HAW HAMBURG

Bachelorarbeit

Uwe Krause

Kukulkan high interaction honeypot: Ein modulares System zur sicheren Protokollierung von Privilegien-Erweiterungs-Angriffen gegen weitverbreitete Serverbetriebssysteme und Dienste

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Uwe Krause

Kukulkan high interaction honeypot: Ein modulares System zur sicheren Protokollierung von Privilegien-Erweiterungs-Angriffen gegen weitverbreitete Serverbetriebssysteme und Dienste

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. Bettina Buth

Eingereicht am: 18. Dezember 2020

Uwe Krause

Thema der Arbeit

Kukulkan high interaction honeypot:
Ein modulares System zur sicheren Protokollierung von Privilegien-Erweiterungs-Angriffen gegen weitverbreitete Serverbetriebssysteme und Dienste

Stichworte

Honeypot, high interaction honeypot, research honeypot, Köder, Container, Namensräume, full system container, Betriebssystemisolierung, Überwachung, Prozessüberwachung, Protokollierung von Angriffen, privilege escalation, capture the flag

Kurzzusammenfassung

In dieser Bachelorarbeit wird der Frage nachgegangen, wie ein System aufgebaut sein kann, mit dem die Interaktionen eines Angreifers nach einem erfolgreichen Einbruch in einen Server gefahrlos protokolliert werden können, um wiederum Einblicke in das Verhalten bei der Privilegienerweiterung und Informationen über die hierfür verwendeten Hilfswerkzeuge zu erhalten. Hierfür wurde ein high interaction honeypot aufgebaut, der einem erfolgreichen Angreifer ein voll funktionsfähiges, aber komplett isoliertes Betriebssystem anbietet und die Interaktionen mit diesem aufzeichnet. Unter Anwendung einer agilen Vorgehensweise wurden die für das Gesamtsystem benötigten Komponenten – der Aufbau eines glaubwürdigen Köders, die sichere Isolation eines Betriebssystems und die detaillierte Ausführungsüberwachung – mehrere technische Ansätze diskutiert. Pro Komponente wurde eine Lösungsmöglichkeit detailliert beschrieben, praktisch erprobt und mit den jeweils anderen zu einem funktionsfähigen, modular aufgebauten high interaction honeypot kombiniert. Dieser kann beispielsweise von Systemadministratoren verwendet werden, entweder unverändert oder mit angepassten Modulen. Das entwickelte System wurde bei einem hierfür veranstalteten capture-the-flag-Event mit etwa einem Dutzend Teilnehmern getestet. Hierbei konnten die Interaktionen der Angreifer wie gewünscht protokolliert werden.

Uwe Krause

Title of Thesis

Kukulkan high interaction honeypot: A modular system for secure logging of privilege escalation attacks against widely-used server operating systems and services

Keywords

Honeypot, high interaction honeypot, research honeypot, container, name spaces, full system container, operation system isolation, surveillance, process surveillance, logging of attacks, privilege escalation, capture the flag

Abstract

This bachelor thesis explores the question of how a system can be built to safely log an attacker's interactions after successfully breaking into a server, in order to gain insight into privilege escalation behavior and information about the assistive tools used for this purpose. For this purpose, a high interaction honeypot was built which provides a successful attacker a fully functional yet completely isolated operating system and records the interactions with it. Using an agile approach, the components needed for the overall system – building a credible bait, securely isolating an operating system, and detailed execution monitoring – several technical approaches were discussed. For each component, one possible solution was described in detail, tested in practice, and combined with the others to create a functional, modular high interaction honeypot. This honeypot can be used by system administrators, for example, either unchanged or with adapted modules. The developed system was tested with about a dozen participants during a capture the flag event which was organized for this purpose. The interactions of the attackers could be logged as desired.

Inhaltsverzeichnis

Gender-Disclaimer

Da in den in dieser Arbeit angesprochenen Themengebieten das Geschlecht der ausführenden Person oft unbekannt und vor allem absolut belanglos ist, wird für eine bessere Lesbarkeit das generische Maskulinum verwendet. Weibliche und anderweitige Geschlechteridentitäten sollen dabei ausdrücklich genauso angesprochen werden. Dies gilt besonders für Formulierungen wie „Angreifer“, „Administrator“ und „Teilnehmer“.

Danksagungen

Viele viele Menschen in meinem Leben haben durch außergewöhnlichen Einsatz und mit persönlichem Engagement dazu beigetragen, dass ich trotz aller Startschwierigkeiten in Kindheit und Jugend überhaupt die Chance erhalten habe, wenn auch verspätet, ein Studium anfangen zu dürfen und es auch erfolgreich abzuschließen.

Neben allen Personen, die für mich eine Mutter- oder Vaterrolle innehatten, möchte ich besonders Sybille, Udo, Hose, Mark, Manuela und Anna danken, die meine Kindheit und Jugend sehr geprägt haben. Ohne Euch wäre ich nicht ich.

Im Studium selbst danke ich denjenigen Professoren und wissenschaftlichen Hilfskräften, die nicht nur gelehrt, sondern begeistert haben. Auch vielen Kommilitonen – teilweise im Sinne der Wortherkunft – möchte ich danken. Allen voran Sean, der in anregenden Gesprächen immer eine weitere Perspektive eröffnet und ein wertvoller Mitstreiter war. „Unter Druck entstehen Diamanten."

Für die Abschlussarbeit gilt mein Dank für Betreuung, Unterstützung und für kritische Anmerkungen Herrn Prof. Dr. Kossakowski und Frau Prof. Dr. Buth.

Für die Teilnahme am ctf-Wettbewerb, einem nicht unwesentlichen Teil dieser Arbeit, dank an die ctf-Teams *Cyclopropenylidene*, *CInsects*, *FOXACID* und alle Einzelspieler.

Den Idlern im *#ccchh* danke ich für erhellende und einordnende Gespräche im IRC.

Nicht zu vergessen sei die Heerschar von Entwicklern, die GNU/Linux und alle anderen verwendeten Werkzeuge geschaffen haben und frei und open source bereitstellen.

Mein Dank gilt auch der Lufthansa Technik AG für finanzielle Unterstützung. Besonderen Dank auch an meine Vorgesetzten, die mit viel Flexibilität das Studium ermöglichten.

Abschließend herzlichen Dank an Sonya für das Lektorat jeder einzelnen der folgenden Zeilen, für Kritik, Diskussion, Unterstützung und für tausend weitere Dinge, deren Aufzählung den Rahmen sprengen würde.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Einleitung

Im Jahr 2016 veröffentlichte *Uber* unbeabsichtigt die Datensätze von 57 Millionen Kunden und Fahrern. “None of this should have happened, and I will not make excuses for it” äußerte sich der CEO des Transportunternehmens in einer öffentlichen Stellungnahme zu diesem Zwischenfall. [Newcomer, 2017] Dem jungen Online-Vermittlungsdienstleister, der sich innerhalb kürzester Zeit auf dem Markt etablierte, gelang es nicht, seine wichtigen Daten zu schützen. Durch versehentlich in falsche Hände geratene Zugangsdaten für einen von *Uber* genutzen *Cloud*-Dienst konnten Angreifer auf ein Firmenarchiv zugreifen und diese unbedingt zu schützenden Daten extrahieren.

Kein Unternehmen – auch ein etabliertes nicht – kommt heutzutage ohne IT-gestützte Kommunikation aus. Besonders Zwischenfälle im Bereich der IT-Sicherheit häufen sich, und werden oft von großem Medienecho begleitet. Immer wieder zeigt sich, dass viele davon vermeidbar gewesen wären, wenn mit vergleichbar geringem Aufwand grundlegende Sicherheitsmaßnahmen beachtet worden wären.

Ein aktuelles Beispiel aus Deutschland zeigt, wie fahrlässig mit Kundendaten umgegangen wird: der Fall der deutschen Autovermietung *Buchbinder*. Wie im Januar dieses Jahres berichtet wurde, waren detaillierte und eindeutig personenbezogene Datensätze von Millionen Kunden des Traditionsunternehmens – darunter teilweise auch die von Prominenten, Politikern und Diplomaten – auf einem ungesicherten *Backupserver* frei im Netz zugänglich. [Tagesschau, 2020]

Wenn es also schon den großen Firmen, die die Marktmacht innehaben, in den Besitz von Millionen von Kundendatensätzen zu kommen, nicht gelingt, diese zu schützen, ist davon auszugehen, dass dies auch kleinen und mittelständischen Unternehmen (KMU) schwerfällt. Diese haben üblicherweise nicht das Budget, um eine komplette, auf alle erdenklichen Gefahren spezialisierte IT-Abteilung zu finanzieren, die gezielte Angriffe abwehren kann. Somit ist es nicht verwunderlich, wenn regelmäßig von erfolgreichen Einbrüchen berichtet wird. Laut Wilczek [2019], der sich auf eine umfangreiche Studie (siehe Cisco [2018]) bezieht, wurde bereits „über die Hälfte der KMU erfolgreich gehackt“.

Ein IT-Systemadministrator spielt eine zentrale Rolle für den Schutz der Firma und sieht sich täglich mit immer wieder neuen Schwierigkeiten konfrontiert. Er muss bei der Einrichtung und dem Betrieb der IT-Unternehmensinfrastruktur nicht nur dauerhafte automatisierte Angriffe abwehren, sondern auch hoch spezialisierte Angreifer, die sich gezielt für das ihm anvertraute Netzwerk interessieren. [Tanriverdi u. a., 2019]

Mit steigender Komplexität der Infrastruktur vergrößert sich ihre Angriffsfläche. Je mehr *Dienste* das Firmennetzwerk mit dem Internet verbinden, desto mehr *Angriffsvektoren* stehen Angreifern zur Verfügung.

Die Angriffsmöglichkeiten sind zahlreich, Angreifer verfügen über spezialisierte Fähigkeiten und Werkzeuge, die ihnen bei ihren Vorhaben dienen. Dem gegenüber hinken die Kenntnisse der Verteidiger denen der Angreifer nicht selten wie bei einem sprichwörtlichen „Katz-und-Maus-Spiel“ hinterher.

1.1 Zielgruppe

Um Systemadministratoren ein Verfahren an die Hand zu geben, mit dem sie ihr eigenes Fachwissen anhand realer Bedrohungen stets erweitern und Angriffe speziell auf die von ihnen eingesetzten Dienste beobachten können, wird in dieser Arbeit ein modular erweiterbares System konzipiert und entwickelt. Dieses wird getestet und verwendet, was einen Ausblick auf seine Leistungsfähigkeit gibt.

Auf Grundlage dieser Arbeit können Systemadministratoren mit freien Linux-Bordmitteln und ohne den Einsatz von kostenpflichtiger Spezialsoftware den hierfür entwickelten *Kukulkan high interaction honeypot* verwenden. Sie können das entwickelte System unverändert einsetzen, um mehr über das Verhalten von Angreifern zu erfahren.

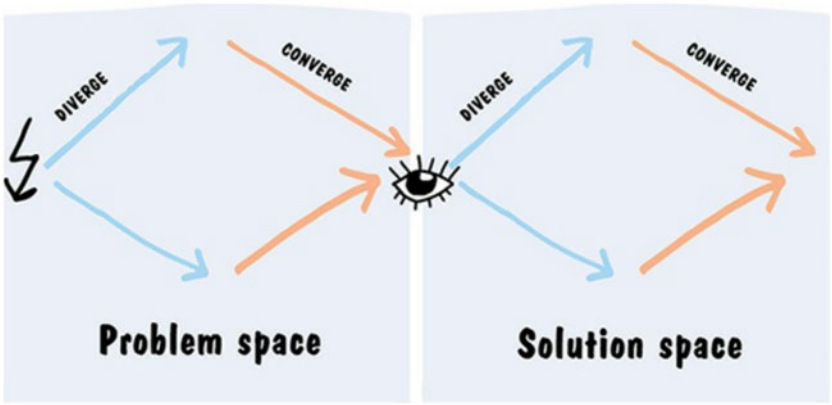
Der Honeypot kann aufgrund seines modularen Aufbaus um weitere – gegebenenfalls in der eigenen Systemlandschaft verwendete – Dienste individuell ergänzt werden. Die Einrichtung eines solchen „Dienst-Moduls“ erfolgt ähnlich der Einrichtung auf einem regulären Server, weshalb keine oder nur wenig Umgewöhnung notwendig ist.

Die für diese Arbeit entwickelten Beispiele können übernommen, angepasst und beliebig erweitert werden. Sie können somit als Grundlage für individuelle und auf spezielle Anwendungsfälle zugeschnittene Installationen von *high interaction honeypots* verwendet werden.

1.2 Vorgehensweise in dieser Arbeit

Diese Arbeit ist nach der als *double diamond* bezeichneten Methode strukturiert, die aus dem agilen Kreativprozess des *design thinking* entliehen ist.

Die *design-thinking*-Methode wurde in den 1990er Jahren von Terry Winograd, Larry Leifer und David Kelley entwickelt. [Leifer u. a., 2018] Das Wort „design“ bezieht sich dabei auf die Konzeptionierung eines Prozesses, eines Produktes, einer Idee oder einer Innovation. Bei Anwendung der Methode wird ein zweistufiges Verfahren durchlaufen, das in Form von zwei Rechtecken skizziert werden kann, die der Methode den Namen des „doppelten Diamanten“ (engl. “double diamond”) geben. Als Erstes wird auf Grundlage der „*design challenge*“ der „*Problemraum*“ beschrieben. Basierend darauf wird anschließend der „*Lösungsraum*“ betrachtet. Die beiden Phasen werden wiederum in zwei Teilschritte untergliedert, wobei beide Räume jeweils zuerst „erweitert“ und anschließend „verfeinert“ werden.



Quelle: [Leifer u. a., 2018]

Abbildung 1.1: Erweiterung und Verfeinerung des Problem- und Lösungsraums

Das Fundament des Vorgehens bildet die *design challenge*, durch die das grobe Ziel des Projekts definiert wird. Dieses kann beispielsweise das Lösen eines vorhandenen und bekannten Problems sein, oder der Wunsch, eine noch nicht näher bekannte Innovation für einen festgelegten Bereich zu schaffen. Die *design challenge* wird bewusst vage formuliert, um Freiheiten für Kreativität und unerwartete Ideen zu lassen.

Der *Problemraum* enthält Probleme oder Chancen, die gelöst oder ergriffen werden können, um eine Verbesserung oder Innovation zu erzielen. Bei dem Schritt der „Erweiterung“ werden möglichst viele solcher Ideen gesammelt. Um kreative Denkprozesse nicht

zu hemmen, werden die Ideen nicht frühzeitig beurteilt. Erst im zweiten Schritt, der „Verfeinerung“, werden sie präzisiert, bewertet und priorisiert. Als Ergebnis dieser Phase steht die Entscheidung fest, von welcher der gesammelten Verbesserungsmöglichkeiten eine konkrete Maßnahme abzuleiten ist.

Auf Grundlage der durch den *Problemraum* ermittelten Chance zur Verbesserung werden Lösungsmöglichkeiten zusammengetragen. Analog zum *Problemraum* wird der *Lösungsraum* ebenfalls erst „erweitert“ und anschließend „verfeinert“. Bei der Erweiterung werden Techniken oder mögliche Methoden gesammelt, die zum Erreichen des verfeinerten Ziels beitragen können. Auch hier werden Lösungsmöglichkeiten nicht frühzeitig beurteilt. Dieser Raum wird anschließend ebenfalls reduziert. Am Ende dieser Phase steht ein Prototyp einer konkreten Lösung des zuvor definierten Problems oder einer zuvor erkannten Innovation.

Dieser Prototyp stellt die erste Version des Produktes dar und kann – gemäß dem agilen Grundgedanken des ständigen Austausches – mit Kunden oder Vertretern der Zielgruppe besprochen und von ihnen getestet werden. Das so erhaltene *Feedback* kann im Rahmen der nächsten Entwicklungs-Iteration berücksichtigt werden.

1.3 Ziel der Arbeit

Gemäß der Philosophie der *design challenge* ist das Ziel anfangs bewusst grob formuliert, wird aber im weiteren Verlauf konkretisiert. Der Wunsch, das Verhalten böswilliger Individuen im Bereich der Informationstechnologie zu verstehen, bildet die Grundlage dieser Arbeit.

Es existieren viele Möglichkeiten, wie dieses Ziel verstanden und – auf diesem Verständnis basierend – erreicht werden kann. Der Methodik folgend werden diese zunächst im *Problemraum* gesammelt.[1] Der Begriff „Verhalten“ könnte sich beispielsweise auf die Informationsbeschaffung des Angreifers beziehen oder darauf, nach welchen Kriterien dieser seine Ziele auswählt. Eine andere Variante wäre eine Beobachtung von Angriffen, die mittels *social engineering* durchgeführt werden. Eine Sammlung und Untersuchung von *malware samples* könnte sowohl Einblicke in die Verbreitung von Schadsoftware geben als auch interessante Analyse-Ergebnisse zum Verhalten dieser liefern. Im Kontext des kürzlich ausgelaufenen Supports des noch immer weit verbreiteten Betriebssystems

[1] Da verworfene Ideen letztlich kein Bestandteil der Arbeit sind, werden sie hier nur erwähnt.

Windows 7 könnte auch untersucht werden, ob sich Angriffe auf diese Version vermehren. Stets aktuell und von dauerhaftem Interesse ist die Frage, wie Angreifer in IT-Systeme eindringen und zu welchem Zweck sie einen erbeuteten Server missbrauchen.

Eine Bearbeitung all dieser und noch vieler anderen Themen könnte zwar jeweils für sich die Anforderung der formulierten *design challenge* erfüllen, die Methode der *Verfeinerung* des *Problemraums* gibt jedoch vor, sich – nach Bewertung und Priorisierung – aus der Fülle der Möglichkeiten eine erfolgversprechende Idee herauszugreifen.

Das im Rahmen dieser Bachelorarbeit durchzuführende Projekt sollte die folgenden Kriterien erfüllen: Es sollte in sich abgeschlossen sein, das daraus entstehende Produkt wenige Abhängigkeiten haben. Die zu erstellende Lösung sollte praktisch und überprüfbar sein. Erkenntnisse aus der Arbeit sollten einen konkreten Nutzen für Andere darstellen.

Für diese Arbeit wurde daher folgende Problemstellung ausgewählt: **Entwurf und Umsetzung eines Systems, mit dem sich das Verhalten von Angreifern gefahrlos auf einem hierfür präparierten Server beobachten lässt.**

1.4 Leitfragen

Auf der gewählten Problemstellung basierend lässt sich folgende Leitfrage formulieren:

Wie kann ein System aufgebaut sein, mit dem sich ein Angreifer und seine manuelle Interaktion mit einem Server nach erfolgreichem Einbruch gefahrlos beobachten lassen?

1.4.1 Grundlegende Teilfragen

Zur Konkretisierung der Problemstellung muss diese zuallererst in mehrere Teilfragen untergliedert werden. Diese Fragen dienen als Basis für die Erweiterung des Lösungsraumes. Die ermittelten Lösungsmöglichkeiten bilden später die theoretische Grundlage, um den Lösungsraum letztlich durch die Entwicklung der ersten Produktversion schließen zu können.

Wie ist eine realistische, angreifbare Serverumgebung aufzubauen?
Wie verhält sich ein typischer Firmenserver? Welche Technologien und Dienste sind weit verbreitet? Wie interagieren Angestellte mit diesen? Wie ist der Zugang zu diesen Diensten gesichert? Wie kann ein geeignetes, unauffälliges Einfallstor offen gelassen werden? Wie lassen sich automatisierte Skripts möglichst aussperren, gleichzeitig aber einem Angreifer trotzdem der manuelle Zugang ermöglicht werden?

Wie kann die Umgebung zuverlässig beobachtet werden?
Welche Hilfsmittel eignen sich, um die isolierte Umgebung zu beobachten, ohne dass der Angreifer dies merkt? Wie können Messwerte vertrauenswürdig protokolliert werden, obwohl der Angreifer sich umfassende Berechtigungen erschlichen haben könnte?

Wie kann der Beobachtende vor Schaden geschützt werden?
Welche Schutzgrenzen können gezogen werden? Welche Technologien erlauben die Isolation eines nicht-vertrauenswürdigen Anwenders/Angreifers? Wie kann sichergestellt werden, dass der Angreifer nicht frühzeitig bemerkt, bewusst eingesperrt worden zu sein? Welche Sicherheitsmaßnahmen müssen ergriffen werden, um die gesamte Infrastruktur zu schützen? Wie kann ein Ausbruch des Angreifers verhindert werden?

Die Frage nach dem Aufbau der anzugreifenden Umgebung ist eher praktischer Natur und zu Beginn zu beantworten. Dies soll bewirken, dass sich die Ideenfindung an einer bestehende Umgebung orientiert und nicht, andersherum, eine – bei der Zielgruppe eventuell schon bestehende – Umgebung an das entwickelte System angepasst werden muss.

Um die Hauptfrage nach einem möglichen Systemaufbau auch praktisch beantworten zu können, müssen die Teilfragen zunächst theoretisch beleuchtet werden. Mögliche Techniken zum Erreichen der Teilziele müssen gesammelt werden, was der Erweiterung des *Lösungsraums* entspricht.

1.4.2 Praktische Teilfragen

Für den Abschluss des *Lösungsraums* werden ausgewählte Technologie-Bausteine miteinander zu einem Prototyp kombiniert. Dem agilen Grundgedanken folgend dient dieser mehreren Zwecken: Für den Entwicklungsprozess selbst kann ein *experimentelle Prototyp* den Entwicklern dabei helfen, die Realisierungsmöglichkeit abzuwägen und den Umgang

mit neuen Werkzeugen, Methoden und Technologien zu erlernen. Bei der Softwareentwicklung sollte jedoch nicht der Entwickler, sondern immer der Anwender im Vordergrund stehen. Ein *explorativer Prototyp* kann dies unterstützen, indem er mit Vertretern der Zielgruppe getestet wird, was wertvolles Feedback liefert. Dieses wiederum erlaubt dem Entwickler, Annahmen der Entwurfsphase zu kontrollieren und gegebenenfalls zu korrigieren. [Krypczyk und Bochkor, 2016]

Wie lassen sich diese Bausteine zu einem Gesamtsystem verbinden?
Welche der vorgestellten Technologien und Verfahren sind für das Vorhaben am besten geeignet? Welche Eigenschaften müssen die Bausteine besitzen, um miteinander kombinierbar zu sein? Müssen aufgrund bestimmter Entscheidungen Kompromisse eingegangen werden? Welche rechtlichen, ethischen und technischen Risiken bleiben bestehen?

Erfüllt das entwickelte System die Anforderungen?
Lassen sich die Interaktionen eines Angreifers wie gedacht beobachten?

1.5 Anforderungen an das Gesamtsystem

Durch die agile Methode ist im Gegensatz zur klassischen Vorgehensweisen, in der zu Beginn eines Vorhabens eine Anforderungsanalyse durchgeführt wird, der weitere Weg nicht bereits vorgezeichnet, sondern muss explorativ ermittelt werden.

Weil das Ausloten der verfügbaren Möglichkeiten ein wesentlicher Teil des Vorgehens ist und weil das zu entwickelnde System erst dann spezifiziert werden kann, wenn genügend Hilfsmittel bekannt sind, ist es vor der experimentellen Phase noch gar nicht möglich, die Anforderungen an das Gesamtsystem genauer festzuschreiben. Allgemein lässt sich jedoch sagen, dass die Anforderungen an das im Praxisteil zu entwickelnde System den Antworten auf die bisher gestellten Teilfragen entsprechen. Beispielsweise ergibt sich aus der Teilfrage „Wie verhält sich ein typischer Firmenserver?“ die Anforderung „Das System muss sich nach außen verhalten wie ein typischer Firmenserver“.

Konkrete („messbare“) Anforderungen werden erst im Praxisteil erhoben und dementsprechend auch erst im Abschnitt „Anforderungen an das zu entwickelnde System“ (6.4) erläutert.

1.6 Aufbau der Arbeit

Diese Arbeit besteht – neben Einleitung und Fazit – aus einem theoretischen und einem praktischen Teil. Die in drei Grundlagenkapiteln vorgestellten Methoden und Technologien werden in drei Praxiskapiteln zu einem Gesamtsystem kombiniert und getestet.

In Kapitel 1 wird in die Arbeit eingeführt. Kontext und Zielgruppe werden beschrieben, die gewählte Methodik des *double diamond* als Teil der agilen Vorgehensweise des *design thinking* wird vorgestellt. Das grobe Ziel, ein System zur Beobachtung eines Angreifers aufzubauen, wird formuliert und mit Hilfe von Leitfragen – aus denen sich auch die Anforderungen an das System ergeben – feiner gegliedert. Außerdem wird hier der strukturelle Aufbau der Arbeit dargelegt. Abschließend wird der Namensgeber der Arbeit und des Honeypots vorgestellt.

Die theoretische Grundlage der *Honeypots* wird in Kapitel 2 erläutert. Der Betrieb und die Entwicklung eines Honeypots wird motiviert und verschiedene Ausprägungen werden gemeinsam mit ihrem jeweiligen Einsatzzweck vorgestellt.

Nur ein realistisch wirkendes Ziel wird einem Angreifer als lohnenswert erscheinen. Der grundlegende Aufbau eines Webservers nach aktuellem Stand der Technik wird in Kapitel 3 angesprochen. Für den Köder wird eine Firma „gegründet“ und eine (unsichere) Firmenpräsenz für das Internet entwickelt.

Um einen Angreifer oder nicht vertrauenswürdigen Anwender „einzusperren“, existieren verschiedene Methoden. Neben *virtuellen Maschinen* wird in Kapitel 4 die *Containerisierung* vorgestellt. Techniken zur Separation und Limitierung von Prozessen werden beleuchtet und ausgetestet. Weiterhin wird beschrieben, wie es durch isolierte Prozessgruppen möglich ist, ein vollständig abgetrenntes Betriebssystem zu betreiben, auf dem ein Angreifer sich frei bewegen kann, ohne wirklichen Schaden anrichten zu können.

In Kapitel 5 werden einige Techniken zur Überwachung eines in einem Container isolierten Anwenders/Angreifers beschrieben. Eine Möglichkeit ist der Einsatz eines entsprechendem Kernel-Moduls. Eine konkrete Implementierung dieser Möglichkeit wird vorgestellt und ausgetestet.

In Kapitel 6 werden die zuvor genannten Elemente miteinander zu einer Software verbunden, um den *Kukulkan high interaction honeypot* zum Leben zu erwecken. Die Architektur wird skizziert und die verwendeten Hilfsmittel aufgezählt. Der Honeypot wird einmal komplett manuell aufgebaut. Auf Grundlage dieser Erfahrungen werden die notwendigen

Schritte in einer Skriptsammlung gesichert, wodurch das entwickelte Honeypotsystem auf einem beliebigen Server installiert werden kann.

Das entwickelte Systems wird in Kapitel 7 getestet. Hierfür wird ein *capture-the-flag*-Event veranstaltet, bei dem die Teilnehmer gebeten werden, das entwickelte System anzugreifen. Dadurch lassen sich ihre Interaktionen beobachten. Eingeleitet wird dieses Kapitel durch eine Beschreibung des *capture-the-flag*-Begriffs. Außerdem wird erklärt, warum die erfolgreiche Durchführung eines *capture-the-flag*-Events als geeigneter Nachweis für die Funktion des erstellten Systems angesehen werden kann. Es wird zudem geschildert, wie das *ctf*-Event aufgebaut wurde und welche Beobachtungen sich währenddessen anstellen ließen.

Nachdem der entwickelte Honeypot für einen längeren Zeitraum online war, werden im Exkurskapitel 8 die Protokolldateien ausgelesen und analysiert.

In Kapitel 9 wird zusammengefasst, was erarbeitet wurde. Es wird dargelegt, warum das Ziel der Arbeit erreicht wurde. Verbesserungspotenziale und Anregungen für weitere Arbeiten werden hier gegeben.

1.7 Namensgeber *Kukulkan*

Namensgeber des *Honeypot*-Systems und somit dieser Arbeit ist die mittelamerikanische Schlangengottheit Quetzalcōātl, die mit den bunten bis zu einem Meter langen Schwanzfedern einer lateinamerikanischen Vogelart geschmückt ist. In der Sprache der Maya heißt der Gott „K'uk'ulkan“, was auch für westliche Ohren wohlklingend ist und sich daher sowohl als Firmen- als auch als Produktname eignet.

Für die Arbeit wird ein fiktives Unternehmen „gegründet“, das sich als Luftfahrtzulieferbetrieb präsentiert und somit ein attraktives Ziel für einen potenziellen Angreifer darstellt. Der Name dieses Unternehmens bezieht sich hierbei auf die positiven Aspekte der mythologischen Zuschreibungen des „K'uk'ulkan“. Dieser wird als Gott des Windes, des Himmels, der Erde und als Schöpfergott dargestellt.

Für den Angreifer nicht sichtbar ist jedoch die Existenz des gleichnamigen Honeypotsystems. Dieses erfüllt die Aufgabe, manuelle Interaktionen zu „fangen“. Seine Namensgebung orientiert sich an einer alternativen, zur Funktionsweise des Honeypotsystem passenden Darstellung: Der „K'uk'ulkan“ beim Verschlingen eines Menschen.

Quelle: [Wikipedia, 2019]

Abbildung 1.2: Quetzalcōātl (Mayathan: „K'uk'ulkan"), eine mesoamerikanische Gottheit beim Verspeisen eines Menschenopfers.

2 Grundlagen von Honeypots

Der Wunsch nach einem tieferen Verständnis des Verhaltens von Angreifern kann mit verschiedenen Mitteln erfüllt werden. Eines dieser Mittel ist der Aufbau eines Honeypotsystems. Der Begriff des Honeypots beschreibt im allgemeinen Sinne (unter anderem) eine „Person oder Sache, die als Köder einer Falle, eines Betrugs oder einer Intrige handelt“[1]. Im Bereich der digitalen Technologie ist die Rede von einem „Computersystem, das – isoliert vom Rest seines Netzwerks – als Köder einen bösartigen Hacker zur Interaktion verlocken soll, um so die Identität oder Herangehensweise des Täters aufzudecken, ohne das Netzwerk oder seine Daten zu gefährden“[2]. [Dictionary.com, 2020, eigene Übersetzungen]

Oft besteht der Köder aus einem bewusst verwundbaren System und üblicherweise werden Interaktionen mit dem Honeypot detailliert protokolliert, um Erkenntnisse oder Reaktionen daraus ableiten zu können. Meist soll ein Angreifer nicht – oder erst möglichst spät – bemerken, dass er mit einem extra für seinesgleichen eingerichteten System interagiert.

In diesem Kapitel wird ein grober Überblick über verschiedene Motivationen oder Ziele eines Honeypotbetreibers und die hierfür verfügbaren Varianten – mit ihren jeweiligen Risiken – gegeben. Der Fokus liegt hierbei auf den für die Erreichung der Ziele dieser Arbeit relevanten Varianten. Für Alternativen, weitere Aspekte und detaillierte Beschreibungen sei auf Grimes [2005] sowie Provos und Holz [2010] verwiesen.

[1] “a person or thing that acts as a lure or decoy in a trap, scam, or scheme”

[2] “a computer system, isolated from the rest of its network, established as bait to lure malicious hackers into engaging with it, thereby revealing the identity or technique of the perpetrator without endangering the network or its data”

2.1 Unterschiedliche Varianten für unterschiedliche Zielsetzungen

Da es für unterschiedliche Zielsetzungen verschiedene Honeypot-Varianten gibt, wird hier ein Überblick über mögliche Einsatzzwecke und die jeweils entsprechenden Alternativen gegeben.

2.1.1 Motivation des Betreibers: Schutz oder Forschung

Eine grundlegende, für die Wahl des Honeypot-Systems entscheidende, Frage ist die Motivation des Betreibers:

Möchte der Betreiber einer bereits vorhandenen Systemlandschaft diese zusätzlich schützen, wird von einem *protection honeypot* gesprochen. Dem bereits vorhandenen Netzwerk wird ein System hinzugefügt, das Schwachstellen enthält, aber speziell überwacht wird. Ein Betreiber könnte beispielsweise von produktiv genutzten Maschinen ablenken wollen. Außerdem könnte er versuchen, durch strategisch positionierte angreifbare Systeme den Aufwand (und somit die Kosten) für den Angreifer zu erhöhen. Honeypots können in diesem Fall auch ein Teil des Angriffserkennungssystems (engl. "*intrusion detection system*", "*IDS*") sein. Wird der Honeypot innerhalb des zu verteidigenden Netzes auf einer Maschine eingerichtet, die keinen sonstigen Zweck verfolgt und keine realen Dienste anbietet, sind unerwartete Aktivitäten auf diesem System grundsätzlich als verdächtig einzustufen. [Grimes, 2005]

Eine andere Motivation für den Betreiber eines Honeypots kann eher allgemeiner Natur sein: Möchte dieser beispielsweise Informationen über aktuell verwendete *Malware* erhalten oder insgesamt mehr über schädlichen Datenverkehr in öffentlichen Netzen erfahren, so bietet es sich für diesen an, einen *research honeypot* einzurichten. Dieser kann Einsichten in das Verhalten von Angreifern und Informationen zu Bedrohungen liefern, ohne die Angriffsoberfläche eines Produktivnetzwerks zu vergrößern. Er ist darauf ausgelegt, zu lernen, wie Angreifer sich verhalten, welche Hilfsmittel sie verwenden und welche Ziele sie verfolgen. Mit Hilfe von Honeypotsystemen können auch Angriffe auf bisher unveröffentlichte Schwachstellen erkannt werden. [Grimes, 2005]

2.1.2 Unterschiedlicher Interaktionsgrad für unterschiedliche Ziele

Ein weiteres Unterscheidungsmerkmal ist der sogenannte *Interaktionsgrad*: Die Spanne reicht hier von der vergleichsweise einfachen Simulation einzelner Dienste bis hin zur Bereitstellung eines kompletten Netzwerks realer Maschinen.

Beispielsweise protokolliert die Software *Kojoney* als *low interactive honeypot* die verwendeten Nutzernamen und – nach erfolgreichem Login mit einem schwachen Passwort – die abgegebenen Befehle. Eine Reaktion des Betriebssystems auf die Befehle des Angreifers wird jedoch nicht simuliert. [Coret, 2006]

Ein Beispiel für einen *medium interaction honeypot* ist die direkte Erweiterung, *Kippo*. Dieses Programm simuliert eine Reaktion auf Nutzereingaben. Dateien können abgelegt und wieder abgerufen werden. [desaster, 2016] Dadurch, dass der Angreifer die Reaktion erhält, die er erwartet, soll er das System nicht sofort als Falle erkennen. Die Hoffnung ist, dass ein Angreifer für die Verfolgung seiner Ziele weitere Hilfswerkzeuge nachlädt. Diese können anschließend vom Verteidiger analysiert werden. Aufgrund der bewusst limitierten Möglichkeiten besteht zu keinem Zeitpunkt die Gefahr, dass der Angreifer reale Systeme beeinträchtigt.

Der Nachteil der vorgenannten Systeme ist, dass ein Angreifer diese schnell als Honeypot erkennen kann und daher meist entsprechend wenig Interaktionen mit ihnen zu erwarten ist. Je realistischer eine Falle, desto unwahrscheinlicher oder später wird das System als solche erkannt. Diejenigen Systeme, die ein komplettes Betriebssystem nicht nur simulieren, sondern tatsächlich bereitstellen, werden als *high interaction honeypot* bezeichnet. Ein Angreifer erhält nach einem erfolgreichen Einbruch Zugriff auf ein komplettes Betriebssystem mit allen üblicherweise verfügbaren Hilfswerkzeugen und Dienstprogrammen. Mit diesen kann er seine Ziele verfolgen und ggf. versuchen, sich weitere Berechtigungen auf der Maschine zu erschleichen. Wird ein solches System von außen beobachtet, ohne dass der Angreifer dies merkt, kann es passieren, dass der Eindringling wertvolle Hinweise hinterlässt, die Aufschluss über seine Motivation geben. Verwendete Werkzeuge können analysiert werden und ihre Herkunft kann nachverfolgt werden. Durch den hohen Realitätsgrad besteht jedoch ein erhöhtes Risiko, dass diese Systeme für weitere Angriffe missbraucht werden und realer Schaden an anderen oder gar fremden Systemen angerichtet wird. [Provos und Holz, 2010]

2.2 Risiken beim Betrieb eines Honeypots

Beim Umgang mit potenziell kriminellen Angreifern besteht immer das Risiko, selbst in rechtliche oder ethische Dilemmata zu geraten. Hierfür gibt es teilweise Lösungsansätze, diese sind jedoch nicht Teil dieser Arbeit. Die Risiken sollen trotzdem nicht unerwähnt bleiben:

2.2.1 Rechtliche Probleme

Ein großes Problem stellt die Gefahr dar, dass die Ködersysteme für weitere Angriffe – ggf. auch gegen Dritte – missbraucht werden können. Ein wichtiges Detail hierbei ist, dass in diesem Fall bei dem angegriffenen System die *IP-Adresse* des Honeypot-Betreibers angezeigt wird. Abgesehen davon, dass dies dazu führen kann, dass eine etwaige Anzeige also beim Honeypot-Administrator eintrifft, ist es grundsätzlich nicht wünschenswert, einem Kriminellen eine Infrastruktur zur Verfügung zu stellen, die dieser missbrauchen kann.

2.2.2 Ethische Probleme

Ein ethisches Problem – das potenziell auch ein rechtliches werden kann – ist die Argumentation, dass das Bereitstellen eines absichtlich verwundbaren Systems einen Angreifer überhaupt erst zu einem Angriff „verführen“ könnte (im englischen Sprachraum als “*entrapment*” bezeichnet). Dies kann ggf. dazu führen, dass mit Hilfe eines Honeypotsystems erworbene Beweismittel in einem Gerichtsverfahren nicht verwendet werden können.

3 Aufbau der anzugreifenden Umgebung

Damit ein Honeypotsystem seine Aufgabe erfüllen kann, wird eine Art Köder benötigt mit dem Angreifer angelockt werden: Eine jede Falle muss so beschaffen sein, dass der angebotene Köder von der potenziellen Beute sowohl als lohnenswert als auch als realistisch wahrgenommen wird. Der Aufwand, der hierfür betrieben werden muss, ist auch abhängig von der zu fangenden Beute: Für einen Honeypot mit der Aufgabe, automatisierte Angriffe zu protokollieren, kann es reichen, einen simulierten *ssh-Server* anzubieten.[1] Das Ziel dieser Arbeit ist jedoch die Protokollierung manueller Interaktionen, daher muss die anzugreifende Umgebung realistisch genug sein, um für einen manuellen Angreifer von Interesse zu sein. Zusätzlich muss ein Weg gefunden werden, automatisierte Angriffe nicht zu begünstigen, um die Protokolldateien nicht übermäßig mit ihnen zu belasten.

Vertreter der Zielgruppe dieser Arbeit haben eventuell bereits eine Umgebung, die, gegebenenfalls mit kleineren Anpassungen, als Köder verwendet werden kann. Um zu forcieren, dass das später zu entwickelnde Honeypotsystem sich an den Bedürfnissen der Anwender orientiert – und nicht andersherum eine bestehende Umgebung zu stark an das System angepasst werden muss – wird zuerst eine Köderumgebung entwickelt.

3.1 Repräsentation einer fiktiven Firma

Um als lohnenswertes Ziel zu erscheinen, wird als Köder eine Firma simuliert. Das Unternehmen präsentiert sich selbst als mittelständisches Industrieunternehmen, das mit maßgeschneiderten Lösungen eine potenzielle Zulieferfirma für größere Industriezweige ist. Dies ist eine Anspielung auf Fälle, in denen die IT-Infrastruktur solcher Zulieferer als Einfallstor zu höherwertigen Zielen genutzt wurde. Ein Beispiel hierfür ist eine 2019 bekannt gewordenen Angriffsserie auf mehrere Partnerunternehmen von Airbus, die über Umwege auf die Kommunikationskanäle des Konzernnetzwerks abzielte. [Böhm, 2019]

[1] Siehe auch Abschnitt „Unterschiedlicher Interaktionsgrad für unterschiedliche Ziele“ (2.1.2)

Um einen realistischen Eindruck zu erwecken, wird für die fiktive Firma eine Website erstellt, die ein Informations- und Interaktionsangebot für die Kunden simuliert. Für die Präsentation der Firma werden allgemein gehaltene Inhalte erstellt.

3.1.1 Angestellte der fiktiven Firma

Die Anzahl und die Rollen der Angestellten werden definiert. Für die Mitarbeiter werden firmenwebsiteübliche Vorstellungstexte formuliert und – um die Seite möglichst realistisch aussehen zu lassen – Fotos von Personen eingefügt. Hier stellt sich allerdings die Frage, woher diese Fotos genommen werden sollen, da aus naheliegenden Gründen keine Abbildungen von real existierenden Personen verwendet werden können. Um dieses Problem zu lösen wurde auf den Internetdienst thispersondoesnotexist.com zurückgegriffen. Mit Hilfe von maschinellen Lernverfahren werden dort Porträts von nicht existierenden Personen[2] generiert. Die auf diese Art kreierten Angestellten mit ihren ausgedachten Namen und einer ihnen zugewiesenen Rolle innerhalb der Firma werden in Abbildung 3.1 gezeigt.

Zu den Fotos wurden Personenbeschreibungen formuliert, um den Angestelltenprofilen eine gewisse Tiefe zu verleihen und dem Firmenauftritt glaubwürdiger zu gestalten. Der auf Abbildung 3.1(d) vorgestellte IT-Administrator „*Lucas*“ ist die Schlüsselperson für den weiteren Verlauf.

Frank Berlepsch, CEO „Als Gründungsmitglied von Kukulkan ist Frank Berlepsch einer der erfahrensten Mitarbeiter. Viele Jahre arbeitete er in verschiedenen Abteilungen einer Fluggesellschaft, wo er Gabriella Sanchez-Ortiz kennenlernte. Gemeinsam machten sie sich selbständig und gründeten Kukulkan.“

Gabriella Sanchez-Ortiz, CTO „Im mexikanischen Valladolid aufgewachsen, gelangte die Informatikerin durch ihre berufliche Tätigkeit in einem großen Luftfahrtunternehmen nach Deutschland. Unter ihrer Ägide wird die Entwicklung von unserem Portfolio vorangetrieben. Aus der Mythologie der Maya, die Teil des mexikanischen Kulturerbes ist, ging unser Firmenname hervor.“

Svea Jensen, HR „Unser fröhliches Nordlicht Svea kümmert sich um alle Personalangelegenheiten und hat immer ein offenes Ohr.“

[2] Und Katzen (thiscatdoesnotexist.com) und diverse andere Dinge (thisxdoesnotexist.com).

(a) Frank Berlepsch, CEO (b) Gabriella Sanchez-Ortiz, CTO (c) Svea Jensen, HR (d) **Lucas Alves, IT-Administrator**

(e) Sebastian Frühschütz, Ingenieur (f) Alexander Arnold, Ingenieur (g) Pepito, Office-Cat

Quelle: Alle Abbildungen basierend auf [Karras u. a., 2020] und [Nvidia, 2020]

Abbildung 3.1: Die fiktiven Mitarbeiter der nicht existierenden Firma

Lucas Alves, IT „Lucas kümmert sich liebevoll um die Einrichtung und Pflege unserer IT. Ob Webserver, Kollaborationsmöglichkeiten oder die Einrichtung unserer Desktops, Lucas ist ein wahres Multitalent!“

Sebastian Frühschütz, Ingenieur „Sebastian ist die treibende Kraft in der Entwicklung unserer Produkte. Er bedient unsere CAD-Software wie kein Zweiter.“

Alexander Arnold, Ingenieur „Gleich nach seinem Studium der Maschineningenieurwissenschaften an der ETH-Zürich stieg Alex bei Kukulkan ein. Gemeinsam mit Sebastian leitet er die Forschung und Entwicklung unseres Fachbereichs der Faserverbundwerkstoffe.“

Pepito, Office-Cat „Einen Office-Dog hat jeder, aber nur wir haben eine Office-Cat! Pepito schnurrt sich seit Gründung der Firma in unsere Herzen und in die unserer Kunden.“

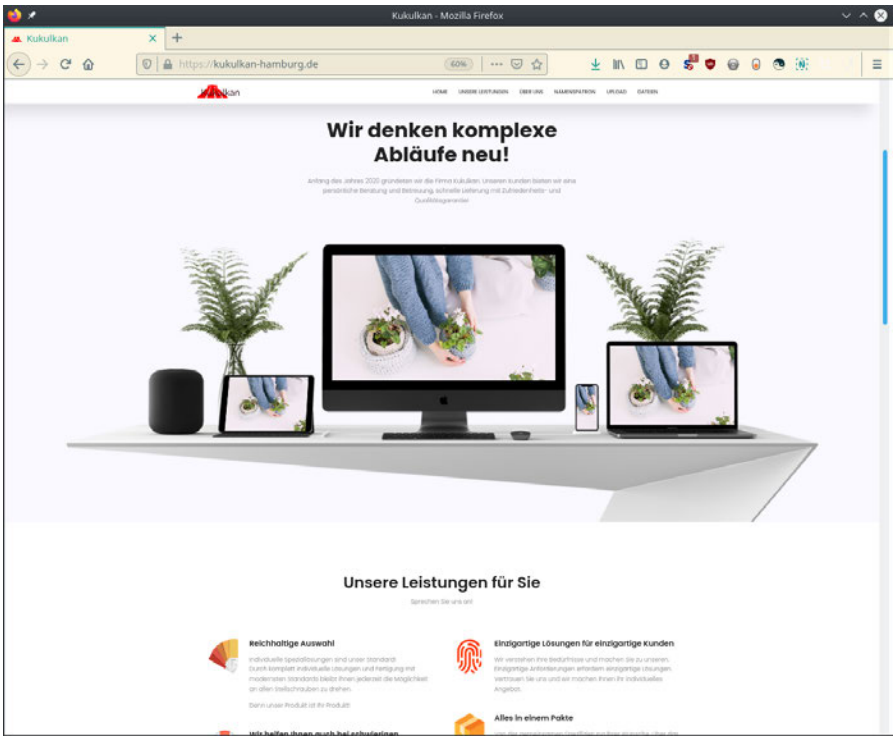
3.1.2 Website der fiktiven Firma

Mit Hilfe eines Onlinewerkzeugs wird ein zum Stil der Firma passendes Logo (Abbildung 3.2(b)) kreiert. Das Logo wird auf der Website verwendet, die auf Grundlage einer für nicht kommerzielle Einsatzzwecke kostenfreien Vorlage erstellt wird. Ein Ausschnitt der Website wird in Abbildung 3.3 gezeigt.

(a) Icon als Grundlage (b) Firmenlogo

Quelle: „Created my free logo at LogoMakr.com“

Abbildung 3.2: Erstellung des Firmenlogos



Quelle: Eigener Screenshot, Template unter Lizenz „CC BY 3.0“ colorlib.com

Abbildung 3.3: Website im modernen Design einer aufstrebenden Firma

3.2 Wahl der zu verwendenden Dienste

Aufgrund der Anforderung, dass die auf dem Honeypot installierten Dienste Angreifer anlocken sollen und weil der Firmenserver möglichst realistisch aussehen soll, werden weit verbreitete Dienste verwendet:

Der Webserver soll mit der Webserversoftware *apache2* betrieben werden und mit Hilfe der Skriptsprache *php* dynamische Internetseiten anzeigen. Für einen höheren Realitätsgrad soll – wie heutzutage üblich – die Kommunikation zwischen Besuchern der Website und dem Webserver mittels Transportverschlüsselung (*transport layer security*, TLS) verschlüsselt werden.

Zusätzlich zur Website soll für die Fernwartung *ssh* verwendet werden, wobei der Zugang nicht korrekt abgesichert wird, wie später genauer beschrieben.

Durch den modularen Aufbau des Kukulkan-Honeypots können von einem anderen Betreiber andere Dienste verwendet und auch andere Lücken hinterlegt werden.

3.3 Einfallstore in der simulierten Firmeninfrastruktur

In der simulierten Firmeninfrastruktur sollen bewusst Sicherheitslücken enthalten sein. Für diese Arbeit werden zwei Einfallstore vorbereitet. Die verwendeten Sicherheitslücken erwecken den Eindruck, dass sie versehentlich durch einen unerfahrenen Administratoren verursacht worden sind. Dies soll dazu beitragen, dass ein potenzieller Angreifer diese Sicherheitslücken nicht als absichtlich hinterlegt identifiziert.

3.3.1 Einfallstore in der Firmenwebsite

Die Website wird so angepasst, dass ein „Kunde“ die Möglichkeit hat, Dateien oder Abbildungen wie beispielsweise Fertigungsskizzen bequem über ein entsprechendes Formular hochladen zu können. Der hierfür benötigte Code in der Scriptsprache *php* wurde direkt aus dem *php*-Anwenderhandbuch[3] entnommen. Dieser bietet jedoch keine ausreichende Überprüfung der hochgeladenen Datei. Ein Ausschnitt des Codes wird in Listing 3.1 gezeigt. Bleibt dieser Code unverändert, erhält ein Angreifer die Möglichkeit, beliebige Dateien auf den Server zu laden.

[3] www.php.net/manual/de/features.file-upload.post-method.php (Zuletzt abgerufen 4. Oktober 2020)

```
1  <?php
2  //...
3  $uploaddir = '/var/www/uploads/';
4  $uploadfile = $uploaddir . basename($_FILES['userfile']['name']);
5  //...
6  if (move_uploaded_file($_FILES['userfile']['tmp_name'], $uploadfile)) {
7      echo "Datei ist valide und wurde erfolgreich hochgeladen.\n";
8  } else {
9      echo "Möglicherweise eine Dateiupload-Attacke!\n";
10 }
11 //...
```

Quelle: PHP-Dokumentationsgruppe [2020], verwendet unter Lizenz CC BY 3.0
(Auslassungen mit `//...` gekennzeichnet)

Listing 3.1: Irreführender *php*-Dateiupload-Code aus dem Anwenderhandbuch

Der Webserver ist in der Standardkonfiguration bestrebt, jede Datei mit der Dateiendung `.php` auszuführen. Führt er eine vom Angreifer hochgeladene *php*-Datei aus, erhält der Angreifer *remote code execution* auf den Server und kann ihn somit fernsteuern und für seine eigenen Einsatzzwecke missbrauchen.

Besonders fatal ist an dieser Stelle, dass der in der Dokumentation angegebene Beispielcode durch die Textausgaben für einen unerfahrenen Systemadministrator den Eindruck erwecken könnte, gegen genau eine solche Dateiupload-Attacke zu schützen (siehe Zeilen 6-10). In der gezeigten Form schützt der Code jedoch nicht gegen Angriffe.[4]

Für den bewusst verwundbaren Webserver wird dieser Beispielcode unverändert übernommen.

3.3.2 Einfallstor in der Fernwartung

Der Fernwartungszugang wird – entsprechend der Standardkonfiguration des Hosters, die zur Anwendung kommt, wenn kein SSH-Schlüssel hinterlegt wird – mit einem Benutzernamen und einem Passwort abgesichert. Da der Fokus dieser Arbeit nicht auf der Beobachtung von automatisierten Angriffen liegt, sondern auf der Auswertung manueller Angriffe, wurde keine offensichtliche Kombination, wie beispielsweise `root:admin` verwendet. Ein privilegierter Anwender auf dem Server wird mit dem Benutzernamen

[4] Gegenmaßnahmen gegen diesen Angriff wäre zum einen, hochgeladene Dateien besser zu überprüfen. Php liefert hierfür Funktionen, die anhand der sogenannten „magic bytes“ am Anfang einer Datei den Dateityp ermitteln. Da auch dies nicht ausreicht, sollte der Webserver so konfiguriert werden, dass es php untersagt wird, Dateien, die sich in einem festgelegten Uploadverzeichnis befinden, auszuführen. Ein entsprechender Hinweis fehlt auf der Handbuchseite mit dem Beispielcode.

`lucas` und dem Passwort `kukulkan` angelegt.[5] Im Gegensatz zu einem automatisierten Skript wird einem manuellen Angreifer die Fähigkeit unterstellt, Informationen aus dem Kontext der Website entnehmen zu können. Aus den erstellten Mitarbeiterprofilen sollte ein Mensch ableiten können, dass „Lucas“ der Vorname des IT-Administrators ist. Aufgrund der mit dieser Rolle verbundenen Berechtigungen könnte es sich lohnen, diesen Benutzernamen als Angriffsvektor zu verwenden.[6]

[5] Mir wurde erst später bekannt, dass das Passwort „kukulkan“ doch in der weit verbreiteten Passwortliste „rockyou.txt“ enthalten ist.

[6] Gegen leicht zu erratende Passwörter gibt es diverse Gegenmaßnahmen. Tools wie *fail2ban* sperren oder drosseln die Anmeldung für eine ip-Adresse nach mehreren erfolglosen Anmeldeversuchen. Eine sicherere Anmeldung verweigert die Authentifikation mittels Passwort und lässt ausschließlich Schlüsselpaare zu.

4 Grundlagen der Prozessisolation

Ein selbst gestecktes Ziel dieser Arbeit ist, den Betreiber vor den Aktionen des Angreifers zu schützen. In einem ersten Schritt ist die Vorgehensweise dafür näher zu definieren. Dabei stellen sich folgende Fragen: Wie kann eine Art Grenze zwischen dem Angreifer und dem Betreiber gezogen werden? Kann der Angreifer „eingesperrt“ werden? Welche Möglichkeiten sollen ihm bleiben, mit dem System zu interagieren, besonders im Hinblick darauf, dass Interaktionen mit dem Server eines der Hauptinteressen dieses Vorhabens sind?

Für dieses Projekt soll eine realistische Serverbetriebssystemumgebung aufgebaut werden, in der ein Angreifer sich frei bewegen kann, wobei seine Interaktionen jedoch zuverlässig protokolliert werden. Entscheidend ist, dass dieser für ihn abgegrenzte Bereich wie bei einem halbdurchlässigen Einwegspiegel von der Seite des Beobachtenden beliebig einsehbar ist. Der Beobachtete hingegen sollte idealerweise jedoch nicht merken, dass er überwacht wird.

Um dem Gesamtsystem keinen Schaden zufügen zu können und damit die Protokollierung vertrauenswürdig bleibt, darf der Angreifer auf keinen Fall aus dieser geschützten Umgebung ausbrechen. Es wird also eine wirksame Abgrenzung zwischen dem Angreifer und dem Beobachter benötigt.

4.1 Security Boundaries

Eine solche Abgrenzung zwischen bestimmten Bereichen eines Systems wird als *security boundary*, oder auch als *trust boundary* bezeichnet. [Rice, 2020]

Eine der Möglichkeiten, eine solche isolierte Umgebung aufzubauen, bietet der Einsatz von *virtuellen Maschinen*. Diese ermöglichen es, innerhalb eines Betriebssystems in einem

isolierten Bereich ein weiteres Betriebssystem zeitgleich auszuführen. Die Virtualisierungen oder Nachbildungen (engl. "to emulate") der Hardware, beispielsweise der *CPU*, des *Arbeitsspeichers* sowie der Ein- und Ausgabegeräte wie Tastatur, Maus und Bildschirm stellen hierbei die Grenzübergänge zwischen diesen Systemen dar.

Innerhalb einer *virtuellen Maschine* wird ein komplettes Betriebssystem inklusive der für die Interaktion mit der Hardware benötigten Software-Komponenten gestartet. Dies erlaubt es, ein anderes Betriebssystem als *Gast* zu betreiben. Wird innerhalb einer solchen Umgebung eine Überwachungssoftware installiert, ist diese jedoch für den Angreifer sichtbar. Dies gilt besonders dann, wenn der Angreifer sich für das verwundbare System Administrationsrechte erschleicht. Überwachungsprozesse könnten dann manipuliert werden und etwaigen Protokolldateien ist nicht mehr zu vertrauen.[1]

Alternativ zu *virtuellen Maschinen* gibt es noch eine andere Möglichkeit, eine abgeschottete Umgebung einzurichten, die die Anforderungen an Sicherheit und Beobachtbarkeit erfüllt: *Container* liefern die Möglichkeit, Prozesse auszuführen, die von anderen Prozessen isoliert sind und keinen Zugriff auf Ressourcen haben, die ihnen nicht explizit zugewiesen wurden.

Der Systemkern (engl. "*kernel*") des Betriebssystems des *Host* bleibt weiterhin für die Kommunikation der Software mit der Hardware zuständig und kann somit den Zugriff auf sämtliche Ressourcen des Computers verwalten, einschränken und auch überwachen.

4.2 Der Grundgedanke von Containern

Dem Konzept des *Containers* liegt die Einhaltung von klar definierten *security boundaries* zugrunde. Container sollen demnach zur Separierung einzelner Prozesse dienen und dazu beitragen, dass andere Prozesse nicht in ihrer Funktion beeinträchtigt werden.[2]

Spätestens seit 1979 werden Wege gesucht, Prozesse voneinander zu isolieren. Das in diesem Jahr erstmals eingesetzte *chroot* war aber fehleranfällig[3]. Seit dem Jahr 2000 wurde

[1] Hierfür gibt es Lösungen, wie beispielsweise ein „append-only“ Protokoll auf einem anderem Webserver. Dies wäre jedoch ebenfalls für einen Angreifer deutlich zu sehen und würde zusätzlich eine weitere Adresse des Betreibers offenlegen.

[2] Einzelne Linux-Entwickler wie Glauber [2012] gehen sogar so weit zu sagen, dass Container jetzt erst korrigieren, was eigentlich von Anfang an selbstverständlich hätte sein müssen.

[3] Es war möglich, aus einer *chroot*-Umgebung wieder zurück in die übergeordnete *chroot*-Umgebung zu wechseln.

für das Betriebssystem *freeBSD* das Konzept der *jails* entwickelt. Seitdem wurden von einigen Firmen Bemühungen unternommen, eigene Versionen des Linux-*Kernel* zu entwickeln, die eine Prozessisolation gewährleisten sollen. [Osnat, 2020] Ergebnisse dieser Anstrengungen wurden nach und nach dem Hauptentwicklungszweig des Linux-*Kernel* zugeführt. Erst seit 2007 ist es durch die Einführung von *PID-Namespaces* (siehe Abschnitt „PID namespace“ (4.3.1)) möglich, Prozesse zu starten, die kein Wissen über ihre wahren Elternprozesse besitzen. [Emelyanov und Kolyshkin, 2007] Dies stellt eine der Voraussetzungen für die zeitgleiche Ausführung mehrerer Betriebssysteme dar.

Hinsichtlich des Container-Begriffs sei betont, dass mit dem Wort „Container“ die Abgrenzung bestimmter Prozesse bezeichnet wird, die auf einem Computer auf derselben Hardware und durch dasselbe Betriebssystem ausgeführt werden. Hierfür werden keine Zwischenebenen benötigt, was im Vergleich zu *virtuellen Maschinen* zu einer höheren Performanz führt.

Durch den Einsatz eines Containers wird kontrolliert, was für bestimmte Prozesse sichtbar ist, also auch welche Ressourcen durch einen Prozess genutzt werden können und in welchem Umfang. [Rice, 2020]

4.3 Sicherheitsmaßnahmen der Container

Das Design von Containern kann im Grunde als sicher betrachtet werden, da Zugriffe auf sämtliche Ressourcen blockiert und/oder eingeschränkt werden können. [Rice, 2020]

Durch Fehlkonfigurationen oder Programmierfehler können jedoch unerwartete Probleme auftreten. Das Design und die grundlegenden Mechanismen von Containern gelten jedoch weithin als sicher, auch wenn es widersprüchliche Aussagen dazu gibt. Diese werden im Abschnitt „Sicherheitsprobleme bei Containern“ (4.5) angesprochen.

Die Grundgedanken, auf denen diese Mechanismen basieren, sind unabhängig von einer konkreten Implementierung und werden im Folgenden beispielhaft anhand ihrer Umsetzungen im Linux-*Kernel* beschrieben. Bei anderen Betriebssystemen werden teilweise andere Bezeichnungen für ähnliche Konzepte verwendet.

4.3.1 Namensräume

Im Rahmen der Prozessisolation spielen sogenannte Namensräume (engl. "namespaces") eine entscheidende Rolle. Im *Linux-Programmierhandbuch* werden sie folgendermaßen beschrieben:

> Ein Namensraum verpackt eine globale Systemressource innerhalb einer Abstraktion, die dafür sorgt, dass es für einen Prozess innerhalb dieses Namensraums so erscheint, als hätte er seine eigene isolierte Instanz dieser globalen Ressource.[4] (Kerrisk [2020b], eigene Übersetzung)

Namensräume bieten also eine Unterteilungsmöglichkeit für grundlegende Eigenschaften und Ressourcen des Betriebssystems und der darauf ausgeführten Prozesse. Erst die Einführung des Konzepts des Namensraumes erlaubt die Unterteilung einer Maschine in „mehrere" Maschinen. Zuvor existierte genau ein (impliziter) Namensraum pro Maschine. Unterscheidbare Namensräume erlaubt es, mehrere solcher unterschiedlichen Abgrenzungsebenen auf einer Maschine bereitzustellen, was wiederum die Ausführung mehrerer Betriebssysteme auf selbiger ermöglicht.

Durch eine Gruppierung von Prozessen, die wiederum von anderen Prozessen isoliert sind, lässt sich das Konzept eines Containers umsetzen. Die hierfür wichtigsten Eigenschaften werden im Folgenden beschrieben und anhand von Beispielen demonstriert:

UTS namespace

Jedem laufenden Betriebssystem ist ein *Hostname* zugeordnet, der zur Identifikation der Maschine dient sowie ein *Domain name*, der dessen Zugehörigkeit zu einer Gruppierung von Rechnern anzeigt. [Tobias u. a., 2009]

Die Vergabe eines eigenen Namensraumes für die Benennungen des Host und der Domain ist der grundlegende erste Schritt, der die Ausführung mehrerer Betriebssysteme auf einer Maschine ermöglicht: Der Hostname eines innerhalb eines Containers ausgeführten Betriebssystems kann von dem Hostnamen des ausführenden Rechners abweichen. Ebenso kann das Gastbetriebssystem Teil einer anderen Domäne sein, da der *domain name* ebenfalls abweichen kann.

[4] "A namespace wraps a global system resource in an abstraction that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource." [Kerrisk, 2020b]

Mount namespace

Das Dateisystem ist die Repräsentation aller verfügbaren Dateien, meist hierarchisch organisiert in Verzeichnissen. Bei *unixoiden* Betriebssystemen wird diese Hierarchie dargestellt, indem einzelne Ebenen mit dem Symbol / getrennt werden. Oberstes Element dieser Baum-Hierarchie ist das sogenannte Wurzelverzeichnis. Es wird als einzelnes / dargestellt. [Kerrisk, 2010]

Wird beispielsweise ein USB-Massenspeichermedium (umgangssprachlich: „USB-Stick“) angeschlossen, wird dieses als Verzeichnis innerhalb des Dateisystems angezeigt. Das verbreitete Betriebssystem Microsoft Windows beispielsweise verwaltet seine Geräte mit Laufwerksbuchstaben wie `C:\`. Auf *unixoiden* Betriebssystemen wie *GNU/Linux* ist es möglich, ein solches Verzeichnis an jeder beliebigen Stelle innerhalb des Dateisystems einzuhängen. Ein USB-Stick könnte am Ort `/media/myStick/` eingehängt werden oder an jeder beliebigen anderen freien Stelle.

Die Verwendung verschiedener *mount*-Namensräumen erlaubt es, bestimmten Prozessen eine unterschiedliche Repräsentation des vorhandenen Dateisystems anzubieten. Den Prozessen innerhalb eines solchen Namensraums wird eine individuelle Hierarchie angezeigt. [Kerrisk, 2020a]

Auf diese Art kann für isolierte Prozesse die Sichtbarkeit von eingehängten Geräten unterbunden werden. Außerdem kann durch eine gesonderte Repräsentation des Dateisystems einem Prozess auch ein abweichendes Verzeichnis als Wurzelverzeichnis vorgegeben werden. Ein auf diese Art vom ursprünglichem Dateisystem isolierter Prozess hat keine Möglichkeit, auf eine höhere Hierarchie-Ebene zu wechseln. Er kann auch keine Kenntnis von einer höheren Ebene erhalten, da die Ebene / per Definition die oberste Ebene darstellt. Für diesen Prozess ist die Existenz einer höheren Ebene sozusagen „unvorstellbar“.

Zudem ist es möglich, die für Prozesse wichtigen Kontextinformationen unterschiedlich darzustellen: Die Verzeichnisse `/sys` und `/dev` enthalten Informationen über angeschlossene Hardware. Wird einem einzelnen Prozess ein zugeschnittenes `/sys`- oder `/dev`-Verzeichnis präsentiert, kann diesem Prozess gegenüber somit auch die Gegenwart bestimmter Hardware verschleiert werden. Ebenso ist es möglich, dem Prozess nicht die real verbauten Komponenten anzuzeigen, sondern andere, simulierte Hardware.

PID namespace

Jeder Prozess besitzt eine eindeutige Prozessidentifikationsnummer (*pid*), anhand derer er identifiziert werden kann. Die Nummerierung startet bei 1 und wird für jeden neu ausgeführten Prozess inkrementiert. [Kerrisk, 2020d]

Das Verzeichnis `/proc` enthält (unter anderem) Informationen zu allen Prozessen, die derzeit auf der Maschine ausgeführt werden. Einzelne Prozesse können hier Informationen über ihre Umgebung erhalten. Dieses Verzeichnis wird durch den Kernel bereitgestellt, der die Verwendung der Namensräume verwaltet. Alle Prozesse sehen in dem `/proc`-Verzeichnis nur die Prozesse innerhalb ihres eigenen Namensraumes. [Kerrisk, 2010]

Da auch die Prozesse innerhalb eines Containers auf dem Host ausgeführt werden, sind diese für das Hostbetriebssystem sichtbar und können von ihm verwaltet werden. Prozessen werden bei Verwendung von pid-Namensräumen aus Sicht des Host mehrere Prozessidentifikationsnummern zugewiesen, die vom Kernel verwaltet werden. Aus Sicht des Gasts gibt es pro Prozess innerhalb des Containers genau eine *pid*. Vertieft wird dies in Abschnitt „Veranschaulichung unterschiedlicher Namensräume“ (4.3.2).

Der erste Prozess (*pid 1*) übernimmt üblicherweise die Aufgabe eines *init*-Systems: Dieses erledigt als Bestandteil des Betriebssystems das Starten und die Verwaltung aller weiteren Prozesse. Durch die Verwendung der *pid*-Namensräume ist es möglich, Prozessen eine von der Realität abweichende *pid* vorzugaukeln. Dadurch kann innerhalb eines Containers ein neuer Prozess mit der *pid 1* gestartet werden, der innerhalb des Containers die Aufgaben des *init*-Systems übernimmt. Außerhalb des Containers hat dieser aber eine andere *pid*. [Kerrisk, 2020d]

Dies wiederum schafft in Kombination mit den anderen genannten Namensräumen die Voraussetzungen für die Erstellung eines sogenannten *full system container*. Diese werden in Abschnitt „Full System Container“ (4.4.2) betrachtet.

User namespace

Auf Mehrbenutzerbetriebssystemen wie beispielsweise Windows oder Systemen mit *Linux*-Kernel verwaltet das Betriebssystem mehrere Benutzer und Benutzergruppen. Diese werden anhand einer eindeutigen Anwenderidentifikationsnummer (“*user identification number*” “*uid*”) bzw. Gruppenidentifikationsnummer (“*group identification number*”, “*gid*”)

voneinander unterschieden. Verschiedenen Nutzerkonten können unterschiedliche *Nutzungsrechte* eingeräumt werden. Auf Systemen, die Linux verwenden, ist das Benutzerkonto mit der *uid* 0 per Definition das Systemadministratorkonto. Sämtliche Zugriffskontrollen und Berechtigungseinschränkungen werden ignoriert, wenn ein Anwender mit der *uid* 0 auf Dateien oder Systemressourcen zugreifen möchte.

Benutzernamensräume ermöglichen dem Betriebssystem, einem Benutzer innerhalb eines Namensraumes eine bestimmte *uid* und außerhalb eine andere *uid* zuzuteilen. Innerhalb eines Containers kann einem Nutzerkonto also beispielsweise die *uid* 0 (Systemadministrator) zugewiesen sein, während demselben Benutzerkonto außerhalb des Containers über eine andere Identifikationsnummer andere (oder eben keine) Berechtigungen zugewiesen sein können. [Kerrisk, 2020e]

Network namespace

Dienstprogramme, die auf einem Rechner ausgeführt werden, werden über das Netzwerk mittels einer Kombination aus Netzwerkadresse und Portnummer angesprochen, wobei die Netzwerkadresse den Rechner und die Portnummer eine bestimmte Anwendung auf dieser Maschine adressiert.

Der Netzwerknamensraum isoliert die Systemressourcen, die für Netzwerkverbindungen benötigt werden: Netzwerkgeräte, Routingtabellen, Firewallregeln, Portnummern und ähnliche Ressourcen. [Kerrisk, 2020c]

Das bedeutet konkret für Container, dass bei der Verwendung von Netzwerknamensräumen einzelne Dienste (also Prozesse) eine jeweils eigene Sicht auf das Netzwerk erhalten. Auf diese Art können beispielsweise Portnummern mehrfach verwendet werden. Zudem können mehrere Prozesse aus ihrer jeweils eigenen Sicht beispielsweise den Port 80 verwenden, wenn jeder dieser Prozesse diesen Port in einem anderem Namensraum blockiert. Dies wäre ohne Namensräume nicht möglich.

Sollen die Dienste, die innerhalb eines Namensraumes betrieben werden, auch von außen, beispielsweise aus dem Internet, ansprechbar sein, muss auf dem Host ein entsprechendes Mapping vorgenommen werden, das Anfragen dem gewünschten Namensraum zuordnet.

4.3.2 Veranschaulichung unterschiedlicher Namensräume

Zur Demonstration folgt ein Beispiel, das mehrere der genannten Konzepte veranschaulicht. Hierfür wird die Prozessliste aus Sicht des Gastbetriebssystems (Listing 4.1) und des Host (Listing 4.2) gegenübergestellt.

Durch den Befehl `ps` können Prozesse aufgelistet werden. Mit dem Argument `-e` („every“) wird angewiesen, alle Prozesse aufzulisten. Das Argument `-o pid,ppid,uid,user,cmd` gibt vor, welche Informationen angezeigt werden sollen, wobei `pid` die Prozessidentifikationsnummer des Prozesses ist, `ppid` die *pid* des Elternprozesses, `uid` die ID des Anwenders, unter dessen Identität der Prozess ausgeführt wird, `user` der Name des Benutzerkontos und `cmd` der ausgeführte Befehl mit Argumenten. Der Zusatz `--forest` gibt an, dass die Hierarchie der Prozesse mittels eines Liniendiagramms dargestellt werden soll.

```
1  lucas@container1:~$ ps -eo pid,ppid,uid,user,cmd --forest
2    PID  PPID   UID USER     CMD
3      1     0     0 root     /sbin/init
4  ...
5    108     1     0 root     /usr/sbin/sshd -D
6  32230   108     0 root      \_ sshd: lucas [priv]
7  32267 32230  1001 lucas     |   \_ sshd: lucas@pts/0
8  32268 32267  1001 lucas     |       \_ -bash
9   1765 32268  1001 lucas     |           \_ sleep 420
10  1820 32268  1001 lucas     |           \_ ps -eo pid,ppid,uid,user,cmd --forest
11 ...
12   118     1     0 root     /usr/sbin/apache2 -k start
13  6705   118    33 www-data  \_ /usr/sbin/apache2 -k start
14 ...
```

(Auslassungen mit ... gekennzeichnet)

Listing 4.1: Prozessliste aus Sicht des Gasts

In der ersten Zeile des Listing 4.1 ist zu sehen, dass der unprivilegierte Anwender *lucas*[5] an einer Maschine mit dem Hostnamen `container1` angemeldet ist.

Als *pid* 1 wurde das *init-System* gestartet, also der erste Prozess, dessen Aufgabe das Koordinieren und Starten aller anderen Prozesse ist.

In Zeile 9 ist der `sleep`-Befehl zu sehen, der als Beispiel für einen Befehl genutzt wird, der für längere Zeit ausgeführt wird. Er hat innerhalb des Containers in diesem Beispiel die

[5] Siehe Abschnitt „Angestellte der fiktiven Firma“ (3.1.1)

pid 1765. Gestartet wurde dieser von dem Anwender *lucas*, dem die *uid* 1001 zugeordnet ist.

Zeile 10 zeigt den Prozess, der gerade ausgeführt wird, also die Auflistung aller Prozesse.

In Zeile 12 bis 14 ist der *apache2*-Webserver zu sehen. Der Hauptprozess des Webservers wurde mit Administrationsrechten gestartet, sodass dieser den privilegierten Port 80 verwenden kann. Die folgenden Prozesse, die einzelne *HTTP*-Anfragen annehmen und verarbeiten (*worker threads*), haben die Rechte des Nutzers *www-data*, der wiederum nur die für seine Aufgabe unbedingt notwendigen Berechtigungen besitzt.

Im Listing 4.2 wird derselbe Zeitpunkt mit dem gleichen Befehl dargestellt, aber aus Sicht des Host: Der Anwender ist *root*, angemeldet an der Maschine mit dem Hostnamen „host“.

```
1  root@host:~# ps -eo pid,ppid,uid,user,cmd --forest
2    PID  PPID     UID USER     CMD
3  ...
4      1     0       0 root     /lib/systemd/systemd --system --deserialize 35
5  ...
6  20007     1       0 root     /usr/sbin/sshd -D
7  27203 20007       0 root      \_ sshd: root@pts/1
8  27295 27203       0 root          \_ -bash
9  15914 27295       0 root              \_ ps -eo pid,ppid,uid,user,cmd --forest
10 ...
11 23432     1       0 root     [lxc monitor] /var/snap/lxd/common/lxd/containers u1
12 23466 23432 1065536 1065536  \_ /sbin/init
13 ...
14 23747 23466 1065536 1065536      \_ /usr/sbin/sshd -D
15 27317 23747 1065536 1065536      |   \_ sshd: lucas [priv]
16 27385 27317 1066537 1066537      |       \_ sshd: lucas@pts/0
17 27386 27385 1066537 1066537      |           \_ -bash
18 15200 27386 1066537 1066537      |               \_ sleep 420
19 23771 23466 1065536 1065536      \_ /usr/sbin/apache2 -k start
20 11806 23771 1065569 1065569      |   \_ /usr/sbin/apache2 -k start
21 ...
22 13335     1       0 root     SCREEN -d -m -S kukulkan sysdig -c kukulkan
23 13336 13335       0 root      \_ sysdig -c kukulkan
```

(Auslassungen mit ... gekennzeichnet)

Listing 4.2: Prozessliste aus Sicht des Host

Da es sich dabei um dieselbe Maschine handelt, ist der abweichende Hostname ein Beispiel für die Verwendung verschiedener *UTS*-Namensräume. Auch die unterschiedlichen

pid-Namensräume sind sichtbar: Zeile 4 zeigt den Prozess, der wirklich[6] *pid* 1 hat, das *init-System* des Host.

Zeile 9 zeigt die Ausführung des aktuellen Befehls durch *uid* 0 (*root*).

In Zeile 11 wird der Container gestartet; die Prozesse der folgenden Zeilen (12–20) werden also innerhalb des Containers ausgeführt. Die Prozesse entsprechen denen aus Listing 4.1:

Zeile 12 zeigt den Aufruf des *init-Systems* innerhalb des Containers. Statt wie aus Sicht des Containers als *pid* 1 gestartet, hat dieser auf dem Host die *pid* 23466. Außerdem hat dieser Prozess nicht wie aus Sicht des Gastsystems die *uid* 0, sondern die *uid* 1065536. Auf dem Host ist dieser *uid* kein Username zugewiesen. Ebenfalls sind auf dem Host für diese *uid* keine Berechtigungen hinterlegt, was effektiv bedeutet, dass diese *uid* auf dem Host *nobody* ist und selbst keinen Zugriff auf Ressourcen erhält.

Zeile 18 zeigt den Demonstrationsprozess. Dieser wird jetzt mit der *pid* 15200 angezeigt und nicht mehr wie aus Sicht des Containers mit *pid* 1765. Auch die UserID ist anders: Zu erkennen ist, dass aus Anwender 1001 (*lucas*) Anwender 1066537 geworden ist. Für diese Anwendernummer existiert auf dem Host keine Zuordnung zu einem Benutzernamen. Der *apache2*-Webserver läuft als Anwender 1065569 und nicht wie innerhalb des Containers als 33 (*www-data*).[7]

Die Zeilen 22 und 23 zeigen den auf dem Host vom Systemadministrator (*root*) ausgeführten *sysdig*-Überwachungsprozess, der aus dem Container heraus nicht sichtbar ist.

4.3.3 Control groups

Mit der Verwendung von *control groups* kann einzelnen Prozessen oder einer Gruppierung von Prozessen ein Limit für bestimmte Ressourcen gesetzt werden. Der Zugriff auf *CPU*, *Arbeitsspeicher* oder Netzwerk kann somit eingeschränkt und geregelt werden.

Da sowohl diese Ressourcen als auch die *control groups* innerhalb des *Kernels* verwaltet werden, kann dieser die Limitierungen wirksam durchsetzen. Diese Einschränkungen sind keine „Empfehlung“: Limitierte Prozesse sind nicht in der Lage, mehr als die zugewiesenen Ressourcen zu nutzen.

[6] Da es sich um eine virtuelle Maschine eines Cloudanbieter handelt, ist das nicht ganz korrekt.
[7] 1065536 + 33 = 1065569

Diese Einschränkungen lassen sich nicht nur auf einzelne Prozesse anwenden, sondern werden innerhalb der Prozesshierarchie vererbt. Ist ein Prozess limitiert, fallen auch die von ihm gestarteten Unterprozesse unter die Limitierungen. Da ein Container aus Prozesssicht ein *init*-Prozess mit diversen Unterprozessen ist, können daher auch ganze Containerumgebungen auf diese Weise wirksam limitiert werden.

4.3.4 Veranschaulichung der control groups

Listing 4.3 und 4.4 zeigen den verfügbaren Arbeitsspeicher (1945 Mebibyte) auf dem Hostrechner und in einem Container ohne Arbeitsspeicherlimitierung. Ebenfalls zu sehen ist die getrennte Abrechnung des verwendeten Speichers innerhalb und außerhalb des Containers: Innerhalb des Containers werden 56 MiB verwendet, während auf dem Host durch andere Prozesse (inklusive der Containerverwaltung selbst) 455 MiB genutzt werden.

```
root@host:~# free --mebi
              total        used        free      shared  buff/cache   available
Mem:           1945         455          74           2        1461        1357
Swap:             0           0           0
```

Listing 4.3: Verfügbarer Arbeitsspeicher des Host (1945 Mebibyte)

```
user@container1:~$ free -mebi
              total        used        free      shared  buff/cache   available
Mem:           1945          56        1674           1         260        1934
Swap:             0           0           0
```

Listing 4.4: Verfügbarer Arbeitsspeicher des Gasts (ohne Limit ebenfalls 1945 Mebibyte)

Mit Hilfe des Befehls `lxc config set container1 limits.memory 1GiB` wird `lxc` angewiesen, das Limit des Arbeitsspeichers für den Container mit dem Namen `container1` auf 1 Gibibyte (= 1024 Mebibyte) zu beschränken. Listing 4.5 zeigt die erfolgreiche Limitierung auf 1024 MiB.

```
user@container1:~$ free -mebi
              total        used        free      shared  buff/cache   available
Mem:           1024          55         683           1         260         944
Swap:             0           0           0
```

Listing 4.5: Limitierter Arbeitsspeicher des Gasts (1024 Mebibyte)

Die Limitierung wird vom Gastsystem so wahrgenommen, als sei tatsächlich nicht mehr Arbeitsspeicher vorhanden. Auch andere Methoden, die angewendet werden können, um den verfügbaren Arbeitsspeicher anzufragen, zeigen den limitierten Wert, wie beispielsweise der im `/proc/meminfo` hinterlegte Wert[8] belegt. (Siehe Listing 4.6)

```
1 user@container1:~$ cat /proc/meminfo | grep MemTotal
2 MemTotal:        1048576 kB
```

Listing 4.6: Limitierter Arbeitsspeicher des Gasts (`/proc/meminfo`)

Der Befehl `lxc config set container1 limits.cpu.allowance 50ms/100ms` beschränkt die maximale CPU-Auslastung, die durch Prozesse innerhalb von `container1` ausgelöst werden kann auf 50 % der verfügbaren Prozessorleistung. Auch die verfügbare Netzwerkgeschwindigkeit kann auf ähnliche Art limitiert werden.

4.3.5 Mandatory Access Control

Als zusätzliche Sicherheitsmaßnahme verwenden die meisten Containerimplementierungen Zugriffskontrollen, die auf *Kernel*-Ebene unter anderem festlegen, welche Prozesse auf welche Dateien und Verzeichnisse zugreifen dürfen (engl. "*mandatory access control*"). Bei GNU/Linux-Betriebssystemen sind die weitverbreitetsten Implementierungen *AppArmor* und *SELinux*. Mit diesen Werkzeugen können Ressourcen für bestimmte Anwendungen freigeschaltet und/oder blockiert werden.

Im Rahmen dieser Arbeit wurden die vorkonfigurierten *Ubuntu*-Standardeinstellungen übernommen. Daher wird auf eine detailliertere Beschreibung verzichtet.

4.4 Container-Arten

Verschiedene Containerimplementationen wurden unter unterschiedlichen Voraussetzungen und mit anderen Zielsetzungen konzipiert, aber grundsätzlich sind die verwendeten Konzepte miteinander vergleichbar, oder nutzen sogar die gleichen, vom *Kernel* bereitgestellten, Techniken.

Grundsätzlich geht es bei der Isolation durch Namensräume immer darum, einzelne Prozesse oder Prozessgruppen voneinander zu trennen. Wird ein Container nur für einen

[8] Die Umrechnung der Einheiten *GiB* zu *kB* wird hier vom System nicht einheitlich vorgenommen.

einzigen Zweck verwendet, beispielsweise für den Betrieb eines Webservers, wird von einem *application container* gesprochen. Wird in einer isolierten Prozessgruppe ein eigenes *init-System* gestartet, also die Komponente des Betriebssystems, die alle weiteren Prozesse koordiniert, wird von einem *full system container* gesprochen, in dem ein weiteres Betriebssystem ausgeführt wird.

4.4.1 Application Container

Diese Container isolieren einzelne Applikationen vom Rest des Betriebssystems. Im Falle eines erfolgreichen Angriffs begrenzen sie (bei entsprechender Implementation) das Ausmaß dessen, was ein Angreifer sehen kann. Ihr Haupteinsatzzweck ist aber eine einheitliche Verbreitung von Anwendungen, da innerhalb des Containers alle benötigten Konfigurationsdateien oder Anwendungsbibliotheken (*libraries*) für den Betrieb der gewünschten Applikation bereitgestellt werden und somit unabhängig vom Host in der richtigen Version vorliegen.[9]

4.4.2 Full System Container

Unter dem Begriff „Betriebssystem“ werden oft unterschiedliche Konzepte vereinheitlicht. Zum einen wird so die zentrale Verwaltung der Hardwareressourcen bezeichnet und zum anderen die grundlegenden Standardprogramme, wie Eingabekonsolen, grafische Benutzeroberflächen, Dateimanager und Texteditoren. [Kerrisk, 2010, S. 22]

Die Verwaltung der Hardware übernimmt bei Containern der *Kernel* des Hostsystems. Die Standardprogramme, die den Rest des Betriebssystems ausmachen, den Teil also, mit dem der Endanwender aktiv interagiert, können innerhalb eines Containers ausgetauscht werden.

Generell werden beim Start eines Betriebssystems, wenn die Hardwareverwaltung bereit ist, die Benutzeranwendungen gestartet. Die Koordinierung dieser übernimmt der *init*-Prozess. Seine Hauptaufgabe ist das Erstellen und Verwalten der Prozesse, die für ein funktionsfähiges System benötigt werden. [Kerrisk, 2010, S. 33]

[9] Code unabhängig vom verwendeten Hostbetriebssystem ausführen zu können ist eine Eigenschaft, die sich auch Angreifer zunutze machen können, um beispielsweise *Cryptomining*-Anwendungen zu verbreiten. [Morag, 2020]

Wird innerhalb isolierter Namensräume ein solcher *init*-Prozess gestartet, kann dieser den sichtbaren Teil eines Betriebssystems verwalten und – ähnlich wie bei einer *virtuellen Maschine* – kann ein anderes Betriebssystem innerhalb des eigentlichen Betriebssystems gestartet werden, sofern dieses mit dem verwendeten *Kernel* kompatibel ist oder der *Kernel* eine entsprechende Kompatibilitätsschicht anbietet.

Ein Linux-Gast auf einem Linux-Host entspricht dem Grundgedanken der Containerisierung, während der *Kernel* von Windows mit der Virtualisierungstechnologie *Hyper-V* eine Kompatibilitätsschicht anbietet, die es erlaubt, Linux-Container auf Windows-Betriebssystemen auszuführen. [Neugebauer, 2017] Aufgrund der verwendeten Virtualisierung entspricht dies eher dem Grundgedanken von *virtuellen Maschinen* als dem des Container-Konzepts.

4.5 Sicherheitsprobleme bei Containern

Trotz aller beschriebenen Sicherheitsmaßnahmen können Sicherheitsprobleme entstehen, beispielsweise durch falsche Konfiguration oder durch Sicherheitslücken im *Kernel* selbst.

4.5.1 Unsichere Konfiguration

Abgesehen von schlicht fehlerhaft durchgeführten Konfigurationen können auch korrekt umgesetzte Anforderungen, die jedoch mit den Sicherheitsbestimmungen in Konflikt stehen, zu Problemen führen.

Beispielsweise ist der Internetzugriff immer problematisch, da die Möglichkeiten eines erfolgreichen Angreifers nicht mehr lokal beschränkt sind (siehe auch Abschnitt „Risiken beim Betrieb eines Honeypots“ (2.2)). Der Zugriff auf ein internes Netzwerk (*Intranet*) begünstigt oder erlaubt gegebenenfalls eine *horizontal privilege escalation*, also die Übernahme mehrerer Maschinen.

Auch die Verwendung von Verzeichnissen, auf die sowohl der Host als auch Gastbetriebssysteme Zugriff haben, kann zu Sicherheitsproblemen führen.

4.5.2 Sicherheitslücken im Kernel

Dadurch, dass bei der Verwendung von Containern – im Gegensatz zu *virtuellen Maschinen* – der *Kernel* mit dem Host geteilt wird, können Sicherheitslücken im *Kernel* ausgenutzt werden, was zu Problemen führen kann. Wird eine solche Sicherheitslücke aufgespürt und ausgenutzt, kann es trotz aller aufgezeigten Sicherheitsmaßnahmen passieren, dass der *Kernel* aus einem Container heraus dazu gebracht wird, Code mit den höchsten Privilegien auszuführen. [Edwards, 2019]

4.5.3 Vergleich mit virtuellen Maschinen

Auch beim Einsatz von *virtuellen Maschinen* treten gelegentlich ähnlich gravierende Probleme auf, bei denen das Risiko besteht, dass der Host aus einer *virtuellen Maschine* heraus übernommen wird. Eines der aktuellen Beispiele betrifft das Virtualisierungsprodukt *VMware*, bei dem Schwachstellen[10] im Grafiktreiber dazu führen können, dass aus einer *virtuellen Maschine* heraus Code auf dem Hostrechner ausgeführt wird. [VMWARE, 2020]

Trotz gelegentlicher Kritik an Containern oder Virtualisierungen haben beide Konzepte ihre Daseinsberechtigung. Besonders im Bereich der Container wird aktuell stetig an einer wirksamen Isolierung weitergearbeitet, wie beispielsweise der neue *time namespace* zeigt, der 2019 in die Version 5.6 des Linux-*Kernel* eingearbeitet wurde. Dieser Namensraum erlaubt die Isolierung der gesamten Zeitmessung, also beispielsweise der lokale Uhrzeit oder der Ausführungszeiten von Prozessen. [Safonov, 2019]

4.6 Containerlösungen

Es existieren diverse Containerimplementierungen mit jeweils unterschiedlichen Zielsetzungen. Durch die Fortschritte der letzten Jahre im Linux-*Kernel* stützen sich einige dieser Implementierungen auf die gleichen vom *Kernel* angebotenen Funktionalitäten.

[10] *Use After Free*: Auf einen bereits freigegebenen Bereich des Arbeitsspeichers wird vom Programmcode später noch einmal referenziert. Ist ein Angreifer in der Lage, an dieser freien Stelle Programmcode zu hinterlegen, kann es passieren, dass dieser im Kontext der verwundbaren Anwendung ausgeführt wird. [mitre, 2020]

Die Auswahl der zu verwendenden Container-Technologie sollte daher danach getroffen werden, welche Anforderungen man an die Container stellt und welches Produkt diese am besten erfüllt. Da einigen Umsetzungen die gleichen Technologien zugrunde liegen, die jeweils vom *Kernel* selbst angeboten werden, kann im direkten Vergleich dieser kein großer Unterschied hinsichtlich der Performance auftreten.

Eine häufig verwendete Containerlösung ist *docker*, dessen Einsatzzweck sich auf *application container* beschränkt. Durch enge Zusammenarbeit mit Microsoft ist es möglich, Linux-Docker-Container auch auf Windows auszuführen, was dem Versprechen der Host-Unabhängigkeit deutlich entgegenkommt. [Neugebauer, 2017] Vergleichbare und direkt konkurrierende Alternativen zu *docker* stellen *rkt* und *podman* dar. Das Hauptargument für *podman* ist die besondere Fokussierung auf *unprivilegierte* Container, die nicht im Kontext des root Users ausgeführt werden, was geringere Auswirkungen im Fall eines Containerausbruchs verspricht.

Für verschiedene Betriebssysteme und *Kernel* gibt es unterschiedliche Containerströmungen. So werden etwa im Bereich des *freeBSD*-Betriebssystems *jails* verwendet, um Prozesse vom Rest des Systems zu isolieren. [Riondato, 2020]

Im Bereich der Betriebssysteme mit einem Linux-*Kernel* ist eine der Implementierungsformen, deren Hauptaugenmerk auf vollständig isolierten *full system containern* liegt, die möglichst realistisch erscheinen sollen, die von *Canonical* vorangetriebenen *linux container* (*lxc*). Die Einrichtung eines *lxc*-Containers wird in Abschnitt 6.5.3 beschrieben.

4.7 Einsatzmöglichkeiten für den Honeypot

Innerhalb eines *full system containers* lässt sich ein eigenes Betriebssystem starten, auch wenn dieses sich Komponenten mit dem Host-Betriebssystem teilt. Mit dem Gast-Betriebssystem kann interagiert werden wie mit einem eigenständigen Betriebssystem. Ein Endanwender wird kaum[11] bemerken, ob das Betriebssystem als einziges System auf einer Maschine (*bare metal*) oder innerhalb eines Containers ausgeführt wird.

Container liefern durch die konsequente Verwendung von *namespaces* trotz gelegentlicher Zwischenfälle eine grundsätzlich ausreichende Isolierung zwischen dem Host- und

[11] Es gibt Anzeichen dafür und mit entsprechendem Fachwissen wird ein Anwender seine Umgebung identifizieren können, aber einzelne Containeranbieter arbeiten daran, die Containerumgebungen immer mehr an eine vollwertige Umgebung anzupassen.

dem Gastbetriebssystem. Die Verwendung der Namensräume begrenzt die Ressourcen, die ein Anwender innerhalb eines Containers sehen kann, wobei *control groups* eine wirksame Limitierung dieser darstellen. Somit kann auch einem nicht-vertrauenswürdigem Anwender innerhalb eines Containers Systemverwaltungszugriff gewährt werden. Die Interaktionsmöglichkeiten einer mit dem Honeypotsystem zu überwachenden Personen sind also nahezu uneingeschränkt.

Von außerhalb des Containers ist es möglich, auf dem Host Programme auszuführen, deren Existenz innerhalb des Containers nicht zu erahnen ist. Auf diese Weise kann der Host den Gast unbemerkt beobachten.

Container stellen demnach grundsätzlich eine geeignete Lösung für die im Abschnitt „Anforderungen an das Gesamtsystem“ (1.5) beschriebenen Anforderungen dar.

5 Grundlagen der Ausführungsüberwachung

Eines der Hauptziele des im Rahmen der Arbeit zu erstellenden Werkzeuges besteht darin, eine Möglichkeit zu schaffen, mit der Informationen über das Verhalten eines manuellen Angreifers gesammelt werden können.

Hierfür interessante Informationen können beispielsweise die Namen der vom Angreifer verwendeten Programme sein oder – besonders im Falle von nicht öffentlich verfügbarer *Malware* – die Herkunft dieser. Auch die Art der Informationsbeschaffung des Angreifers, also an welchen Orten er selbst nach Informationen über das kompromittierte System sucht, können lehrreich sein.

Missbraucht ein Angreifer das von ihm übernommene System, so lassen sich Informationen ablesen, beispielsweise über die Herkunft[1] des Angreifers oder über andere Ziele, die er gegebenenfalls mit Hilfe des gekaperten Systems angreift.

Sollte ein Angreifer es schaffen, seine eigenen Zugriffsrechte nach einem erfolgreichen Angriff unerlaubterweise zu erhöhen (engl. "*privilege escalatation*"), so soll ersichtlich werden, auf welchem Wege er dies erreicht hat.

Es soll nicht darum gehen, den erfolgreichen Einstieg ins System zu detektieren. Hierfür gibt es eine Reihe von Angriffserkennungssystemen (engl. "*intrusion detection systems*", "*ids*"), die hier nicht weiter betrachtet werden.

5.1 Mögliche Lösungsansätze

Um derartige Informationen zu erhalten, gibt es verschiedene Lösungsansätze, von denen eine Auswahl kurz vorgestellt werden soll, bevor eine Variante gewählt wird:

[1] Die Herkunft aus Netzwerksicht lässt nicht unbedingt Rückschlüsse auf die geografische Herkunft zu.

Der Netzwerkverkehr zwischen Angreifer und Server könnte protokolliert werden. Beispielsweise erlauben Werkzeuge wie *dsniff* [Song, 2000] oder *bifrozt* [creatorbifrozt, 2016] eine Überwachung aller Befehle der Fernwartungssoftware *ssh*. Eine *man-in-the-middle*-Überwachung wie von *dsniff* funktioniert jedoch mit neueren ssh-Protokollversionen nicht mehr so einfach, *bifrozt* wird seit 2016 nicht weiterentwickelt.

Eine andere Möglichkeit könnte darin bestehen, den virtuellen „Bildschirm“ eines laufenden Betriebssystems zu beobachten. Einige Untersuchungswerkzeuge für *Malware*, wie beispielsweise die Cuckoo Sandbox, erzeugen (unter anderem) *Screenshots* der *virtuellen Maschine*, in der *Malwaresamples* ausgeführt werden. Dieser Ansatz ist jedoch nur für einen kurzen Zeitraum praktikabel, weil eine Auswertung von Bilddateien einen hohen manuellen Aufwand erfordert.

Einen weiteren spannenden Ansatz verfolgt beispielsweise *Argos* mit einem als „Speicherfärbung“ (engl. “*dynamic taint analysis*”) bezeichneten Vorgehen: Argos emuliert ein Betriebssystem und markiert im Arbeitsspeicher alle Daten, die von Benutzereingaben oder aus dem Internet stammen. Sobald ein entsprechend markierter Speicherbereich ausgeführt wird, wird das System eingefroren. Dies hindert den Angreifer sofort daran, Schaden bei Dritten anzurichten. Der Betreiber des Systems hat anschließend die Möglichkeit, das System zu analysieren und ggf. abgelegte Malware zu extrahieren. [Portokalidis u. a., 2006]. Das System wird jedoch seit ca. 2014 nicht mehr instand gehalten und die Verwendung ist nicht mehr empfohlen.[2]

Wenn es primär nur darum gehen sollte, *Malware* aufzuspüren, könnte ein vergleichsweise einfaches Werkzeug verwendet werden, dass das Dateisystem auf Veränderungen überprüft. Beispielsweise könnte eine komplette Tabelle aller Dateien und ihrer individuellen Eigenschaften erstellt werden, die regelmäßig mit dem Dateisystem abgeglichen wird. Taucht eine neue Datei auf oder wird eine Datei verändert, könnte dies ein Hinweis auf eine gegebenenfalls unerwünschte Aktivität im System sein. Allerdings muss hierbei beachtet werden, dass auch automatisierte Abläufe und die normale Funktion des Betriebssystems Dateien anlegen und verändern können. Werkzeuge wie beispielsweise *rkhunter* verfolgen unter anderem einen solchen Weg.

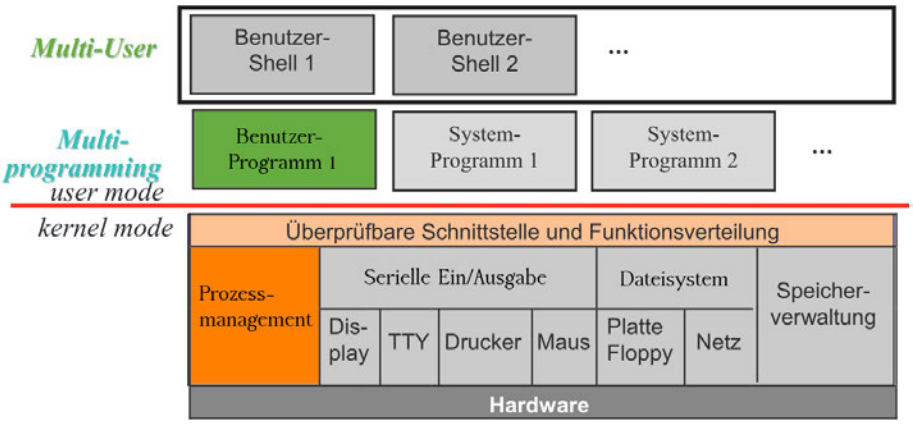
Eine andere Ebene, auf der eine Systemüberwachung stattfinden kann, ist die Schnittstelle zwischen Anwendungsprogrammen und dem Betriebssystemkern: Sämtliche Ressourcen eines Computers werden vom Betriebssystemkern (engl. “*Kernel*”) verwaltet.

[2] “Argos is ancient and we have not maintained it for years and years. I would not touch this anymore”, Prof. Herbert Bos, Vrije Universiteit Amsterdam, per E-Mail.

Die Anwendungsprozesse fordern den Zugriff auf (unter anderem) Netzwerk und Dateisystem an, auch die Ausführung eines jeden Programms wird vom Betriebssystemkern gesteuert. Wird diese Schnittstelle entsprechend manipuliert, können darüber beliebige Zugriffs- und Ausführungsanfragen protokolliert werden. Dieser Ansatz wird im Folgenden genauer beleuchtet.

5.2 Protokollierung auf Betriebssystemebene

Das was als „Betriebssystem“ bezeichnet wird, ist das Zusammenspiel verschiedener Bausteine, die jeweils unterschiedliche Aufgaben übernehmen. Einige davon sind in Abbildung 5.1 dargestellt. Die für diese Arbeit entscheidenden Teile werden im Folgenden kurz beschrieben.



Quelle: [Brause, 2017], eigene Hervorhebungen.

Abbildung 5.1: Schichtmodell eines UNIX-Kernels

5.2.1 Betriebssystemkern

Ein direkter Zugriff auf Systemressourcen wie beispielsweise *Prozessor*, *Arbeitsspeicher*, *Netzwerk*, *Festplatten* und sonstige Geräte würde einen hohen Koordinierungsaufwand bedeuten und kann gegebenenfalls sogar zu Sicherheits- und Hardwareproblemen führen. Normale Anwendungen werden daher in einem restriktiven Modus ausgeführt und haben *Benutzerstatus* (engl. “*user mode*”). Solche Anwendungen besitzen keine direkte Zugriffserlaubnis auf die Ressourcen des Systems und können auch nicht alle Funktionen (Instruktionen) des Prozessors aufrufen.

Als ein Teil des Betriebssystems hat der Betriebssystemkern als koordinierende Schnittstelle die Aufgabe, sämtliche Systemressourcen und den Zugriff auf diese zu verwalten. Konkurrierender Zugriff auf endliche Ressourcen werden koordiniert. Beispielsweise wird mit Hilfe einer Ablaufsteuerung der Prozesse (engl. "*scheduling*") dafür gesorgt, dass alle Prozesse fair behandelt werden und der Eindruck paralleler Ausführung entsteht. Soll ein Benutzerprogramm auf eine Ressource zugreifen, so adressiert dieses diesen Wunsch an den Kernel, der (ggf. nach u.a. Zugriffsprüfung) die hierfür notwendigen Schritte übernimmt. Das Programm benötigt somit keinerlei Kenntnisse der Details der verwendeten Hardware oder beispielsweise dem tatsächlichen Aufenthaltsort einer Datei.

5.2.2 Systemaufrufe

Die Schnittstelle des Kernel kann von Benutzerprogrammen mit sogenannten „Systemaufrufen" (engl. "*system calls*", kurz "*syscalls*") angesprochen werden. Hinter dieser Schnittstelle werden die Anfragen in die notwendigen Hardwareinstruktionen übersetzt und weitergeleitet. Das Ergebnis wird im Erfolgsfall an das aufrufende Programm zurückgemeldet.

5.2.3 Kernel-Module

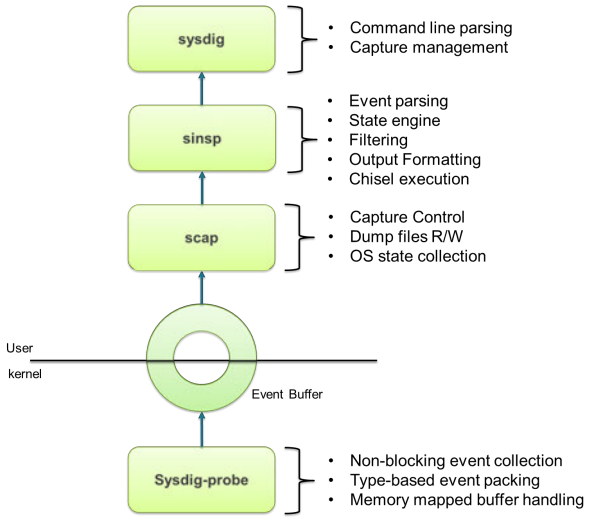
Damit der Kernel diese Aufgabe erfüllen kann, aber nicht sämtliche Instruktionen für jede erdenkliche Hardware dauerhaft im Kernel hinterlegt sein müssen, gibt es die Möglichkeit, den Kernel modular zu erweitern. Hardwarehersteller können somit ein passendes Erweiterungsmodul liefern, ohne dass Betriebssystemhersteller diese Aufgabe erfüllen müssen.

Mit Hilfe solcher Module kann jedoch auch das Verhalten des Kernels manipuliert werden, was sie zu einem mächtigen Werkzeug macht.[3] Ein möglicher Einsatzzweck ist die erweiterte Protokollierung der Interaktionen mit dem Kernel. Hierfür wird der Kernel so angepasst, dass alle *Systemaufrufe* protokolliert werden können.

[3] Wie jedes Werkzeug kann auch dieses missbraucht werden. Ist es einem Angreifer gelungen, den ausgeführten Kernel mit einem bösartigen Kernelmodul zu erweitern, kann dieses beispielsweise die Ausführung bestimmter Prozesse von allen anderen Prozessen verbergen, was einen wirksamen Schutz vor den üblichen Systemdiagnosewerkzeugen darstellt. Diese als *rootkit* bezeichneten Angriffe sind besonders hinterhältig und besonders schwer aufzuspüren. Als aktuelles Beispiel wird das angeblich von russischen Geheimdiensten eingesetzte Rootkit „Drovorub" sehr detailliert in [NSA, 2020] beschrieben.

5.2.4 Sysdig als konkrete Implementierung

Eine konkrete Implementierung der vorgestellten Vorgehensweise ist das Überwachungswerkzeug *sysdig*. Ein Teil der Anwendung wird im Benutzermodus ausgeführt. Um Ereignisse im Kernel protokollieren zu können, wird zusätzlich ein Kernelmodul verwendet. Die Ausführung aller Operationen in Kernelmodulen sind zeitkritisch, weshalb solche Module möglichst in maschinennahen Sprachen wie beispielsweise *c* oder *c++* geschrieben werden. Erschwerend kommt hinzu, dass Programmierfehler schwerwiegende Auswirkungen auf die Stabilität und Sicherheit des Systems haben können. Um dem Endanwender trotzdem die Möglichkeit zu geben, auch aufwendige Filteroperationen und Ausgabeformatierungen vorzunehmen, sendet das Kernel-Modul – wie in Abbildung 5.2 skizziert – alle Ereignisse ungefiltert aus dem geschützten Betriebssystemkern an das unprivilegierte Benutzerprogramm, welches die weitere Verarbeitung vornimmt. [Degioanni, 2014]



Quelle: [Degioanni, 2014]

Abbildung 5.2: Kommunikation zwischen sysdig-Kernelmodul und user space Client

Da der Kernel die Systemaufrufe koordiniert und das angesprochene Kernel-Modul Teil dieses Kernels wird, können alle Aufrufe aus dem Kernel extrahiert werden. Bei hoher Last des Systems und einer hohen Anzahl an Ereignissen kann es passieren, dass der im Benutzermodus laufenden Programmteil nicht alle Meldungen zeitnah verarbeiten kann. Im Extremfall kann auch der *event buffer* keine verzögerte Bearbeitung sicherstellen. [Degioanni, 2014] Wird der Angreifer jedoch durch Einsatz eines durch „control groups“ limitierten Containers isoliert, kann eine solche Überlastsituation nicht vom Angreifer provoziert werden.

5.3 Veranschaulichung Systemaufrufprotokollierung

Da es eine Vielzahl von protokollierbaren Events innerhalb des Kernel gibt, müssen die Protokollmeldungen gefiltert werden, um beherrschbar zu bleiben. Ein Aufruf des Befehls `sysdig 'evt.type = execve'` setzt einen Filter, der bewirkt, dass ausschließlich `execve`-Events, also Ausführungs-Systemaufrufe, angezeigt werden. Eine beispielhafte Ausführung des Befehls `cat /etc/passwd` wird protokolliert. Die detaillierte Ausgabe des Protokolls wird in Listing 5.1 gezeigt. Nicht alle ausgegebenen Informationen sind für Auswertungen sinnvoll, teilweise werden andere nützliche Informationen unterschlagen.

```
1 root@host:~# sysdig evt.type=execve
2 ...
3 14630 18:30:34.210803396 0 bash (9532) > execve filename=/bin/cat
4 14631 18:30:34.211226545 0 cat (9532) < execve res=0 exe=cat
  ↪ args=/etc/passwd. tid=9532(cat) pid=9532(cat) ptid=31447(bash) cwd=
  ↪ fdlimit=1024 pgft_maj=0 pgft_min=28 vm_size=360 vm_rss=4 vm_swap=0
  ↪ comm=cat
  ↪ cgroups=cpuset=/.cpu=/user.slice/user-0.slice/session-10.scope.cpuacct=/user.slice/us...
  ↪ env=LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01...
  ↪ tty=34816 pgid=9532(cat) loginuid=0
5 ...
```

Listing 5.1: Standardprotokollierung von `cat /etc/passwd`

Mit Hilfe sogenannter „chisel“ (Deutsch: „Meißel“) können umfangreiche Skripte geschrieben werden, in denen individuelle Filter und weitere Möglichkeiten der Verarbeitung festgelegt werden. Für viele Aufgaben gibt es bereits vorgefertigte chisel. So enthält das Skript `spy_users.lua` alle notwendigen Einstellungen, um manuelle Interaktionen von Anwendern in einem auswertbaren Format zu protokollieren. Die Ausgabe einer einfache Protokollierung des Beispiel-Befehls wird im Listing 5.2 gezeigt. Das Protokollierungsskript ermittelt die Prozess-ID des ersten Prozesses, der eine *shell* darstellt. Auf diese Weise sind mehrere Ereignisse, die aus der selben shell ausgelöst werden, zu einem Strang zusammenfassbar. Protokolliert wird die Prozess-ID des ersten shell-Prozesses in der Aufrufhierrarchie, die Uhrzeit, der aufrufende Benutzer sowie der Befehl, inklusive etwaiger Argumente.

```
1 root@host:~# sysdig -c spy_users
2 31447 18:40:33 root) cat /etc/passwd
```

Listing 5.2: `spy_users.lua`-Protokollierung von `cat /etc/passwd`

Werden von einem Prozess Unterprozesse erzeugt, erhöht sich der Einzug des Protokolls. In Listing 5.3 ist zu sehen, dass die Ausführungshierarchie in dem Protokoll erkennbar ist und dass der Ausführungskontext erhalten bleibt. In diesem Beispiel ändert sich durch den Befehl `sudo` der ausführende Anwender. Zeile 1 zeigt den Aufruf des Befehls `whoami` aus Sicht eines unprivilegierten Anwenders. In Zeile 2 wird mit Hilfe von `sudo` ein Anwenderwechsel zum Systemadministratorkonto ausgelöst. In Zeile 3 ist zu sehen, dass die Aufrufhierarchie ersichtlich ist und dass Anwenderwechsel sichtbar sind. Die angezeigt „oberste“ shell-Prozess-ID ist in diesem Fall die *pid* eines Fernwartungszugangs, weil sysdig auf einem Server ausgeführt wurde. Zu sehen ist also auch, dass alle abgegebenen Befehle dieser shell zugeordnet werden können. Wird ein zweier Zugang geöffnet, erhält dieser eine andere *pid*. Hierdurch bleiben die Zugänge voneinander unterscheidbar.

```
1 2918 18:56:43 uwe) whoami
2 2918 18:56:46 uwe) sudo whoami
3     2918 18:56:46 root) whoami
```

Listing 5.3: `spy_users.lua`-Protokollierung eines Anwenderwechsels

In der Standardvariante werden die Ausführungen auf der Kommandozeile protokolliert. Das Skript wurde daher dahingehend angepasst, dass zum Start eine neue Protokolldatei angelegt wird, die in einem Verzeichnis mit dem Tagesdatum und der aktuellen Uhrzeit abgelegt wird. *Sysdig* ist in der Lage, Containerumgebungen zu erkennen und entsprechende Informationen auszugeben. Daher ist es möglich, Filter zu setzen, die ausschließlich Ereignisse innerhalb eines bestimmten Containers ausgeben. Das Skript wurde so angepasst, dass nur Ereignisse innerhalb eines Containers protokolliert werden. Außerdem wird in der Protokollzeile zusätzlich der Containername mit ausgegeben. Das resultierende Protokollformat ist in Listing 5.4 dargestellt.[4]

```
1 707 21:24:26 <NA>@Container1) whoami
2 707 21:24:31 <NA>@Container1) sudo whoami
3     707 21:24:31 root@Container1) whoami
```

Listing 5.4: Angepasste Protokollierung unter Beachtung des Containerkontexts

[4] Kann der Anwendername innerhalb des Containers nicht korrekt ermittelt werden, wird als Anwendername `<NA>` angezeigt.

6 Aufbau des *Kukulkan high interaction honeypot*

Die in den vorangegangenen Kapiteln vorgestellten Methoden und Techniken können, jeweils für sich betrachtet, die einzelnen Teilaspekte des definierten Ziels erfüllen. Werden diese als Bausteine zu einem Gesamtsystem kombiniert, lässt sich hiermit das übergeordnete Ziel einer sicheren Beobachtung der Interaktionen eines Angreifers erreichen. Dies wiederum entspricht der Bewältigung der ursprünglich gestellten *design challenge*, mehr über das Vorgehen von Angreifern erfahren zu können.

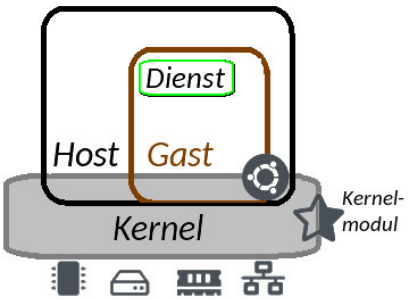
Für die vorliegende Arbeit wird ein *Honeypotsystem* entwickelt, das im Internet bereitgestellt werden kann.[1] Das Ziel ist die Beobachtung der Interaktionen eines Angreifers, unabhängig davon, über welchen Weg dieser sich Zugriff zum System verschafft. Zwei misskonfigurierte Dienste dienen als Einfallstor für ein sicher isoliertes Betriebssystem.

Der modulare Aufbau des Systems erlaubt einen flexiblen Austausch dieser Dienste. Dies befähigt Systemadministratoren, sich einen eigenen, angepassten *Honeypot* aufzubauen, der einer ihnen vertrauten Umgebung entspricht und den sie mit den ihnen geläufigen Administrationswerkzeugen verwalten können.

6.1 Aufbau des Systems

Das zu entwickelnde System setzt sich aus drei wesentliche Komponenten zusammen, deren Schnittstellen – wie in Abbildung 6.1 skizziert – vom Betriebssystem-*Kern* verwaltet werden. Die Dienste der als Köder dienenden anzugreifende Umgebung werden als „Gast“ mit Hilfe eines *full system containers* vom eigentlichen Betriebssystem (Host) isoliert und mit Hilfe eines Kernel-Moduls überwacht.

[1] Die Ergebnisse des Experiments werden im Kapitel „Exkurs: Betrieb des Honeypots und Auswertung“ (8) beschrieben.



Quelle: Eigene Darstellung, Icons: Fontawesome, verwendet unter Lizenz: CC BY 4.0

Abbildung 6.1: Skizze des Systemaufbaus

Die verwendeten Komponenten wurden bereits detailliert beschrieben. Ihre wesentliche Funktionsweise soll hier noch einmal konzentriert zusammengefasst werden:

6.1.1 Der Gast mit den Diensten

Das Gastbetriebssystem und die beiden darauf installierten weit verbreiteten Serveranwendungen sind das Ziel des Angreifers. Die Dienste sind der *apache2*-Webserver, unterstützt durch die häufig verwendete Skriptsprache *php*, sowie die auf vielen Systemen zum Standardumfang gehörende Fernwartungssoftware *ssh*. Im Falle des Webservers wird ein unsicheres Dateiuploadformular verwendet, dessen Code aus den *php*-Herstellerunterlagen entnommen wurde. Ein Angreifer kann dies zur Ausführung seines eigenen Codes (engl. "*remote code execution*") missbrauchen. Der Fernwartungszugriff verwendet in der Standardkonfiguration eine Kombination aus Benutzername und Passwort, was ohne weitere Maßnahmen nur unzureichenden Schutz gegen Einbrüche bietet. Ein erfolgreicher Eindringling kann über die Fernwartung den kompletten Server übernehmen und für seine Zwecke verwenden.

6.1.2 Die Trennung zwischen Host und Gast

Das *Ubuntu*-Betriebssystem, auf dem diese verwundbaren Dienste ausgeführt werden, wird als Gast in einem isolierenden *full system container* betrieben. Die Separation des Gast-Betriebssystems ermöglicht dem Betreiber, den Zugriff auf wichtige Ressourcen, wie CPU, Dateisystem, Arbeitsspeicher und das Netzwerk, zu limitieren. Dies ist jedoch für einen Angreifer nicht offensichtlich.

6.1.3 Die Überwachung

Alle Aufforderungen, eine Datei und somit Code oder Programme auszuführen, werden mit Hilfe eines Kernelmoduls über die Ausführungs-Systemaufrufe (engl. "*exec syscalls*") beobachtet. Das Protokollieren erlaubt, die Vorgehensweise eines Angreifers nach einem Einbruch zu beobachten, was wiederum Einblicke geben kann, wie Eindringlinge sich auf einem gekaperten System verhalten, für welchen Zweck sie dieses missbrauchen, wie sie sich erhöhte Rechte erschleichen (engl. "*privilege escalation*") und welche Software sie zur Hilfe nehmen und woher sie diese beziehen.

6.2 Kompromisse

Auch andere häufig verwendete Dienste – beispielsweise zur Dateiablage oder gemeinsamen Bearbeitung von Dokumenten – könnten geeignete Einfallstore darstellen. Aus der Vielzahl von Diensten, die auf einem Firmenserver verfügbar sein können, muss sich diese Arbeit auf zwei Beispieldienste beschränken.

Obwohl *virtuelle Maschinen* eine wirksamere Isolierung des Gastbetriebssystems bieten und im Cloudserverumfeld häufig eingesetzt werden, überwiegen für dieses Vorhaben die Vorteile der Container. Diese lassen sich durch die gemeinsame Verwendung des Kernel unauffällig und vergleichsweise einfach beobachten.

6.3 Vorgehen

Die Entwicklung des *Honeypots* erfolgt in fünf Vorgehensschritten:

1. Als Basis wird ein Server gemietet, auf dem der *Honeypot* eingerichtet wird.
2. Da ein Teil des *Honeypots* eine Internetseite sein soll, wird eine Domain registriert, über die die Website verfügbar sein wird.
3. Der Container, der das Gastbetriebssystem beinhalten soll, wird eingerichtet. Die für den Gast verfügbaren Ressourcen werden limitiert.
4. Das Gastbetriebssystem wird eingerichtet und die verwundbaren Beispieldienste werden konfiguriert.
5. Das Überwachungsmodul wird an die Bedürfnisse angepasst und aktiviert.

Für jeden dieser Schritte werden wiederum vier Unterschritte durchgeführt:

a. Die Anforderungen an den Schritt werden bestimmt.
b. Die Komponente wird vorbereitet.
c. Die Komponente wird manuell eingerichtet.
d. Notwendige Einrichtungsschritte werden für eine automatisierte Ausführung in Skripten gesichert.

6.4 Anforderungen an das zu entwickelnde System

An alle verwendeten Komponenten werden Anforderungen gestellt, die die einzelnen Bausteine erfüllen müssen. Durch die Kombination der Elemente soll keine dieser Anforderungen vernachlässigt werden, das Gesamtsystem erfüllt jede der genannten Anforderungen.

6.4.1 Anforderungen an das Hosting

Um Angreifer anlocken zu können, soll der Honeypot aus dem öffentlichen Internet dauerhaft[2] erreichbar sein. Hierfür soll ein Server verwendet werden, der bereits mit dem Internet verbunden ist und über eine statische *IP-Adresse* ansprechbar ist. Für die zu erstellende Website wird ein Domainname benötigt, der an den Firmennamen angelehnt sein soll.

Da der Server nur wenige Anfragen gleichzeitig beantworten können muss, sind keine hohen Leistungen erforderlich. Weder ein leistungsstarker Prozessor noch ein besonders großer Arbeitsspeicher werden benötigt. Da zusätzlich zu dem eigentlichen Betriebssystem noch das Gastbetriebssystem installiert werden muss, werden insgesamt mindestens 20 Gigabyte Festplattenspeicher benötigt.

Die niedrigen Anforderungen an die Leistung gehen einher mit der Notwendigkeit, dass der Betrieb des Systems keine hohen laufenden Kosten verursachen darf. Abgesehen davon ist es wichtig, dass die Kosten vorhersehbar sind und auch ein eventueller Ressourcenverbrauch nicht zu unerwarteten Kostensteigerungen führt. Im Zweifelsfall soll das

[2] Etwaige kurzfristige oder unerwartete Ausfälle würden das Vorhaben nicht ernstlich gefährden. Ein gesondertes *service level agreement*, in dem die Erreichbarkeit vertraglich zugesichert wird, ist weder für den Server noch für die Domain erforderlich.

System bei Leistungsspitzen – im Gegensatz zu reale Firmensystemen – lieber deaktiviert werden, als unter Forderung höherer Kosten weiter betrieben zu werden. Auch für den Domainnamen gilt, dass er nur für wenige Monate benötigt wird.

Als Ausschlusskriterium für die Wahl des Serveranbieters darf dieser den Betrieb eines Honeypots nicht verbieten.

6.4.2 Anforderungen an den Host

Systemvoraussetzungen

Das Betriebssystem des Host kann grundsätzlich frei gewählt werden. Die einzusetzenden Technologien der Containerisierung und des Tracing von *syscalls* sind grundsätzlich wie beschrieben oder in ähnlicher Ausprägung für die weitverbreitetsten Betriebssysteme verfügbar.

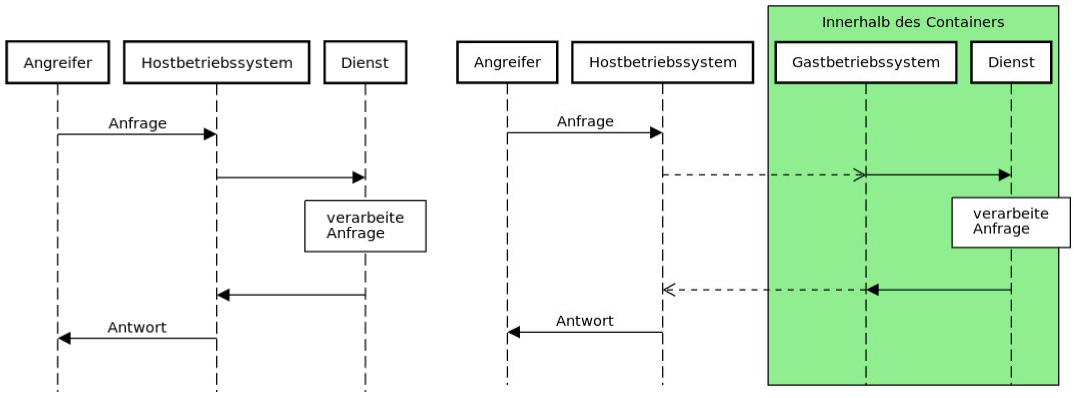
Für Webserver sind auf *Linux* basierende Betriebssysteme weit verbreitet, daher soll für diese Arbeit ein solches verwendet werden. Grundsätzlich ist hierbei die verwendete *Distribution* nicht entscheidend.

Da mit *Kernel*-Version 3.8 *user namespaces* eingeführt wurden [Kerrisk, 2013], wird mindestens diese Version benötigt. Mit Version 5.6 wurde ein zusätzlicher Namespace eingeführt, der alle zeitlichen Aspekte isoliert, dieser ist jedoch für den Betrieb dieses *Honeypots* nicht erforderlich.

Netzwerk

Aus Sicht von außen und aus Sicht von innerhalb des Containers soll der Host transparent – also nicht sichtbar – sein. Für einen Anwender bzw. Angreifer soll die Isolation des Gast-Betriebssystems nicht offensichtlich sein. Die Interaktion mit dem System soll so erfolgen, als ob das Gastbetriebssystem das einzige Betriebssystem auf der Maschine sei, wie in den Abbildungen 6.2(a) und 6.2(b) skizziert. Alle Zugriffe[3] aus dem Internet sollen direkt an das Gastbetriebssystem durchgereicht werden. Vom Gast aus soll kein Zugriff auf nicht explizit freigegebene lokale Maschinen möglich sein.

[3] Mit Ausnahme der Fernwartung des Hostbetriebssystems.



(a) Sicht des Angreifers (b) Sicht des Betreibers

Abbildung 6.2: Ablauf der Interaktion zwischen Angreifer und verwundbaren Diensten

Die *Firewall* soll so konfiguriert bleiben, wie es der Hostinganbieter vorgibt. Sollte es einem Angreifer gelingen, innerhalb des Gasts einen Netzwerkport zu öffnen, soll dieser – wie vom Angreifer erwartet – aus dem Internet erreichbar sein.

Einfache Installation von Standardwerkzeugen

Für eine einfache Einrichtung des Honeypots sollen alle verwendeten Programme aus den *Paketquellen* der *Distribution* installierbar sein. Von den jeweiligen Herstellern freigegebene Sicherheitsupdates verwendeter Programme sind somit direkt verfügbar.

6.4.3 Anforderungen an den Container

Das Gastbetriebssystem soll sich verhalten wie ein eigenständiges Betriebssystem. Es soll möglichst so aussehen, als sei das Gastbetriebssystem das einzig installierte Betriebssystem auf dieser Maschine.

Die Ressourcen, die das Gastbetriebssystem verwenden kann, sollen nicht nur isoliert, sondern auch limitiert sein: Der Gast darf nicht in der Lage sein, beispielsweise CPU, Arbeitsspeicher oder sonstige Hardware des Host vollständig auszulasten.

Ein Anwender, der sich zwar innerhalb des Containers gegebenenfalls sogar als Systemadministrator bewegen darf, darf unter keinen Umständen Zugriff auf nicht ausdrücklich

freigegebene Ressourcen außerhalb des Containers erhalten. Werden einem Anwenderkonto innerhalb des Containers Berechtigungen erteilt, so haben diese außerhalb des Containers keinerlei Gültigkeit.

Ein in dem Container eingesperrter Anwender darf nicht aus dem Containment ausbrechen können. Sollte unerwarteterweise doch ein Ausbruch möglich sein, ist sicherzustellen, dass die Auswirkungen minimal bleiben und ein ausgebrochener Angreifer trotzdem nur geringstmöglichen Zugriff auf den Rest der Infrastruktur hat.

6.4.4 Anforderungen an den Gast

Das Gastbetriebssystem soll so eingerichtet sein, dass es wie ein „typischer“ Firmen-Webserver aussieht. Dieser soll Dienste bereitstellen, die in dem jeweiligen Anwendungsgebiet weit verbreitet sind. Eine Website soll eine fiktive Firma repräsentieren: Sie ist das Aushängeschild und präsentiert die „angebotenen“ Leistungen und die Mitarbeiter.

Der Server muss Einfallstore haben, die sich von einem Angreifer verwenden lassen, um in das System einbrechen zu können. Die Sicherheitslücken sollten jedoch nicht zu offensichtlich sein, um kein Misstrauen zu erwecken. Außerdem sollten sie auch nicht leicht durch typische automatisierte Angriffe ausnutzbar sein, sondern eine manuelle Interaktion eines Angreifers erfordern.

Schon bei der Entwicklung des System soll durch die Verwendung unterschiedlicher „Dienstmodule“ sichergestellt werden, dass der modulare Aufbau von Anfang an berücksichtigt wird. Die Beispieldienste sollen von einem Anwender durch andere ersetzt werden können; weitere Dienste sollen sich leicht entsprechend den vorgestellten Schritten hinzufügen lassen.

6.4.5 Anforderungen an die Überwachung

Die Überwachung eines Angreifers soll die Ausführung von Code auf einem Server, in den eingebrochen wurde, protokollieren. Die Protokollierung muss lückenlos sein und in einem einheitlichen, auswertbaren Format.

Für den Angreifer soll die Protokollierung nicht offensichtlich sein, er soll möglichst nicht bemerken, dass er beobachtet wird.

6.5 Vorbereitungen

Nachdem die Anforderungen formuliert sind, müssen die Schritte vor der eigentlichen Ausführung vorbereitet werden. Hierzu zählt ein Vergleich verschiedener Hostinganbieter und die Auswahl der zu verwendenden Isolationstechnologie.

6.5.1 Vorbereitungen für das Hosting

Um Angreifer anlocken zu können, soll der Honeypot aus dem Internet erreichbar sein. Hierfür wird ein sprechender Domainname benötigt und der Service eines *hosting providers*. Dieser vermietet den Server und gewährleistet die Anbindung an das Internet.

Domainanbieter

Verschiedene Anbieter von Domainnamen bieten unterschiedliche Preismodalitäten am Markt an. Die Angebote lassen sich in zwei hauptsächliche Kategorien einteilen. Einige Anbieter bieten günstige Preise für das erste Jahr, teurere in den Folgejahren, andere bieten ausgeglichene Preise über die Gesamtlaufzeit. Nach einer kurzen Recherche sticht ein Anbieter (*domain.de*) heraus, der im Rahmen einer Werbeaktion mit einem günstigen Preis von 1,69€ für das erste Jahr überzeugt. Da die Domain nur für wenige Monate benötigt wird und sich der Vertrag innerhalb des ersten Jahres kündigen lässt, ist dieses Angebot für das Vorhaben ideal.

Angebotsvergleich Hoster

Aus der Vielzahl der Hostinganbieter werden zwei Anbieter[4] ausgewählt und miteinander verglichen. Sollte keiner der Anbieter ein zufriedenstellendes Produkt im Angebot haben, werden weitere Provider betrachtet. Das Hauptaugenmerk liegt aufgrund der Anforderungen auf einem günstigen und vor allem planbaren Preis. Aufgrund der geringen Anforderungen an die Leistung werden die jeweils günstigsten Cloudserver-Produkte zum Vergleich herangezogen. Außerdem ist zu prüfen, ob mit den angebotenen Diensten ein Honeypot betrieben werden darf. In Tabelle 6.1 werden die beiden Produkte gegenübergestellt.

[4]Mit diesen beiden wurden in in der Vergangenheit bereits gute Erfahrungen gemacht, was sie in die nähere Auswahl gebracht hat.

Anbieter	google Cloud	Hetzner
Produktname	e2-micro	CX11
Leistung	2 vCPU 1 GB Arbeitsspeicher	1 vCPU 2 GB Arbeitsspeicher
Preis	ca. 6,92 $ im Monat Abrechnung pro Sekunde, zusätzlich Kosten für statische IP	2,69 € im Monat Abrechnung pro Monat, auch wenn Maschine ungenutzt
AGB	keine Erwähnung von Honeypots oder Ähnlichem	keine Erwähnung von Honeypots oder Ähnlichem

Tabelle 6.1: Vergleich zweier niedrigpreisiger Cloudserver-Angebote

Weil der Anbieter *Hetzner* – trotz des Hauptnachteils der monatlichen Abrechnung auch ungenutzter Ressourcen – alle genannten Voraussetzungen erfüllt und über einen ganzen Monat gerechnet günstiger ist als *google Cloud*, wird das Cloudserverprodukt *Hetzner CX11* gewählt.

6.5.2 Vorbereitungen für den Host

Als Vorbereitungen für das Hostbetriebssystem ist ein geeignetes System zu wählen. Aufgrund der formulierten Anforderungen soll ein aktuelles *Ubuntu*-Betriebssystem verwendet werden. Dieses kann über die Managementkonsole des Hostinganbieters bezogen werden. Der Hostinganbieter richtet das Basisbetriebssystem und einen Fernwartungszugang ein.

6.5.3 Vorbereitungen für den Container

Eine Implementierung der vorgestellten Konzepte und Techniken zur Isolierung von einzelnen Prozessen bis hin zu einer Gruppe von Prozessen, die die interaktiven Komponenten eines Betriebssystems darstellen, wird als *Linux Container* (*lxc*) bezeichnet. Um einen Container zu starten, übernimmt das Administrationsprogramm *lxc* den Download des benötigten Gast-Dateisystems, erstellt alle notwendigen Dateien für das neue Wurzelverzeichnis, bindet zugeschnittene Versionen der Verzeichnisse `/proc`, `/dev` und `/sys` ein, erstellt die zu verwendenden Namespaces und startet innerhalb des Containers den *init*-Prozess. Durch die zugeschnittenen Systemverzeichnisse verhalten diese sich wie bei einem vollwertigen Betriebssystem. Wird dem Container beispielsweise ein beschränkter

Bereich des Arbeitsspeichers zugewiesen, spiegelt sich die Limitierung in der Darstellung des `/proc`-Verzeichnisses (siehe Demonstration im Kapitel „Veranschaulichung der control groups“ (4.3.4)). *Lxc* übernimmt auf Wunsch auch die Einrichtung des Netzwerkzugriffs vom Gast zum Internet über eine Netzwerkbrücke, die den *network namespace* des Host mit dem des Gasts verbindet.

Lxc erfüllt alle an das Containment gestellten Anforderungen und ermöglicht eine einfache Einrichtung eines *full system containers*. Das Gastbetriebssystem ist „root save“. [lxc project, 2020] Dies bedeutet, dass es – sofern bei der Einrichtung keine groben Konfigurationsfehler passieren – ungefährlich ist, innerhalb des Gasts Systemadministrationsrechte zu vergeben.

6.5.4 Vorbereitungen für den Gast

Die Vorbereitungen des Gasts beinhalten die Planung, wie sich die fiktive Firma präsentieren soll, die Wahl des Gastbetriebssystems und die zu verwendenden Dienste. Einfallstore für den Angreifer sind dabei zu berücksichtigen. Diese Schritte wurden detailliert im Kapitel „Aufbau der anzugreifenden Umgebung“ (3) beschrieben.

6.5.5 Vorbereitungen für die Überwachung

Abgesehen von einer allgemeinen Einarbeitung in die Bedienung der verwendeten Software wird für die Überwachung keine weitere Vorbereitung benötigt. Die Installation und die notwendigen Anpassungen werden im Abschnitt „Manuelle Ausführung für die Überwachung“ (6.6.5) beschrieben.

6.6 Manuelle Ausführung

Nachdem die vorbereitenden Tätigkeiten abgeschlossen sind, muss der Server und das anzugreifende System eingerichtet werden. Um die Einrichtungsschritte einüben zu können, geschieht dies für einen ersten Testserver manuell.[5] Sobald ein Testserver erfolgreich eingerichtet wurde, werden die notwendigen Schritte in einer Skriptsammlung festgehalten, damit das *deployment* weiterer Systeme automatisiert durchgeführt werden kann.

[5] Die hier beschriebenen Schritte sind die, die letztlich zum Erfolg führten; Fehlversuche werden nicht beschrieben.

6.6.1 Manuelle Ausführung für das Hosting

Um einen dauerhaft aus dem Internet erreichbaren Server zu erhalten, wird jeweils ein Benutzerkonto bei dem ausgewählten Hoster sowie dem *domain registrar* angelegt. Nach erfolgreicher Verifizierung kann der Domain über den *A record* eine IP-Adresse zugewiesen werden. Alternativ können der Domain mittels *NS record* andere Nameserver zugewiesen werden, in diesem Fall die Domainserver des Hostinganbieters.

In der Administrationsoberfläche des Cloud-Hostinganbieters können nun *virtuelle Maschinen* bestellt und konfiguriert werden, deren IP-Adressen einer *Domain* oder *Subdomain* zugewiesen werden.

6.6.2 Manuelle Ausführung für den Host

Da vom Hostinganbieter ein komplettes funktionsfähiges Betriebssystem in einer *virtuellen Maschine* bereitgestellt wird, kann dieses als Grundlage genutzt werden. Der Zugang zur Fernwartung wird sicher konfiguriert, indem ein Nutzer angelegt wird, der Administrationsbefehle via `sudo` ausführen kann und sich mit seinem *private-key* am Server anmelden kann. Anschließend wird die Anmeldung für den root-Account unmöglich gemacht, indem das Passwort gelöscht wird. Der Fernwartungszugang wird so eingerichtet, dass die Anmeldung nur noch mit Hilfe eines *private-key* möglich ist.

Da der Zugang auf die Fernwartung üblicherweise über Port 22 freigegeben ist, aber die Fernwartung zukünftig nicht mehr den Host, sondern den Gast erreichen soll, muss der Zugangsport geändert werden. Nach der Änderung wird die Verbindung getrennt und unter Verwendung des neuen Ports neu aufgebaut.

6.6.3 Manuelle Ausführung für den Container

Die Containerisierungslösung *lxc*, der die Verwaltung vereinfachende *lxc daemon* (*lxd*) sowie das für die Überwachung notwendige Paket *sysdig* werden installiert. Nach der Installation übernimmt *lxd* die automatisierte Initialisierung und Einrichtung des Host, damit Container auf ihm gestartet werden können. Außerdem übernimmt *lxd* grundlegende Konfigurationen, wie beispielsweise die Einrichtung einer Netzwerkbrücke vom Host zum Containerdienst, damit aus den Gästen heraus ein problemloser Zugriff auf das Internet möglich ist.

Anschließend wird ein Container gestartet. Den Download des benötigten *Betriebssystemimage* übernimmt *lxc*. Die für das Gast-Betriebsystem verfügbaren Ressourcen, wie beispielsweise Rechenkapazität (CPU) und Arbeitsspeicher, können nun limitiert werden, wie im Abschnitt „Veranschaulichung der control groups“ (4.3.4) beschrieben. Das innerhalb des Containers ausgeführte Gast-Betriebssystem ist über die Netzwerkbrücke vom Host aus ansprechbar und verhält sich wie eine Maschine im lokalen Netzwerk (LAN).

Um Anfragen aus dem Internet, die den Host auf festgelegten Ports erreichen, direkt an den Gast weiterzuleiten, gibt es verschiedene Möglichkeiten. Eine Lösung ist die Verwendung des Netzwerkkonfigurationswerkzeugs *iptables*, mit dem (unter anderem) *Portweiterleitungen* eingerichtet werden können. Der Host wird so konfiguriert, dass alle Anfragen aus dem öffentlichen Netzwerk – mit Ausnahme des Fernwartungsports des Host – an die IP-Adresse des Gasts weitergeleitet werden. Die hierfür notwendigen Konfigurationsbefehle werden in Listing 6.1 gezeigt.

```
1  # ggf vorhandene alte Regeln loeschen
2  iptables -F
3  iptables -F -t nat
4
5  # Weiterleitung aller Ports vom Anfang der Portrange bis 22221
6  iptables -A PREROUTING -t nat -i eth0 -p tcp --dport 1:22221 -j DNAT --to
   ↪  10.46.73.156:1-22221
7  iptables -A FORWARD -p tcp -d 10.46.73.156 --dport 1:22221 -j ACCEPT
8
9  # Weiterleitung aller Ports von 22223 bis zum Ende der Portrange
10 iptables -A PREROUTING -t nat -i eth0 -p tcp --dport 22223:65535 -j DNAT --to
   ↪  10.46.73.156:22223-65535
11 iptables -A FORWARD -p tcp -d 10.46.73.156 --dport 22223:65535 -j ACCEPT
```

Listing 6.1: Weiterleitung aller Ports außer 22222 vom Host zum Gast

6.6.4 Manuelle Ausführung für den Gast

Innerhalb des Gasts wird das Betriebssystem eingerichtet und die zum Grundumfang gehörenden Pakete aktualisiert. Anschließend werden die benötigten Dienste installiert und – wie in Abschnitt „Einfallstore in der simulierten Firmeninfrastruktur“ (3.3) beschrieben – unsicher konfiguriert. Ein privilegierter Benutzer wird mit einem einfach zu erratenen Benutzernamen (`lucas`) und einem verhältnismäßig unsicheren Passwort (`kukulkan`) eingerichtet, der das Gastbetriebssystem auch über die Fernwartung administrieren kann.

Die für den Webserver erforderlichen Dienste werden installiert und bleiben so konfiguriert, wie vom Systemstandard vorgegeben. Zusätzlich zu der absichtlich eingebauten Lücke auf der Website selbst verbirgt sich hier weiteres Potenzial für Sicherheitsprobleme, was aber für den Honeypot nicht relevant ist.[6]

Damit die Website glaubwürdiger erscheint, wird ein offizielles SSL-Zertifikat eingerichtet. Dieses wird von dem Anbieter *Let's Encrypt* bezogen. Der Webserver wird so eingerichtet, dass Zugriffsversuche über das nicht abgesicherte *http*-Protokoll automatisch auf das *https*-Protokoll erweitert werden. Das Zertifikat wird bei jedem Seitenaufruf mit ausgeliefert.

Die vorbereitete Website mit der Sicherheitslücke wird für den Webserver hinterlegt, sodass dieser sie ausliefern kann. Nur die für die Webseite benötigten Dateien werden von außerhalb des Containers in den Container abgelegt. Meta-Informationen von dem verwendeten Versionskonstrollsystem gelangen auf diese Weise nie in den Container.

6.6.5 Manuelle Ausführung für die Überwachung

Für die Überwachung wird die Software *sysdig* installiert. Die Funktionsweise von *sysdig* wurde im Kapitel „Grundlagen der Ausführungsüberwachung“ (5) beschrieben.

Bei der Installation aus den Paketquellen gab es Probleme, die zu einem kompletten Systemabsturz geführt haben, daher wurde – abweichend von der Anforderung, sämtliche Software aus den Paketquellen zu beziehen – die neueste Produktivversion vom Hersteller bezogen und installiert.[7]

Da *sysdig* ein vielseitiges Werkzeug ist, das mehrere verschiedene Module anbietet, die wiederum flexibel konfiguriert und erweitert werden können, lässt sich das Modul, das *syscalls* auf der *Konsole* protokolliert, so anpassen, dass das Protokoll in eine Datei

[6] Beispielsweise ist in der Standardeinstellung von dem Apache Webserver das sogenannte *directory listing* aktiviert. Dieses erlaubt einem Besucher, den Inhalt eines Verzeichnisses zu sehen, selbst wenn der Websitebetreiber keine html-Seite hinterlegt hat. In vielen Fällen ist dies das gewünschte Verhalten. Im Falle von besonderen Verzeichnissen, beispielsweise dem, in dem die Metadaten der Versionskontrollsoftware (z. B. *git*) abgelegt sind, birgt dies die Gefahr von Informationsverlust. [Internetwache.org, 2015] Obwohl in der Standard-Konfigurationsdatei des apache2-Webservers darauf hingewiesen wird und ein Beispielblock angegeben wird, mit dem sich das Problem lösen lässt, ist dieser nicht standardmäßig aktiviert. Sourcecode siehe: [Debian Entwicklerteam, 2020] Für den Honeypot ist dies nicht relevant, da das *.git*-Verzeichnis entfernt wird. Bei dem im Kapitel „Funktionstest durch *Capture-the-flag*-Event“ (7) beschriebenen Vorgehen ist dieses Verhalten allerdings relevant.

[7] Ein *bug report* wurde erstellt, siehe [Krause, 2020a].

geschrieben wird und dass zusätzlich auch die Ausführungszeit notiert wird. Weiterhin wird der Code des Überwachungsmoduls so angepasst, dass nur die gewünschten *syscalls* protokolliert werden.

6.7 Erstellung der Skripte für automatisiertes Deployment

Nachdem der Testserver erfolgreich eingerichtet wurde, werden die notwendigen Einrichtungsschritte zu einer ausführbaren Software kombiniert. Der Kern des *Kukulkan high interaction honeypot* – der isolierte Betrieb und die Beobachtung eines Betriebssystems und seiner Dienste – stellt ein Gerüst dar, das zukünftig auch mit anderen Anwendungen betrieben werden kann.

Für die Installation des Honeypots sind von der manuellen Einrichtung abweichende Schritte notwendig und teilweise führen die Skripte die vorher beschriebenen Schritte in einer abgewandelten Form aus. Dies wird im Folgenden beschrieben.

6.7.1 Automatisierte Ausführung für das Hosting

Das Hosting des Servers erfolgt manuell, wie im Abschnitt „Manuelle Ausführung für das Hosting“ (6.6.1) beschrieben.

6.7.2 Automatisierte Ausführung für den Host

Sowohl die Installationsdateien für den Honeypot als auch die Website als Köder werden mithilfe der Code-Verwaltung *git* bezogen. Hierfür wird auf dem gewünschten Server ein *git-client* benötigt. Das Installationsskript erzeugt ein neues Schlüsselpaar, das als *deployment key* dient: Der *public key* sollte im *repository* hinterlegt werden, damit das Skript den benötigten Code auch aus nicht-öffentlichen Verzeichnissen automatisiert beziehen kann.

Das Skript kopiert die für die Installation benötigten Dateien aus den *repositories*. Über Änderungen an einer Konfigurationsdatei können Einstellungen vorgenommen werden, die von den vorkonfigurierten Optionen abweichen. Beispielsweise kann angegeben werden, unter welcher Domain der Server erreichbar sein wird, damit das Script automatisch ein SSL-Zertifikat für diese Domain beziehen kann.

6.7.3 Automatisierte Ausführung für den Container

Da die Netzwerkkonfigurationen, wie beispielsweise die Weiterleitungen aller Ports zu einem bestimmten Container, durch das Skript verändert werden, wird die aktuelle Konfiguration zwischengespeichert. Wird später ein neuer Container gestartet, kann das Skript auf dieser Grundlage die Weiterleitungen zu diesem neuen Container einrichten.

Abweichend zur manuellen Einrichtung des Honeypots richtet die automatische Installation die Limitierungen der Prozessorleistung, des Arbeitsspeichers und des Netzwerks für den Container erst ein, nachdem alle Dienste innerhalb des Containers installiert sind.

6.7.4 Automatisierte Ausführung für den Gast

Das Installationsskript richtet den Gast ein, indem die notwendigen Dienste innerhalb des Gasts installiert werden. Die benötigten Dateien für die Konfiguration der verwundbaren Dienste werden berücksichtigt und die Daten für die Website werden von außen in den Gast kopiert. Das *repository* mit den Daten der Website wird nicht zum Gast kopiert, da sonst die Gefahr bestünde, Informationen über den Betreiber des Honeypots preiszugeben. Sofern in der Kukulkan-Konfigurationsdatei gewünscht, wird automatisch ein SSL-Zertifikat bezogen und der Webserver entsprechend eingerichtet.

6.7.5 Automatisierte Ausführung für die Überwachung

Im Gegensatz zur manuellen Einrichtung soll die automatisierte Installation den Server so konfigurieren, dass dieser für einen längeren Zeitraum ohne manuelle Interaktion betrieben werden kann. Da die Überwachungsssoftware *sysdig* in der verwendeten Form nicht dafür konzipiert ist[8] und im Laufe der Zeit Probleme aufgetaucht sind, wird der Server mit Hilfe des Dienstes für zeitgesteuerte Ausführung (*cron*) so konfiguriert, dass das Überwachungsskript einmal täglich neu gestartet wird.

[8] Ein verwandtes Produkt, *sysdig falco*, wäre für den Dauerbetrieb besser geeignet gewesen. *Falco* hat aber einen anderen Einsatzzweck.

6.8 Verwendung der erstellten Skripte

Mit den in den Skripten festgehaltenen Einrichtungsschritten kann ein Linux-Server mit wenigen Handgriffen zu einem *Kukulkan* Honeypotsystem umgewidmet werden.

Die Einrichtung erfolgt in zwei Schritten, wobei der erste Schritt aus der Übertragung der für die Installation benötigten Skripte und eventuellen Anpassungen in der Konfiguration besteht, während die komplette Installation anschließend automatisiert erfolgt. Die vom Skript durchgeführten Einzelschritte entsprechen den Beschreibungen der letzten Abschnitte. Die Struktur der Skripte wird im Anhang „Ausführung der Installationsskripte“ (A.1) aufgelistet.

6.9 Veröffentlichung als open-source-Projekt

Das entwickelte System wurde als *open-source*-Projekt unter der „BSD-3-clause“-Lizenz [BSD, 2020] veröffentlicht und kann unter gitlab.com/kukulkanhoneypot eingesehen und bezogen werden. Die gewählte Lizenz erlaubt es anderen Entwicklern, das System selbst für eigene Zwecke einzusetzen und beliebig anzupassen.

Um für andere Entwickler einen Mehrwert zu liefern, wurde eine englischsprachige Installationsanleitung geschrieben und als Teil des Codes ebenfalls veröffentlicht. [Krause, 2020b]

7 Funktionstest durch *Capture-the-flag*-Event

Um zu überprüfen, ob das entwickelte System wie geplant funktioniert, also die Interaktion eines Angreifers mit dem Server protokolliert werden kann, wurde nach der Entwicklungsphase ein *capture-the-flag*-Event als abschließender Test veranstaltet.

Als Einsteig in dieses Kapitel wird ein Überblick gegeben, was ein *ctf* ist und warum eine erfolgreiche Durchführung eine Aussage über die Funktionsfähigkeit des zu testenden Systems erlaubt. Anschließend wird beschrieben, welche von der Realität abweichenden Rahmenbedingungen bei einem solchen Event gelten und welche Anpassungen deswegen vorgenommen wurden. Abschließend wird gezeigt, welche Beobachtungen während der Veranstaltung gemacht wurden und warum diese beweisen, dass das entwickelte System wie geplant funktioniert.

7.1 *Capture the flag* im IT-Sicherheits-Bereich

Ursprünglich wird mit dem Begriff „*capture the flag*“ ein Spiel bezeichnet, bei dem die Spieler zweier Teams versuchen, die Flagge des jeweils anderen Teams zu erobern und in ihre eigene Spielfeldhälfte zu bringen, ohne vom gegnerischen Team gefangen zu werden. [Merriam-Webster.com Dictionary, 2020] Im Fachbereich der *IT-Sicherheit* ist ein Wettbewerb gemeint, bei dem die Teilnehmer Zugang zu speziell präparierten Computer-Systemen erhalten, meist in Form *virtueller Maschinen*. Das Ziel ist in ein System einzudringen und dort versteckte „Schätze“ („*flags*“) zu entdecken. [Aigner, 2018] Als *Flaggen* werden im Kontext von *capture the flag* Geheimcodes oder andere Informationen bezeichnet, die an einem geschützten Ort auf einem solchen System hinterlegt sind. Erlangt ein Teilnehmer Kenntnis von einer eigentlich unzugänglichen Information, kann er nachweisen, dass er auf diesen vermeintlich geschützten Bereich zugreifen konnte. In der Realität entspricht dies einer Verletzung der IT-Sicherheits-*Schutzziele*.

Mit solchen Events, die sowohl als Präsenzveranstaltung als auch komplett online veranstaltet werden können, werden als Zielgruppe Teilnehmer verschiedener Fähigkeitslevel angesprochen: Sowohl Einsteiger als auch IT-Sicherheits-Profis sind auf solchen Veranstaltungen anzutreffen. Oft werden an die Gewinner solcher Wettbewerbe verhältnismäßig hohe Preisgelder vergeben: Beispielsweise hat die IT-Sicherheitsfirma *Trend Micro* im Jahr 2018 auf die ersten drei Plätze eines Wettbewerbs insgesamt 11 500 Euro verteilt. Als Motivation für dieses Investment wird unter anderem „gravierender Fachkräftemangel" im Bereich der IT-Sicherheit genannt. [Computerwelt.at, 2018]

Die wachsende Anzahl an Teilnehmern weist auf eine steigende Beliebtheit dieser Veranstaltungen hin. Eine Auswertung der Teilnehmerzahlen, die für bekannte Events auf *CTFtime.org [2020]* protokolliert werden, zeigt beispielsweise für eines der ersten erfassten *capture-the-flag*-Events, das *def con qualifier*, eine steigende Nachfrage: In den Jahren 2011 bis 2017 lagen die Teilnehmerzahlen dieser online stattfindenden Veranstaltung jeweils knapp unter 500 Teilnehmerteams, im Jahr 2019 waren es schon mehr als 1 300 und die Zahl stieg im Mai 2020 auf mehr als 1 400 Teilnehmerteams an.

7.1.1 Motivationen für *capture-the-flag*-Events

Motivation für Teilnehmer

Teilnehmer eines *ctf*-Wettbewerbs können sich beispielsweise durch die Herausforderungen, die ein solches Event mit sich bringt, motiviert fühlen, Neues zu lernen. Auf solchen Veranstaltungen lernen sie in einem spielerischen, aber dennoch kompetitiven Umfeld sowohl „richtiges hacken" als auch wichtige Programmier- und Problemlösungsfähigkeiten, nicht nur im Bereich der IT-Sicherheit. Die gemeinsame Aktivität mit anderen „Nerds" macht Spaß, geteilte Erfolgsmomente schweißen zusammen, was auch dem *team building* dient. Einige Teilnehmer demonstrieren gerne ihre eigenen Fähigkeiten und auch das teilweise ausgelobte Preisgeld oder die Chance auf einen Einstieg in ein interessantes Berufsfeld dürften einige von ihnen zur Teilnahme animieren.

Motivation für Veranstalter

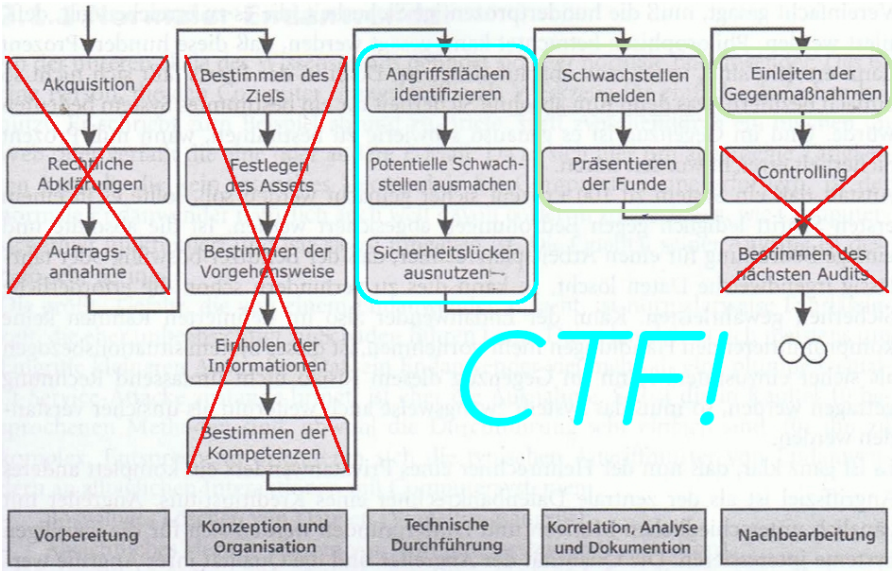
Capture-the-flag-Veranstaltungen werden teilweise als Teil von Universitätskursen angeboten, um theoretisches und praktisches Wissen auf spielerische Art zu vermitteln. [Harsa Wara Prabawa, 2017]

Für kommerzielle Anbieter könnte es beispielsweise attraktiv sein, mit der Ausrichtung solcher Events als Werbeveranstaltung den eigenen Namen in der IT-Sicherheits-Community bekannt zu machen oder eventuell eigene Produkte und Dienstleistungen dort zu platzieren. Als Motivation wird von Veranstaltern auch angegeben, dem Fachkräftemangel entgegentreten zu wollen, indem man „Young Professionals“ dazu motiviere, sich „diese wichtigen Fähigkeiten anzueignen“ [Computerwelt.at, 2018].

Zudem ist es denkbar, IT-Produkte mit Hilfe von *ctf*-Events bei vergleichsweise geringem Investment auf Schwachstellen testen zu lassen, wobei interessante Ergebnisse zutage gefördert werden könnten.

7.1.2 Ähnlichkeit zu Penetrationstests

Obwohl der Realitätsgrad der in solchen Wettbewerben gestellten Aufgaben sehr schwankt und die Herausforderungen gelegentlich eher an das Lösen von Puzzles erinnern, reicht die Bandbreite der zu bearbeitenden Aufgabenstellungen nah an die Probleme heran, die bei einer professionellen Sicherheitsüberprüfung zu lösen sind. Für die erfolgreiche Teilnahme an diesen Wettbewerben werden die gleichen oder zumindest ähnliche Fähigkeiten trainiert, die auch für die Arbeit als *Penetrationstester* benötigt werden.



Quelle: Ruef [2007]; Eigene farbliche Hervorhebungen

Abbildung 7.1: Schnittmenge zwischen *ctf* und den Tätigkeiten des *Penetrationstesters*

Zwar entfallen beim *ctf* viele der notwendigen Aspekte eines professionellen Penetrationstests, wie beispielsweise der direkte Kundenkontakt und ein Großteil anderer administrativer Tätigkeiten, außerdem herrscht hier Wettbewerbs- und Zeitdruck; im Kern sind jedoch viele Tätigkeiten gleich (siehe Abbildung 7.1). Primär konzentrieren sich die Teilnehmer darauf, Angriffsflächen zu identifizieren, potentielle Schwachstellen ausfindig zu machen und diese Sicherheitslücken dann auszunutzen. Bei einigen Veranstaltungen gibt es außerdem Extrapunkte fürs Melden von Schwachstellen oder für eine Präsentation der gefundenen Angriffsvektoren und durchgeführten Angriffe (engl. "write ups"). Es gibt auch Spielmodi, bei denen eigene Systeme gegen andere Mitspieler verteidigt werden. Hier müssen dann zur Sicherheitslücke passende Gegenmaßnahmen eingeleitet werden.

Insgesamt lässt sich sagen, dass ein *ctf*-Event auch so aufgebaut werden kann, wie ein „Penetrationstest light".

7.2 Vorbereitungen für das *capture-the-flag*-Event

Um das im Rahmen dieser Arbeit aufgebaute Honeypotsystem – und vor allem die korrekte Protokollierung von Ereignissen – zu testen, wird ein *capture-the-flag*-Event veranstaltet, zu dem mehrere Studenten verschiedener Hochschulen eingeladen werden sowie IT-Fachkräfte aus verschiedenen Bereichen.

7.2.1 Persönliche Motivation

Mithilfe eines *ctf*-Wettbewerbs können mehrere IT-Sicherheitsfachkräfte und/oder an dem Fachgebiet interessierte Personen dazu gebracht werden, das im Zuge dieser Arbeit aufgebaute System zu testen. Von den Teilnehmern wird erwartet, dass sie bei dem Versuch, die gestellten Aufgaben zu lösen, auch nicht vorhergesehene Wege einschlagen. Eventuell können sie sogar auf Probleme aufmerksam machen, die zuvor gar nicht erkannt wurden.

Im Wesentlichen aber sollen die Hauptfunktionalität der konstruierten Überwachungssoftware – das Beobachten von serverseitigen Interaktionen bösartiger Subjekt – durch das Anlocken verschiedener Angreifer getestet werden.

7.2.2 Technische Infrastruktur

Grundsätzlich bleibt das System, das mit dem *capture-the-flag*-Event getestet werden soll, so bestehen, wie im Kapitel „Aufbau des *Kukulkan high interaction honeypot*“ (6) entwickelt. Um den Wettbewerb spannender zu gestalten und den Teilnehmern mehr zum Entdecken zu bieten, wurden die in der anzugreifende Umgebung bereits platzierten Sicherheitslücken um zusätzliche Aufgaben erweitert, die die Motivation der Teilnehmer während des *ctf*-Events aufrecht halten sollen.

Abweichende Rahmenbedingungen für das *ctf*-Event

Abweichend von den Ansprüchen, denen das zuvor aufgebaute *Honeypot*-System genügen muss, herrschen für das *ctf*-Event teilweise andere Umgebungsbedingungen und Anforderungen. Die Teilnehmer des Events sollen motiviert bleiben: Eine durch einen Serverabsturz verursachte Turnierunterbrechung würde für Unstimmigkeiten sorgen und das Risiko bergen, die Veranstaltung vorzeitig beenden zu müssen.

Doch auch das Verhalten einzelner Teilnehmer selbst kann Auswirkungen auf die anderen Mitspieler haben: *Brute-Force*-Angriffe könnten das Gesamtsystem bis zur Nichterreichbarkeit verlangsamen und einige Angriffstechniken benötigen eine erhöhte Bandbreite. Außerdem ist es Teil der Aufgabenstellung, Administrationszugang auf dem Server zu erhalten. Ab dem Zeitpunkt, an dem dies einem Team gelingt, ist für die anderen Teilnehmer nicht mehr zu garantieren, dass der Server sich noch so verhält wie erwartet.[1]

Sollte eine unvorhergesehene Interaktion eine der Funktionalitäten des Gesamtsystems beeinträchtigen, würde das auch den Erfolg des Versuchs insgesamt aufs Spiel setzen und die Beobachtung erschweren. Demnach ist also dafür zu sorgen, dass jegliche eventuell auftretenden Probleme lokal eingedämmt bleiben.

Abweichende Konfigurationen aufgrund geänderter Rahmenbedingungen

Aus den genannten Gründen erhält jedes Team eine eigene Server-Instanz. Die Zugangsdaten dieser Instanz sollten den anderen Mannschaften nicht mitgeteilt werden. Damit

[1] Auf einigen *ctf*-Events ist genau diese Interaktion verschiedener Teilnehmer auf einer gemeinsamen Maschine Teil des Konzeptes. Wer es beim *King of the hill*-Modus schafft, die Konkurrenten aus einer gekaperten Maschine auszuschließen, aber selbst Zugriff zu behalten, erhält Punkte.

die Teilnehmer trotzdem nicht nur eine kryptische, nichtssagende *id* erhalten, wurden die Instanzen mit sprechenden Bezeichnern[2] versehen. Die Zugangsdaten bestehen aus einer *Domain*[3] und dem Instanznamen als *Subdomain*.

Im Abschnitt „Angebotsvergleich Hoster“ (6.5.1) wurden mehrere Server-Hosting-Anbieter miteinander verglichen. Da für das *ctf*-Event aber eigene Bedingungen herrschen, ist auch dieser Vergleich neu zu bewerten: Es werden nun mehrere Server benötigt, wobei die genaue Anzahl bis zu Beginn der Veranstaltung noch unklar ist. Außerdem werden die Server für einen eher kürzeren Zeitraum benötigt, teilweise nur ein paar Stunden, längstens jedoch für eine Woche. Die *Cloudserver*-Angebote von Google haben zwar insgesamt einen höheren Monatspreis, ihre Nutzung wird aber sekundengenau abgerechnet, weshalb sie hier die bessere Wahl darstellen.[4]

Wie im Abschnitt „Wahl der zu verwendenden Dienste“ (3.2) beschrieben wurde, gehört zu einem aktuellen, realitätsnahen Aufbau eine Verbindungsverschlüsselung mit *SSL*. Abweichend vom ursprünglichem Vorhaben würde bei einem *SSL-Zertifikat* pro verwendeter Domain das Problem entstehen, dass wegen der nachvollziehbaren und öffentlichen Protokollierung der Erstellung[5] einzelner Zertifikate in sogenannten *Transparenzlogs*[6] die Subdomains und somit die Instanzzugangsinformationen ausspähbar wären. Als Lösung hierfür wurde für die verwendete Domain ein *Wildcard*-Zertifikat verwendet, das Verschlüsselung für die Domain und alle Subdomains erlaubt.[7]

Um dem Wettbewerbscharakter Rechnung zu tragen, werden in geschützten Bereichen des Betriebssystems Geheimcodes hinterlegt. Erlangt ein Teilnehmer Kenntnis von diesen

[2] Bestehend aus einer Kombination aus einem maximal zweisilbigen Adjektiv und einem maximal zweisilbigen Nomen, beispielsweise „fastHacking“, „bigChicken“, „falseMusic“, „sickNews“ oder „usedData“.

[3] Die *Domain kukulkanctf.de* wurde nur fürs Event genutzt, inzwischen ist sie ohne Funktion.

[4] Insgesamt sind für das Event *Hosting*-Kosten in Höhe von ca. 60 Euro entstanden.

[5] Google hatte zu einem bestimmten Zeitpunkt Probleme damit, dass durch „vertrauenswürdige“ *certificate authoritys* (*CA*) Zertifikate unter anderem auf *google.com* ausgestellt wurden. [Google, 2013] Daraufhin unterbreitete Google erst ein Vorschlag zur Lösung dieses Problems. Dieser wurde anschließend mit der Marktmacht des Google-*Chrome*-Browsers und des *Android*-Betriebssystems durchgesetzt. [Sleevi, 2015]

[6] Das Verfahren wird in RFC 6962 zur Diskussion gestellt.

[7] Dieses *wildcard*-Zertifikat und der dazugehörige *Private-Key* lagen für die Verschlüsselung des Datentransfers auf den Servern innerhalb des verwundbaren Bereichs. Das bringt das Problem mit sich, dass diese Daten als kompromittiert zu betrachten sind. Unbefugte Personen sind nun in der Lage ihren eigenen Server gegenüber Dritten, deren DNS-Anfragen umgeleitet oder manipuliert werden, als Server der verwendeten Domain auszugeben. Es gibt Techniken, mit denen dieses Problem gelöst werden kann. Da für das *ctf*-Event eine eigene nur für diesen Zweck verwendete Domain benutzt wurde, kann dieses Risiko ausnahmsweise „akzeptiert“ werden. Das Zertifikat hat eine Gültigkeit bis zum 31. August 2020, 20:47:02 GMT. Danach besteht dieses Problem nicht mehr.

Informationen, kann er damit dem Veranstalter gegenüber nachweisen, dass er sich (mindestens) lesenden Zugriff auf einen vermeintlich geschützten Bereich verschaffen konnte. Im spielerischen Kontext verteilt der Veranstalter hierfür Punkte an den Teilnehmer.

Für das Event wurden an drei verschiedenen Orten Flaggen mit unterschiedlichen Zugriffsbeschränkungen abgelegt: Die erste ist im Wurzeldateisystem des Containers zu finden. Die Webserversoftware selbst erlaubt keinen Zugriff auf diesen Bereich. Die Datei ist für jeden auf dem Server angemeldeten Benutzeraccount lesbar. Die Kenntnis des gefundenen Geheimcodes in der Flagge beweist also den Zugriff auf das Dateisystem außerhalb des öffentlich lesbaren Bereichs. Die zweite Flagge befindet sich innerhalb des Containers im Benutzerverzeichnis des Administrator-Kontos; sie ist somit nur für den Systemadministrator zu lesen. Die Kenntnis des hinterlegten Geheimnisses belegt das Eindringen in diesen privilegierten Account. Die dritte Flagge liegt außerhalb des Containers und ist nur für Benutzer des Host-Betriebssystems lesbar. Wer diese Datei lesen kann, konnte erfolgreich aus dem Containment ausbrechen.

Aus dem Grundsystem übernommene Einfallstore

Aus dem in Kapitel „Einfallstore in der simulierten Firmeninfrastruktur“ (3.3) beschriebenen Versuchsaufbau wurden die beiden absichtlich hinterlegten Fehler übernommen. Diese werden hier der Vollständigkeit halber zusammengefasst:

Viele Internetseiten bieten ihren Anwendern die Möglichkeit, Dateien hochzuladen, beispielsweise um ein Profilbild zu hinterlegen oder um Dateien zwischen Kunde und Dienstleister auszutauschen. Wird dieses Uploadformular nicht mit der gebotenen Vorsicht implementiert und der hochgeladene Inhalt nicht genügend geprüft, kann ein Angreifer ausführbaren Code hinterlegen. Wenn es einem Angreifer gelingt, den Server dazu zu bringen, diesen platzierten Code auszuführen, wird von „*remote code execution*“ gesprochen.

Webserver werden meist mit Fernwartungssystemen, wie beispielsweise *ssh*, administriert. Im Gegensatz zur Realität, in der keine *Fernwartungszugriffssoftware* für die *Authentisierung* eine Kombination aus Benutzernamen und Passwort akzeptieren sollte, wurde dieses unsichere Verfahren für das *ctf*-Event zugelassen und gewährte Systemverwaltungszugriff auf der Serveradministrationskonsole.

Zusätzlich platzierte Einfallstore für das *ctf*-Event

Die Präparierungen, also die bewusst hinterlegten Einfallstore für Angreifer, entsprechen grundsätzlich denen des *Honeypots*, die im Abschnitt „Einfallstore in der simulierten Firmeninfrastruktur“ (3.3) beschrieben wurden. Für das *capture-the-flag*-Event wurden noch weitere *Unsicherheiten* eingebaut. Für die Teilnehmer sollte somit der „Abstand“ zwischen mehreren auszunutzenden Sicherheitslücken verringert werden. Insgesamt ergab sich somit aus der Kombination der Angriffsmöglichkeiten für die Teilnehmer eine Art vorgegebener „Pfad“, dem sie folgen konnten. Das war mit der Absicht verbunden, Motivation und Interesse der Teilnehmern aufrecht zu halten. Die von den Teilnehmern verwendeten Vorgehensweisen konnten so auch miteinander verglichen werden.

Auf der Website wurde zusätzlich zu den in Kapitel „Einfallstore in der Firmenwebsite“ (3.3.1) beschriebenen Bereichen ein Administrationsbereich eingerichtet, der mit einem Benutzernamen und Passwort gesichert ist. In der Realität gibt es oft solche Administrationsbereiche, die beispielsweise erlauben, einem *Content-Management*-System wie *Wordpress* Inhalte hinzuzufügen.[8]

Das Login-Formular für den Administrationsbereich, das eine *Authentisierung* mit Benutzernamen und Passwort verlangt, ist als deutlicher Hinweis darauf zu verstehen, dass die Website im Hintergrund eine Datenbank verwendet. In dieser Datenbank ist der erfundene Name des ausgedachten IT-Administrators der fiktiven Firma hinterlegt. Das dazugehörige Passwort ist eine Kombination aus mehreren Wörtern, inklusive des Firmennamens. Ein *Brute-Force-Angriff* oder eine *Wörterbuch-Attacke* sollte daher nicht zum Erfolg führen.[9] Anstelle des Passworts selbst, ist in dieser Datenbank ein *MD5-Hash*, also eine *Einweg-Abbildung* des Passworts hinterlegt. Die Wahrscheinlichkeit, dass diese Kombination in einer *rainbow table*, also einer vorab berechneten Datenbank vieler Zeichenketten und ihrer *Hashwerte*, enthalten ist, ist gering, obwohl kein *salt string*, also eine Erweiterung des Eingabewerts für die *Hashfunktion*, verwendet wurde.

Der Code von Software, unter anderem auch von Webseiten, wird oft mit Hilfe einer *Versionsverwaltungssoftware* in sogenannten *Repositories* versioniert und verwaltet, um

[8] Werden für diese Administrationsbereiche schwache oder Standardpasswörter verwendet, ermöglicht dies auch unautorisierten Personen Inhalte hinzuzufügen. Auf diese Weise können beispielsweise auf eigentlich seriösen Seiten mit einem positivem *Page-Rank* Links auf unseriöse Angebote hinterlegt werden, um ihren Rang bei Suchmaschinen zu erhöhen.

[9] Selbst wenn es einem Teilnehmer gelingen sollte, das Passwort anders als gedacht zu knacken, wäre das kein Problem. In diesem Experiment geht es hauptsächlich darum, das Verhalten eines Angreifers zu beobachten. Dieses Ziel wäre demnach nicht gefährdet.

Entwicklern bessere Werkzeuge für die gemeinsame Programmierung an die Hand zu geben. Im Gegensatz zum Aufbau des in Abschnitt „Manuelle Ausführung für den Gast“ (6.6.4) beschriebenen Vorgehens wird für das *capture-the-flag*-Event das Repository nicht gelöscht. Dieses „Versehen“ unterläuft Administratoren auch in der realen Welt und kann zu einer unbeabsichtigten Veröffentlichung von *Sourcecode*, Zugangsdaten für *Datenbanken* oder anderen Geheimnissen führen. [Internetwache.org, 2015] [Strozyk, 2020, 1. Juli] Durch diesen für das *ctf*-Event absichtlich eingebauten Fehler kann ein Teilnehmer eine alte Version der Website finden, in der das Passwort des Website-Administrators noch nicht als *Hash* abgelegt war, sondern als Klartext.

Das auf diese Weise extrahierte Administratorpasswort der Website wurde von dem fiktiven Systembetreuer auch als Anmeldepasswort für den Server-Administrations-Zugang genutzt. Ein Angreifer kann also wegen des als *password reuse* bezeichneten Problems seine Zugriffsrechte noch einmal deutlich erhöhen (*privilege escalation*). Diese Phänomen fällt in die Kategorie der menschlichen Fehler (*human factors*). Angriffe gegen diese Fehlerklasse werden unter dem Oberbegriff *social engineering* zusammengefasst.

Für den Ausbruch aus dem *Container* gibt es für dieses *capture-the-flag*-Event keinen präparierten Weg.

Angedachter Lösungspfad

Ein Teilnehmer, der den vom Veranstalter des *capture-the-flag*-Events vorgesehenen Lösungsweg nutzt, verwendet das unsichere Dateiuploadformular für *remote code execution* und entdeckt so die *Datenbank*, in der der *Hashwert* des Passworts zu finden ist. Mithilfe des Zugriffs auf das Versionskontrollsystem extrahiert er eine alte Version der *Datenbank*, in der noch kein *Hashing* eingesetzt wurde, sondern das Passwort im *Klartext* vorlag. Von diesem Websiteadministrationspasswort kann angenommen werden, dass es für den Administrationsbereich des Webservers wiederverwendet wird.

Andere Kombinationen der vorgesehenen Sicherheitslücken sind durchaus denkbar und es ist auch möglich, dass noch andere Lücken existieren. Sollte ein Teilnehmer einen anderen Weg finden, wäre dies für beide Seiten als Erfolg zu verbuchen, solange dies mit dem Überwachungswerkzeug beobachtet werden kann.

7.2.3 Wettbewerbssituation

Zu einem Wettbewerb gehören Einladungen, Teilnehmer, Teams, Gewinne und Gewinner sowie eine Portion Nervenkitzel:

Motivation am Kukulkan-Event teilzunehmen

Um Teilnehmer zu motivieren, an dem vorbereiteten Wettbewerb teilzunehmen, wurde ein Preisgeld im Sinne eines sogenannten „*Pwn2Own*"-Wettbewerbs ausgeschrieben. Diese Idee ist angelehnt an das „*Pwn2Own*"-Turnier, das seit 2007 mindestens jährlich auf der *CanSecWest* IT-Security-Konferenz in Vancouver, Kanada, durchgeführt wird.[10] Für den Wettbewerb auf der Konferenz werden verschiedene Geräte, – ursprünglich waren es zwei *MacBook Pro* – an ein öffentliches *WLAN* angeschlossen. Wer es als erstes schafft, unautorisierten Code darauf auszuführen, darf das entsprechende Gerät mit nach Hause nehmen. [Goodin, 2007]

Im Fall des Events dieser Arbeit sollte ein Webserver angegriffen werden, dementsprechend soll der Gewinn ein Webserver sein. Insgesamt wurden drei Preise ausgelost, jeweils für das erste Erreichen[11] der drei Kriterien Benutzerzugriff, Systemverwaltungszugriff und Containerausbruch.

Einladung und Ankündigung

Die Einladung zum *capture-the-flag*-Event wurde in verschiedenen *Mailinglisten*, *instant-messenger*-Gruppen und *Chaträumen* geteilt und hat geschätzte 300 bis 350 Personen aus dem Umfeld der *HAW Hamburg*, der *Universität Hamburg* der *LMU München* und dem *Chaos Computer Club* erreicht.

Teambildung

Für die Teilnahme an dem *ctf*-Event wurde Teambildung empfohlen, unter anderem weil das gemeinsames Knobeln mit einem Partner, mit dem man sich gut ergänzt, am meisten

[10] Ins Leben gerufen wurde dieser Wettbewerb von Dragos Ruiu, weil er unzufrieden war, wie *Apple* damals mit der Veröffentlichung von Sicherheitslücken umging. Ruiu warf Apple vor, seine Probleme „unter den Tisch zu kehren"[Ruiu, 2007].

[11] In der *ctf*-Szene mit dem aus der *e-Sport*-Szene entliehenem Begriff „*first blood*" bezeichnet.

Spaß macht. Ein handfester Grund ist zudem, dass nicht so viele einzeln abgerechnete Instanzen gestartet werden müssen, wenn sich mehrere Teilnehmer zu Teams zusammenfinden.

Insgesamt haben schließlich elf Personen an dem Event teilgenommen, die sich auf zwei Teams mit je zwei Personen und sieben Einzelpersonen aufgeteilt haben.[12] Von diesen neun Teams haben sich sechs Teams Zugriff auf die Server erarbeitet. Von diesen sechs Teams erweiterten fünf ihre Zugriffsrechte auf dem Server zum Systemadministrator.

7.3 Verhaltensbeobachtung während des Events

Während des Events wurde die Interaktion der Teilnehmer mit dem entwickelten Überwachungssystem beobachtet und protokolliert. Alle manuellen Eingaben auf den Serverkonsolen (inklusive der Tippfehler) konnten beobachtet werden. Von den Teilnehmern verwendete Hilfserkzeuge und die von diesen abgesetzten Befehle konnten erfolgreich mitgeschnitten werden. Teilweise wurden die „Bildschirmausgaben“ der Server (*stdout*) beobachtet, wie beispielsweise die Diagnoseergebnisse verwendeter Tools.

Für die allgemeine Kommunikation zwischen Veranstalter und Teilnehmern wurden mehrere *IRC-Channel* eingerichtet. Ein Kanal wurde für Fragen und Absprachen genutzt, die alle Teams betreffen. Zusätzlich wurde jeweils ein Kanal für jedes Teilnehmerteam zur Verfügung gestellt. Dadurch, dass die Zugangsdaten zu der jeweiligen Team-Instanz in dem entsprechenden Teamkanal mitgeteilt wurden, war sichergestellt, dass mindestens ein Teammitglied erreichbar ist.

Die Interaktionen der Teilnehmer mit ihren jeweiligen Servern wurde – wie im Abschnitt „Veranschaulichung Systemaufrufprotokollierung“ (5.3) beschrieben – überwacht. Jeder Aufruf eines *syscalls* wurde inklusive übergebener Argumente als einzelne Zeile in einer Datei protokolliert. Da einzelne Prozesse bei der Verwendung von Unterprogrammen Unterprozesse erzeugen, wurde ebenfalls die Tiefe der Prozesshierarchie festgehalten und die Prozessidentifikationsnummer des Elternprozesses. Hierdurch konnten Verkettungen zusammengehöriger Aktionen erkannt werden. Wurden mehrere Aktionen parallel ausgeführt, ließen sich diese mit der Prozessidentifikationsnummer einer Sequenz zuordnen.

[12] An dieser Stelle vielen Dank an die CTF-Teams Cyclopropenylidene (c3h2), CInsects, FOXACID und alle Einzelspieler!

Durch die angesprochenen Kommunikationskanäle zwischen Veranstalter und den Teilnehmerteams konnten diese Beobachtungen verifiziert werden. Im Dialog mit den einzelnen Teilnehmern wurde sichergestellt, dass alle Interaktionen korrekt protokolliert wurden. Kein Teilnehmerteam hat mit dem Server interagiert, ohne dass das Überwachungsmodul dies protokolliert hätte.

7.3.1 Verhaltensbeobachtung beim Eindringen in den Server

Eine Protokollierung der Ausnutzung der zum Einbruch führenden Sicherheitslücke ist nicht Gegenstand der Überwachung und in den Protokolldateien auch nicht ersichtlich, soll der Vollständigkeit halber jedoch erwähnt werden:

Exploit-Phase

Gegen die in Abschnitt „Einfallstore in der Firmenwebsite“ (3.3.1) beschriebene *remote-code-execution*-Lücke wurde über das Dateiuploadformular verschiedener Code hochgeladen. Der in Listing 7.1 beispielhaft gezeigte *exploit* führt auf dem Server das mit `'c'` benannte `$_GET`-Argument aus und ist als *proof of concept* zu betrachten. Hiermit lassen sich Befehle auf dem Server ausführen, mit denen Informationen gesammelt werden oder gegebenenfalls hochgeladene *Binärdateien* ausgeführt werden. Wird als `$_GET`-Argument beispielsweise `cat /etc/passwd` angegeben, wird auf dem Betriebssystem von php das Programm `cat` mit dem Argument `/etc/passwd` ausgeführt. Das Ergebnis des Programmaufrufs wird an php zurück gegeben. Php liefert es über den Webserver an den Browser, der es wiederum dem Anwender anzeigt.

```
1 <?php system($_GET['c']);
```

Listing 7.1: Minimaler *exploit* / *proof of concept* für *remote code execution*

Das Überwachungsmodul protokolliert die Ausführung des Programms inklusive seiner Prozessidentifikationsnummer, den aufrufenden Benutzeraccount, den Hostnamen der Maschine (hier `container1`) und die aktuelle Uhrzeit, wie in Listing 7.2 gezeigt.

```
1 23780 14:24:07 www-data@container1) cat /etc/passwd
```

Listing 7.2: Protokollierte *remote code execution*

Post-exploit-Phase

Wurde die vermutete Sicherheitslücke durch den Angreifer mit Hilfe des *proof of concept* verifiziert, kann sie ausgenutzt werden. Das Ausführen von Programmen ist auf diesem Weg mühsam, weshalb beispielsweise frei verfügbare php *reverse shells* einen komfortablen Zugriff auf das System erlauben. Sie funktionieren grundsätzlich wie gezeigt, bieten jedoch unterschiedliche Unterstützung hinsichtlich Komfort oder Automatisierung.

Für die weitere Informationsbeschaffung verwendeten einige Teilnehmer jeweils unterschiedliche spezielle Hilfswerkzeuge. Die Herkunft dieser Software ließ sich wie erhofft nachvollziehen. Im Kontext des *ctf*-Events wurden zwar frei verfügbare Werkzeuge verwendet, in einem realen Angriffsfall kann durch diese Protokollierung auch die Herkunft unbekannter Schadsoftware nachvollzogen werden.

Als Beispiel sei hier das von einem Teilnehmerteam nachgeladene Werkzeug *Linpeas* genannt, das darauf spezialisiert ist, eine Checkliste häufiger Fehlkonfigurationen abzuarbeiten, die zu *privilege escalation* führen können. [Polop, 2020] Es wurde mit Hilfe des Systemprogramms `wget` nachgeladen, wie in Listing 7.3 gezeigt. Die Herkunft des Hilfswerkzeugs wird durch die Protokollierung des Downloads offengelegt.[13]

```
1  8214 16:54:23 www-data@container1) wget
   ↪  https://raw.githubusercontent.com/carlospolop/┘
   ↪  privilege-escalation-awesome-scripts-suite/master/linPEAS/linpeas.sh
```

Listing 7.3: Protokolliertes Nachladen zeigt die Herkunft des Hilfswerkzeugs

Auch andere Werkzeuge wurden verwendet: Die *p0wny@shell* bietet im Gegensatz zu einigen anderen Konsolenwerkzeugen hohen Bedienkomfort. Zum Funktionsumfang gehören eine automatische Vervollständigung von Konsolenbefehlen mit der Tabulator-Taste und eine Navigation durch die zuletzt verwendeten Befehle, so wie bei einer modernen normalen *shell*. [@flogisoft, 2020] Die *weevely* „weaponized shell“, ist ein Werkzeug, das Fähigkeiten für die *exploit*- und *post-exploit*-Phase miteinander verbindet. Laut eigener Darstellung unterstützt dieses Tool bei „administrativen Aufgaben“, bietet „Lagebewusstsein“, unterstützt beim Ausweiten von Privilegien und verteilt sich im Zielnetzwerk. [epinna, 2020] Als einzige nicht *php-shell* wurde von einem Teilnehmer eine als *Client* für das Penetrationstestwerkzeug *Metasploit* dienende *Binärdatei* hochgeladen. Auch die mit dieser *reverse shell* getätigten Eingaben konnten erfolgreich mitgelesen werden.

[13] Es wäre genauso gut möglich gewesen, das Werkzeug über das unsichere Uploadformular hochzuladen.

7.3.2 Verhaltensbeobachtung bei der manuellen Interaktion

Das weitere Verhalten der Teilnehmer konnte unabhängig von der verwendeten *shell* protokolliert werden. Beispielsweise konnten Angreifer bei der Informationsbeschaffung beobachtet werden. Listing 7.4 zeigt den Anwender beim Auslesen der Protokolldatei, in der *Linpeas* die gefundenen Informationen sichert.

```
1  ...
2  25957 16:56:36 www-data@container1) ls -al
3  25959 16:56:45 www-data@container1) cat linpeas.out
4  25961 16:56:47 www-data@container1) less -R linpeas.out
5  ...
```

Listing 7.4: Verhaltensbeobachtung bei Auslesen der *Linpeas*-Ergebnisdatei

Listing 7.5 zeigt, dass der Angreifer an verschiedenen Orten des Systems nach Informationen zur Systemkonfiguration sucht. Zuerst wird in den *bash*-Konfigurationsdateien nach Hinweisen gestöbert. In Zeile 11 wird mit dem Befehl `sudo -i` probiert, ob der unprivilegierte Anwender `www-data` berechtigt ist, Systemadministrationsbefehle auszuführen. Die Zeilen 12 und 14 zielen darauf ab, Informationen zu geöffneten Netzwerkverbindungen anzuzeigen.

```
1  ...
2  26136 17:04:37 www-data@container1) cat /etc/bash.bashrc
3  26138 17:05:07 www-data@container1) ls -al
4  ...
5  26142 17:05:14 www-data@container1) cat /home/lucas/.bashrc
6  ...
7  26160 17:05:35 www-data@container1) cat .bashrc
8  ...
9  26183 17:06:33 www-data@container1) cat .profile
10 ...
11 26204 17:08:23 www-data@container1) sudo -i
12 26223 17:09:25 www-data@container1) nc -v
13 ...
14 26365 17:16:43 www-data@container1) ss -ln
15 ...
```

Listing 7.5: Verhaltensbeobachtung bei der weiteren Informationsbeschaffung

Listing 7.6 zeigt, wie der Angreifer den zur Website gehörenden Source-Code inspiziert. Zuerst werden die Dateien einzeln angesehen, später – sichtbar ab Zeile 9 – wird versucht, ein Archiv aller zur Website gehörenden Dateien sowie der des Versionskontrollsystems zu erstellen. Ein solches Archiv ist als einzelne Datei bequemer herunterladbar. Die Zeilen 9 und 11 zeigen fehlgeschlagene Versuche, weil in dem aktuellen Verzeichnis kein Schreibrecht besteht. Erst in Zeile 13 wird mit dem Befehl `tar cvzf /tmp/git.tgz .git` erfolgreich ein Archiv aller Meta-Dateien der Versionskontrollsoftware erstellt. In den folgenden Zeilen sind auch Unteraufrufe des tar-Programmes sichtbar. Die Hierarchie der Unteraufrufe ist durch die Einrückung sichtbar. Ebenfalls sichtbar ist in der ersten Spalte die Prozessidentifikationsnummer des aufrufenden Prozesses.

```
1  ...
2  26542 17:28:15 www-data@container1) cat mail.php
3  ...
4  26563 17:29:30 www-data@container1) cat database.db
5  ...
6  26677 17:39:25 www-data@container1) file database.db
7  ...
8  26753 17:41:06 www-data@container1) less index.php
9  26879 17:51:46 www-data@container1) tar cvzf html.tgz html
10 ...
11 26897 17:52:27 www-data@container1) tar cvzf git.tgz .git
12 ...
13 26903 17:52:58 www-data@container1) tar cvzf /tmp/git.tgz .git
14     26903 17:52:58 www-data@container1) /bin/sh -c gzip
15         26903 17:52:58 www-data@container1) gzip
16 ...
```

Listing 7.6: Verhaltensbeobachtung Informationsbeschaffung der Website

Nicht sichtbar ist die Auswertung der gesammelten Informationen auf den Maschinen der Angreifer. Erst aus den folgenden Interaktionen – dargestellt in Listing 7.7 – wird ersichtlich, dass die Angreifer erfolgreich das Passwort des Anwenders `lucas` aus den Informationen des git-Repositories extrahieren konnten (siehe Abschnitt „Zusätzlich platzierte Einfallstore für das *ctf*-Event“ (7.2.2)). Der Loginversuch ist protokolliert; anhand der in den folgenden Zeilen geänderten Anzeige des angemeldeten Anwenders von `www-data` zu `lucas` ist der Erfolg sichtbar. Ebenfalls zu erkennen ist das erfolgreiche Ausführen des `sudo -i` Befehls, das dem Angreifer nun Systemverwaltungsprivilegien gibt. Mit diesen erweiterten Nutzerrechten liest er nun die geheime Information aus, die in der Datei `/root/flag.txt` hinterlegt ist.

```
...
27051 18:03:59 www-data@container1) login lucas
...
27107 18:05:41 lucas@container1) sudo -i
    27107 18:05:41 root@container1) -bash
            ...
        ...
        27107 18:05:47 root@container1) cat /root/flag.txt
...
```

Listing 7.7: Verhaltensbeobachtung Login nach erfolgreicher Passwortextraktion

7.3.3 Erfolgloser Scan nach verdächtigen Prozessen

Alle Teilnehmer haben sich, wenn auch auf unterschiedliche Weise, nach laufenden Prozessen umgesehen. Die Teilnehmer, die sich nicht auf Tools verlassen haben, haben das mit dem *Linux*-Befehl `ps -aux` getan und die Ausgaben manuell analysiert. Auch die verwendeten Tools *weevely* und *Linpeas* haben nach „verdächtigen Prozessen“ gesucht. Da die Überwachung von außerhalb des *Containers* stattfindet, ist sie nicht über die Liste der laufenden Prozesse zu erkennen.[14]

7.4 Erfolge des Events

7.4.1 Persönliche Erfolge

Aus der Sicht der Teilnehmer, die eine Rückmeldung gegeben haben, war das *capture-the-flag*-Event ein voller Erfolg. Mehrere Teilnehmer haben berichtet, dass sie Spaß bei der Lösung der gestellten Aufgaben hatten.

Ein Team hat für einen längeren Zeitraum an der falschen Stelle nach der nächsten Lösung gesucht. Die Beobachtung der Interaktionen erlaubte es, ihnen zu einem geeignetem Zeitpunkt einen Hinweis anzubieten. Diesen hat das Team angenommen. Anschließend wurde beobachtet, dass das Team dadurch wieder auf die richtige Spur gekommen ist.

[14] Da die eingesetzte Überwachungssoftware *Sysdig* die Überwachung mithilfe eines *Kernel*-Moduls durchführt, das *syscalls* protokolliert, und die *Containerisierung* für den *Gast* den *Kernel* des *Host* teilt, gibt es andere Wege, diese Überwachung aus dem *Container* heraus zu detektieren. Die von den Teilnehmern verwendeten Tools waren dazu aber nicht in der Lage.

Das Team hat die Aufgabe anschließend erfolgreich gelöst.[15] Dies hat dafür gesorgt, dass diese Teilnehmer auch den Erfolgsmoment der entdecketen Flagge spüren konnte.

7.4.2 Technischer Erfolg

Aus technischer Sicht war die Veranstaltung ebenfalls erfolgreich. Die Interaktionen der Teilnehmer konnte wie geplant mitgelesen und protokolliert werden, was als Beweis angesehen werden kann, dass das System grundsätzlich wie angedacht funktioniert. Obwohl einzelne Teilnehmer unvorhergesehene Werkzeuge verwendeten und Vorgehensweisen wählten, konnten auch diese Interaktionen ausgewertet werden.

So wie jeder Programmtest nur die Anwesenheit von Fehlern zeigen kann und nie ihre Abwesenheit[16], so kann auch der durchgeführte Test nur zeigen, was protokolliert werden kann. Die erfolgreiche Durchführung dieses Tests kann nicht beweisen, dass es nicht möglich ist, das System zu hintergehen.

[15] Sicherlich wäre das Team auch ohne den Hinweis erfolgreich gewesen, aber sie haben sich nach eigenen Angaben „voll verannt“. Außerdem war es teilweise auch meine Schuld als Veranstalter, dass überhaupt Dateien auf dem Server lagen, die nichts mit der gestellten Aufgabe zu tun hatten. Durch die Beobachtung hatte ich also die Chance, meinen Fehler zu korrigieren.

[16] “Program testing can be used to show the presence of bugs, but never to show their absence!” – Dijkstra

8 Exkurs: Betrieb des Honeypots und Auswertung

Das Ziel der Arbeit war das Erstellen eines Systems, mit dem sich Interaktionen eines Angreifers nach einem erfolgreichem Einbruch protokollieren lassen. Dass das entwickelte System dies ermöglicht, wurde im vorhergehenden Kapitel mit Hilfe eines *capture-the-flag*-Events erfolgreich getestet und somit bewiesen.

Als Teil des Experiments wurde der Honeypot über den Zeitraum mehrerer Monate über eine eigene Domain im Internet verfügbar gemacht. Von Anfang an war das größte Risiko dieses Experimentes, dass es keinen nicht eingeweihten Angreifer geben könnte, der das System tatsächlich angreift. Nach mehreren Monaten hat sich herausgestellt, dass dieser Fall eingetreten ist.

Der Vollständigkeit halber sollen die Beobachtungen der Betriebsphase trotzdem zusammengefasst dargelegt werden:

8.1 Betrieb und Protokollierung

Sowohl Fernwartungszugang als auch Webserver – also die Köder des Honeypots – waren über den gesamten Zeitraum erreichbar.

Auf den Fernwartungszugang gab es erwartungsgemäß eine hohe Anzahl automatisierter Angriffsversuche. Hier ist solange nicht sinnvoll zwischen automatisierten und eventuellen manuellen Versuchen zu unterscheiden, bis ein Angreifer es geschafft hat, tatsächlich ins System einzubrechen. Auf eine Protokollierung erfolgloser Versuche wird daher verzichtet.

Der Webserver innerhalb des Containers protokolliert in der Standardeinstellung alle Zugriffe auf die Website. Ebenfalls festgehalten werden etwaige Fehler, beispielsweise im php-Code der Website.

Bei der im Rahmen der Arbeit entwickelten Prozessüberwachung gab es im ersten Monat noch vereinzelte Probleme hinsichtlich der Zuverlässigkeit der Aufzeichnung der Ereignisse. Daher wurde der Honeypot dahingehend erweitert, dass das Überwachungsmodul automatisch täglich neu gestartet wird. Ab dem zweiten Monat lief die Aufzeichnung durchgängig und zuverlässig.

8.2 Auswertung

Für die Auswertung stehen die Protokolldaten des Webservers sowie die des eigentlichen Honeypotsystems zur Verfügung.

8.2.1 Auswertung der Webserverprotokolle

Da diese Protokolldateien innerhalb des Containers gespeichert werden, könnten sie nach einem erfolgreichen Einbruch manipuliert werden und liefern daher im Zweifelsfall keine belastbaren Ergebnisse. Dieses Problem ließe sich mit verhältnismäßig wenig Aufwand lösen. Da der Fokus der Auswertung jedoch auf den vom Honeypot protokollierten Ereignissen liegt, geschieht dies hier nicht.

Trotzdem lieferte das Zugriffsprotokoll interessante Hinweise auf aktuelle Sicherheitslücken, die hier erwähnt werden sollen: Schwachstellenscanner, die automatisiert das Internet nach verwundbaren Systemen durchsuchen, können Hinweise auf aktuelle Bedrohungen liefern. Obwohl viele dieser Scanner von Organisationen betrieben werden, die nach bereits bekannten Schwachstellen suchen, können Honeypots auf diese Art auch eine bisher noch nicht allgemein bekannte Angriffsmöglichkeit detektieren.

Die Angriffsvektoren aller im Rahmen dieser Arbeit erkannten Angriffsversuche sind jedoch bereits bekannt. Stellvertretend für mehrere soll hier eine mehrfach protokollierte `POST`-Anfrage nach der Ressource `/boaform/admin/formLogin` genannt werden. Diese versucht vermutlich Router zu finden, deren Administrationsoberfläche für die erst im Juli 2020 veröffentlichte Sicherheitslücke CVE-2020-8958 anfällig ist. Mit speziellen

Anfragen können beliebige Konsolenbefehle auf verwundbaren Routern ausgeführt werden, was eine Verletzung aller IT-Sicherheits-Schutzziele darstellt. (CVSS3 Base Score 7,2) [National Vulnerability Database, 2020]

8.2.2 Auswertung der Honeypotprotokolle

Die Protokolldateien des Honeypots werden vom Host-Betriebssystem – also außerhalb des Containers – erfasst und gespeichert. Im Falle eines erfolgreichen Einbruchs ist ein Angreifer nicht in der Lage, diese Protokolle zu manipulieren.

Bei der Auswertung aufgetretene Probleme

Ein Problem bei der Auswertung der Prozessüberwachung ist, dass auf dem Server auch innerhalb des Containers mehrere Dienste standardmäßig installiert sind, die automatisch ausgeführt werden, um verschiedene Wartungsaufgaben zu erfüllen. Beispielsweise werden alte Protokolldateien komprimiert oder gelöscht („logrotate“), der Webserver wird regelmäßig von temporären Session-Daten befreit. Die Paketverwaltung ermittelt upgradbare Pakete, um die Anzahl dieser beim nächsten Login eines Administrators anzeigen zu können. Da es sich hierbei um *bash*-Skripte handelt, gibt es aus Sicht des Überwachungsmoduls keinen Unterschied zwischen einer manuellen Ausführung oder der Automatik.

Diese Dienste und Tätigkeiten verschmutzen das Protokoll. Die Unterscheidung zwischen gewünschtem Verhalten und einem eventuellen Angriff ist nicht direkt offensichtlich. Die Einträge, die durch die erwünschten Systemwartungstätigkeiten auftreten, müssen vom Honeypotbetreiber erlernt werden, um sie von eventuellen anderen Einträgen unterscheiden zu können. Die Honeypot-Protokolldatei enthält über den Zeitraum mehrerer Monate einen Großteil nicht relevanter Einträge.

Lösung der aufgetretenen Probleme

Mit Hilfe verschiedener Systemwerkzeuge kann eine duplikatfreie Auflistung aller protokollierten Befehle erreicht werden. Der Kommandozeilenbefehl `cat 2020*/* | awk '{$1=$2=""; print $0}' | sort | uniq | less` liest alle Zeilen aller Logdateien ein, entfernt die ersten beiden Spalten (Prozess-id, Datum), sortiert alle Zeilen

(notwendige Vorbedingung für uniq), entfernt alle Duplikate und zeigt die verbleibenden Zeilen an. Auf diese Weise werden ca. 27 000 Protokollzeilen auf ca. 600 individuelle Befehle reduziert. Für diese kann deutlich leichter ausgewertet werden, ob nennenswerte Ereignisse stattgefunden haben.

Auswertung

Im Auswertungszeitraum wurden keine ungewöhnlichen Ereignisse protokolliert.

9 Fazit

9.1 Erarbeitetes System

In dieser Bachelorarbeit wurde der Frage nachgegangen, wie manuelle Interaktionen eines Angreifers auf einem Server gefahrlos protokolliert werden können. Bei der Ausgestaltung der Zielsetzung wurde eine agile Vorgehensweise gewählt, die ermöglicht hat, die komplexe Aufgabenstellung erfolgreich zu bearbeiten.

Als Lösungsmöglichkeit wurde die Entwicklung eines Honeypots gewählt. Da es eine Vielzahl verschiedener Honeypot-Arten gibt, wurden potenzielle Alternativen diskutiert. Der *high interaction honeypot* wurde als passendes Konzept ausgewählt. Ein solches System kann mit unterschiedlichen Hilfsmitteln aufgebaut werden, weshalb zunächst die zur Zielerreichung benötigten Komponenten identifiziert wurden. Hierfür wiederum wurden die notwendigen Teilaspekte beleuchtet, wie der Aufbau eines glaubwürdigen Köders, einer sicheren Isolation eines Betriebssystems und einer detaillierten Ausführungsüberwachung. Für diese Themengebiete wurden technische Ansätze und Alternativen diskutiert, wobei letztlich jeweils eine Lösungsmöglichkeit ausgewählt und praktisch getestet wurde. Nach erfolgreichen Tests der einzelnen Komponenten wurden diese zu einem Gesamtsystem kombiniert. Zuerst geschah dies manuell, um die notwendigen Schritte aufeinander abzustimmen und gegebenenfalls noch Änderungen vornehmen zu können. Anschließend wurden die Schritte, die zum Erfolg geführt haben, zu einem Gesamtsystem kombiniert. Dieses lässt sich mit Hilfe einer Skriptsammlung automatisiert auf einem Server installieren.

Das Gesamtsystem wurde in einem hierfür veranstalteten Event mit etwa einem Dutzend Angreifern daraufhin getestet, ob es die Anforderungen erfüllt und ob die Interaktionen eines Angreifers erfolgreich protokolliert werden können.

Das System wurde in der Entwicklungsphase von Anfang an so gestaltet, dass die zu beobachtende Umgebung modular aufgebaut ist, damit andere Entwickler die Ergebnisse

nicht nur nachvollziehen, sondern das System auch leicht an eigene Bedürfnisse anpassen können. Damit das System einen Nutzen für die Allgemeinheit hat, wurde es unter einer *open-source*-Lizenz veröffentlicht[1]. Dies erlaubt anderen Entwicklern nicht nur die Ergebnisse, sondern auch den Aufbau und die Funktion des Systems nachzuvollziehen und es selbst unverändert – oder beliebig für eigene Ansprüche modifiziert – einzusetzen.

9.2 Zielerreichung

Für die identifizierten Teilprobleme des glaubwürdigen Köders, der Betriebssystemisolierung sowie der Überwachung wurden jeweils Lösungen gefunden, die die Anforderungen erfüllen. Die einzelnen Lösungsansätze der Teilprobleme wurden erfolgreich miteinander zu einem voll funktionsfähigen *high interaction honeypot* kombiniert, wodurch eine praktisch umsetzbare Möglichkeit entwickelt wurde, wie ein System aufgebaut sein kann, mit dem manuelle Interaktionen eines Angreifers auf einem Server gefahrlos protokolliert werden können.

Bei dem durchgeführten Test mit etwa einem Dutzend Angreifern konnten die Interaktionen zuverlässig protokolliert werden. Aus den Protokollen wurden verschiedene Informationen über die Vorgehensweise der Angreifer identifiziert, wie diese ihre Privilegien erweitern und welche Hilfsmittel sie hierfür verwenden. Die Wege der Informationsbeschaffung des Angreifers konnten nachvollzogen werden, außerdem konnten verwendete Angriffswerkzeuge sowie die Herkunft dieser aus den Protokolldateien abgelesen werden.

Die Forschungsfrage, wie ein System aufgebaut sein kann, mit dem sich manuelle Interaktionen eines Angreifers gefahrlos protokollieren lassen, wird also beantwortet:

Ein solches System kann ein *high interaction honeypot* sein. Dieser sollte als Köder beispielsweise einen realistisch wirkenden Firmenauftritt im Internet darstellen. Er kann bewusst Einfallstore bereitstellt, mit dem ein Angreifer eindringen kann. Wird die Systemumgebung, in der die notwendigen Dienste betrieben werden, ausreichend isoliert, kann ein Angreifer sich hierin bewegen, ohne eine wirkliche Gefahr für Andere darzustellen. Hierfür kann eine Containerisierung verwendet werden, die auf technischer Grundlage der Namensräume eine wirksame Isolation sicherstellt, jedoch genügend Interaktionsmöglichkeiten und einen ausreichenden Realitätsgrad bietet. Das Vorgehen des Angreifers, also

[1] gitlab.com/kukulkanhoneypot

seine Interaktionen, können durch den Einsatz eines entsprechend angepassten Kernel-Moduls zuverlässig und umfangreich beobachtet werden. Werden diese Komponenten miteinander kombiniert, kann eine zuverlässige Überwachung der Interaktionen stattfinden, ohne dass der Angreifer dies bemerkt oder in der Lage ist, dies herauszufinden.

9.3 Ausblick

Der Ansatz der Containerisierung liefert bei der notwendigen Isolierung einen für die Zielsetzung ausreichenden Realitätsgrad, könnte aber auf Basis der ihr zugrundeliegenden Isolationstechniken der Namensräume besser auf die Bedürfnisse eines Honeypotsystems zugeschnitten werden. Abgesehen von der Neuentwicklung eines speziellen Containersystems für Honeypots unter Verwendung der entsprechenden Namensräume könnte die vorhandene Containerisierungslösung *lxc* um weitere Maßnahmen und Aspekte ergänzt werden, die die Existenz des Containers verschleiern, um eine noch realistischere Umgebung für einen *high interaction honeypot* darzustellen.

Um Schäden am eigenen und an fremden Systemen zu minimieren und um einem erfolgreichen Einbrecher kein weiteres Werkzeug an die Hand zu geben, ihm aber trotzdem eine realistische Umgebung anzubieten, wurde die Leistungsfähigkeit des verwundbaren Bereichs auf feste Werte limitiert. Basierend auf einer automatischen Auswertung könnten Limits dynamischer gestaltet werden, um dem Angreifer zu Beginn eine realistische Umgebung zu bieten, den potenziellen Schaden jedoch im weiteren Verlauf noch weiter zu begrenzen.

Für die Überwachung wurde eine vorhandene Prozessüberwachungslösung angepasst. Zur Erreichung des Ziels dieser Arbeit werden genügend Informationen protokolliert. Aber die hierfür eingesetzte Software bietet eine Vielzahl weiterer Möglichkeiten, die noch weiter ausgeschöpft werden könnten, um noch mehr Interaktions-Aspekte aufzuzeichnen. Als weiterer Ansatz könnte auch das Überwachungsmodul auf Grundlage der verwendeten Technologie eines Kernel-Moduls neu entwickelt werden, damit ein zukünftiges eigenständiges *high-interaction-honeypot*-System weniger von Fremdsoftware abhängig ist. Abgesehen von den Möglichkeiten, die ein Kernel-Modul bietet, gibt es viele weitere Tracing-Ansätze. In diesem Bereich wird derzeit intensiv geforscht und die Weiterentwicklung entsprechender Systeme wird aktiv vorangetrieben.

Literaturverzeichnis

[Aigner 2018] Aigner, Roland ; Gebeshuber, Klaus (Hrsg.) ; Hackner, Thomas (Hrsg.) ; Kania, Stefan (Hrsg.) ; Kloep, Peter (Hrsg.) ; Kofler, Michael (Hrsg.) ; Neugebauer, Frank (Hrsg.) ; Widl, Markus (Hrsg.) ; Zingsheim, André (Hrsg.): *Hacking & Security das umfassende Handbuch*. 1. Auflage 2018, 1. korrigierter Nachdruck 2018. Rheinwerk Verlag, 2018 (Rheinwerk Computing). – URL http://d-nb.info/1149384573/04

[Brause 2017] Brause, Rüdiger W. 1.: *Betriebssysteme Grundlagen und Konzepte*. 4., erweiterte Auflage. Springer Vieweg, 2017. – URL http://dx.doi.org/10.1007/978-3-662-54100-5

[BSD 2020] BSD: *The 3-Clause BSD License*. 2020. – URL https://opensource.org/licenses/BSD-3-Clause. – Zugriffsdatum: 16. Dezember 2020

[Böhm 2019] Böhm, Markus: *Hacker griffen offenbar Airbus-Zulieferer an*. 2019. – URL https://www.spiegel.de/netzwelt/netzpolitik/airbus-hacker-griffen-offenbar-mehrere-zulieferer-an-a-1288825.html. – Zugriffsdatum: 29. September 2020

[Cisco 2018] Cisco: *Small and Mighty - How Small and Midmarket Businesses Can Fortify Their Defenses Against Today's Threats*. 2018. – URL https://www.cisco.com/c/dam/global/en_hk/products/security/security-reports/Cisco_2018_SMB_Final.pdf?oid=wprsc013702. – Zugriffsdatum: 24. Juli 2020

[Computerwelt.at 2018] Computerwelt.at: *Capture-the-Flag-Wettbewerb: Security-Nachwuchs ist gefragt*. August 2018. – URL https://computerwelt.at/news/capture-the-flag-wettbewerb-security-nachwuchs-ist-gefragt/. – Zugriffsdatum: 09. Juni 2020

[Coret 2006] Coret, Jose A.: *Kojoney - A honeypot for the SSH Service*. 2006. – URL http://kojoney.sourceforge.net/. – Zugriffsdatum: 20. Oktober 2020

[creatorbifrozt 2016] creatorbifrozt: *Bifrozt High interaction honeypot solution for Linux based systems*. 2016. – URL https://sourceforge.net/projects/bifrozt/. – Zugriffsdatum: 2. November 2020

[CTFtime.org 2020] CTFtime.org: *DEF CON CTF Qualifier*. 2020. – URL https://ctftime.org/ctf/1/. – Zugriffsdatum: 09. Juni 2020

[Debian Entwicklerteam 2020] Debian Entwicklerteam: *security.conf*. 2020. – URL https://git.launchpad.net/ubuntu/+source/apache2/tree/debian/config-dir/conf-available/security.conf?h=ubuntu/focal. – Zugriffsdatum: 8. Oktober 2020

[Degioanni 2014] Degioanni, Loris: *Sysdig vs DTrace vs Strace: A technical discussion*. 2014. – URL https://sysdig.com/blog/sysdig-vs-dtrace-vs-strace-a-technical-discussion/. – Zugriffsdatum: 12. November 2020

[desaster 2016] desaster: *Kippo*. 2016. – URL https://github.com/desaster/kippo. – Zugriffsdatum: 20. Oktober 2020

[Dictionary.com 2020] Dictionary.com: *Definition of Honeypot at Dictionary.com*. 2020. – URL https://www.dictionary.com/browse/honeypot. – Zugriffsdatum: 18. November 2020

[Edwards 2019] Edwards, Brandon: *Container Escapes: An Exercise in Practical Container Escapology*. 2019. – URL https://capsule8.com/blog/practical-container-escape-exercise/. – Zugriffsdatum: 10. Juli 2020

[Emelyanov und Kolyshkin 2007] Emelyanov, Pavel ; Kolyshkin, Kir: *PID namespaces in the 2.6.24 kernel*. 2007. – URL https://lwn.net/Articles/259217/. – Zugriffsdatum: 30. Juni 2020

[epinna 2020] epinna: *epinna/weevely3 Weaponized web shell*. 2020. – URL https://github.com/epinna/weevely3. – Zugriffsdatum: 10. Juni 2020

[@flogisoft 2020] @flogisoft: *p0wny@shell, un shell PHP simple, mais (trop) efficace*. 2020. – URL https://blog.flozz.fr/2020/01/21/p0wny-shell-un-shell-php-simple-mais-trop-efficace/. – Zugriffsdatum: 10. Juni 2020

[Glauber 2012] Glauber, Costa: *Resource Isolation: The Failure of Operating Systems & How We Can Fix It.* 2012. – URL https://linuxconeurope2012.sched.com/event/bf1a2818e908e3a534164b52d5b85bf1. – Zugriffsdatum: 30. Juni 2020

[Goodin 2007] Goodin, Dan: *Safari zero-day exploit nets $10,000 prize - Pwn'd in 12 hours.* 2007. – URL https://www.theregister.com/2007/04/20/pwn-2-own_winner/. – Zugriffsdatum: 10. Juni 2020

[Google 2013] Google: *What is Certificate Transparency?* 2013. – URL https://www.certificate-transparency.org/what-is-ct. – Zugriffsdatum: 09. Juni 2020

[Grimes 2005] Grimes, Roger A.: *Honeypots for Windows.* Apress, 2005 (The expert's voice). – 392 S. – URL http://www.gbv.de/dms/bowker/toc/9781590593356.pdf

[Harsa Wara Prabawa 2017] Harsa Wara Prabawa, Yana P.: Using Capture the Flag in Classroom: Game-based Implementation in Network Security Learning. In: *3rd International Conference on Science in Information Technology (ICSITech)* (2017)

[Internetwache.org 2015] Internetwache.org: *Don't publicly expose .git or how we downloaded your website's sourcecode - An analysis of Alexa's 1M.* 2015. – URL https://en.internetwache.org/dont-publicly-expose-git-or-how-we-downloaded-your-websites-sourcecode-an-analysis-of-alexas-1m-28-07-2015/. – Zugriffsdatum: 10. Juni 2020

[Karras u. a. 2020] Karras, Tero ; Laine, Samuli ; Aittala, Miika ; Hellsten, Janne ; Lehtinen, Jaakko ; Aila, Timo: Analyzing and Improving the Image Quality of StyleGAN. (2020)

[Kerrisk 2010] Kerrisk, Michael: *The Linux programming interface: a Linux and UNIX system programming handbook.* No Starch Press, 2010. – URL http://www.gbv.de/dms/tib-ub-hannover/615218326.pdf

[Kerrisk 2013] Kerrisk, Michael: *Namespaces in operation, part 5: User namespaces.* 2013. – URL https://lwn.net/Articles/532593/. – Zugriffsdatum: 30. Juli 2020

[Kerrisk 2020a] Kerrisk, Michael: *mount_namespaces(7) Linux Programmer's Manual*. 2020. – URL https://man7.org/linux/man-pages/man7/mount_namespaces.7.html. – Zugriffsdatum: 9. Juli 2020

[Kerrisk 2020b] Kerrisk, Michael: *namespaces(7) Linux Programmer's Manual*. 2020. – URL https://man7.org/linux/man-pages/man7/namespaces.7.html. – Zugriffsdatum: 9. Juli 2020

[Kerrisk 2020c] Kerrisk, Michael: *network_namespaces(7) Linux Programmer's Manual*. 2020. – URL https://man7.org/linux/man-pages/man7/network_namespaces.7.html. – Zugriffsdatum: 9. Juli 2020

[Kerrisk 2020d] Kerrisk, Michael: *pid_namespaces(7) Linux Programmer's Manual*. 2020. – URL https://man7.org/linux/man-pages/man7/pid_namespaces.7.html. – Zugriffsdatum: 9. Juli 2020

[Kerrisk 2020e] Kerrisk, Michael: *user_namespaces(7) Linux Programmer's Manual*. 2020. – URL https://man7.org/linux/man-pages/man7/user_namespaces.7.html. – Zugriffsdatum: 9. Juli 2020

[Krause 2020a] Krause, Uwe: *Debian Bug report logs - #958643 - sysdig: crash the whole machine and disables login after reboot by just installing package*. 2020. – URL https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=958643. – Zugriffsdatum: 30. Juli 2020

[Krause 2020b] Krause, Uwe: *Kukulkan*. 2020. – URL https://gitlab.com/kukulkanhoneypot/core/-/blob/master/README.md. – Zugriffsdatum: 16. Dezember 2020

[Krypczyk und Bochkor 2016] Krypczyk, Veikko ; Bochkor, Olena: *Prototyping - Eine Einführung*. 2016. – URL https://www.informatik-aktuell.de/entwicklung/methoden/prototyping-eine-einfuehrung.html. – Zugriffsdatum: 21. Juli 2020

[Leifer u. a. 2018] Leifer, Larry ; Lewrick, Michael ; Link, Patrick: *Das Design Thinking Playbook mit traditionellen, aktuellen und zukünftigen Erfolgsfaktoren*. 2. überarbeitete Auflage. Verlag Franz Vahlen GmbH, 2018. – URL http://www.gbv.de/dms/zbw/1009947125.pdf

[Merriam-Webster.com Dictionary 2020] Merriam-Webster.com Dictionary: *Capture the flag*. 2020. – URL https://www.merriam-webster.com/dictionary/capture%20the%20flag. – Zugriffsdatum: 09. Juni 2020

[mitre 2020] mitre: *CWE-416: Use After Free*. 2020. – URL https://cwe.mitre.org/data/definitions/416.html. – Zugriffsdatum: 11. Juli 2020

[Morag 2020] Morag, Assaf: *Threat Alert: An Attack Against a Docker API Leads To Hidden Cryptominers*. 2020. – URL https://blog.aquasec.com/container-vulnerability-dzmlt-dynamic-container-analysis. – Zugriffsdatum: 14. Juli 2020

[National Vulnerability Database 2020] National Vulnerability Database: *CVE-2020-8958*. 2020. – URL https://nvd.nist.gov/vuln/detail/CVE-2020-8958. – Zugriffsdatum: 30. November 2020

[Neugebauer 2017] Neugebauer, Rolf: *Preview: Linux Containers on Windows*. 2017. – URL https://www.docker.com/blog/preview-linux-containers-on-windows/. – Zugriffsdatum: 12. Juli 2020

[Newcomer 2017] Newcomer, Eric: *Uber Paid Hackers to Delete Stolen Data on 57 Million People*. November 2017. – URL https://www.bloomberg.com/news/articles/2017-11-21/uber-concealed-cyberattack-that-exposed-57-million-people-s-data. – Zugriffsdatum: 17. April 2020

[NSA 2020] NSA: *Russian GRU 85th GTsSSDeploys PreviouslyUndisclosed Drovorub Malware*. 2020. – URL https://media.defense.gov/2020/Aug/13/2002476465/-1/-1/0/CSA_DROVORUB_RUSSIAN_GRU_MALWARE_AUG_2020.PDF. – Zugriffsdatum: 12. November 2020

[Nvidia 2020] Nvidia: *thispersondoesnotexist.com*. 2020. – URL https://thispersondoesnotexist.com/. – Zugriffsdatum: 3. Oktober 2020

[Osnat 2020] Osnat, Rani: *A Brief History of Containers: From the 1970s Till Now*. 2020. – URL https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016. – Zugriffsdatum: 30. Juni 2020

[PHP-Dokumentationsgruppe 2020] PHP-Dokumentationsgruppe: *Dateiuploads mit POST*. 2020. – URL https://www.php.net/manual/de/features.file-upload.post-method.php. – Zugriffsdatum: 4. Oktober 2020

[Polop 2020] Polop, Carlos: *Checklist - Linux Privilege Escalation*. 2020. – URL https://book.hacktricks.xyz/linux-unix/linux-privilege-escalation-checklist. – Zugriffsdatum: 10. Juni 2020

[Portokalidis u. a. 2006] Portokalidis, G. ; Slowinska, A. ; Bos, Herbert: Argos: An emulator for fingerprinting zero-day attacks. In: *Proc. ACM SIGOPS EUROSYS* (2006), 01, S. 15–27

[lxc project 2020] project lxc: *Linux Containers - LXC - Security*. 2020. – URL https://linuxcontainers.org/lxc/security/. – Zugriffsdatum: 4. August 2020

[Provos und Holz 2010] Provos, Niels ; Holz, Thorsten: *Virtual honeypots: from botnet tracking to intrusion detection*. 4. Auflage. Addison-Wesley, 2010. – URL http://www.gbv.de/dms/bowker/toc/9780321336323.pdf

[Rice 2020] Rice, Liz: *Container Security*. O'Reilly Media, Inc., 2020. – URL https://www.oreilly.com/library/view/container-security/9781492056690/

[Riondato 2020] Riondato, Matteo: *freeBSD Handbuch: Teil III. Systemadministration Kapitel 14. Jails*. 2020. – URL https://www.freebsd.org/doc/de_DE.ISO8859-1/books/handbook/jails.html. – Zugriffsdatum: 12. Juli 2020

[Ruef 2007] Ruef, Marc: *Die Kunst des Penetration Testing*. Computer & Literatur, Böblingen, 2007. – URL http://computec.ch/mruef/?s=dkdpt

[Ruiu 2007] Ruiu, Dragos: *PWN to OWN (was Re: How Apple orchestrated web attack on researchers)*. 2007. – URL https://seclists.org/dailydave/2007/q1/289. – Zugriffsdatum: 10. Juni 2020

[Safonov 2019] Safonov, Dmitry: *[PATCH 01/32] ns: Introduce Time Namespace*. 2019. – URL https://lkml.org/lkml/2019/2/5/867. – Zugriffsdatum: 11. Juli 2020

[Sleevi 2015] Sleevi, Ryan: *Sustaining Digital Certificate Security*. 2015. – URL https://security.googleblog.com/2015/10/sustaining-digital-certificate-security.html. – Zugriffsdatum: 09. Juni 2020

[Song 2000] Song, Dug: *dsniff*. 2000. – URL https://www.monkey.org/~dugsong/dsniff/. – Zugriffsdatum: 4. November 2020

[Strozyk 2020] Strozyk, Jan L.: *Daten von Zehntausenden Webseiten zugänglich*. Juli 2020. – URL https://www.tagesschau.de/investigativ/ndr/it-sicherheit-quellcodes-101.html. – Zugriffsdatum: 03. Juli 2020

[Tagesschau 2020] Tagesschau: *Millionen Kundendaten im Netz: Riesiges Datenleck bei Autovermieter*. 2020. – URL https://www.tagesschau.de/inland/datenleak-autovermietung-buchbinder-101.html. – Zugriffsdatum: 17. April 2020

[Tanriverdi u. a. 2019] Tanriverdi, Hakan ; Eckert, Svea ; Strozyk, Jan ; Zierer, Maximilian ; Ciesielski, Rebecca: *Winnti: Angriff auf das Herz der deutschen Industrie*. 2019. – URL https://web.br.de/interaktiv/winnti/. – Zugriffsdatum: 24. Juli 2020

[Tobias u. a. 2009] Tobias, Peter ; Eckenfels, Bernd ; Meskes, Michael: *hostname(1) Linux Programmer's Manual*. 2009. – URL https://manpages.debian.org/buster/hostname/hostname.1.en.html. – Zugriffsdatum: 9. Juli 2020

[VMWARE 2020] VMWARE: *VMSA-2020-0015.2*. 2020. – URL https://www.vmware.com/security/advisories/VMSA-2020-0015.html. – Zugriffsdatum: 11. Juli 2020

[Wikipedia 2019] Wikipedia: *Quetzalcoatl — Wikipedia, Die freie Enzyklopädie*. 2019. – URL https://de.wikipedia.org/w/index.php?title=Quetzalcoatl&oldid=193371818. – Zugriffsdatum: 20. April 2020

[Wilczek 2019] Wilczek, Marc: *Über die Hälfte der KMU erfolgreich gehackt*. 2019. – URL https://www.computerwoche.de/a/ueber-die-haelfte-der-kmu-erfolgreich-gehackt,3546999. – Zugriffsdatum: 24. Juli 2020

A Anhang

A.1 Ausführung der Installationsskripte

Die Installation erfolgt mit Hilfe der erstellten Skripte. Die einzelnen Unterskripte repräsentieren den modularen Aufbau des Honeypotsystems. Die notwendigen Befehle für die einzelnen Schritte lassen sich aus der Installationsanleitung in eine Administrationskonsole kopieren.

A.1.1 Manuelle Vorarbeit

- git installieren
- Repositoryadressen (Kukulkan und Website) auf Host hinterlegen
- ssh deployment key generieren
  - Falls nicht öffentlich zugänglich:
    Private Key in Kukulkan und Website Repository hinterlegen!
- Kukulkan Code aus Repository klonen
- Konfiguration überprüfen und ggf. anpassen
  Die wichtigsten Punkte:
  - Auf Host verwendeter Netzwerkadapter
  - Für die Verbindung zum Host genutzter ssh-Port
  - Limits für Container (CPU, RAM, Netzwerk)

A.1.2 Von Skripten durchgeführte Einrichtungsschritte

Der Host wird einmalig initialisiert.

0. initHost.sh
   a) initHostContainment.sh
      i. Installiert lxc deamon (lxd)

   ii. Initialisiert lxd
   iii. Sichert initialen Stand der iptables
b) initHostSurveillance.sh
   i. Installiert sysdig und sysdig Kernelmodul
   ii. Sieht Überwachungs-Skript für täglichen Neustart vor
c) initHostSshd.sh
   i. Konfiguriert sshd Zugangsport auf dem Host

Der Anwender muss sich nun neu mit dem Host unter Verwendung des neuen Zugangsports verbinden. Anschließend wird der Gast eingerichtet und die Überwachung gestartet, indem das Skript `kukulkan.sh` ausgeführt wird, das die folgenden vier Unterskripte automatisch durchführt:

1. initGuest.sh
   a) initGuestOperatingSystem.sh
      i. Erstellt Container mit Betriebssystem-Image
      ii. Aktualisiert Gast-Betriebssystem
   b) initGuestSshd.sh
      i. Installiert im Gast sshd
      ii. Erstellt im Gast Anwender mit Passwort
      iii. Berechtigt im Gast Anwender als Systemadministrator
   c) initGuestApache.sh
      i. Installiert im Gast apache2 und php
   d) initGuestApacheHttps.sh (optional)
      i. Installiert im Gast LetsEncrypt certbot
      ii. Führt im Gast certbot aus
         A. fordert SSL-Zertifikat von LetsEncrypt an
         B. konfiguriert Webserver im Gast zur Nutzung des Zertifikats
   e) initGuestLimits.sh
      i. Limitiert CPU-Ressourcen für den Gast
      ii. Limitiert dem Gastsystem zustehenden Arbeitsspeicher
      iii. Limitiert Netzwerkgeschwindigkeit des Gastsystems
2. startGuest.sh
   a) Startet Container
   b) portForward.sh
      i. Konfiguriert iptables des Host zur Weiterleitung aller Ports vom Host zum Gast (Ausgenommen Host-SSH-Port)

3. populateGuestApacheWebsite.sh
    a) lädt Website-Dateien auf den Host
    b) kopiert Website-Dateien in den Gast
4. startSurveilance.sh
    a) Stellt das angepasste Kukulkan chisel für sysdig bereit
    b) startet Überwachung in screen-Sitzung

Der Honeypot ist jetzt komplett eingerichtet und die Überwachung wurde gestartet. Die Überwachungsdaten werden auf dem Dateisystem des Host in Protokolldateien gespeichert.

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: _____

Vorname: _____

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

Kukulkan high interaction honeypot:
Ein modulares System zur sicheren Protokollierung von Privilegien-Erweiterungs-Angriffen gegen weitverbreitete Serverbetriebssysteme und Dienste

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