



Hochschule Darmstadt

– Fachbereich Informatik –

Dynamische Analyse von Linux-Malware

Abschlussarbeit zur Erlangung des akademischen Grades
Bachelor of Science (B.Sc.)

vorgelegt von

Nils Rogmann

Matrikel-Nr. 725915

Referent: Prof. Dr. Marian Margraf

Korreferent: Prof. Dr. Michael Braun

Ausgabedatum: 12. Dezember 2014

Abgabedatum: 13. März 2015

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht. Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen. Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Darmstadt, den 13. März 2015

Abstrakt

Sicherheitsvorfälle können im Unternehmenskontext trotz vielseitiger Schutzmaßnahmen und dem Einsatz aktueller IT-Sicherheitsinfrastrukturen niemals vollständig ausgeschlossen werden. Wird während der Bearbeitung eines Sicherheitsvorfalls, beispielsweise durch ein Incident Response-Team, potentielle Malware auf einem System entdeckt, muss diese zur Regulierung von Schäden und weiteren Infektionen sowie zum generellen Schutz von Unternehmensdaten umgehend analysiert werden. Hierbei erfolgt grundsätzlich eine Unterscheidung zwischen der statischen Analyse zur Untersuchung des zu Grunde liegenden Codes eines Schadprogramms und der dynamischen Analyse, also der kontrollierten Ausführung und dedizierten Beobachtung einer Malware.

Die Malware-Analyse soll die Eindämmung eines Sicherheitsvorfalls unterstützen sowie eine vollständige Säuberung infizierter Systeme innerhalb der Infrastruktur eines Unternehmens ermöglichen. Während zur sicheren und automatisierten Analyse von Windows-Malware bereits etablierte kommerzielle sowie nicht-kommerzielle Lösungen existieren, muss die Untersuchung von Linux-Malware bisher unter einem großen Zeitaufwand manuell durchgeführt werden.

Zielsetzung der Bachelorarbeit ist es daher, einen Prototyp zur sichereren und automatisierten Analyse von potentieller Linux-Malware zu entwickeln, der zukünftig beispielsweise im Zuge eines Incident Response-Einsatzes verwendet werden kann. Hierzu soll ein bereits zur statischen Analyse realisierter und auf der Cuckoo Sandbox basierender Prototyp, der während des vorangegangenen Praxisprojekts entwickelt wurde, um Funktionen und Techniken zur dynamischen Analyse von Linux-Malware erweitert werden.

Als Ergebnis der Bachelorarbeit wird ein Prototyp zur statischen und dynamischen Analyse von Linux-Malware zur Verfügung gestellt. Hierzu erfolgte während der Ausarbeitung die Entwicklung verschiedener Techniken zur Erfassung von Prozess-, Dateisystem- und Netzwerkaktivitäten eines zu Grunde liegenden Schadprogramms. Die Techniken wurden als Module zur dynamischen Analyse von Linux-Malware in den bestehenden Prototyp implementiert und können zukünftig zur effizienten, sicheren sowie automatisierten Bestimmung des Verhaltens einer Malware eingesetzt werden.

Danksagung

An dieser Stelle möchte ich mich bei allen Personen bedanken, die mich bei der Erstellung dieser Bachelorarbeit unterstützt haben.

Zunächst gilt mein Dank meinen Kollegen bei der Controlware GmbH. Petra Weiß, die mich stets mit kreativen Ideen und wertvollen Anmerkungen unterstützt und meine Ausarbeitung damit bereichert hat. Nico Peikert und Andreas Bunten, die als Fachbetreuer fortwährend ein offenes Ohr für verschiedenste strukturelle und technische Fragestellungen hatten und diese Bachelorarbeit durch gemeinsame Diskussionen verbessert haben. Johannes Bachmann, der als Korrekturleser die gesamte Arbeit durchgelesen hat, um Verständnisschwierigkeiten und sprachliche Unstimmigkeiten zu identifizieren.

Weiterhin möchte ich mich bei Prof. Dr. Marian Margraf bedanken. Er hat mir als Referent während der Erstellung meiner Arbeit als Ansprechpartner zur Verfügung gestanden und mir gleichzeitig den notwendigen Freiraum und das Vertrauen gegeben, diese Bachelorarbeit nach meinen Vorstellungen zu gestalten. Auch möchte ich Prof. Dr. Michael Braun meinen Dank aussprechen, der sich als Korreferent die Zeit genommen hat, mich und meine Arbeit zu betreuen.

Mein besonderer Dank gilt meiner Lebensgefährtin und wichtigsten Freundin, Denise Muth, die mich während der Erstellung dieser Bachelorarbeit durch ihre einzigartig liebevolle und zugleich strenge Art unterstützt hat. Sie hinterfragte komplexe technische Details und diskutierte mit mir, ob an Tag oder Nacht, über konzeptionelle Feinheiten und Formulierungen innerhalb meiner Arbeit. Zuletzt sorgte sie immer wieder dafür, dass ich neben dem Ernst des Studiums die schönen und wesentlichen Dinge des Lebens nicht aus den Augen verlor und gab mir so die nötige Motivation zur Erstellung dieser Arbeit.

Abschließend möchte ich meiner Mutter danken. Karin Abels, die mir während des gesamten Bachelorstudiums Rückhalt gegeben und an mich geglaubt hat. Dank ihrer Unterstützung und Fürsorge war es mir möglich, mein bisheriges Studium und diese Arbeit so erfolgreich zu meistern.

Inhaltsverzeichnis

ABBILDUNGSVERZEICHNIS	VII
1 EINLEITUNG.....	1
1.1 MOTIVATION	1
1.2 ZIELSETZUNG.....	2
1.3 AUFBAU DER ARBEIT	3
2 GRUNDLAGEN	4
2.1 INCIDENT RESPONSE.....	4
2.1.1 <i>Notwendigkeit</i>	4
2.1.2 <i>Prozess</i>	5
2.2 MALWARE	8
2.2.1 <i>Definition</i>	8
2.2.2 <i>Klassifizierung</i>	8
2.3 MALWARE-ANALYSE	11
2.3.1 <i>Statische Analyse</i>	11
2.3.2 <i>Dynamische Analyse</i>	12
2.3.3 <i>Sichere Umgebung</i>	14
3 TECHNIKEN.....	17
3.1 PROZESSAKTIVITÄTEN.....	17
3.1.1 <i>Kernel-Modul „Process Events Connector“</i>	18
3.1.2 <i>Systemaufruf „Process Trace“</i>	20
3.2 DATEISYSTEMAKTIVITÄTEN	23
3.2.1 <i>Systemaufruf „Process Trace“</i>	23
3.2.2 <i>Kernel-Modul „Inode Notify“</i>	24
3.3 NETZWERKAKTIVITÄTEN	26

3.3.1	<i>Systemaufruf „Process Trace“</i>	26
3.3.2	<i>Kernel-Modul „Packet Capture“</i>	28
4	REALISIERUNG	31
4.1	CUCKOO SANDBOX	31
4.1.1	<i>Konzept</i>	31
4.1.2	<i>Funktionsweise</i>	33
4.2	IMPLEMENTIERUNG	34
4.2.1	<i>Modul „Syscall Tracer“</i>	35
4.2.2	<i>Modul „Filesystem Tracer“</i>	39
4.2.3	<i>Schnittstelle „Result Logger“</i>	41
5	ANWENDUNG	44
5.1	BEISPIELSZENARIO	44
5.2	MALWARE-ANALYSE	45
5.2.1	<i>Webschnittstelle</i>	45
5.2.2	<i>Analyse</i>	46
5.2.3	<i>Ergebnisbericht</i>	48
6	RESÜMEE	51
6.1	RÜCKBLICK	51
6.2	ERGEBNIS	52
6.3	AUSBLICK	52
6.4	FAZIT	53
ANHANG A	WEITERFÜHRENDE MATERIALIEN	54
A.1.	QUELLCODE „CREATE_FILE.C“	54
ANHANG B	ABKÜRZUNGSVERZEICHNIS	55
ANHANG C	LITERATURVERZEICHNIS	56

Abbildungsverzeichnis

Abbildung 1: Incident Response-Prozess (vgl. [Nat12], S. 21).....	5
Abbildung 2: Prozessüberwachung mithilfe von proc connector	19
Abbildung 3: Prozessüberwachung unter Verwendung von ptrace	21
Abbildung 4: Überwachung von Dateisystemzugriffen mittels strace	24
Abbildung 5: Dateisystemüberwachung mithilfe von inotify.....	25
Abbildung 6: Überwachung von Netzwerkzugriffen mittels strace	27
Abbildung 7: Netzwerküberwachung unter Verwendung von pcap.....	28
Abbildung 8: Ablauf der Malware-Untersuchung innerhalb einer Analyse-Sandbox.....	32
Abbildung 9: Übersicht über die Funktionsweise der Cuckoo Sandbox	33
Abbildung 10: Realisierung der Überwachung mit Syscall Tracer	35
Abbildung 11: Detektion einer Prozessüberwachung mithilfe von ptrace	37
Abbildung 12: Verschleierung der Verfolgung durch ptrace	38
Abbildung 13: Realisierung der Überwachung mit Filesystem Tracer	40
Abbildung 14: Implementierung der Schnittstelle Result Logger	42
Abbildung 15: Start einer Malware-Analyse über das Webformular von Cuckoo.....	45
Abbildung 16: Übermittlung der eingegebenen Daten in die Cuckoo-Datenbank	46
Abbildung 17: Echtzeit-Übertragung der erfassten Malware-Aktivitäten.....	47
Abbildung 18: Auszug aus einem Ergebnisbericht der Cuckoo Sandbox	48
Abbildung 19: Auszug der durch Cuckoo erfassten Netzwerkaktivitäten.....	49
Abbildung 20: Quellcode des Programms „create_file“	54

1 Einleitung

Trotz einer Vielzahl von Schutzmaßnahmen und dem Einsatz aktueller Sicherheitsinfrastrukturen können Sicherheitsvorfälle in Form von Systemeintrüben im Unternehmenskontext nicht ausgeschlossen werden. Aufgrund der Anbindung zu öffentlichen Netzen ereignet sich ein Einbruch präferiert über Serversysteme, die wiederum zur Verbreitung von Schadsoftware, zum Datendiebstahl oder zum Angriff weiterer Systeme missbraucht werden können. Hieraus kann ein enormer wirtschaftlicher Schaden für ein Unternehmen entstehen, weshalb ein solcher Vorfall nicht zu ignorieren ist. Erfolgt eine Detektion sicherheitsrelevanter Anomalien innerhalb der IT-Infrastruktur eines Unternehmens, ist daher eine umgehende Analyse und Behebung notwendig.

Das National Institute of Standards and Technology (NIST) empfiehlt hierzu den Einsatz eines Incident Response-Teams, welches sich reaktiv der Untersuchung und Behebung von Sicherheitsvorfällen, wie Serversystemeintrüben, widmet (vgl. [Nat12], S. 13). Wird im Zuge der Bearbeitung eines Vorfalls auf einem System potentielle Schadsoftware (Malware) vorgefunden, ist diese unverzüglich durch das Incident Response-Team als solche zu identifizieren und ihr Verhalten zu analysieren. Im Allgemeinen wird hierbei grundsätzlich zwischen der **statischen Analyse** und **dynamischen Analyse** unterschieden (vgl. [SH12], S. 2-3). Während Erstere eine Untersuchung des vorliegenden Codes einer Software beinhaltet, *ohne* diese auszuführen, wird im Rahmen der dynamischen Analyse eine Software kontrolliert gestartet und ihr zu Grunde liegendes Verhalten beobachtet.

1.1 Motivation

Die Malware-Analyse dient in diesem Kontext der Beantwortung einiger Fragen, welche sowohl zur vollständigen Beseitigung des Vorfalls als auch aus wirtschaftlicher Sicht für ein Unternehmen essentiell sind:

- ▶ Wie erfolgte die Infizierung des Systems und wie funktioniert die Malware?
- ▶ Welche Daten wurden vom System und dem Unternehmen entwendet?
- ▶ Hat eine Infizierung weiterer Systeme stattgefunden und was sind Indizien dafür?
- ▶ Wie lässt sich die Malware vollständig entfernen?

Durch die Analyse soll insgesamt die weitere Verbreitung von Malware unterbunden und gleichzeitig eine Säuberung der infizierten Systeme ermöglicht werden.

Zum Schutz des Unternehmens ist folglich eine schnelle sowie effektive Untersuchung wesentlich. Damit kein weiterer Schaden entsteht oder zusätzliche Systeme infiziert werden, sollte die Malware aus Sicherheitsgründen ausschließlich auf einem speziellen Testsystem und ferner innerhalb einer geschützten Umgebung, einer Sandbox, ausgeführt werden. Das Prinzip der Sandbox wird demnach genutzt, um Schadsoftware von anderen Systemkomponenten zu isolieren und die Modifikation oder Beschädigung des zu Grunde liegenden Systems zu verhindern. Innerhalb der isolierten Umgebung ist schließlich die sichere Untersuchung des Verhaltens von unbekanntem und unmittelbar verdächtigen Dateien möglich. Zur sicheren automatisierten Analyse von Windows-Malware existieren neben einigen kommerziellen Produkten auch nicht-kommerzielle Lösungen und Open Source-Projekte. Falls jedoch Schadsoftware für Linux-Systeme sichergestellt wird, muss diese aufgrund einer fehlenden automatisierten Lösung bisher unter einem großen Zeitaufwand manuell untersucht werden.

1.2 Zielsetzung

Als Ergebnis des vorangegangenen Praxisprojekts wurde auf Grundlage einer quelloffenen Malware Analyse-Umgebung für Windows-Systeme, der Cuckoo Sandbox, ein Prototyp realisiert, welcher bereits im Rahmen eines Incident Response-Einsatzes zur statischen Analyse von Linux-Malware eingesetzt werden kann (vgl. [Rog14], S. 18-23). Im Zuge der Bachelorarbeit gilt es nun, sich den verschiedenen Techniken der dynamischen Analyse von Malware auf Linux-Systemen zu widmen sowie den Prototyp dahingehend zu erweitern. Hierbei ist es zunächst erforderlich, aktuell verfügbare Techniken, die zur umfassenden dynamischen Analyse eingesetzt werden können, herauszuarbeiten und tiefgehend zu beleuchten. Basierend auf den gesammelten Ergebnissen, soll dann die Implementierung der Funktionen zur dynamischen Analyse erfolgen.

Das Ziel der Bachelorarbeit besteht darin, zukünftig durch die Verwendung des Prototyps eine sichere statische und dynamische Untersuchung von potentieller Linux-Malware, beispielsweise im Zuge eines Incident Response-Einsatzes, zu ermöglichen.

Ferner sollen die in der Motivation herausgearbeiteten grundlegenden Fragen mithilfe der Lösung beantwortet und in Zukunft eintretende Sicherheitsvorfälle somit effizienter behoben werden können.

1.3 Aufbau der Arbeit

Die Bachelorarbeit umfasst neben der **Einleitung**, welche die Motivation und Zielsetzung beinhaltet, insgesamt fünf weitere Kapitel.

Beginnend mit den **Grundlagen** erfolgt eine Einführung in verschiedene Themengebiete, die zum Verständnis der weiteren Inhalte dieser Bachelorarbeit relevant sind. Hierbei wird der allgemeine Incident Response-Prozess beleuchtet, die Definition sowie eine Klassifizierung von Malware durchgeführt und zuletzt ein grundlegendes Wissen über die Malware-Analyse vermittelt.

Zur dynamischen Analyse von Linux-Malware werden im Kapitel **Techniken** verschiedene Ansätze zur Erfassung von Prozess-, Dateisystem- und Netzwerkaktivitäten während der Ausführung eines Schadprogramms erarbeitet und tiefgehend betrachtet.

Innerhalb der **Realisierung** erfolgt in Anlehnung an die Cuckoo Sandbox zunächst die Darstellung eines grundlegenden Konzepts und der Funktionsweise einer Malware Analyse-Umgebung, die zur sicheren Ausführung von Schadprogrammen eingesetzt werden kann. Ferner werden die während der Ausarbeitung entwickelten Module der Cuckoo Sandbox zur dynamischen Analyse von Linux-Malware beschrieben.

Zur Demonstration der konkreten Anwendbarkeit der entwickelten Analyse-Module wird im Kapitel **Anwendung** zunächst ein Beispielszenario auf Grundlage einer echten Malware definiert. Unter Verwendung der entwickelten Module werden daran anknüpfend der beispielhafte Ablauf und das Ergebnis der automatisierten dynamischen Analyse des vorgestellten Schadprogramms beschrieben.

Abgeschlossen wird die Bachelorarbeit mit dem Kapitel **Resümee**, welches den Kern der Arbeit zusammenfasst, die Ergebnisse nochmals herausstellt, eine Verifizierung der gesetzten Ziele durchführt und zuletzt die Arbeit durch ein persönliches Fazit abrundet.

2 Grundlagen

In diesem Kapitel erfolgt die Einführung in die zum Verständnis der Bachelorarbeit relevanten Themengebiete. Die Grundlagen sind hierzu in die Bereiche „Incident Response“, „Malware“ und „Malware-Analyse“ gegliedert. Im ersten Teil erfolgt die Beschreibung des allgemeinen Incident Response-Prozesses zur Behandlung eines Sicherheitsvorfalls sowie die Einordnung der Analyse von Malware in diesen Kontext. Daran anknüpfend beschäftigt sich das zweite Unterkapitel mit der Definition und Klassifizierung von Malware, während im letzten Teil essentielle Grundlagen zur statischen und dynamischen Analyse sowie zur Verwendung einer sicheren Analyse-Umgebung herausgearbeitet werden.

2.1 Incident Response

Das Bundesamt für Sicherheit in der Informationstechnik (BSI) versteht unter dem Begriff „(Security) Incident Response“ die Bearbeitung und Behebung von Sicherheitsvorfällen innerhalb der Informationstechnik (vgl. [Bun09]). Hierzu ergänzend definiert das NIST einen Sicherheitsvorfall (engl. Computer Security Incident) als eine Verletzung oder unmittelbar bevorstehende Gefährdung der IT-Sicherheitsrichtlinien eines Unternehmens (vgl. [Nat12], S. 6).

2.1.1 Notwendigkeit

Ein klassischer Sicherheitsvorfall besteht in einem Angriff auf das Unternehmensnetzwerk über öffentlich bereitgestellte Dienste, zum Beispiel einem Webserver. Verschafft sich ein Angreifer Zugriff auf ein solches System, besteht die Gefahr der Kompromittierung weiterer Systeme innerhalb der Infrastruktur. Ferner könnten Kunden- und Firmendaten entwendet, manipuliert oder gelöscht werden. Somit ist zur Behebung des Vorfalls sowie zur Vermeidung größerer Schäden ein effizientes und systematisches Vorgehen, wie es während eines Incident Response-Einsatzes angewendet wird, erforderlich. Die Analyse von unbekannter Malware ist hierbei ein kleiner, jedoch wesentlicher Bestandteil eines umfassenden Prozesses, der die Erhaltung bzw. Wiederherstellung der Informationssicherheit im laufenden Betrieb eines Unternehmens gewährleisten soll.

2.1.2 Prozess

Das NIST unterteilt den Incident Response-Prozess (siehe Abbildung 1) in die vier Phasen „Preparation“, „Detection & Analysis“, „Containment, Eradication & Recovery“, und „Post-Incident Activity“ (vgl. [Nat12], S. 21). Nachfolgend werden die wesentlichen und für die Bachelorarbeit relevanten Inhalte der einzelnen Phasen beleuchtet.

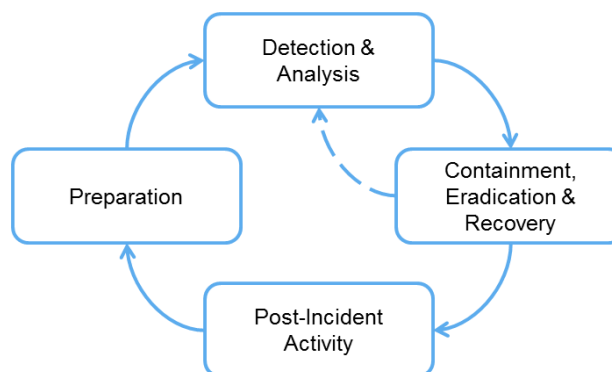


Abbildung 1: Incident Response-Prozess (vgl. [Nat12], S. 21)

Preparation

Die erste Phase beginnt mit der Etablierung eines Incident Response-Teams, welches die Behandlung von Sicherheitsvorfällen organisiert und durchführt. Dieses muss grundlegende Informationen über das Unternehmen, wie den Aufbau des Netzwerks, die eingesetzte IT-Sicherheitsinfrastruktur sowie die aktiven Dienste und Systeme zusammenstellen und aufbereiten. Zusätzlich sind Werkzeuge, wie spezielle Hard- und Software zur Malware-Analyse, zu beschaffen sowie für zukünftige Einsätze regelmäßig die ordnungsgemäße Funktionalität und Aktualität sicherzustellen. Auf diese Weise ist stets die Möglichkeit zur schnellen Reaktion auf einen Sicherheitsvorfall gewährleistet.

Auch wenn ein Incident Response-Team grundsätzlich nicht für die Implementierung der Sicherheitsinfrastruktur zur Absicherung eines Unternehmens zuständig ist, sollte es im Rahmen der Vorbereitung durchaus beratend zur Vermeidung bzw. Reduzierung von Sicherheitsvorfällen beitragen. Hierzu kann unter anderem eine regelmäßige Risikoabschätzung der Systeme und Anwendungen im Unternehmensnetzwerk durchgeführt werden. Dies umfasst auch eine grundlegende Bewertung der etablierten IT-Sicherheitsinfrastruktur. Nach Möglichkeit sind im Zuge dessen aktuelle Sicherheitsupdates auf den Systemen und Anwendungen zu installieren.

Ferner sollten alle Systeme mit einer Software zur Erkennung und Beseitigung von Malware ausgestattet werden. Zuletzt sind alle Mitarbeiter im Hinblick auf die Unternehmensrichtlinien zur sicheren Verwendung der vorhandenen Infrastruktur zu schulen (vgl. [Nat12], S. 24).

Detection & Analysis

Die Erkennung von Sicherheitsvorfällen kann einerseits mithilfe automatisierter Lösungen, wie Firewalls, Intrusion Detection Systemen (IDS) oder Anti Virus-Software erfolgen. Ferner erlaubt ein Content Filter, wie zum Beispiel ein Proxy, den unmittelbaren Schutz von Endgeräten vor infizierten E-Mails oder Webseiten. Auch durch den Einsatz eines Security Information & Event Management-Systems (SIEM), welches zur Erkennung von Anomalien eine zentrale Auswertung von Log-Dateien aller sicherheitsrelevanten Systeme durchführt, ist eine frühzeitige Erkennung von Sicherheitsvorfällen möglich. Darüber hinaus sollten im Allgemeinen auch die Hinweise von Nutzern sowie öffentlich zugängliche Berichte über Schwachstellen berücksichtigt werden.

Aufgrund der Vielzahl der zu prüfenden Informationsquellen stellt die Erkennung von echten Vorfällen in der Regel die größte Herausforderung innerhalb des Incident Response-Prozesses dar. Eine weitere Schwierigkeit besteht in der Filterung von Falschmeldungen, welche häufig durch automatisierte Systeme, aber auch durch die Meldungen der Nutzer entstehen können (vgl. [Nat12], S. 29). Wird jedoch ein echter Sicherheitsvorfall als solcher identifiziert, muss das Incident Response-Team umgehend eine initiale Untersuchung zur Bestimmung des weiteren Vorgehens einleiten. Hierbei gilt es insbesondere zu bestimmen,

- ▶ welche Systeme und Anwendungen des Unternehmens betroffen sind,
- ▶ wer oder was als Auslöser für den Vorfall identifiziert werden kann, und
- ▶ ob bereits die Ausnutzung einer spezifischen Schwachstelle oder der Einsatz von bestimmten Angriffswerkzeugen zu erkennen ist.

Wird im Rahmen der initialen Untersuchung festgestellt, dass Malware in die Infrastruktur des Unternehmens eingeschleust wurde, muss diese unverzüglich tiefgehend analysiert werden.

Hierzu sollte zunächst die Abfrage der Signatur-Datenbanken von Herstellern etablierter Anti Virus-Lösungen erfolgen. Handelt es sich jedoch um eine unbekannte Malware, ist eine manuelle Untersuchung zur Beantwortung der in Kapitel 1.1 herausgearbeiteten Fragen notwendig. Demnach erfolgt die Durchführung der statischen und dynamischen Analyse in der zweiten Phase des Incident Response-Prozesses.

Containment, Eradication & Recovery

Durch die Eindämmung eines Sicherheitsvorfalls soll die Gefahr der Entstehung weiterer Schäden für das betroffene Unternehmen minimiert werden. Abhängig von der Art und den konkreten Gegebenheiten des Vorfalls, ist hierzu eine Vielzahl verschiedener Strategien denkbar. So kann die temporäre Blockierung von externen IP-Adressen an der Firewall einen netzwerkbasieren Denial of Service-Angriff unterbinden, während die Isolierung eines infizierten Servers oder die gezielte Abschaltung eines verwundbaren Dienstes die weitere Ausbreitung von Malware innerhalb des Unternehmensnetzwerks verhindern kann. Bei jeder Strategie muss grundsätzlich eine Abwägung der möglicherweise damit einhergehenden temporären Einschränkungen für den Betrieb des Unternehmens erfolgen.

Zuletzt wird der Normalbetrieb aller betroffenen Systeme wiederhergestellt. Im Zuge dessen erfolgen, falls notwendig, die Bereinigung infizierter Systeme, die Behebung von Schwachstellen sowie die Installation von Sicherheitsupdates. Ferner werden die zur Eindämmung durchgeführten Maßnahmen nach Möglichkeit wieder rückgängig gemacht. Falls während dieser Schritte noch weitere infizierte Systeme identifiziert werden, ist die Phase „Detection & Analysis“ erneut zu durchlaufen (siehe Abbildung 1).

Post-Incident Activity

Nach der erfolgreichen Behebung des Sicherheitsvorfalls sind alle gesammelten Informationen in einem Abschlussbericht festzuhalten. Dieser kann zukünftig während der Bearbeitung vergleichbarer Vorfälle herangezogen werden. Darüber hinaus sollte in einem Abschlussmeeting aller Beteiligten eine gemeinsame Reflektion stattfinden. Hierbei können die gesammelten Erfahrungen ausgetauscht und ggf. Maßnahmen zur Verbesserung der Sicherheitsinfrastruktur des Unternehmens sowie des Incident Response-Prozesses selbst herausgearbeitet werden (vgl. [Nat12], S. 38-39).

2.2 Malware

Die Zahl der durch Malware infizierten Systeme steigt zunehmend an. Das BSI hält in seinem Bericht „Die Lage der IT-Sicherheit in Deutschland 2014“ allein in Deutschland eine monatliche Zunahme um ungefähr eine Millionen Malware-Infektionen fest. Ferner werden nach Aussage des BSI jeden Tag ca. 300.000 neue Varianten von Schadprogrammen entdeckt (vgl. [Bun14], S. 16).

2.2.1 Definition

Der Begriff „Malware“ lässt sich aus den Wörtern *malus*, lateinisch für schlecht bzw. *malicious*, englisch für bösartig, und dem englischen Wort *Software* ableiten¹.

Das NIST beschreibt Malware als ein Computerprogramm, welches vorwiegend unbemerkt in ein fremdes System eingeschleust wird, um dieses zu kompromittieren (vgl. [Nat05], S. 15). Es existiert eine Vielzahl verschiedener Typen dieser Programme, die sich in ihrer Funktionalität und somit auch hinsichtlich ihres Einsatzzwecks voneinander unterscheiden.

2.2.2 Klassifizierung

Nachfolgend werden die allgemeinen Eigenschaften der am meist verbreiteten Malware-Typen beschrieben und Unterschiede sowie Gemeinsamkeiten herausgearbeitet.

Virus

Ein Virus ist ein Schadprogramm, welches sich durch seine eigenständige Verbreitung auf einem Computer oder innerhalb eines Computernetzwerks auszeichnet (vgl. [Nat05], S. 15). Die Verbreitung erfolgt in der Regel durch eine kontinuierliche Infizierung von Dateien, Prozessen oder Diensten. Viren werden durch einen Auslöser, zum Beispiel die Ausführung einer Datei oder das Öffnen eines E-Mail-Anhangs, aktiviert. Zur Verbreitung ist also die Interaktion eines Benutzers erforderlich. Das NIST unterscheidet darüber hinaus zwischen *kompilierten* und *interpretierten* Viren.

¹ Pons Wörterbuch, Herleitung *malus*: <http://de.pons.com/übersetzung/latein-deutsch/malus>

Während Ersterer eigenständig auf einem System gestartet werden können, erfolgt die Ausführung interpretierter Viren ausschließlich durch einen Dienst oder eine bestimmte Anwendung, wie einem PDF-Reader (vgl. [Nat05], S. 15-16).

Wurm

Ein Wurm zeichnet sich, wie der Virus, ebenfalls durch seine eigenständige Verbreitung aus. Diese kann jedoch ohne Interaktion eines Benutzers erfolgen (vgl. [Nat05], S. 17-18). Demnach verbreitet sich ein Wurm, indem er sich selbst kopiert und anschließend ausführt. Während ein Virus also den Anhang einer E-Mail infiziert und darauf wartet, von einem Benutzer verschickt zu werden, würde ein Wurm sowohl einen infizierten Anhang generieren als auch den E-Mail-Versand durchführen. Der Versand wird beispielsweise mittels einer gezielten Durchsuchung des Systems nach bestehenden Accounts und Adressbüchern realisiert. Zur automatisierten Verbreitung werden außerdem oft bekannte Schwachstellen sowie fehlerhafte Konfigurationen von Netzwerken und Diensten ausgenutzt.

Trojanisches Pferd

Ein trojanisches Pferd, kurz Trojaner, tarnt sich als gutartiges Programm, das verdeckt bösartige Funktionen ausführt (vgl. [Nat05], S. 18-19). Eine solche Software wird verwendet, um bestehende Programme oder Dienste eines Systems durch eine modifizierte Version zu ersetzen. Ein mögliches Ziel kann darin bestehen, unbemerkt Daten wie Passwörter, Kreditkarteninformationen oder personenbezogene Daten zu entwenden. Im Gegensatz zu Viren und Würmern, verbreitet sich ein Trojaner in der Regel nicht selbst, sondern wird vielmehr gezielt durch die Entwickler bei ausgewählten Zielen eingeschleust.

Backdoor

Das NIST beschreibt eine Backdoor als Programm (Client), welches auf einem bereits infizierten System installiert wird und auf Befehle wartet, die über eingehende Netzwerkverbindungen von einem Angreifer (Server) gesendet werden (vgl. [Nat05], S. 21). Durch eine zu einem späteren Zeitpunkt etablierte Verbindung kann der Server den Client und somit das Zielsystem kontrollieren. Wie groß der Grad der Kontrolle tatsächlich ist, hängt von der Berechtigung des Clients ab.

Eine Backdoor wird häufig durch schädliche Programme, wie Trojaner, Viren und Würmer auf einem System etabliert, damit ein Angreifer sich zu einem späteren Zeitpunkt erneut darauf verbinden kann.

Bot

Im Malware-Kontext handelt es sich bei einem Bot in der Regel um ein sich selbst verbreitendes Schadprogramm, welches ein Zielsystem infiziert und eine Verbindung zu einem „Command and Control“-Server aufbaut (vgl. [Cis15]). Über den Server erfolgt die zentrale Steuerung eines oder mehrerer Bots, die wiederum zur Ausführung von Befehlen innerhalb der infizierten Systeme verwendet werden. Der Zusammenschluss von mehreren Bots wird als Botnetz bezeichnet.

Keylogger

Ein Keylogger wird dazu eingesetzt, unbemerkt Tastatureingaben aufzuzeichnen (vgl. [Nat05], S. 21). Das Ziel besteht meist darin, vertrauliche Informationen wie Passwörter, PINs oder Kreditkartennummern abzuspeichern und einem Angreifer zukommen zu lassen. Ein Keylogger wird, ähnlich wie eine Backdoor, häufig durch einen Trojaner zu weiteren Spionagezwecken auf ein Zielsystem installiert. Ferner kann die Etablierung über modifizierte Hardware, wie einem vermeintlichen USB-Stick oder einer Tastatur erfolgen. Die gesammelten Daten werden entweder in regelmäßigen Zeitabständen über verfügbare Systemdienste an den Angreifer übermittelt oder von diesem manuell abgeholt.

Rootkit

Bei einem Rootkit handelt es sich um eine Sammlung von Dateien und Programmen, die zur Verschleierung von böartigen Systemaktivitäten auf einem Zielsystem reguläre Systemdateien und -routinen ersetzen (vgl. [Nat05], S. 21). Hierbei wird auch die Existenz des Rootkits selbst verborgen. Dies erfolgt zum Beispiel durch den gezielten Austausch von Systemdateien und Kernelmodulen sowie durch eine Modifizierung von Systemaufrufen und Logdateien. Rootkits werden häufig dazu genutzt, weitere Malware auf einem Zielsystem zu installieren und zu verstecken.

2.3 Malware-Analyse

Potentielle Malware sollte zunächst mit herkömmlichen Anti Virus-Programmen untersucht werden. Diese arbeiten in der Regel auf Basis von Virensignaturen, welche regelmäßig von den Software-Herstellern aktualisiert und in großen Datenbanken verwaltet werden. Darüber hinaus erfolgt die Suche nach bekannten Verhaltensmustern mittels heuristischer Verfahren (vgl. [ESE09]). Somit können auch leicht modifizierte Schadprogramme identifiziert und bekannte schädliche Funktionen wiedererkannt werden. Bei neuartiger Malware oder der Verschleierung bekannter Verhaltensmuster reichen diese herkömmlichen Methoden zur Erkennung jedoch nicht mehr aus. Infolgedessen muss zur Bestimmung des Verhaltens der Malware eine Untersuchung mithilfe von Techniken und Werkzeugen zur statischen sowie dynamischen Analyse erfolgen.

2.3.1 Statische Analyse

Während der statischen Analyse erfolgt eine Untersuchung von Code, ohne diesen auszuführen. Hierbei ist für das Vorgehen die vorliegende Form des zu untersuchenden Programms entscheidend. Steht der Quellcode zur Verfügung, kann dieser im Detail analysiert und somit die vollständige Funktionalität der Malware herausgearbeitet werden. Ist die Datei jedoch ausschließlich im Binärcode vorhanden, muss dieser entweder mit einem **Disassembler** in einen für Menschen lesbaren Code übersetzt oder mit verschiedenen Techniken direkt untersucht werden. Da der Schwerpunkt der Bachelorarbeit auf der dynamischen Analyse liegt, werden nachfolgend zur Übersicht lediglich einige wesentliche Techniken zur statischen Analyse beleuchtet.

Im Allgemeinen wird zu Beginn der Analyse zur eindeutigen Identifizierung der Malware ein **Hashing** durchgeführt. Hierzu kommen häufig der Message-Digest Algorithm 5 (MD5) und der Secure Hash Algorithm-1 (SHA-1) zum Einsatz (vgl. [SH12], S. 10). Die generierten Hash-Werte können im Rahmen des Incident Response-Prozesses zur Identifizierung der Malware auf weiteren Systemen eingesetzt und darüber hinaus in öffentlich zugängliche Viren-Datenbanken eingetragen werden.

Neben der Erzeugung von Signaturen auf Basis von Hash-Werten wird typischerweise während der statischen Analyse der zu Grunde liegende Binärcode nach bestehenden ASCII- und Unicode-**Zeichenketten** durchsucht.

Hierbei können unter anderem Informationen über den Autor, die Herkunft der Malware oder die Funktion erlangt werden (siehe hierzu [Rog14], S. 21).

Auch die Ermittlung von **importierten Funktionen** kann weitere Informationen über den Verwendungszweck eines Schadprogramms liefern (vgl. [SH12], S. 15-16). Dies führt jedoch nur zum gewünschten Ergebnis, wenn die von der Malware benötigten Bibliotheken und Funktionen während der Kompilierung dynamisch eingebunden und somit erst während der Ausführung importiert werden. Bei einer statischen Einbindung hingegen ist die Rekonstruktion der Funktionsnamen mit einem in der Regel unverhältnismäßig hohen Aufwand verbunden. Grundsätzlich können die extrahierten Informationen lediglich einen Hinweis darauf geben, welche Funktionen potentiell von der Malware ausgeführt werden könnten. Der Import allein stellt zwar keine Garantie für die tatsächliche Verwendung einer Funktion dar, ist aber zumindest eine gute Indikation.

Darüber hinaus können weitere Herausforderungen die statische Analyse erschweren. Werden von den Malware-Entwicklern zur Verschleierung des Binärcodes sogenannte *Packer*² oder sogar Verschlüsselungsmechanismen eingesetzt, müssen diese vor Beginn der eigentlichen Analyse tiefgehend untersucht und entsprechende Gegenmaßnahmen entwickelt werden. Abhängig von der Komplexität der Schutzmaßnahmen ist die statische Analyse entsprechend zeitaufwendig.

2.3.2 Dynamische Analyse

Im Rahmen der dynamischen Analyse wird eine Malware kontrolliert gestartet und ihre tatsächliche Funktion ermittelt. Hierbei ist grundsätzlich zwischen einer Untersuchung *während* oder unmittelbar *nach* der Ausführung zu unterscheiden (vgl. [SH12], S. 39).

Die Untersuchung nach der Ausführung beinhaltet in der Regel den Einsatz von forensischen Werkzeugen zur Erkennung von Veränderungen innerhalb des zu Grunde liegenden Systems.

² Packer werden in der Softwaretechnik zur Komprimierung von Binärdateien eingesetzt. Während des Kompressionsvorgangs wird das Aussehen einer Binärdatei grundlegend verändert und somit das Lesen des Binärcodes erschwert.

Diese Methode unterliegt allerdings zwei wesentlichen Einschränkungen. Einerseits besteht die Gefahr, temporäre Dateien oder Netzwerkverbindungen, die nur kurzzeitig während der Ausführung der Malware existieren, nicht zu erfassen. Somit würden wesentliche Aktivitäten des Schadprogramms nicht detektiert werden können. Andererseits kann auch die Unterscheidung zwischen einem „normalen“ Systemverhalten und einer Verhaltensanomalie eine erhebliche Herausforderung darstellen. Demnach werden nachfolgend ausschließlich Techniken zur unmittelbaren Analyse *während* der Ausführung beleuchtet.

Zur Laufzeit der potentiellen Malware sind hierzu im Allgemeinen alle **Prozess-**, **Dateisystem-** und **Netzwerkaktivitäten** zu protokollieren und im nächsten Schritt zu analysieren (vgl. [Rog14], S. 12-13). Grundsätzlich existieren eine Vielzahl von Techniken und Methoden zur Beobachtung der verschiedenen Aktivitäten auf einem Linux-System.

Prozessaktivitäten

Die Überwachung von Prozessaktivitäten soll einen umfassenden Einblick in das Verhalten eines Prozesses bzw. einer Malware liefern. Hierbei können unter anderem essentielle Informationen über verwendete Bibliotheken, ausgeführte Funktionen und gestartete Programme erfasst werden. Versucht beispielhaft ein Trojaner einen neuen Benutzer im System anzulegen und zwecks Realisierung einer Backdoor mit entsprechenden Berechtigungen auszustatten, würde es sich hierbei um typische Aktivitäten handeln, die im Rahmen einer Prozessüberwachung zu detektieren sind. Ferner ist hierdurch grundsätzlich auch die Protokollierung von Dateisystem- und Netzwerkzugriffen möglich. Demnach können durch die Überwachung von Prozessaktivitäten verschiedenste Zugriffe und Veränderungen einer Malware innerhalb eines Systems erfasst werden.

Dateisystemaktivitäten

Auf dem Dateisystem befindet sich eine Vielzahl an wichtigen sowie häufig verwendeten Bibliotheken, Programmen und Konfigurationsdateien. Gelingt einem Schadprogramm unbemerkt die Modifizierung oder der Austausch dieser Dateien, kann dies weitreichende Folgen für die Sicherheit aller Programme sowie der Aktivitäten innerhalb des zu Grunde liegenden Systems haben.

Tauscht eine Malware beispielsweise den Secure Shell Client `ssh` im Verzeichnis `/usr/bin/` gegen eine modifizierte Version aus, wie es das SSH-Rootkit „Ebury“ praktiziert, können sensible Daten, wie Benutzernamen und Passwörter bereits vor der Verschlüsselung kopiert und gestohlen werden (vgl. [CER14]). Demnach ist zur Erkennung einer Verletzung der Integrität von Systemdateien und Programmen eine dedizierte Überwachung der Dateisystemaktivitäten essentiell.

Netzwerkaktivitäten

Die Analyse der Netzwerkaktivitäten ist besonders zur Erkennung von weiteren infizierten Systemen innerhalb der Netzwerkinfrastruktur von Bedeutung. Ferner können Verbindungen zu externen Server, die zur Steuerung oder Verbreitung der Malware dienen, identifiziert und im Rahmen der Eindämmungsphase innerhalb des Incident Response-Prozesses (siehe Kapitel 2.1.2) blockiert werden. Beispielhaft wäre somit auch ein Portscan zu erkennen, der von einer Malware zur Identifizierung weiterer Systeme und dedizierter Dienste innerhalb eines internen Netzwerks initiiert wurde.

2.3.3 Sichere Umgebung

Die Ausführung von Malware bringt im Allgemeinen hohe Sicherheitsrisiken für das zu Grunde liegende System und alle weiteren Komponenten im Netzwerk mit sich. Während Prozesse und Daten des Betriebssystems ausgelesen, modifiziert oder gelöscht werden können, besteht auch die Gefahr der vollständigen Zerstörung des Systems sowie der Infizierung weiterer Komponenten innerhalb des Netzwerks. Daher sollte ein Schadprogramm im Rahmen der dynamischen Analyse ausschließlich innerhalb einer sicheren Umgebung, einer Sandbox, untersucht werden. Aus den bisher gesammelten Erfahrungen des Incident Response-Teams der Controlware GmbH ergeben sich hierbei grundlegende Anforderungen, die eine zur Malware-Analyse bestimmte Sandbox-Umgebung zu erfüllen hat:

(1) Die Ausführung der Sandbox sollte nach Möglichkeit auf einem speziellen Testsystem erfolgen, welches sich innerhalb eines **dedizierten Netzwerks** befindet. Somit wird grundsätzlich die Infizierung weiterer produktiver Systeme unterbunden.

(2) Viele Arten von Malware laden zur Laufzeit weiteren Schadcode aus dem Internet herunter oder werden über externe Server gesteuert. Demnach ist zur umfassenden Verhaltensanalyse die Anbindung der Sandbox an das **Internet** essentiell. Hierbei sind jedoch die damit verbundenen Risiken zu beachten. Die Malware könnte zum Beispiel vom Testsystem aus weitere Angriffe auf entfernte Systeme im Internet starten. Somit ist ein Internetzugriff nur eingeschränkt und kontrolliert zu gestatten. Falls generell keine externe Kommunikation erlaubt werden kann, ist ebenso die Simulation von häufig verwendeten Protokollen, wie SSH, DNS, HTTP oder IRC denkbar.

(3) Vor Beginn einer Untersuchung sollte sich die Sandbox in einem definierten Ausgangszustand befinden, der jederzeit durch eine **Wiederherstellung** rekonstruiert werden kann. Wird das System während der Analyse von der Malware beschädigt, ist somit eine schnelle Wiederherstellung und folglich die Fortsetzung der Untersuchung möglich. Ferner ist prinzipiell jedes System, welches zur Ausführung von Malware genutzt wurde, als infiziert zu betrachten und nach Abschluss der Untersuchung in den Ausgangszustand zu versetzen.

(4) Prinzipiell sollte eine hohe logistische und softwareseitige **Flexibilität** der Analyse-Umgebung gegeben sein. Dies ermöglicht einerseits im Rahmen eines Incident Response-Einsatzes eine unmittelbare Untersuchung der Malware am Kundenstandort. Darüber hinaus muss eine zur Ausführung sowie Analyse der Malware notwendige Anpassung der Umgebung effizient umzusetzen sein.

Basierend auf den Anforderungen ist zur Realisierung einer Sandbox beispielsweise der Einsatz einer Virtualisierungslösung wie *VMware* oder einer Emulationslösung wie *QEMU* zu empfehlen. Ein Emulator realisiert die Funktionalität zur Simulation einzelner Programme oder spezifischer Hardware, wie einer CPU, ausschließlich in Software. Im Zuge der Virtualisierung wird hingegen über eine Zwischeninstanz, dem *Hypervisor*³, ein kontrollierter Zugriff auf die Hardware eines Systems ermöglicht (vgl. [Kru14], S. 2).

³ Bei dem Hypervisor handelt es sich um das Kernstück der Virtualisierung. Dieser emuliert die physikalische Hardware eines zu Grunde liegenden Systems und übernimmt die Verwaltung sowie die Verteilung der zur Verfügung stehenden Ressourcen an die virtuellen Maschinen.

Mithilfe beider Lösungen lässt sich die Simulation eines vollständigen Betriebssystems realisieren, welches von dem Host-System isoliert ist und demnach als Sandbox zur Ausführung von Malware eingesetzt werden kann. Befindet sich die Sandbox auf einem ausschließlich zur Malware-Analyse eingesetzten Testsystem, kann die vollständige Testumgebung während eines Incident Response-Einsatzes flexibel transportiert (4) und leicht in einem dedizierten Netzwerk gestartet werden (1). Ferner kann der Internetzugriff über das separierte Netzwerk durch den Host beschränkt und kontrolliert werden (2). Zuletzt erlauben beide Lösungen die Verwendung von Speicherabbildern zur Wiederherstellung von definierten Systemzuständen der Sandbox (3).

Im Kapitel „Grundlagen“ wurde zu Beginn der vom NIST bereitgestellte allgemeine Incident Response-Prozess zur Behandlung von Sicherheitsvorfällen dargestellt. Dieser umfasst die vier Phasen „Preparation“, „Detection & Analysis“, „Containment, Eradication & Recovery“ und „Post-Incident Activity“. Innerhalb der zweiten Phase wird im Allgemeinen die während eines Sicherheitsvorfalls eingesetzte Malware analysiert.

Darauf aufbauend erfolgten eine Definition sowie eine Klassifizierung von Malware. Grundsätzlich existiert eine Vielzahl verschiedener Schadprogramme, die hinsichtlich ihrer Funktionalität und ihres Einsatzzwecks variieren können.

Abgerundet wurden die Grundlagen durch eine Einführung in die Malware-Analyse. Hierbei erfolgte eine Unterscheidung zwischen der statischen Analyse zur Untersuchung des zu Grunde liegenden Codes eines Schadprogramms und der dynamischen Analyse, also der kontrollierten Ausführung und dedizierten Beobachtung des Verhaltens einer Malware.

3 Techniken

Die dynamische Analyse von Linux-Malware kann im Allgemeinen mit einer Vielzahl von verschiedenen Methoden und Techniken erfolgen. Wie in Kapitel 2.3.2 „Dynamische Analyse“ bereits thematisiert, werden nachfolgend ausschließlich Techniken zur Erfassung von Prozess-, Dateisystem- und Netzwerkaktivitäten *während* der unmittelbaren Ausführung einer Malware erarbeitet.

3.1 Prozessaktivitäten

Moderne Betriebssysteme verfügen über mehrere Privilegierungsstufen, in denen ein Prozess ausgeführt werden kann. Der Linux-Kernel läuft in einem privilegierten *Kernel-Modus* und verfügt über einen unbeschränkten Zugriff auf alle Systemressourcen sowie den kompletten Befehlssatz der CPU. Die Ausführung herkömmlicher Anwendungsprogramme erfolgt hingegen aus Sicherheitsgründen in einem unprivilegierten *Benutzer-Modus* (vgl. [Tan09], S. 30-31). Benötigt ein Anwendungsprogramm den Zugriff auf Systemressourcen wie den Hauptspeicher oder das Dateisystem, muss dies über einen **Systemaufruf** erfolgen. Infolge des Aufrufs wird eine vom Betriebssystem bereitgestellte Funktion zunächst durch den Kernel ausgeführt und das Ergebnis dann an den aufrufenden Prozess des Anwendungsprogramms zurückgegeben. Somit erhält ein unprivilegierter Prozess zu keinem Zeitpunkt einen privilegierten Zugriff auf das System, sondern nur auf das Ergebnis. Demnach erfolgen alle privilegierten und daher für die Malware-Analyse entscheidenden Aktivitäten, die ein Schadprogramm zur Manipulation oder Beschädigung eines zu Grunde liegenden Systems ausführen muss, ausschließlich über den Kernel. Die gezielte Überwachung der Systemaufrufe ermöglicht somit bereits die Erfassung des konkreten Verhaltens einer Malware, wie zum Beispiel die Erstellung weiterer Prozesse oder den Zugriff auf den Hauptspeicher.

Prinzipiell wären auch weitere Techniken, wie die Beobachtung aller von einer Malware ausgeführten **Assembler-Instruktionen** denkbar. Dies könnte zum Beispiel mithilfe eines Debuggers realisiert werden, indem der vollständige Programmablauf einer Malware zur Laufzeit Schritt für Schritt verfolgt wird. Dieser Ansatz ist jedoch in der Regel aufgrund der großen Menge des zu analysierenden Codes sehr zeitintensiv.

Darüber hinaus wird zur Auswertung der erfassten Instruktionen ein tiefgehendes Verständnis der Assembler-Sprache benötigt, die je nach zu Grunde liegendem Prozessor und Betriebssystem stark in der Implementierung variieren kann. Demnach ist die Verwendung dieser Technik im Rahmen eines Incident Response-Einsatzes nicht praktikabel.

Folglich werden innerhalb dieses Kapitels ausschließlich Techniken zur Überwachung von Systemaufrufen auf Linux-Systemen tiefgehend betrachtet. Hierzu konnten im Zuge der Untersuchung zwei verschiedene Ansätze herausgearbeitet werden. Ersterer basiert auf der Verwendung des Kernel-Moduls „Process Events Connector“ (*proc connector*), während im zweiten Ansatz der Systemaufruf „Process Trace“ (*ptrace*) eingesetzt wird.

3.1.1 Kernel-Modul „Process Events Connector“

Der Linux-Kernel beinhaltet das Modul *proc connector*, welches zur unmittelbaren Benachrichtigung eines Prozesses über ausgeführte Systemaufrufe einer zu untersuchenden Malware eingesetzt werden kann. Hierbei sind jedoch zwei grundlegende Einschränkungen im Rahmen einer Malware-Analyse zu berücksichtigen. Das Kernel-Modul ermöglicht zum einen keine kontrollierte Ausführung des überwachten Prozesses, sondern kann lediglich über durchgeführte Aktivitäten benachrichtigen. Darüber hinaus können mithilfe des Moduls bisher ausschließlich die zur Ausführung neuer Prozesse verwendeten Systemaufrufe `fork` und `exec` sowie `setuid`, `setgid` und `setsid`, die zur Modifizierung der zugehörigen IDs eines Prozesses dienen, protokolliert werden (vgl. [Rem11]). Trotz dieser Einschränkungen handelt es sich bei diesem Ansatz um eine vielversprechende Technik, die im Rahmen der Bachelorarbeit zumindest theoretisch betrachtet und daher in Abbildung 2 zur Übersicht dargestellt sowie nachfolgend näher erläutert wird:

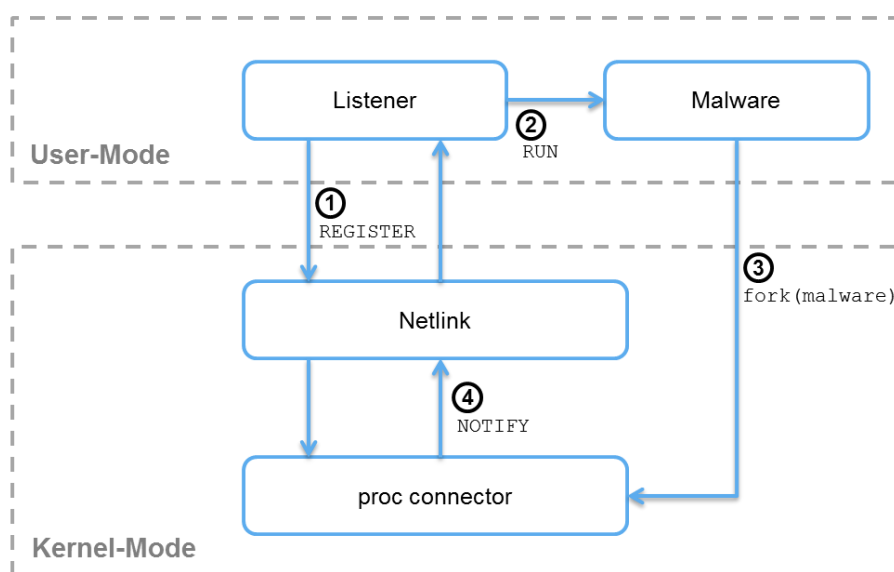


Abbildung 2: Prozessüberwachung mithilfe von *proc connector*

(1) Im ersten Schritt muss sich ein zur Überwachung etablierter Prozess (*Listener*) bei *proc connector* registrieren. Die Kommunikation erfolgt hierbei über die vom Kernel bereitgestellte Schnittstelle *Netlink*, welche ursprünglich zum Austausch von Netzwerk-Konfigurationsdaten zwischen im Benutzer-Modus laufenden Anwendungen und dem Kernel entwickelt wurde (vgl. [KQ08]). Darüber hinaus hat sich *Netlink* zu einer essentiellen Schnittstelle zur Interprozess-Kommunikation etabliert, die einen grundlegenden Datenaustausch zwischen einem Anwendungsprogramm und dem Kernel ermöglicht.

(2) Nach Abschluss der Registrierung ist der *Listener* in der Lage, durch den Kernel Benachrichtigungen über Systemaufrufe in Form von Events zu erhalten. Anschließend kann die *Malware* gestartet und auf die Ausführung der überwachten Aufrufe gewartet werden.

(3) Der im Benutzer-Modus laufende *Malware*-Prozess kann beispielsweise mithilfe des Systemaufrufs `fork` einen weiteren Unterprozess starten. Da der Kernel generell zur Ausführung aller Systemaufrufe eingesetzt wird, kann im Zuge dessen eine effiziente Aufzeichnung der relevanten Daten erfolgen, die wiederum an *proc connector* weitergeleitet werden.

(4) Falls der entsprechende Systemaufruf im Kernel-Modul implementiert ist, sendet dieses ein Event an den zur Überwachung registrierten *Listener* (gekennzeichnet durch ein `NOTIFY` in Abbildung 2).

Der Kernel erzeugt jedoch prinzipiell Benachrichtigungen über die Aktivitäten aller laufenden Prozesse, weshalb die Malware Prozess-ID explizit zur Filterung der relevanten Aktivitäten heranzuziehen ist.

Wie bereits angedeutet, ermöglicht das Kernel-Modul bisher ausschließlich eine Benachrichtigung bei der Ausführung von fünf der ungefähr 390 derzeit verfügbaren Systemaufrufe innerhalb eines Linux-Systems (vgl. [Ker15c]). Eine Verwendung von *proc connector* wird daher aktuell nur in Kombination mit weiteren Techniken empfohlen. Dennoch handelt es sich um einen sehr effektiven Ansatz, dessen Potential durch eine Erweiterung des Kernel-Moduls gesteigert werden kann.

3.1.2 Systemaufruf „Process Trace“

Bei dem Systemaufruf *ptrace* handelt es sich um eine in Linux eingebaute Technik, die einem Prozess die dedizierte Beobachtung und kontrollierte Ausführung eines anderen Prozesses ermöglicht (vgl. [Ker15a]). Im Gegensatz zur eingangs dargestellten Technik wird hierbei dem überwachten Prozess ein vollständiger lesender sowie schreibender Zugriff auf den zugewiesenen Speicher des anderen Prozesses und den zur Ausführung von Instruktionen verwendeten Registern der CPU gestattet.

Die Überwachung kann grundsätzlich auf zwei Arten realisiert werden. Der erste Ansatz besteht darin, sich zur Beobachtung mittels *ptrace* an einen bereits in der Ausführung befindlichen Prozess anzuhängen. In der zweiten Variante wird hingegen ein zu analysierendes Programm explizit zur Untersuchung gestartet und somit von Beginn an überwacht.

Im Zuge einer Malware-Analyse gilt es eine möglichst vollständige Einsicht über die getätigten Systemaufrufe zu bekommen. Aus diesem Grund sollte zur Untersuchung einer Malware ausschließlich der zweite Ansatz genutzt werden. Der entsprechende Ablauf ist in Abbildung 3 im Überblick dargestellt und wird nachfolgend im Detail beschrieben:

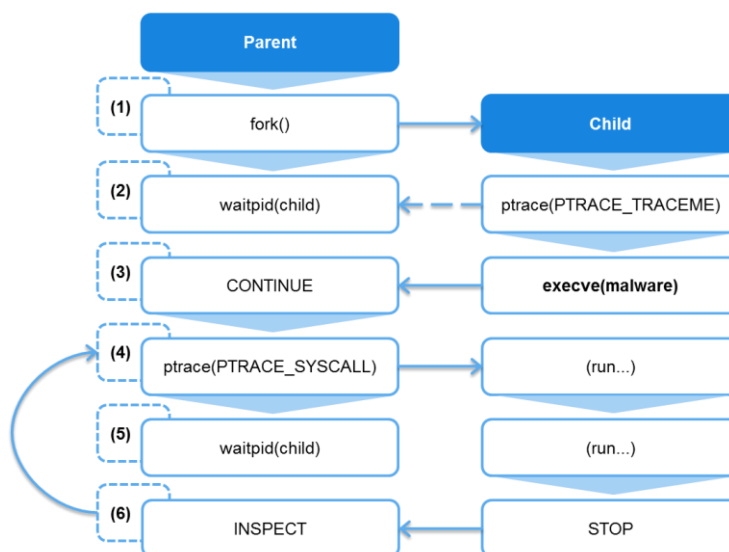


Abbildung 3: Prozessüberwachung unter Verwendung von ptrace

(1) Zu Beginn der Untersuchung erzeugt ein zur Überwachung etablierter und im Benutzer-Modus laufender Elternprozess (*Parent*) einen Kindprozess (*Child*), der zur kontrollierten Ausführung und dedizierten Beobachtung einer Malware verwendet wird. Die Erzeugung erfolgt unter Verwendung des Systemaufrufs `fork`. Hierdurch wird eine exakte Kopie von *Parent* erstellt und ausgeführt.

(2) Nachdem die Erzeugung abgeschlossen ist, wird *Parent* durch Verwendung der Funktion `waitpid` bis zur Ausführung der Malware angehalten (vgl. [Ker15b]). Unterdessen initialisiert *Child* die Beobachtung, indem mithilfe des Systemaufrufs `ptrace` und dem Parameter `PTRACE_TRACEME` eine Anfrage zur Verfolgung an *Parent* gestellt wird (in Abbildung 3 durch einen gestrichelten Pfeil angedeutet).

(3) Anschließend wird die Malware durch Aufruf von `execve` in den Speicherbereich von *Child* geladen und dort gestartet. Unmittelbar nachdem das Schadprogramm initialisiert wurde, erfolgt bereits eine Unterbrechung durch den Linux-Kernel. Im Zuge dieser Zustandsänderung wird die Ausführung von *Child* angehalten und von *Parent* fortgesetzt (gekennzeichnet durch ein `CONTINUE` in Abbildung 3).

(4) Der Kernel erhält mithilfe des Aufrufs `ptrace` und des Parameters `PTRACE_SYSCALL` die Anweisung, die Ausführung von *Child*, also des Malware-Prozesses, bis zum nächsten Systemaufruf fortzusetzen.

(5) *Parent* wartet in der Zwischenzeit unter Verwendung von `waitpid` auf eine erneute Unterbrechung des Malware-Prozesses durch den Kernel. Diese wird durch die Ausführung bzw. die Rückkehr eines Systemaufrufs ausgelöst.

(6) Sobald der Kernel eine Anfrage für einen Systemaufruf empfangen hat, erfolgt im Allgemeinen die Ausführung der zur Bearbeitung hinterlegten Kernel-Funktion. Da jedoch eine Überwachung mittels `ptrace` erfolgt, springt der Kernel stattdessen in die von ihm bereitgestellte Funktion `tracesys`. Diese stoppt zunächst den Malware-Prozess und informiert *Parent* über den durchzuführenden Systemaufruf. Nach Abschluss der Ausführung wird erneut angehalten, um den Rückgabewert des Aufrufs zu übermitteln (vgl. [Sch08]). Hierbei hat *Parent* grundsätzlich die Möglichkeit, den Systemaufruf sowie den Rückgabewert zu modifizieren und somit eine mögliche Detektion der Überwachung zu verhindern (siehe hierzu Kapitel 4.2.1). Dies ist in Abbildung 3 durch ein `INSPECT` gekennzeichnet. Bis zum Abschluss der Untersuchung erfolgt die wiederholte Ausführung des Aufrufs `ptrace` unter Verwendung des Parameters `PTRACE_SYSCALL` und somit ein Sprung zu Schritt 4.

Werden von einer zu analysierenden Malware beispielsweise durch die erneute Benutzung von `fork` weitere Prozesse zur Laufzeit gestartet, gilt es diese ebenso zu überwachen. Hierzu wurde während der Implementierung (siehe Kapitel 4.2.1) mithilfe des Aufrufs `ptrace` zu Beginn der Überwachung die Option `PTRACE_O_TRACEFORK` gesetzt (vgl. [Sta11a]). Infolgedessen werden alle erstellten Malware-Prozesse automatisch und parallelisiert überwacht (vgl. [Ker15a]).

3.2 Dateisystemaktivitäten

Ein im Benutzer-Modus laufendes Anwendungsprogramm, wie zum Beispiel eine Malware, kann ausschließlich mithilfe von Systemaufrufen auf das Dateisystem zugreifen. Demnach können grundsätzlich auch die Dateisystemaktivitäten einer zu untersuchenden Malware durch eine dedizierte und wie in Kapitel 3.1 beschriebene Beobachtung der entsprechenden Systemaufrufe überwacht werden. Eine weitere Variante besteht in der Verwendung des Kernel-Moduls „Inode Notify“ (*inotify*), welches zur Protokollierung von Veränderungen an Dateien und Verzeichnissen auf der Kernel-Ebene implementiert ist. Innerhalb dieses Kapitels werden beide Ansätze betrachtet.

3.2.1 Systemaufruf „Process Trace“

Die Protokollierung der für einen Dateisystemzugriff benötigten Systemaufrufe kann analog zur Erfassung der Prozessaktivitäten unter Verwendung von *ptrace* erfolgen. Hierzu sollte grundsätzlich der in Kapitel 3.1.2 herausgearbeitete Ansatz eingesetzt werden, der eine Malware explizit zur Untersuchung startet und somit von Beginn an eine Überwachung ermöglicht.

Aufgrund der Vielzahl von Systemaufrufen, die ein Programm während der Laufzeit ausführt, entsteht jedoch grundsätzlich eine große Datenmenge, die es zur Identifizierung von relevanten Dateisystemzugriffen entsprechend zu filtern gilt. Zur Veranschaulichung dieser Problematik wird mithilfe des innerhalb von Linux etablierten Programms *strace* die Ausführung der beispielhaft entwickelten Datei „create_file“ beobachtet.

Das Programm versucht während der Ausführung lediglich eine leere Datei „new_file.dat“ im aktuellen Arbeitsverzeichnis zu öffnen und erzeugt diese, falls sie noch nicht existiert. Der vollständige Quellcode kann in Anhang A.1 eingesehen werden. Zur übersichtlicheren Darstellung wurde durch Ergänzung des Parameters `-e trace=file` eine Filterung aller Systemaufrufe vorgenommen, die keinen Dateinamen enthalten:

```
$ strace -e trace=file ./create_file

execve("./create_file", ["/create_file"], [/* 60 vars */]) = 0

access("/etc/ld.so.nohwcap", F_OK) = -1
```

```
access("/etc/ld.so.preload", R_OK) = -1  
  
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3  
  
access("/etc/ld.so.nohwcap", F_OK) = -1  
  
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3  
  
open("new_file.dat", O_RDWR) = -1  
  
open("new_file.dat", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
```

Abbildung 4: Überwachung von Dateisystemzugriffen mittels *strace*

Wie in Abbildung 4 aufgezeigt, werden trotz einer ersten Filterung der Systemaufrufe für die Erzeugung einer einzigen Datei bereits sieben Dateisystemzugriffe von *strace* protokolliert, wobei auch die zur Programm-Initialisierung getätigten lesenden Zugriffe auf Systembibliotheken erfasst wurden.

Mit dem Umfang an Funktionen steigt zusätzlich die Datenmenge, die es im Zuge der dynamischen Analyse zu filtern sowie zu bewerten gilt. Auch können verschiedene Systemaufrufe zur Realisierung eines identischen Dateisystemzugriffs verwendet werden. Ein Schreibvorgang kann beispielsweise mit einem `write` sowie mit einem `mmap` erfolgen (vgl. [Stallb]). Die Vielseitigkeit dieser sowie weiterer Systemaufrufe erschwert zusätzlich die Erfassung und Auswertung von Dateisystemaktivitäten. Aufgrund dieser beiden Herausforderungen ist die Suche nach den im Zuge eines Dateisystemzugriffs ausgeführten Systemaufrufen, den Auslösern, im Allgemeinen sehr komplex und zeitintensiv. Nach Möglichkeit sollte daher zur Erfassung der Zugriffe auf das Dateisystem stets eine Überwachung der Ergebnisse erfolgen, die aus den Systemaufrufen resultieren. Dies umfasst beispielsweise ein neu angelegtes Verzeichnis oder eine modifizierte Datei.

3.2.2 Kernel-Modul „Inode Notify“

Bei *inotify* erfolgt die Überwachung einzelner Dateien oder vollständiger Verzeichnisse indem der Kernel die entsprechenden Aktivitäten event-basiert an ein zuvor registriertes Anwendungsprogramm überträgt. Der Ablauf, welcher in Abbildung 5 dargestellt ist, wird nachfolgend schematisch beschrieben:

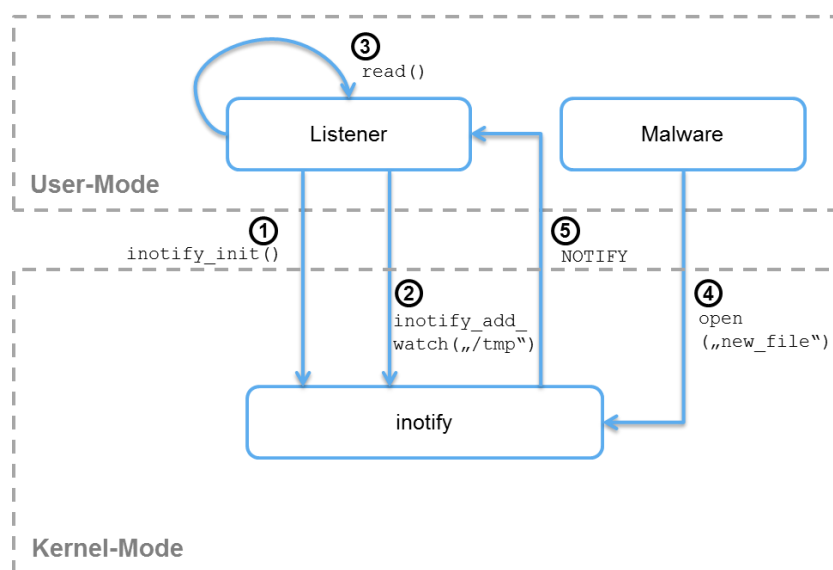


Abbildung 5: Dateisystemüberwachung mithilfe von *inotify*

(1) Zu Beginn muss der beobachtende Prozess (*Listener*) mit der Funktion `inotify_init` eine neue *inotify*-Instanz erzeugen (vgl. [Ker14a]). Diese wird während der Analyse einer *Malware* zur Kommunikation mit dem Kernel-Modul verwendet.

(2) Die Spezifizierung der zu überwachenden Dateien und Verzeichnisse erfolgt unter Verwendung der Funktion `inotify_add_watch`. Ferner sind hier auch die Events wie `ON_ACCESS`, `ON_OPEN`, `ON_WRITING` oder `ON_CLOSE` festzulegen, über die der Kernel den Prozess zukünftig benachrichtigen soll. Eine vollständige Liste der verfügbaren Events kann beispielsweise im „Linux Programmer's Manual“ eingesehen werden (vgl. [Ker14a]).

(3) Unter Verwendung der Funktion `read` wartet der *Listener* auf Event-Benachrichtigungen durch den Kernel. Hierzu wird die Ausführung gestoppt und erst bei Erhalt einer Nachricht wieder fortgesetzt. Falls mehrere Verzeichnisse oder Dateien zu überwachen sind, sollte daher bevorzugt ein `select` zur selektiven Behandlung von mehreren Events eingesetzt werden.

(4) Führt die *Malware* einen Dateisystemzugriff zur Erstellung einer neuen Datei aus, wird diese Aktivität zwangsläufig durch den Kernel ausgeführt. Somit wird auch das Kernel-Modul *inotify* über diesen Vorgang benachrichtigt. Innerhalb von Abbildung 5 wurde hierzu beispielhaft der Aufruf `open` zur Erstellung der Datei `new_file` verwendet.

(5) Falls der *Listener* zuvor das entsprechende Verzeichnis zur Überwachung registriert hat, erfolgt die Benachrichtigung über die neu erstellte Datei unmittelbar durch die Meldung des eingetretenen Events (gekennzeichnet durch ein `NOTIFY` in Abbildung 5).

Im letzten Schritt wird ein wesentlicher mit dieser Technik einhergehender Nachteil ersichtlich. Die zu überwachenden Verzeichnisse und Dateien sind explizit bei dem Kernel-Modul zu spezifizieren. Die Verwendung des root-Verzeichnisses (`/`) ist an dieser Stelle nicht möglich, da eine Überwachung aller Pfade des Dateisystems das zu Grunde liegende System nahezu vollständig auslasten würde. Zur Erfassung aller kritischen Dateisystemzugriffe einer Malware müsste daher eine Vielzahl von Dateien und Verzeichnissen im Vorfeld manuell herausgearbeitet sowie unmittelbar vor jeder Untersuchung registriert werden. Darüber hinaus können Zugriffe auf das Dateisystem, die mittels `mmap`, `msync` und `munmap` erfolgen, laut der Dokumentation des Moduls bisher nicht erfasst und in Folge dessen nicht gemeldet werden (vgl. [Ker14a]). Idealerweise sollte also zur umfassenden und effizienten Erfassung der Dateisystemaktivitäten einer Malware während der dynamischen Analyse eine Kombination aus beiden erarbeiteten Techniken erfolgen.

3.3 Netzwerkaktivitäten

Der Netzwerkzugriff eines im Benutzer-Modus ausgeführten Programms erfolgt, wie auch ein Zugriff auf das Dateisystem, ausschließlich über Systemaufrufe. Somit kann zur Beobachtung der Netzwerkaktivitäten einer Malware ebenfalls *ptrace* eingesetzt werden. Ein weiterer Ansatz besteht in der Überwachung eines bzw. mehrerer Netzwerkinterfaces. Hierbei wird unter Einsatz des Kernel-Moduls „Packet Capture“ (*pcap*) eine vollständige Erfassung sowie tiefgehende Untersuchung des Netzwerkverkehrs ermöglicht.

3.3.1 Systemaufruf „Process Trace“

Der grundlegende Ablauf zur Überwachung von Systemaufrufen mithilfe von *ptrace* wurde bereits erarbeitet und kann aus Kapitel 3.1.2 sowie der darin enthaltenen Abbildung 3 entnommen werden. Mit dieser Technik können im Allgemeinen die grundlegenden Informationen über eingehende sowie ausgehende Verbindungen effizient erfasst und ausgewertet werden.

Dies wird nachfolgend in Abbildung 6 unter der erneuten Verwendung des Programms *strace* zur Ausführung eines echten IRC-Bots demonstriert, der während eines Incident Response-Einsatzes sichergestellt und im Zuge dessen bereits umfassend analysiert wurde. Zur übersichtlichen Darstellung wurden alle Systemaufrufe, die keine Netzwerkfunktionen abbilden, mit dem Parameter `-e trace=network` gefiltert.

```
$ strace -e trace=network ./irc_bot

socket(PF_INET, SOCK_STREAM, IPPROTO_TCP) = 3

socket(PF_LOCAL, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, 0) = 4

connect(4, {sa_family=AF_LOCAL, sun_path="[...] "}, 110) = -1

socket(PF_LOCAL, SOCK_STREAM|SOCK_CLOEXEC|SOCK_NONBLOCK, 0) = 4

connect(4, {sa_family=AF_LOCAL, sun_path="[...]"}, 110) = -1

socket(PF_INET, SOCK_DGRAM|SOCK_NONBLOCK, IPPROTO_IP) = 4

connect(3, {sa_family=AF_INET, sin_port=htons(6667),
          sin_addr=inet_addr("83.140.172.210")}, 16) = 0

recvfrom(3, "NOTICE AUTH :*** Looking up your"..., 4096, 0,
          NULL, NULL) = 150

recvfrom(3, "PING :266964139\r\n", 4096, 0, NULL, NULL) = 17

recvfrom(3, ":port80a.se.quakenet.org 001 GUJ"..., 4096, 0,
          NULL, NULL) = 4096
```

Abbildung 6: Überwachung von Netzwerkzugriffen mittels *strace*

Der zu Grunde liegende IRC-Bot etabliert eine Verbindung zu einem entfernten Server. Dieser ist, wie in der Abbildung hervorgehoben, über die IP-Adresse 83.140.172.210 und dem häufig von IRC-Servern verwendeten Port 6667 zu erreichen. Ferner erfolgt ein Austausch von Nachrichten zwischen dem Client und dem Server, die ebenfalls hervorgehoben wurden. Die durch den Client empfangene Zeichenkette „NOTICE AUTH...“ lässt demnach auf eine Kommunikation mit einem IRC-Server schließen, der typischerweise unmittelbar nach dem Verbindungsaufbau eine Authentifizierung des Clients durchführt.

Der Einsatz von *ptrace* erlaubt somit eine effiziente Erfassung grundlegender Netzwerkaktivitäten. Werden jedoch komplexere Datenstrukturen oder umfangreiche Protokolle zur Übermittlung von Nachrichten verwendet, ist eine Auswertung der ausgetauschten Informationen in der Regel nicht mehr ohne Einsatz von weiteren Techniken und Werkzeugen möglich.

3.3.2 Kernel-Modul „Packet Capture“

Bei dem Modul *pcap* handelt es sich um eine im Linux-Kernel etablierte Funktion, die das Mitschneiden des Netzwerkverkehrs eines oder mehrerer Netzwerkinterfaces ermöglicht (vgl. [Ker14b]). Die Überwachung eines Interfaces gestattet einen umfassenden Einblick in die von der Malware zur Kommunikation verwendeten Protokolle und Datenstrukturen. Hierbei entsteht in der Regel aufgrund der globalen Überwachung eines Interfaces und der damit einhergehenden Protokollierung der Netzwerkaktivitäten der aktiven Programme eine bedeutsame Datenmenge. Diese kann jedoch unmittelbar nach Abschluss der Erfassung durch den Einsatz von Filtern, die beispielsweise zur Festlegung der zu beobachtenden Protokolle oder IP-Adressen einzusetzen sind, reduziert werden. Der Ablauf zur Überwachung eines Netzwerkinterfaces ist im Überblick in Abbildung 7 dargestellt und wird nachfolgend im Detail beschrieben:

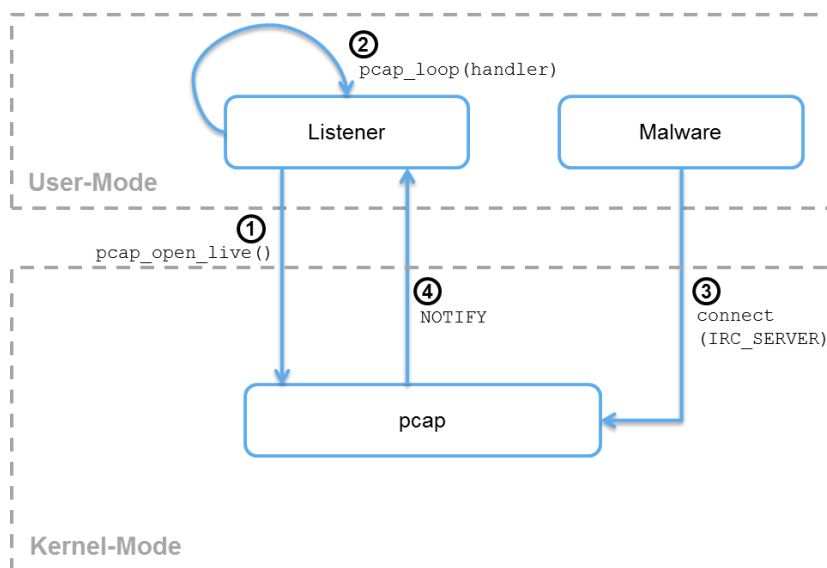


Abbildung 7: Netzwerküberwachung unter Verwendung von *pcap*

(1) Der überwachende Prozess (*Listener*) initialisiert die Protokollierung des Netzwerkverkehrs eines Netzwerkinterfaces, beispielsweise `eth0`, unter Verwendung der Funktion `pcap_open_live`. Diese wird von der *pcap*-Bibliothek bereitgestellt und kann von jedem im Benutzer-Modus laufenden Programm verwendet werden.

(2) Nach Abschluss der grundlegenden Konfiguration wird die Überwachung durch Ausführung der Funktion `pcap_loop` gestartet. Hierbei wird der *Listener* bis zum Empfang eines Pakets angehalten. Ferner wird eine Funktion `handler`, die zur Verarbeitung der mitgeschnittenen Pakete vorgesehen ist, als Parameter übergeben.

(3) Die Netzwerkkommunikation wird grundsätzlich durch den Linux-Kernel realisiert. Sobald eine *Malware* Pakete versendet oder empfängt, kann demnach das Kernel-Modul *pcap* die entsprechenden Daten mitschneiden und eine Übertragung an den überwachenden Prozess durchführen.

(4) Bei Erhalt eines Pakets führt `pcap_loop` die entsprechende `handler`-Funktion zur weiteren Verarbeitung der empfangen Daten aus. Der Inhalt der Funktion ist somit entsprechend vor Beginn der Überwachung zu definieren. Unterdessen wartet `pcap_loop` bereits auf den Empfang weiterer Pakete.

Wird, wie in Kapitel 2.3.3 beschrieben, eine Sandbox als sichere Umgebung zur Untersuchung einer Malware eingesetzt, besteht die Möglichkeit, ein Netzwerkinterface entweder innerhalb oder außerhalb der Sandbox zu überwachen. Bei einer internen Erfassung der Netzwerkaktivitäten über das *pcap*-Modul besteht theoretisch die Gefahr einer Manipulation der mitgeschnittenen Pakete. Hierzu müsste eine mit `root`-Rechten privilegierte Malware lediglich selbst die von dem Kernel bereitgestellte Schnittstelle verwenden. Die externe Überwachung über den Host der Sandbox verhindert hingegen eine Manipulation des Netzwerkverkehrs durch die Malware. Ferner wird hierbei eine vollständige Protokollierung aller versendeten Pakete des Schadprogramms ermöglicht.

Während mithilfe dieser Technik ein umfassender Mitschnitt von versendeten und empfangen Paketen einer Malware realisiert werden kann, gibt es dennoch Netzwerkaktivitäten, die hiermit nicht zu erfassen sind.

Öffnet beispielsweise ein Trojaner zur Etablierung einer Backdoor einen Port, ohne während der dynamischen Analyse darüber Daten zu empfangen oder zu versenden, würde diese Aktivität unter ausschließlicher Verwendung von *pcap* nicht identifiziert werden können. Eine Überwachung der netzwerk-spezifischen Systemaufrufe würde dies hingegen ermöglichen. Demnach sollte idealerweise zur dynamischen Analyse eine Kombination beider vorgestellten Techniken erfolgen.

Innerhalb des Kapitels „Techniken“ wurden verschiedene Ansätze zur dynamischen Analyse von Linux-Malware herausgearbeitet. Hierbei lag der Fokus auf Techniken, die grundsätzlich zur Erfassung von Prozess-, Dateisystem-, und Netzwerkaktivitäten *während* der Ausführung einer Malware eingesetzt werden können.

Als besonders vielversprechend hat sich die Verwendung des Systemaufrufs „Process Trace“ erwiesen. Dieser erlaubt neben der kontrollierten Ausführung und dedizierten Überwachung von Prozessaktivitäten auch die Erfassung von Dateisystem- und Netzwerkzugriffen. Die Überwachung wird hierbei über eine ausschließliche Beobachtung der Systemaufrufe einer zu Grunde liegenden Malware realisiert. Zusätzlich wurden verschiedene Kernel-Module betrachtet, die aufgrund ihrer strategisch guten Position im Kernel ebenfalls zur effizienten dynamischen Analyse von Linux-Malware herangezogen werden können.

4 Realisierung

Die Realisierung der dynamischen Analyse von Linux-Malware erfolgt unter Verwendung einer renommierten Malware Analyse-Umgebung, der Cuckoo Sandbox. Diese ermöglicht bereits eine statische und dynamische Untersuchung von Schadprogrammen für Windows-Systeme und wurde zudem im Rahmen des vorangegangenen Praxisprojekts um Funktionen zur statischen Analyse von Linux-Malware erweitert (vgl. [Rog14]). In diesem Kapitel erfolgt einleitend eine grundlegende Beschreibung der Cuckoo Sandbox. Darauf aufbauend wird im zweiten Abschnitt die Implementierung der zur dynamischen Analyse von Linux-Malware entwickelten Funktionen des Prototyps vorgestellt und tiefgehend betrachtet.

4.1 Cuckoo Sandbox

Die Cuckoo Sandbox ist eine quelloffene Software, die zur automatisierten Analyse des Verhaltens von unbekanntem sowie unmittelbar verdächtigen Dateien und Programmen eingesetzt wird (vgl. [Cuc14a]). Sie unterliegt der GNU General Public License (Version 3) und kann demnach frei verwendet, modifiziert und verbreitet werden (vgl. [Cuc14c]). Nachfolgend wird das allgemeine Konzept einer Analyse-Umgebung unter Berücksichtigung der Cuckoo Sandbox im Überblick dargestellt und darüber hinaus die grundlegende Funktionsweise dieser Lösung im Detail beschrieben.

4.1.1 Konzept

Das Konzept der Analyse-Umgebung umfasst im Allgemeinen drei wesentliche Bausteine, die zur effizienten und sicheren Untersuchung von Malware essentiell sind (vgl. [Rog14], S. 13-17). Der Zusammenhang dieser drei Bausteine wird in Abbildung 8 zur Übersicht dargestellt und nachfolgend näher betrachtet:

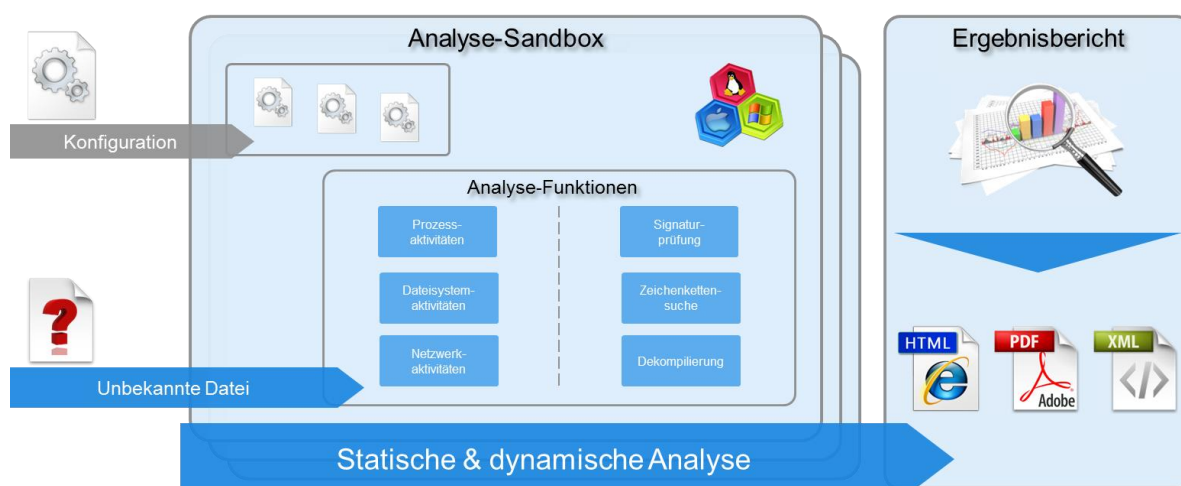


Abbildung 8: Ablauf der Malware-Untersuchung innerhalb einer Analyse-Sandbox

Analyse-Sandbox

Die Analyse-Sandbox stellt den Kern der Umgebung zur sicheren Ausführung und Untersuchung von Malware dar. Die Cuckoo Sandbox setzt hierbei eine virtuelle Maschine ein (siehe Kapitel 2.3.3 „Sichere Umgebung“), welche zur Untersuchung des Verhaltens von unbekanntem und unmittelbar verdächtigen Dateien verwendet wird. Diese verfügt über eine grundlegende Konfiguration, die bei Bedarf zur Analyse einer spezifischen Malware, beispielsweise durch die Installation zusätzlicher Programme und Dienste, angepasst werden kann. Eine Analyse-Umgebung sollte möglichst mehrere Analyse-Sandboxen beinhalten, die eine Vielzahl von unterschiedlichen Konfigurationen und Betriebssystemen bereitstellt.

Analyse-Funktionen

Wird eine potentielle Malware in die Analyse-Sandbox übertragen, erfolgt mithilfe von verschiedenen Analyse-Funktionen eine tiefgehende Untersuchung der Datei. Die Funktionen umfassen verschiedene Techniken zur statischen und dynamischen Analyse eines Schadprogramms, welche bereits in Kapitel 2.3 sowie in Kapitel 3 erarbeitet wurden. In dem vorangegangenen Praxisprojekt erfolgte bereits auf Grundlage der Cuckoo Sandbox die Implementierung verschiedener Funktionen zur statischen Analyse von Linux-Malware in einen Prototyp. Der Fokus dieser Ausarbeitung liegt daher auf der Erweiterung des Prototyps um Funktionen zur dynamischen Analyse von Malware für Linux-Systeme.

Ergebnisbericht

Nach Abschluss des Analyse-Prozesses wird unter Verwendung der gesammelten Daten automatisiert ein Ergebnisbericht erzeugt. Dieser soll das Verhalten der Malware sowie ungewöhnliche Systemaktivitäten während der Ausführung aufzeigen. Ferner sollen die gesammelten Informationen beispielsweise für ein Incident Response-Team in aufbereiteter und strukturierter Form zur Verfügung gestellt werden.

4.1.2 Funktionsweise

Zur Veranschaulichung der Funktionsweise wird der Ablauf eines vollständigen Analyse-Prozesses der Cuckoo Sandbox vereinfacht in Abbildung 9 dargestellt und nachfolgend beschrieben:

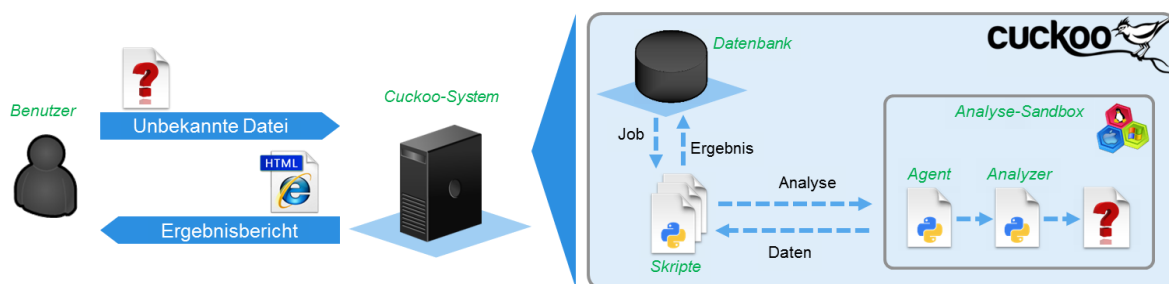


Abbildung 9: Übersicht über die Funktionsweise der Cuckoo Sandbox

Cuckoo-System

Zunächst wird eine unbekannte Datei über eine Webschnittstelle in das Cuckoo-System, also in die Analyse-Umgebung, übertragen. Daran anknüpfend erfolgt die Generierung eines neuen Jobs zur Untersuchung der Datei, der in eine Datenbank zwischengespeichert wird. Im nächsten Schritt liest der Server des Cuckoo-Systems, der in Form von mehreren Python-Skripten realisiert ist, den Job aus und initialisiert den Analyse-Prozess.

Analyse-Sandbox

Mit der Initialisierung des Analyse-Prozesses wird eine verfügbare virtuelle Maschine, die Analyse-Sandbox, hochgefahren und innerhalb dieser alle Funktionen initialisiert, die zur Untersuchung des Verhaltens der Datei benötigt werden.

Agent

Unmittelbar nach dem Start der Analyse-Sandbox wird ein Agent initialisiert, welcher einen RPC-Server realisiert und für die Kommunikation mit dem Cuckoo-System zuständig ist. Der RPC-Server ermöglicht den Austausch von Daten zwischen beiden Kommunikationspartnern sowie die ferngesteuerte Ausführung von Funktionen innerhalb der Analyse-Sandbox. Demnach kann über diese Schnittstelle die potentielle Malware sowie die zur Untersuchung verwendeten Analyse-Module und Konfigurationsdateien übertragen werden.

Analyzer

Innerhalb der Analyse-Sandbox erfolgt die Untersuchung der Malware durch den Analyser. Dieser wird vor der zu untersuchenden Datei über den RPC-Server in die virtuelle Maschine übertragen und durch den Agent kontrolliert. Daher erfolgt die Steuerung der Analyse letztlich über das Cuckoo-System selbst. Der Analyser startet und kontrolliert wiederum die Module, welche zur Erfassung der Prozess-, Dateisystem-, und Netzwerkaktivitäten einer zu untersuchenden Datei eingesetzt und somit zur Durchführung der dynamischen Analyse innerhalb der Sandbox verwendet werden.

Nach Abschluss der Untersuchung wird der Job aus der Datenbank entfernt und aus den gesammelten Informationen der statischen sowie dynamischen Analyse automatisiert ein Ergebnisbericht erzeugt (siehe Kapitel 4.1.1).

4.2 Implementierung

Im Zuge der Bachelorarbeit erfolgte die Realisierung der Module *Syscall Tracer* und *Filesystem Tracer*. Die Implementierung wurde hierbei unter Verwendung der Skript-Sprache „Python“ durchgeführt, welche auch zur Entwicklung der zu Grunde liegenden Cuckoo Sandbox eingesetzt wurde. Die Module beinhalten die wesentlichen der in Kapitel 3 beleuchteten Techniken zur Überwachung von Prozess-, Dateisystem-, und Netzwerkaktivitäten auf Linux-Systemen. Somit wird die Cuckoo Sandbox um essentielle Funktionen zur dynamischen Analyse von Linux-Malware erweitert. Ferner können die Module über eine zusätzlich entwickelte Schnittstelle *Result Logger* direkt mit dem Cuckoo-System kommunizieren und somit die erfassten Daten in Echtzeit übertragen werden.

4.2.1 Modul „Syscall Tracer“

Der *Syscall Tracer* beinhaltet Funktionen, die mithilfe des Systemaufrufs *ptrace* eine kontrollierte Ausführung und dedizierte Beobachtung einer zu untersuchenden Datei bzw. einer Malware ermöglichen. Wie in Kapitel 3 „Techniken“ herausgearbeitet, können unter Verwendung von *ptrace* sowohl Prozess- als auch Dateisystem- und Netzwerkaktivitäten überwacht werden.

Zur Implementierung wurde die Python-Bibliothek *python-pttrace* verwendet. Hierbei handelt es sich um eine spezielles Erweiterungspaket, das entweder über das Python-Paketverwaltungsprogramm *pip* oder direkt von der Entwicklerseite bezogen werden kann (vgl. [Sti14]). Die Bibliothek ist zur Verwendung des *Syscall Tracers* während der Konfiguration innerhalb der Analyse-Sandbox zu installieren.

Der grundlegende Ablauf zur Initialisierung einer Überwachung mithilfe von *python-pttrace* ist im Wesentlichen mit dem Ablauf der in Kapitel 3.1.2 herausgearbeiteten Technik identisch. Die auf dem *ptrace*-Modul aufsetzende Python-Bibliothek erleichtert jedoch die Handhabung der bereitgestellten Funktionen des Betriebssystems. Der in Abbildung 10 zur Übersicht dargestellte Ablauf einer mithilfe von *python-pttrace* durchgeführten Überwachung wird nachfolgend näher erläutert:

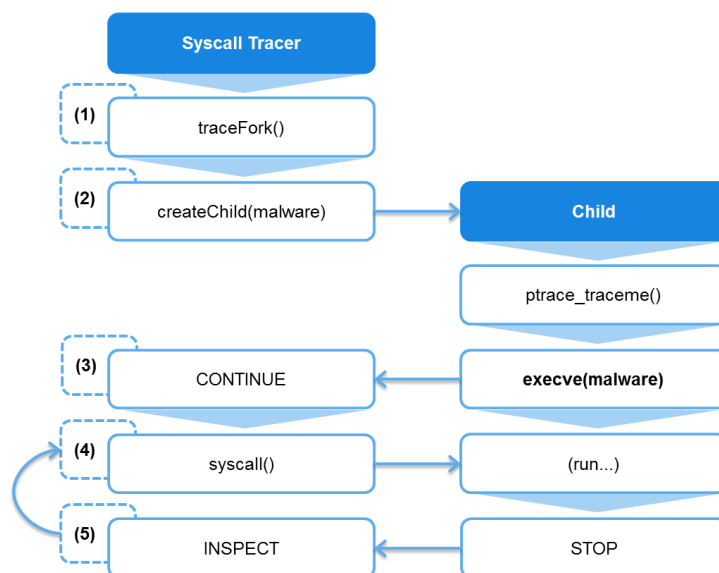


Abbildung 10: Realisierung der Überwachung mit *Syscall Tracer*

(1) Zu Beginn der Untersuchung wird durch den Analyzer (siehe Kapitel 4.1.2) ein zur Überwachung etablierter und im Benutzer-Modus laufender Prozess (*Syscall Tracer*) initialisiert. Dieser ermöglicht die kontrollierte Ausführung und dedizierte Beobachtung einer Malware. Die Methode `traceFork` aktiviert die Option `PTRACE_O_TRACEFORK`, welche in der *ptrace*-Schnittstelle hinterlegt ist (siehe Kapitel 3.1.2). Hierdurch erfolgt während der Ausführung eine automatische und parallelisierte Überwachung aller durch die Malware ausgeführten Unterprozesse.

(2) Die Funktion `createChild` erzeugt den zu überwachenden Kindprozess (*Child*). Im Detail erfolgt hierzu, wie auch in Kapitel 3.1.2, ein Aufruf von `fork`. Bevor *Syscall Tracer* die Ausführung fortsetzen kann, wird auf die Rückkehr dieser Funktion gewartet. Demnach ist bei Verwendung der Python-Bibliothek kein explizites Warten auf die Unterbrechung von *Child* notwendig. Durch den Aufruf der Methode `ptrace_traceme` wird anschließend eine Anfrage zur Verfolgung an *Syscall Tracer* gestellt.

(3) Die potentielle Malware wird mithilfe des Systemaufrufs `execve` in den Speicherbereich von *Child* geladen und dort gestartet. Somit wird das Schadprogramm von Beginn an überwacht. Unmittelbar nachdem die Malware initialisiert wurde, erfolgt bereits eine Unterbrechung durch den Linux-Kernel. Hierdurch wird die Funktion `createChild` beendet und die Ausführung von *Syscall Tracer* fortgesetzt (gekennzeichnet durch ein `CONTINUE` in Abbildung 10).

(4) Mit dem Aufruf der Funktion `syscall` erhält der Linux-Kernel die Anweisung, die Ausführung von *Child*, also des Malware-Prozesses, bis zum nächsten Systemaufruf fortzusetzen. Hierbei handelt es sich ebenfalls um einen Aufruf, der bis zur Rückkehr der Funktion wartet und somit eine kurzzeitige Unterbrechung von *Syscall Tracer* erzwingt.

(5) Sobald der Malware-Prozess aufgrund eines Systemaufrufs durch den Kernel angehalten wurde, kann *Syscall Tracer* eine Einsicht der mit dem Aufruf einhergehenden Parameter vornehmen. An dieser Stelle protokolliert das Modul alle verfügbaren Details sowie den Aufruf selbst (gekennzeichnet durch ein `INSPECT` in Abbildung 10). Ferner werden die gesammelten Informationen über die Schnittstelle *Result Logger* (siehe Kapitel 4.2.3) an das Cuckoo-System übermittelt. Bis zum Abschluss der Analyse wird wiederholt die Funktion `syscall` aufgerufen und somit ein Rücksprung zu Schritt 4 durchgeführt.

Hierdurch wird der Kernel angewiesen, *Child* sowohl vor der Ausführung als auch unmittelbar nach der Rückkehr eines Systemaufrufs anzuhalten. Besonders die Unterbrechungen nach Abschluss des Aufrufs können unter Umständen zur erfolgreichen Durchführung der dynamischen Analyse essentiell sein.

Detektion

In der Regel sind Malware-Entwickler darum bemüht, eine Detektion der Schadprogramme sowie die Untersuchung des zu Grunde liegenden Verhaltens zu erschweren. Hierzu versuchen die Entwickler häufig die zur Untersuchung eingesetzten Techniken und Mechanismen frühzeitig während der Ausführung der Malware zu erkennen und zu manipulieren. Dies umfasst beispielsweise die Identifizierung einer Analyse-Sandbox oder die Detektion von Techniken wie *ptrace*.

Der beispielhaft in Abbildung 11 aufgezeigte C-Quellcode zeigt eine einfache Überprüfung, die es einer Malware ermöglicht, eine mittels *ptrace* realisierte Überwachung des eigenen Prozesses zu erkennen:

```
#include <stdio.h>
#include <sys/ptrace.h>

int main() {
    if (ptrace(PTRACE_TRACEME, 0, 1, 0)) {
        printf("nobody traces me!\n");
        return 0;
    }
    printf("stop tracing me!\n");
    return 1;
}
```

Abbildung 11: Detektion einer Prozessüberwachung mithilfe von *ptrace*

Ein Prozess kann aufgrund der technischen Realisierung von *ptrace* nur von exakt einem Prozess überwacht werden (vgl. [Ker15a]).

Versucht ein bereits überwachttes Schadprogramm unter Verwendung des Aufrufs `ptrace` und des Parameters `PTRACE_TRACEME` eine weitere Verfolgung auf sich selbst zu initialisieren, führt dies somit zu einem Fehler. Folglich wird die Bedingung der in Abbildung 11 hervorgehobenen `if`-Anweisung nicht erfüllt und die Überwachung detektiert. Die Malware kann demnach die Beobachtung durch einen fremden Prozess auf einfache Weise erkennen und ihre tatsächliche Funktion verschleiern oder, wie im Beispiel gezeigt, die Ausführung beenden.

Verschleierung

Die Malware muss jedoch im Allgemeinen zur Detektion der Überwachung einen oder mehrere Systemaufrufe ausführen. Da mithilfe von `ptrace` sowohl die Parameter als auch der Rückgabewert eines Aufrufs eingesehen und modifiziert werden können, ist grundsätzlich eine Verschleierung der eingesetzten Analyse-Technik möglich.

Hierzu wurde *Syscall Tracer* um eine zusätzliche Methode `hide_me` erweitert. Diese modifiziert innerhalb des Schritts `INSPECTION` in Abbildung 10 den Rückgabewert des Aufrufs `ptrace(PTRACE_TRACEME, ...)` und verschleiert somit gegenüber dem Malware-Prozess die Überwachung. Wie nachfolgend in aufzeigt, werden hierzu in Python nur wenige Zeilen Code benötigt:

```
def hide_me(self, syscall, process):  
    # Identify ptrace syscall  
    if "ptrace" in syscall.name:  
        # change return value to zero  
        process.setreg('rax', 0)
```

Abbildung 12: Verschleierung der Verfolgung durch `ptrace`

Die dargestellte Funktion prüft zunächst, ob der Systemaufruf `ptrace` durch den Malware-Prozess aufgerufen wurde. Wird die Bedingung erfüllt, setzt der *Syscall Tracer* das CPU-Register `RAX` auf den Wert „0“. Das Register beinhaltet nach den „Microsoft Calling Conventions“, die ebenfalls unter Linux-Systemen berücksichtigt werden, den Rückgabewert einer ausgeführten Funktion (vgl. [Mic15]).

Durch die Modifizierung scheint der Aufruf aus Perspektive des Schadprogramms erfolgreich durchgeführt worden zu sein. Hierdurch wird von der Malware die Verfolgung mittels *ptrace* ausgeschlossen. Die entwickelte Funktion realisiert für das in Abbildung 11 aufgezeigte Beispiel folglich eine gelungene Verschleierung der Untersuchung.

Grundsätzlich ist an dieser Stelle die Vielzahl der verschiedenen Möglichkeiten zur Detektion einer Malware-Analyse zu beachten. Prinzipiell kann jedoch durch eine gezielte Modifizierung von Systemaufrufen mithilfe *ptrace* jedem Ansatz zur Erkennung sowie zur Manipulation der Untersuchung eines Schadprogramms entgegengewirkt und somit eine Verschleierung realisiert werden.

4.2.2 Modul „Filesystem Tracer“

Das Modul *Filesystem Tracer* dient zur Ergänzung der bereits mithilfe von *Syscall Tracer* realisierten Überwachung der Dateisystemaktivitäten. Letzterer protokolliert die Zugriffe auf das Dateisystem während der Beobachtung aller ausgeführten Systemaufrufe eines Malware-Prozesses. *Filesystem Tracer* erfasst hingegen unter Einsatz des Kernel-Moduls *inotify* die unmittelbar aus den Aufrufen resultierenden Aktivitäten innerhalb eines überwachten Verzeichnisses.

Die Implementierung erfolgte unter Einsatz der Python-Bibliothek *python-watchdog*, die wiederum direkt auf die durch das Betriebssystem bereitgestellte *inotify*-Schnittstelle zugreift. Die Bibliothek kann über das Python-Paketverwaltungsprogramm *pip* bezogen werden und ist innerhalb der zur Analyse verwendeten Sandbox zu installieren. Der grundlegende Ablauf zur Verwendung des Kernel-Moduls wurde bereits in Kapitel 3.2.2 herausgearbeitet. Dieser unterscheidet sich im Wesentlichen nicht von dem mittels *python-watchdog* implementierten und nachfolgend in Abbildung 13 zur Übersicht dargestellten Ablauf:

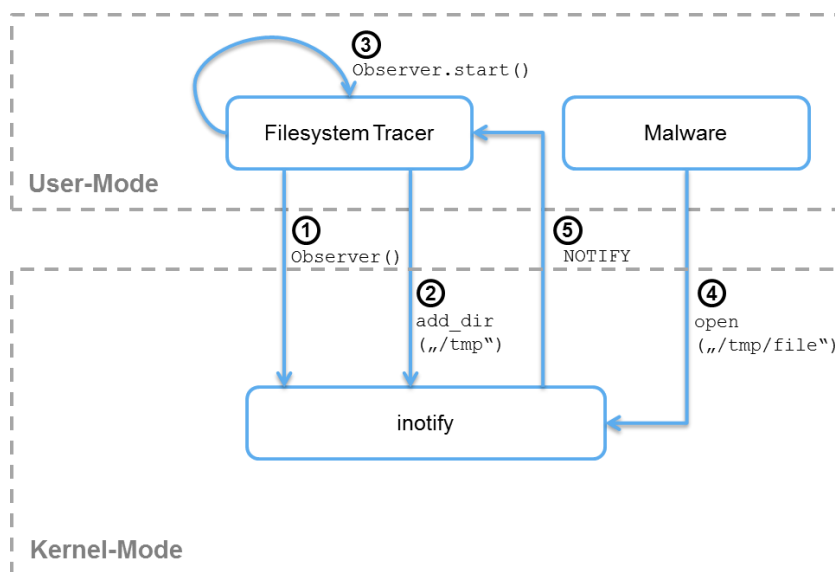


Abbildung 13: Realisierung der Überwachung mit *Filesystem Tracer*

- (1) *Filesystem Tracer* initialisiert zu Beginn einen Observer, der zur unmittelbaren Kommunikation mit dem Kernel-Modul verwendet wird.
- (2) Über den Observer werden die zu überwachenden Dateien oder Verzeichnisse spezifiziert und dem Kernel übermittelt. Innerhalb der Abbildung erfolgt beispielhaft unter Verwendung der Methode `add_dir` die Registrierung des Verzeichnisses `/tmp` sowie aller darin enthaltenen Dateien und Unterverzeichnisse.
- (3) Die Überwachung der spezifizierten Pfade erfolgt mit dem Aufruf der Observer-Methode `start`. Der Observer wird grundsätzlich innerhalb eines Unterprozesses ausgeführt. Auf diese Weise können die von der *Malware* erzeugten Verzeichnisse und Dateien nachträglich an den Observer übermittelt und in die Liste der beobachteten Pfade aufgenommen werden.
- (4) Nachdem die Beobachtung durch *Filesystem Tracer* erfolgreich etabliert wurde, erfolgt die Ausführung der *Malware*. Falls diese beispielsweise durch die Verwendung des Systemaufrufs `open` eine neue Datei innerhalb des `tmp`-Verzeichnisses anlegt, wird dies von *inotify* registriert.
- (5) Das Kernel-Modul generiert das Event `IN_CREATE` und übermittelt dieses über den Observer an *Filesystem Tracer*.

An dieser Stelle ermöglicht *Result Logger* eine Übertragung der Aktivität an das Cuckoo-System. Eine vollständige Liste aller verfügbaren Events ist beispielsweise aus dem „Linux Programmer’s Manual“ zu entnehmen (vgl. [Ker14a]).

Wie bereits in Kapitel 3.2.2 erwähnt, müssen die zu beobachtenden Verzeichnisse und Dateien explizit beim Kernel-Modul registriert werden. Innerhalb des Linux-Dateisystems existiert grundsätzlich eine Vielzahl von Verzeichnissen mit wichtigen Systemdateien. Eine Überwachung aller relevanten Verzeichnisse kann jedoch zu einem Performance-Verlust führen, da das Kernel-Modul nicht für eine umfassende Beobachtung beliebig vieler Pfade innerhalb des Dateisystems entwickelt wurde. Daher sollte während der Durchführung einer dynamischen Analyse ausschließlich eine Auswahl relevanter Verzeichnisse zur Beobachtung registriert werden. Typische Beispiele hierfür sind die Verzeichnisse `/sbin/`, welches im Allgemeinen die Binärdateien der vom Betriebssystem bereitgestellten Dienste enthält, und `/etc/`, das in der Regel die Konfigurationsdateien verschiedener Systemprogramme und Dienste beinhaltet.

Zur umfassenden und effizienten Detektion der von einer Malware ausgeführten Dateisystemaktivitäten sollten *Syscall Tracer* und *Filesystem Tracer* grundsätzlich in Kombination verwendet werden.

4.2.3 Schnittstelle „Result Logger“

Der *Result Logger* etabliert während der dynamischen Analyse eine Schnittstelle zwischen dem Cuckoo-System und der zur Untersuchung eingesetzten Analyse-Sandbox. Hierdurch wird eine Übermittlung der erfassten Daten über das Verhalten einer Malware in Echtzeit ermöglicht.

Die Kommunikation erfolgt konkret zwischen der Schnittstelle *Result Logger* innerhalb der Analyse-Sandbox und einem *Result Server* auf dem Cuckoo System. Der *Result Server* wurde bereits zur Analyse von Windows-Malware entwickelt. Für die Verwendung im Linux-Umfeld ist daher eine Adaption des bereits zum Datenaustausch etablierten Protokolls notwendig. Der generelle Ablauf der Kommunikation wird in Abbildung 14 dargestellt und an einem Beispiel nachfolgend im Detail erklärt:

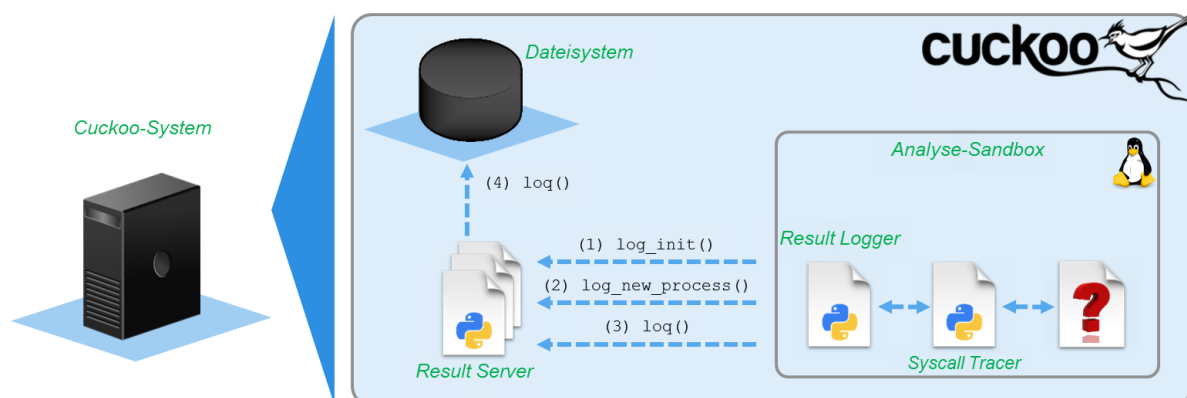


Abbildung 14: Implementierung der Schnittstelle *Result Logger*

(1) Der *Syscall Tracer* initialisiert zu Beginn der Untersuchung unter Verwendung der von *Result Logger* bereitgestellten Funktion `log_init` eine Socket-Verbindung zu *Result Server*, der innerhalb des Cuckoo Systems ausgeführt wird.

(2) Nachdem der Malware-Prozess gestartet wurde, erfolgt die Ausführung der Funktion `log_new_process`. Die Nachricht über die Ausführung eines neuen Prozesses wird an *Result Server* übermittelt und als Parameter unter anderem die zu Grunde liegende Prozess-ID übergeben.

(3) Sobald die Protokollierung eines Systemaufrufs erfolgt, wird eine entsprechende Nachricht generiert und mit der Funktion `log` an den *Result Server* innerhalb des Cuckoo-Systems übertragen. Die Nachricht wird hierbei im BSON-Format (Binary JSON⁴) erzeugt und versendet. Die `log`-Funktion überträgt die Nummer und den Namen des Systemaufrufs, den Rückgabewert sowie alle zur Ausführung des Aufrufs übergeben Parameter.

(4) Die Daten werden durch *Result Server* entgegengenommen und extrahiert. Zuletzt erfolgt die Archivierung der Informationen innerhalb einer Log-Datei auf dem Dateisystem. Die Datei wird nach Abschluss der Analyse zur Generierung des Ergebnisberichts eingesetzt (siehe Kapitel 4.1.1 „Konzept“).

⁴ Bei JSON handelt es sich um ein strukturiertes Datenformat, welches zum Datenaustausch zwischen zwei Anwendungen eingesetzt werden kann (vgl. [Mon15]).

Eine Echtzeit-Übermittlung der Daten während der Malware-Ausführung wirkt einem Datenverlust entgegen. Würden beispielsweise die gesammelten Informationen in Form einer Log-Datei bis zum Abschluss der Untersuchung gesammelt und erst anschließend übertragen werden, könnte bereits eine Modifizierung der zu Grunde liegenden Daten durch die Malware erfolgt sein. Somit wäre die Integrität der Daten nicht mehr gewährleistet. Ferner besteht während der Malware-Analyse grundsätzlich die Gefahr der Zerstörung eines Betriebssystems durch das Schadprogramm. Demnach handelt es sich bei der Schnittstelle *Result Logger* um ein essentielles Werkzeug, das zukünftig während der automatisierten Analyse von Linux-Malware innerhalb der Cuckoo Sandbox eingesetzt werden kann.

In dem Kapitel „Realisierung“ wurde zunächst in Anlehnung an die Cuckoo Sandbox das allgemeine Konzept einer Malware Analyse-Umgebung herausgearbeitet sowie die konkrete Funktionsweise dieser Lösung beschrieben.

Im zweiten Abschnitt erfolgte die Darstellung der während der Ausarbeitung entwickelten Module *Syscall Tracer* und *Filesystem Tracer* sowie der Schnittstelle *Result Logger*. Innerhalb der realisierten Module wurde unter Verwendung verschiedener Python-Bibliotheken die zuvor in Kapitel 3 erarbeiteten Techniken implementiert. Diese erweitern zukünftig die Cuckoo Sandbox um Funktionen zur dynamischen Analyse von Linux-Malware. Die realisierte Schnittstelle erlaubt schließlich während der Malware-Analyse den Echtzeit-Datenaustausch zwischen dem Cuckoo System und der zur Untersuchung eingesetzten Analyse-Sandbox.

5 Anwendung

Die Analyse von unbekannter Malware ist ein wesentlicher Bestandteil eines umfassenden Incident Response-Prozesses, der die Erhaltung bzw. Wiederherstellung der Informationssicherheit eines Unternehmens während und unmittelbar nach dem Eintritt eines Sicherheitsvorfalls gewährleisten soll. Aufgrund dieser Relevanz erfolgt abschließend die Demonstration der konkreten Anwendbarkeit des innerhalb dieser Ausarbeitung entwickelten Prototyps zur dynamischen Analyse von Linux-Malware. Hierzu wird zunächst ein Beispielszenario definiert, in dem das konkrete Verhalten einer echten Malware erläutert wird. Diese wurde während eines Incident Response-Einsatzes zur Behebung eines Sicherheitsvorfalls sichergestellt und in diesem Kontext bereits tiefgehend untersucht. Die Verwendung einer bereits analysierten Malware ermöglicht die Verifizierung der korrekten Arbeitsweise des Prototyps. Im Anschluss erfolgt eine beispielhafte Untersuchung des Schadprogramms, welche unter Verwendung der für die Cuckoo Sandbox entwickelten Module zur dynamischen Analyse von Linux-Malware durchgeführt wurde.

5.1 Beispielszenario

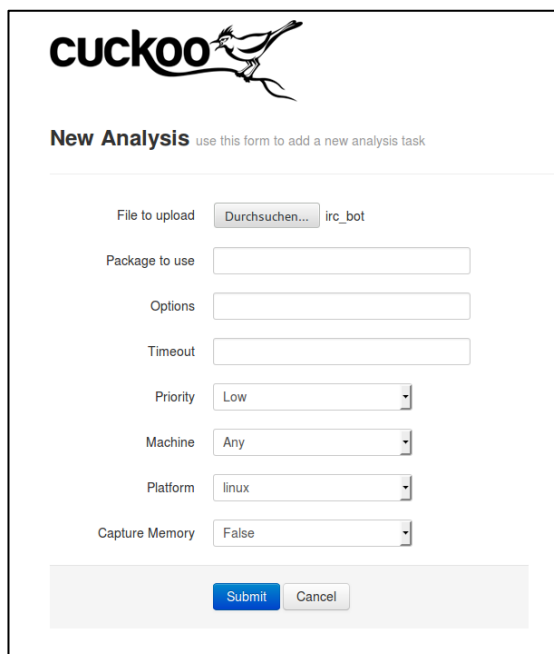
Bei der zu analysierenden Linux-Malware handelt es sich um einen IRC-Bot, der im Binärformat sichergestellt wurde. Dieser versucht unmittelbar zu Beginn der Ausführung eine Backdoor innerhalb eines zu Grunde liegenden SSH-Servers zu etablieren. Hierzu legt die Malware einen öffentlichen Schlüssel in die Datei `/root/.ssh/authorized_keys` ab und ermöglicht somit dem Besitzer des zugehörigen privaten Schlüssels per Public-Key-Authentifizierung eine Anmeldung als `root`-Benutzer (vgl. [Ubu15]). Zusätzlich erfolgt eine Verbindung zu einem auf `irc.quakenet.org` gehosteten IRC-Channel. Über eine IRC-Verbindung wird in der Regel die Steuerung eines Bots vorgenommen sowie ein Datenaustausch zwischen der Malware und dem Angreifer realisiert. Während der manuellen Analyse des vorliegenden Schadprogramms konnten jedoch keine weiteren Aktivitäten identifiziert werden, da keine Befehle über den IRC-Chat an die Malware verschickt wurden.

5.2 Malware-Analyse

Der Prozess der Malware-Analyse wird zur Übersichtlichkeit nachfolgend in die drei Abschnitte „Webschnittstelle“, „Analyse“ und „Ergebnisbericht“ unterteilt. In dem ersten Unterkapitel wird einleitend veranschaulicht, wie die automatisierte Analyse eines Schadprogramms innerhalb der Cuckoo Sandbox im Allgemeinen zu initialisieren ist. Darauf aufbauend erfolgt eine Darstellung der Arbeitsweise von den in Kapitel 4.2 beschriebenen Modulen, die während der Ausarbeitung in die Analyse-Umgebung implementiert wurden. Zuletzt werden die erfassten Ergebnisse beispielhaft erläutert und mit den im Beispielszenario beschriebenen Erkenntnissen verglichen. Auf diese Weise wird die Nutzbarkeit des im Zuge der Ausarbeitung entwickelten Prototyps aufgezeigt.

5.2.1 Webschnittstelle

Der Untersuchungsprozess beginnt im Wesentlichen mit der Einreichung einer Malware über ein von Cuckoo bereitgestelltes Webformular (vgl. Abbildung 15):



The screenshot shows the Cuckoo web interface. At the top left is the Cuckoo logo, which includes a stylized bird. Below the logo, the text reads "New Analysis use this form to add a new analysis task". The form contains several input fields and dropdown menus:

- "File to upload" with a "Durchsuchen..." button and the text "irc_bot".
- "Package to use" with an empty text input field.
- "Options" with an empty text input field.
- "Timeout" with an empty text input field.
- "Priority" with a dropdown menu set to "Low".
- "Machine" with a dropdown menu set to "Any".
- "Platform" with a dropdown menu set to "linux".
- "Capture Memory" with a dropdown menu set to "False".

At the bottom of the form are two buttons: "Submit" (in blue) and "Cancel".

Abbildung 15: Start einer Malware-Analyse über das Webformular von Cuckoo

Über das Webformular können verschiedene Parameter zur Konfiguration der geplanten Untersuchung gesetzt werden. Während der Entwicklung des Linux-Prototyps erfolgte eine Erweiterung des Formulars um den Parameter `Platform`.

Dieser erlaubt zukünftig eine Auswahl zwischen der Malware-Untersuchung innerhalb einer Windows- und Linux-Sandbox. Da es sich bei dem IRC-Bot um eine Linux-Malware handelt, wird an dieser Stelle die Linux-Sandbox ausgewählt. Die Funktionalität weiterer Parameter, wie `Timeout` oder `Options` ist bei Bedarf auf der Entwicklerseite nachzulesen (vgl. [Cuc14b]).

Mit einem Klick auf `Submit` sendet der Webserver die Malware sowie die eingegebenen Daten an das Cuckoo-System. Dies erfolgt über das Python-Skript `submit.py`, welches durch den Server mit den spezifizierten Parametern ausgeführt wird (vgl. Abbildung 16):

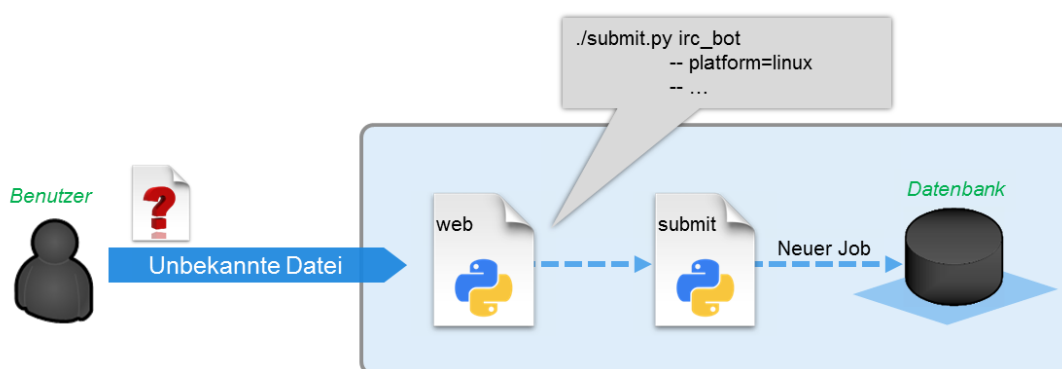


Abbildung 16: Übermittlung der eingegebenen Daten in die Cuckoo-Datenbank

Das Skript generiert daraufhin einen neuen Job und überträgt diesen in die Datenbank des zu Grunde liegenden Cuckoo-Systems. Wie der Abbildung zu entnehmen, wird auch der Parameter `Platform` mit dem Wert `linux` in die Datenbank geschrieben.

5.2.2 Analyse

Der Analyse-Prozess wird durch das Cuckoo-System initialisiert, indem der übermittelte Job ausgelesen und eine virtuelle Maschine, die Analyse-Sandbox, gestartet wird. Während des Startvorgangs erfolgt auf dem Cuckoo-System die Etablierung der Überwachung des (virtuellen) Netzwerkinterfaces und somit aller eingehenden sowie ausgehenden Verbindungen der Sandbox. Der innerhalb der virtuellen Maschine ausgeführte Agent realisiert nach Abschluss des Startvorgangs den zum Datenaustausch vorgesehenen RPC-Server (siehe Kapitel 4.1.2). Über den RPC-Server erfolgen die Übertragung des IRC-Bots sowie die Initialisierung der Malware-Ausführung innerhalb der Analyse-Sandbox.

Zur Initialisierung startet der von dem Agent kontrollierte Analyser die entwickelten Module *Syscall*- und *Filesystem Tracer* sowie die zur Echtzeit-Kommunikation entwickelte Schnittstelle *Result Logger*. *Syscall Tracer* initialisiert wiederum die Überwachung des Malware-Prozesses und führt die kontrollierte Ausführung des Schadprogramms durch, während *Filesystem Tracer* relevante Verzeichnisse und Systemdateien beobachtet (siehe Kapitel 4.2.1 sowie Kapitel 4.2.2).

Die während der Überwachung gesammelten Daten werden in Echtzeit an das Cuckoo-System übermittelt und dort abgespeichert. Ein beispielhafter Auszug der über *Result Server* empfangen Daten ist zur Übersicht in Abbildung 17 dargestellt:

```
DEBUG: New connection from: 192.168.56.102:41041

DEBUG: New process (pid=8900, ppid=8886, name=unkown_binary2,
                  path=/tmp/unkown_binary2)

DEBUG: log_call> tid:8900 apiname:brk

DEBUG: log_call> tid:8900 apiname:access

DEBUG: log_call> tid:8900 apiname:mmap

DEBUG: log_call> tid:8900 apiname:access

DEBUG: log_call> tid:8900 apiname:open

DEBUG: log_call> tid:8900 apiname:fstat

DEBUG: log_call> tid:8900 apiname:mmap
```

Abbildung 17: Echtzeit-Übertragung der erfassten Malware-Aktivitäten

Result Server akzeptiert zu Beginn eine neue Verbindung, die von *Result Logger* initiiert wurde. Weiterhin erfolgt eine Benachrichtigung über die Ausführung eines neuen Prozesses sowie dessen Prozess-ID und des zu Grunde liegenden Namens des Schadprogramms. Außerdem sind in der Abbildung beispielhaft die durch *Syscall Tracer* erfassten Systemaufrufe der Malware zu entnehmen. Zur Übersicht wurde hier auf die Angabe der Parameter verzichtet. Die empfangenen Daten werden von *Result Server* in eine Log-Datei gespeichert, die im nächsten Schritt zur Erzeugung des Ergebnisberichts herangezogen wird.

5.2.3 Ergebnisbericht

Der Ergebnisbericht umfasst grundsätzlich alle während der statischen und dynamischen Analyse gesammelten Daten und wird nach Abschluss der Untersuchung im HTML-Format zur Verfügung gestellt. Auf die Darstellung der Ergebnisse einer statischen Analyse wird jedoch innerhalb dieser Ausarbeitung nicht weiter eingegangen.

Während der dynamischen Analyse des IRC-Bots konnten sowohl Prozess- als auch Datei- und Netzwerkaktivitäten protokolliert werden. Einige der erfassten Aktivitäten werden zur Übersicht in Abbildung 18 dargestellt und nachfolgend erläutert:

Processes			
registry filesystem process services network synchronization			
irc_bot PID: 8900, Parent PID: 8886			
irc_bot fork1 PID: 8901, Parent PID: 8900			
Timestamp	Thread	Function	Arguments
23:49:10,727	8901	getppid	
23:49:10,727	8901	brk	brk => NULL
23:49:10,727	8901	brk	brk => 0x0000000000c5a000
23:49:10,728	8901	open	mode => <O_WRONLY O_CREAT O_APPEND> (02101) filename => '/root/.ssh/authorized_keys'
23:49:10,728	8901	fstat	fd => 3 statbuf => 0x00007fffbb3e2220
23:49:10,728	8901	mmap	prot => <PROT_READ PROT_WRITE> (3) start => NULL length => 4096 flags => 34 offset => 0 fd => -1
23:49:10,728	8901	fstat	fd => 3 statbuf => 0x00007fffbb3e22f0
23:49:10,729	8901	lseek	origin => 34 fd => 3 result => NULL offset => 0
23:49:10,729	8901	write	buf => 0x00007f3020fa0000 fd => 3 count => 405
23:49:10,730	8901	close	fd => 3
23:49:10,731	8901	munmap	length => 4096 addr => 0x00007f3020fa0000
23:49:10,731	8901	socket	type => SOCK_STREAM domain => AF_INET protocol => 6

Abbildung 18: Auszug aus einem Ergebnisbericht der Cuckoo Sandbox

Im Zuge der Untersuchung konnten insgesamt zwei Malware-Prozesse erfasst werden. Das Schadprogramm `irc_bot` hat über den Befehl `fork` einen Kindprozess erzeugt. Dieser greift über den Systemaufruf `open` schreibend auf die Datei `authorized_keys` in dem Verzeichnis `/root/.ssh/` zu.

Daran anknüpfend wurde mit einem `write` die an der Adresse `buf=0x7f3020fa0000` befindliche Zeichenkette mit der Länge `count=405` in die zuvor geöffnete Datei geschrieben. Dieses Verhalten lässt auf die Etablierung einer Backdoor schließen, da in `authorized_keys` die zur SSH-Verbindung benötigten öffentlichen Schlüssel des Public-Key-Verfahrens abgelegt sind (vgl. [Ubu15]).

Neben den beleuchteten Prozess- und Dateisystemaktivitäten konnten auch, wie oben mit dem Befehl `socket` zu entnehmen, Netzwerkaktivitäten erfasst werden. Ergänzend zu der chronologischen Auflistung aller protokollierten Systemaufrufe wird innerhalb des Ergebnisberichts eine entsprechende Übersicht der Netzwerk-Analyse aufgeführt. Diese ist auszugsweise in Abbildung 19 dargestellt und wird nachfolgend näher beschrieben:

Network Analysis	
Hosts Involved	
IP Address	
8.8.8.8	
80.65.57.26	
DNS Requests	
Domain	
irc.quakenet.org	
IRC Requests	
Command	Params
NOTICE	AUTH :*** Looking up your hostname
NOTICE	AUTH :*** Checking Ident
NOTICE	AUTH :*** Found your hostname

Abbildung 19: Auszug der durch Cuckoo erfassten Netzwerkaktivitäten

In der Kategorie „Hosts Involved“ werden alle IP-Adressen aufgelistet, zu denen während der Untersuchung eine Verbindung aufgebaut wurde. Ferner erfolgte ein DNS-Request zur Auflösung der Domain `irc.quakenet.org` auf die Adresse `80.65.57.26`.

Zuletzt erlaubt die externe Überwachung des virtuellen Netzwerkinterfaces eine detaillierte Beschreibung der vollständigen IRC-Kommunikation, wie beispielhaft in der Kategorie „IRC-Requests“ in zu einzusehen ist.

Der Ergebnisbericht zeigt, dass *keine* selbstständige und automatisierte Verbreitung der Malware erfolgt ist. Ferner wurden während der Untersuchung keine Daten vom zu Grunde liegenden System entwendet oder gelöscht. Jedoch hat eine Modifizierung der Datei `authorized_keys` im Verzeichnis `/root/.ssh/` stattgefunden. Zur Behebung eines echten Sicherheitsvorfalls wäre als nächstes der vermeintlich dort hinterlegte öffentliche Schlüssel des Angreifers sicherzustellen und alle potentiell gefährdeten Systeme innerhalb eines Unternehmens auf ein weiteres Vorkommen dieses Schlüssels hin zu untersuchen.

Des Weiteren kann die Eindämmung des Sicherheitsvorfalls durch eine temporäre Blockierung des IRC-Protokolls an einer Firewall oder dem Content Filter realisiert werden. Auf diese Weise könnte ein Angreifer keine weiteren Instruktionen an die potentiell im Unternehmen eingeschleusten IRC-Bots senden. Da ausschließlich ein öffentlicher Schlüssel als Backdoor etabliert wurde, kann eine Säuberung der betroffenen Systeme durch eine Entfernung des Schlüssels sowie der Malware selbst erfolgen.

Zusammenfassend lieferte der realisierte Prototyp während der dynamischen Analyse des IRC-Bots essentielle Informationen über das Verhalten des Schadprogramms. Diese können im Rahmen eines Sicherheitsvorfalls zur Beantwortung der in Kapitel 1.1 „Motivation“ herausgearbeiteten Fragestellungen eingesetzt werden.

Im Kapitel „Anwendung“ wurde die konkrete Anwendbarkeit der entwickelten Module zur dynamischen Analyse von Linux-Malware aufgezeigt. Hierzu erfolgte die Definition eines Beispielszenarios, in dem das Verhalten eines echten IRC-Bots erläutert wurde.

Der entwickelte Prototyp erkannte während der beispielhaft durchgeführten Analyse des IRC-Bots automatisiert die wesentlichen Verhaltensmerkmale, die zuvor im Beispielszenario beschrieben wurden. Die im Zuge dieser Ausarbeitung herausgearbeiteten und implementierten Techniken konnten demnach zur effizienten Erfassung von relevanten Prozess-, Dateisystem- und Netzwerkaktivitäten eingesetzt werden. Die im Ergebnisbericht zusammengefassten Informationen dienen zur Beantwortung der in Kapitel 1.1 herausgearbeiteten Fragen, die wiederum im Zuge der vollständigen Behebung eines Sicherheitsvorfalls essentiell sind.

6 Resümee

Das Resümee gibt eine Zusammenfassung über den Kern der Bachelorarbeit und stellt die erarbeiteten Ergebnisse heraus. Zunächst erfolgt hierzu innerhalb eines Rückblicks die Beleuchtung der Motivation sowie der Zielsetzung der Arbeit. Daran anknüpfend werden die Ergebnisse dieser Ausarbeitung dargestellt und die Umsetzung der gesetzten Ziele verifiziert. Weiterhin wird ein Ausblick über die zukünftige Einsatzmöglichkeit des entwickelten Prototyps sowie hinsichtlich verschiedener Ansätze zur Erweiterung aufgezeigt. Zuletzt erfolgt eine Abrundung der Bachelorarbeit durch ein persönliches Fazit über die dynamische Analyse von Linux-Malware.

6.1 Rückblick

Im Unternehmenskontext können Sicherheitsvorfälle trotz der Etablierung von vielseitigen Schutzmaßnahmen, wie zum Beispiel dem Einsatz aktueller Sicherheitsinfrastrukturen, nicht vollständig ausgeschlossen werden. Zu einem klassischen Sicherheitsvorfall gehört unter anderem ein Systemeinbruch, der aufgrund der Anbindung zu öffentlichen Netzen präferiert über Serversysteme erfolgt. Wird während der Bearbeitung eines Sicherheitsvorfalls, beispielsweise durch ein Incident Response-Team, auf einem System potentielle Malware vorgefunden, gilt es diese umfassend zu analysieren. Während zur sicheren und automatisierten Analyse von Windows-Malware bereits eine Vielzahl von kommerziellen und nicht-kommerziellen Produkten existieren, muss die Untersuchung von Linux-Malware aufgrund einer fehlenden Lösung bisher manuell durchgeführt werden.

Ziel der Bachelorarbeit war es, einen Prototyp zur sicheren und automatisierten Analyse von potentieller Linux-Malware zu entwickeln, der zukünftig beispielsweise im Zuge eines Incident Response-Einsatzes verwendet werden kann. Hierzu galt es einen zuvor realisierten sowie auf der Cuckoo Sandbox basierenden Prototyp zur statischen Analyse von Malware für Linux-Systeme um Funktionen und Techniken der dynamischen Analyse zu erweitern. Im Zuge der Ausarbeitung sollten die zu diesem Zweck benötigten Techniken erarbeitet und tiefgehend beleuchtet werden.

6.2 Ergebnis

Mit dieser Bachelorarbeit wird ein Prototyp zur statischen und dynamischen Analyse von Linux-Malware zur Verfügung gestellt. Die erarbeiteten Techniken zur Erfassung der Prozess-, Dateisystem- und Netzwerkaktivitäten eines zu Grunde liegenden Schadprogramms wurden hierzu in eine bestehende Open Source-Lösung, der Cuckoo Sandbox, implementiert. Die in der Motivation herausgearbeiteten grundlegenden Fragen, die zur vollständigen Behebung eines Sicherheitsvorfalls zu klären sind, können mithilfe der entwickelten Lösung effizient beantwortet werden:

- ▶ Wie erfolgte die Infizierung des Systems und wie funktioniert die Malware?
- ▶ Welche Daten wurden vom System und dem Unternehmen entwendet?
- ▶ Hat eine Infizierung weiterer Systeme stattgefunden und was sind Indizien dafür?
- ▶ Wie lässt sich die Malware vollständig entfernen?

Die realisierte Lösung ermöglicht, wie beispielhaft demonstriert, eine effiziente Bestimmung des Verhaltens einer Malware. Dadurch können in Zukunft eintretende Sicherheitsvorfälle schneller analysiert und behoben werden.

6.3 Ausblick

Der entwickelte Prototyp kann zukünftig vom Incident Response-Team der Controlware GmbH zur statischen sowie dynamischen Analyse von unbekanntem und unmittelbar verdächtigen Linux-Dateien eingesetzt werden.

Da es sich bei der Cuckoo Sandbox um eine unter der GNU GPLv3 lizenzierten Open Source-Lösung handelt, soll in Rücksprache mit den Entwicklern eine Integration der realisierten Funktionen in das Hauptprojekt erfolgen. Nach erfolgreicher Zusammenführung wird der Allgemeinheit auf diese Weise zukünftig ein nicht-kommerzielles Werkzeug zur statischen sowie dynamischen Analyse von Windows- und Linux-Malware zur Verfügung gestellt.

Im Zuge der Ausarbeitung kristallisierten sich zudem weitere Aufgaben zum Ausbau des Prototyps heraus, die in zukünftigen Arbeiten umgesetzt werden können.

Die Cuckoo Sandbox stellt während der Windows Malware-Analyse ein Werkzeug zur Erzeugung forensischer Speicherabbilder zur Verfügung. Folglich kann zukünftig auch eine Erweiterung des Linux-Prototyps um Funktionen zur Anfertigung dieser Abbilder erfolgen. Hierbei besteht die Möglichkeit, das bereits in Cuckoo zur Windows-Analyse etablierte Modul zur Bereitstellung der Daten innerhalb des Ergebnisberichts einzusetzen.

Ferner werden die während der Analyse gesammelten Informationen über das Verhalten der Malware im Zuge der Auswertung auf verhaltensbasierte Signaturen geprüft. Bisher existieren jedoch ausschließlich Signaturen zur Identifizierung von Windows-spezifischen verdächtigen Verhaltensmustern, wie die Modifizierung der Windows-Registry zur Etablierung einer persistenten Malware. In einem zukünftigen Projekt könnten Linux-spezifische Signaturen erstellt und dem Prototyp hinzugefügt werden.

6.4 Fazit

Die dynamische Analyse von Linux-Malware stellt ein interessantes und zugleich anspruchsvolles Themengebiet dar, welches zukünftig im Rahmen von Incident Response eine bedeutende Rolle spielen wird. Aufgrund der Tatsache, dass primär Windows-Systeme im Fokus von Angreifern und Malware-Entwicklern stehen, wird im Linux Server-Umfeld häufig der Einsatz von Sicherheitslösungen wie Anti Malware-Produkten vernachlässigt. Auch erfordert die Administration von diesen komplexen und vielseitigen Betriebssystemen ein umfangreiches Wissen und tiefgehendes technisches Verständnis der Materie. Ferner erfolgt innerhalb von Unternehmen zunehmend eine Auslagerung sensibler Daten von Endgeräten auf Server. Folglich steigt auch die Anzahl der Linux-Server, die sensible Informationen enthalten und bereitstellen müssen. Diese und weitere Faktoren machen die Entwicklung von Malware für Linux-Systeme zunehmend attraktiver. Mit dem in der Bachelorarbeit entwickelten Prototyp zur automatisierten Analyse von Linux-Schadprogrammen kann den Malware-Analysten und Incident Response-Teams zukünftig ein Werkzeug zur sicheren und effizienten Unterstützung bei der Behandlung von Sicherheitsvorfällen im Linux-Umfeld zur Verfügung gestellt werden.

Anhang A Weiterführende Materialien

A.1. Quellcode „create_file.c“

Nachfolgend ist der Quellcode des in Kapitel 3.2.1 verwendeten Programms, das zur Veranschaulichung der Überwachung von Dateisystemaktivitäten mithilfe des Systemaufrufs *ptrace* entwickelt wurde, vollständig abgebildet.

```
#include <stdio.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <unistd.h>

int main() {

    FILE *pFile;

    pFile = fopen("new_file.dat", "rb+");

    // Does the file already exist?

    if(pFile == NULL)

    {

        pFile = fopen("new_file.dat", "wb");

    }

}
```

Abbildung 20: Quellcode des Programms „create_file“

Anhang B Abkürzungsverzeichnis

BSI	Bundesamt für Sicherheit in der Informationstechnik
BSON	Binary JavaScript Object Notation
CPU	Central Processing Unit
DNS	Domain Name System
GPL	General Public License
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDS	Intrusion Detection System
IRC	Internet Relay Chat
JSON	JavaScript Object Notation
MD5	Message-Digest Algorithm 5
NIST	National Institute of Standards and Technology
RPC	Remote Procedure Call
PDF	Portable Document Format
SHA	Secure Hash Algorithm
SIEM	Security Information and Event Management
SSH	Secure Shell
USB	Universal Serial Bus

Anhang C Literaturverzeichnis

[Bun09] Bundesamt für Sicherheit in der Informationstechnik: *B 1.8 Behandlung von Sicherheitsvorfällen* : IT-Grundschutz Kataloge.

https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/_content/baust/b01/b01008.html, erstellt in 2009, zuletzt besucht am 27. Januar 2015.

[Bun14] Bundesministerium für Sicherheit in der Informationstechnik: : Die Lage der IT-Sicherheit in Deutschland 2014.

https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Lageberichte/Lagebericht2014.pdf?__blob=publicationFile, erstellt im November 2014, zuletzt besucht am 04. Februar 2015.

[CER14] CERT Bund: *Ebury SSH Rootkit*. <https://www.cert-bund.de/ebury-faq>, erstellt am 01. September 2014, zuletzt besucht am 18. Februar 2015.

[Cis15] Cisco: *What Is the Difference: Viruses, Worms, Trojans, and Bots?* : Cisco Systems. <http://www.cisco.com/web/about/security/intelligence/virus-worm-diffs.html>, zuletzt besucht am 5. März 2015.

[Cuc14a] Cuckoo Foundation: *cuckoosandbox.org* : Cuckoo Sandbox Book.

<http://docs.cuckoosandbox.org/en/latest/>, erstellt am 10. Oktober 2014, zuletzt besucht am 28. Februar 2015.

[Cuc14b] Cuckoo Foundation: *cuckoosandbox.org* : Submit an Analysis.

<http://docs.cuckoosandbox.org/en/latest/usage/submit/>, erstellt am 3. Oktober 2014, zuletzt besucht am 3. März 2015.

[Cuc14c] Cuckoo Foundation: *GitHub.com* : cuckoo/LICENSE.

<https://github.com/cuckoobox/cuckoo/blob/master/docs/LICENSE>, erstellt am 29. April 2014, zuletzt besucht am 28. Februar 2015.

[ESE09] ESET: *Detecting Unkown Viruses* : Heuristic Analysis.

http://www.eset.com/us/resources/white-papers/Heuristic_Analysis.pdf, erstellt in 2009, zuletzt besucht am 5. Februar 2015.

[Ker14a] Kerrisk, Michael: *inotify(7)* : Linux Programmer's Manual.

<http://man7.org/linux/man-pages/man7/inotify.7.html>, erstellt am 31. 12 2014, zuletzt besucht am 26. Februar 2015.

[Ker14b] Kerrisk, Michael: *pcap(3)* : Linux Manual Page. [http://man7.org/linux/man-](http://man7.org/linux/man-pages/man3/pcap.3pcap.html)

[pages/man3/pcap.3pcap.html](http://man7.org/linux/man-pages/man3/pcap.3pcap.html), erstellt im April 2014, zuletzt besucht am 27. Februar 2015.

[Ker15a] Kerrisk, Michael: *ptrace(2)* : Linux Programmer's Manual.

<http://man7.org/linux/man-pages/man2/ptrace.2.html>, erstellt am 22. Januar 2015, zuletzt besucht am 20. Februar 2015.

[Ker15b] Kerrisk, Michael: *waitpid(2)* : Linux Programmer's Manual.

<http://man7.org/linux/man-pages/man2/waitpid.2.html>, erstellt am 21. Februar 2015, zuletzt besucht am 24. Februar 2015.

[Ker15c] Kerrisk, Michael: *syscalls(2)* : Linux Programmer's Manual.

<http://man7.org/linux/man-pages/man2/syscalls.2.html>, erstellt am 22. Hanuar 2015, zuletzt besucht am 10. März 2015.

[KQ08] Kunst, Eva-Katharina; Quade, Jürgen: *linux-magazin.de* : Kernel- und

Treiberprogrammierung mit dem Kernel 2.6 - Folge 40. [http://www.linux-](http://www.linux-magazin.de/Ausgaben/2008/07/Kern-Technik)
[magazin.de/Ausgaben/2008/07/Kern-Technik](http://www.linux-magazin.de/Ausgaben/2008/07/Kern-Technik), erstellt im Juli 2008, zuletzt besucht am 24. Februar 2015.

[Kru14] Kruegel, Christopher: *Achieving Successful Automated Dynamic Analysis of*

Evasive Malware : Full System Emulation. [https://www.blackhat.com/docs/us-](https://www.blackhat.com/docs/us-14/materials/us-14-Kruegel-Full-System-Emulation-Achieving-Successful-Automated-Dynamic-Analysis-Of-Evasive-Malware-WP.pdf)
[14/materials/us-14-Kruegel-Full-System-Emulation-Achieving-Successful-Automated-](https://www.blackhat.com/docs/us-14/materials/us-14-Kruegel-Full-System-Emulation-Achieving-Successful-Automated-Dynamic-Analysis-Of-Evasive-Malware-WP.pdf)
[Dynamic-Analysis-Of-Evasive-Malware-WP.pdf](https://www.blackhat.com/docs/us-14/materials/us-14-Kruegel-Full-System-Emulation-Achieving-Successful-Automated-Dynamic-Analysis-Of-Evasive-Malware-WP.pdf), erstellt in 2014, zuletzt besucht am 16. Februar 2015.

[Mic15] Microsoft: *x64 Architecture* : Microsoft Developer Network.

<https://msdn.microsoft.com/en-us/library/windows/hardware/ff561499.aspx>, erstellt in 2015, zuletzt besucht am 3. März 2015.

[Mon15] MongoDB: *mongodb.com* : JSON and BSON. [http://www.mongodb.com/json-and-](http://www.mongodb.com/json-and-bson)

[bson](http://www.mongodb.com/json-and-bson), erstellt in 2015, zuletzt besucht am 4. März 2015.

[Nat05] National Institute of Standards and Technology: *Guide to Malware Incident Prevention and Handling* : NIST Special Publication 800-83.

<http://csrc.nist.gov/publications/nistpubs/800-83/SP800-83.pdf>, erstellt im November 2005, zuletzt besucht am 21. Januar 2015.

[Nat12] National Institute of Standards and Technology: *Computer Security Incident Handling Guide* : Special Publication 800-61 Revision 2.

<http://csrc.nist.gov/publications/nistpubs/800-61rev2/SP800-61rev2.pdf>, erstellt im August 2012, zuletzt besucht am 26. Januar 2015.

[Rem11] Remnant, Scott: *netsplit.com* : The Proc Connector and Socket Filters.

<http://netsplit.com/the-proc-connector-and-socket-filters>, erstellt am 9. Februar 2011, zuletzt besucht am 24. Februar 2015.

[Rog14] Rogmann, Nils: *Untersuchung von Sandbox-Lösungen zur sicheren Analyse von Linux-Malware*. Dietzenbach: Controlware GmbH, 2014.

[Sch08] Schilli, Michael: *Perl-Skript nutzt Ptrace zur Prozessüberwachung* : Linux-Magazin. [http://www.linux-magazin.de/Ausgaben/2008/04/Prozess-Spion/\(offset\)/2](http://www.linux-magazin.de/Ausgaben/2008/04/Prozess-Spion/(offset)/2), erstellt im April 2008, zuletzt besucht am 7. März 2015.

[SH12] Sikorski, Michael; Honig, Andrew: *Practical Malware Analysis*. San Francisco: No Starch Press, 2012.

[Sta11a] Stackoverflow: *How to ptrace a multi-threaded application?*.

<http://stackoverflow.com/questions/5477976/how-to-ptrace-a-multi-threaded-application>, erstellt am 30. März 2011, zuletzt besucht am 20. Februar 2015.

[Sta11b] Stackoverflow: *What is the difference between writing to a file and a mapped memory?*. <http://stackoverflow.com/questions/7280867/what-is-the-difference-between-writing-to-a-file-and-a-mapped-memory>, erstellt im September 2011, zuletzt besucht am 25.

Februar 2015.

[Sti14] Stinner, Victor: *readthedocs.org* : python-pttrace. [http://python-](http://python-pttrace.readthedocs.org/en/latest/)

[pttrace.readthedocs.org/en/latest/](http://python-pttrace.readthedocs.org/en/latest/), erstellt in 2014, zuletzt besucht am 1. März 2015.

[Tan09] Tanenbaum, Andrew: *Moderne Betriebssysteme*. München: Person Studium, 2009.

[Ubu15] Ubuntu.com: *SSH/OpenSSH/Keys* : Community Help Wiki.

<https://help.ubuntu.com/community/SSH/OpenSSH/Keys>, erstellt am 19. Januar 2015,
zuletzt besucht am 4. März 2015.