeinnützigen Stiftung *Open Information Security Foundation (OISF)* entwickelt.

Das IDS *Snort* [28] verwendet die Regeldefinitionen von *Cisco Talos* [4], dessen jüngste Erkenntnisse - einsatzbereit aufbereitet - in dem kostenpflichtigen Snort-Abonnement [29] zur Verfügung stehen. Erst 30 Tage später sind diese Definitionen frei erhältlich.

Im März 2019 wurde die aktuelle Version der Honeypot-Plattform *T-Pot*, welches von der Deutschen Telekom in einem Community Project entwickelt wird, veröffentlicht [24]. Das Konzept des *T-Pot* besteht darin, eine Menge verschiedener Honeypots, *Suricata* und Logging-Server neben weiteren Werkzeugen auf einer Linux-Distribution über Docker zu verbinden und das Gesamtsystem ohne administrativen Aufwand direkt betreiben zu können [31].

Roo [26] - etwas veraltet - bietet eine Werkzeugsammlung, welche auf Linux lauffähig ist und Windows-Dienste simulieren kann. Dabei werden Verbindungen protokolliert und auch Logdaten anderer Programme wie *Argus* [3], *Snort* [28], *Sebek* [27] oder *p0f* [21] können abgespeichert und analysiert werden. *Roo* wird als Layer 2 Bridge im Netzwerk eingebunden und ist deshalb augenscheinlich unsichtbar [39].

Wie die Auflistung zeigt, werden Honeypots und darauf aufbauende Lösungen stets weiterentwickelt. Der Betrieb dessen ist nicht unproblematisch. Auf wichtige Aspekte im Umgang mit Honeypots wird ab dem folgenden Kapitel 2.1.4 hingewiesen.

2.1.4 Schwierigkeiten mit Honeypots

Problematisch kann das Anlocken von Angreifern werden, sobald das System übernommen wird und als Schädlingshotspot im eigenen Netzwerk arbeitet. Diese Möglichkeit stellt bei der Überwachung einen wichtigen Aspekt dar und bereits vor der Inbetriebnahme sollte ein Notfall-Konzept für diesen Fall existieren und geübt worden sein. Idealerweise wurde der erfolgreiche und schädliche Angriff sowohl nachvollziehbar als auch integer protokolliert und kann zukünftig vermieden werden, indem fachkundiges Personal das Protokoll analysieren, Regeldefinitionen ableiten und diese Definitionen an geeigneter Stelle zum Einsatz kommen.

Ein Honeypot ist üblicherweise auf eine definierte Menge von unterstützten Protokollen oder Diensten abgestimmt und protokolliert entsprechend lediglich Daten zu erwarteten Vorgängen. Soll hingegen neuartiges oder unerwartetes Verhalten an den Honeypots

protokolliert werden, so bedarf die Auswahl an aufzuzeichnenden Daten eine geringere Limitierung. Wird dem Honeypot eine solche weitreichende Datenerfassung implementiert, dann werden massenweise Aufzeichnungen angelegt, dessen schlüssige Auswertung einen hohen Aufwand verursachen könnte. Dies erfordert viel Erfahrung bei den Definitionen der Überwachungsregeln, um einer übermäßigen Datensammlung vorzubeugen ohne wichtige Informationen auszulassen.

Vor der Anwendung fremder Regeldefinitionen im IPS ist das Vertrauen und die Seriosität der Quelle zu prüfen. Es könnten Regeldefinitionen mit sogenannten *False Positives* oder *False Negatives* angewendet werden, welche - möglicherweise absichtlich - harmlose Verbindungen als Angriff oder bekannte Bedrohungen als harmlos definieren. Zur Verdeutlichung und Sensibilisierung zur Problematik zeige ich folgend jeweils ein Beispiel auf:

False Positives sind harmlose Vorgänge, welche als schadhaft deklariert werden. Verbindungen zu einem Update-Server könnten verweigert und somit das Einspielen notwendiger Sicherheitspatches unterbunden werden.

False Negatives stellen schadhafte Verhalten dar, welche jedoch als harmlos eingestuft werden. Dies erlaubt Angreifern oder Malware den Zugang, welcher nicht zulässig ist.

2.1.5 Rechtliche Rahmenbedingungen im Umgang und Betrieb

Ohne Anspruch auf Vollständigkeit werden folgend rechtlich zu bedenkende Punkte aufgezeigt, um einen groben Einblick in die Rechtsprechung zu ermöglichen. Eine Rechtsberatung findet hier nicht statt, diese sollte zur Planung der Anwendung von Honeypots bei Fachjuristen angefragt werden.

2.1.5.1 Haftung

Nachdem ein Honeypot erfolgreich angegriffen wurde, könnte der Halter dessen für Aktivitäten des Systems haftbar gemacht werden. Insbesondere, weil die Honeypot-Plattform absichtlich für den Zweck der Malware- und Angriffsausbreitung betrieben wird. Entsprechend gewinnt die Beobachtung von ausgehenden Verbindungsaufbauten des Honeypots

eine große Bedeutung. Es bedarf der Sicherstellung, dass Systeme Dritter durch den Betrieb des Honeypots nicht kompromittiert werden können.

2.1.5.2 Datenschutz

Ein Honeypot könnte als Live-System betrachtet werden, woraufhin Benutzer, Geschäftspartner, Administratoren oder weitere Personen echte Daten eingeben könnten, welche anschließend als Klartext im Logging-Server protokolliert sein würden. Je nach System oder Dienst könnte vor der Anmeldung mit einer Willkommensnachricht auf eine Protokollierung hinweisen werden, dies ist jedoch nur bei interaktiven Umgebungen durchführbar. Dadurch, dass zumindest die *Production Honeypots* nicht bekannt gegeben werden, sollte die Erfassung und Weiterverarbeitung von Daten gutartiger Benutzer durch solche Honeypots unmöglich sein. Der Ausschluss einer Datenerfassung von sensiblen Eingaben sollte bei jedem Betrieb von Honeypots garantiert werden können.

Für die eigene Absicherung als Betreiber von Honeypots kann die Dokumentation, wann und wo ein solches System eingesetzt und wie beziehungsweise wo dieses publiziert oder im Netzwerk eingebunden wird, sicherlich dienlich sein. Die Inhalte einer rechtssicheren Dokumentation könnten durch Fachjuristen in Erfahrung gebracht werden.

2.2 Virtualisierung

Der Grundstein meines Honeypot-Konzepts wird aus der Virtualisierung bestehen. Folgend wird oberflächlich darauf eingegangen, wodurch diese Technologie auszeichnet wird und wie dabei entsprechende Hardware unterstützen kann. Darauf folgt eine kurze Zusammenfassung der Container-Technik und eine Vorstellung des Docker-Projekts schließt dieses Kapitel ab.

2.2.1 Virtualisierungstechnologie

Eine Abstraktionsschicht wird vom zu virtualisierenden Objektes erzeugt. Dies bezweckt, dass andere Instanzen, welche von einem anderen Objekt stammen dürfen, auf dieses virtualisierte Objekt in der Abstraktionsschicht zugreifen können und dabei logisch voneinander trennbar sind. Dieses Vorgehen ist sowohl auf Hardware- als auch Software-Ebene anwendbar und spart jeweilige Ressourcen.

2.2.2 AMD V und Intel VT-x

Die Prozessorhersteller AMD und Intel bieten zur effektiveren Virtualisierung eine Hardwareunterstützung mit den Namen AMD V beziehungsweise Intel VT-x. Diese Technologien ermöglichen die Beschleunigung des Umgangs mit Abstraktionsschichten und können über die Speicheradressierung eine Trennung dessen zum Host-System sicherstellen [1, 11].

2.2.3 Container

Anwendungen lassen sich zusammen mit der benötigten Laufzeitumgebung verpacken - in einen Container - und können getrennt voneinander virtualisiert betrieben werden. Zum Betrieb dieser Container gibt es unterschiedliche Hypervisor-Technologien wie *LXD* [14], *Docker* [7] oder *OpenStack* [20].

2.2.4 Containerisierungstechnologie Docker

Mit Programmen des Docker-Projekts lassen sich Linux-Container verwalten und betreiben. Die Software wird von der Docker Incorporated entwickelt und quelloffen unter der Apache-Lizenz in Version 2.0 zur Verfügung gestellt [2, 15].

Anwendungen können mitsamt der benötigten Laufzeitumgebung in Images verpackt werden. Für den Betrieb wird zu jeder Instanz das Image in eigene Container geladen und erhält somit eine große Kompatibilität zu unterschiedlichen Systemen. Die jeweiligen Container können voneinander logisch getrennt als auch miteinander verknüpft werden, um eine Zusammenarbeit unterschiedlicher Anwendungen zu gewährleisten.

Zum Docker-Projekt gehören noch der Dienst *containerd* und das Programm *docker-compose*.

Als Grundlage, dass Container überhaupt lauffähig sind und betrieben werden können, dient *containerd*. Der Dienst verwaltet die Instanzen der Images, Hardwareabstraktionen von Netzwerk und Speicher sowie Snapshots der jeweiligen Container.

Das Programm *Docker* ist eine Command Line Interface (CLI)-Anwendung, welche sowohl zum Bauen von Images und zur Steuerung einzelner Container dient als auch eine Benutzer-Schnittstelle zwischen dem Administrator und *containerd* darstellt.

Komplexere Projekte, sogenannte Kompositionen, können mittels *docker-compose* zusammengestellt und gesteuert werden. Dessen Konfiguration erlaubt die Einstellung von Abhängigkeiten zwischen den jeweiligen Containern, Schnittstellendefinitionen von Netzwerken oder Speicher und auch den gemeinsamen Start und Stopp des Projekts. Eine unabhängige Instanziierung der jeweiligen Projekte - sowohl auf demselben als auch vernetzter Hosts - wird ermöglicht. Dies erlaubt ein zentrales Deployment von untereinander abhängigen oder unabhängigen Microservices.

Diese Grundlagenerklärungen sollen die eingesetzten Technologien und Systeme, welche im Konzept aus dem folgenden Kapitel erwähnt und im Prototypen aus Kapitel 4 umgesetzt werden, verdeutlichen.

3 Konzept

Für eine zuverlässige Aufzeichnung von Verbreitungswegen nehme ich an, dass sowohl das bereits kompromittierte als auch anzugreifende System überwacht werden müssen. Zur Sicherstellung dessen setzt mein Konzept auf eine Demilitarisierte Zone (DMZ) als Honeypot, wie sie auch in der realen Welt Anwendung findet. Sie soll eine von außen zugängliche Administrationsschnittstelle auf einem sogenannten Jumphost beinhalten, von welchem die anderen Dienste verwaltet werden. Meine Annahme besteht darin, dass Malware oder Angreifer zuerst den Jumphost übernehmen wollen, um von dort aus Zugang zu Administrationsschnittstellen der anliegenden Dienste zu erhalten.

3.1 Anforderungen an das Konzept

Mein Konzept stellt Voraussetzungen an die Umgebung und soll gewisse Anforderungen erfüllen, welche folgend gelistet werden.

3.1.1 Voraussetzungen an die Umgebung

Das Konzept setzt eine TCP/IP-basierte Netzwerkumgebung voraus, welche Verbindungen zwischen den eingebundenen Geräten sicherstellt. Zur Gewährleistung der richtigen Adressierung eingesetzter Komponenten wird ein DHCP-Dienst erwartet.

3.1.2 Parallele Honeypot-Instanzen

Zur Bedienung mehrerer Angreifer oder Malware soll eine Vielzahl an Honeypot-Instanzen, logisch getrennt voneinander, realisiert werden können.

3.1.3 Sandbox-Technik

Zur Sicherstellung, dass Angreifer oder Malware aus dem Honeypot heraus nicht zum jeweiligen Host übergreifen kann, soll der Honeypot innerhalb einer sogenannten Sandbox betrieben werden.

3.1.4 Tarnung

Ein Honeypot darf gegenüber Angreifern oder Malware nicht offensichtlich erkannt werden.

Deshalb hat eine manipulierte Umgebung bei der Wiederkehr eines Angreifenden oder der Malware wieder so vorzufinden sein soll, wie diese hinterlassen wurde. Zur Umsetzung dessen wird eine Zuordnung auf Basis der IP-Adresse der Netzwerkanfrage durchgeführt; eine Verschleierung des Vorgangs wird mit einem Hashwert der Absenderadresse erreicht, welcher anstelle der IP verwendet wird. Erstmalige Netzwerkanfragen erzeugen einen neuen Honeypot, sofern die angefragten Dienste von diesem unterstützt werden. Eingehende Verbindungsanfragen an einen nicht implementierten Dienst können optional protokolliert aber müssen nicht angenommen werden.

Weiterhin soll der Begriff *Honeypot* oder dergleichen bei der Umsetzung nicht verwendet, sondern mit anderen, plausiblen Bezeichnungen ersetzt werden.

3.1.5 Kompatibilität

Das gesamte System soll von der Laborumgebung ohne große Anpassungen auf andere Systeme portiert werden können.

Zur Gewährleistung von einfachen Wartungen und Aktualisierungen, sollen verwendete Anwendungen nach Möglichkeit innerhalb virtualisierter Container betrieben werden.

3.1.6 Datenerfassung

Der Honeypot soll dem Typus eines Research-Honeypots entsprechen, um neue Vorgehensweisen bei der Ausbreitung von angreifenden und Malware erkennen zu können. Hierzu ist eine umfangreiche Datenerfassung vom Verbindungseingang bis zum Beenden

des Honeypots notwendig. Eine Protokollierung von Verbindungen an nicht verwendeten Diensten ist optional.

3.1.7 Integrität

Ermittelte Daten des Honeypots müssen auf einem anderen Host schreibgeschützt gesammelt werden. Der Daten vorhaltende Host ist so abzusichern, dass lediglich die Datenübertragung der Honeypots und Auswertungen von Analysten ermöglicht wird. Weitere Netzwerkverbindungen sind grundsätzlich zu verhindern; Wartungsaufgaben bilden eine Ausnahme. Dies schützt die Aufzeichnungen vor Manipulation durch angreifende oder Malware, wodurch die Integrität dieser Daten sichergestellt werden soll.

3.1.8 Analyse der Daten

Aufgezeichnete Events müssen analysiert werden können. Hierzu bedarf es die Möglichkeit der Auswertung auf dem Daten vorhaltenden Host. Eine Schnittstelle zum lesenden Datenabgleich für Analysten ist optional.

3.1.9 Lizenzierung der Komponenten

Verwendete Komponenten müssen mindestens zur Forschung und Lehre frei lizenziert sein. Wünschenswert ist eine kostenlose Nutzung zu gewerblichen Zwecken, um eine Portierung des Konzepts beziehungsweise Realisierung unbürokratisch in Unternehmensumgebungen zu ermöglichen. Offener Quelltext der jeweiligen Komponenten zur Ermöglichung von Reviews an diesen wäre ideal, ist jedoch optional.

3.2 Grobe Architektur

Das Konzept umfasst mindestens zwei Maschinen, welche über ein Netzwerk miteinander verbunden sind. Mindestens eine Maschine dient dem Betrieb des Honeypots, die andere als Logging-Server zum Festhalten der aufgezeichneten Daten. In der weiteren Beschreibung wird die Anzahl der Maschinen für den Honeypotbetrieb auf 1 definiert, zusätzliche Realisierungen erfolgen analog.

3.2.1 Honeypot-Maschine

Zur Gewährleistung der Portabilität zu anderen Systemen und zur Umsetzung der Sandbox-Technik wird eine Hypervisor-Virtualisierung eingesetzt. Innerhalb des virtualisierten Systems arbeiten mehrere Komponenten miteinander:

1. Ein Superserver zur Annahme von Netzwerkverbindungen
2. Die Honeypot-Instanzen
3. Software zum Starten neuer Honeypots
4. Software zur Zuordnung von Netzwerkanfragen an die jeweiligen Honeypot-Instanzen
5. Eine Komponente zur Beobachtung und Protokollierung interessanter Events
6. Dateibeobachtung von Log-Dateien mit unverzüglicher Übermittlung der protokollierten Daten an den Logging-Server
7. Ressourcenüberwachung der Maschine mit Datenübermittlung an den Logging-Server
8. Bereinigung alter oder störender Honeypots

Der chronologische Ablauf und die Zusammenarbeit der Komponenten wird folgendermaßen angenommen. Nachdem über das Netzwerk eine Anfrage an den Superserver eingegangen ist, prüft eine Software auf die Existenz eines bestehenden Honeypots zur Absender-IP-Adresse. Im positiven Fall wird die eingehende Netzwerkanfrage an den Honeypot durchgeschliffen. Andernfalls startet die Software eine neue Honeypot-Instanz, zu welcher die Netzwerkanfrage anschließend weitergeleitet wird. Dieses Vorgehen soll von einer Komponente beobachtet werden, welche die jeweiligen sowie zukünftigen Events zur Speicherung in eine Log-Datei ausgibt. Mittels der Beobachtung dieser Log-Datei erfährt eine weitere Komponente, dass ein Event stattgefunden hat und übermittelt die neuen Daten an den Logging-Server.

Unabhängig der Vorgänge soll in regelmäßigen Abständen die Ressourcen des Hosts gemessen und die ermittelten Daten an den Logging-Server übermittelt werden, um mögliche Änderungen am jeweiligen Gebrauch feststellen zu können. Honeypot-Instanzen,

welche ein störendes Verhalten aufweisen oder nicht mehr von Interesse sind, sollen beendet werden können; mit dieser Aufgabe wird die letzte Komponente in der oberen Auflistung betraut.

3.2.2 Logging-Server

Auf der zweiten Maschine, welche den Logging-Server betreibt, werden eingehende Daten gespeichert und zur Auswertung aufbereitet und ausgegeben. Eingereichte Daten vom Honeypot müssen richtig interpretiert werden können, daher ist eine hohe Kompatibilität zur datensendenden Einheit zwingend sicherzustellen. Eine Bedienoberfläche durch ein Web-Interface ist anzustreben, sodass erste Analysen keine besonderen Programme bei den jeweiligen Experten voraussetzen. Die Möglichkeit, Daten auf Basis ihrer erzeugten Quelle oder Uhrzeit zu ermitteln, soll gegeben sein.

4 Prototyp

Zur Demonstration, dass mein Konzept des vorhergegangenen Kapitels 3 zur Erkennung und Protokollierung der Ausbreitung von Malware und Angriffen innerhalb eines Netzwerks funktioniert, habe ich einen Prototyp entwickelt, welcher in diesem Kapitel vorgestellt wird. Auf die grundsätzlichen Konzepte, Technologien und Systeme, welche für diesen Prototyp von besonderer Bedeutung sind, wird näher eingegangen.

4.1 Umgebung

Meine Laborumgebung besteht aus einem lokalen IPv4 Netzwerk mit der Adressverwaltung durch einen zentralen DHCP-Dienst. Alle erwähnenswerten Systeme bauen auf Linux oder vergleichbarem Derivat, sei es als Host-System oder virtualisiert.

4.2 Auswahl konkreter Komponenten und Anwendungen

Zwischen der Konzeptionierung und Realisierung erfolgt die Entscheidung von möglichen Systemen, welche zum Einsatz kommen sollen. Folgend gehe ich auf die Ergebnisse meiner Suche der passenden Komponenten ein. Grundsätzlich entscheide ich mich stets für die aktuelle stabile Version jeder Komponente, sofern dies mit der Kompatibilität zu in Abhängigkeit stehenden Bausteinen vertretbar ist.

4.2.1 Server für den Betrieb des Honeynets

Neben der Entwicklung des Honeynets mit ressourcenschonenden Deployment-Phasen ist es mir besonders wichtig, bei möglichen Fehlfunktionen schnell korrigierend eingreifen und Sicherungen von Entwicklungsständen wiederherstellen zu können. Daher entscheide ich mich für eine Typ-2-Hypervisor-Virtualisierung mittels VirtualBox auf meinem

Notebook, dessen x64 Prozessor mit Intel VT-x Technologie ausgestattet ist [11]. Mit diesem Notebook wird ebenfalls die Entwicklung des Prototyps durchgeführt, welches die Deployment-Phasen kurz hält. Eine Virtualisierung bietet neben Snapshots den Vorteil der leichten Portierbarkeit des Projekts auf andere Hardware und Host-Betriebssysteme sowie den Einsatz der Sandbox-Technik. VirtualBox in der Version 6.1.10 soll auf der Linux-Distribution Ubuntu 20.04 Verwendung finden - beide Produkte sind aktuell, stabil und werden jeweils mit einer guten Abwärtskompatibilität weiterentwickelt [33, 34].

Debian 10.4 (Buster) als Gastsystem in der virtuellen Maschine wird gewählt, weil Debian meiner persönlichen Auffassung nach eine sehr robuste und gut gepflegte Grundlage bietet. Mit der Administration über das CLI bin ich seit gut 15 Jahren vertraut. Scripte zur automatisierten Installation und Konfiguration von zusätzlichen Anwendungen beschleunigen das Aufsetzen des Systems, dokumentieren die zum Einsatz gewählten Programme sowie Einstellungen und stellen sicher, dass keine Abweichungen zwischen Neueinrichtungen entstehen.

In dieser Linux-Distribution findet standardmäßig der Linux-Kernel 4.19 Anwendung und wird für dieses Projekt beibehalten, weil dieser alle Voraussetzungen erfüllen kann. Dazu gehören die Unterstützung der VirtualBox Gast-Erweiterungen zur Sicherstellung der Virtuelle Maschine (VM)-Shares sowie die Möglichkeit den Kernel zu überwachen.

Als Superserver wird *xinetd* in Version 2.3.15 verwendet. Ausgeschrieben bedeutet der Name *Extended Internet Service Daemon* und geboten ist ein einfach zu administrierender Dienst für Verbindungsanfragen über TCP/IP. Der Programmcode ist bewährt [37]: Sicherheitsschwachstellen zur aktuellen Version dürften bereits gefunden worden sein und meine Recherche auf bekannte und offene Lücken fiel erfreulicherweise negativ aus. Aufgrund dessen und dem maßvollen Funktionsumfang wird nur wenig Angriffsfläche gegen das System für Malware oder Angreifer geboten. Weiterhin können eigene Programme wie Shell-Scripte an TCP- oder UDP-Ports gebunden werden, welches eine benötigte Eigenschaft zur Verbindung von Netzwerkanfragen an den zugehörigen Container im richtigen Honeynet darstellt. Zusätzlich loggt *xinetd* zuverlässig die absendende IP-Adresse der Verbindung neben dem aufgerufenen Port, generierte Prozess-ID des Scripts und Zeitpunkt, dessen Informationen zur Analyse von Bedeutung sein werden.

Somit erfüllt *xinetd* alle notwendigen Voraussetzungen für den Verbindungsempfang, welche dem Konzept im Kapitel 3 entnommen werden können.

Die Containervirtualisierung darf *containerd* 1.2.13 von der Docker Incorporated [7] realisieren. Dieser Dienst betreibt die jeweiligen Instanzen der Honeynets und des Logging-Clients. Verwaltet werden diese mit dem Werkzeug *docker* Version 19.03.12. Zum Erstellen (bauen), Starten und Stoppen der benötigten Kompositionen wird *docker-compose* in der Version 1.26.2 gewählt.

Diese drei Werkzeuge, welche kompatibel zueinander arbeiten können und ihren gemeinsamen Ursprung bei dem Docker Projekt innehaben, werden die zentralen Verwalter der Containerisierung.

Nebenbei sei erwähnt, dass aktuell die Projektstruktur bei Docker umgestellt und zukünftig Teile in das neue *moby*-Projekt ausgegliedert werden [16], doch betrifft es diese Thesis nicht; Lediglich bei einer zukünftigen Weiterentwicklung könnte diese Umstrukturierung Bedeutung erlangen.

Eine Event- & Ressourcen-Protokollierung soll innerhalb der virtuellen Maschine mit einer Zusammenstellung aus *falco*, *filebeat* sowie *metricbeat* umgesetzt werden. Gruppiert bezeichne ich diese Komposition als *Logging-Client*. Dazu schreibt *falco* alle zu den zu definierenden Filterregeln gefundenen Kernel-Events zusammen mit den jeweiligen Informationen und der ID des verursachenden Containers in das StdOut der virtuellen Maschine. Diese Ausgabe wird von *containerd* in die jeweilige Log-Datei des Containers übertragen, welche wiederum von *filebeat* überwacht und letztendlich formatiert an den Logging-Server übermittelt wird.

Metricbeat dient lediglich der Überwachung von verwendeten Ressourcen innerhalb der VM, somit können Verwendungen an Prozessor, Arbeitsspeicher, Festplatte oder Netzwerk erkannt und bei einer Auswertung des Logging-Servers möglichen Angriffen oder Programmen kausal zugewiesen werden.

Das Honeynet wird als Komposition aus zwei zu entwickelnden Container-Images zusammengesetzt, auf welche in den folgenden Absätzen eingegangen wird.

Das Jumphost-Image des Honeynets wird auf einem schlanken Debian 10.4 (Buster-Slim) [6] aufgesetzt, auf welchem der jeweils aktuelle *openssh-server* [19] sowie *telnetd* [32] über das Advanced Packaging Tool (apt) installiert werden. Zusätzlich, als Dreingabe für mögliche Malware oder erfolgreich angemeldete Angreifer, sollen die *net-tools* [17] installiert bereitliegen, womit aktuelle Netzwerkeinstellungen des Containers ausgelesen oder manipuliert werden können. Die Absicht hinter dieser gebotenen Möglichkeit besteht darin zu ermitteln, ob eigene Netzwerke durch schadhafte Programme oder eigentlich ungebetenen Gästen eingerichtet und wie diese konfiguriert werden.

Das Web-Image erhält planmäßig *nginx* 1.18 als primären Dienst zur Bedienung des HTTP-Ports 80. Debian 10.4 (Buster-Slim) soll ebenfalls als solide Grundlage des Image die Basis liefern und eine gewisse Homogenität zur Weiterentwicklung, Administration und Wartung sicherstellen. Zusätzlich wird dieses Image mit der gleichen Zusatz-Umgebung wie die des Jumphosts ausgestattet, sodass eine Verbindung von diesem auf den Jumphost-Container erleichtert wird. Dieser Fall kann eintreten, sofern der Web-Container zuerst übernommen wird und sich ein Angreifer oder Malware von dort ausbreiten möchte.

Aufgrund manueller Konfiguration der Images wird jede technisch realistische Ein- und Zusammenstellung der jeweiligen Container ermöglicht.

In meinem Prototyp plane ich die jeweiligen Dienste unverändert zu verwenden, jedoch unsicher zu konfigurieren. Die Unsicherheit soll mit einfach zu erratenen Benutzernamen und Kennwörtern umgesetzt werden, sowohl für einfache Benutzer als auch dem Superuser *root*. Eine Liste der anzulegenden Benutzernamen je Image soll zentral vorgegeben und die Kennwörter sollen separat, je nach Benutzergruppe oder *root*, konfigurierbar werden. Zusätzlich soll *sshd* eine Anmeldung als *root* ermöglichen, da dieser Benutzername bei Brute-Force Angriffen meiner Erfahrung nach am häufigsten Anwendung findet und Auswertungen der protokollierten Ereignisse von Befehlen des Administrationskontos auf dem Honeypot zu aufschlussreichen Ergebnissen führen kann.

Eine weitere Konfiguration an *nginx* ist hier nicht vorgesehen, die Standardeinstellungen sind samt der einfachen Startseite ausreichend: Die Standard-Startseite soll Angreifer vermuten lassen, dass der Dienst noch nicht weiter gehärtet wurde und Angriffe möglicherweise mit Erfolg durchgeführt werden können. Es wird damit potenziellen Angreifern und Malware die Möglichkeit gegeben, die zugehörige *index.html* zu modifizieren oder weitere Ordner und Dateien hinzuzufügen.

4.2.2 Server zur Aufzeichnung und Aufarbeitung der Logs

Der Logging-Server soll, im Gegensatz zum Honeynet, keine große Entwicklungsarbeit binden, sondern mit einer marktüblichen Lösung funktionieren. Die grundsätzliche Aufgabe, Log-Informationen zu empfangen, durchsuchbar aufzubereiten und formatiert mit Filtermöglichkeiten auszugeben, bieten bereits einige Systeme und sind oftmals zueinander kompatibel.

Ich habe mich für den Elasticsearch, Logstash & Kibana (ELK)-Stack Version 7.6.2 entschieden. Dort enthalten sind Elasticsearch als Datenbankmanagementsystem für dokumentenbasierte Datenbanken, Logstash zur Aufarbeitung und Interpretation von eingehenden Log-Einträgen sowieso Kibana, eine grafische Oberfläche mit Anbindung an Elasticsearch zur komfortablen Filterung und optischen Aufarbeitung der Dokumente, um eine Auswertung möglichst zeitsparend zu ermöglichen.

Dieses Bundle wird vom Unternehmen Elastic zur Verfügung gestellt, wie auch die Komponenten Filebeat und Metricbeat. Eine große Kompatibilität zueinander ist somit gewährleistet.

Absichtlich wählte ich hier nicht die aktuell neueste Version 7.8.0, denn ich vertraue bei solch großen Projekten auf mehrfach aktualisierte Minor-Versionen mit der Erwartung, dass auftretende Probleme bereits gefunden und behoben sein werden. Meine Erwartung an der Version 7.6.2 ist eine zuverlässige Laufzeit [8].

Lizenziert werden die Software-Produkte unter der Apache-Lizenz und ermöglichen somit Einblick in den Code sowie eine kostenfreie Verwendung - sowohl für die Forschung & Lehre als auch zu gewerblichen Zwecken [2].

Die Hardware für den Logging-Server wird von den Honeynets getrennt, wofür einige Argumente sprechen. Zum einen ist die Trennung der Performance dienlich. Ausgelastete Honeypots könnten das Logging negativ beeinflussen, wodurch beobachtete Events gegebenenfalls nicht verarbeitet werden würden. Umgekehrt wäre es ebenfalls denkbar, dass Auswertungen auf dem Logging-Server, oder andere Prozeduren auf diesem, die Honeynets negativ beeinflussen könnten. Aktuelle Technologien erlauben logische Trennungen verschiedener Systeme inklusive Ressourcenzuweisungen auf derselben Hardware, jedoch kann dies eine unerwünschte Abhängigkeit zu diesen Technologien fördern.

Weitere Argumente für eine klare Trennung sind, dass auch Prozesse des Logging-Servers mit denen des Honeypots durch die Überwachung des Logging-Clients korrelieren; schließlich würden sie sich mindestens den Prozessor und Arbeitsspeicher teilen, welche von *Metricbeat* überwacht werden.

Weiterhin könnte eine sehr erfolgreiche Malware oder ein überaus intelligenter Angreifer möglicherweise aus den Sandboxen der Container- und Hypervisor-Virtualisierung ausbrechen. In diesem Fall wäre es nicht auszuschließen, dass an den Daten des Logging-Servers Manipulationen stattfinden oder gar Schadcode an den Analysten ausgegeben wird.

Ebenfalls wäre es denkbar, dass anstelle des Honeypots, der Logging-Server von Angreifern oder Malware zum Ziel wird. In diesem Fall wären Aufzeichnungen nicht mehr zuverlässig möglich oder die Integrität der Daten gefährdet. Firewall-Lösungen müssen außerhalb der Laborumgebung den Zugang zum Logging-Server absichern, indem nur Verbindungen vom Honeypot-Servers als auch aus dem Analysten-Netzwerk an die jeweils notwendigen Ports zugelassen und restliche Netzwerkanfragen abgewiesen werden.

Die Wahl der Hardwareplattform fiel auf ein gut ausgestattetes Network Attached Storage (NAS), eine QNAP TS-453Be mit der aktuellen Firmware Version 4.4.3.

Diese ist mit 16 GByte Arbeitsspeicher, 4 Festplatten im RAID5- und 2 SSDs im RAID1-Verbund ausreichend performant. Angebunden ist die NAS mit einer GBit-Ethernetleitung im Netzwerk, eine vorgesetzte Unterbrechungsfreie Stromversorgung (USV) sichert sie vor einem kurzzeitigen Stromausfall, wodurch die Gefahr des Datenverlusts reduziert wird.

Vorteilhaft für die Experimente dürfte es sein, dass die NAS bereits durchgehend im Betrieb ist und eine Containervirtualisierung über das Programm ContainerStation in Version 2.1.3 unterstützt. Dies ermöglicht die Ausführung des Logging-Servers als Container und beschleunigt die Inbetriebnahme im Vergleich zur manuellen Installation. Während Entwicklungspausen kann das System im Hintergrund weiterlaufen und ist für weitere Tests des Honeynets bereit unverzüglich Daten zu empfangen und weiterzubearbeiten.

4.2.3 Graphische Darstellung

Das folgende Diagramm zeigt den Aufbau der zusammenhängenden Komponenten des Systems, welches in einer virtuellen Maschine umgesetzt werden soll.

Der detailliertere Aufbau eines Honeynets ist hierauf anschließend abgebildet.

Nicht ersichtlich, aber dennoch notwendig, sind die Aufzeichnungen des Superservers bei Verbindungseingang und -abbruch sowie die manuellen Logs, welche beim Start beziehungsweise Stop eines Honeynets angelegt werden. Diese müssen ebenfalls von Filebeat gelesen und an den Logging-Server übertragen werden.

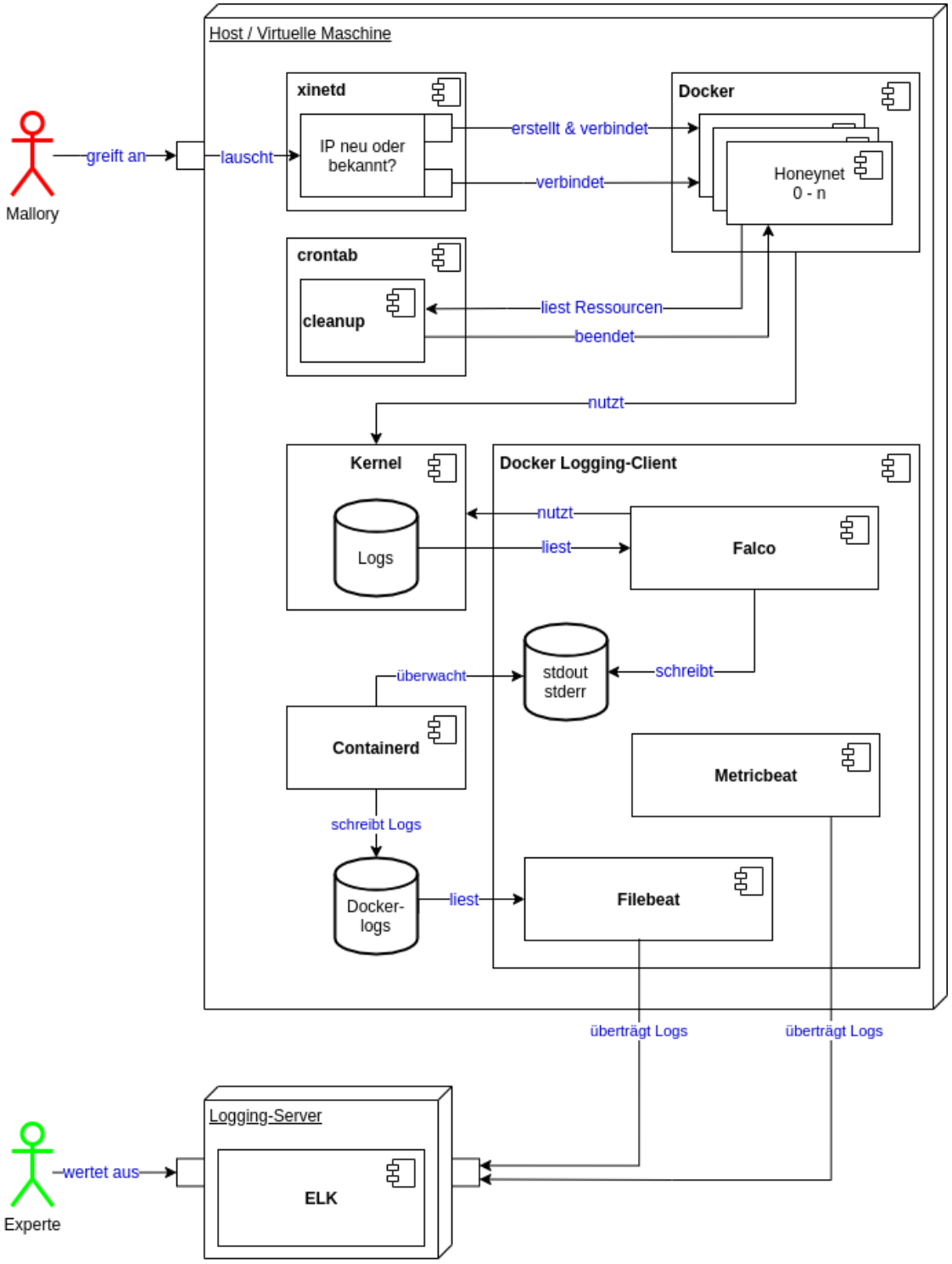

Abbildung 4.1: Vollständiges Komponentendiagramm

Die jeweiligen Honeynets bestehen aus einer Komposition zweier Container, welche aus den beiden Images *Jumphost* und *Web* bestehen. Je nach angefragten Port wird der jeweilige Container zur zugehörigen Netzwerkverbindung durch den Superserver xinetd beziehungsweise einer von ihm aufgerufener Software zugewiesen. Innerhalb des Honeynets ist eine Kommunikation zwischen den Containern uneingeschränkt möglich, um die Aufzeichnung von versuchten oder erfolgreichen Verbindungen zu ermöglichen.

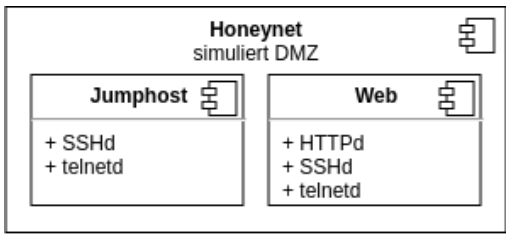

Abbildung 4.2: Honeynet im Detail

4.3 Realisierung

Zuerst fange ich mit dem Honeypot- beziehungsweise Honeynet-Server an, welches die Einrichtung der virtuellen Maschine mit dem Honeynet und Logging-Client umfasst. Anschließend wird die Realisierung des Logging-Servers auf dem NAS umgesetzt. Der Betriebsaufnahme des ELK-Stack folgt die detailliertere Konfigurationen vom Logging-Client: Es wird die vom DHCP-Dienst zugewiesene IP-Adresse des Logging-Servers in der zentralen docker-compose.yaml [A.2.1] eingestellt. Die Experimente als auch Entwicklung der Falco-Regeln erfolgen im Vorgangsmodell des *Rapid Prototypings* im späteren Kapitel 5.

4.3.1 Server für den Betrieb des Honeynets

Beginnend wird auf dem vorhanden Server, meinem Notebook mit Ubuntu, die virtuelle Maschine eingerichtet. Sobald in dieser das Gast-Betriebssystem Debian Buster hochfährt, erfolgen die Installationen der benötigten Zusatzprogramme. Dafür habe ich ein Installationsscript geschrieben, welches sowohl der Dokumentation dienlich ist als auch dazu beiträgt, die Umsetzung auf anderen Systemen portieren zu können und gleichbleibende Grundinstallationen sicherzustellen [A.1.2.1].

Die jeweiligen Scripte und Konfigurationen werden mit einem schreibgeschützten VM-Share der VM zugänglich gemacht, wodurch Zugriffe auf diese Dateien als auch der Schutz vor Manipulationen an diesen gewährleistet wird. Zeitgleich ermöglicht diese Konfiguration, dass Anpassungen durch den Administrator oder Entwickler in der VM ankommen und durch die jeweiligen Programme verwendet werden können. Details dazu erfolgen im kommenden Absatz.

4.3.1.1 Die virtuelle Maschine...

...erhält eine eigene, virtuelle Netzwerkkarte samt MAC-Adresse. Die kompatible und gleichbleibende Adressierung durch den DHCP-Dienst im Netzwerk als auch die notwendige logische Trennung des Netzwerkverkehrs werden dadurch ermöglicht.

Weiterhin wird der VM zwei geteilte Ordner, die VM-Shares, zugewiesen, worüber die auf dem Host liegenden Scripte und Konfigurationen der VM zugänglich gemacht werden. Die Möglichkeit der Modifikation an diesen Dateien innerhalb der VM ist nicht notwendig, daher werden diese VM-Shares schreibgeschützt eingebunden, was auch der Sicherheit des Host-Systems dienlich ist, vor Manipulation der Konfigurationen schützt und somit die Integrität sicherstellt.

Listing 4.1: VM-Shares

```
1 01 - Docker [idRoot=0 readonly auto-mount host-icase guest-icase mnt-
      pt=/mnt/docker]
2 02 - HP-Host [idRoot=1 readonly auto-mount host-icase guest-icase mnt
      -pt=/mnt/hphost]
```

Im Ordner *Docker* werden die Kompositionen aller Umgebungen mitsamt der jeweiligen Dockerfiles und zugehörigen Konfigurationen bereitgestellt. Aus diesen Informationen können innerhalb der VM die Images erstellt und Container betrieben werden.

Der *HP-Host* Ordner beinhaltet das Script zur Installation und Einrichtung notwendiger Komponenten innerhalb der VM, das auszuführende Script für xinetd [A.1.2.6] um potenzielle Angreifer und Malware entsprechend in Empfang nehmen zu können und ein Script zum aufräumen veralteter Honeynets oder denjenigen Containern, welche zu viel CPU-Zeit in Anspruch genommen haben [A.1.2.2].

Einstellungen der Debian-Installation wurden im Installations-Assistenten des Betriebssystems durchgeführt. Hier wurde ein SSH-Server als zusätzlicher Dienst ausgewählt, während andere Dienste bis auf die Standard-Systemwerkzeuge deselektiert wurden. Der Standard-Benutzername wurde mit *hpuser* festgelegt und erhielt für meine Laborumgebung das Kennwort *123456*. Für root wurde ebenfalls eine sehr einfache Zeichenkette, nämlich *123456789* als Kennwort festgelegt.

Das Script zum Einrichten der Linux-Installation für die geplante Honeynet-Umgebung in Form von Containervirtualisierung muss nach der erfolgreichen Betriebssystem-Installation von *root* ausgeführt werden. Beginnend wird dem Benutzer *hpuser* das Recht zugesprochen, auf die VM-Shares zuzugreifen. Dies erfolgt, indem dieser der Benutzer-Gruppe *vboxsf* hinzugefügt wird. Anschließend werden die Programme *Docker* und *docker-compose* installiert, wie es die jeweiligen Installationsanleitungen vorgeben. Auch hier wird dem Standardbenutzer wieder das Recht eingeräumt Docker überhaupt verwenden zu dürfen, indem er in der Gruppe *docker* aufgenommen wird. Um für die erste Netzwerkanfrage bereits die benötigten Images vorrätig zu haben, werden die Kompositionen für das Honeynet als auch des Logging-Clients anschließend gebaut, wobei die abhängigen Layer heruntergeladen und zusammengesetzt werden.

Es folgt die Installation von *xinetd* und *socat*, welche zusammen die Aufgabe übernehmen werden, eingehende Netzwerkanfragen an definierte Ports den zuständigen Containern in den richtigen Honeynets weiterzuleiten. Diese Zusammenarbeit wird mit einem Script [A.1.2.6] koordiniert. Für *xinetd* werden nach der Installation die Konfigurationen aus dem VM-Share *hphost* in den zugehörigen Konfigurationsordner kopiert und mit einem Schreibschutz versehen. Es folgt der Start des Dienstes, woraufhin anschließend zur Kontrolle die aktuelle Portbelegung über *netstat* ausgegeben wird.

Abschließend erhält der Dienst für regelmäßige Aufgaben, *crontab*, die Aufgabe minütlich das Aufräum-Script [A.1.2.2] auszuführen.

Die Einrichtung der VM ist damit abgeschlossen, es laufen der Logging-Client, xinetd empfängt Netzwerkanfragen für das Honeynet und zur Bereinigung läuft das Script mit den Aufräumaktionen in regelmäßigen Abständen.

4.3.1.2 Eingehende Netzwerkanfragen

Verbindungsaufbauten an den unterstützten Ports 22, 23 und 80 müssen entgegengenommen und richtig zugewiesen werden.

Die Zuweisungen von Netzwerkanfragen an die passenden Container beziehungsweise das Starten von Honeynets bei neuen Anfragen wird durch ein Script [A.1.2.6] durchgeführt, welches von xinetd bei bedienenden Ports aufgerufen wird. Auf Basis der IP-Adresse wird ein alphanumerischer Hashwert erzeugt, welcher als Projektname des Honeynets verwendet wird. Dieses Hashen dient zum einen der Vermeidung möglicher Komplikationen in der Verarbeitung von Sonderzeichen wie Punkten (IPv4) oder Doppelpunkten (IPv6) als auch der leichten Verschleierung des Honeypots vor Angreifern oder Malware. Zusätzlich kann eine Prüfung auf die Nichtexistenz des Honeynets, als auch eine spätere Zuteilung zur IP-Adresse, mit einem Aufwand von $\mathcal{O}(1)$ durchgeführt werden, weil der konkrete Name des Projekts bereits feststeht und eine iterative Suche durch die aktiven Honeynet-Projekte vermieden wird.

Als Hashverfahren findet *md5* Verwendung, da dies sehr schnell Hashwerte erzeugt als auch auf nahezu jedem Linux-System in Form des Programms *md5sum* vorzufinden oder installierbar ist. Andere Hashverfahren können ebenfalls implementiert werden, es wäre lediglich die Installation des jeweiligen Programms sicherzustellen als auch eine zentrale Funktion im verarbeitenden Script *hp-welcome.sh* anzupassen:

Listing 4.2: Hahsverfahren

```
1 # returns hashed string. String must be submitted as first arg
2 str_hash() {
3  echo $(printf '%s' $1 | md5sum | cut -d ' ' -f 1)
4 }
```

Das Starten neuer Honeynets mit ermitteltem Projektnamen erfolgt nur, sofern diese nicht bereits aktiv sind. Der Befehl *docker ps* liefert mit filterndem Parameter die Information auf das Vorhandensein und wurde in der Funktion *is_up()* programmiert, welche einen boolschen Wert ausgibt. Beim Start des Honeynets über *docker-compose* wird neben dem Projektnamen noch ein Pfad einer Datei übermittelt, in welche durch *docker-compose* entstehenden Ausgaben protokolliert werden sollen. Meldungen an *STDOUT* als auch *STDERR* finden Beachtung und werden gleichermaßen erfasst.

Dies ermöglicht es dem Logging-Client diese informativen Daten unverzüglich einzulesen und weiterzuverarbeiten.

Listing 4.3: Honeynet bei Bedarf erstellen und starten

```
# create_hn, submit PROJECT_NAME as arg 1
create_hn() {
 `is_up "${1}"` || docker-compose -f "${PROJECT_YML}" -p "${1}" up -d
     >/var/log/dockercompose_${1}.log 2>&1
return $?
}
```

Verbunden werden Netzwerkanfragen zu den jeweiligen Containern innerhalb des zur IP-Adresse passenden Honeynet-Projekts mit dem Programm *socat*. Auch dies wurde wieder zentral implementiert, um zukünftige Anpassungen zu vereinfachen. Die Ermittlung der benötigten IP-Adresse des Containers und Port zum Service wird durch *docker inspect* mit entsprechenden Filtern realisiert und zentral in der Funkion namens *get_container_ip()* programmiert.

Listing 4.4: interne IP und Port ermitteln

```
# get_container_ip PROJECT_NAME SERVICE_NAME
get_container_ip() {
 echo $(docker inspect --format='{{range .NetworkSettings.Networks
     }}{{.IPAddress}}{{end}}' $(docker ps -q --filter "label=com.
     docker.compose.project=${1}" --filter "label=com.docker.compose.
     service=${2}" ))
}
```

Listing 4.5: Verbindung mit socat an IP und Port

```
# connect request to IP (arg 1) and port (arg 2)
connect() {
 exec /usr/bin/socat stdin tcp:${1}:${2},interval=1,retry=30
}
```

4.3.1.3 Beendigung von Honeynets

Verlassene Honeynets oder ressourcenraubende Container sollen für eine klare Übersicht beziehungsweise Schonung der verfügbaren Ressourcen beendet werden.

Ein minütliches aufräumen wird durch *crontab* und einem Script sichergestellt. Dieses erkennt Container in Honeynets anhand verbrauchter CPU-Zeit oder Alter und kann diese entsprechend behandeln. In meinem Prototyp werden die identifizierten Systeme einfach gestoppt, nachdem ein Log-Eintrag mit der Information des Aufräumens zur jeweiligen Log-Datei geschrieben wird.

Damit die dem Honeynet zugehörigen Container identifiziert werden können und nicht der Logging-Client vom aufräumenden Script behandelt wird, wurden die Honeynet-Container mit einem unauffälligen Label *dmz=1* versehen, wonach zuverlässig gefiltert werden kann. Der Name des Labels *dmz* entstand daher, dass der Aufbau des Honeynets einer DMZ angelehnt ist.

Listing 4.6: Filterung nach Container mit Label "dmz"

```
1 for CID in $(docker ps --no-trunc -q --filter "label=dmz=1")
2 do
3   ...
4 done
```

Die Identifizierung nach Alter gewährt Malware die Möglichkeit, auch zu einem späteren Zeitpunkt aktiv zu werden. So manche Malware nistet sich erst mal ein und beginnt zu einem späteren Zeitpunkt mit der Wirkung [5]. Oder auch Angreifer sammeln erst einmal eine Menge von erbeuteten Systemen und verwenden diese für weitere Zwecke, sobald sie benötigt werden. Daher kann es dienlich sein, ein befallenes Honeynet längere Zeit vorzuhalten. Nach einer definierten Zeit können diese dennoch beendet werden, um sich von Altlasten zu befreien. Es ist durchaus wahrscheinlich, dass sich die IP-Adresse des Angreifers zwischendurch geändert hat, wodurch eine richtige Zuweisung nicht mehr gewährleistet werden kann.

Zu bedenken gilt, dass mittels sogenannter *Revers-Shells* Angreifer zu einem späteren Zeitpunkt trotz geänderter IP-Adresse wieder eine Verbindung zu ihren Containern erhalten könnten. Hier kann zukünftig das Aufräum-Script weiterentwickelt und mehr Fingerspitzengefühl implementiert werden.

Listing 4.7: Definition alter, verlassener Container

```
1 # Choose old container to kill
2 MAX_RUNTIME_SEC=$((60*60*12))
```

Das Alter der Container wird über den Docker Inspektor ermittelt. Dieser liest das Datum des Starts aus, welches dann in die Differenz zum aktuellen Zeitpunkt in Sekunden konvertiert wird.

Listing 4.8: Alter über Docker Inspektor ermitteln

```
1 STARTED_AT=$(docker inspect --format='{{ .State.StartedAt }}'  "${CID
    }")
2 RUNTIME_SEC=$(($(date +%s) - $(date -d "${STARTED_AT}" +%s)))
```

Die CPU-Zeit Filterung erkennt Honeynets anhand verbrauchter CPU-Ressourcen. Damit soll Malware identifiziert werden, welche eine definierte CPU-Last, beispielsweise zum Generieren von Kryptowährungen, verursacht hat. Zur Schonung des Host-Systems können diese Systeme gestoppt werden.

Listing 4.9: Einstellung der CPU-Zeit

```
1 # max cpuacct.usage_user per container.
2 # if its reached, the hole project will be stopped
3 # @see https://www.kernel.org/doc/Documentation/cgroup-v1/cpuacct.txt
4 # 1000000000 = 1 second
5 MAX_CPU_USER_USAGE=$((120*1000000000))
```

Ausgelesen wird die verbrauchte CPU-Zeit der Container dank den *cgroups* des Linux-Kernels:

Listing 4.10: Alter über cgroups auslesen

```
1 CPU_USER_USAGE=$(cat /sys/fs/cgroup/cpuacct/docker/${CID}/cpuacct.
    usage_user)
```

4.3.2 Server zur Aufzeichnung und Aufarbeitung der Logs

Als Hardware wird wie geplant ein NAS von QNAP verwendet, auf welchem der ELK-Stack in einer Komposition aus Docker-Containern Realisierung findet. Zur eindeutigen Zuordnung des Netzwerkverkehrs wird eine virtuelle Netzwerkkarte für diesen Container angelegt, wodurch dieser eine eigene MAC- und IP-Adresse erhalten wird.

Bei der QNAP NAS wird hierfür das benötigte Netzwerkinterface mit einer Eingabe über das CLI eingerichtet:

Listing 4.11: Docker-Netzwerk mit Anbindung an eth1 der NAS

```
1 docker network create -d qnet --ipam-driver=qnet --ipam-opt=iface=
    eth1 qnet-dhcp-eth1
```

Im Anschluss lässt sich die Netzwerkkonfiguration zur zugehörigen docker-compose.yaml übertragen:

Listing 4.12: Netzwerkzuweisung und MAC-Konfiguration

```
1 mac_address: ba:c8:e6:ba:08:ee
2 networks:
3   - qnet-dhcp-eth1
4 ...
5 networks:
6   qnet-dhcp-eth1:
7     external: true
```

Eine strikte Zuweisung der MAC-Adresse vereinfacht es, im DHCP-Dienst des Labor-Netzwerks eine IP-Adresse gleichbleibend zuzuweisen.

Vor der Inbetriebnahme des ELK-Stacks bedarf es noch die Erstellung eines Ordners, in welchem die Konfigurationen abgelegt werden. Die Pfade dieser Dateien finden sich ebenfalls in der *yaml* zum Logging-Server wieder, wie auch die Einrichtung des persistenten Speichers für das Datenbankmanagementsystem *elasticsearch*:

Listing 4.13: Konfigurationen und Speicher einbinden

```
1 volumes:
2   - /share/CACHEDEV1_DATA/hp-logging/elk-docker/elasticsearch.yml:/
      opt/elasticsearch/config/elasticsearch.yml:ro
3   - /share/CACHEDEV1_DATA/hp-logging/elk-docker/kibana.yml:/opt/
      kibana/config/kibana.yml:ro
4   - data-elk:/var/lib/elasticsearch
5 ...
6 volumes:
7   data-elk:
8     driver: local
```

Die finale Bereitstellung des Logging-Servers erfolgt durch Einfügen der docker-compose.yaml [A.2.1] in der Web-Oberfläche des Programms *ContainerStation* auf der NAS von wo aus auch die Komposition gestartet wird.

Nachdem das System hochgefahren ist, lässt sich die Oberfläche von Kibana innerhalb meines Labor-Netzwerks mit folgender URL im Browser öffnen. Abweichungen der IP-Adresse oder Port sind von der Umgebung und Konfiguration abhängig:

http://192.168.22.90:5601/app/kibana

Der Logging-Server ist nun bereit, mit dem Logging-Client zusammenzuarbeiten und die Daten entgegenzunehmen. Hierfür bedarf es nur noch einer Konfiguration des Logging-Clients: Die IP-Adresse des Logging-Servers muss ihm bekannt gegeben werden. Nachdem dies erfolgt ist und diese Einstellung zur Anwendung kommt, können wir im folgenden Kapitel 5 die Experimente durchführen.

5 Experimente

Mit der Umsetzung des Prototyps aus dem vorhergegangenen Kapitel 4 werden folgend einige Experimente durchgeführt und Regelwerke im Vorgehensmodell *Rapid Prototyping* entwickelt.

Innerhalb meines Labor-Netzwerks finden die Experimente mit simulierten Angriffen manuell durch mich statt. Als Grundlage dient ein aktuelles Android-Tablet mit dem SSH- und Telnet-Client *JuiceSSH* [12], sowie einem aktuellen Firefox-Browser zur Verbindungsaufnahme an den HTTP-Port des Honeynets. Bei dem Test auf die Unterstützung parallel betriebener Honeynet-Umgebungen für unterschiedliche Angriffsquellen zog ich mein Smartphone mit gleicher Konfiguration hinzu.

Das Honeynet soll, wie zuvor im Kapitel 3.1.6 erwähnt, ein Research-Honeypot werden. Entsprechend müssen die Falco-Regeln definiert oder Konfigurationen verwendeter Anwendungen angepasst werden.

Besonderes Interesse habe ich an folgenden Events und Eigenschaften:

1. Grundsätzlich immer der Zeitpunkt des jeweiligen Vorgangs
2. Projektname mit jeweiliger Container-ID und -Name für eine genaue Zuordnung
3. Anfragende IP-Adresse, um den Client zu kennen und diesen im weiteren Verlauf dem Projekt zuordnen zu können
4. Zugriffe auf Ports 22, 23 und 80, da diese durch das Honeynet bedient werden.
5. Anfragen an andere Ports - dies kann über weitere Pseudo-Dienste im xinetd realisiert werden oder außerhalb des Honeynets erfolgen
6. Gescheiterte sowie erfolgreiche Anmeldeversuche an SSHd und telnetd

7. Verbindungen zwischen dem Jumphost- und Web-Container an SSHd oder telnetd und wie beziehungsweise wodurch diese aufgebaut wurden
8. Verwendete oder gesuchte Programme sowie Laufzeitumgebungen des Angreifers beziehungsweise der Malware
9. Geplantes Stoppen von Honeynets nach einer definierten CPU- oder Laufzeit durch das automatisierte Aufräum-Script.

Diese Informationen sind innerhalb des Linux-Kernels und teilweise in den Log-Dateien von docker-compose zugänglich, jedoch bedarf es passender Regeldefinitionen zur Filterung dieser Daten. Die folgenden Kapitel gehen auf die Informationen aus dem Kernel und zusätzlichen Logs ein und erklären das jeweilige Vorgehen.

5.1 Falco Regeln zur Eventidentifizierung

Die interessanten Events, welche Falco beobachten soll, teilen sich eine Eigenschaft: Deren Container-ID weicht von der des Hosts ab. Dies trifft jedoch auch auf den Logging-Client zu, weshalb zusätzlich die übergreifende Teilzeichenkette *_web-* des Container-Namens berücksichtigt werden muss. Daraus lässt sich das Macro zur Erkennung von Honeypot-Containern definieren, welches bei weiteren Regeln Anwendung findet:

Listing 5.1: Falco Macro is_honeypot_container

```
1 - macro: is_honeypot_container
2   condition: container.id != host and container.name contains _web-
```

Die vollständigen Regelsätze sind in der Anlage A.1.1.8 enthalten.

5.2 Herkunft der Netzwerkanfragen

xinetd protokolliert neben der obligatorischen Angabe von Datum und Uhrzeit zusätzlich die Herkunft der Anfragen und welcher Service abgefragt wurde. Weiterhin ist ersichtlich, welche Prozess-ID gestartet wurde. Diese Information wird zur weiteren Beobachtung des Verlaufs dienlich sein. Nach dem Verbindungsende wird zusätzlich die aktive Sitzungsdauer mit angegeben.

Listing 5.2: Absender-IP und angefragter Dienst

```
20/7/30@17:33:27: START: hp_web pid=15969 from=::ffff:192.168.22.74
20/7/30@17:33:33: START: hp_ssh pid=16284 from=::ffff:192.168.22.74
20/7/30@17:33:41: START: hp_telnet pid=16326 from=::ffff
    :192.168.22.74
20/7/30@17:34:11: EXIT: hp_telnet status=0 pid=16326 duration=30(sec)
20/7/30@17:34:14: EXIT: hp_ssh status=0 pid=16284 duration=41(sec)
20/7/30@17:34:35: EXIT: hp_web status=0 pid=15969 duration=68(sec)
```

Standardmäßig ist dieses Logging jedoch nicht eingestellt. Zur Konfiguration werden lediglich 3 Zeilen in der *xinetd.conf* [A.1.2.3] benötigt:

Listing 5.3: xibetd.conf Logging Optionen

```
log_type        =  FILE /var/log/xinetdlog
log_on_success  =  PID USERID HOST DURATION EXIT
log_on_failure  =  USERID HOST ATTEMPT
```

Die angegebene Datei muss zusätzlich als weiteren Input im Filebeat des Logging-Clients konfiguriert werden, welcher die generierten Einträge dem ELK-Stack überträgt:

Listing 5.4: Filebeat inputs

```
filebeat.inputs:
  - type: log
    paths:
      - /var/log/host/dockercompose_* # docker-compose logs and
        custom messages from project
      - /var/log/host/xinetdlog # xinetd
```

5.3 Verwendete Benutzernamen und Kennwörter

Dieser Prototyp soll primär die Verbreitungswege von Angreifern oder Malware aufzeichnen, wofür es genügt, die Versuche von Anmeldungen aufzuzeigen und ob diese erfolgreich waren oder nicht. Die jeweils verwendeten Passwörter der Anmeldeversuche können mit einem Pluggable Authentication Module (PAM) für den openssh-server oder durch patchen und neuübersetzen des Quellcodes zuverlässig protokolliert werden [22, 23].

5.4 Verbreitungswege

Schadsoftware und Angreifer können sehr einfallsreich sein, wenn es um die Ausbreitung geht. Ein einfallsloser Weg wäre das Absuchen der Umgebenen mit anschließendem Portscan um bereitgestellte Dienste zu erahnen, für welche es möglicherweise bekannte Schwachstellen gibt. Hilfreich sind hier Tools wie *nmap*, welches ein Bild der Netzwerkumgebung erstellen kann. Doch ist dies eines der bekanntesten Vorgänge und dürfte in jedem Regelsatz für IDS bereits enthalten sein. Zusätzlich sollte *nmap* auf keinem Produktivsystem installiert und zugänglich sein. Alternativ sind die Scans auch mit eigenen Scripten möglich oder könnten manuell durchgeführt werden. Hier beginnt der spannende Teil: die Aufzeichnung unbekannter Verbreitungswege - beziehungsweise die Analyse der Wege anhand protokollierter Daten. Hierbei ist es von Interesse, welche Verbreitungswege sich erfolgreich zeigten sowie die gestarteten Versuche sich auszubreiten und andere Systeme anzugreifen.

Zusätzliche Werkzeuge wie *telnet*, *ssh*, *wget* oder *curl* sowie Laufzeitumgebungen für eigenen Programmcode sind potenziellen Angreifern und der Malware hilfreich, weil sich diese und ähnliche Programme auf Verbindung zu weiteren Netzwerkpunkten spezialisiert haben. Bei der Auswertung von aufgezeichneten Daten kann die Verwendung solcher Werkzeuge als Indiz auf den Verbreitungsweg dienen und sollte nicht unbeachtet bleiben.

Wie stehen diese Werkzeuge zur Verfügung? Sofern der Schädling Programmcode als privilegierten Nutzer ausführen kann, steht der Installation weiterer Programme nicht mehr viel im Weg. Einfacher könnte das Nachladen von eigenem Programmcode über Hilfsprogramme oder gar einer Remote-Shell sein.

Zur Überwachung dieser Tätigkeiten setze ich auf ein nicht zu detailliertes Regelwerk von Falco, wodurch alle Befehlseingaben und aufgerufene Programme samt Argumente erfasst und protokolliert werden.

Hier kann für eine zukünftige Weiterentwicklung nach einem Studium des Benutzerhandbuchs von Falco großes Potenzial für eine detailliertere Überwachung und beschleunigte Analyse ermöglicht werden.

5.5 Parallelbetrieb von Honeynets

Zur Überprüfung der Vorgabe aus Kapitel 3.1.2, dass mehrere Instanzen parallel betrieben werden können, habe ich mit Verbindungsanfragen durch einen weiteren Netzwerk-Client an den Honeypot eine weitere Honeynet-Instanz durch den Superserver starten lassen. Dass diese zusätzliche Umgebung gestartet wurde, zeigt ein Aufruf von *docker ps* in der Shell der VM auf. Zur besseren Darstellung wurde die folgende Ausgabe um die Spalten *COMMAND*, *CREATED* und *PORTS* reduziert sowie IDs gekürzt.

Listing 5.5: Aktive Docker Container

```
1 # docker ps
2 C.ID      IMAGE                 STATUS        NAMES
3 a...7     debian-ssh            Up 8 min.     52a..._web-jumphost
4 4...c     nginx-ssh-telnet      Up 8 min.     52a..._web-nginx
5 a...d     debian-ssh            Up 19 min.    560..._web-jumphost
6 c...6     nginx-ssh-telnet      Up 19 min.    560..._web-nginx
7 4...c     filebeat762           Up 2 weeks    fb
8 b...a     falcosecurity/falco   Up 2 weeks    falco
9 5...b     metricbeat762         Up 2 weeks    mb
```

Es ist zu erkennen, dass seit 2 Wochen die drei Container zum Logging-Client betrieben werden sowie die darüber gelisteten zwei Honeynet-Instanzen. Die jüngere Instanz wurde 8 Minuten zuvor gestartet und der Hashwert der anfragenden IP-Adresse beginnt mit *52a* wohingegen die ältere Instanz bereits seit 19 Minuten eingeschaltet ist und dessen IP-Hashwert mit *560* beginnt.

5.6 Ermittelte Daten

Nachdem die simulierten Angriffe durchgeführt wurden, beginnt in Kibana die Analyse der ermittelten Logs. Hierzu wird folgend die Möglichkeit der Filterung im *Kibana Discover* gezeigt und anschließend Bildschirmfotos der jeweils interessanten Events.

5.6.1 Filter

Es werden praktische Filter angeboten, wo direkt die Quelle der Aufzeichnungen wie Beispielsweise Falco, xinetd oder die zum Honeynet-Projekt gehörende dockercompose.log

mit dem Projektnamen als Suffix. Aufgrund der weitgreifenden Falco-Regeln beinhaltet dessen Log mit Abstand am meisten Einträge:

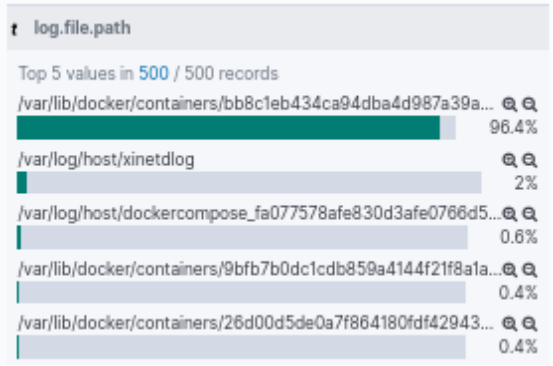

Abbildung 5.1: Filter nach Log-Quelle

Zur Übersichtsverbesserung innerhalb der Falco-Logs, bietet sich eine Filterung nach den jeweiligen Regel-Bezeichnungen an, welche in der Abbildung 5.2 gezeigt werden.

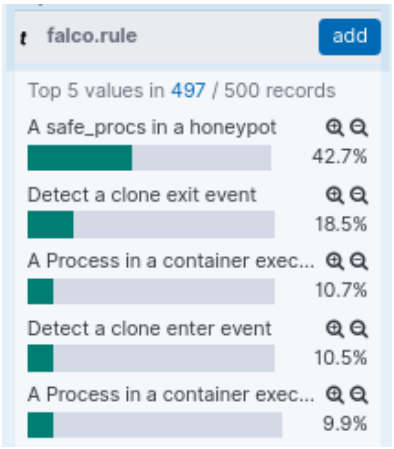

Abbildung 5.2: Filter nach Falco-Regel

5.6.2 Installation zusätzlicher Anwendungen

Im folgenden Auszug ist ersichtlich, dass über apt das Programm *wget* installiert wurde. Weiterhin ist ersichtlich, dass apt einen Unterprozess von *dpkg* gestartet und weitere Parameter übergeben hat:

```
Detect    2020-08-07T08:36:42.929633966+0000: Notice container.info=(52a859528cb260161c1648303c69dd4b_web-jumphost (id=3069cb072533)) evt.nu
a clone   m=357884535 evt.cpu=0 proc.name=dpkg proc.cmdline=(dpkg --status-fd 15 --no-triggers --unpack --auto-deconfigure --recursive /tmp/
enter e   apt-dpkg-install-eOWPmU) proc.pcmdline=(apt install wget -y) (thread.tid=1875) evt.info=() evt.dir=> evt.type=clone evt.args=()
vent
```

Abbildung 5.3: Installation von wget

Abstrakt betrachtet erschließt sich aus der Abbildung 5.3, dass die Verwendung von Unterprozessen aus Malware heraus erkannt und zusammen mit dem ausführenden Prozess protokolliert wird.

5.6.3 Herunterladen von Malware

Im Anschluss der Installation von *wget* wurde dieses Programm eingesetzt, um mögliche Schadsoftware herunterzuladen. Auch dies kann den Log-Einträgen von Falco entnommen werden:

```
Write b   2020-08-07T08:36:48.681894606+0000: Error File below / or /root opened for writing (user=root command=wget hackertools.org/script.
elow ro   sh parent=bash file=/root/script.sh program=wget container_id=3069cb072533 image=debian-ssh)
ot
```

Abbildung 5.4: Script heruntergeladen

Weiterhin belegt dieser Eintrag, dass Veränderungen am Dateisystem mitsamt des verwendeten Programms erkannt werden können.

5.6.4 Ausbreitung

Um sicherzustellen, dass interne Ausbreitungen aufgezeichnet werden können, habe ich eine Telnet-Verbindung vom Jumphost- auf den Web-Container durchgeführt. Verwendet wurde dazu der Benutzername *web* mit dem einfach zu erratendem Passwort *123456*.

Wie erwartet, war auch dies zweimal zu erkennen: Einmal vom Jumphost-Container, zu dargestellt in Abbildung 5.5 und anschließend auf dem Web-Container, welches der Abbildung 5.6 entnommen werden kann.

```
A safe_   2020-08-07T08:04:10.429922368+0000: Warning container.info=(fd2bca8fb929de148c72d3ef57f6c037_web-jumphost (id=c2acdbd643e1)) evt.n
procs i   um=345650301 evt.cpu=0 proc.name=login proc.cmdline=(login -h 172.25.0.1 -p) proc.pcmdline=(in.telnetd) proc.exe=login (thread.tid
n a hon   =32024) evt.info=(res=4 data=web. ) evt.dir=< evt.type=read evt.args=(res=4 data=web. )
eypot
```

Abbildung 5.5: Telnet vom Jumphost

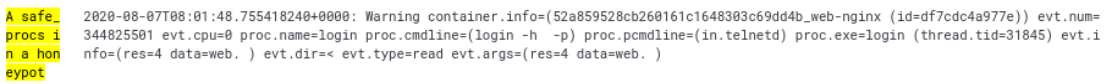

Abbildung 5.6: Telnet am Web-Container

5.6.5 Herkunft der Anfragen

Die xinetd-Logs offenbaren den Ursprung der Netzwerkverbindungen, den angefragten Service und bei Beendigung die Angabe zur Sitzungszeit. Hier kann die Herkunft *192.168.22.83* ausgelesen werden; es wurde der SSH-Dienst angefragt und für 6 Sekunden verbunden.

Abbildung 5.7: Logs von xinetd

5.6.6 Erkennung von Malware oder Angreifern

Mit der Protokollierung von Verbindungsaufbauten angefangen lassen sich Angriffe bereits erahnen, weil Netzwerkanfragen an einen Honeypot nicht zum operativen Netzwerksystem zuordenbar sein können. Ausnahmen bilden jedoch mögliche Fälle, wo Menschen Falscheingaben oder fehlerhafte Konfigurationen durchführen, wodurch Netzwerkanfragen durchaus zu einem Honeypot geleitet werden könnten. Zur Unterscheidung ist eine weitere Begutachtung der aufgezeichneten Logs erforderlich, denn weiteres Analysieren der Daten zur Verbindung erlaubt die Erhärtung des Anfangsverdachts oder lassen diesen fallen.

Nach der Feststellung, dass es sich bei der jeweiligen Verbindung um einen Angriff gehandelt hat, ist eine weitreichende Analyse der aufgezeichneten Daten erforderlich und

wird offenbaren, wie der Angriff inhaltlich vonstattenging und ob beziehungsweise wie eine Ausbreitung versucht oder gar erfolgreich durchgeführt wurde.

Die aufgezeichneten Daten stimmen mich positiv, dass auch echte Angriffe gleichermaßen festgehalten werden und Analysen ermöglichen.

6 Zusammenfassung

Zum Ende der Thesis erörtere ich einige Ausbaumöglichkeiten im Ausblick und schließe mit einem kurzen Fazit die Arbeit ab.

6.1 Ausblick

Diese Arbeit ist, wie nahezu alle IT-Projekte, nicht abschließend und ist offen für Ergänzungen. Aufbauend hierauf sind einige Erweiterungen denkbar. Server werden üblicherweise mittels Domain Name System (DNS) erreicht, hier könnte diese Architektur um Domains erweitert werden. Daraufhin sind Zertifikate für gesicherte HTTPS-Verbindungen an Port 443 möglich, welche wiederum neue Beobachtungsaspekte bieten und überwacht werden möchten. Die Bereitstellung von Zertifikaten könnte automatisiert erfolgen, beispielsweise von Anbietern wie Let's Encrypt [13]. Diese Automatisierung ermöglicht es, die Zertifikate stets aktuell zu halten damit diese nicht über das Ablaufdatum hinaus betrieben werden.

Weitere Protokolle wie Beispielsweise das Network Time Protocol (NTP) oder Simple Mail Transfer Protocol (SMTP) könnten ebenfalls eingebunden werden. Für jedes weitere Protokoll müssen xinetd, die docker-compose.yml des Honeynet und das Regelwerk von Falco erweitert werden. Zugleich sind, ideal jeweils, eigene Docker-Images einzurichten. Die Trennung in separate Images, welche in eigenen Containern ausgeführt werden, garantiert die sichere Zuordnung aus den protokollierten Daten. Um die Auswertung zu erleichtern, sollten die Container mit sinnvollen Namen bezeichnet werden.

Denkbar ist es auch, anstelle der Realimplementierung lediglich die weiteren Dienste in der Konfiguration von xinetd einzubinden. Dies ermöglicht die Protokollierung von Verbindungsversuchen dieser Dienste, welche eine interessante Information darstellen kann.

IPv6 könnte Verwendung finden, indem die jeweiligen Dienste entsprechend konfiguriert werden. Zu beachten ist, dass die Multicast-Adressen für reservierte Dienste zu einer Erhöhung der False-Positive-Verbindungen führen kann, sofern ein solcher Dienst betrieben wird. Dass die VM und das umgebene Netzwerk ebenfalls die Unterstützung von IPv6 bieten muss, sollte jedem selbstverständlich sein. Komplett andere Protokolle wie beispielsweise IPX sind bei diesem Honeypot nicht vorgesehen und dürften, in Abhängigkeit zur jeweiligen Technologie, unterschiedlich komplex zu implementieren sein.

Die Auswertungen der Daten könnten vereinfacht werden, indem die Oberfläche von Kibana mit einem eigenen Dashboard erweitert wird. Alternativ sind weitere Auswertungs- und Analyseprogramme denkbar, welche direkt auf die dokumentenbasierte NoSQL-Datenbank des Datenbankmanagementsystems *Elasticsearch* zugreifen können. Da auch der Logging-Server containervirtualisiert ist, sollten die zusätzlichen Anwendungen ebenfalls dockerisierbar sein, damit sie lediglich mithilfe der docker-compose.yml des Logging-Servers hinzugefügt werden können.

Weitere Daten können gesammelt werden, indem die jeweiligen Home-Verzeichnisse der Benutzer auf einem eingebundenen Docker-Volume angelegt werden. Hieraus bietet sich die Möglichkeit der Überwachung von History-Einträgen, wie sie durch viele Shell-Programme standardmäßig generiert werden.

Ein detaillierteres Logging nach einem Studium des Benutzerhandbuchs von Falco kann den Umfang an zu speichernden Daten sowie die Analyse optimieren. Falco ist sehr komplex und mächtig, wodurch sich weitreichende Kenntnisse der Details von Regeldefinitionen als interessant erweisen.

Virtuelle Maschinen könnten automatisiert auf Ebene des Hypervisors gestoppt oder vom Netzwerk getrennt werden, sobald oder kurz bevor erfolgreiche Angreifer oder Malware aus ihrer Honeynet-Umgebung ausbrechen. Dies kann mit Schnittstellen der jeweiligen Virtualisierungslösung realisiert werden, worüber auch ein anschließender Start von einer neuen Honeypot-VM ermöglicht wird.

6.2 Fazit

Resümierend bin ich mit dem Prototyp sehr zufrieden. Dieser kann dank der Virtualisierung und Verwendung von Containerformaten einfach portiert werden und mit der zentralen docker-compose.yml sind Erweiterungen und Anpassungen ohne große Programmierkenntnisse möglich.

Aufgezeichnete Daten sind über die standardisierte Oberfläche von Kibana schnell zugänglich und können von entsprechenden Spezialisten gefiltert sowie ausgewertet werden. Aufgrund der Verwendung einer standardisierten Datenbanklösung können die Dokumente exportiert und extern weiterverarbeitet werden.

Einer praktischen Anwendungen des Honeynets oder Logging-Servers sehe ich positiv entgegen und werde dies wohl in naher Zukunft realisieren wollen.

Zur Verbesserung der Sicherheit des Honeypotbetriebs interessiert mich bei einer Weiterentwicklung am ehesten die automatische Steuerung von VMs. Es bedarf einer Konzeption zur Erkennung eines Ausbruchs aus einer Honeynet-Umgebung und anschließenden Aktionen wie beispielsweise die Erzeugung eines Snapshots mit folgendem Herunterfahren der VM. Im Anschluss wird wohl eine neue VM gestartet werden müssen oder eine als *Hot Spare* im Hintergrund laufende VM umkonfiguriert und online gestellt.

Weiterhin denke ich zur besseren Verschleierung des Honeynets den Web-Container mit HTTPS auszustatten, denn ein Web-Auftritt ohne Verwendung dieses verschlüsselnden Protokolls ist heutzutage eher unüblich: 100 % der Top-100-Websites nach Google bieten dieses Protokoll an, davon 96 % standardmäßig [10]. Es wird eine Konzeption erfordern, wie die Zertifikate erstellt und gegebenenfalls beglaubigt werden. Die glaubwürdige Beglaubigung von Zertifikaten setzt eine zeitgleiche oder vorhergegangene Implementierung von Domain Namen und DNS voraus, wodurch dies ebenfalls eine priorisierte Erweiterung darstellt.

Literaturverzeichnis

[1] : *AMD Virtualisierungslösungen.* – URL https://www.amd.com/de/technologies/virtualization-solutions

[2] : *Apache License 2.0.* – URL https://www.apache.org/licenses/LICENSE-2.0

[3] : *Argus.* – URL https://openargus.org/

[4] : *Cisco Talos.* – URL https://www.cisco.com/c/en/us/products/security/talos.html

[5] : *Clop Ransomware.* – URL https://www.mcafee.com/blogs/other-blogs/mcafee-labs/clop-ransomware/

[6] : *Debian Buster Slim.* – URL https://packages.debian.org/buster/slim

[7] : *Docker.* – URL https://www.docker.com/

[8] : *ELK-Stack from sebp on docker hub.* – URL https://hub.docker.com/r/sebp/elk

[9] : *ET Pro Ruleset.* – URL https://www.proofpoint.com/us/threat-insight/et-pro-ruleset

[10] : *HTTPS-Verschlüsselung im Web.* – URL https://transparencyreport.google.com/https/overview

[11] : *Intel® Virtualisierungstechnik (Intel® VT).* – URL https://www.intel.de/content/www/de/de/virtualization/virtualization-technology/intel-virtualization-technology.html

[12] : *JuiceSSH.* – URL https://juicessh.com/

[13] : *Let's Encrypt.* – URL https://letsencrypt.org

[14] : *Linux Containers - LXD - Introduction.* – URL https://linuxcontainers.org/lxd/introduction/

[15] : *Moby License.* – URL https://github.com/moby/moby/blob/master/LICENSE

[16] : *Moby Project.* – URL https://mobyproject.org/

[17] : *net-tools debian package.* – URL https://packages.debian.org/de/buster/net-tools

[18] : *Number of connected devices reached 22 billion, where is the revenue?.* – URL https://www.helpnetsecurity.com/2019/05/23/connected-devices-growth/

[19] : *openssh-server debian package.* – URL https://packages.debian.org/buster/openssh-server

[20] : *OpenStack.* – URL https://www.openstack.org/

[21] : *p0f.* – URL https://lcamtuf.coredump.cx/p0f3/

[22] : *PAM module in Python for openssh-server.* – URL https://superuser.com/a/963509

[23] : *patch for openssh 7.2.* – URL https://superuser.com/a/1500404

[24] : *Release T-Pot 19.03.* – URL https://dtag-dev-sec.github.io/mediator/feature/2019/04/01/tpot-1903.html

[25] : *RFC 4949.* – URL https://tools.ietf.org/html/rfc4949

[26] : *roo iso download.* – URL https://www.honeynet.org/projects/old/honeywall-cdrom/

[27] : *Sebek.* – URL https://github.com/honeynet/sebek

[28] : *Snort.* – URL https://www.snort.org/

[29] : *Snort FAQ.* – URL https://snort.org/faq/what-does-having-a-snort-subscriber-rule-set-subscription-entitle-me-to

[30] : *Suricata.* – URL https://suricata-ids.org/features/

[31] : *T-Pot: A Multi-Honeypot Platform.* – URL https://dtag-dev-sec.github.io/mediator/feature/2015/03/17/concept.html

[32] : *telnetd debian package*. – URL https://packages.debian.org/buster/telnetd

[33] : *Ubuntu Releases*. – URL http://releases.ubuntu.com/

[34] : *VirtualBox Changelog*. – URL https://www.virtualbox.org/wiki/Changelog

[35] : *What is a honeypot?*. – URL https://www.kaspersky.com/resource-center/threats/what-is-a-honeypot

[36] : *What's new with the Internet of Things?*. – URL https://www.mckinsey.com/industries/semiconductors/our-insights/whats-new-with-the-internet-of-things

[37] : *xinetd-org / xinetd on github.com*. – URL https://github.com/xinetd-org/xinetd

[38] Hilbig, Heino: *Zugaben*. S. 185–191. In: *Zukunftsmanagement für den Mittelstand : Was Sie tun können und was Sie besser lassen sollten, um auch morgen noch im Geschäft zu sein*. Wiesbaden : Springer Fachmedien Wiesbaden, 2018. – URL https://doi.org/10.1007/978-3-658-19616-5_7. – ISBN 978-3-658-19616-5

[39] Kelm, Stefan: Verlockend. In: *iX* 1 (2006), S. 74–80. – URL https://heise.de/-221592

[40] Stevens, R. ; Pohl, H.: Honeypots und Honeynets. In: *Informatik-Spektrum* 27 (2004), Jun, Nr. 3, S. 260–264. – URL https://doi.org/10.1007/s00287-004-0404-y. – ISSN 1432-122X

[41] Symantec: *W32.Stuxnet Dossier*. 2010. – URL https://community.broadcom.com/symantecenterprise/communities/community-home/librarydocuments/viewdocument?DocumentKey=1a29f93b-41f8-4265-a7d1-44c3c94d3b52&CommunityKey=1ecf5f55-9545-44d6-b0f4-4e4a7f5f5e68&tab=librarydocuments

A Anhang

A.1 Honeypot-Server

A.1.1 Honeynet

A.1.1.1 Honeynet docker-compose.yaml

```
1  version: "3.7"
2  services:
3    jumphost:
4      build:
5        context: ../Images/jumphost
6        args:
7          - ROOT_PW=admin
8          - USER_LIST=admin,administrator,cms,files,ftp,guest,portal,
               user,web,wordpress,www
9          - USER_PW=123456
10     image: debian-ssh
11     container_name: ${PROJECT_NAME:-noname}_web-jumphost
12     labels:
13       dmz: "1" # needed to filter in cleanup.sh
14     logging: # https://docs.docker.com/config/containers/logging/json-
           file/
15       driver: "json-file"
16       options:
17         max-size: "5m"
18         max-file: "1"
19     networks:
20       - dmznet
21     ports:
22       - "22"
23       - "23"
```

```
24      restart: "no"
25    web:
26      build:
27        context: ../Images/nginx-ssh-telnet
28        args:
29          - ROOT_PW=admin
30          - USER_LIST=admin,administrator,cms,files,ftp,guest,portal,
              user,web,wordpress,www
31          - USER_PW=123456
32      image: nginx-ssh-telnet
33      container_name: ${PROJECT_NAME:-noname}_web-nginx
34      labels:
35          dmz: "1" # needed to filter in cleanup.sh
36      logging: # https://docs.docker.com/config/containers/logging/json-
            file/
37        driver: "json-file"
38        options:
39          max-size: "5m"
40          max-file: "1"
41      networks:
42        - dmznet
43      ports:
44        - "80"
45        - "443"
46      restart: "no"
47
48  networks:
49    dmznet:
50      driver: bridge
```

A.1.1.2 Jumphost Dockerfile

```
1  FROM debian:buster-slim
2  ARG ROOT_PW
3  ARG USER_LIST
4  ARG USER_PW
5
6  RUN apt-get update && apt-get install -y openssh-server telnetd net-
       tools
7
8  # Setup ssh
```

```
9  RUN mkdir /var/run/sshd
10 RUN sed -i 's/#PermitRootLogin.*/PermitRootLogin\ yes/' /etc/ssh/
       sshd_config
11
12 # Login logging https://linoxide.com/security/enable-sshd-logging/ -
       test DEBUG3
13 RUN sed -i 's/#LogLevel.*/LogLevel\ VERBOSE/' /etc/ssh/sshd_config
14 RUN sed -i 's/#SyslogFacility.*/SyslogFacility\ AUTH/' /etc/ssh/
       sshd_config
15
16 # SSH login fix. Otherwise user is kicked off after login
17 RUN sed -i 's@session\s*required\s*pam_loginuid.so@session optional
       pam_loginuid.so@g' /etc/pam.d/sshd
18 ENV NOTVISIBLE 'in users profile'
19 RUN echo 'export VISIBLE=now' >> /etc/profile
20
21 # Set root password
22 RUN echo "root:${ROOT_PW:-admin}" | chpasswd
23
24 # Create some users
25 RUN bash -c 'for username in  ${USER_LIST//,/ } ; do useradd -m "${
       username}" && echo "${username}:${USER_PW:-nonsecure}" | chpasswd
       ; done'
26
27 # Add welcome message
28 RUN echo 'Welcome to the DMZ jumphost' > /etc/motd
29
30 EXPOSE 22
31 EXPOSE 23
32
33 COPY start.sh /start.sh
34 RUN chmod 500 /start.sh
35 ENTRYPOINT ["/start.sh"]
```

A.1.1.3 Jumphost start.sh

```
1 #!/usr/bin/env bash
2
3 service ssh start
4 /etc/init.d/openbsd-inetd restart
5
```

```
6 /bin/bash -c "trap : TERM INT; sleep infinity & wait"
```

A.1.1.4 Web Dockerfile

```
1  FROM nginx:stable
2  ARG ROOT_PW
3  ARG USER_LIST
4  ARG USER_PW
5
6  # Make website editable for users
7  RUN chmod -R +w /usr/share/nginx
8
9  # for UBUNTU: https://docs.docker.com/engine/examples/
       running_ssh_service/
10 RUN apt-get update && apt-get install -y openssh-server telnetd net-
       tools
11
12 # Setup ssh
13 RUN mkdir /var/run/sshd
14 RUN sed -i 's/#PermitRootLogin.*/PermitRootLogin\ yes/' /etc/ssh/
       sshd_config
15 # SSH login fix. Otherwise user is kicked off after login
16 RUN sed -i 's@session\s*required\s*pam_loginuid.so@session optional
       pam_loginuid.so@g' /etc/pam.d/sshd
17 ENV NOTVISIBLE 'in users profile'
18 RUN echo 'export VISIBLE=now' >> /etc/profile
19
20 # Set root password
21 RUN echo "root:${ROOT_PW:-admin}" | chpasswd
22
23 # Create some users
24 RUN bash -c 'for username in  ${USER_LIST//,/ } ; do useradd -m "${
       username}" && echo "${username}:${USER_PW:-nonsecure}" | chpasswd
       ; done'
25
26 # Add welcome message
27 RUN echo 'Welcome to the web server. Document root is /usr/share/
       nginx/html' > /etc/motd
28
29 EXPOSE 22
30 EXPOSE 23
```

```
31 EXPOSE 80
32
33 COPY start.sh /start.sh
34 RUN chmod 500 /start.sh
35 ENTRYPOINT ["/start.sh"]
```

A.1.1.5 Web start.sh

```
1 #!/usr/bin/env bash
2
3 service ssh start
4 /etc/init.d/openbsd-inetd restart
5 nginx -g "daemon off;"
```

A.1.1.6 Logging-Client docker-compose.yaml

```
1  version: "3.7"
2  services:
3    falco:
4      image: falcosecurity/falco
5      command: ["/usr/bin/falco", "-A"]
6      container_name: falco
7      privileged: true
8      restart: always
9      volumes:
10       - ../Images/falco-docker-stable/honeypot-rules.yml:/etc/falco/
           falco_rules.local.yaml:ro
11       - ../Images/falco-docker-stable/falco.yml:/etc/falco/falco.yaml
           :ro
12       - /var/run/docker.sock:/host/var/run/docker.sock:ro
13       - /dev:/host/dev
14       - /proc:/host/proc:ro
15       - /boot:/host/boot:ro
16       - /lib/modules:/host/lib/modules:ro
17       - /usr:/host/usr:ro
18   filebeat:
19     build: ../Images/filebeat
20     image: filebeat762
21     container_name: fb
22     environment:
23       - ELASTICSEARCH_HOST=192.168.22.90
```

```
24        - ELASTICSEARCH_PORT=9200
25      privileged: true
26      restart: always
27      user: root:root
28      volumes:
29        - /var/log:/var/log/host:ro
30        - /var/lib/docker/containers:/var/lib/docker/containers:ro
31        - /var/run/docker.sock:/var/run/docker.sock:ro
32    metricbeat:
33      build: ../Images/metricbeat
34      image: metricbeat762
35      container_name: mb
36      environment:
37        - ELASTICSEARCH_HOST=192.168.22.90
38        - ELASTICSEARCH_PORT=9200
39      privileged: true
40      restart: always
41      volumes:
42        - /:/hostfs:ro
43        - /var/run/docker.sock:/var/run/docker.sock:ro
```

A.1.1.7 falco.yml

```
1  #
2  # Copyright (C) 2019 The Falco Authors.
3  #
4  #
5  # Licensed under the Apache License, Version 2.0 (the "License");
6  # you may not use this file except in compliance with the License.
7  # You may obtain a copy of the License at
8  #
9  #     http://www.apache.org/licenses/LICENSE-2.0
10 #
11 rules_file:
12   - /etc/falco/falco_rules.yaml
13   - /etc/falco/falco_rules.local.yaml
14   - /etc/falco/rules.d
15
16 time_format_iso_8601: true
17
18 # Whether to output events in json or text
```

```
19 json_output: true
20
21 # When using json output, whether or not to include the "output"
      property
22 # itself (e.g. "File below a known binary directory opened for
      writing
23 # (user=root ....") in the json output.
24 json_include_output_property: true
25
26 # Send information logs to stderr and/or syslog Note these are *not*
      security
27 # notification logs! These are just Falco lifecycle (and possibly
      error) logs.
28 log_stderr: true
29 log_syslog: true
30
31 # Minimum log level to include in logs. Note: these levels are
32 # separate from the priority field of rules. This refers only to the
33 # log level of falco's internal logging. Can be one of "emergency",
34 # "alert", "critical", "error", "warning", "notice", "info", "debug".
35 log_level: info
36
37 # Minimum rule priority level to load and run. All rules having a
38 # priority more severe than this level will be loaded/run.  Can be
      one
39 # of "emergency", "alert", "critical", "error", "warning", "notice",
40 # "info", "debug".
41 priority: debug
42
43 # Whether or not output to any of the output channels below is
44 # buffered. Defaults to false
45 buffered_outputs: false
46
47 # Falco uses a shared buffer between the kernel and userspace to pass
48 # system call information. When falco detects that this buffer is
49 # full and system calls have been dropped, it can take one or more of
50 # the following actions:
51 #   - "ignore": do nothing. If an empty list is provided, ignore is
      assumed.
52 #   - "log": log a CRITICAL message noting that the buffer was full.
```

```
53 #   - "alert": emit a falco alert noting that the buffer was full.
54 #   - "exit": exit falco with a non-zero rc.
55 #
56 # The rate at which log/alert messages are emitted is governed by a
57 # token bucket. The rate corresponds to one message every 30 seconds
58 # with a burst of 10 messages.
59
60 syscall_event_drops:
61   actions:
62     - log
63     - alert
64   rate: .03333
65   max_burst: 10
66
67 # A throttling mechanism implemented as a token bucket limits the
68 # rate of falco notifications. This throttling is controlled by the
     following configuration
69 # options:
70 #  - rate: the number of tokens (i.e. right to send a notification)
71 #    gained per second. Defaults to 1.
72 #  - max_burst: the maximum number of tokens outstanding. Defaults to
     1000.
73 #
74 # With these defaults, falco could send up to 1000 notifications
     after
75 # an initial quiet period, and then up to 1 notification per second
76 # afterward. It would gain the full burst back after 1000 seconds of
77 # no activity.
78
79 outputs:
80   rate: 1
81   max_burst: 1000
82
83 # Where security notifications should go.
84 # Multiple outputs can be enabled.
85
86 syslog_output:
87   enabled: true
88
89 # If keep_alive is set to true, the file will be opened once and
```

```
90  # continuously written to, with each output message on its own
91  # line. If keep_alive is set to false, the file will be re-opened
92  # for each output message.
93  #
94  # Also, the file will be closed and reopened if falco is signaled
        with
95  # SIGUSR1.
96
97  file_output:
98    enabled: false
99    keep_alive: false
100   filename: ./events.txt
101
102 stdout_output:
103   enabled: true
104
105 # Falco contains an embedded webserver that can be used to accept K8s
106 # Audit Events.
107 webserver:
108   enabled: false
109   listen_port: 8765
110   k8s_audit_endpoint: /k8s_audit
111   ssl_enabled: false
112   ssl_certificate: /etc/falco/falco.pem
113
114 # Also, the program will be closed and reopened if falco is signaled
        with
115 # SIGUSR1.
116 program_output:
117   enabled: false
118   keep_alive: false
119   program: "jq '{text: .output}' | curl -d @- -X POST https://hooks.
          slack.com/services/XXX"
120
121 http_output:
122   enabled: false
123   url: http://some.url
124
125 # gRPC server configuration.
```

```
126 # The gRPC server is secure by default (mutual TLS) so you need to
      generate certificates and update their paths here.
127 # By default the gRPC server is off.
128 # You can configure the address to bind and expose it.
129 # By modifying the threadiness configuration you can fine-tune the
      number of threads (and context) it will use.
130 grpc:
131   enabled: false
132   bind_address: "0.0.0.0:5060"
133   threadiness: 8
134   private_key: "/etc/falco/certs/server.key"
135   cert_chain: "/etc/falco/certs/server.crt"
136   root_certs: "/etc/falco/certs/ca.crt"
137
138 # gRPC output service.
139 # By default it is off.
140 # By enabling this all the output events will be kept in memory until
      you read them with a gRPC client.
141 grpc_output:
142   enabled: false
```

A.1.1.8 honeypot-rules.yml

```
1
2 - list: known_binaries
3   items: [shell_binaries, userexec_binaries, user_mgmt_binaries]
4
5 - list: remote_console
6   items: [sshd, telnetd, in.telnetd]
7
8 - macro: safe_procs
9   condition: proc.name in (known_binaries)
10
11 - macro: is_honeypot_container
12   condition: container.id != host and container.name contains _web-
13
14 # Shell commands
15 - rule: A safe_procs in a honeypot
16   desc: A safe_procs in a honeypot
17   condition: evt.type=read and is_honeypot_container and safe_procs
```

```
18    output: container.info=(%container.info) evt.num=%evt.num evt.cpu=%
          evt.cpu proc.name=%proc.name proc.cmdline=(%proc.cmdline) proc.
          pcmdline=(%proc.pcmdline) proc.exe=%proc.exe (thread.tid=%thread
          .tid) evt.info=(%evt.info) evt.dir=%evt.dir evt.type=%evt.type
          evt.args=(%evt.args)
19    tags: [customrule]
20    priority: WARNING
21  # program starts
22  - rule: Program in honeypot starts output
23    desc: Program in honeypot starts output
24    condition: evt.type=execve and is_honeypot_container and evt.dir
          contains > and not proc.name contains udevadm
25    output: container.info=(%container.info) evt.num=%evt.num evt.cpu=%
          evt.cpu proc.name=%proc.name proc.cmdline=(%proc.cmdline) proc.
          pcmdline=(%proc.pcmdline) proc.exe=%proc.exe (thread.tid=%thread
          .tid) evt.info=(%evt.info) evt.dir=%evt.dir evt.type=%evt.type
          evt.args=(%evt.args)
26    tags: [customrule]
27    priority: INFO
28  - rule: Program in honeypot starts input
29    desc: Program in honeypot starts input
30    condition: evt.type=execve and is_honeypot_container and evt.dir
          contains < and not proc.name contains udevadm
31    output: container.info=(%container.info) evt.num=%evt.num evt.cpu=%
          evt.cpu proc.name=%proc.name proc.cmdline=(%proc.cmdline) proc.
          pcmdline=(%proc.pcmdline) proc.exe=%proc.exe (thread.tid=%thread
          .tid) evt.info=(%evt.info) evt.dir=%evt.dir evt.type=%evt.type
          evt.args=(%evt.args)
32    tags: [customrule]
33    priority: INFO
34  # program starts another one
35  - rule: Detect a clone enter event
36    desc: Detect a clone enter event
37    condition: evt.type=clone and is_honeypot_container and evt.dir
          contains > and not proc.name contains udevadm
38    output: container.info=(%container.info) evt.num=%evt.num evt.cpu=%
          evt.cpu proc.name=%proc.name proc.cmdline=(%proc.cmdline) proc.
          pcmdline=(%proc.pcmdline) (thread.tid=%thread.tid) evt.info=(%
          evt.info) evt.dir=%evt.dir evt.type=%evt.type evt.args=(%evt.
          args)
```

```
39    tags: [customrule]
40    priority: INFO
41  - rule: Detect a clone exit event
42    desc: Detect a clone exit event
43    condition: evt.type=clone and is_honeypot_container and evt.dir
          contains < and not proc.name contains udevadm
44    output: container.info=(%container.info) evt.num=%evt.num evt.cpu=%
          evt.cpu proc.name=%proc.name proc.cmdline=(%proc.cmdline) proc.
          pcmdline=(%proc.pcmdline) (thread.tid=%thread.tid) evt.info=(%
          evt.info) evt.dir=%evt.dir evt.type=%evt.type evt.args=(%evt.
          args)
45    tags: [customrule]
46    priority: INFO
47  # input/output logging
48  - rule: Remote console
49    desc: Remote console
50    condition: evt.type=write and is_honeypot_container and proc.exe
          contains accepted and proc.name in (remote_console) and evt.dir
          contains <
51    output: container.info=(%container.info) evt.num=%evt.num evt.cpu=%
          evt.cpu proc.name=%proc.name proc.cmdline=(%proc.cmdline) proc.
          pcmdline=(%proc.pcmdline) (thread.tid=%thread.tid) evt.info=(%
          evt.info) evt.dir=%evt.dir evt.type=%evt.type evt.args=(%evt.
          args)
52    tags: [customrule]
53    priority: INFO
54  # network in- and outgoing
55  - rule: Detect a network accept
56    desc: Detect a network accept
57    condition: evt.type=accept and fd.type=ipv4 and
          is_honeypot_container
58    output: container.info=(%container.info) evt.num=%evt.num evt.cpu=%
          evt.cpu proc.name=%proc.name proc.cmdline=(%proc.cmdline) proc.
          pcmdline=(%proc.pcmdline) (thread.tid=%thread.tid) evt.info=(%
          evt.info) evt.dir=%evt.dir evt.type=%evt.type evt.args=(%evt.
          args)
59    tags: [customrule]
60    priority: INFO
61  # hostwide network with superserver xinetd
62  - rule: Detect hostwide a network accept with xinet
```

```
63    desc: Detect hostwide a network accept with xinet
64    condition: evt.type=accept and (fd.type=ipv4 or fd.type=ipv6) and
          proc.name contains xinetd
65    output: container.info=(%container.info) evt.num=%evt.num evt.cpu=%
          evt.cpu proc.name=%proc.name proc.cmdline=(%proc.cmdline) proc.
          pcmdline=(%proc.pcmdline) (thread.tid=%thread.tid) evt.info=(%
          evt.info) evt.dir=%evt.dir evt.type=%evt.type evt.args=(%evt.
          args)
66    tags: [customrule]
67    priority: INFO
```

A.1.1.9 Filebeat Dockerfile

```
1  # https://www.elastic.co/guide/en/beats/filebeat/current/running-on-
       docker.html
2  FROM docker.elastic.co/beats/filebeat:7.6.2
3
4  USER root
5  COPY filebeat.yml /usr/share/filebeat/filebeat.yml
6  RUN chown root:root /usr/share/filebeat/filebeat.yml && chmod 0400 /
       usr/share/filebeat/filebeat.yml
7
8  USER filebeat
```

A.1.1.10 Filebeat filebeat.yml

```
1  setup.dashboards.enabled: true
2  setup.dashboards.retry.enabled: true
3
4  # Duration interval between Kibana connection retries.
5  setup.dashboards.retry.interval: 1s
6
7  # Maximum number of retries before exiting with an error, 0 for
       unlimited retrying.
8  setup.dashboards.retry.maximum: 0
9
10 setup.kibana:
11   host: "http://${ELASTICSEARCH_HOST:}:5601"
12
13 filebeat.config:
14   modules:
```

```
15      path: ${path.config}/modules.d/*.yml
16      reload.enabled: false
17
18  filebeat.autodiscover:
19    providers:
20      - type: docker
21        templates:
22          - condition:
23              contains:
24                docker.container.image: falco
25            config:
26              - type: log
27                paths:
28                  - /var/lib/docker/containers/${data.docker.container.
                      id}/*.log
29                processors:
30                  - decode_json_fields:
31                      fields: ["message"]
32                      process_array: false
33                      max_depth: 1
34                      target: "tmp-msg"
35                      overwrite_keys: true
36                      add_error_key: true
37                  - decode_json_fields:
38                      fields: ["tmp-msg.log"]
39                      process_array: false
40                      max_depth: 1
41                      target: "falco"
42                      overwrite_keys: true
43                      add_error_key: true
44                  - timestamp:
45                      field: falco.time
46                      layouts:
47                        - '2020-07-30T13:27:02.621157494Z'
48                  - drop_fields:
49                      fields: ["tmp-msg"]
50                      ignore_missing: true
51      - type: docker
52        templates:
53          - condition:
```

```
54              contains:
55                docker.container.labels.dmz: "1"
56            config:
57              - type: log
58                paths:
59                  - /var/lib/docker/containers/${data.docker.container.
                      id}/*.log
60                exclude_lines: ["^\\s+[\\-`('.|_]"]  # drop asciiart
                     lines
61
62  filebeat.inputs:
63    - type: log
64      paths:
65        - /var/log/host/dockercompose_* # docker-compose logs and
            custom messages from project
66        - /var/log/host/xinetdlog # xinetd
67
68  filebeat.modules:
69    - module: auditd
70
71  output.elasticsearch:
72    hosts: '${ELASTICSEARCH_HOST:elasticsearch}:${ELASTICSEARCH_PORT
         :9200}'
73    username: '${ELASTICSEARCH_USERNAME:}'
74    password: '${ELASTICSEARCH_PASSWORD:}'
```

A.1.1.11 Metricbeat Dockerfile

```
1  # https://www.elastic.co/guide/en/beats/metricbeat/current/running-on
       -docker.html
2  FROM docker.elastic.co/beats/metricbeat:7.6.2
3
4  USER root
5  COPY metricbeat.yml /usr/share/metricbeat/metricbeat.yml
6  RUN chmod 0444 /usr/share/metricbeat/metricbeat.yml
7
8  # Add user "metricbeat" to group "docker" for earding /var/run/docker
       .sock
9  RUN usermod -aG 999 metricbeat
```

A.1.1.12 Metricbeat metricbeat.yml

```
setup.dashboards.enabled: true
setup.dashboards.retry.enabled: true

# Duration interval between Kibana connection retries.
setup.dashboards.retry.interval: 1s

# Maximum number of retries before exiting with an error, 0 for
    unlimited retrying.
setup.dashboards.retry.maximum: 0

setup.kibana:
  host: "http://${ELASTICSEARCH_HOST:}:5601"

processors:
  - add_host_metadata: ~

metricbeat.config.modules:
  # Glob pattern for configuration loading
  path: ${path.config}/modules.d/*.yml

  # Set to true to enable config reloading
  reload.enabled: true

  # Period on which files under path should be checked for changes
  reload.period: 10s

#==================== Elasticsearch template setting
    ==========================
setup.template.settings:
  index.number_of_shards: 1
  index.codec: best_compression
  #_source.enabled: false

metricbeat.modules:
  - module: docker
    metricsets:
      - "container"
      - "cpu"
      - "diskio"
```

```
38          - "event"
39          - "healthcheck"
40          - "info"
41          #- "image"
42          - "memory"
43          - "network"
44        hosts: ["unix:///var/run/docker.sock"]
45        period: 10s
46        enabled: true
47
48
49  output.elasticsearch:
50    hosts: '${ELASTICSEARCH_HOST:elasticsearch}:${ELASTICSEARCH_PORT
          :9200}'
51    username: '${ELASTICSEARCH_USERNAME:}'
52    password: '${ELASTICSEARCH_PASSWORD:}'
```

A.1.2 Debian-VM

A.1.2.1 vm-setup.sh

```
1  #!/bin/bash
2
3  # Shared Folder mappings (2):
4  #
5  # 01 - Docker [idRoot=0 readonly auto-mount host-icase guest-icase
       mnt-pt=/mnt/docker]
6  # 02 - HP-Host [idRoot=1 readonly auto-mount host-icase guest-icase
       mnt-pt=/mnt/hphost]
7  #
8
9  # Add user hpuser to group vboxsf to get VM-Shares accessible
10 /usr/sbin/usermod -aG vboxsf hpuser
11
12 # Docker Installation
13 # https://docs.docker.com/engine/install/debian/
14 apt-get update && apt-get install -y apt-transport-https ca-
       certificates curl gnupg-agent software-properties-common
15 curl -fsSL https://download.docker.com/linux/debian/gpg | apt-key add
       -
```

```
16 add-apt-repository "deb [arch=amd64] https://download.docker.com/
       linux/debian $(lsb_release -cs) stable"
17 apt-get update && apt-get install -y docker-ce docker-ce-cli
       containerd.io
18
19 # Docker Compose Installation
20 # https://docs.docker.com/compose/install/
21 curl -L "https://github.com/docker/compose/releases/download/1.26.2/
       docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-
       compose
22 chmod +x /usr/local/bin/docker-compose
23
24 # Manage Docker as a non-root user
25 # https://docs.docker.com/engine/install/linux-postinstall/
26 /usr/sbin/usermod -aG docker hpuser
27 /usr/bin/newgrp docker
28
29 # download and build images of honeynet
30 docker-compose -f /mnt/docker/DMZ/docker-compose.yml build
31
32 # download and build images of logging client (falco, file- &
       metricbeat)
33 docker-compose -f /mnt/docker/Logging-Client/docker-compose.yml build
34
35 # start logging client
36 docker-compose -f /mnt/docker/Logging-Client/docker-compose.yml up -d
37
38 # xinetd und socat (zur Portweiterleitung des Angreifers/Malware auf
       die jeweilige Docker-Umgebung)
39 apt-get install -y xinetd socat
40
41 # copy xinet.d configs and set readonly
42 cp /mnt/hphost/xinetd.conf /etc/xinetd.conf
43 chmod 444 /etc/xinetd.conf
44 cp /mnt/hphost/xinet.d/hp_* /etc/xinetd.d/
45 chmod 444 /etc/xinetd.d/hp_*
46
47 service xinetd restart
48 service xinetd status
49
```

```
50 # display listened ports
51 netstat -tulpen
52 # should display:
53 # tcp6       0      0 :::80                   :::*
                          LISTEN      0          16087      855/xinetd
54 # tcp6       0      0 :::22                   :::*
                          LISTEN      0          16085      855/xinetd
55 # tcp6       0      0 :::23                   :::*
                          LISTEN      0          16086      855/xinetd
56 # tcp6       0      0 :::443                  :::*
                          LISTEN      0          16088      855/xinetd
57
58 # add cleanup script as crontab
59 (crontab -l 2>/dev/null; echo "*/1 * * * * bash /mnt/hphost/cleanup.
       sh") | crontab -
60
61 echo "all done. It's time for popcorn"
```

A.1.2.2 cleanup.sh

```
1  #!/usr/bin/env bash
2
3  #
4  # SETTINGS
5  #
6  PROJECT_YML="/mnt/docker/DMZ/docker-compose.yml"
7
8  # max cpuacct.usage_user per container.
9  # if its reached, the hole project will be stopped
10 # @see https://www.kernel.org/doc/Documentation/cgroup-v1/cpuacct.txt
11 # 1000000000 = 1 second
12 MAX_CPU_USER_USAGE=$((120*1000000000))
13 # Choose old container to kill
14 MAX_RUNTIME_SEC=$((60*60*12))
15
16 stop_by_cid() {
17  CID="$1"
18
19  PROJECT_NAME=$(docker inspect --format='{{ index .Config.Labels "com
       .docker.compose.project"}}'  "${CID}")
20
```

```
21   echo "$(date -Iseconds): ContainerID ${CID} from project ${
        PROJECT_NAME} consumed too much cpu. Will be killed" >> "/var/log
        /dockercompose_${PROJECT_NAME}.log"
22
23   # disconnect networks
24   # needed because of error during docker-compose down: ERROR: network
         XXX id YYY has active endpoints
25   for NETWORK in $(docker inspect --format='{{ range $key,$val:=.
        NetworkSettings.Networks }}{{$key}}{{end}}' "${CID}")
26   do
27    docker network disconnect "${NETWORK}" "${CID}"
28   done
29
30   # stop project
31   export PROJECT_NAME && docker-compose -f "${PROJECT_YML}" -p "${
        PROJECT_NAME}" down -t 1
32   sleep 600 && rm "/var/log/dockercompose_${PROJECT_NAME}.log"
33  }
34
35
36  # Getting all containers which belongs to honey-nets. Filtered by "
       label=dmz=1"
37  for CID in $(docker ps --no-trunc -q --filter "label=dmz=1")
38  do
39      CPU_USER_USAGE=$(cat /sys/fs/cgroup/cpuacct/docker/${CID}/cpuacct
           .usage_user)
40      echo "CPU-Usage of ${CID}: ${CPU_USER_USAGE}"
41      if [[ ${CPU_USER_USAGE} -ge ${MAX_CPU_USER_USAGE} ]] ; then
42    stop_by_cid ${CID}
43   else
44    STARTED_AT=$(docker inspect --format='{{ .State.StartedAt }}'  "${
         CID}")
45    RUNTIME_SEC=$(($(date +%s) - $(date -d "${STARTED_AT}" +%s)))
46    if [[ ${RUNTIME_SEC} -ge ${MAX_RUNTIME_SEC} ]] ; then
47     stop_by_cid ${CID}
48    fi
49      fi
50  done
```

A.1.2.3 xinetd-Konfiguration

```
defaults
{
    log_type        =  FILE /var/log/xinetdlog
    log_on_success  =  PID USERID HOST DURATION EXIT
    log_on_failure  =  USERID HOST ATTEMPT
}
includedir /etc/xinetd.d
```

A.1.2.4 xinetd-Konfiguration jumphost

```
service hp_ssh
{
        socket_type = stream
        protocol    = tcp
        port        = 22
        type        = UNLISTED
        wait        = no
        user        = root
        server      = /mnt/hphost/data/hp-welcome.sh
        server_args = 22
}

service hp_telnet
{
        socket_type = stream
        protocol    = tcp
        port        = 23
        type        = UNLISTED
        wait        = no
        user        = root
        server      = /mnt/hphost/data/hp-welcome.sh
        server_args = 23
}
```

A.1.2.5 xinetd-Konfiguration web

```
service hp_web
{
        socket_type = stream
        protocol    = tcp
        port        = 80
```

```
6           type        = UNLISTED
7           wait        = no
8           user        = root
9           server      = /mnt/hphost/data/hp-welcome.sh
10          server_args = 80
11  }
```

A.1.2.6 von xinetd ausführendes Script hp-welcome.sh

```
1   #!/usr/bin/env bash
2
3   #
4   # SETTINGS
5   #
6   PROJECT_YML="/mnt/docker/DMZ/docker-compose.yml"
7   DUMMY_PROJECT_NAME="public"
8
9   #
10  # FUNCTIONS
11  #
12
13  # connect request to IP (arg 1) and port (arg 2)
14  connect() {
15      exec /usr/bin/socat stdin tcp:${1}:${2},interval=1,retry=30
16  }
17
18  # create_hn, submit PROJECT_NAME as arg 1
19  create_hn() {
20      `is_up "${1}"` || docker-compose -f "${PROJECT_YML}" -p "${1}" up
            -d >/var/log/dockercompose_${1}.log 2>&1
21      return $?
22  }
23
24  # get_container_ip PROJECT_NAME SERVICE_NAME
25  get_container_ip() {
26      echo $(docker inspect --format='{{range .NetworkSettings.Networks
            }}{{.IPAddress}}{{end}}' $(docker ps -q --filter "label=com.
            docker.compose.project=${1}" --filter "label=com.docker.
            compose.service=${2}" ))
27  }
28
```

```
29 # returns hashed STRING. String must be submitted as first arg
30 str_hash() {
31     echo $(printf '%s' $1 | md5sum | cut -d ' ' -f 1)
32 }
33
34 # is_up PROJECT_NAME
35 is_up() {
36     [[ "$(docker ps -q --filter 'label=com.docker.compose.project='$
           {1}'')" ]]
37 }
38
39 #
40 # PROGRAM
41 #
42
43 export PROJECT_NAME=$(str_hash "${REMOTE_HOST}")
44 DST_PORT=$1
45
46 case "${DST_PORT}" in
47     22|23)
48         SERVICE=jumphost
49         ;;
50     80|443)
51         SERVICE=web
52         # select dummy-project if its a new web request - no, don't
               do so to recognize web attacks
53         # `is_up "${PROJECT_NAME}"` || PROJECT_NAME="${
               DUMMY_PROJECT_NAME}"
54         ;;
55     *) exit 1
56 esac
57
58 create_hn "${PROJECT_NAME}"
59 # echo "getting container ip for project ${PROJECT_NAME} and service
       ${SERVICE}"
60 CONTAINER_IP=$(get_container_ip "${PROJECT_NAME}" "${SERVICE}")
61 connect "${CONTAINER_IP}" "${DST_PORT}"
62
63 exit 0
```

A.2 Aufzeichnungs-Server

A.2.1 docker-compose.yaml

```
1  version: "3.7"
2  services:
3    elk:
4      image: sebp/elk:762
5      container_name: elk
6      mac_address: ba:c8:e6:ba:08:ee
7      networks:
8        - qnet-dhcp-eth1
9      ports:
10       - "5044:5044"
11       - "5601:5601"
12       - "9200:9200"
13       - "9300:9300"
14     restart: always
15     volumes:
16       - /share/CACHEDEV1_DATA/hp-logging/elk-docker/elasticsearch.yml
           :/opt/elasticsearch/config/elasticsearch.yml:ro
17       - /share/CACHEDEV1_DATA/hp-logging/elk-docker/kibana.yml:/opt/
           kibana/config/kibana.yml:ro
18       - data-elk:/var/lib/elasticsearch
19
20 # docker network create -d qnet --ipam-driver=qnet --ipam-opt=iface=
     eth1 qnet-dhcp-eth1
21 networks:
22   qnet-dhcp-eth1:
23     external: true
24
25 volumes:
26   data-elk:
27     driver: local
```

A.2.2 elasticsearch.yml

```
1 # Use a descriptive name for the node:
2 node.name: elk
3 # Path to snapshots for backups:
```

```
4  path.repo: /var/backups
5  # Set the bind address to a specific IP (IPv4 or IPv6):
6  network.host: 0.0.0.0
7  # Set a custom port for HTTP:
8  #http.port: 9200
9
10 # allow_mmap = false, fixes vm.max_map_count=262144 gibt
11 # Alternativ, bei Hostzugang: sysctl -w vm.max_map_count=262144
12 # @see: https://www.elastic.co/guide/en/elasticsearch/reference/
       current/index-modules-store.html
13 node.store.allow_mmap: false
14 index.store.type: niofs
```

A.2.3 kibana.yml

```
1  # Default Kibana 5 file from https://github.com/elastic/kibana/blob/
       master/config/kibana.yml
2  #server.port: 5601
3  # Specifies the address to which the Kibana server will bind.
4  server.host: "0.0.0.0"
```

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: Deichen

Vorname: Michael

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Konzept zur Erkennung der Ausbreitung von Malware und internen Angriffen mittels Medium-Interaction Honeypots

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original