Technische Universität Dresden

Fakultät Informatik

Institut für Systemarchitektur

Professur für Datenschutz und Datensicherheit

Honeypots und Honey-Netze

Großer Beleg

Vorgelegt von: Pascal Brückner
geboren am 16.08.1988 in Zwickau
Matrikelnummer 3390554

Betreuer: Dr.-Ing. Stefan Köpsell
Dr,-Ing. Sebastian Clauß

Beginn der Arbeit: Dresden, den 10.12.2012
Tag der Einreichung: Dresden, den 10.06.2013

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die von mir eingerechte Belegarbeit zum Thema **Honeypots und Honey-Netze** vollkommen selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Dresden, den

Unterschrift:

(Pascal Brückner)

Abstract Sich wandelnde Trends und immer neue Gefahrenquellen durch Internetkriminalität erfordern schnelle Reaktionen bei der Etablierung von Verteidigungsstrategien. Honeypots sind bereits seit Jahren eine wichtige Komponente, die insbesondere bei der Aufspürung und Erforschung neuer, unbekannte Angriffe große Hilfe leisten kann. Die vorliegende Arbeit untersucht den aktuellen Entwicklungsstand auf diesem Gebiet. Es werden Grundlagen und Technologien aus dem Bereich der Honeypots vorgestellt und konkrete Implementierungen näher betrachtet. Zusätzlich werden die Einsatztauglichkeit in der Praxis, rechtliche Aspekte und mögliche zukünftige Entwicklungen behandelt. Es schließt sich ein umfangreicher Praxistest an, in dem miteinander kombinierte Honeypots in Form eines 50 Sensoren umfassenden, weltweilt verteilten Netzwerks zum Einsatz kommen und ihre gesammelten Daten ausgewertet werden.

Inhaltsverzeichnis

1 Einleitung **7**

2 Grundlagen **8**

2.1 Das Prinzip der Täuschung 8

2.2 Klassifikation 11

2.2.1 Interaktivität 11

2.2.2 Endpunkt 14

2.2.3 Sonstige 15

2.3 Einsatz 16

2.3.1 Sicherheitsaspekte 16

2.3.2 Integration ins Netzwerk 19

2.3.3 Spezialfälle 20

2.4 Rechtliche Aspekte 21

2.5 Hybride Verfahren 23

3 Ausgewählte Software **25**

3.1 Server-Side-Honeypots 25

3.1.1 kippo 25

3.1.2 dionaea 33

3.1.3 amun 36

3.1.4 SCADA-Honeypots 38

3.1.5 Mobile Honeypots 40

3.2 Client-Side-Honeypots 42

3.2.1 shelia 42

3.2.2 argos 47

3.2.3 Ghost USB Honeypot 49
3.3 Zusammenfassung 51

4 Planung und Realisierung von Honeypots/Honeynets 51
4.1 Vorüberlegungen 52
4.2 Architektur 53
4.2.1 Virtualisierung 53
4.2.2 Honeypot-Auswahl 54
4.2.3 Überwachung 55
4.2.4 Protokollierung 56
4.3 Implementierung 58
4.3.1 Installation 58
4.3.2 Betrieb 62

5 Auswertung der gesammelten Daten 65
5.1 Aggregierte Verbindungsdaten 65
5.2 Malware 68
5.3 Individuelle Sensoren 70

6 Zusammenfassung 74

Abbildungsverzeichnis

1 Häufigste Benutzernamen und Passwörter 32
2 Häufigste Benutzername/Passwort-Kombinationen 32
3 Die Komponenten des verteilten Honeypot-Netzwerks 58
4 Verbindungen und Angriffe im Verlauf des Experiments 65
5 Die TCP-Ports mit den meisten aufgezeichneten Verbindungen 66
6 Gesammelte Verbindungsdaten im Testzeitraum 68
7 Gesammelte Verbindungsdaten im Testzeitraum 69
8 Exploits 70
9 Die zehn am häufigsten von Angriffen betroffenen Sensoren 73
10 An den einzelnen Tagen im Messzeitraum aktive Sensoren 75

1 Einleitung

Der Norton Cybercrime Report[24] verzeichnet 556 Millionen Menschen, die im Jahr 2012 Opfer von Computerkriminalität geworden sind. Damit sind statistisch gesehen zwei Drittel der Erwachsenen, die sich im Internet bewegen, direkt betroffen gewesen. Im selben Jahr belief sich die Summe der durch Computerkriminalität verursachten Kosten auf 110 Milliarden Dollar. Das weltweite Spamaufkommen umfasst etwa 66% aller versendeten E-Mails[27], während jeden Tag weltweit über 1500 neue Websites geblockt werden, da sie Schadcode verbreiten. Angesichts dieser Zahlen ist es eine Selbstverständlichkeit, dass Regierungen, Universitäten und Unternehmen weltweit an Möglichkeiten forschen, der Kriminalität im Internet entgegenzuwirken. Die Trends der letzten Jahre zeigen, dass diese Aktivitäten wirkungsvoll sind: So verzeichnete Norton noch 2011 388 Milliarden Dollar verursachte Kosten durch Computerkriminalität, während jeden Tag noch mehr als 2300 Websites in Blacklists aufgenommen wurden[25]. Dieser positiven Entwicklung steht jedoch ein sich verschiebender Fokus aus Sicht der Cyberkriminellen gegenüber. Während "traditionelle" Angriffsmethoden stagnieren oder sogar rückläufig sind, weiten Angreifer ihren Aktionsradius nun auf neue Medien und Sachgebiete aus. Social-Media-Plattformen wie Facebook und Twitter, Cloud Services und Mobilgeräte auf Android- oder iOS-Basis rücken verstärkt in den Fokus[26]. Da Kriminelle hierzu stetig neue Angriffsmethoden entwickeln, gewinnen schnelle, im Idealfall automatisierte Reaktionen auf neue, unbekannte Angriffsszenarien und Exploits zunehmend an Bedeutung.

Neben den etablierten Verteidigungssystemen wie Firewalls, Intrusion Detection/Prevention Systemen und Virenscannern, die aktiv oder passiv nach Bedrohungen suchen und diese zu beseitigen versuchen, existieren auch noch die sogenannten Honeypots. Diese Systeme bedienen sich im Gegensatz zu den zuvor Genannten dem Prinzip der Täuschung. Anstatt potentielle Angreifer aus dem System auszuschließen, versuchen Honeypots, wie attraktive Ziele auszusehen und damit die Aufmerksamkeit von Kriminellen auf sich zu ziehen. Diese Systeme liefern wertvolle Daten über Verhaltensweisen und Angriffsmuster, können aber auch gleichzeitig zur Verbesserung der Sicherheit beitragen: Angreifer werden getäuscht und gelangen nicht an die gewünschten Informationen. Im Falle von nicht-automatisierten Angriffen wendet sich der Eindringling im Idealfall von den eigentlich gefährdeten Produktivsystemen ab und verliert dabei wertvolle Zeit. Zudem können aus den gewonnenen Daten über Angriffe Signaturen generiert werden, die von einem *Network Intrusion Prevention System* nutzbar sind. Leider ist mit Honeypots überlicherweise auch ein im Vergleich zu anderen Technologien deutlich größerer Installations- und Wartungsaufwand verbunden[11], was in der Praxis viele Administratoren davon abhält, diese in bestehenden Netzwerkinfrastrukturen zu integrieren.

Im Rahmen dieser Arbeit soll untersucht werden, welche verschiedenen Technologien auf dem Gebiet der Honeypots für Administratoren produktiver Netzwerke oder aber auch nur zur Erforschung von Trends zur Verfügung stehen und wie diese in der Praxis angewendet werden können. Nach einem allgemeinen Überblick über das Thema und nach einer Betrachtung der verschiedenen Arten von Honeypots werden konkrete Implementierungen im Detail vorgestellt. Die dabei gewonnen Erfahrungen fließen anschließend in zwei ausführlichere Praxistests ein: Der Erste umfasst ein System aus kombinierten Honeypot-Technologien an einem privaten Internetanschluss, der Zweite dagegen ein weltweit verteiltes Netzwerk aus 50 Sensoren, auf denen verschiedene Honeypots miteinander kombiniert und deren Daten zentral gesammelt und ausgewertet werden. Es werden außerdem rechtliche Aspekte beim Einsatz von Honeypots in Unternehmen oder auch im privaten Umfeld betrachtet, sowie ein kurzer Ausblick auf mögliche zukünftige Entwicklungen gegeben.

2 Grundlagen

2.1 Das Prinzip der Täuschung

Im Gegensatz zu herkömmlichen Verteidigungsstrategien zum Schutz von Computernetzwerken wie Firewalls oder Intrusion Detection Systemen, die versuchen Angreifer aus dem System auszusperren, setzen Honeypots auf das Prinzip der Täuschung. Die Idee ist, für Angreifer ein möglichst attraktives Ziel darzustellen und somit von den eigentlichen Produktivsystemen abzulenken. Während das Prinzip der Täuschung historisch gesehen schon sehr alt und seit jeher Teil militärischer Tradition ist, wurde es im Bereich der IT-Sicherheit zunächst in den späten 80er Jahren in der Literatur erwähnt[1] und erst 1998 mit der Entwicklung und Veröffentlichung des *Deception Toolkits* auch in der Praxis angewendet[8].

Definition Da die Technologie noch verhältnismäßig jung ist, existieren verschiedene Definitionen zum Begriff der "Honeypots" und ihrer Einsatzziele, die jedoch alle derselben Kernidee folgen. So definieren beispielsweise Christian Plattner und Reto Baumann den Begriff folgendermaßen[1]:

> "A honeypot is a resource which pretends to be a real target. A honeypot is expected to be attacked or compromised. The main goals are the distraction of an attacker and the gain of information about an attack and the attacker."

Alternativ dazu beschreibt die "Honeypot Mailing List" von *SecurityFocus* einen Honeypot als "an information system resource whose value lies in unauthorized or illicit use of that resource". Am weitesten verbreitet ist jedoch die Definition nach Lance Spitzner[38]:

> "A honeypot is a closely monitored computing resource that we want to be probed, attacked or compromised."

Analog dazu definieren Provos und Holz [33] den Begriff des „Honeynet":

> "We can also combine several honeypots to a network of honeypots: a honeynet."

Ziel einer Honeypot-Installation ist also das Errichten eines Systems, das für einen Angreifer möglichst nicht von einem echten System gleichen Typs zu unterscheiden ist. Das Honeypot-System ist möglichst gut überwacht, um die Schritte des Angreifers bei einem potentiellen Angriff auf das System nachvollziehen zu können und um zu verhindern, dass der Honeypot bei einer letztendlich unvermeidlichen, da ja gewünschten Manipulation als Ausgangspunkt zum Angriff auf andere Systeme dient. Die einen Honeypot errichtende Partei profitiert von den gewonnenen Daten über die Vorgehensweise und eventuell auch Identität des Angreifers und gewinnt zudem Zeit, da der Kriminelle von den eigentlich produktiven Systemen abgelenkt ist.

[1] *Kuckucksei*, Sachbuch von Clifford Stoll, 1989

Wenn ein Angreifer keinen Unterschied zwischen einem echten System und einem Honeypot erkennen kann, ist die Täuschung perfekt und der Honeypot hat seinen Einsatzzweck erfüllt. Dieser Idealfall ist in der Praxis jedoch, wenn überhaupt, nur mit exorbitant hohem Aufwand zu erreichen. Da es häufig schon ausreicht, einen Angreifer nur für kurze Zeit auf eine falsche Fährte zu locken und die in dieser Zeit gewonnenen Daten genutzt werden können, um auf die potentielle Gefahr angemessen zu reagieren oder der Angreifer das Interesse verliert, sind Honeypots im Normalfall keinesfalls genau so komplex wie ein Produktivsystem. Viel mehr handelt es sich üblicherweise nur um Systeme, die auf den ersten Blick ähnlich wie echte Systeme aussehen, sich aber erst nach einem erfolgreichen Angriff oder bei genauerer Inspektion der Systemeigenschaften als eine Täuschung herausstellen. Die Vergleichbarkeit mit einem realen System ist eines der wichtigsten Merkmale bei der Klassifikation und Bewertung der Qualität von Honeypot-Software.

Eine der größten Stärken einiger Honeypot-Technologien im Vergleich zu *Network Intrusion Detection Systemen* (NIDS) ist die Fähigkeit von Honeypots, *Zero-Day-Attacken* zu erkennen. Bilge und Dumitras definieren diesen Begriff folgendermaßen: „A zero-day attack is a cyber attack exploiting a vulnerability that has not been disclosed publicly“ [2]. Sogenannte High-Interaction-Honeypots stellen umfassend überwachte, vollwertige Systeme dar, die auf alle Zustandsänderungen im Dateisystem oder Arbeitsspeicher reagieren können. Da jegliche Kommunikation mit einem solchen System, abgesehen von einer im Bedarfsfall nutzbaren Whitelist, um ein kommunizierendes Produktivsystem nachzustellen, als verdächtig einzustufen ist, kann so ein Honeypot auch Daten über bisher noch unbekannte Angriffe sammeln. NIDS hingegen können beispielsweise benutzt werden, um Angriffs- oder Malware-Signaturen im Netzwerk-Paketfluss zu erkennen. Wenn Netzwerkdienste allerdings auf Verschlüsselung setzen, um die Sicherheit für Nutzer in öffentlichen Netzen zu erhöhen, verliert dieser NIDS-Betriebsmodus an Bedeutung. Da Honeypots aber die Endpunkte dieser Ende-zu-Ende-Kommunikation nutzenden Dienste darstellen, können sie die Daten, die ausgetauscht werden, unverschlüsselt einsehen und bei Bedarf ähnlich wie ein *Network Intrusion Prevention System* (NIPS) darauf reagieren.

Ein Honeypot, der beispielsweise in einem Unternehmensnetzwerk platziert wurde, weist im Normalfall keinen Datenverkehr auf, da die von ihm angebotenen Dienste lediglich Angreifer täuschen sollen. Für die Mitarbeiter des Unternehmens ist das System somit im Idealfall unsichtbar. Ein Honepot ist daher ein *passives System*, von dem im regulären Betrieb keine Netzwerkverbindungen ausgehen. Sobald an einem Honeypot jedoch Datenverkehr auftritt, deutet dies auf einen möglichen Angriff hin [33]. Honeypots werden im Gegensatz zu NIDS in der Regel nicht an zentralen Knotenpunkten im Netzwerk installiert, sondern als separate Systeme, die beispielsweise möglichst den normalen Arbeitsrechnern oder Servern im Unternehmen ähneln sollen. NIDS, die signaturbasiert arbeiten, sind im Allgemeinen anfällig, false-negatives zu liefern, da die Qualität der Klassifikation insbesondere von den verwendeten Signaturdaten abhängt. Honeypots hingegen stufen jeglichen auftretenden Datenverkehr zuerst einmal als verdächtig ein. Daraus lässt sich allerdings auch schlussfolgern, dass NIDS/NIPS und Honeypots nicht sich gegenseitig ausschließende, sondern viel mehr sich ergänzende Technologien darstellen. Damit ein Honeypot für einen Angreifer möglichst glaubwürdig aussieht, kann es sinnvoll sein, gezielt Netzwerkverkehr zu generieren und Aktivität vorzutäuschen. In diesem Fall bietet sich der Einsatz einer Whitelist auf dem Honeypot-System an, um diese „Dummydaten“ von der Klassifikation auszuschließen.

Vorteile Die Installation eines Honeypots in einem Produktivnetzwerk bietet eine Reihe von Vorteilen:

- Über den Angreifer und seine Vorgehensweise beim Untersuchen, Angreifen und Kompromittieren

des Systems werden wertvolle Daten gewonnen, die in späteren Schritten ausgewertet und zum Absichern der IT-Infrastruktur verwendet werden können. Dies ist besonders im akademischen Umfeld interessant, da Honeypots neben statistisch wertvollen Daten wie der Häufigkeit von Angriffen auch sehr detaillierte Informationen über die Vorgehensweise liefern und es mit ihnen möglich ist, neben bereits bekannten Angriffen auch neue, bisher unbekannte zu erkennen (Zero-Day).
- Der Angreifer kann durch das Vorhandensein von Honeypots getäuscht werden und bekommt ein falsches Bild von der Netzwerkinfrastruktur. Es besteht die Chance, dass er anstatt einem realen Produktivsystem seine ganze Aufmerksamkeit auf einen Honeypot richtet und somit viel Zeit verliert, ohne zu Ergebnissen zu gelangen. Dies trifft insbesondere auf „manuelle“, menschliche Angreifer zu und weniger auf Bots, die automatisch große Mengen an Zielen anzugreifen versuchen.
- Je nach eingestzter Honeypot-Technologie ist eine sofortige Reaktion auf das Eindringen eines Angreifers möglich, wie beispielsweise das gezielte Aussperren desselben aus dem Netzwerk.

Nachteile Honeyppots gehören noch immer nicht zu den Standard-IT-Sicherheitslösungen, wie es beispielsweise Firewalls oder Antivirensoftware tun. Dies hat mehrere Gründe:

- Es gibt keine Referenzsoftware auf diesem Gebiet. Die meisten Honeypotlösungen sind OpenSource und im Rahmen von akademischen Arbeiten entstanden. Nur wenige werden nach längerer Zeit noch gepflegt oder besitzen eine aktive Community.[7] Gleichzeitig gibt es eine große Auswahl an Software für die verschiedensten Einsatzzwecke, aus denen die “Richtige” auszuwählen für Administratoren einen hohen Zeitaufwand bedeutet.
- Der Installations- und Wartungsaufwand kann je nach Honeypot-Typ (eine Beschreibung der verschiedenen Arten erfolgt im nächsten Kapitel) den von herkömmlichen IT-Sicherheitslösungen wie Firewalls und Antivirenlösungen deutlich überschreiten. Für die vielen Informationen, die durch einen Honeypot gewonnen werden können, existieren bis heute nur wenige und oft nicht ganz zuverlässige Methoden zur automatisierten Auswertung. Es ist daher unvermeidlich, die Honeypot-Systeme und ihre anfallenden Daten aktiv im Auge zu behalten und auszuwerten.
- Risiko: Je genauer ein Honeypot-System ein reales System imitiert, desto größer ist auch die Gefahr, dass dessen Kompromittierung dazu führt, dass der Honeypot selbst die Kontrolle übernimmt und ausgehend vom Honeypot weitere Angriffe vornimmt. Dies ist allein schon rechtlich gesehen ein großes Risiko, für dessen Minimierung eine exakte Überwachung des Systems erforderlich ist. Auch dies resultiert in zusätzlichem Zeitaufwand.
- Die bei einem Honeypot anfallenden Daten liegen meistens nur in sehr technischem, für wenig erfahrene Administratoren nicht verständlichem Format vor. Der Betrieb eines Honeypots bedarf also auch gesondert geschultes Personal.
- Ein Honeypot-System innerhalb eines Netzwerk bietet für das betreibende Unternehmen keinen wirtschaftlichen Nutzen, da es nicht wie ein normales Produktivsystem genutzt werden kann.
- Honeypots können lediglich als Ergänzung zusammen mit anderen Sicherheitslösungen wie *Intrusion Detection Systemen* (IDS), Firewalls oder Antivirensoftware verwendet werden. Ein Honeypot allein bietet keinen ausreichenden Schutz vor Gefahren und ersetzt nicht die anderen Verfahren. Er kann allerdings deren Effizienz verbessern, da er unter Umständen zusätzlich noch Zero-Day-Angriffe erkennen und die daraus gewonnen Daten beispielsweise zur Entwicklung neuer Signaturen für ein *Network Intrusion Detection System* (NIDS) verwenden kann.

Zusammenfassend können Honeypots also sowohl die Sicherheit im Netzwerk oder Unternehmen verbessern, in welchem sie zum Einsatz kommen, als auch wertvolle Informationen über potentielle Angreifer liefern. Sie bringen jedoch einen großen Installations- und Wartungsaufwand mit sich und sollten nur zusätzlich zu bereits etablierten Sicherheitslösungen eingesetzt werden. Wie eine Studie zeigt, überwiegen für viele Unternehmen noch immer die Nachteile und sie entscheiden sich deshalb gegen das Aufsetzen von Honeypots[11].

2.2 Klassifikation

Die Idee des Honeypots ist ein theoretisches Konzept mit den zuvor genannten Zielen, das auf vielfache Weise in die Praxis umgesetzt werden kann. Im Laufe der Zeit haben sich dabei verschiedene Herangehensweisen hervorgehoben, für deren Klassifikation wiederum verschiedene Taxonomien vorgeschlagen wurden, wie beispielsweise die von Seifert et al. vorgeschlagene „Taxonomy of Honeypots“ [37]. Den meisten Klassifikationsversuchen gemein ist die Einteilung nach Interaktivität (*Low-Interaction-Honeypots* und *High-Interaction-Honeypots*) und Endpunkt (*Client-Side-Honeypots* und *Server-Side-Honeypots*). Das Verhalten jeder Honeypot-Softwarelösung kann nach diesen beiden Kategorien präzisiert werden. Es existieren zudem auch hybride Varianten, die wahlweise als Client- oder als Server-Side-Honeypot eingesetzt werden können und in einem separaten Abschnitt behandelt werden.

2.2.1 Interaktivität

Die Täuschung eines Angreifers gelingt um so besser, je mehr das betreffende Honeypot-System einem Produktivsystem ähnelt. Wenn der Angreifer keinen Unterschied mehr zwischen beidem ausmachen kann, ist die Täuschung gelungen. Je intensiver mit dem Honeypot interagiert werden kann, ohne dass die Täuschung offensichtlich wird, desto mehr Daten können letztendlich über einen Angreifer und seine Vorgehensweise gewonnen werden.

Die umfangreichste Interaktivität bieten sogenannte **High-Interaction-Honeypots**. Es handelt sich dabei um vollwertige Systeme, auf denen dieselbe Software läuft, die auch auf einem Produktivsystem zum Einsatz kommen würde. Niels Provos und Thorsten Holz definieren in ihrem Standardwerk *Virtual Honeypots* einen *High Interaction Honeypot* wie folgt[33]:

> “High-Interaction honeypots offer the adversary a full system to interact with. This means that the honeypot does not emulate any services, functionality, or base operating systems. Instead, it provides real systems and services, the same used in organizations today. Thus, the attacker can completely compromise the machine and take control of it.”

Dies lässt sich beispielsweise mit einem separaten Rechner realisieren, auf dem ein ganz normales Betriebssystem und für die jeweilige Anwendungsdomäne typische Programme installiert werden. Zusätzliche Software, die Teil einer Honeypot-Distribution ist, übernimmt dann die Überwachung dieses Systems: So kann nur der Datenfluss im Netzwerk von und zum Honeypot, in anderen Fällen sogar jegliche Aktivität auf dem System selbst protokolliert werden (das Anlegen oder Verändern von Dateien und Verzeichnissen, das Starten und Beenden von Prozessen usw.). Wie bereits im vorigen Kapitel angesprochen, bedarf dieses System noch einer zusätzlichen Überwachungsinfrastruktur, die sicherstellt, dass

ein Angreifer im Falle eines Einbruchs in den Honeypyot keinen Schaden anrichten kann. Hierzu existiert wiederum spezialisierte Software, insbesondere die weit verbreitete *Honeywall*, die in Form einer unsichtbaren Netzwerkbrücke das Logging und das Absichern eines IP-Adressbereiches übernimmt.

High-Interaction-Honeypots liefern folglich sehr umfangreiche Datenmengen und sind nützlich, um detaillierte Informationen über die Vorgehensweise und Hilfsmittel eines Angreifers zu erfahren. Da Dienste und andere Schnittstellen nicht nur emuliert werden, eignet sich diese Technologie hervorragend zum Erkennen von Zero-Day-Exploits. Zusätzlich ähneln einige High-Interaction-Honeypots in ihrer Architektur einer *Malware-Sandbox* und können somit nach einem Exploit in einem weiteren Schritt auch zur Analyse von Malware herangezogen werden. Das System kann gescannt, angegriffen und auch erfolgreich kompromittiert werden und gewährt dem Angreifer auch anschließend noch größtmögliche Freiheit. Dies stellt allerdings gleichzeitig auch den größten Nachteil dieser Technik dar: Das System kann durch einen Angreifer vollständig übernommen werden und ist somit selbst gefährlich. Es ist dann die Aufgabe des Honeypot-Betreibers, das System genauestens zu überwachen und zu verhindern, dass vom Honeypot aus andere Ziele angegriffen werden oder anderweitig Schäden entstehen. Für das sogenannte "Containment" von High-Interaction-Honeypots existieren verschiedene Hilfsmittel, die allerdings viel Hintergrundwissen, Erfahrung und vor allen Dingen Zeit voraussetzen.

Nachdem ein solches System kompromittiert und die Aktionen des Angreifers genauestens beobachtet wurden, muss der Honeypot wieder in den Ursprungszustand zurückversetzt werden. Wenn hierfür ein gesonderter Rechner verwendet wurde, muss ein bei der Installation angefertigtes, nicht kompromittiertes Image des Systems wiederhergestellt oder im schlimmsten Fall sogar alles von Hand neu installiert werden. Als Alternative dazu schlagen Niels Provos und Thorsten Holz vor, auf die Variante der Virtualisierung von High-Interaction-Honeypots zurückzugreifen[33]: Gängige Virtualisierungslösungen erlauben das Erstellen von *Snapshots* eines Systems im laufenden Betrieb und erleichtern das Wiederherstellen enorm. Zusätzlich können auf einem entsprechend leistungsfähigem Host mehrere Honeypot-Instanzen gleichzeitig betrieben werden, was ein deutlich effizienteres Setup im Vergleich zu mehreren gesonderten physischen Systemen darstellt. Die Isolation der virtuellen Maschinen wird zudem bereits durch die Virtualisierungssoftware gewährleistet. Niels Provos und Thorsten Holz bezeichnen diese beiden Varianten folglich als **Physical Honeypots** und **Virtual Honeypots**.

Honeypot-Systeme mit geringer Interaktivität werden als **Low-Interaction-Honeypots** bezeichnet. Es handelt sich dabei im Wesentlichen um Emulationen von Diensten oder Betriebssystemen. In "Virtual Honeypots" findet sich folgende Definition[33]:

> "A low-interaction honeypot often simulates a limited number of network services and implements just enough of the Internet protocols, usually TCP and IP, to allow interaction with the adversary and make him believe he is connecting to a real system."

Da die Tiefe und Genauigkeit des zu nachzubildenden Systems limitiert sind, stellt die Dauer, für die ein Angreifer sich mit dem Honeypot aufhält, bevor die Täuschung evident wird, den entscheidenden Faktor bei der Bewertung der Qualität dar. Low-Interaction-Honeypots eigenen sich in der Regel besonders, um Informationen über womöglich bösartige Aktivitäten im Netzwerk zu sammeln, das "Hintergrundrauschen" im Internet für bestimmte IP-Adressen/Domains zu analysieren oder Angreifer für begrenzte Zeit von Produktivsystem abzulenken. Sobald dem Angreifer jedoch bewusst geworden ist, dass er einer Täuschung unterliegt, wird er sich schnell zurückziehen oder sich anderen Systemen zuwenden. Einige Low-Interaction-Honeypots sind zudem bereits mit einfachen Scan-Techniken von außen zu erkennen,

da sie beispielsweise Protokolle oder den Netzwerk-Stack des Betriebssystem, das sie simulieren, nicht korrekt implementieren.

Der Betreiber eines Low-Interaction-Honeypots profitiert insbesondere vom geringeren Risiko, das beim Betrieb einer solchen Softwarelösung entsteht. Da ein Angreifer nur mit einer Emulation interagiert, die in der Regel nur Teile eines Protokolls nachbildet und jegliche Schritte des Eindringlings vom System überwacht und bei Bedarf „entschärft“ werden, ist es nicht möglich, dass das vermeintliche System vollständig übernommen wird. Ein solcher Honeypot ist daher mit einer virtuellen Maschine zu vergleichen, die nur minimale Interaktionsmöglichkeiten bereitstellt. Falls ein Angreifer für den Low-Interaction-Honeypot unbekannte Angriffsmuster verwendet, werden diese zu keiner Kompromittierung führen, da die Software das jeweilige Protokoll nicht implementiert, sondern nur simuliert. Was bleibt, ist die Gefahr, dass in der Honeypot-Software selbst Fehler gefunden werden, die es dem Angreifer ermöglichen, aus der künstlichen Hülle des Honeypots auszubrechen. Zur Vorbeugung ist es in diesem Zusammenhang sinnvoll, Low-Interaction-Honeypots zusätzlich abzusichern: Beispielsweise mit dem Betrieb innerhalb einer *chroot*-Umgebung[41], *User-Mode-Linux* (UML), Virtualisierungstechniken wie *QEMU* oder *KVM* oder einem Tool wie *systrace*[33]. Zusätzlich zu den bereits genannten Eigenschaften haben Low-Interaction-Honeypots geringere Systemanforderungen als High-Interaction-Honeypots, da nur ausgewählte Teile eines Systems oder Services emuliert werden. Es ist folglich leicht möglich, mehrere verschiedene Dienste unter Ausnutzung verschiedenster Software auf einem einzelnen System zu betreiben, was im praktischen Teil im Rahmen dieser Arbeit auch getan wurde. Software wie *honeyd*[2] ermöglicht es sogar, mehrere hundert oder tausend Hosts, ganze Netzwerke oder Routing-Topologien mit nur einem einzelnen Rechner zu emulieren, allerdings wie bereits erwähnt auf Kosten der simulierten Systemtiefe.

Während High-Interaction-Honeypots immer vollständige Systeme darstellen und alle Softwarelösungen damit den maximal möglichen Grad an Interaktion bieten, gibt es für Low-Interaction-Honeypots verschiedene Abstufungen: Manche Honeypots spezialisieren sich nur darauf, für Angreifer von außen wie ein bestimmtes Ziel auszusehen, erlauben jedoch über das Aufbauen einer (TCP-)Verbindung und eventuell dem Senden eines protokollspezifischen Banners hinaus keine weitere Interaktion. Ein Beispiel hierfür ist das *Deception Toolkit*[3], das heutzutage aber nur noch historischen Wert genießt. Moderne Low-Interaction-Honeypots sind im Vergleich dazu häufig komplexer und emulieren die für den regulären Betrieb oder den Angreifer interessantesten Protokolleigenschaften.

Projekte wie der SSH-Honeypot *kippo* emulieren neben dem SSH-Protokoll selbst sogar eine vollständige Shell, mit der ein Angreifer interagieren und auch Downloads tätigen kann. Die Entwickler dieses Projekts ordnen kippo selbst als **Medium-Interaction-Honeypot** ein[17]. Für diese Kategorie gibt es in der Literatur voneinander abweichende Definitionen. So beschreibt sie Georg Wicherski in [44] folgendermaßen:

> “Medium Interaction Honeypots try to combine the benefits of both approaches in regards to botnet detection and malware collection while removing their shortcomings. [...] These kind of honeypots do not aim at fully simulating a fully operational system environment, nor do they implement all details of an application protocol. All that these kind of honeypots do is to provide sufficient responses that known exploits await on certain ports that will trick them into sending their playload.”

[2] `http://www.honeyd.org/`
[3] `http://www.all.net/dtk/index.html`

Im Anschluss beschreibt er *nepenthes*[20] als prominentes Beispiel für einen solchen Honeypot. Dabei handelt es sich um eine Software, die Schwachstellen verschiedener Dienste simuliert und über den eigentlichen Exploit hinaus versucht, den vom Angreifer übertragenen Shellcode zu analysieren und Malware herunterzuladen, die zur Übernahme des Systems verwendet werden soll. Da diese Funktionalität über das bloße Emulieren von Schwachstellen hinausgeht, ähnlich wie der SSH-Honeypot kippo nach dem erfolgreichen "Exploit" (einem Login mit gültigen Token) eine komplette Linux-Shell simuliert. Da die Abgrenzung der Medium-Interaction-Honeypots nicht eindeutig ist, werden im Rahmen dieser Arbeit lediglich High- und Low-Interaction-Honeypots unterschieden.

2.2.2 Endpunkt

Während die Interaktivität eines Honeypots die simulierte Systemtiefe kategorisiert, bestimmt der Endpunkt dessen Rolle in seiner Kommunikation. Die Einteilung orientiert sich dabei am klassischen Client-/Server-Modell: Ein Server bietet einen Service oder Dienst an, der von einem oder mehreren Clients in Anspruch genommen wird. Dieses Modell trifft auf die wichtigsten Dienste im Internet zu, die dann wiederum auch von Honeypots angeboten werden. Eine derartige Kategorisierung ist daher naheliegend.

Sämtliche frühe Honeypot-Software, die im Laufe der 90er Jahre entstanden ist, fällt unter die Kategorie der **Server-Side-Honeypots**. Es handelt sich dabei im Falle von High-Interaction-Honeypots um Systeme mit Betriebssystem und Serversoftware, wie sie auch in der Praxis zum Einsatz kommt. Ein Beispiel hierfür ist ein in einer virtuellen Maschine installiertes, umfassend überwachtes GNU/Linux-System, auf dem der *apache*-Webserver installiert ist. Es kann von Angreifern über Scans gefunden oder über Suchmaschinen bewusst beworben werden und Schwachstellen in Form von Bugs in der Serversoftware selbst oder durch eine darauf laufende, unsichere Webanwendung offenbaren. Der Honeypot fungiert in diesem Falle als Server und wartet auf sich verbindende Clients, die die Dienste in Anspruch nehmen.

Da in der Vergangenheit Systeme überwiegend von außen durch bösartige Clients, wie beispielsweise sich selbst weiterverbreitende Würmer, kompromittiert wurden, konzentrierte sich die Honeypot-Entwicklung ursprünglich insbesondere auf die Server-Seite. Die hohe Verfügbarkeit von schnellen Breitband-Internetzugängen für die Bevölkerung und das flächige Ausstatten von Haushalten mit eingebetteten Routern mit integrierter Firewall haben allerdings den Effekt, dass viele potentielle Ziele wie beispielsweise nicht aktuelle Betriebssysteme, die standardmäßig gefährdete Dienste anbieten (wie Microsoft Windows Dateifreigaben mit SMB/CIFS), nicht mehr direkt über das Internet erreichbar sind. Dies hat zur Folge, dass sich die Angriffsvektoren der Angreifer in Richtung Client-Seite verschoben haben. Ziel ist nicht mehr das Kompromittieren eines Serverdienstes, sondern viel mehr das Anbieten von Services, die Sicherheitslücken der sich verbindenden Clients ausnutzen. Hauptangriffspunkt hierfür sind derzeit Web-Browser und im Zusammenhang mit diesen verwendete Plugins wie *Java*[4] oder *Adobe Flash*[5], da diese Software auf vielen gängigen Computersystemen zum Einsatz kommt.

Als Reaktion auf diesen Trend entstanden **Client-Side-Honeypots**. Es handelt sich hierbei um Software, die im Low-Interaction-Fall Client-Software wie Web-Browser mit bekannten Schwachstellen simuliert und verschiedene Web-Sites besucht. Falls diese auf eine Seite mit bösartigem JavaScript-Code trifft (eine gängie Variante von clientseitigen Exploits), können weitere Schritte wie das Herunterladen und Analysieren der vom Angreifer angebotenen Malware oder das Speichern der gesammelten Daten über

[4] `http://java.oracle.com`
[5] `http://adobe.com/flash/`

den Angriff vorgenommen werden. Es ist auch möglich, eine derartige Website sofort an eine öffentliche Blacklist weiterzuleiten, damit auch andere Internetnutzer von diesem Fund profitieren.

Wie auch schon zuvor beschrieben, können Low-Interaction-Clients nur bekannte Schwachstellen, die sie implementiert haben, erkennen. Zum Auffinden von Zero-Day-Exploits auf bösartigen Servern ist es folglich notwendig, auf High-Interaction-Client-Honeypots zurückzugreifen. Das sind im praktischen Einsatz virtualisierte Computer mit bewusst ausgewählter Clientsoftware (häufig auch älterer Versionen), die von außen gesteuert wird und der Reihe nach Verbindung zu einer Liste von Servern aufbaut. Sie ähnelt stark dem Konzept der Malware-Sandbox, sprich das virtuelle System selbst wird genauestens überwacht, um sofort auf unerwartete Veränderungen, die auf einen erfolgreichen Exploit hindeuten können, reagieren zu können. Dieses Setup ist dazu geeignet, bisher noch unbekannte Exploits zu finden, bedarf jedoch auch eines vielfach höheren Aufwands bei der Installation und im laufenden Betrieb. Eine detaillierte Beschreibung und eines modernen High-Interaction-Client-Side-Honeypots folgt in Kapitel 3.

Die Unterscheidung in Server- und Client-Side-Honeypots ist insbesondere wichtig, da diese sich in der Installation und im Betrieb massiv voneinander unterscheiden. Damit Server-Side-Honeypots eine möglichst gute Täuschung bieten, unterliegen sie ähnlichen Ansprüchen bei der Platzierung im Netzwerk wie auch produktive Systeme. Zum korrekten Betrieb müssen zudem Router so konfiguriert sein, dass sie ausgewählten Datenverkehr an Honeypots weiterleiten. Im Kontrast dazu können Client-Side-Honeypots auf jedem potentiellen Client, im Falle der meistgenutzten Protokolle also auf jedem beliebigen Rechner zum Einsatz kommen. Einzige Voraussetzung für den erfolgreichen Betrieb ist, dass ein solcher Client auch die zu untersuchenden Server erreichen kann. Trotzdem ist in beiden Fällen natürlich eine umfassende Absicherung und Überwachung der Honeypots erforderlich, damit diese bei einer Übernahme keinen weiteren Schaden anrichten können.

2.2.3 Sonstige

Es existieren sicherheitsrelevante Konzepte, die häufig im Zusammenhang mit Honeypots erwähnt werden oder zum Einsatz kommen, sich jedoch nicht ohne Weiteres in die herkömmlichen Kategorien einordnen lassen.

Ein **Network Telescope** ist ein passiver Sensor, der einen bestimmten IP-Adressbereich überwacht und Informationen über den an diese Adressen gerichteten Verkehr sammelt, oder, wie von David Moore et al. definiert[12]: „A network telescope is a portion of routed IP address space in which little or no legitimate traffic exists. Monitoring unexpected traffic arriving [...] provides the opportunity to view remote network security events [...]." Netzwerkteleskope, manchmal auch als *Internet Motion Sensor* bezeichnet, ähneln in ihrem Aufbau den Low-Interaction-Honeypots, arbeiten jedoch ausschließlich passiv - sie können lediglich Daten empfangen, aber nicht darauf antworten. Zum Einsatz kommt diese Technik beispielsweise, um auf Wurmaktivitäten innerhalb eines Unternehmensnetzwerk aufmerksam zu werden. Würmer durchsuchen das Netzwerk aktiv nach weiteren Hosts, um sich weiterverbreiten zu können und versuchen dazu Verbindung zu IP-Adressen aufzubauen, die nicht bereits an einen Host vergeben sind. Mit Hilfe eines Netzwerkteleskops kann eine solche Gefahr zumindest auf statistischer Ebene erkannt werden. Ebenso bietet es sich an, diese Technik zusammen mit normalen Honeypots einzusetzen, um die noch freien IP-Adressen zu überwachen und damit einen besseren Überblick über die Aktivitäten von Malware oder Angreifern nach erfolgreichen Einbrüchen in einen Honeypot zu erhalten. Da Netz-

werkteleskope keine Honeypots im herkömmlichen Sinne sind, wird im Rahmen dieser Arbeit auf eine Evaluation entsprechender Software verzichtet.

Hinter traditionellen Honeypots steht wie bisher beschrieben immer die Annahme, dass es sich um physisch oder virtuelle Systeme handelt. Dieses Konzept kann jedoch auch von Computersystemen im Speziellen auf *Ressourcen* verallgemeinert werden: Sogenannte **Honeytoken** sind digitale Entitäten, deren Nutzen im unrechtmäßigen Zugriff liegt. Lance Spitzner definiert[39] dies folgendermaßen:

> "(A honeytoken is) a honeypot that is not a computer. Instead it is some type of digital entity. Honeytokens come in many shapes or sizes, however they all share the same concept: a digital or information system resource whose value lies in the unauthorized use of that resource."

Honeytoken können folglich nahezu beliebige Gestalt annehmen: Bilder, Textdokumente, Einträge in Datenbanken, E-Mails usw. Die Idee nach Erstellen des jeweiligen Tokens ist, dass im Normalfall niemand darauf zugreifen sollte. So dürfte beispielsweise eine Patientenakte mit einem frei erfundenem Namen, die zu keinem realen Patienten gehört, nie abgerufen werden. Falls ein solcher Zugriff trotzdem erfolgt, deutet dies auf ein Problem hin, wie beispielsweise einen möglichen Einbruch in das System oder eine Verletzung der Richtlinien seitens der Angestellten („*The Insider Threat*"). Honeytoken sind zudem sehr flexibel. Mit geschickt verteilten Token ist es möglich, nicht nur Einbrüche an sich feststellen zu können, sondern auch Rückschlüsse auf den Angreifer und seine Motive zu ziehen. Dies wäre beispielsweise möglich, indem in den E-Mail-Postfächern der Chefetage eines Unternehmens gezielt E-Mails mit sensiblen Daten platziert werden, darunter Hyperlinks zu finanziellen Daten, Firmengeheimnissen usw. Durch Individualisierung der Links für jeden einzelnen E-Mail-Adressaten kann festgestellt werden, welches E-Mail-Konto kompromittiert wurde und woran der Angreifer interessiert ist.

Honeytoken eignen sich somit insbesondere für den internen Unternehmenseinsatz und zeichnen sich durch ihre Einfachheit aus. Sie verursachen keine Kosten, sind im Vergleich zu Software-Honeypots leicht zu installieren und zu überwachen und liefern trotzdem wertvolle Daten. Selbstverständlich ist ihr Einsatzbereich limitiert, weshalb sie als Ergänzung zu anderen Honeypot- und Sicherheitslösungen eingesetzt werden sollten und nicht als Alternative.

2.3 Einsatz

Beim praktischen Einsatz von Honeypots stellt sich neben der Auswahl der passenden Honeypot-Software, die individuell anhand der Gegebenheiten des zu schützenden Netzwerks und zu erwartenden Datenverkehrs abzuschätzen ist, die Frage, wie und an welcher Stelle die Systeme in der Netzwerkhierarchie installiert werden sollen. Dieses Kapitel beleuchtet die verschiedenen Möglichkeiten, die Honeypot-Administratoren zur Verfügung stehen und gibt einen Überblick über die Vor- und Nachteile des jeweiligen Setups.

2.3.1 Sicherheitsaspekte

Für die Installation von serverseitig operierenden Honeypots bieten sich eine Reihe von Möglichkeiten, die sich insbesondere in der Menge und Art der gesammelten Daten unterscheiden. In jedem Fall

ist der Abschirmung des Honeypots/Honeynets vom Produktivnetzwerk dabei besondere Aufmerksamkeit zu schenken, um den von Angreifern anrichtbaren Schaden zu minimieren. Diese Aufgabe kann im einfachsten Fall direkt vom Gateway und der darauf laufenden Firewall übernommen werden. Einfache Hardwarelösungen mit integrierter Firewall, die häufig beim Anmelden eines neuen Internetanschlusses mitgegeben werden, bieten unter Umständen nicht ausreichend Konfigurationsoptionen, um ausgehende Verbindungsversuche von den Honeypots in das Internet zu limitieren. Eine professionelle Hardwarelösung oder ein zu diesem Zweck eingerichteter handelsüblicher PC bieten hierfür deutlich mehr Möglichkeiten, da so beispielsweise das sehr mächtige, im Linux-Kernel integrierte *Netfilter*-Framework genutzt werden kann.

Um diese Aufgabe zu vereinfachen, wurde im Rahmen des Honeynet-Projekts die **Honeywall** entwickelt [40]. Das ist eine CentOS-basierte Linux-Distribution, die als transparente Bridge zwischen einem Netzwerk von Honeypots und dem Produktivnetzwerk fungiert. Ein Angreifer hat also von einem übernommenen Honeypot aus nicht die Möglichkeit festzustellen, dass er sich in einem abgetrennten Netzwerksegment befindet. Die Honeywall nutzt eine Kombination von Softwarelösungen zur Erfüllung ihrer Aufgabe, dem Protokollieren der Kommunikation mit den Honeypots und dem Absichern des restlichen Netzwerks vor Angriffen durch kompromittierte Systeme:

- Das Network-Intrusion-Detection-System **Snort**, bzw. das darauf basierende Tool `snort-inline`, das es ermöglicht, Firewall-Regeln auf Snort-Basis zu erstellen. Diese Software läuft passiv im Hintergrund und untersucht jedes weitergeleitete Paket auf bekannte gefährliche Signaturen. Falls solche gefunden werden, kann es die betreffenden Pakete verwerfen oder zumindest mitloggen.
- Die vom Linux-Kernel bereitgestellte Firewall **netfilter**. Bei der Installation der Honeywall können Regeln angelegt werden, die den Durchsatz (also beispielsweise die Anzahl der Verbindungsversuche, die ein Honeypot ins Internet unternehmen darf) für bestimmte Protokolle limitieren und somit verhindern kann, dass ein Honeypot als Teil einer DoS- oder DDoS-Attacke zum Einsatz kommt. Zudem ist es auch möglich, Datenverkehr mit dem üblicherweise in einem separaten IP-Adressbereich liegenden Produktivnetzwerk völlig zu verbieten, um dieses zusätzlich abzusichern. Da alle Daten, die von außen in das Honeynet gelangen oder es verlassen als verdächtig einzustufen sind, erlaubt netfilter das Aufzeichnen aller Pakete.
- **hflow2**[6], ein Tool zum Sammeln und Archivieren von Daten aus verschiedenen Quellen zur späteren Analyse. **sebek**[7] ist ein Server-Side-High-Interaction-Honeypot, der Informationen über Systemaufrufe über das Netzwerk versendet - wenn dieser auf den Honeypots zum Einsatz kommt, loggt hflow2 diese Daten automatisch mit und wertet sie soweit möglich aus. **p0f**[8] ist ein Toolkit zum passiven OS-Finterprinting[9] von TCP/IP-Kommunikationssträngen. Es sammelt Informationen über Angreifer und gibt diese an hflow2 weiter.
- Alle von Snort, hflow2 und netfilter gesammelten Daten werden in einer Datenbank gespeichert und können mit **Walleye** betrachtet werden, einer interaktiven Webanwendung, die auch Teil der Honeywall-Distribution ist. Hiermit ist die Auswertung der von sebek übermittelten Daten und auch vollständiger TCP-Sessions besonders komfortabel. Zusätzlich werden Statistiken und Graphen generiert, die einen detaillierten Überblick über die Aktivität im Honeynet geben. Zur Nutzung von Walleye ist jedoch ein drittes Netzwerkinterface zusätzlich zu den beiden überbrückten Interfaces, die für die Grundfunktion der Honeywall als Netzwerkbrücke notwendig sind, erforder-

[6] `https://projects.honeynet.org/hflow/`
[7] `https://projects.honeynet.org/sebek/`
[8] `http://lcamtuf.coredump.cx/p0f3/`
[9] Die Identifizierung von Betriebssystemen anhand ihres Netzwerkverhaltens (Quelle: Wikipedia)

lich. Über dieses kann das System zudem gefahrlos administriert werden.

Die Honeywall ist somit eine flexible und praktische Lösung und insbesondere als Teil einer Installation, die High-Interaction-Honeypots umfasst, in Erwägung zu ziehen. Leider scheint das Honeywall-Projekt selbst inaktiv zu sein. Die letzte veröffentlichte Version ist *roo 1.4* aus dem Jahr 2009. Das als Basis dienende CentOS ist in der enthaltenen Version inzwischen veraltet und es kann zu Problemen beim Einsatz dieser Software auf moderner Hardware kommen. Im Rahmen dieser Arbeit wurde die Honeywall deshalb virtualisiert und in Form einer virtuellen Maschine als Bridge vor den restlichen Honeypots platziert. Da, wie zuvor bereits ausgeführt, virtuelle Honeypots den physischen gegenüber viele Vorteile bieten, integriert sich die Honeywall sehr gut in das restliche System: Sowohl das Honeynet als auch die schützende Honeywall basieren auf virtuellen Maschinen, die über virtuelle Netzwerkinterfaces miteinander kommunizieren. Ein komplettes Honeypot-Setup lässt sich somit auf einem einzelnen, entsprechend leistungsstarken Host realisieren.

Während sich das Honeywall-Projekt auf die Eindämmung der von insbesondere von High-Interaction-Honeypots ausgehenden Risiken auf Netzwerkebene spezialisiert hat, existieren zusätzlich noch eine Reihe weiterer Sicherheitsmaßnahmen, die sich speziell für den Einsatz zusammen mit Low-Interaction-Honeypots eignen. Die von diesen Honeypots ausgehenden Gefahren sind zwar geringer, da sie nicht komplett von einem Angreifer übernommen werden könnnen, jedoch sind Sicherheitsprobleme und Bugs in der Honeypot-Software selbst nicht auszuschließen. Aus diesem Grund ist es ratsam, derartige Systeme auf separaten Hosts laufen zu lassen und mindestens eine der folgenden Eingrenzungstechniken einzusetzen:

- Das Einrichten einer **chroot**-Umgebung in Linux- oder BSD-basierten Umgebungen stellt ein wirkungsvolles Mittel dar, um einen risikobehafteten Prozess einzuschließen. Die Software wird in einer Umgebung mit einem individuellen, minimalem Root-Dateisystem gestartet und kann somit im Falle eines Exploitversuchs keine Dateien und Verzeichnisse außerhalb dieses Containers sehen oder gar modifizieren. Ein Angreifer ist, falls er den Honeypot-Prozess kompromittieren kann, also in der chroot-Umgebung „gefangen“ und hat kaum Möglichkeiten, mehr Schaden anzurichten. Diese Technik ist auf unixoiden Betriebssystem nativ verfügbar.
- Das von Niels Provos entwickelte Tool **systrace** verfolgt einen etwas anderen Ansatz: Es überwacht die Systemaufrufe, die ein gestarteter Prozess tätigt und lässt nur solche zu, die in einer Whitelist definiert wurden. Da eine korrekte Konfiguration dieser Software stark von der abzusichernden Honeypot-Software abhängig und demzufolge recht aufwändig ist, unterstützt das Tool auch einen „Lernmodus“, in dem alle Systemaufrufe, die eine gestartete Anwendung im normalen Betrieb tätigt, automatisch in die Whitelist aufgenommen werden. Falls im anschließenden praktischen Einsatz nicht explizit erlaubte Systemaufrufe getätigt werden, ist dies ein Indiz für einen möglicherweise erfolgreichen Exploitversuch auf die Honeypot-Software selbst. Diese verbotenen Systemaufrufe werden dann von systrace geblockt und eine entsprechende Warnmeldung generiert.

Falls die zuvor genannten Optionen aus bestimmten Gründen nicht eingesetzt werden können, kann natürlich immer noch auf Virtualisierung zurückgegriffen werden. Virtuelle High-Interaction-Honeypots profitieren implizit von der Absicherung durch die Einkapselung der Software in eine virtuelle Maschine. Auf genau dieselbe Art können auch Low-Interaction-Honeypots in auf virtuellen Maschinen laufenden Betriebssystemen zum Einsatz kommen, um Risiken zu minimieren. Der Ausbruch aus diesen Umgebungen ist nicht unmöglich, wie sich in der Vergangenheit bereits gezeigt hat: So erlaubte 2008 ein Fehler in der weitverbreiteten Virtualisierungslösung *VMware* einen Ausbruch aus dem Gastsystem [21].

2.3.2 Integration ins Netzwerk

Die am häufigsten anzutreffende Art der Installation ist direkt hinter einem Internet-Gateway. Es gibt in diesem Fall die Möglichkeit, ausgewählten Traffic gezielt an einen hinter einer separaten Firewall platzierten Honeypot oder auch in ein gesondertes Honeynet weiterzuleiten (das im Idealfall das Produktivnetzwerk widerspiegelt). Alternativ dazu kann ein Honeypot auch in der DMZ eines Unternehmens betrieben werden. In beiden Fällen ist eine Rekonfiguration des Internet-Gateways erforderlich, an der Struktur des verbleibenden Netzwerks ändert sich jedoch nichts. Dieses Setup ist für viele Anwendungsfälle unabhängig vom eingesetzten Honeypot-Typ gut geeignet: Um Malware einzufangen, die Wurmaktivität und das sog. „Hintergrundrauschen“ des Internets (alle Portscans und Verbindungsversuche an einer IP) zu analysieren oder Wurmaktivitäten im Netzwerk zu beobachten. Zero-Day-Exploits können mit diesem Setup ebenso gefunden werden [7].

Alternativ dazu bietet sich das Aufsetzen eines Honeypots im internen Netzwerks einer Organisation an. Die Absicherung durch die zuvor erwähnten Maßnahmen ist in diesem Fall besonders wichtig, da ein kompromittiertes System direkt mit den Produktivsystem kommunizieren kann. Dieses Risiko birgt aber zugleich den Vorteil, dass das den Honeypot umgebende Netzwerk für einen Angreifer besonders realistisch aussieht und sich die Wahrscheinlichkeit erhöht, dass er länger auf dem Honeypot verweilt und mehr nützliche Daten liefert. Das Platzieren eines Honeypots im internen Netzwerk hat zudem den Vorteil, dass Angriffe, die nicht aus dem Internet, sondern von Hosts aus dem lokalen Netzwerk kommen, analysiert werden können. Auf diese Art können beispielsweise mit Computerwürmern infizierte Rechner aufgespürt werden.

Im Falle eines solchen Setups muss darauf geachtet werden, dass der Honeypot den regulären Betrieb des Netzwerks nicht behindert. Hierzu sollte er entweder in einem separaten Segment des Netzwerks (z.B. einem eigenen IP-Adressbereich) oder zumindest mit einer bisher nicht genutzten IP-Adresse platziert werden. Damit keine false-positives auftreten, darf der Honeypot von anderen Geräten im internen Netzwerk nicht kontaktiert werden.

Das Honeynet-Projekt präsentierte zusammen mit der zuvor beschriebenen Honeywall-Live-CD auch verschiedene Architekturen zur Integration von Honeypots in ein bestehendes Netzwerk. Sie unterscheiden dabei zwischen Honeynets der ersten, zweiten und dritten Generation, die zugleich auch mit den Entwicklungsschritten der Honeywall identisch sind [34]:

1st Generation Honeynet : Ein oder mehrere Honeypots hinter einer auf OSI-Layer 3 operierenden Firewall, also einem regulären Router. Mit Hilfe von *Network Address Translation* (NAT) werden ausgewählte Pakete an die Honeypots weitergeleitet, während der restliche Datenverkehr normal in das produktive Netzwerk gelangt. Die separate Firewall vor dem Honeynet protokolliert und limitiert ein- und ausgehende Verbindungen. Die Architektur enthält Mittel zur Datenflussüberwachung (*Data Control*), zum Sammeln von Daten (*Data Capture*) und zur Benachrichtigung des Administrators im Falle eines Einbruchs (*Alerting*).

2nd Generation Honeynet : Der Aufbau ähnelt dem Honeynet erster Generation, allerdings operiert die Firewall vor den Honeypots nun auf dem zweiten OSI-Layer. Dies hat den großen Vorteil, dass sie aus Sicht der Honeypots (und damit auch für Angreifer) nciht wahrnehmbar ist. Außerdem wurde das Intrusion-Detection-System Snort als zusätzliche Sicherheitsmaßnahme hinzugefügt [14].

3rd Generation Honeynet : Die wesentliche Neuerung in dieser Generation ist die Möglichkeit, die

gesammelten Daten an zentraler Stelle zu analysieren und Informationen von verschiedenen Quellen miteinander in Beziehung zu setzen. In der Praxis bedeutet dies die Einführung des **Walleye**-Webinterfaces.

Sowohl für einen großflächigen Einsatz von Honeypot-Technologien im Unternehmensnetzwerk, als auch für einen weltweit verteilten Honeynet-Einsatz ist es möglich, ein sog. *Network of Sensors* [7] zu errichten. Hierzu werden beispielsweise verschiedene Honeypotsysteme, die je nach Anforderungen auch aus Kombinationen verschiedener Low-Interaction-Honeypots bestehen, verteilt aufgestellt und die entstehenden Daten an zentraler Stelle zur einfacheren Analyse gesammelt. Alternativ dazu können auch einfache Sensoren verwendet werden, die alle eingehenden Daten lediglich über einen Tunnel (VPN, GRE etc.) an eine zentrale Instanz weitergeben, auf der die eigentliche Honeypot-Software läuft. Der Unterschied zwischen einem solchen Sensornetzwerk und einem Honeynet ist, dass die Sensoren an verschiedensten Stellen in einem oder mehreren Netzwerken aufgestellt werden und nicht zwangsweise direkt miteinander kommunizieren können, also autonom sind. Dieses Setup ermöglicht es einer Organisation, einen Überblick über Gefahren wie Wurmaktivitäten im ganzen Netzwerk zu erhalten und nicht nur an einer konzentrierten Stelle, wie es bei den zuvor erwähnten Konfigurationen der Fall wäre. Ein solcher Ansatz wurde zudem für den praktischen Teil dieser Arbeit gewählt, worauf an späterer Stelle im Detail eingegangen wird.

Im Gegensatz zu Server-Side-Honeypots ist die Integration von Client-Side-Honeypots bedeutend einfacher. Damit ein solches System grundsätzlich funktioniert, muss es die zu kontaktierenden Server über das Netzwerk erreichen können. Folglich können diese relativ frei im Netzwerk platziert werden, es müssen allerdings trotzdem auf entsprechende Containment-Maßnahmen ergriffen werden, um zu verhindern, dass von kompromittierten Systemen weitere Angriffe ausgehen. In der Praxis hat es sich bewährt, mehrere virtuelle High-Interaction-Client-Honeypots auf ein- und derselben Maschine laufen zu lassen, um vorhandene Ressourcen besser nutzen und verschiedene Server parallel kontaktieren zu können.

Ein prominentes Beispiel für ein effektives Setup stammt von Microsoft aus dem Jahr 2005: der Honeypot **„HoneyMonkey“** [43]. Es handelte sich dabei um mehrere Instanzen von Windows-basierten High-Interaction-Client-Honeypots, die mit verschiedenen Update-Snapshots versehen waren: Es wurden frisch installierte Systeme ohne Updates, teilweise mit Updates versorgte und auch vollständig aktuelle Systeme benutzt, um mit dem Internet Explorer automatisch Websites zu besuchen und Exploits zu finden. Der Prozess war insofern vollständig automatisiert, als dass die virtuellen Instanzen eine große Liste von URLs abarbeiteten und regelmäßig auf einen umkompromittierten Zustand zurückgesetzt wurden. Entscheidend für die Effizienz dieser Art von Honeypots war die Technik, mit der versucht wurde festzustellen, ob ein Exploit stattfand. Microsoft verwendete hierzu den sogenannten *Strider Flight Data Recorder* (FDR), um jede Veränderung am Dateisystem und an der Registry aufzuzeichnen. Zusätzlich überprüfte der *Strider Gatekepper*, ob neue Prozesse für den automatischen Systemstart registriert wurden. Der *Strider GhostBuster* konnte zudem Malware entdecken, die sich vor anderen Prozessen zu verstecken versuchte. Nach jedem Besuch einer Website wurde der Systemzustand ausgewertet und im Falle eines erkannten Exploits ein Alarm ausgelöst.

2.3.3 Spezialfälle

Wie später in Kapitel 3 noch gezeigt werden wird, existieren Sonderformen von Honeypots, für deren Einsatz zusätzliche Maßnahmen ergriffen werden müssen. Dies trifft beispielsweise auf eine ganze Reihe

von Software zu, die sich als **Web Application Honeypots** zusammenfassen lässt. Dies sind Webanwendungen, die nach Möglichkeit auf ganz normalen, viel besuchten Webservern eingesetzt werden. Ziel ist es, versteckte, nur für Bots (oder Quellcode lesende Individuen) Hyperlinks, die auf die Honeypot-Webanwendungen zeigen, auf etablierten Websites zu platzieren. Die Websites, die der Honeypot bereitstellt, simulieren Varianten von realen Webanwendungen, die bekannte Schwachstellen aufweisen. Da die Hyperlinks in normalen Browsern unsichtbar sind, würden normalerweise nur Suchbots, die den Quellcode der Website parsen und alle Links absuchen, auf die täuschende Webanwendung stoßen. Wenn der Honeypot im Index einer großen Suchmaschine gelandet ist, werden Angreifer, die über spezifische Suchbegriffe nach verwundbaren Webanwendungen suchen, früher oder später auch auf diesen stoßen und versuchen, die vermeintliche Anwendung zu kompromittieren. Ein Web Application Honeypot filtert die Spider der Suchmaschinen heraus und protokolliert anschließend alle Aktionen eines Angreifers und bietet je nach Implementierung geringfügige Interaktionsmöglichkeiten, um mehr Daten sammeln zu können. Beispiele für derartige Honeypots sind der **Google Hack Honeypot** und **PHP.HoP**. Bei der Integration dieser Honeypots ist der Einsatz auf einer etablierten Website mit einem hohen Suchranking vorteilhaft, um in der Liste der Suchergebnisse der Angreifer möglichst weit oben zu erscheinen.

Eine andere Sonderform sind Honeypots, deren Einsatzbereich nicht Angriffe über große Netzwerke wie WANs oder das Internet sind, sondern die auf lokaler Ebene eingesetzt werden. Der **Ghost USB Honeypot** ist eine solche Software, die lokal auf einem Windows-System installiert wird und einen eingesteckten USB-Stick simuliert, um sich über USB weiterverbreitende Malware einzufangen. Eine Integration eines solchen Produktes in die Sicherheitsinfrastruktur eines Unternehmens würde bedeuten, den Honeypot auf allen Client-Rechnern, die hinsichtlich USB-Sticks (beispielsweise von Mitarbeitern) ein Risikopotential bieten, zu installieren. Ähnlich verhält es sich mit dem Bluetooth Honeypot **Bluepot**[10], der über Bluetooth-Hardware eine Station simuliert, die alle an sie geschickten Dateien akzeptiert und abspeichert.

2.4 Rechtliche Aspekte

Beim Einsatz von Honeypots kommen rechtliche Aspekte sowohl für den Angreifer als auch für den Honeypot-Betreiber selbst zum Tragen. Dieses Thema ist in sich so komplex, dass eine detaillierte Betrachtung an dieser Stelle aufgrund der begrenzten Zeit nicht durchgeführt werden kann. Stattdessen wird das Gebiet überblicksweise beleuchtet und dem Leser die Konsultation eines auf diese Sachverhalte spezialisierten Rechtsanwalt nahegelegt. Die Informationen in diesem Kapitel basieren im Wesentlichen auf den von Ralf Hofstetter in [8] und Sebastian Reitenbach in [35] zusammengetragenen Fakten und stellen die Situation in Deutschland dar.

Ein Angreifer, der auf welche Art auch immer ein Computersystem kompromittiert, führt damit eine **Datenveränderung** nach §303a StGB durch:

> "Jegliches rechtswidriges Verändern, Löschen, Unterdrücken oder Unbrauchbarmachen fremder Daten ist nach dem deutschen Strafrecht eine Datenveränderung."

Schwerere Störungen von Systemen, darunter auch *Denial of Service*-Angriffe[11], erfüllen den Tatbestand

[10] `http://code.google.com/p/bluepot`
[11] Gezielte Überlastung von Infrastruktursystemen in der Datenverarbeitung (Quelle: Wikipedia)

der **Computersabotage** nach §303b StGB. Diese Regelungen umfassen sowohl den privaten, als auch den betrieblichen Bereich. Weiterhin sind für Angreifer §202a StGB, „*Ausspähen von Daten*“ und der sogenannte „Hackerparagraph“ nach §202c StGB relevant, die in der Summe den Zugang zu fremden, gegen Zugriff besonders geschützte Daten und die Erstellung, Verwendung und Verbreitung von für diesen Zweck bestimmter Software unter Strafe stellen. Kriminelle müssen in diesen Fällen mit Geld- und sogar Haftstrafen rechnen.

Eine nur schwer zu klänrende Frage ist jedoch, ob in einem die ganze Welt umspannenden Netz wie dem Internet überhaupt deutsches Recht angewendet werden kann. Gemäß Prof. Eric Hilgendorf [32] ist der Begehungsort der Tat entscheidend. Im Falle von Tätigkeitsdelikten[12] genügt es beispielsweise, den Handlungsort zur Bewertung heranzuziehen. Wenn dieser oder der Ort, an dem der Erfolg eingetreten ist, in Deutschland liegt, kann deutsches Strafrecht angewendet werden. Problematischer wird es, wenn Honeypots in globalem Rahmen in verschiedenen Ländern installiert werden. Es gelten dann die Rechtsgrundsätze der jeweiligen Staaten, worauf an dieser Stelle nicht weiter eingegangen werden kann.

Interessanter im Rahmen dieser Arbeit sind die für den Betreiber von Honeypots relevanten rechtlichen Aspekte. Laut **Strafrecht** nach §27 StGB kann ein kompromittiertes System, das zum Angriff auf Dritte verwendet wird, als Beihilfe zu einer Straftat bewertet werden. Unter Berücksichtigung der Tatsache, dass der Administrator Sicherheitsmaßnahmen, wie die an frührerer Stelle erwähnten Containment-Verfahren im Vorfeld angewendet hat, um derartiges zu verhindern, kann allerdings nicht von *vorsätzlicher Hilfeleistung* ausgegangen werden. Auch angesichts der unzähligen ungeschützten Systeme von Privatanwendern, die im Rahmen von Botnets ebenfalls zum Angriff auf Dritte eingesetzt werden, kann der Betrieb eines Honeypots nicht strafrechtlich verfolgt werden. Ähnliches kann über das **Zivilrecht** gesagt werden. Die *Verkehrssicherungspflicht* nach §§923 ff. GBG besagt:

> “Derjenige, der eine Gefahrenquelle schafft oder unterhält, hat die Pflicht, die notwendigen und zumutbaren Vorkehrungen zu treffen, um Schäden anderer zu verhindern.”

Da Honeypots normalerweise mit den zuvor beschriebenen Maßnahmen gut überwacht sind und sich die Fahndung erfahrungsgemäß gegen die Urheber von Angriffen, anstatt gegen Verteidiger richtet, braucht beim Betrieb von Honeypots nicht mit zivilrechtlichen Folgen gerechnet zu werden.

Für den Betreiber von Honeypots am Entscheidendsten ist das Schutzziel des **Datenschutzes**. Hierbei steht der Schutz persönlicher Daten vor Missbrauch im Vordergrund, um die Privatssphäre von Individuen zu schützen. Dies fußt auf dem *Recht auf informationelle Selbstbestimmung*. Es sind vier Fälle näher zu betrachten:

- Der **Angreifer** verliert das Recht auf den Schutz seiner Daten, wenn er sich nach den zuvor erwähnten Tatbeständen der Datenveränderung oder Computersabotage strafbar macht. Es ist in diesem Zusammenhang sogar sinnvoll, Daten über den Angreifer zu sammeln, da diese im Falle einer Ermittlung gegen ihn verwendet werden können.
- Personendaten von **Unbeteiligten** unterliegen hingegen dem Datenschutzrecht und sind entsprechend zu schützen. Die durch einen Honeypot angestrebte Täuschung wirkt realistischer, wenn ein Angreifer darauf vertrauenswürdig aussehende Daten vorfindet. Da der Honeypot-Betreiber dazu verpflichtet ist, personengezogene Daten Dritter zu schützen, dürfen keine originalen Daten (wie

[12] Delikt, bei dem der Tatbestand bereits durch das im Gesetz beschriebene Tätigwerden erfüllt wird (Quelle: juraform.de)

Beispielsweise eine Kundendatenbank eines Unternehmens) darauf abgelegt werden, um Angreifer anzulocken. Natürlich spricht gleichzeitig nichts dagegen, frei erfundene persönliche Daten in diesem Zusammenhang zu verwenden. Je nachdem, wie leicht diese nachprüfbar sind, kann dies einem Angreifer aber auch einen Hinweis auf die Täuschung geben, der er unterliegt.

- Falls ein Angreifer selbst personenbezogene Daten Dritter auf einem Honeypot hinterlässt, sollten diese Informationen nach Möglichkeit anonymisiert oder entfernt werden.
- Es ist individuell zu untersuchen, welche potentiell persönliche Daten ein Honeypot speichert. Ob beispielsweise die von nahezu jeder Honeypot-Software gespeicherten IP-Adressen personenbezogene Daten sind, ist im deutschen Recht umstritten [16]. Da zumindest feststeht, dass statische IP-Adressen personenbezogen sind und es für den Honeypotbetreiber nicht ohne Weiteres festzustellen ist, ob eine vom Honeypot gespeicherte IP-Adresse statisch oder dynamisch vergeben wurde, muss im Zweifel davon ausgegangen werden, dass es sich um eine personenbezogene Information handelt. Es ist folglich sicherzustellen, dass die aufgezeichneten Daten bestmöglich gegen Diebstahl gesichert oder im Idealfall sogar anonymisiert sind.

2.5 Hybride Verfahren

Die in Kapitel 2.2 beschriebene Klassifikation von Honeypots macht deutlich, dass aktuelle Honeypots für verschiedene, teilweise stark spezialisierte Einsatzzwecke entwickelt werden und somit unterschiedliche Vor- und Nachteile mit sich bringen. Systeme, die existierende Honeypot-Technologien vereinen, also Low- und High-Interaction-Honeypots miteinander kombinieren, werden als *Hybride Honeypots* bezeichnet [33]. Sie zielen darauf ab, die Skalierbarkeit und Performance von Honeypots zu erhöhen und große Netzwerke umfassend mit diesen auszustatten. Sie unterscheiden sich von den zuvor genannten Architekturen beispielsweise dadurch, dass Honeypots nicht mehr nur an einer einzelnen Stelle im Netzwerk platziert werden. Um etwa ein vollständiges Klasse-B-Netz mit über 65000 IP-Adressen großteils oder ausschließlich mit Honeypots auszustatten, um Wurmaktivitäten in diesem Adressbereich zu analysieren, wäre eine Installation von unzähligen einzelnen virtuellen oder gar physischen Systemen hinsichtlich des Aufwands nicht vertretbar. Durch die geschickte Kombination von Low- und High-Interaction-Honeypots ist es jedoch möglich, dieses Setup zu vereinfachen: So könnten beispielsweise wenige, gut skalierende Low-Interaction-Honeypots alle neuen Verbindungen entgegennehmen, analysieren und nur unbekannte oder anderweitig als interessant klassifizierte Pakete an einige wenige High-Interaction-Honeypots weiterleiten. Weiterhin unterscheidet man auch bei hybriden Honeypots wie schon zuvor zwischen Server- und Client-basierten-Systemen.

Ein Beispiel für ein solches hybrides, serverseitiges System ist **Collapsar**, das 2006 von Xuxian Jiang und Dongyan Xu entworfen wurde [9]. Ihre Grundidee ist, Traffic von möglichst vielen IP-Adressen gleichzeitig verarbeiten zu können. Hierfür entwickelten sie eine Architektur bestehend aus mehreren Sensoren, die autonom in verschiedenen Netzwerken platziert werden und GRE-Tunnel[13] zu einem zentralen Server-Frontend aufrechterhalten, an das sie jeglichen eingehenden Datenverkehr weiterleiten. Das Frontend ist wiederum ein komplexes System, das „Firewalling“, Containment und Routing aller Anfragen übernimmt. Es hat die Aufgabe, eingehende Verbindungen an eine Farm von virtuellen High-Interaction-Honeypots weiterzuleiten und deren entsprechende Antworten wieder zu den Sensoren unter Berücksichtigung von Containment-Maßnahmen zurückzuschicken. Eine Besonderheit dieses Systems ist, dass die

[13] **G**eneric **R**outing **E**ncapsulation: Netzwerkprotokoll, das dazu dient, andere Protokolle einzukapseln und in Form eines Tunnels über das IP-Protokoll zu transportieren (Quelle: Wikipedia)

virtuellen Honeypots an die spezifischen „Produktionsnetzwerke“ angepasst sind, in denen die Sensoren stehen, von denen sie Anfragen erhalten. Dies geschieht durch Rekonfiguration der jeweiligen Gateway- und DNS-Adressen in den virtuellen Maschinen. Auf der Seite des Frontends werden viele Containment-Maßnahmen ergriffen, um zu verhindern, dass von Angreifern übernommene Honeypots zusätzlichen Schaden anrichten können: Umfassendes Logging, Tarpitting[14] und ein sog. *Correlation Module*, das den Netzwerkverkehr aufzeichnet und nach DDoS-Angriffen, Wurmaktivität und IRC-Verbindungen durchsucht.

Das **Collapsar**-System ist leider nicht frei verfügbar und wurde nur im Rahmen des von Jiang und Xu veröffentlichten Papers erwähnt und getestet. Inzwischen gibt es jedoch ein hybrides System, dass sich an der zuvor beschriebenen Architektur orientiert und unter einer OpenSource-Lizenz allen Interessierten zur Verfügung gestellt wurde: **SurfCERT IDS**[15] schickt eingehenden Traffic via VPN[16] miteinander verbundener Sensoren zu einem zentralen Server, der verschiedene High- und Low-Interaction-Honeypots als Endpunkte bereitstellt. Das clientseitige **HoneySpider Network 2**[17] erlaubt hingegen die umfassende Analyse von Ressourcen im Netz mit Hilfe von Verfahren zur statischen Analyse (JavaScript, PDF) und dem Einsatz verschiedener Honeypot-Technologien. Es ist auch möglich, ein derartiges System selbst zu entwerfen, indem bereits vorhandene Honeypot-Software, falls nötig mit selbstgeschriebenen Plugins, über gemeinsamen Schnittstellen miteinander kombiniert wird. Beispielsweise erlauben viele Honeypots das optionale Protokollieren zu einer MySQL-Datenbank, über die andere Software wiederum auf alle gewonnenen Daten Zugriff hat und zum Beispiel in einem Webinterface konsolidieren und aufbereiten kann.

[14] Verfahren, mit dem unerwünschte Netzwerkvebrindungen künstlich verlangsamt werden (Quelle: Wikipedia)
[15] `http://ids.surfnet.nl/`
[16] **V**irtual **P**rivate **N**etwork
[17] `http://www.honeyspider.org/`

3 Ausgewählte Software

Im Rahmen dieses Kapitels wird aktuelle Software zum Errichten von Honeypots und Honeynets vorgestellt. Es werden jeweils die theoretischen Konzepte und praktischen Aspekte, sowie, falls vorhanden, Erfahrungswerte mit dem Einsatz der jeweiligen Software präsentiert. Im Rahmen dieser Arbeit sollen sowohl etablierte, sowie auch neuartige und spezialisierte Honeypotkonzepte berücksichtigt werden. Aus diesem Grund wird zunächst auf Honeypots eingegangen, die herkömmliche Desktop- oder Serversysteme simulieren. Anschließend werden einige Konzepte vorgestellt, die versuchen, das Konzept der Täuschung auf anderen, neuartigen Gebieten anzuwenden.

Die Auswahl der vorgestellten Projekte richtet sich teilweise nach einer von der *ENISA*[18] durchgeführten Studie über die Nützlichkeit einer großen Anzahl von Honeypot-Projekten [7]. Obwohl der Fokus besagter Studie auf der Nützlichkeit der untersuchten Software für CERTs[19] liegt, sind trotzdem viele hilfreiche Informationen beispielsweise zur Qualität der gesammelten Daten, zur Skalierbarkeit oder zur Erweiterbarkeit enthalten, die auch aus einem akademischen Standpunkt interessant sind. Zudem versucht die Studie, die analysierten Honeypot-Projekte im Kontext zu betrachten: Der Einsatz mancher Software macht angesichts eines andere Produkts, das in jeder Hinsicht überlegen ist, keinen Sinn. Dieser zeitgemäße Überblick ist in den vielen anderen Quellen, die sich meist nur gezielt auf einen einzelnen oder sehr wenige Honeypots stützen und bereits ein paar Jahre alt sind, nicht gegeben.

3.1 Server-Side-Honeypots

Dieser Abschnitt beschäftigt sich zuerst mit einer Reihe von Low-Interaction-Server-Side-Honeypots, die sich in der Testphase dieser Arbeit als nützlich erwiesen haben. Einige dieser Projekte werden anschließend im Rahmen einer großflächigen Installation von Honeypots verwendet, auf die im vierten Kapitel näher eingegangen wird. Nach der Vorstellung von klassischen Produkten, die auf Computern weit verbreitete Serverdienste wie SSH oder HTTP anbieten, wird zudem auf einige speziellere Lösungen, u.a. zur Simulation von Industriesteuerungsanlagen, eingegangen.

Low-Interaction-Honeypots lassen sich gemäß ENISA[7] zusätzlich noch anhand der von ihnen angebotenen Dienste unterscheiden. So gibt es spezialisierte Software, die lediglich einen bestimmten Dienst (wie beispielsweise SSH oder HTTP(S)) anbietet (**Single Purpose Honeypot**) und im Kontrast dazu sog. **Multi-Purpose-Honeypots**, die verschiedene Dienste gleichzeitig offerieren. Auf die nachfolgenden Softwarelösungen, für die eine solche Zuordnung eindeutig ist, wird im jeweiligen Abschnitt gesondert hingewiesen.

3.1.1 kippo

Das **S**ecure-**S**hell-Protokoll (SSH) erlaubt es mit entfernten Rechnern über einen verschlüsselten Datenkanal zu kommunizieren. Das Protokoll ist der inoffizielle Nachfolger der *Remote Shell* (RSH), bei der die Kommunikation noch unverschlüsselt erfolgte. Für die Authentifizierung mit einem SSH-Server bie-

[18] European Network and Information Security Agency
[19] Computer Emergency Response Team

tet das Protokoll eine Reihe von Möglichkeiten: Neben traditionellen *Challenge-Response-Verfahren*, bei der ein gültiger Benutzername und das dazugehörige Passwort angegeben werden müssen, wird auch asymmetrische *Public-Key-Authentifizierung* unterstützt. Dieses Verfahren ist, wenn korrekt umgesetzt, sicherer als die Challenge-Response-Variante. Da SSH schon lange zum Industriestandard für die Remote-Administration von unixoiden Betriebssystemen gehört, existiert trotzdem eine große Zahl an Endgeräten, die die Authentifizierung mit Benutzername und Passwort zulassen. Dies stellt ein Sicherheitsrisiko da, dass auch aktiv ausgenutzt wird, wie leicht festzustellen ist, wenn man die Aktivität auf TCP Port 22 beobachtet, auf dem SSH standardmäßig angeboten wird. Wie in [28] zu sehen, versuchen Angreifer über alle IP-Adressbereiche hinweg SSH-Dienste aufzuspüren und dort über Dictionary- oder BruteForce-Angriffe unsichere Benutzername/Passwort-Kombinationen zu finden. Um mehr über diese Angreifer und die von ihnen verwendeten Angriffsversuche zu lernen, können spezialisierte SSH-Honeypots eingesetzt werden.

Bei kippo[20] handelt es sich um den derzeit beliebtesten und ausgereiftesten *Single Purpose* SSH-Honeypot. Die Entwickler ordnen die Software selbst als sogenannten **Medium-Interaction-Honeypot** ein. Diese Einteilung beruht auf der Tatsache, dass kippo nicht nur Verbindungs- und Loginversuche aufzeichnet, sondern dem Angreifer bei erfolgreichem Login auch eine interaktive, emulierte Shell zur Verfügung stellt, die Interaktion mit dieser aufzeichnet und somit deutlich größere Mengen an Daten über einen Angreifer gewonnen werden können. Die Software selbst ist komplett in Python geschrieben, wird als OpenSource veröffentlicht und steht unter der *New BSD License*. Das Projekt wird zu diesem Zeitpunkt aktiv weiterentwickelt und hat eine große Community hinter sich, die sich mit Skripten zur Datenauswertung und Erweiterungen zur Verbesserung der virtuellen Shell-Umgebung aktiv einbringt.

Installation und Konfiguration Wie viele andere Low-Interaction-Honeypots auch, ist ein kippo-Honeypot relativ schnell aufzusetzen. Da Python eine interpretierte Programmiersprache ist und in diesem Projekt ausnahmslos zum Einsatz kommt, entfällt die Notwendigkeit des separaten Kompilierens und Installierens. Wenn die auf der Website aufgelisteten Abhängigkeiten installiert sind, muss lediglich das Archiv mit der aktuellsten Version von der kippo-Website heruntergeladen, entpackt und die im Hauptverzeichnis gelegene Datei `start.sh` ausgeführt werden. Der Honeypot startet anschließend als Daemon im Hintergrund und verrichtet seine Arbeit. Zur Installation der nötigen Abhängigkeiten liefert das Wiki des Projektes[21] eine genauere Anleitung für Windows und populäre Linux-Distributionen.

Kippo lässt sich komfortabel über eine einzelne Konfigurationsdatei, `kippo.cfg` anpassen. Alle verfügbaren Optionen sind hier gut kommentiert und mit Beispielen versehen. Es ist möglich, kippo mehrere Interfaces und IP-Adressen gleichzeitig abhören zu lassen, um somit auf einfache Art und Weise ein größeres Netzwerk zu simulieren. Dies erfordert allerdings Modifikationen der Routing-Tabelle des zuständigen Routers oder den Einsatz einer alternativen Technik, wie beispielsweise *Proxy ARP*[22]. Zusätzlich lässt sich der Port, auf dem der Dienst gestartet werden soll, festlegen. Neben Parametern zur Konfiguration der virtuellen Shell-Session ist es außerdem noch möglich, verschiedene Adapter zur Protokollierung hinzuzufügen. Somit können die gesammelten Daten in Dateien auf dem Dateisystem abgelegt, an einen (*MySQL*)-Datenbankserver oder auch an ein hybrides System wie *SurfCERT IDS* gesendet werden.

[20] `http://code.google.com/p/kippo/`

[21] `http://code.google.com/p/kippo/w/list/`

[22] Der Router beantwortet ARP-Anfragen für mehrere IP-Adressen und leitet die Daten an den Honeypot weiter

Funktionsweise Die Hauptaufgabe von kippo ist die Simulation eines SSH-Dienstes, der ausschließlich Challenge-Response-Authentifizierung unterstützt. Kippo erstellt beim ersten Start ein neues Server-Schlüsselpaar, mit dem Clients sicherstellen können, dass sie tatsächlich mit dem gewünschten Server verbunden sind und nicht einer Täuschung unterliegen. Dieses in der SSH-Protokollspezifikation vorgeschriebene Sicherheitsmerkmal wird von kippo korrekt emuliert. Beim Versuch, sich mit einem SSH-Client mit dem Service zu verbinden, wird der Benutzer zuerst zur Eingabe eines Benutzernamens und anschließend eines Passworts aufgefordert. Wenn diese Zugangsdaten korrekt sind, kann mit einer einfachen simulierten, interaktiven Shell interagiert werden. Um das Setup realistischer zu gestalten und mehr Angriffsfläche zu bieten, was letztendlich die Wahrscheinlichkeit von qualitativ hochwertigen gesammelten Daten erhöht, können beliebige „gültige“ Zugangsdaten in der Textdatei `usersdb.txt` im unixtypischen *passwd*-Format hinterlegt werden. Aus Sicht des Angreifers ist es letztendlich egal, mit welchem Login ein Einbruchsversuch erfolgreich war - die anschließend emulierte Shell ist immer gleich.

Falls ein Angreifer eine gültige Kombination aus Benutzername und Passwort gefunden hat, bekommt er ein Shell-Prompt und ein Text-Banner angezeigt, wie es für Shell-Sessions je nach Systemkonfiguration üblich ist. Der simulierten Shell liegt ein virtuelles Dateisystem zugrunde, das in der Datei `fs.pickle` gespeichert ist. Es enthält lediglich Dateinamen, Pfade und Zugriffsberechtigungen, aber keine Dateiinhalte (Inhalte für ausgewählte Dateien können aber im `honeyfs`-Verzeichnis hinterlegt werden). Zusätzlich bietet die Shell eine Reihe von auf den meisten Linux-Systemen verfügbaren Standardkommandos wie `dmesg` und `mount` an und erlaubt es sogar, mit `wget` Dateien von entfernten Quellen herunterzuladen. Weitere Befehle können leicht durch das Anlegen von Textdateien im `txtcmds`-Verzeichnis hinzugefügt werden, das die Dateisystemstruktur des virtuellen Systems repräsentiert. So würde das Anlegen einer Textdatei `ps` in `txtcmds/bin/` mit dem Inhalt „test“ bewirken, dass auf der simulierten Shell der Befehl `ps` zur Verfügung steht und die Ausgabe „test“ liefert. An dieser Stelle werden schnell die Grenzen der Simulation sichtbar: Die Shell-Emulation ist sehr oberflächlich und eignet sich höchstens dazu, automatisierte Bots oder sehr unerfahrene Angreifer für eine gewisse Weile zu beschäftigen. Durch Individualisierung des virtuellen Dateisystems und der verfügbaren Befehle kann das Verweilen auf dem System eventuell hinausgezögert werden, erfahrenere Angreifer werden jedoch trotzdem schnell feststellen, dass sie einer Täuschung unterliegen und das System wieder verlassen.

Der Wert des kippo-Honeypots liegt neben dem Aufzeichnen von Login-Versuchen in den Protokollen der virtuellen Shell-Sessions. Da Angreifer normalerweise, nachdem Sie Zugriff auf ein System erlangt haben, zusätzliche Software wie Rootkits herunterladen, um ihre Rechte auszudehnen und eine Backdoor zu installieren, die den Zugriff zum Rechner aufrechterhält, ist das Sammeln eben solcher heruntergeladenen Malware Hauptziel der Software. Die Shell-Interaktion selbst ist zu primitiv, um daraus sonderlich wertvolle Informationen zu beziehen. In einigen Demo-Sessions, die von Angreifern auf produktiven Honeypots aufgezeichnet wurden, ist schnell zu erkennen, dass die Limitierungen der virtuellen Shell dem Angreifer kaum Freiräume lassen, um mehr wertvolle Daten zu liefern. Interessant ist noch ein weiteres Feature der virtuellen Shell: Das Kommando `exit` oder die normalerweise zum Schließen einer Session verwendete Tastenkombination `Strg+d` mit der Ausgabe „Connection to server closed.“ zu kommentieren, die Verbindung aber trotzdem nicht abzubrechen, kann einen unaufmerksamen Angreifer glauben lassen, sich wieder auf seinem lokalen System zu befinden. Er gibt dann möglicherweise durch die Befehle, die er im Folgenden eingibt, mehr Informationen über sich preis.

Da bis zum jetzigen Zeitpunkt keine Sicherheitslücken in kippo selbst bekannt sind und der Prozess mit normalen Benutzerrechen ausgeführt werden kann, ist das von diesem Honeypot ausgehende Risiko gering. Die Möglichkeiten der Shell-Sessions sind zu begrenzt, als dass ein Angreifer Schaden anrichten

könnte. Aufgrund der geringen mit ihrem Einsatz verbundenen Implikationen eignet sich die Software deshalb sehr gut zur Installation in Unternehmensnetzwerken.

Protokollierung und Auswertung Kippo zeichnet alle Kombinationen aus Benutzername und Passwort auf, mit denen versucht wurde sich einzuloggen. Diese Information wird je nach Konfiguration im Dateisystem abgelegt oder in eine Datenbank geschrieben. Wenn ein Angreifer korrekte Zugangsdaten angibt, wird die Interaktion mit der virtuellen Shell-Session in einem eigenen Binärformat aufgezeichnet und im Verzeichnis `log/tty` gespeichert. Mit der Honeypot-Distribution mitgeliefert werden Tools zum Auswerten dieser Binärdateien, darunter eines, das die Session auf einer Konsole interaktiv (ähnlich einem Video) wiedergibt. Dies eignet sich sehr gut dazu, die Aktionen, die ein Angreifer unternimmt, mental besser nachvollziehen zu können.

Die Logdateien enthalten neben den vom Angreifer verwendeten Login-Daten auch noch Informationen über den zum Einsatz kommenden SSH-Client und das Betriebssystem, die mit Hilfe des passiven Fingerprinting-Tools *p0f* gesammelt werden. Außerdem werden alle in der virtuellen Shell ausgeführten Befehle mit aufgezeichnet. Wenn der Eindringling Dateien mit `wget` herunterläd, wird eine Kopie im Log-Verzeichnis zur späteren Analyse vorgehalten.

Zur Auswertung und Analyse der gesammelten Daten gibt es einige von anderen Entwicklern bereitgestellte Tools und Skripte, die die Überwachung des Honeypots erheblich vereinfachen[23]. Darunter befinden sich mehrere Webanwendungen, die das interaktive Abfragen und Auswerten der Daten ermöglichen. Die bisher Beliebteste unter diesen ist **kippo-stats**[24], ein Perl-basiertes Webfrontend, das Verbindungsstatistiken über Zeiträume oder Trends (häufigste Benutzernamen und Passwörter) in grafischer Form aufbereitet. **kippo-graph**[25] ist hingegen dazu geeignet, eine große Anzahl von Daten, die über einen langen Zeitraum hinweg gesammelt wurden, zu visualisieren. Es erstellt in einem Durchlauf insgesamt 24 Graphen, die viele verschiedene Aspekte (beliebteste Clients/Benutzernamen/Passwörter, IPs, Verbindungen pro Land usw.) beleuchten. Da kippo-graph seine Daten nur von einem MySQL-Server beziehen kann, hat der Autor der Software zusätzlich noch das Skript **kippo2mysql**[26] geschrieben, mit dem zumindest die wichtigsten Informationen aus den Log-Dateien extrahiert und in eine Datenbank geschrieben werden können. Dies erlaubt die Nutzung reiner datenbankbasierter Auswertungstools, obwohl nur in Textdateien gespeichert wurde. Im Rahmen dieser Arbeit wurde davon gebrauch gemacht, was an späterer Stelle noch detaillierter erklärt werden wird.

Erweiterbarkeit Neben der bereits angesprochenen Option, neue Befehle mit statischer Ausgabe in Form von einfachen Textdateien zu hinterlegen, ist es auch möglich über eine Python-Schnittstelle interaktive Kommandos hinzuzufügen. Hierfür hat die aktive Community hinter kippo ein Repository[27] auf github eingerichtet, um die Standarddistribution um einige häufig verwendete Programme zu erweitern. Eine andere Erweiterungsmöglichkeit ist die Anpassung des virtuellen Dateisystems, worauf im vorherigen Kapitel bereits eingegangen wurde.

[23] `http://code.google.com/p/kippo/wiki/RelatedSoftware/`
[24] `https://github.com/mfontani/kippo-stats`
[25] `http://bruteforce.gr/kippo-graph`
[26] `http://bruteforce.gr/kippo2mysql`
[27] `https://github.com/basilfx/kippo-commands`

Zusätzlich ist es noch möglich, durch Definition einer weiteren in Python geschriebenen `DBLogger`-Klasse einen Log-Adapter zu schreiben. Kippo liefert bereits Adapter zum Protokollieren in lokale Textdateien, zu einem MySQL-Server oder über XMPP mit. Im Rahmen der großflächigen Installation von Honeypots in Kapitel 4 wurde diese Erweiterungsmöglichkeit genutzt und ein Modul zum Loggen zu einem *SurfCERT IDS*-Server integriert[28].

Experimenteller Einsatz Da kippo laut [7] der derzeit beliebteste, frei verfügbare SSH-Honeypot ist, wurde die Software im Rahmen dieser Arbeit in der Praxis erprobt. Ziel war die Analyse der SSH-Loginversuche, dem Systeme an einem privaten Internetanschluss in einem Wohngebiet auf dem SSH-Port 22 ausgesetzt sind. Hierzu wurde der Honeypot 14 Tage lang von jeweils ca. 10 bis 22 Uhr in Betrieb genommen, um einen normalen, an einem solchen Anschluss zu erwartenden Host zu simulieren. Zum Einsatz kam die kippo-Version 0.5. An der Standardkonfiguration wurde nur wenig modifiziert, lediglich der Server-Port auf 22 (TCP) festgesetzt und der Hostname vom standardmäßigen `sales` auf `saturn` geändert. Letzteres war eine Sicherheitsmaßnahme, um die Identifizierung des Honeypots geringfügig zu erschweren, da davon auszugehen ist, dass viele kippo-Honeypots in der unveränderten Standardkonfiguration mit dem Standard-Hostname betrieben werden - ein Angreifer könnte somit schnell ahnen, dass es sich um kein echtes System handelt. Das virtuelle Dateisystemen und die zur Verfügung stehenden Befehle wurden nicht erweitert, da es das Ziel des Experiments war, kippo in der Standarddistribution zu testen. Als einziges gültiges Login wurde der Benutzer `root` mit dem Passwort `123456` belassen.

Nach dem Ende des Experiments wurde zur Auswertung das bereits beschriebene, Web-basierte Tool *kippo-graph* benutzt. Es erfordert das Vorhandensein eines PHP-interpretierenden Webservers wie `apache2` und generiert eine Reihe von Graphen über die gesammelte Datenmenge. Da es Daten lediglich aus einer MySQL-Datenbank beziehen kann, wurden die nur als lokale Logdateien vorhandenen Protokolle zuvor noch mit *kippo2mysql* in die Datenbank geschrieben. Leider gingen bei dieser Transformation einige Daten verloren und es konnten schlussendlich nur einige wenige Graphen über Trends (z.B. häufigste Benutzernamen/Passwörter) generiert werden. Aus diesem Grund erfolgte die restliche Auswertung mit den GNU-Standardtools `awk`, `sed`, `sort`, `uniq` usw. durch manuelles Parsen der Logdateien.

Bei der **Auswertung** zeichnete sich das in Tabelle 1 gezeigte Bild ab.

Insgesamt wurden innerhalb des Testzeitraumes 4246 Loginversuche von insgesamt 42 verschiedenen Quell-IP-Adressen am Honeypot gezählt. Immerhin 20, also 69% der Angreifer, erlangten erfolgreich die gültigen Zugangsdaten und wurden mit der interaktiven, virtuellen Shell konfrontiert. Besonders auffällig ist die Unregelmäßigkeit der Angriffe: Während am vierten Tag ein einzelner Aggressor lediglich acht verschiedene Logins tätigte und dabei nicht erfolgreich war, wurde am 11. Tag das gemessene Maximum von ganzen 1973 Loginversuchen von sieben verschiedenen Hosts aus erreicht. Bei dieser großen Anzahl an Angriffen waren schlussendlich am selben Tag auch bis auf einen alle Angreifer erfolgreich. Am sechsten Tag griffen hingegeben ebenfalls sieben Hosts an, führten jedoch insgesamt nur 610 Loginversuche durch.

Tabelle 2 visualisiert die aus Datenschutzgründen anonymisierten IP-Adressen der Angreifer, die Anzahl der Loginversuche und das jeweilige Land, dem die Adresse zuzuordnen ist. Für diese Zuordnung wurden

[28] `http://ids.surfnet.nl/wiki/doku.php?id=kb:installing_kippo`

Tag	Loginversuche	Erfolgreiche Logins	Anz. Angreifer
1	121	2	5
2	290	0	2
3	24	0	1
4	8	0	1
5	137	0	2
6	610	3	7
7	56	1	1
8	376	2	5
9	100	1	5
10	144	0	2
11	1973	6	7
12	354	2	5
13	20	0	1
14	33	3	5

Tabelle 1: Gesammelte Daten des kippo-Honeypots

die WHOIS-Datenbank der *ARIN*[29][30] und der Geolocation-Service *IPLocation*[31] verwendet.

Angriffe kamen demzufolge aus insgesamt 12 verschiedenen Ländern, davon 48.5% aus China, $42,4\%$ aus Frankreich. Die verbleibenden 9.1% verteilten sich auf Südkorea, Venezuela, den Iran, die U.S.A., Deutschland, Kanada, den Vietnam, Italien und die Seychellen. Bemerkenswert ist, dass alle Loginversuche aus Frankreich ihren Ursprung in einer einzigen IP-Adresse hatten, während sich die chinesischen Angriffe auf 28 verschiedene Adressen verteilten. Außerdem fällt auf, dass einige Anfragen aus China von mehreren IP-Adressen eines einzelnen Netzwerksegments kamen, beispielsweise von vier verschiedenen Hosts aus dem Netzwerk 125.39.8.0/24. Es ist anzunehmen, dass die Angreifer dort ein vollständiges Netzwerk besitzen oder unter ihre Kontrolle gebracht haben und eine Art Load-Balancing betreiben, um das vorzeitige Blacklisten ihrer IPs beispielsweise durch lokale Gegenmaßnahmen wie *fail2ban* zu verhindern. Den meisten Hosts mit nur sehr wenigen Loginversuchen gelang ein Login bereits sehr schnell, was die wenigen gezählten Verbindungen erklärt. Der Host mit den zweitmeisten Logins aus China, 117.79.91.x, führte sogar einen langanhaltenden Angriff über acht Tage hinweg aus, jedoch ohne Erfolg.

Bedauerlicherweise ist es im Testzeitraum nicht gelungen, eine interaktive Shell-Session aufzuzeichnen. Die Ursache ist hier jedoch nicht in Softwarefehlern zu suchen, da das Protokollieren einer solchen Session zuvor mit verschiedenen SSH-Clients erfolgreich getestet wurde. Viel mehr liegt das am Verhalten der Angreifer: Alle 20 Eindringlinge, die die korrekten Zugangsdaten verwendeten, erhielten das Prompt der virtuellen Shell, `saturn:~$`, und schlossen anschließend die Verbindung. Dies lässt den Schluss zu, dass es sich bei den Angreifern um Bots gehandelt haben muss, die systematisch Systeme (oder ganze Netzwerke) scannen und angreifen, um im Falle eines Erfolgs die gültigen Zugangsdaten zu speichern. Eine Interaktion mit dem System zur weiteren Datengewinnung fand nicht statt, weshalb davon auszugehen ist, dass zu irgendeinem späteren Zeitpunkt ein menschlicher Angreifer manuell auf das System zugegriffen hätte.

[29] American Registry for Internet Numbers
[30] `http://whois.arin.net/`
[31] `http://www.iplocation.net/`

IP-Adresse	Logins	Ursprungsland	IP-Adresse	Logins	Ursprungsland
93.184.xxx.xxx	1801	Frankreich	125.39.xxx.xxx	25	China
117.79.xxx.xxx	769	China	219.235.xxx.xxx	20	China
60.171.xxx.xxx	302	China	123.49.xxx.xxx	17	Bangladesch
222.240.xxx.xxx	268	China	125.39.xxx.xxx	16	China
218.59.xxx.xxx	156	China	64.6.xxx.xxx	13	U.S.A.
210.44.xxxx.xxx	125	China	204.9.xxx.xxxx	8	Kanada
121.191.xxx.xxx	114	Südkorea	125.62.xxx.xxx	8	China
181.208.xxx.xxx	108	Venezuela	218.74.xxx.xxx	4	China
89.165.xxx.xxx	65	Iran	61.236.xxx.xxx	3	China
114.112.xxx.xxx	57	China	222.59.xxx.xxx	3	China
202.121.xxx.xxx	56	China	219.138.xxx.xxx	3	China
119.161.xxx.xxx	55	China	113.52.xxx.xxx	3	Vietnam
70.39.xxx.xxx	41	U.S.A.	60.10.xxx.xxx	2	China
202.103.xxx.xxx	40	China	218.104.xxx.xxx	2	China
61.155.xxx.xxx	32	China	182.71.xxx.xxx	2	China
125.39.xxx.xxx	32	China	166.111.xxx.xxx	2	China
218.26.xxx.xxx	31	China	82.221.xxx.xxx	1	Island
166.78.xxx.xxx	28	U.S.A.	37.182.xxx.xxx	1	Italien
124.160.xxx.xxx	28	China	211.142.xxx.xxx	1	China
125.39.xxx.xxx	27	China	193.107.xxx.xxx	1	Seychellen
46.20.xxx.xxx	25	Deutschland	120.209.xxx.xxx	1	China

Tabelle 2: Loginversuche und Ursprungsländer der Angreifer auf den kippo-Honeypot

Weiterhin soll die Angriffsmethode selbst noch genauer beleuchtet werden: Welche Benutzernamen und Passwörter haben die Angreifer verwendet? Sind Muster zu erkennen?

Abbildung 1 visualisiert die häufigsten Benutzernamen und Passwörter, die an den SSH-Honeypot gesendet wurden. Es ist zu erkennen, dass der mit Abstand häufigste Benutzername `root` ist. Angesichts der Tatsache, dass dieser Benutzer auf allen unixoiden Betriebssystemen der administrative Standardnutzer mit den höchsten systemweiten Rechten ist, ist dieser Umstand verständlich. Andere Benutzernamen orientieren sich an weiterbreiteten proprietären und freien Produkten, so richtet *Oracle* beispielsweise auf seinen Systemen einen gleichnamigen Benutzer ein, `support`, `git` oder `nagios` beschreiben Versuche, Wartungs- und Verwaltungsaccounts zu kompromittieren.

Anhand der Liste der meistverwendeten Passwörter lässt sich schnell erkennen, dass gegen den Honeypot hauptsächlich Wörterbuchangriffe[32] (*Dictionary Attack*) eingesetzt wurden. Die gelisteten Strings entsprechen einfachen, sehr unsicheren Passwörtern, die Teil einer jeden typischen Passwortliste sind. Interessanterweise ist das „korrekte“ Passwort 123456 zugleich das mit den meisten Vorkommen (121 mal) im Testzeitraum, obwohl es nur 20 erfolgreiche Logins gab. Eine genauere Analyse der Protokolle ergab, dass das Passwort `123456` neben `root` auch mit unzähligen anderen typischen Benutzernamen wie `temp`, `felix` oder `gast` ausprobiert wurde. Das Passwort `changeme` zielt auf Systeme ab, deren Aufforderung nach einer Neuinstallation beim erstmaligen Login zur Änderung der Logindaten nicht nachgekommen wurde.

[32]Methode, ein unbekanntes Passwort mit Hilfe einer Passwörterliste zu entschlüsseln (Quelle: Wikipedia)

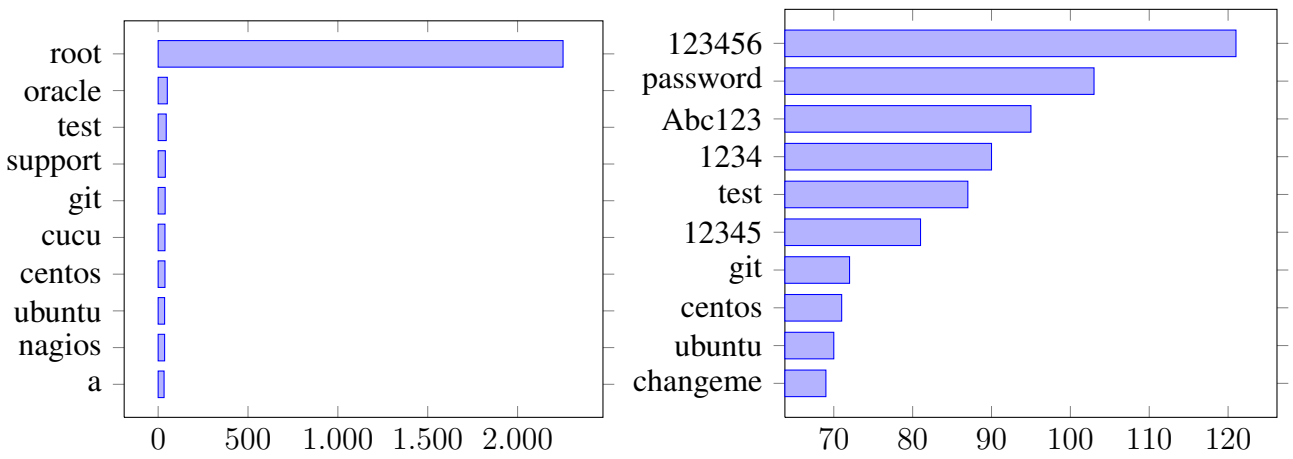

Abbildung 1: Häufigste Benutzernamen und Passwörter

Abschließend soll noch ein Blick auf die häufigsten Kombinationen aus Benutzernamen und Passwörtern geworfen werden, wie in Abbildung 2 zu sehen.

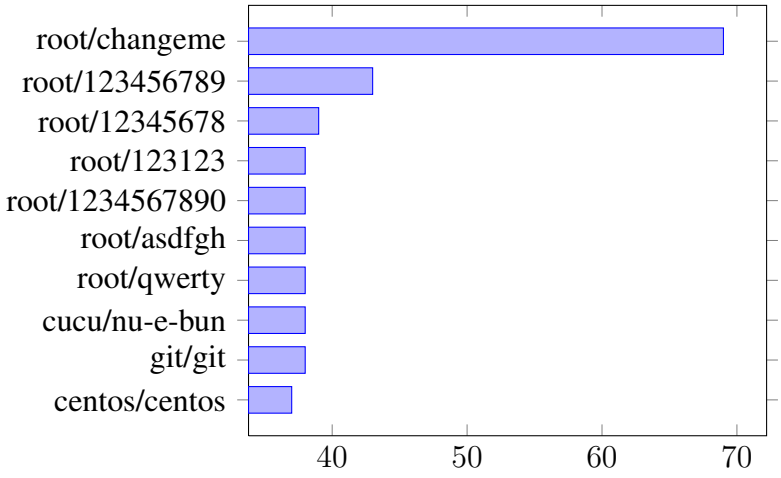

Abbildung 2: Häufigste Benutzername/Passwort-Kombinationen

Das zuvor bereits beschriebene Passwort `changeme` wurde ausschließlich zusammen mit dem Benutzernamen `root` verwendet. Diese Kombination wird als Standardzugang für den SSH-basierten *Integrated Lights Out Manager* auf Sun- und Oracle-Servern verwendet [15]. Die restlichen Kombination beinhalten hingegen wenige Überraschungen und entsprechen typischen Wörterbuchangriffen: Der meistverwendete Benutzername `root` in Kombination mit einfachen, schwachen Passwörtern oder, wie im Falle von `git/git`, identische Benutzernamen und Passwörter. Auffällig ist die Kombination `cucu/nu-e-bun`, die von mehrere chinesischen Hosts verwendet wurde, teilweise auch mehrmals in Folge. Eine Recherche zum Ursprung dieser Zugangsdaten führte jedoch zu keinem Ergebnis.

Zusammenfassend lässt sich festhalten, dass der SSH-Honeypot `kippo` innerhalb seiner spezifischen Rolle zum Sammeln von Informationen über Angriffe auf den SSH-Dienst gut geeignet ist. Die Indi-

vidualisierungsmöglickeiten erlauben das Anpassen der Software in einem solchen Umfang, dass die Täuschung für Angreifer nicht sofort zu erkennen ist. Laut [7] ist es theoretisch möglich, auch ohne erfolgreiches Login den Honeypot als solchen auf Netzwerkebene zu identifizieren, dies ist jedoch mit erheblichem Aufwand verbunden und wird zumindest von einfachen Bots nicht geprüft. Der experimentelle Betrieb von `kippo` war von keinen softwareseitigen Problemen begleitet und führte zu dem Schluss, dass die Abschaltung des `root`-Logins und die Verwendung von sicheren Passwörtern Schutz vor den unvermeidlichen Bot-Wörterbuchangriffen bieten.

3.1.2 dionaea

Eine der größten Gefahren im mit den Internet verbundenen Netzwerken ist sogenannte *Malware*[33]. Diese lässt sich in verschiedene Kategorien einteilen, die wiederum spezifische Gegenmaßnahmen erfordern. So versucht beispielsweise ein (menschlicher) Angreifer nach einem erfolgreichen Login auf einem System, seine Rechte auszuweiten und anschließend ein sog. *Rootkit* zu installieren, das elementare Systembestandteile modifiziert und es ihm ermöglicht, zu einem späteren Zeitpunkt unbemerkt auf das System zurückzukehren. Viel häufiger werden Systeme jedoch von *Viren*, *Trojanern* oder *Würmern* infiziert, die von einem neuen befallenen Host automatisch weitere Angriffsversuche starten.

`Dionaea` ist ein Honeypot mit dem Ziel der Analyse der Wurmaktivität und dem Sammeln von Malware, alles möglichst automatisiert. Die Software ist ein direkter Nachfolger des Honeypots `Nepenthes`, der wiederum auf dem Daemon `mwcollectd` basiert. Es handelt sich um Open-Source-Software, die unter der GPLv2 veröffentlicht wird. Hinter dem Projekt steht eine aktive Entwicklergemeinschaft, die Webseite befindet sich unter `http://dionaea.carnivore.it`. Der Kern von `Dionaea` ist in der Programmiersprache C geschrieben, alle Module hingegen in Python. Die Software wird als serverseitiger Low-Interaction-Honeypot klassifiziert, genau wie das zuvor beschriebe kippo. Im Gegensatz zu diesem ist `dionaea` allerdings ein Multi-Purpose-Honeypot, da eine ganze Reihe verschiedener Protokolle implementiert sind.

Das Hauptaugenmerk des Honeypots liegt auf der Emulation des **SMB**-Protokolls (**S**erver **M**essage **B**lock). Dieses wurde von Microsoft entwickelt und wird unter Windows-Betriebssystemen (in neueren Versionen unter dem Akronym **CIFS**) für Drucker- und Dateifreigaben verwendet. In der Vergangenheit gab es immer wieder Exploits auf Basis dieses Protokolls, weshalb es bis zum heutigen Tag zu einem der beliebtesten Angriffsziele von sich selbstständig weiterverbreitender Malware zählt. Die ständige Weiterentwicklung des Protokolls im Zuge neuer Windows-Versionen und damit immer wieder neu auftretende Sicherheitslücken tragen ebenfalls dazu bei. Zusätzlich zum SMB-Protokoll emuliert `dionaea` allerdings auch noch eine Reihe weiterer Protokolle in unterschiedlich ausgereifter Form, wie `HTTP`, `FTP`, `TFTP`, `MSSQL`, `MySQL` oder `SIP`.

Architektur `Dionaea` ist vollständig modular aufgebaut und orientert sich dabei am Ablaut eines typischen Angriffs auf das SMB-Protokoll. Das `core`-Modul ist für die Orchestrierung aller anderen Module zuständig, öffnet und verwaltet die benötigten Ports. Ein Angriff läuft dann in einer Reihe fixer Stufen ab:

[33] Computerprogramme, die entwickelt wurden, um vom Benutzer unterwünschte und gebenenfalls schädliche Funktionen auszuführen. (Quelle: Wikipedia)

- Sogenannte **Vulnerability Modules** emulieren jeweils ein Protokoll und eine Reihe von bekannten Sicherheitslücken. Ziel ist es, Angreifer glauben zu lassen, dass sich hinter der IP ein echter Host verbirgt, der den entsprechenden Dienst anbietet. Hierfür genügt es bereits, ein Protokoll nur auf rudimentäre Weise zu simulieren. Ein Angriff läuft dann üblicherweise so ab, dass eine der bekannten Sicherheitslücken durch einen gezielten Exploit ausgenutzt wird. Dieser erlaubt es dem Angreifer, beispielsweise in Folge eines Bufferüberlaufs eine sogenannte *Payload* auszuführen. Dabei handelt es sich um ausführbare Daten, die vom Eindringling als Teil des Exploits mit zum Server gesendet werden.
- Payloads können in verschiedenen Formaten vorliegen und unterschiedliche Ziele verfolgen. Sie enthalten *Shellcode*[34] und sind laut [33] meistens mit einem *XOR encoder* verschlüsselt , um einer Entdeckung durch signaturbasierte *NIDS*-Systeme zu entgehen. Da Angreifer verschiedene Epxloits mit verschiedensten Payloads kombinieren können, sind ihre Funktionsweisen sehr verschieden: So kann eine Payload einen Port abhören und dem Eindringling damit eine interaktive Shell anbieten, auf die er sich von außen verbinden kann. Eine andere, häufig im Kontext von sich selbstständig weiterverbreitender Malware anzutreffende Variante ist, dass die Payload weiteren Schadcode von einem entfernten Server herunterläd und ausführt. Die Aufgabe von **Shellcode Parsing Modules** ist das Entschlüsseln und Analysieren von Payloads, entweder mit einer in `dionaea` eingebauten Analyse-Engine, die versucht mit statistischen Techniken Informationen über die Payload zu sammeln oder einer auf `libemu` basierenden Emulations-Engine, in der Shellcode ausgeführt und die Ergebnisse untersucht werden können. Ziel dieser Module ist das Gewinnen der URL, über die der Angreifer versucht weitere Schadsoftware nachzuladen und auszuführen.
- Die Aufgabe der **Fetch Modules** ist das Herunterladen der Malware, die der Angreifer versucht in Folge seines Exploits auszuführen. `Dionaea` ist an dieser Stelle auf das Einsammeln von Malware spezialisiert: Wenn ein Shellcode allerdings nicht versucht, Malware herunterzuladen, sondern stattdessen beispielsweise eine Shell nach außen öffnen will, kann auch diese rudimentär simuliert werden. Zum Beziehen der Schadsoftware werden verschiedene Protokolle unterstützt, darunter `HTTP`, `FTP` und `TFTP`, die typischerweise zum Einsatz kommen.
- Nachdem ein Exemplar der vom Angreifer verwendeten Malware heruntergeladen wurde, wird es an ein oder mehrere **Submission Modules** übergeben. Diese kümmern sich um die weitere Verarbeitung: Im einfachsten Fall das Abspeichern der Malware auf der lokalen Festplatte, das Senden an einen MySQL-Server oder aber auch das Inanspruchnehmen von externen Malware-Analyse-Diensten wie beispielsweise *virustotal*[35]. Diese überprüfen die übermittelte Malware und senden den Analysebericht an eine in der `dionaea`-Konfiguration angegebene E-Mail-Adresse.
- Module zum **Logging** erlauben, analog zu `kippo`, das Speichern von Protokolldaten in lokalen Dateien, auf *MySQL*- oder *PostgreSQL*-Datenkbank-Servern oder auch das Senden an einen *Surf-CERT IDS*-Server.

Zusätzlich zu den zuvor genannten liefert `dionaea` noch einige Module mit, die bei Bedarf zusätzliche, aber nicht nowendige Funktionalität bieten: Mit dem `pcap`-Modul können jegliche Anfragen, auch welche, die an geschlossene Ports gerichtet sind, aufgezeichnet werden. Darauf aufbauend erlaubt das `nfq`-Modul sogar, für jeden angefragten, eigentlich ungenutzten Port dynamisch Dienste zu simulieren. In der derzeitigen Implementierung leitet das Modul jede Anfrage wieder zum Angreifer zurück, so dass dieser im Idealfall einen Angriff gegen sein eigenes System startet.

[34] Eine Serie von Maschinenbefehlen

[35] `https://virustotal.com/`

Auf der Emulation des SMB-Protokolls liegt der Fokus des Projekts, allerdings werden auch andere Dienste emuliert. Die Implementierung der entsprechenden Protokolle ist jedoch zum Zeitpunkt dieser Untersuchung von schwankender Qualität, die meisten erlauben das Verbinden und geringfügige Kommunikation, simulieren jedoch (noch) keine Sicherheitslücken und können demzufolge nicht zum Sammeln von Malware verwendet werden. So beantwortet zum Beispiel das *MySQL Vulnerability Module* einfache SQL-Fragen mit Hilfe einer internen SQLite-Datenbank, kann jedoch diese Interaktion lediglich protokollieren.

Installation und Konfiguration Die Installation von `dionaea` ist nicht trivial. Da die Software im Moment nicht in den Software-Repositories der großen Linux-Distributionen vorhanden ist, muss sie zusammen mit einer Reihe von Abhängigkeiten manuell installiert werden. Bei auf Debian basierenden Distributionen ist dieser Prozess etwas einfacher, da viele der Abhängigkeiten bereits vorhanden sind. Bibliotheken wie `liblcfg` für das Format der Konfigurationsdatei und `libemu` für die Shellcode-Emulation müssen jedoch trotzdem von Hand nachinstalliert werden. Wenn alle Abhängigkeiten installiert sind, ist das Kompilieren und Installieren von `dionaea` selbst unkompliziert. Die vollständige Prozedur und alle Abhängigkeiten sind auf der Website des Projekts detailliert beschrieben.

Die Konfiguration der Software erfolgt in einer einzigen Konfiguratonsdatei, `dionaea.conf`, und ist ebenfalls gut auf der Website dokumentiert. Da `dionaea` vollständig modular aufgebaut ist, besteht die Konfiguration aus einer Aneinanderreihung von kurzen Konfigurationsabschnitten für jedes einzelne Modul. Es kann ausgewählt werden, welche der unterstützen Serverdienste aktiviert werden sollen, die verwendeten Ports sind jedoch hardcoded. `Dionaea` kann beliebig viele Netzwerkinterfaces und IP-Adressen gleichzeitig abhören.

Protokollierung und Auswertung Wie schon bei `kippo` richtet sich die Qualität der Protokolle nach den verwendeten Modulen. Jedes Vulnerability-Modul bietet eine unterschiedlich akkurate Simulation von Protokollen und Sicherheitslücken. Da jedes dieser Module über die `dionaea`-API Daten aufzeichnet, hängt von diesen die Brauchbarkeit der gewonnenen Informationen ab. Der Honeypot zeichnet in der Standardkonfiguration sehr viele Informationen auf, was über die Konfigurationsdatei allerdings auf das Wesentliche reduziert werden kann. `Dionaea` schreibt bei Verwendung des `file`−Log-Moduls alle anfallenden Daten in eine einzelne Logdatei und bietet keinen eingebauten Algorithmus zum Rotieren der Logdateien an, weshalb bei längerem Betrieb des Honeypots der von unixoiden Systemen bekannte systemweite `logrotate`-Daemon verwendet werden sollte.

Da die reguläre Logdatei nur schwer zu parsen ist und sehr viele für die Auswertung der Daten unnötige Informationen erhält, speichert `dionaea` die relevanten Informationen über Angriffe (bzw. Angriffsversuche) standardmäßig gesondert in einer *SQLite*-Datenbank ab. Dies erleichtert die Auswertung der Daten enorm, da so via SQL interaktive Abfragen generiert werden können. Alternativ, bzw. zusätzlich dazu ist auch die Verwendung von *MySQL*- oder *PostgreSQL*-Datenbanken möglich. Für jede diese Optionen existiert ein gesondertes Logging-Modul. Auch für hybride Systeme wie das *SurfCERT IDS* ist ein eigenes Logging-Modul bereits enthalten. Dieses wurde im Rahmen der in Kapitel 4 durchgeführten großflächigen Honeypot-Installation verwendet, erwies sich jedoch als inkompatibel mit einer aktuellen Version des SurfCERT IDS. Es waren jedoch nur geringfügige Modifikationen nötig, um die Funktionalität wiederherzustellen. Dies wird in Kapitel 4 noch im Detail erläutert.

Erweiterbarkeit Der Honeypot `dionaea` wurde mit Rücksicht auf umfassende Erweiterungsmöglichkeiten entwickelt. Das komplette System ist modular und es können für jede der im vorherigen Abschnitt erwähnten Stufen eigene Module geschrieben werden. Am interessantesten für den Benutzer ist letztendlich das Hinzufügen eigener oder die Ergänzung existierender „Vulnerability Modules“, um die Funktionalität des Honeypots zu erweitern. Alle Module sind in Python geschrieben, der Code ist laut der ENISA-Studie [7] zumindest teilweise dokumentiert. Es konnte jedoch keine ausführliche API-Dokumentation gefunden werden, was die Entwicklung neuer Module erschweren dürfte. Die bereits mitgelieferten Module liefern jedoch genügend Beispiele, um eigene Funktionalität nach einer gewissen Einarbeitungszeit umsetzen zu können.

Experimenteller Einsatz `Dionaea` wurde im Rahmen dieser Arbeit zuerst für kurze Zeit an einem Breitband-Privatanschluss getestet und stellte sich als sehr effizient heraus. Der Ressourcenverbrauch war auch auf einem älteren Rechner (auf Pentium III-Basis) im normalen Betrieb nicht spürbar und es gelang schon binnen kurzer Zeit, an zahllose Malware-Exemplare zu gelangen. Aus diesem Grund wurde `dionaea` als ein elementarer Bestandteil der großflächigen Honeypot-Installation im nächsten Kapitel gewählt. Eine Auswertung der gesammelten Daten erfolgt dann in Kapitel 5.

3.1.3 amun

Bei `amun` handelt es sich um einen weiteren Low-Interaction-Honeypot, dessen Ziel das Sammeln von Malware-Exemplaren ist. Die Software ist vollständig in Python geschrieben und steht unter der GPL. Der entscheidende Unterschied zum zuvor behandelten Honeypot `dionaea` ist, dass `amun` nicht nach emulierten Diensten oder Protokollen kategorisiert, sondern nach emulierten Sicherheitslücken. Hauptaugenmerk liegt somit darauf, Verwundbarkeiten möglichst akkurat abzubilden. Das bedeutet in der Praxis, dass eine „normale“ Kommunikation mit den vom Honeypot angebotenen Diensten nicht möglich ist. Im Kontrast dazu erlaubt `dionaea` es beispielsweise, sich mit einer *MySQL*-Clientanwendung mit dem emulierten *MySQL*-Server zu verbinden und SQL-Abfragen in begrenztem Rahmen auszuführen. `Amun` bietet diese Freiheiten nicht - eine „erfolgreicher“ Dialog kommt nur zustande, wenn ein Angriff stattfindet, der auf eine der emulierten Sicherheitslücken abzielt. Jeder Dienst wird nur so weit simuliert, bis die zum erfolgreichen Ausführen eines Exploits notwendigen Rahmenbedingungen gegeben sind. Da ausschließlich spezifische Sicherheitslücken simuliert werden, ist ein Erkennen von Zero-Day-Angriffen nicht möglich.

Architektur `Amun` ähnelt in seiner Architektur stark dem zuvor beschriebenen Honeypot `dionaea`. Die Software ist ebenfalls vollständig modular aufgebaut und kategorisiert Module folgendermaßen [6]:

- Die vom Honeypot unterstützten Verwundbarkeiten werden in Form von **Vulnerability Modules** vorgehalten. Es handelt sich dabei um endliche Zustandsaustomaten, die immer für genau eine Sicherheitslücke im betreffenden Service verantwortlich sind. Diese Module besitzen folglich eine Reihe von *Stages*, die durchlaufen werden müssen, bis der Exploit erfolgreich war und die Payload vom Angreifer empfangen und ausgewertet werden kann. Jede Antwort des Clients wird überprüft und nur falls sie individuelle Kriterien (wie z.B. die Länge der Antwort in Bytes) erfüllt, wird zur nächsten Stage übergegangen.

- Ein **Request Handler** wird für jede neue eingehende Verbindung separat erstellt und verwaltet die für den jeweiligen Port zuständigen *Vulnerability Modules*. Vom Client eingehende Nachrichten werden an diese Module verteilt. Die Module, welche *keine* gültige Antwort zurückliefern, werden aus der Liste der noch verfügbaren Module ausgeschlossen. Im Idealfall bleibt zuletzt nur noch ein Modul übrig, das die eigentliche Payload des Exploits erhält und diese an den *Shellcode Analyzer* weitergibt. Falls in irgendeiner Phase der Interaktion kein Modul mehr eine Antwort liefert, wird die Verbindung geschlossen, da in diesem Fall kein von `amun` unterstützter Exploit vorlag.
- Der **Shellcode Analyzer** entschlüsselt und analysiert die übertragene Payload. Im Vergleich zu `dionaea` unterstützt `amun` keine Shellcode-Emulation via `libemu`, sondern nur eine statische Shellcode-Analyse mit regulären Ausdrücken. Falls es der Software nicht gelingt, automatisch eine URL mit der herunterzuladenden Malware zu finden, wird die Payload im Binärformat zur späteren manuellen Analyse lokal gespeichert. Neue gefundene reguläre Ausdrücke können nach erfolgreicher Analyse hinzugefügt werden, um die Arbeitsweise des Honeypots in Zukunft zu verbessern.
- Aufgabe der **Download Modules** ist es wiederum, die Malware von einer eventuell gefundenen URL zu beziehen. Dies geschieht wiederum mit separaten Modulen für in diesem Fall häufig verwendete Protokolle wie `FTP`, `HTTP` oder `TFTP`.
- **Logging Modules** übernehmen die Protokollierung der gewonnenen Daten. Wie schon bei `dionaea` existieren Module, um Daten lokal in Dateien zu speichern, zu einem MySQL-Server oder via E-Mail zu versenden. Auch das Hybridsystem *SurfCERT IDS* wird unterstützt, wovon im praktischen Teil dieser Arbeit gebraucht gemacht wurde.
- **Submission Modules** dienen der Weiterverarbeitung der im Erfolgsfall gewonnenen Malware. Im Standardfall wird das Modul `submit-md5` genutzt, das Malware lokal mit der MD5-Summe als Dateinamen abspeichert. Andere Module ermöglichen es, die Malware-Samples an Auswertungsdienste wie *CWSandbox* oder eine *MySQL*-Datenbank zu senden.

Installation und Konfiguration Da die Software vollständig in Python geschrieben ist, genügt es nach Installation eines Python-Interpreters, `amun` herunterzuladen und die Datei `amun_server.py` auszuführen.

Konfigurieren lässt sich der Honeypot über eine zentrale Konfigurationsdatei, in der sich neben dem zu verwendenden Netzwerkinterface und den IP-Adressen oder Netzwerken, für die `amun` zuständig sein soll, auch die zu verwendenden Module spezifizieren lassen. Die größte Rolle spielt an dieser Stelle die Konfiguration der aktiven *Vulnerability Modules*, von denen mehrere für einen einzelnen Dienst aktiviert werden können. Der zuständig *Request Handler* kümmert sich dann wie oben beschrieben um die Verwaltung aller für einen Port zuständigen Module.

Erweiterbarkeit Die Software ist modular aufgebaut und erlaubt somit das Hinzufügen von in Python geschriebenen Modulen. Dass `amun` ausschließlich in Python geschrieben wurde erleichtert zudem die Einarbeitung. Im Normalfall ist das Schreiben zusätzlicher *Vulnerability Modules* die beste Möglichkeit, die Effizienz des Honeypots zu verbessern. Um diese Aufgabe zu vereinfachen, hat der Autor ein Tool mitgeliefert, das die Konvertierung von in XML beschriebenen Verwundbarkeiten zu Python-Modulen ermöglicht. Somit ist es auch für Nicht-Programmierer möglich, aus bekannten zu einem Exploit gehörigen Interaktionsmustern neue Module zu generieren. Eine kurze Anleitung auf der `amun`-Website[36]

[36] `http://amunhoney.sourceforge.net/write_simple_vulns.php`

beschreibt den Prozess genauer. Selbstverständlich können Module auch von Hand in Python verfasst werden.

Experimenteller Einsatz Leider scheint das `amun`-Projekt inaktiv zu sein, die letzte freigegebene Version der Software stammt aus dem Jahr 2010. Zudem wurde ein Großteil der Dokumentation von der Website entfernt. Trotzdem wurde die Software im Rahmen dieser Arbeit untersucht und auch im praktischen Teil eingesetzt, da sie das ebenfalls verwendete `dionaea` sehr gut um zusätzliche emulierte Sicherheitslücken ergänzt und somit einem Angreifer mehr Angriffsvektoren bietet, was wiederum die Menge und Qualität der sammelbaren Daten erhöht. `Amun` wurde im Rahmen der in Kapitel 4 beschriebenen praktischen Honeypot-Installation eingesetzt.

3.1.4 SCADA-Honeypots

Computersysteme kommen in der industriellen Fertigung und bei unzähligen anderen Industrieprozessen, in Kraftwerken oder Pumpstationen zum Einsatz und sind auch dort miteinander vernetzt. Sie werden in diesem Zusammenhang als **ICS** (Industrial Control System) bezeichnet und dienen zum Überwachen und Steuern von einzelnen Geräten oder auch komplexeren Anlagen. **SCADA**-Systeme (Supervisory Control and Data Acquisition) sind ICS-Systeme, die das großflächige Vernetzen von an verschiedenen physischen Orten ablaufenden Industrieprozessen über große Entfernungen ermöglichen. Da diese Netzwerke das Internet als Transportmedium nutzen, sind SCADA-Systeme wie alle anderen TCP/IP-Netzwerkanwendungen auch durch externe Angriffe gefährdet. Dass Schwachstellen auch aktiv ausgenutzt werden, hat unter anderem der Computerwurm **Stuxnet** gezeigt, der iranische Nuklearanlagen befiel [5]. Wie eine Untersuchung ergeben hat, sind allein in Deutschland mehrere hundert Industrieanlagen teilweise ungeschützt direkt über das Internet erreichbar [3]. Grund ist hierfür insbesondere, dass Anlagenbetreiber die Webschnittstelle dieser Systeme ohne zusätzliche Absicherung frei zugänglich machen. Diese Steuerungs- und Überwachungswebsites werden anschließend von den Suchmaschinen indiziert und können über diese mit sogenannten „Dork"-Abfragen gefunden werden. Es ist daher sinnvoll, Honeypots zu entwickeln, die SCADA-Systeme simulieren, um mehr über die Angriffsmuster auf derartige Netzwerke zu erfahren. Da die zuvor genannten großflächigen Angriffe erst in jüngster Vergangenheit bekannt geworden sind, ist das Themenfeld der SCADA-Honeypots allerdings noch vergleichsweise neu.

Die Firma *Trend Micro* untersuchte im März 2013 die Angriffe auf Industriesteuerungsanlagen in einem Praxistest [45]. Sie entwarf hierzu drei verschiedene Honeypot-Systeme, die mit jeweils einer IP-Adresse im Internet platziert wurden:

- Ein auf einer Amazon-EC2-Instanz laufendes Ubuntu-System mit einem `apache2`-Webserver und „selbst entwickelten Webseiten", die der Steuerung eines PLC[37]-Systems ähneln. Die Überwachung erfolgte u.a. mit dem NIDS *snort* und `tcpdump`. Zusätzlich wurden die Protokolle `FTP`, `HTTP` und Modbus emuliert, die in SCADA-Systemen häufig zum Einsatz kommen.
- Ein High-Interaction-Honeypot, bestehend aus einem Server mit Software zur PLC-Steuerung und einem angeschlossenen PLC-Gerät. Modifizierungen auf dem Server konnten den Zustand des PLC-Systems physisch ändern, dies spiegelt das Verhalten eines realen Systems wider.

[37] Programmable Logic Controller, ein Gerät, das zur Steuerung oder Regelung einer Maschine oder Anlage eingesetzt und digital programmiert wird (Quelle: Wikipedia)

- Ein Low-Interaction-Honeypot mit emulierten Services eines Produktionssystems. Es ist lediglich bekannt, dass der Honeypot `dionaea` eingesetzt wurde, über andere verwendete Software machten die Autoren keine Angaben.

Mit Start des Experiments wurde die Sichtbarkeit der Honeypots verbessert, indem die Webinterfaces bei der Suchmaschine *Google* angemeldet und somit indiziert wurden. Erste Angriffsversuche, das heißt unautorisierte Zugriffe auf sichere Seitenbereiche, Änderungen an PLC-Controllern oder gezielte Angriffe auf die angebotenen Protokolle wurden erstmals nach 18 Stunden verzeichnet. Binnen des Testzeitraums von 28 Tagen wurden letztendlich 38 Angriffe aus 14 verschiedenen Ländern gezählt, 13 davon wurden von den gleichen Angreifern mehrmals wiederholt und deshalb von den Sicherheitsforschern als „gezielt“ eingestuft. Die Mehrzahl der Angriffe erreichte die in den USA platzierten Honeypots zudem aus China (35%) und den USA (19%). Insbesondere die wiederholt angreifenden Eindringlinge bereiteten den Autoren sorgen, da sich deren Angriffsmuster im Laufe der Zeit änderten, wenn die vorangegangenen Versuche fehlgeschlagen waren. Es kann sich also nicht nur um automatisierte Anriffe gehandelt haben. Die Studie zeigt, dass Angriffe auf Industriesteuerungen über das Internet sowohl automatisiert als auch gezielt durchgeführt werden. Leider wurde die verwendete Softwarekonfiguration nicht genauer spezifiziert und auch nicht veröffentlicht, weshalb Netzwerkbetreiber zum Aufsetzen von SCADA-Honeypots zu Alternativen greifen müssen.

Das **SCADA HoneyNet Project**[38] bietet eine Reihe von Python-Skripten unter einer freien Lizenz an, die als Service-Plugins für den serverseitigen Low-Interaction-Honeypot *honeyd* gedacht sind. Jedes Skript emuliert einen in SCADA-Netzwerken häufig anzutreffenden Dienst: `Telnet`, `FTP`, `HTTP` und das Modbus-Protokoll. Aus diesen Grundbausteinen kann mit Hilfe von *honeyd* ein komplexeres virtuelles Netzwerk errichtet werden. Leider wurde das Projekt seit 2005 nicht mehr weiterentwickelt und ist laut einer Studie [7] auch nur von geringem Nutzen. Bemängelt werden die mangelnde Qualität der Protokolldaten und die nur sehr geringe emulierte Systemtiefe. Vorteilhaft ist jedoch, dass sich diese Skripte leicht installieren lassen, nur ein geringes Sicherheitsrisiko für den Honeypotbetreiber darstellen, gut erweiterbar sind und zudem von der guten Skalierbarkeit des *honeyd*-Honeypots profitieren. Die zuvor genannte Studie rät vom Einsatz dieses Honeypots mit der Begründung ab, dass nur innerhalb eines echten SCADA-Netzwerkes verlässliche Daten gesammelt werden könnten. Das zuvor beschriebene Experiment von *Trend Micro* zeigt jedoch, dass Industriesteueranlagen auch direkt über das Internet angesprochen werden können, entsprechend gefährdet sind und solche außerhalb der SCADA-Netzwerke platzierten Honeypots auch aktiv angegriffen werden.

Die Firma *Digital Bond* veröffentlichte im Jahr 2004 erstmals die Software **SCADA Honeynet**. Die letzte aktualisierte Version erschien 2007 und umfasst zwei Festplattenabbilder, die in virtuellen Maschinen wie VirtualBox oder QEMU verwendet werden können. Auf der ersten virtuellen Maschine ist die *Honeywall*-Distribution, die bereits in Kapitel 2.3.1 ausführlicher beschrieben wurde, installiert. Sie wurde um zusätzliche Regeln für das *snort* IDS ergänzt, um auf Angriffe auf in Industrieanlagen oft genutzte Protokollen wie `DNP3` oder `ICCP` reagieren zu können. Als Honeypot „hinter“ der Honeywall kann entweder ein echtes SCADA-System als High-Interaction-Honeypot oder alternativ eine oder mehrere Instanzen der zweiten virtuellen Maschine installiert werden, die einen Low-Interaction-Honeypot darstellt [23]. Auf diesem System laufen verschiedene Serverdienste, darunter der Honeypot *honeyd*, der FTP-Server *iFTPd* und der Webserver *FizmezWebServer*. Diese Dienste sollen im Zusammenspiel ein System in einem SCADA-Netzwerk möglichst detailgetreu simulieren. Laut der ENISA-Studie [7] ist dieses Ziel jedoch nicht erreicht worden, die Emulation der Systeme sei stark limitiert und erlaube zudem anhand

[38] `http://scadahoneynet.sourceforge.net/`

der immer gleichen präsentierten Banner eine einfache Identifikation des Honeypots. Die komplizierte Architektur des Systems sei zudem ein großes Hindernis: Die verwendeten Programme erlauben es nicht, mit einer Honeyppot-Installation mehr als eine IP-Adresse abzudecken. Zudem biete die Software keine Konsistenz bei der Protokollierung, die gesammelten Daten seien teilweise über das Webinterface der Honeywall, teilweise aber auch nur über Logdateien mit inkonsistenten Formaten in der virtuellen Honeypot-Maschine vorzufinden.

Während bisher keine anderweitig praktikablen Alternativen für die zuvor genannten Projekte auffindbar waren, wurde kurz vor Fertigstellung dieser Belegarbeit der Honeypot **conpot**[39] in einer frühen Version veröffentlicht [4]. Die Software basiert auf dem serverseitigen Low-Interaction-Honeypot **glastopf**[40], der verwundbare Webanwendungen simuliert. Conpot steht unter der GPLv2, ist in Python geschrieben und wird aktiv weiterentwickelt. Eine externe Evaluation der Software konnte kurz nach ihrem Erscheinen nicht gefunden werden, ein ausführlicher Test war zudem im Rahmen dieser Arbeit nicht mehr möglich.

3.1.5 Mobile Honeypots

Die starke Verbreitung von mobilen Endgeräten in Form von Smartphones und Tablets ist ein noch relativ junges Phänomen. Aus Sicht der Angreifer sind diese Geräte jedoch lohnenswerte Ziele: Sie sind häufig oder sogar ständig mit dem Internet verbunden und aufgrund ihrer geringen Ressourcen ein verhältnismäßig leichtes Ziel für DoS- oder DDoS-Angriffe. Zahlreiche Probleme insbesondere im Zusammenhang mit Malware, die sich als normale Anwendung („App“) tarnt, sind aus der jüngsten Vergangenheit bekannt.

Da die Mehrheit dieser Mobilgeräte heutzutage auf dem mobilen Betriebssystem Android basiert, dem ein modifizierter Linux-Kernel zugrunde liegt, macht es Sinn, die aus der Linux-Welt bekannten Honeypot-Konzepte zu nutzen, um Angriffe auf Smartphones identifizieren zu können und mehr über die Methoden der Angreifer in diesem Bereich zu erfahren. Auf diesem noch sehr jungen Gebiet gibt es zwar bereits einige wissenschaftliche Arbeiten, die hier vorgestellt werden sollen, allerdings noch keine frei verfügbare Software, um selbst mobile Honeypots aufzusetzen. Die nachfolgend vorgestellten Verfahren basieren jedoch teilweise auf bereits existierenden, modifizierten Honeypot-Lösungen und profitieren somit von der Linux-Basis der Androidgeräte und der bereits auf normalen PCs etablierten Honeypot-„Szene“.

Ein einfaches Beispiel für die Anpassung existierender Honeypots für den mobilen Bereich stellen die Smartphone-Honeypots im Mobilfunknetzwerk der Deutschen Telekom da, über die im Herbst 2012 berichtet wurde [10]. Darüber ist bekannt, dass der zuvor vorgestellte SSH-Honeypot `kippo` zur Simulation von Android- und iOS-basierten Mobilgeräten modifiziert und eingesetzt wurde. Details über den Umfang der Modifikationen sind jedoch nicht bekannt. In einer knappen Auswertung der gesammelten Daten fand die Telekom jedoch heraus, dass in ihrem mobilen Netz zwar Angriffe auf den simulierten SSH-Dienst durchgeführt wurden, die wenigen erfolgreichen Angreifer allerdings kaum gezielt vorgingen: Wie in einer Aufzeichnung der Aktivitäten einiger Eindringlinge auf der simulierten Shell zu sehen war, setzten diese keine gezielten, auf mobile Endgeräte abgestimmte Exploits oder Toolkits ein und schienen sich teilweise auch gar nicht bewusst zu sein, auf was für Hardware sie Zugriff hatten.

Das Experiment der Telekom simulierte jedoch nur einen sehr kleinen Aspekt von Mobilgeräten: SSH-

[39] `https://github.com/glastopf/conpot`
[40] `http://glastopf.org/`

Server sind standardmäßig nicht darauf installiert oder aktiv. Matthias Wählisch, Sebastian Trapp et al. haben deshalb auf ähnliche Art und Weise versucht, Daten über die Anzahl und Art der Angriffsversuche in Mobilfunknetzen zu gewinnen [42]. Sie haben hierfür auf vier Linux-Systemen die Low-Interaction-Honeypots `dionaea` und `Honeytrap` installiert. Jeder dieser Hosts wurde in einem anderen Netzwerk platziert: einer in einem UMTS-Mobilfunknetzwerk, ein anderer im Netzwerk einer Universität, ein weiterer an einem privaten DSL-Anschluss und ein letzter in einem Darknet[41]. Details über die genaue Konfiguration oder die aktivierten Dienste der Honeypots sind leider nicht bekannt, jedoch enthält der Bericht Daten über die TCP-Serverports 22, 80 und 222, was zumindest auf die Simulation von SSH- und HTTP-Servern hinweist. Es hat sich bei dieser Untersuchung herausgestellt, dass die Intensität der Angriffe große Ähnlichkeit zur Anzahl der Angriffe am DSL-Anschluss hat, während der Host im Darknet deutlich weniger und der im Universätsnetzwerk am Häufigsten angegriffen wurde. Während sich also das sog. „Hintergrundrauschen“ im mobilen IP-Adressbereich mit dem eines üblichen Breitbandanschlusses vergleichen lässt, wurde im Rahmen der Studie jedoch festgestellt, dass die Angriffe auf die mobilen Endgeräte von insgesamt weniger voneinander verschiedenen Angreifern stammten, dafür aber mit einer stärkeren Intensität durchgeführt wurden. Matthias Wählisch et al. schließen daraus, dass dies ein Anzeichen dafür sein könnte, dass einige Angreifer ihre Bemühungen verstärkt gegen mobile Endgeräte zu richten beginnen. Eine Langzeituntersuchung dieses Trends ist bisher nicht bekannt.

Die beiden bisher beschriebenen Ansätze bestanden ausschließlich aus serverseitigen Low-Interaction-Honeypots, die mit den für diese Kategorie typischen Nachteilen zu kämpfen haben: Es ist nicht möglich, Zero-Day-Angriffe zu erkennen und die Täuschung wird je nach Implementierung von Angreifern an irgendeinem Punkt durchschaut werden, da diese simulierten Systeme bei der Interaktion enge Grenzen setzen. Zudem bieten mobile Endgeräte typischerweise keine Serverdienste an, sondern agieren primär als Clients. Einen für dieses Szenario geeigneteren Ansatz haben Collin Mulliner et al. bei der Entwicklung des Smartphone-Honeypots **HoneyDroid** verfolgt [13]. Sie versuchen nicht, Mobilgeräte mit herkömmlichen PCs zu emulieren, sondern haben Software spezifisch für den Einsatz auf Smartphones entwickelt, die als High-Interaction-Honeypot klassifiziert werden kann. Die Architektur fußt auf einem *Fiasco.OC*-Microkernel, der direkt auf der Smartphone-Hardware ausgeführt werden kann. Auf diesem läuft dann *L4Android*[42], ein modifizierter Linux-Kernel, auf dem wiederum ein Android-Userland nativ ausgeführt wird. Monitoring, Protokollierung und Kapselung des Systems (Containment) können von direkt auf dem Microkernel laufenden Prozessen übernommen werden und sind innerhalb des virtualisierten Android-Systems nicht sichtbar. Da das auf L4Android laufende Android nur virtuelle Geräte zur Verfügung hat und keinen direkten Hardwarezugriff, können zudem alle Systemaufrufe und Events vom darunterliegenden System aufgezeichnet werden.

Leider ist HoneyDroid nicht zum Testen frei verfügbar, das Paper von Collin Mullinger et a. enthält zudem lediglich eine Beschreibung der Architektur, aber keine weiteren Informationen über die Implementierung oder mit diesem System gesammelte Daten. Es ist somit unklar, ob das beschriebene Verfahren bereits praxistauglich ist oder überhaupt noch weiterentwickelt wird.

[41] Ein P2P-Netzwerk, dessen Teilnehmer ihre Verbindungen untereinander manuell herstellen (Quelle: Wikipedia)
[42] `http://l4android.org`

3.2 Client-Side-Honeypots

Da Angreifer es verstärkt auf die Clientseite abgesehen haben und Nutzer beispielsweise mit *Social Engineering* dazu zu bewegen versuchen, gefährliche URLs zu besuchen, auf denen speziell für Clientsysteme zugeschnittene Exploits warten, hat sich auch der Fokus in der Weiterentwicklung der Honeypots verschoben. Die hier vorgestellten Konzepte simulieren Benutzer, die mit Clientanwendungen auf Ressourcen im Netzwerk zugreifen und sich damit Risiken aussetzen. Es werden zunächst die beiden modernen Client-Side-Honeypots `shelia` und `argos` vorgestellt, die verschiedenen Ansätzen zum Erkennen von Angriffen nachgehen. Anschließend wird der `Ghost` USB Honeypot beschrieben, der einen zu allen bisher präsentierten Verfahren konträren Ansatz bei der Realisierung eines Honeypot-Systems verfolgt.

3.2.1 shelia

Bei `shelia` handelt es sich um einen in C und Python geschriebenen clientseitigen High-Interaction-Honeypot, dessen Ziel das Finden von Exploits ist, die in überwachten Clients bei der Kommunikation mit korrumpierten Servern auftreten. Die Software ist ausschließlich für Windows verfügbar und lässt sich aufgrund der Abhängigkeiten von Windows-Systemaufrufen auch nicht ohne Weiteres auf andere Betriebssysteme portieren. Das Projekt wird nicht aktiv weiterentwickelt (die letzte veröffentliche Version stammt aus dem Jahr 2009), ist jedoch aufgrund seiner Architektur trotzdem weiterhin einsetzbar. Im Gegensatz zu Low-Interaction-Honeypots werden keine spezifischen Sicherheitslücken simuliert, sondern eine Engine eingesetzt, die mögliche Angriffe mit analytischen Mitteln zu erkennen versucht. Typischerweise werden Client-Honeypots verwendet, um „attackierende“ Websites zu finden, die Sicherheitslücken in Web-Browsern oder deren Plugins auszunutzen versuchen. Shelia erlaubt allerdings aufgrund seines im Abschnitt „Funktionsweise“ beschriebenen Angriffs-Erkennungsverfahrens das Ausführen und Überwachen nahezu beliebiger Client-Anwendungen.

`Shelia` setzt eine vollständige Windows-Installation voraus, wahlweise in einer virtuellen Maschine oder auch auf einem physischen System. Der Honeypot kann über zwei verschiedene Quellen URLs oder Dateien eingespeist bekommen, die er anschließend mit einem zuvor auswählbaren, passenden Client-Programm öffnet. So könnte beispielsweise eine URL mit dem *Internet Explorer* und ein PDF-Dokument mit dem *Adobe Reader* geöffnet werden. Anschließend wird eine vorher definierte Zeit abgewartet, während alle vom System in dieser Zeit durchgeführten Aktionen wie Systemaufrufe oder Dateizugriffe aufgezeichnet und überwacht werden. Falls der Verdacht auf einen Exploit besteht, können je nach Konfiguration Containment-Maßnahmen ergriffen werden, um zu verhindern, dass Malware Schaden anrichten kann. Nach Ablauf einer fixen Wartezeit wird der Prozess des gestarteten Clients beendet, die generierten Protokolle via FTP extern gespeichert und die nächste URL oder Datei entgegengenommen. Eine Reihe von mitgelieferten Skripten ermöglichen den vollautomatischen Betrieb des Honeypots.

Funktionsweise Grundsätzlich besteht der Honeypot aus einer Warteschlange, über die neue zu evaluierende Ressourcen (URLs oder Dateien) bereitgestellt werden, einem „Client“, der die betreffende Client-Anwendung startet und einer Analyseeinheit, die die Ausführung überwacht, protokolliert und auf potentielle Exploits reagiert. Die aus diesen Überlegungen resultierende Architektur, die von den Entwicklern schließlich implementiert wurde, besteht aus folgenden Einheiten [36]:

- Ursprünglich, in einer auf der Website des Projekts[43] noch immer verfügbaren älteren Version, wurde die Warteschlange als **E-Mail Processor** implementiert. Dabei handelte es sich um eine Anwendung, die einen konfigurierten *Outlook Express*-Mailclient voraussetzte, den es startete und alle E-Mails aus dem Posteingang verarbeitete. Dabei wurden alle gefundenen URLs und E-Mail-Anhänge der Warteschlange hinzugefügt. Diese Abhängigkeiten wurden in der aktuellen Version entfernt und durch ein flexibles Client/Server-System ersetzt, das die Automatisierung von `shelia` stark vereinfacht. In der aktuellen Implementierung wird im Windows-Honeypot ein Python-Skript gestartet, das sich mit einem auf der Kommandozeile spezifizierbaren Endpunkt zu verbinden versucht. Für diesen existiert ein weiteres Python-Skript, das aus einer *MySQL*-Datenbank neue zu besuchende URLs und Dateien bezieht und bei einer erfolgreichen Verbindung stückweise an den Client weitergibt. Der Client startet für jede vom Server empfangene und zu überprüfende Ressource den Client `shelia.exe`, dessen Arbeitsweise im Anschluss beschrieben wird. Nachdem die Analyse einer Ressource abgeschlossen ist, sendet das Client-Skript die gewonnenen Daten an einen zuvor konfigurierten FTP-Server, auf den nur Daten geschrieben, aber nicht gelöscht oder überschreiben werden können. Somit wird verhindert, dass der möglicherweise von Malware befallene Host die Logs modifiziert. Nachdem die Daten gesichert sind, wird vom Server die nächste Ressource entgegengenommen. Dieser Ansatz ist schwieriger zu konfigurieren als der ältere *E-Mail Processor*, erlaubt jedoch eine deutlich größere Flexibilität: Wenn der Windows-Client in einer VM betrieben wird, kann das zuvor erwähnte Server-Skript auf dem virtualisierenden Host ausgeführt werden und zugleich die VM steuern. Auf diese Art ist ein Unbeaufsichtigtes Arbeiten des Honeypots möglich.
- Die Anwendung `shelia.exe` übernimmt die Hauptaufgabe des Honeypots: Die Ressource mit einer passenden Clientanwendung öffnen und jegliche von der Anwendung durchgeführten Aktivitäten überwachen. Der **Client Emulator** übernimmt erstere Aufgabe, während die **Process Monitor Engine** für die Überwachung und Protokollierung zuständig ist. Hierfür wird einerseits die Technik der **DLL Injection** genutzt [36], um die mitgelieferte Bibliothek `SpyDll.dll` im Kontext der gestarteten Clientanwendung nachzuladen und anschließend via **API Hooking** eine Reihe von Systemaufrufen mit shelia-eigenen Implementierungen ersetzt. Somit ist es möglich, alle Zugriffe auf Dateien und Verzeichnisse und auf die Registry zu überwachen und auf Auffälligkeiten zu überprüfen. Zusätzlich überwacht `shelia` auch noch eine Reihe von Systemaufrufen, die an *Winsock*, den Windows-TCP/IP-Stack, gerichtet sind. Die Idee dahinter ist, Malware-Downloadversuche nach einem erfolgreichen Exploit abzufangen.
- Aufgabe der **Attack Detection Engine** ist es, durch Analyse der von der *Process Monitor Engine* gelieferten Daten herauszufinden, ob ein Exploit stattfindet. Während andere Honeypots wie beispielsweise **Capture-HPC NG** die Klassifikation von Systemaufrufen anhand von White- und Blacklists vornehmen oder nur Änderungen am Dateisystem nach vollendeter Ausführung feststellen können, haben die `shelia`-Entwickler ein Verfahren entwickelt, mit dem eine automatische Klassifikation möglich ist, die angeblich weniger anfällig für false-positives und false-negatives ist. Hierfür analysiert die *Attack Detection Engine* die Stelle im Speicher, von der aus ein Systemaufruf durchgeführt wurde. Wenn es sich dabei um einen Speicherbereich handelt, der nicht als ausführbar markiert wurde und folglich keinen Code enthalten sollte, wird eine Warnung generiert. Falls die Containment-Maßnahmen beim Aufruf von `shelia` aktiviert wurden, treten diese in Kraft und stellen sicher, dass eine möglicherweise heruntergeladene Malware zwar abgespeichert, aber nicht ausgeführt wird. Die Engine analysiert dabei ausschließlich Systemaufrufe und kann folglich nur darauf basierende Payloads erkennen.

[43] `http://www.cs.vu.nl/~herbertb/misc/shelia/`

- Die **Attack Log Engine** übernimmt schließlich das Aufzeichnen aller gesammelten Daten und das Abspeichern im Dateisystem. Hierzu zählen primär Statusmeldungen der Anwendung selbst, als auch die Systemaufrufe, die zum Exploit geführt haben und die gesendete Payload. Falls zur Steuerung des Honeypots der zuvor beschriebene Python Client/Server-Mechanismus genutzt wird, werden diese Daten auf einem externen FTP-Server abgelegt.

Installation und Konfiguration Der Honeypot besteht in der zu diesem Zeitpunkt aktuellen Version von 2009 aus der Anwendung `shelia.exe` selbst, der `SpyDll.dll`, die in gestartete Prozesse „injiziert“ wird und dem zugehörigen Quellcode. Zusätzlich wird noch ein Paket namens `shelia_scripts` zum Download angeboten, das u.a. die zuvor erwähnten Client/Server-Hilfsprogramme enthält. Da die Hauptquelle für diese Skripte ein *MySQL*-Server ist, werden auch noch einige Tools mitgeliefert, die das Durchsuchen von E-Mail-Postfächern via `IMAP` unterstützen und jegliche darin gefundenen Anhänge und URLs in die Datenbank schreiben (`imapcli.py`),
URLs aus Dateien extrahieren (`addurlsfrmtxt.py`) oder einzelne Dateien zur Datenbank hinzufügen (`addattachment.py`). Eine Installation der Software unter Windows ist nicht nötig, `shelia.exe` kann direkt ausgeführt werden.

Es empfiehlt sich, das Windows-System zu virtualisieren und auf die Skripte `shelia_srv.py` und `shelia_cli.py` zurückzugreifen. Eine `shelia`-Konfiguration mit diesen Komponenten läuft folgendermaßen ab:

1. Auswahl einer Virtualisierungsplattform und anschließend Anlegen einer virtuellen Maschine, sowie Installation eines Windows-Systems darin. Die VM muss mit dem Hostsystem kommunizieren können, hierfür bietet sich das Einrichten einer Netzwerkbrücke mit einem physisch vorhandenen und einem virtuellen Interface an. Für die Virtualisierung empfiehlt sich unter unixoiden Systemen *QEMU/KVM*, da `shelia_srv.py` für diesen Fall bereits vorbereitet ist und die entsprechenden Befehle zum Starten und Beenden der VM enthält. Eine Anpassung an andere Software ist allerdings trivial, wenn diese Haltepunkte unterstützt und ein Interface für die Kommandozeile besitzt. Nach Installation des Betriebssystem sollte die Datenausführungsverhinderung (DEP), sowie die Firewall unter Windows deaktiviert werden, eine genaue Anleitung für diese Schritte wird mit `shelia` mitgeliefert. Zuletzt ist es noch wichtig, das System auf den Patch-Stand zu bringen, der getestet werden soll. Es kann wünschenswert sein, nicht alle verfügbaren Sicherheitsupdates zu installieren, um die Anfälligkeit älterer, möglicherweise im Netzwerk vorhandener Computer zu überprüfen und deshalb deren Konfiguration zu spiegeln. Die zusätzliche Installation von zusätzlicher zu testender Software oder Plugins erhöht die Chance, Exploits zu finden. Auf einem Web-Client-Honeypot empfiehlt sich aus diesem Grund die Installation von *Adobe Flash*, dem *Adobe Reader* und *Oracles* Java-Plugin.
2. Installation einer Python-Distribution[44], Download von `shelia` und Start der `shelia_cli.py` mit der IP-Adresse des Hostsystems als Argument im Windows-Client. Auf dem Host wird später das Serverskript laufen. Wenn das Skript gestartet ist, wird ein Schnappschuss der virtuellen Maschine angelegt und abgespeichert. QEMU/KVM unterstützt *copy-on-write*, womit sichergestellt ist, dass Veränderungen nach dem Wiederherstellen des Schnappschusses in einem separaten Image gespeichert werden. Somit kann zu jedem Zeitpunkt zu einem sauberen, nicht infizierten Ausgangssystem zurückgewechselt werden.
3. Auf dem Hostsystem wird gemäß der mitgelieferten Dokumentation ein FTP-Server installiert, der

[44] `http://python.org/`

von der VM heraus erreichbar ist und ausschließlich das Schreiben neuer, nicht aber das Verändern oder Löschen bestehender Daten erlaubt. Für diesen Zweck empfehlen die Autoren den Einsatz von `vsftpd`[45].

4. Die zu besuchenden URLs und zu öffnenden Dateien werden aus einer *MySQL*-Datenbank gelesen. Hierfür ist die Einrichtung eines *MySQL*-Servers auf dem Hostsystem, das Anlegen einer neuen Datenbank und eines neuen Benutzers, sowie das Ausführen des mitgelieferten Skripts `shelia_createdb.sql` nötig, das die Datenstrukturen enthält. Anschließend werden die Zugangsdaten zum Server im Skript `shelia_srv.py` ergänzt. Eine detaillierte Anleitung, die zudem noch geringfügige Anpassungen der Datenbankserverkonfiguration umfasst, ist im `shelia`-Paket enthalten.
5. Reparatur des `shelia_srv.py`-Skripts. Ein Bug in der zuletzt veröffentlichten Version in der Funktion `server_loop` verhindert das erfolgreiche Aufbauen einer Verbindung vom `shelia`-Client aus derm VM heraus zum Server.
6. Befüllen der Datenbank mit URLs, die vom Honeypot besucht werden sollen (Tabelle `url_list`) und zu überprüfenden Dateien (Tabelle `atchm_list`). Abschließend setzen der QEMU-Starparameter in der Datei `shelia.conf`, bei Verwendung einer anderen Virtualisierungslösung muss das Skript `shelia_srv.py` direkt angepasst werden.

Wenn alles korrekt eingerichtet ist, genügt es, `shelia_srv.py` auszuführen. Das Skript startet die virtuelle Instanz mit dem gespeicherten Schnappschuss, wartet darauf, dass der im virtuellen Windows laufende `shelia`-Client eine Verbindung aufbaut und sendet diesem anschließend der Reihe nach die zu verarbeitenden Daten. Nach jeder überprüften Ressource werden die Daten, die in das FTP-Verzeichnis des Hosts geschrieben wurden, in das Protokollverzeichnis verschoben und anschließend die nächste Ressource verarbeitet. Die virtuelle Maschine wird zudem periodisch nach einer frei konfigurierbaren Anzahl von behandelten Ressourcen auf den Schnappschuss-Zustand zurückgesetzt, um zu vermeiden, dass ausgeführte Malware die Funktion beeinträchtigt und wieder eine saubere Ausgangskonfiguration herrscht.

Protokollierung und Auswertung Der Honeypot speichert die vom FTP-Server gewonnenen Protokolldaten für jede einzelne Ressource in einem separaten Verzeichnis ab. Dazu zählt neben der gewonnen Malware, zumeist eine `.exe`- oder `.com`-Datei auch die Datei `log.txt`, in der sämtliche `shelia`-Konsolenausgaben gespeichert werden. Sie enthält neben dem Datum, dem aktiven Anwendungsnamen (z.B. `IEXPLORE.EXE`) und dem Namen der geöffneten Ressource im Wesentlichen eine Liste mit abgefangenen Systemaufrufen, die von der **Attack Detection Engine** als gefährlich eingestuft wurden. Hieraus lässt sich ableiten, was genau der Shellcode, der mit dem Exploit nachgeladen wurde, unternimmt. Falls `shelia` mit aktivierter Containment-Funktion gestartet wurde, werden nach einem erkannten Angriff die meisten nachfolgenden Systemaufrufe unterbunden. Der Honeypot versucht in diesem Fall lediglich, ein Exemplar der Malware zu erhalten, falls der Exploit versucht eine solche herunterzuladen - das Ausführen derselben soll allerdings verhindert werden. Ohne Containment ist der Honeypot dem Angreifer schutzlos ausgeliefert und das System wird möglicherweise infiziert, jedoch mit dem Vorteil, dass mehr Daten über den Angriff gewonnen werden können.

[45] `http:/vsftpd.beasts.org/`

Erweiterbarkeit Die Support-Skripte zur Automatisierung sind in Python geschrieben und lassen sich somit mit wenig Aufwand modifizieren, der eigentliche Honeypot `shelia.exe` bietet jedoch keine API an, mit der neue Funktionalität hinzugefügt werden könnte. Wenn also beispielsweise die Attack Detection Engine geändert werden soll, erfordert dies eine Modifikation der Originalquellen und ein Neukompilieren der `shelia.exe`. Die Anwendung ist also nicht modular aufgebaut, was die Erweiterbarkeit deutlich erschwert.

Experimenteller Einsatz Um zu überprüfen, ob `shelia` Exploits gegen das System erfolgreich erkennt, wurde das Penetration-Testing-Framework **Metasploit**[46] eingesetzt. Eine ausführliche Beschreibung dieser Software würde an dieser Stelle den Rahmen sprengen, deshalb sei nur gesagt, dass das Framework eine große Anzahl von client- und serverseitigen Exploits und eine Reihe von Tools bereitstellt, die Angriff auf andere Systeme eingesetzt werden können. Ziel des Framework ist es, Computersysteme auf mögliche Schwachstellen hin zu überprüfen.

Für das Testsetup wurde in einer auf *KVM*[47] basierenden virtuellen Maschine ein Windows XP aufgesetzt, das bis auf ein installiertes Service Pack 2 mit keinen zusätzlichen Sicherheitsupdates ausgestattet wurde. Außerdem wurden automatische Updates und die Windows-Firewall deaktiviert. Das Metasploit-Framework wurde auf dem Hostsystem eingerichtet und anschließend über die Metasploit-Konsole `msfconsole` Exploits herausgesucht, gegen die sich das System anfällig zeigte. Dieser Prozess gestaltet sich dank Metasploit sehr komfortabel:

1. Finden eines möglicherweise passenden Exploits in der Metasploit-Datenbank[48], starten der Konsole `msfconsole`.
2. Laden des gefundenen Exploits, hier am Beispiel einer „Java Applet Remote Code Execution“: `use exploit/multi/browser/java_jre17_exec`
3. Konfiguration des Exploits, das heißt Auswahl der mitzuliefernden Payload und bei Bedarf Einstellen payloadspezifischer Optionen: `set PAYLOAD java/meterpreter/reverse_tcp`. Diese spezielle Payload öffnet eine Reverse-TCP-Shell, d.h. wenn der Exploit auf dem Zielsystem erfolgreich war, versucht dieses eine Verbindung zum Rechner des Angreifers aufzubauen (`set LHOST <Ziel-IP>`). Auf diesem wartet ein von Metasploit gestarteter Dienst, der dem Angreifer schließlich eine Command-Shell auf dem angegriffenen System zur Verfügung stellt. Diese Payload wird in der Praxis verwendet, wenn das Zielsystem durch eine Firewall geschützt ist und dem Angreifer das Aufbauen neuer Verbindungen zum Ziel nicht möglich ist. Weitere Optionen erlauben u.a. das Festlegen der URL, auf der der Schadcode des Exploits einem Client zugesendet wird oder der lokale Port, auf dem der Exploit-Server starten soll.
4. Der Befehl `exploit` startet schließlich den Exploit: Wenn es sich, wie hier der Fall, um einen serverseitigen, webbasierten Exploit handelt, wird ein minimaler Webserver unter Berücksichtigung der vorher festgelegten Optionen ausgeführt.

Getestet wird der Exploit anschließend durch das Besuchen der eben konfigurierten URL mit dem Internet Explorer im Windows-Zielsystem. Da ein Fehler in Java ausgenutzt wird, ist die Installation eines entsprechenden Java-Browserplugins in einer anfälligen Version Voraussetzung für das Gelingen des Angriffs. Falls sich das System als anfällig herausstellt und der Exploit erfolgreich ist, wird die Payload

[46] `http://metasploit.com/`
[47] Kernel-based Virtual Machine, eine linuxspezifische Virtualisierungslösung
[48] `http://metasploit.com/modules/`

ausgeführt und es ist möglich, aus der Metasploit-Konsole heraus mit Hilfe des `sessions`-Kommandos auf die zuvor eingerichtete Remote-Shell zuzugreifen.

Für das Experiment wurden im virtuellen Windows veraltete Versionen von *Adobe Flash* und *Oracles Java* samt der zugehörigen Browser-Plugins und ActiveX-Steuerelemente installiert, um zusätzliche Angriffsvektoren auf das System anzubieten und `shelia` ausgiebiger testen zu können. Mit Hilfe des Metasploit-Frameworks und der zuvor beschriebenen Prozedur wurden anschließend drei funktionierende Exploits herausgesucht:

- Java 7 Applet Remote Code Execution (`java_jre17_exec`), CVE-2012-4681
- Adobe Flash Player 11.3 Kern Table Parsing Integer Overflow (`adobe_flash_otf_font`), CVE-2012-1535
- Microsoft XML Core Services MSXML Initialized Memory Corruption, (`msxml_get_definition_code_exec`), CVE-2012-1889

Die bisherigen Metasploit-Tests wurden ohne `shelia` durchgeführt, um funktionierende Exploits zu finden. Im Anschluss wurden die Exploits erneut der Reihe nach eingerichtet und diesmal mit Hilfe des `shelia`-Honeypots (unter Nutzung des Internet Explorer 6) besucht. Die Containment-Funktionen wurden nicht aktiviert, da die Metasploit-Payload lediglich aus einer Remote-Shell bestand und das System nicht mit Malware infizierte. Anhand der Konsolenausgaben und Logfiles ergab sich schließlich folgendes Ergebnis: Der Honeypot erkannte erfolgreich den Angriff auf die „XML Core Services“ und Adobe Flash, nicht jedoch den Java Exploit. Da der Flash-Exploits erst nach dem Release des Honeypots bekannt war, ist somit bewiesen, dass `shelia` grundsätzlich in der Lage ist, Zero-Day-Exploits zu erkennen. Dass der Algorithmus der *Attack Detection Engine* jedoch nicht perfekt ist, zeigt sich daran, dass der Java-Exploit unbemerkt geblieben ist. Die Ursache ist möglicherweise in der Java-Plugin-Architektur zu finden: Das Applet wird in einer separat gestarteten Java-VM ausgeführt. Da die `SpyDll.dll` aber nur in den Browser-Prozess injiziert wurde, können in anderen Prozessen ausgelöste Exploits nicht gefunden werden. Eine mögliche Lösung wäre an dieser Stelle, in die *Java Virtual Machine* (und andere nach dem Internet Explorer gestartete Prozesse) die `SpyDll.dll` zusätzlich zu injizieren, dies wird jedoch von `shelia` im Moment nicht unterstützt.

3.2.2 argos

Bei `argos` handelt es sich um einen weiteren High-Interaction-Honeypot, der als Honeyclient entwickelt wurde, aber auch als Server-Side-Honeypot eingesetzt werden kann. Die Software basiert auf der Virtualisierungslösung *QEMU*, ist ebenso in C geschrieben und steht teilweise unter der *GPLv2*- und der *BSD 3-Clause*-Lizenz. Die zum Zeitpunkt dieser Arbeit aktuelle Version erschien Ende 2012, mit der Support für auf Windows 7 basierende Honeypots hinzugefügt wurde. `Argos` ist komplizierter Einzurichten und ressourcenlastiger als `shelia` und wird an dieser Stelle erwähnt, da es einen völlig anderen Ansatz zum Erkennen von Angriffen verfolgt. Aufgrund der *QEMU*-Basis kann dieser Honeypot nicht auf einem physischen System betrieben werden, die darin installierten Betriebssysteme laufen immer in einer virtuellen Maschine.

Funktionsweise Während `shelia` eine unter Windows ausführbare Anwendung ist, versuchen die `argos`-Entwickler das Konzept des High-Interaction-Honeypots weiter zu generalisieren und streben

Plattformunabhängigkeit an. Bei diesem Honeypot handelt es sich um eine modifizierte Variante des PC-Emulators *QEMU*, die automatisch Angriffe unter Zuhilfename des **Dynamic Taint Analysis**-Verfahrens zu erkennen versucht [31]. Die Idee dahinter ist, eingehenden Netzwerkverkehr als *tainted* („schmutzig“) zu kennzeichnen und zusätzlich aufzuzeichnen. Die Verwendung der „verschmutzten“ Daten, die so empfangen werden, wird während der Ausführung des virtuellen Betriebssytems genauestens nachvollzogen: Wenn die Daten in (virtuelle) CPU-Register oder in den (virtuellen) Hauptspeicher der VM kopiert werden, wird dieser neue Speicherort ebenfalls als *tainted* markiert. Anschließend versucht `argos`, die verdächtige Verwendung dieser Daten, beispielsweise als Sprungziel (`JMP`), zu identifizieren. Falls eine derartige Zugriffsverletzung stattfindet, werden alle als *tainted* markierten Blöcke samt Inhalt in einer Protokolldatei gespeichert, zusammen mit zusätzlichen Informationen über den Systemzustand (z.B. Registerinhalte). Zuletzt erlaubt der Honeypot noch das automatische Generieren von Angriffssignaturen aus den gewonnenen Daten, die wiederum zusammen mit einem Network Intrusion Detection System wie *snort* verwendet werden können. Der ganze Prozess ist betriebssystemunabhängig, da er transparent für dieses in der virtuellen Maschine stattfindet.

Zusätzlich besitzt `argos` noch eine betriebssystemspezifische Komponente: Da im Falle eines erkannten Angriffes nur „low-level“-Informationen über Register, Speicherinhalte und CPU-Instruktionen aufgezeichnet werden können, gestaltet sich deren Auswertung als kompliziert. Um also mehr Informationen über den Prozess zu erhalten, von dem beispielsweise eine `JMP`-Anweisung zu „tainted data“ ausgeht, injiziert der Honeypot eigenen, OS-spezifischen Shellcode, der zusätzliche forensische Analysen eines Angriffs erlaubt. Die Autoren des Honeypots beschreiben diesen Vorgang folgendermaßen [31]: „In other words, we ’exploit’ the code under attack with our own shellcode.“ Mit dieser Technik ist es möglich, zusätzlich zu den Speicher- und Registerinformationen u.a. die ID und den Namen des Prozesses, der den Angriff ausgeübt hat, sowie eine Liste von geöffneten Dateien zu erhalten.

Die aus allen gesammelten Daten erstellte Signatur des Angriffs kann zusammen mit einem hybriden Honeypot-System namens *SweetBait* eingesetzt werden [30]. Es verbessert die Signatur automatisch, indem beispielsweise netzwerkspezifische, variable Daten (wie IP-Adressen) aus ihr entfernt werden, damit sie im Kontext einer anderen Netzwerkumgebung Gültigkeit behält. *SweetBait* enthält zudem sowohl ein *Intrusion Detection System*, als auch ein *Intrusion Prevention System*, um aktiv auf Angriffe reagieren zu können. Die neu gewonnenen Signaturen werden automatisch in dieses System eingespeist. Leider ist dieses Projekt bis auf `argos` selbst, das nur einen kleinen Teil der Architektur darstellt, nicht veröffentlicht wurden.

Installation und Konfiguration `Argos` basiert auf *QEMU* und ist nur unter einem Linux-Kernel lauffähig. Es hat nur wenige Abhängigkeiten von externen Bibliotheken, darunter SDL[49] für die Grafikausgabe. Die Installation selbst ist ausführlich in der Dokumentation des Projekts[50] beschrieben und soll deshalb hier nicht näher erläutert werden. Nach erfolgreichem Kompilieren kann die ausführbare Datei `argos` direkt als Ersatz für *QEMU* verwendet werden, die Konfiguration erfolgt ausschließlich über Kommandozeilenparameter, die bereits von *QEMU* bekannt sind. Es empfiehlt sich, das virtuelle Honeypotsystem zuvor via KVM/QEMU zu installieren und wie gewünscht einzurichten, da der von `argos` verursachte Overhead der *Dynamic Taint Analysis* die Installation sonst unnötig ausbremsen würde. Für das Festplattenabbild der virtuellen Maschine bietet sich zudem das Format `qcow2` an, das copy-on-write unterstützt und es erlaubt, eine neue Instanz der VM auf Basis eines schreibgeschützten Images

[49] `http://libsdl.org/`
[50] `http://www.few.vu.nl/argos/docs/uguide.html`

zu starten. Änderungen im Laufe der Session werden dann in ein separates Image geschrieben. Diese Funktionalität erlaubt es, zu jedem Zeitpunkt wieder zum Ursprungszustand des Systems zurückkehren zu können. Die Automatisierung des Client-Honeypots wird dadurch stark vereinfacht. Entsprechende Skripte hierfür werden zusätzlich unter dem Namen `argos-utils` zum Download angeboten und erlauben ein ähnliches Automatisieren wie schon zuvor beim `shelia`-Honeypot.

Experimenteller Einsatz Interessant war der Vergleich mit dem Honeypot `shelia`, der beim Testen nicht alle drei Exploits erkannt hatte (siehe Kapitel 3.2.1). Aus diesem Grund wurde dasselbe Windows-Image erneut verwendet, da zuvor bereits *KVM* als Virtualisierungslösung zum Einsatz gekommen war, das ebenfalls QEMU als Basis nutzt. Leider ist es nicht möglich, `argos` zusammen mit *KVM* zu nutzen, wodurch die durch aktuelle Prozessoren zur Verfügung gestellten Virtualisierungserweiterungen *Intel-VT* und *AMD-V* nicht angesprochen werden können. Die Folge ist, dass die `argos`-VM spürbar langsamer läuft als eine vergleichbare *KVM*-VM.

Für das Experiment wurde `argos` mit dem Festplattenabbild gestartet, das zuvor bereits für das `shelia`-Experiment genutzt wurde. Der Schnappschuss-Modus wurde aktiviert, damit das Abbild nur zum Lesen geöffnet wird und damit unverändert bleibt. Erneut wurde das *Metasploit*-Framework mit den drei zuvor beschriebenen Exploits konfiguriert. Das Experiment verlief schließlich erfolgreich, alle drei Exploits wurden von `argos` als solche erkannt und Protokolle mit den Speicher- und Registerinhalten, die als „tainted“ markiert waren, angelegt. Es muss allerdings berücksichtigt werden, dass diese einfachen Tests nur drei stichprobenartig herausgesuchte Exploits umfassten, um die prinzipielle Funktionsweise der Honeypot-Lösungen zu überprüfen. Um objektive Aussagen über die Qualität treffen zu können, wären ausführlichere Testreihen nötig, die im Rahmen dieser Arbeit nicht durchführbar gewesen wären. Es soll jedoch an dieser Stelle darauf hingewiesen werden, dass sich `argos` im direkten Vergleich besser geschlagen und dank der Betriebssystem- und Anwendungsunabhängigkeit somit als die qualitativ bessere Lösung hinsichtlich der Erkennung von Angriffen herausgestellt hat.

3.2.3 Ghost USB Honeypot

Alle zuvor beschriebenen Honeypots sind im Kern Netzwerkanwendungen. Ihre einzige Schnittstelle mit der „Außenwelt“ ist das Netzwerk, mit dem sie verbunden sind. Einen völlig anderen Ansatz verfolgen Sebastian Poeplau und Jan Gassen mit ihrem **Ghost** getauften *USB-Honeypot* [29]. Wie auch bei `dionaea` oder `amun` ist es das Ziel, Gefahrenquellen zu erkennen und Malware-Exemplare zu sammeln. Sich selbstständig ausbreitende Würmer nutzen allerdings nicht immer nur mit einem Host verbundene Netzwerke zur Infektion anderer Systeme, sondern greifen häufig auch auf die Infektion von entfernbaren Speichermedien wie USB-Sticks oder Speicherkarten zurück. `Ghost` legt den Fokus auf genau diesen Aspekt: Der Honeypot simuliert auf einem Windows-System einen angesteckten USB-Stick. Dass dieser virtueller Natur ist, soll nur schwer feststellbar sein, da die Emulation tief im Windows-Kernel verankert ist. Diese Technik weist allerdings im Vergleich zu den bisher beschriebenen Honeypots einige Unterschiede auf: Wie in Kapitel 2 beschrieben, ist es normalerweise das Ziel eines Honeypots, Angreifer mittelt Täuschung von den Produktivsystemen abzulenken. Im Falle des Ghost Honeypots muss jedoch das Hostsystem bereits infiziert sein, da nur im System schon aktive Malware sich weiterverbreiten kann. Die Autoren der Software empfehlen deshalb den Einsatz derselben nur als letztes Mittel, wenn andere Sicherheitsmaßnahmen wie Antivirensoftware oder IDS bereits versagt haben. Eine weitere Besonderheit

wurde bereits in 2.3.2, Integration ins Netzwerk, näher beleuchtet: Während die zuvor genannten Honeypots auf separaten, speziell zu diesem Zweck installierten Systemen zum Einsatz kommen, muss ein USB Honeypot auf einem Produktivsystem, wie beispielsweise einem Bürocomputer, installiert werden, um wirkungsvoll arbeiten zu können. Die Grenzen der Klassifikation sind an dieser Stelle fließend und insofern könnte der Ghost USB Honeypot auch als *Host Intrusion Detection System* eingeordnet werden.

Funktionsweise Laut Poeplau und Gassen [29] waren die größten Herausforderungen bei der Umsetzung des Konzeptes das Simulieren eines Speichergerätes und das Kennzeichnen desselben als „austauschbar“ (*Removable Storage Device*). Malware, die sich über Wechselmedien verbreitet, hat unter Windows zwei verschiedene Möglichkeiten, sich über neu hinzugefügte Speichermedien zu informieren: Regelmäßiges Abfragen aller vorhandenen Laufwerke (**Polling**) oder die Nutzung von **Events**, unter Windows hieße das konkret, auf die Nachricht `WM_DEVICECHANGE` zu reagieren. Der Ghost USB Honeypot unterstützt beide Möglichkeiten, da das virtuelle Device mit Hilfe von zwei eigenen Gerätetreibern implementiert wird und für andere Software kaum von einem nativen Gerät unterscheidbar ist. Der Treiber `ghostbus.sys` simuliert einen USB-Bus, an den der mit Hilfe von `ghostdrive.sys` simulierte USB-Stick angehangen wird. Da letztgenanntes Treibermodul sich dem System als Plug&Play-fähig präsentiert, wird es wie gewünscht als *Removable Storage Device* angelegt.

Die Software ist nur für Windows-Betriebssysteme verfügbar und wird zusammen mit einem komfortablen Installer geliefert. Sie unterstützt manuellen und automatischen Betrieb, sowohl mit Hilfe einer GUI als auch einem CLI[51]. Es ist außerdem möglich, die Funktionen des Honeypots von externer Software aus zu nutzen. Hierzu liefert der Honeypot eine eigene API in Form der Bibliothek `ghostlib.lib` mit[52]. Im manuellen Betrieb kann der virtuelle USB-Stick von Hand ein- und ausgehangen werden, im Automatikmodus geschieht dies automatisch in festen Zeitabständen. Der Intervall kann vom Nutzer festgelegt werden, ebenso wie die Dauer, für die das virtuelle Gerät eingehangen wird. In einer Untersuchung der Autoren des Honeypots haben sich 30 Sekunden als empfehlenswert herausgestellt, wobei die Infektionen meistens schon nach nur wenigen Sekunden erfolgt waren, in Abhängigkeit von der verwendeten Malware [29].

Erweiterbarkeit Der Ghost USB Honeypot ist in C, sowie C# geschrieben und steht unter der *GPLv3*-Lizenz. Die Software ist nicht modular aufgebaut, ein Erweitern der Funktionalität setzt also Verständnis des bestehenden Quellcodes voraus. Hierfür steht eine rudimentäre Entwicklerdokumentation zur Verfügung, die beim Einstieg helfen soll.

Protokollierung und Auswertung Es werden grundlegende Informationen über auf den USB-Stick zugreifende Prozesse aufgezeichnet, der Hauptnutzen des Honeypots liegt jedoch in der gesammelten Malware. Das Dateisystem des virtuellen USB-Sticks wird in Form eines Images in einer einzelnen Datei gespeichert. Nach dem Ein- und Aushängen des virtuellen Geräts kann überprüft werden, ob sich das Image geändert hat und welche Dateien darauf abgelegt wurden. Dies muss jedoch von Hand erfolgen, die Entwickler liefern hierfür keine gesonderten Tools mit.

In einem von den Autoren durchgeführten Experiment erwies sich der Ghost USB Honeypot als überaus

[51] Command Line Interface

[52] `https://code.google.com/p/ghost-usb-honeypot/wiki/GhostLib`

effektiv. Alle getesteten Malware-Samples, die sich via USB-Wechselmedien weiterzuverbreiten versuchten, unterlagen der Täuschung und schrieben ihren Schadcode auf das virtuelle Gerät. Eine anschließende Analyse der gesammelten Malware könnte mit Antivirenprogrammen oder im Falle von unbekannter Malware unter Zuhilfename einer *Sandbox* erfolgen.

3.3 Zusammenfassung

In diesem Kapitel wurden sowohl aktuelle client- als auch serverseitige Honeypots aus Forschung und Entwicklung vorgestellt. Es hat sich herausgestellt, dass trotz der Reife einiger dieser Lösungen und der Tatsache, dass Honeypot-Technologien nun schon seit mehr als 20 Jahren zur Verfügung stehen, noch immer keine etablierten „Standardlösungen“ existieren und immer wieder neue Projekte, die häufig an Universitäten ihren Ursprung haben, ihre Vorgänger abzulösen versuchen. Die vorgestellten Honeypots wurden nach umfassender Analyse der verfügbaren Software nach Kriterien wie Aktivität, Qualität der gesammelten Daten und Erweiterbarkeit ausgesucht. Alle präsentierten Produkte haben in einem zeitgemäßen Honeypot-Setup ihren (teilweise sehr spezifischen) Platz und können als Teil einer umfassenderen, hybriden Lösung eingesetzt werden. Der Vergleich zwischen `shelia` und `argos` hat zudem gezeigt, wie sich der Fokus in der Entwicklung der High-Interaction-Honeypots mehr auf das schnelle, automatische Erkennen von Kompromittierungen verschoben hat. Bevor diese beiden Honeypots existierten, prüften Systeme wie *Capture-HPC NG* nach jeder Verbindung alle Veränderungen im Festplattenimage und überließen es dem Administrator, gültige und ungültige Zustände zu unterscheiden.

Weiterhin wurden Honeypots für Bereiche beleuchtet, die sich erst in jüngster Vergangenheit als Gefahrenquelle hervorgetan haben: Mobile Systeme wie Smartphones und Tablets, sowie über SCADA-Netzwerke miteinander verbundene industrielle Steuerungsanlagen. In beiden Fällen gab es bereits Konzepte und Prototypen entsprechender Honeypots, in der Praxis einsetzbare Software ist jedoch noch gar nicht vorhanden oder wie im Falle des *conpot*-Projektes gerade erst in einer frühen Version erschienen.

4 Planung und Realisierung von Honeypots/Honeynets

Nach einer ausführlichen Evaluation der verfügbaren Honeypot-Lösungen sollen einige von diesen nun in größerem Maßstab als nur an einem DSL-Privatanschluss, wie bereits in Kapitel 3.1.1 mit dem **kippo**-Honeypot durchgeführt, erprobt werden. Ziel dieses Experiments ist es, herauszufinden, wie gut sich ausgewählte Honeypotlösungen miteinander kombinieren und in ein Gesamtsystem integrieren lassen, wie die anfallenden Datenmengen ausgewertet werden können und wo die Grenzen einer solchen Installation liegen. Das Projekt sah vor, Honeypots weltweit verteilt in verschiedenen Netzwerken zu platzieren und damit möglichst viele verschiedene Länder abzudecken, um Bedrohungen in globalem Umfang analysieren zu können. Dieses Kapitel beschreibt die Architektur und die Implementierung dieses Systems.

4.1 Vorüberlegungen

Da es für Privatpersonen nicht ohne Weiteres möglich ist, IP-Adressen und Hosts in aller Welt ohne erheblichen finanziellen Aufwand zu akquirieren, stellte die Fakultät Informatik der TU Dresden für dieses Projekt einen Zugang zum **PlanetLab**[53] zur Verfügung, einem seit 2002 existierenden globalen Forschungsnetzwerk, an dem über 1000 Universitäten und Forschungseinrichtungen beteiligt sind [22]. Jede sich am Netzwerk beteiligende Institution stellt mindestens zwei mit dem Internet verbundene Systeme bereit, womit verteilte Anwendungen wie beispielsweise neu entwickelte Overlaynetzwerke getestet werden können.

Für dieses Experiment wurde ein neues sog. *Slice* angelegt, das im Prinzip eine Menge von nutzbaren Ressourcen, verteilt über verschiedene Knoten des PlanetLab, darstellt. Auf den von den Institutionen bereitgestellten Servern, im PlanetLab-Jargon als *Nodes* bezeichnet, werden schließlich dynamisch virtuelle Instanzen erstellt, sobald ein entsprechender Node zum Slice hinzugefügt wird. Diese virtuellen Instanzen werden wiederum als *Slivers* bezeichnet und sind als *Linux-VServer*[54] realisiert.

Auf der Basis der Gegebenheiten des PlanetLab wurden bei der Realisierung eines weltweit verteilten Honeynets folgende Vorüberlegungen getroffen:

- Maximale Größe von bis zu 100 aktiven Hosts gleichzeitig. Angesichts des begrenzten Zeitrahmens dieser Arbeit und der Tatsache, dass es sich lediglich um ein Experiment zum Testen von Honeypot-Technologien handelt, wird diese Anzahl als oberstes Limit betrachtet. Da zudem an verschiedenen Tagen von den knapp über 1000 Nodes des PlanetLab immer 25-30% aus verschiedensten Gründen nicht erreichbar waren und jede Institution zwischen zwei und sechs Server bereitstellt, würde eine Erweiterung des Honeynets um weitere Hosts zu keiner nennenswerten zusätzlichen „Verbreitung“ im Sinne des Ziels, Honeypots an vielen verschiedenen Orten zu platzieren, führen, aber den Verwaltungsaufwand des Gesamtsystem signifikant erhöhen.
- Vollautomatische Installation zusätzlicher Nodes und automatische Integration in das Honeynet.
- Die limitierten Ressourcen der Slivers (512MB virtueller Arbeitsspeicher, 5GB Dateisystem) verhindern den sinnvollen Einsatz von High-Interaction-Honeypots. Da diese zudem auch immer ein Sicherheitsrisiko darstellen und umfassendes Monitoring und Containment-Maßnahmen erfordern, die bei einer großen Anzahl von Hosts nicht mehr von einer einzelnen Person gewährleistet werden können, werden im Rahmen dieses Experiments ausschließlich sparsamere Low-Interaction-Honeypots eingesetzt.
- Das zu errichtende Honeynet besteht ausschließlich aus serverseitigen Honeypots.
- Die zum Netzwerk gehörigen Knoten werden mit Hilfe eines Overlaynetzwerks überwacht und bei Bedarf gesteuert. Es bietet sich hierfür an, auf Technologien, die zum Verwalten von Botnetzwerken verwendet werden, zurückzugreifen. Um auf eventuelle Störfälle, wie beispielsweise Probleme durch falsch konfigurierte Knoten oder Angriffe auf die Slivers selbst reagieren zu können, werden über diesen gesonderten Kanal administrative Anweisungen gesendet. Somit ist es möglich, gezielt einzelne Dienste oder auch die Knoten selbst herunterzufahren oder Statusinformationen abzufragen.
- Identische angebotene Dienste auf allen Knoten. Um die gesammelten Daten von allen Knoten auch auswerten und vergleichen zu können, werden alle Systeme mit derselben Honeypot-Konfiguration versehen.

[53] `http://planet-lab.org/`
[54] `http://linux-vserver.org/`

4.2 Architektur

4.2.1 Virtualisierung

Bei einer Analyse einiger Slivers hat sich herausgestellt, dass die auf den PlanetLab-Knoten laufende Software heterogen war. Zwar handelte es sich ausschließlich um mit der Linux-Distribution **Fedora**[55] betriebene VServer, diese war jedoch in unterschiedlichen Versionen installiert. Vorgefunden wurden hauptsächlich Fedora 8 „Werewolf“ aus dem Jahr 2007 und Fedora 12 „Constantine“ aus dem Jahr 2009. Beide Distributionen waren zum Zeitpunkt des Experiments bereits stark veraltet. Eine Installation aktueller Honeypots auf einem System mit dieser Softwarebasis hätte erheblich Aufwand nach sich gezogen, da die meisten bereits installierten Softwarepakete hätten aktualisiert werden müssen. Ein Update auf die zu diesem Zeitpunkt aktuelle, viele Jahre jüngere Version Fedora 18 „Spherical Cow“ wäre über das Paketmanagement nur in mehreren iterativen Schritten möglich und mit erheblichem Zusatzaufwand verbunden, um überhaupt ein lauffähiges, modernes System zu erhalten. Zudem hat die Sliver-Analyse ergeben, dass sich selbst die VServer mit der gleichen Distributionsversion in ihrer Softwareausstattung und Konfiguration voneinander unterscheiden. So waren beispielsweise bei einigen Systemen keine gültigen Keys für die Paketverwaltung installiert, was dazu führte, dass die Installation jeglicher neuer Pakete standardmäßig fehlschlug.

Um dieses Problem zu lösen, musste in irgendeiner Form vom aktuellen System abstrahiert werden, da ein Update nicht in Frage kam. Die Fedora-basierten VServer waren bereits virtuelle Linux-Container - eine Möglichkeit, innerhalb dieser ein anderes System mit aktueller Software zu betreiben, war die Nutzung einer Virtualisierungstechnik. Dies wirkt sich zwar im Allgemeinen negativ auf die Performance des resultierenden Systems aus, erlaubt es jedoch, ein zuvor vorbereitetes Festplattenabbild auf beliebig vielen Hosts gleichzeitig in Betrieb zu nehmen. Das darunterliegende System muss lediglich für die angedachte Virtualisierungstechnik vorbereitet sein. Zusätzlich bietet diese Variante den Vorteil, dass nicht jeder Host alle zum Einsatz kommende Software neu herunterladen und installieren muss. Das Honeypot-System ist in Form des Plattenabbildes bereits vollständig konfiguriert und muss nur noch gestartet werden. Diese Überlegungen führten schließlich zu der Frage, welche Software zur Virtualisierung geeignet wäre.

Die PlanetLab-Nodes waren Server verschiedenster Leistungsklassen. Damit die Virtualisierungslösung skalierbar bleibt, sollte eine möglichst ressourcenschonende Software zum Einsatz kommen. Da es innerhalb der bereitgestellten VServer nicht möglich war, den aktiven Linux-Kernel auszutauschen, umzukonfigurieren oder Module nachzuladen, waren gängige Produkte wie *Oracle VirtualBox* oder *VMware* nicht einsatzbar. Die linuxspezifische, weitverbreitete Kernel Virtual Machine (KVM) war hingegen nur auf Hardware lauffähig, die spezielle Virtualisierungserweiterungen unterstützte. Diese war wiederum nicht auf allen PlanetLab-Nodes vorhanden. Eine sehr leichtgewichtige, einfache Alternative zu den zuvor Genannten stellte hingegen **User Mode Linux**[56] (UML) dar. Im Prinzip handelt es sich dabei um einen speziell konfigurierten Linux-Kernel, der als normaler Prozess innerhalb eines anderen Linux-Systems ausgeführt werden kann. Dieses Verfahren ist somit kernelunabhängig und findet vollständig im Userspace statt. Die Konfiguration erfolgt über Kommandozeilenparameter beim Starten des UML-Kernels, somit lässt sich die Menge des innerhalb der virtuellen Umgebung zur Verfügung stehenden Arbeitsspeichers, die Netzwerkkonfiguration oder das zu verwendende Festplattenabbild einstellen. Da solche UML-Kernel an das Linux-System, auf dem sie gestartet werden, keine besonderen Anforderun-

[55] `http://fedoraproject.org/`
[56] `http://user-mode-linux.sourceforge.net`

gen stellen, wurde dieses Lösung im Rahmen dieses Experiments ausgewählt. Als im virtuellen Container laufende Linux-Distribution wurde **Debian GNU/Linux** ausgewählt, da mit dieser schon im Vorfeld bei der Evaluation der individuellen Honeypots gearbeitet wurde.

4.2.2 Honeypot-Auswahl

Nachdem feststand, dass auf allen PlanetLab-Knoten, die Teil des Experiments werden sollten, ein via UML virtualisiertes Debian laufen würde, folgten Überlegungen zu den einzusetzenden serverseitigen Low-Interaction-Honeypots. Da mehrere PlanetLab-Slices auf einem System (und damit einer einzigen IP-Adresse) gleichzeitig betrieben werden konnten, waren nicht immer alle TCP- und UDP-Ports von außerhalb erreichbar. Zudem begrenzten einige Institutionen den Zugriff zusätzlich noch mit vorgeschaltenen Firewallregeln. So können beispielsweise auf den PlanetLab-Slices der TU-Dresden ausschließlich Ports mit Portnummer größer als 1024 für Serverdienste genutzt werden. Um trotz derartiger Einschränkungen Angreifern möglichst viele Angriffsvektoren anzubieten, was die Menge der gesammelten Daten erhöhen sollte, wurde eine Kombination verschiedener Honeypots ausgewählt: **dionaea**, das ein breites Spektrum an Protokollen emuliert und auch komplexere Interaktion (SQL-Abfragen, SMB/CIFS-Freigaben) ermöglicht, **amun**, das ausschließlich Sicherheitslücken verschiedenster Dienste simuliert und **kippo**, das von allen evaluierten Honeypots die beste SSH-Implementierung anbietet. Die nachfolgenden Tabellen listen alle Dienste/Verwundbarkeiten auf, die von den Honeypots im Experiment angeboten werden sollten.

Port	**Service**	**Port**	**Service**
5060 / TCP	SIP	5061 / TCP	SIP
135 / TCP	epmapper	3306 / TCP	MySQL
1433 / TCP	MSSQL	445 / TCP	SMB/CIFS
5060 / UDP	SIP	69	TFTP

Tabelle 3: Aktive dionaea-Services

Die in Tabelle 3 aufgelisteten Dienste stellen nur einen kleinen Teil aller mit `dionaea` emulierbaren Protokolle dar. Die Auswahl erfolgte nach der Genauigkeit der Emulation gemäß [7] und der Qualität der gesammelten Daten. Da das dionaea-Projekt den Hauptfokus auf das SMB-Protokoll legt und darüber viel Malware verbreitet wird, ist dieses selbstverständlich aktiviert. Die *MySQL*- und *MSSQL*-Emulationen ermöglichen eine Interaktion über das reine Simulieren von Schwachstellen hinaus: So ist es beispielsweise möglich, sich mit einem *MySQL*-Client mit einem solchen Service zu verbinden und einfache SQL-Abfragen abzusetzen. Das SIP-Protokoll wird für *VOIP* eingesetzt, kann ohne einen zusätzlichen VOIP-Server allerdings nur begrenzt mit Angreifern interagieren.

Der **kippo**-Honeypot ist ausschließlich auf das SSH-Protokoll spezialisiert und bietet deshalb nur auf einem TCP-Port genau diesen Dienst an. Da alle PlanetLab-Knoten jedoch Port 22 bereits mit einem SSH-Dienst belegt haben, wurde der SSH-Honeypot auf Port 2222 verlegt.

Die in der Konfiguration des Honeypots amun aktivierten Module, die jeweils eine separate Sicherheitslücke eines Dienstes emulieren, sind in Tabelle 4 aufgelistet. Bei dieser Auswahl handelt es sich im Wesentlich um die mitgelieferte Standardkonfiguration abzüglich der Module, die auf Ports arbeiten, die bereits durch den dionaea-Honeypot abgedeckt wurden. Eine genauere Spezifikation der Sicherheits-

Port	Vulnerability-Modul	Port	Vulnerability-Modul
139	vuln-netdde	10203	vuln-ca
21	vuln-ftpd	1023, 5554	vuln-sasserftpd
42	vuln-wins	6070, 41523, 1900	vuln-arc
2967, 2968	vuln-symantec	3372, 1025	vuln-msdtc
110	vuln-axigen	110	vuln-slmail
110	vuln-mdaemon	5000, 2555	vuln-upnp
443	vuln-iis	9999	vuln-maxdb
8080, 1111, 1581	vuln-tivoli	2103, 2105, 2107	vuln-msmq
27347	vuln-sub7	25, 587	vuln-imail
105	vuln-mercury	143	vuln-lotusdomino
617	vuln-arkeia	6129	vuln-dameware
139	vuln-netbiosname	6101	vuln-veritas
5168, 3268	vuln-trend	2745	vuln-bagle
2380	vuln-goodtech	554	vuln-helix
2954	vuln-hpopenview	80	vuln-http
3127	vuln-mydoom		

Tabelle 4: Aktive amun-Vulnerability-Module

lücken, die von diesen Modulen emuliert werden, war leider sehr schwierig. Der Code der Module selbst enthielt keine weiteren Hinweise und auch die Website des Projekts gab keine weitere Auskunft.

Alle drei Honeypots basieren auf der Programmiersprache Python und haben somit gemeinsame Abhängigkeiten. Dadurch verringerte sich die Größe des resultierenden Festplattenabbildes, das schlussendlich von jedem Knoten gesondert heruntergeladen werden musste.

4.2.3 Überwachung

Obwohl die einzelnen Honeypot-Knoten ihre Arbeit möglichst autonom verrichten sollten, war das Errichten einer Überwachtungsinfrastruktur nötig, um den Zustand der einzelnen Systeme beobachten und auf plötzliche unerwartete Ereignisse reagieren zu können. Beispielsweise war die limitierte Festplattenkapazität einiger UML-Instanzen ein Problem, da je nach Intensität der Angriffe auf die Systeme mehr oder weniger Daten gespeichert wurden und dies ohne externe Kontrolle unweigerlich zum Vollaufen des Dateisystems geführt hätte.

Die Recherche nach Verfahren zum Monitoring und zur Steuerung verteilter Systeme, in diesem Fall von solchen, die sich in verschiedenen Netzwerken befinden und ausschließlich über das Internet miteinander kommunizieren können, verlief zunächst schleppend. Im Linux-Umfeld verbreitete Software wie das freie *Nagios*[57] oder das kommerzielle *Puppet Enterprise*[58] waren sehr komplex und würden nicht zur Philosophie, möglichst platzsparend und ressourcenschonend zu arbeiten, passen. Aus diesem Grund wurde schließlich ein Ansatz gewählt, der häufig zur Steuerung von Botnets verwendet wird: Die Nutzung des IRC-Protokolls (*Internet Relay Chat*). Das textbasierte Chatsystem ist eine Variante, mit der Kriminel-

[57] `http://nagios.org/`
[58] `http://puppetlabs.com`

le häufig übernomme Rechnersysteme, sogenannte *Zombies*, von einer zentralen Stelle aus zu steuern versuchen [33]. Solche privaten IRC-Server werden als C&C[59]-Server bezeichnet. Jeder Bot verbindet sich zu diesem und tritt einem Chatraum, die als *Channels* bezeichnet werden, bei. Dort überwacht ein IRC-Bot die Anwesenheit und sendet Befehle, die dann von den Bots ausgeführt werden. Da das IRC-Protokoll ausschließlich textbasierte Kommunikation ermöglicht, entsteht bei dessen Nutzung nur ein geringer Overhead. Zudem verursacht die Integration eines IRC-Clients, der auf jedem PlanetLab-Node laufen würde, nur geringen Aufwand: Es existieren bereits viele Bibliotheken und Projekte, die das offene Protokoll implementieren und genutzt werden können.

Im Rahmen dieses Experiments wurde das auf Python basierende **irckit**[60] eingesetzt, eine Implementierung eines einfachen IRC-Botnets. Die Software stellt eine quelloffene Bibliothek mit grundlegender IRC-Funktionalität und darauf basierenden Skripten zum Einrichten eines Botnet-Masters (als „Boss“ bezeichnet) und der einzelnen Bots (als „Worker“ bezeichnet) dar. Auf jeder virtuellen UML-Instanz sollte ein auf dieser Software basierender IRC-Client laufen und permanent Verbindung zu einem zuvor angelegten IRC-Channel halten, in dem der „Boss“ alle „Worker“ regelmäßig anpingt und auszuführende Kommandos mitteilt. Der „Boss“ erlaubt zudem nur IRC-Benutzern, die sich zuvor mit einem Passwort authentifiziert haben, den Status des Netzwerks abzufragen oder Kommandos zu versenden. Somit wird sichergestellt, dass niemand Unbeteiligtes die Kontrolle über das Honeypot-Netz übernimmt, wenn gerade kein Administrator anwesend ist. Die Überwachung via IRC bietet zudem den Vorteil, dass entsprechende IRC-Clients, die für die Administration des Honeypot-Netzwerkes benötigt werden, für alle Plattformen verfügbar sind. So kann zu jedem Zeitpunkt anhand der im IRC-Channel angemeldeten Nutzer, welche PlanetLab-Slivers gerade online sind. Zusätzlich erlaubt das System ein kontrolliertes Herunterfahren aller UML-Instanzen am Ende des Experiments.

4.2.4 Protokollierung

Der Wert von Honeypots liegt, wie in Kapitel 1 bereits besprochen, unter anderem in den gesammelten Daten über Angreifer und ihre Vorgehensweise. Auf jedem PlanetLab-Sliver waren drei Low-Interaction-Honeypots installiert, deren Daten in irgendeiner Form aggregiert und an zentraler Stelle ausgewertet werden mussten. Zur Lösung dieser Fragestellung bot es sich an, das Grundprinzip des in Kapitel 2.5 vorgestellten **Collapsar**-Systems zu übernehmen. Sogenannte „Sensoren“ sind über GRE-Tunnel mit einer zentralen Honeypot-Farm verbunden und leiten jeglichen eingehenden Traffic weiter. Der entscheidende Unterschied zum PlanetLab-Experiment war jedoch, dass aufgrund der stark schwankenden Bandbreite und Latenz der individuellen Nodes ein solches System nur schwer realisierbar war. Zudem wäre ein Server nötig gewesen, der jeglichen ein- und ausgehenden Datenverkehr aller Knoten zeitnah verarbeiten kann. Eine solche Infrastruktur war im Rahmen dieser Arbeit nicht vorhanden. Aus diesem Grund wurden die Honeypots in diesem praktischen Versuch auf die jeweiligen PlanetLab-Slivers verschoben. Dies bot den Vorteil, dass Anfragen sofort mit der für jeden Knoten individuellen Bandbreite und Latenz beantwortet werden konnten und Angreifer keinen Unterschied zwischen dem Honeypot und anderen Hosts im jeweiligen Netzwerk erkennen konnten. Zudem wurde die verfügbare Rechenleistung der PlanetLab-Knoten genutzt, wodurch sich ein im Betrieb effizienteres System ergab. Der Nachteil dieser Lösung ist jedoch ein erheblicher Administrationsaufwand: Da die einzelnen Sensoren deutlich komplexer aufgebaut waren als bei Collapsar, erfordert eine Veränderung der Konfiguration oder das Update oder Hin-

[59] Command and Control

[60] `https://github.com/coleifer/irc/`

zufügen einer Honeypot-Software die wiederholte Durchführung der entsprechenden Schritte auf jedem einzelnen Knoten. Bei Collapsar hingegen sind die Honeypots nur einmalig auf der Serverseite installiert und können ohne Weiteres ausgetauscht werden.

Da die Honeypots also auf allen PlanetLab-Knoten separat liefen, ergaben sich zwei Möglichkeiten für die Protokollierung:

- Alle verwendeten Honeypots werden mit Modulen ausgeliefert, die alle Protokolldaten an einen **MySQL**-Server schicken. Ein solcher zentraler Server würde das Auswerten der Daten stark vereinfachen. Die schiere Menge an Daten, die von bis zu 100 Knoten mit jeweils drei Honeypots gesendet werden könnten, erforderte jedoch erneut einen Server mit großer Bandbreite.
- Alle Honeypots speichern ihre Protokolle lokal auf den jeweiligen PlanetLab-Slivers. Am Ende des Experiments oder, falls zu viele Daten anfallen und das Dateisystem zu voll wird, schon vorher, werden die Daten über **SSH/SFTP** eingesammelt.

Da ein für die MySQL-Lösnug entsprechend angebundener Server nicht vorhanden war, wurde diese Variante verworfen und auf SSH/SFTP zurückgegriffen. Mit Hilfe des im vorigen Kapitel beschriebenen IRC-Überwachungssystems wurde in regelmäßigen Abständen der verbleibende Speicherplatz der UML-Instanzen überprüft und Daten bei Bedarf manuell verschoben. Das Einsammeln aller Protokolldaten am Ende des Experiments übernahm der *Logfile Collector*, ein separates System, das sich über SSH mit allen UML-Instanzen verband und die aktuellen Protokolldaten verschob.

Um jedoch zusätzlich zu den großen Mengen an Logfiles noch einen groben Überblick über die gesammelten Daten zu erhalten, wurde ein weiteres Protokollierungssystem aufgesetzt, das sich an Collapsar orientierte: Die Software **SURFcert IDS** basiert ebenfalls auf diesem Prinzip, ihre Autoren beschreiben sie als „an open source Distributed Intrusion Detection System based on passive sensors“[61]. Auch hier laufen alle zum Einsatz kommenden Honeypots (`dionaea`, `amun` und `argos`) auf einem zentralen Server, zu dem alle Sensoren einen VPN-Tunnel geöffnet halten und alle ein- und ausgehenden Daten darüber weiterleiten. Der entscheidende Unterschied ist jedoch die Existenz eines separaten **Logging Servers**, auf dem eine *PostgreSQL*-Datenbank und ein Webinterface installiert sind, das die gesammelten Daten aufbereitet. Für das vorliegende Experiment wurde ausschließlich diese Komponente des SurfCERT IDS auf einem gesonderten Server, der nicht Teil des PlanetLab war, installiert. Alle eingesetzten Honeypots boten standardmäßig oder mit Hilfe eines zusätzlichen Plugins Unterstützung für das SurfCERT-System an. Im Unterschied zum Schicken der Protokolldaten an einen MySQL-Server wurden an den SurfCERT-Logging-Server nur wenige Informationen über Angriffe gesendet, darunter die Quelle und das Ziel eines jeden Angriffs, sowie einige protokollabhängige zusätzliche Metadaten. Die anfallenden Datenmengen und damit die Anforderungen an die Bandbreite des Logging-Servers waren somit geringer als bei der Nutzung der MySQL-Module.

Einen Überblick der aus diesen Überlegungen resultierenden Architektur gibt Abbildung 3.

[61] `http://ids.surfnet.nl/wiki/doku.php`

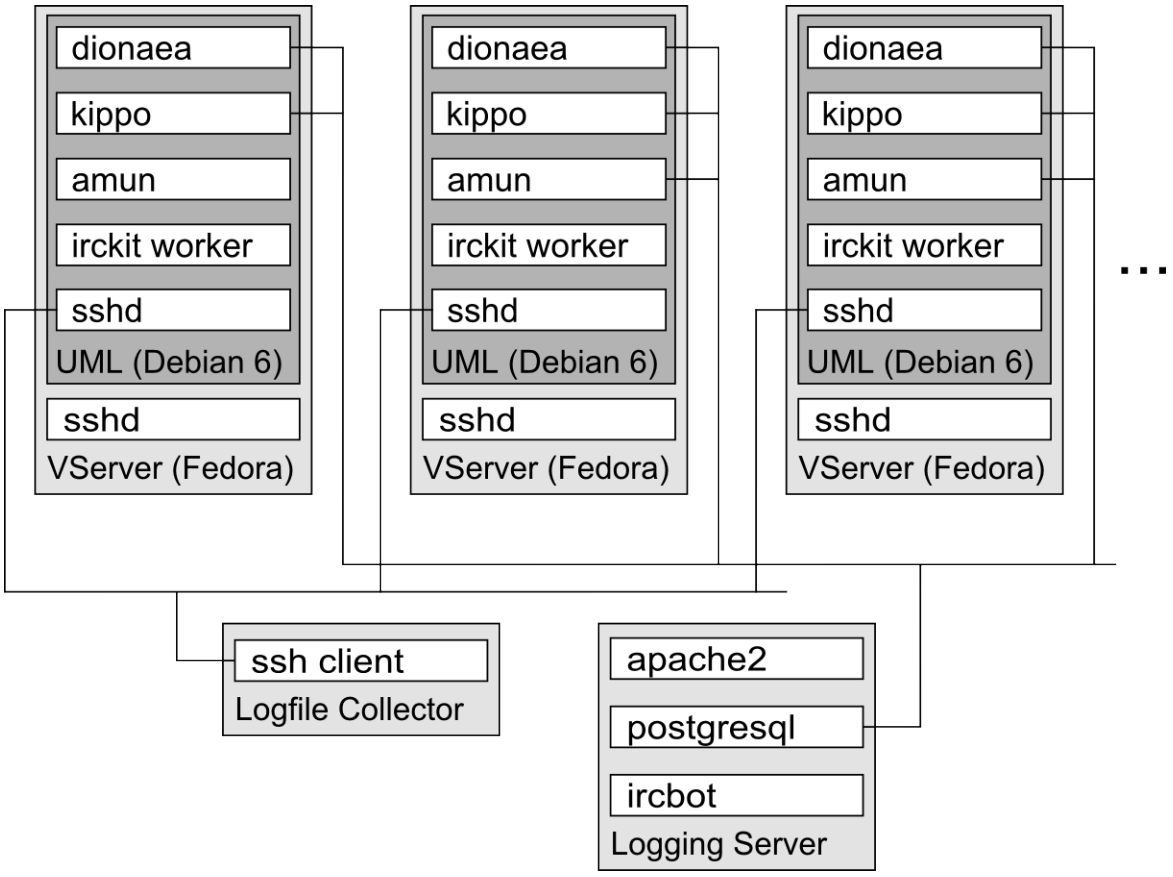

Abbildung 3: Die Komponenten des verteilten Honeypot-Netzwerks

4.3 Implementierung

Dieser Abschnitt beschreibt einige der Hürden, die beim Aufbau des Honeypot-Netzwerks genommen werden mussten. Es wird versucht, die chronologische Reihenfolge der hier beschriebenen Ereignisse beizubehalten. Zunächst wird der Installationsprozess des Logging-Servers und eines neuen Honeypots ausführlich beschrieben. Der darauffolgende Abschnitt beschäftigt sich mit den Maßnahmen, die zur Wartung und Überwachung des Systems unternommen wurden.

4.3.1 Installation

Logging Server Zuerst wurde alle erforderliche Software auf dem *Logging Server* installiert, der zugleich auch als Master-IRC-Bot (oder „Boss“) dienen sollte. Auf der Maschine lief ein *Debian GNU/-Linux* in der Version 6. Die Installation der Logging-Server-Komponente des SurfCERT IDS war unter dieser Distribution trivial, da die Entwickler ein Repository mit Paketen für Debian bereitstellten und der Installationsprozess auf der Website des Projekts sehr detailliert beschrieben war[62]. Das Webinterface wurde zusätzlich mit HTTP-Authentifizierung abgesichert.

[62] `http://ids.surfnet.nl/wiki/doku.php?id=latest_docs:1_logging_server:1._installation`

Der IRC-Bot sollte nach Möglichkeit permanent online sein, da die auf dem *irckit* basierenden „Worker“ nur bei dessen Anwesenheit den Kontrollkanal betreten können. Um dies zu gewährleisten, wurde die Software wie auch schon zuvor die SurfCERT-Software auf einem gemieteten, privaten VServer installiert, dessen Betreiber eine hohe Verfügbarkeit garantierte. Ihm war eine statische IP-Adresse und ein eindeutiger Domainname zugewiesen, über den er von allen Sensoren erreicht werden konnte. Vor der Installation der Software wurde zunächst noch ein IRC-Channel benötigt, in dem sich die Worker versammeln konnten. Da das Aufsetzen eines eigenen IRC-Server nicht trivial ist und bereits unzählige öffentliche Server verfügbar waren, wurde im kleinen Rahmen dieses Experiments auf einen solchen zurückgegriffen: Auf dem *Freenode* IRC Server[63] wurde ein neuer Channel mit dem Namen `#hpnettud` angelegt. Zur Installation des *irckit* genügte die Einrichtung von *Pip*, einem Paketmanagementsystem für Python. Damit ließen sich *irckit* und dessen Abhängigkeiten ohne großen Aufwand nachinstallieren. Der IRC-Bot konnte im Anschluss durch Ausführen der Datei `boss.py` gestartet werden. Die Konfiguration erfolgte ausschließlich durch Kommandozeilenparameter, die u.a. den Benutzernamen des Bots im IRC-Netzwerk, den zu betretenden IRC-Channel und das Passwort, mit dem sich ein Benutzer dem Bot gegenüber als Administrator authentifizieren konnte, umfassten. Der IRC-Channel `#hpnettud` diente in diesem Fall als Kommandokanal, die Worker versammelten sich im Kanal `#hpnettud-cmd`, in dem die Kommunikation zwischen Master und Worker abgewickelt wurde. Da der Boss im laufenden Betrieb bei zu vielen gleichzeitig anwesenden Workern in unregelmäßigen Abständen wegen „Flooding“, also zu vielen gleichen gesendeten Nachrichten, vom Freenode-Netzwerk getrennt wurde, war das Einrichten eines kleinen Skriptes notwendig, das via `cron` alle 10 Minuten ausgeführt wurde und `boss.py` neu startete, falls es nicht mehr lief. Das IRC-Skript wurde zudem in einem `screen`-Terminal gestartet, damit es auch ohne aktive SSH-Verbindung zum Server weiterlief.

Sensoren - Virtuelles System Die Einrichtung der Sensoren erfolgte in zwei separaten Schritten: Der Vorbereitung des Festplattenabbildes der UML-Instanz und dem Schreiben und Ausführen von Skripten, die das vollautomatische Installieren neuer Sensoren übernahmen.

Als Grundlage der virtuellen UML-Lösung diente ein vorbereiteter UML-Kernel von `http://uml.devloop.org.uk/kernels.html` in der Version 2.6.39.4. Der letzte Kernel der Linux 2.6er-Serie kam zum Einsatz, da die getesteten Kernel ab Version 3.0 bei vorherigen Tests instabil liefen. Ein virtuelles Dateisystem eines minimalen *Debian Squeeze*-Systems wurde zudem von `http://fs.devloop.org.uk/` bezogen. Die UML-Instanz wurde ausschließlich über Kommandozeilenparameter des ausführbaren Kernels konfiguriert. Bei der Vorbereitung des Debiansystems wurden zunächst die bereits in Kapitel 3 beschriebenen Honeypots `dionaea`, `amun` und `kippo` installiert und konfiguriert. Um alle Abhängigkeiten zu erfüllen, wurde dafür teilweise auf das Debian `testing`-Repository zurückgegriffen. Alle Honeypots speicherten ihre Protokolle lokal im Dateisystem und sendeten zusätzlich Informationen an den Logging Server. Mit `amun` wurde bereits ein funktionierendes Modul für *SurfCERT IDS*-Unterstützung mitgeliefert, es musste lediglich aktiviert und die Adresse und Zugangsdaten zur *PostgreSQL*-Datenbank des Logging Servers eingetragen werden. `Dionaea` enthielt ebenfalls ein solches Plugin, dieses war jedoch für eine ältere Version des *SurfCERT IDS* bestimmt und funktionierte erst nach einigen Modifikationen bezüglich der in der Datenbank verwendeten Tabellennamen und Parameter. Für `kippo` stellten die Entwickler des *SurfCERT IDS* ein eigenes Modul zur Verfügung, das lediglich in das Verzeichnis des Honeypots kopiert und konfiguriert werden musste[64]. Nach der Installation aller Honeypots wurde mit Tests sichergestellt, dass das virtuelle System Daten über eingehende Verbindungen zum

[63] `http://freenode.net`
[64] `http://ids.surfnet.nl/wiki/doku.php?id=kb:installing_kippo`

Logging Server sendete. Zuletzt wurde für jeden Honeypot ein Init-Skript geschrieben, was das Starten und Beenden der Dienste im Rahmen anderer Skripte und über die IRC-Schnittstelle stark vereinfachte.

Zusätzlich zu den Honeypots wurde erneut das *irckit* installiert. Analog zum „Boss“ wurden die Worker durch das Ausführen der Datei `worker.py` mit entsprechenden Kommandozeilenparametern gestartet. Da das UML-Festplattenimage nur maximal 1024MB Platz bot, wurden die Protokolle des *irckit* deaktiviert. Zusätzlich wurde dem Worker beim Start noch der Benutzername des „Bosses“ auf dem IRC-Server mitgegeben, da diese sich bei diesem Registrieren mussten, um dem IRC-Channel beitreten zu können. Da das Festplattenimage möglichst generisch sein sollte, wurde der zu verwendende Benutzername auf dem IRC-Server vom lokalen `hostname` einer jeden Instanz abgeleitet: Um den Konventionen des *Freenode*-Servers gerecht zu werden, wurden alle Punkte im Hostnamen durch Pipe-Symbole (`|`) ersetzt und nur die letzten 16 Zeichen für den Benutzernamen verwendet (Maximallänge). Wenn die UML-Instanz aus irgendeinem Grund die Verbindung zum IRC-Server kurzzeitig verlor, stellte sie die Verbindung nicht automatisch wieder her. Um dem entgegenzuwirken, wurde ein init-Skript zum Starten und Stoppen des IRC-Bots geschrieben, das via eines cron-Jobs periodisch zu jeder vollen Stunde ausgeführt wurde und somit ein Reinitialisieren des Workers erzwang.

Um das Einsammeln der Protokolldaten der Honeypots zu ermöglichen, wurde zudem ein Skript namens `collectsensordata` verfasst, das zuerst alle drei laufenden Honeypots beendete, alle Logdateien in ein neues Verzeichnis verschob und anschließend alle Honeypots neu startete. Ein Host, der via SSH die Daten einsammeln wollte, musste lediglich das Skript ausführen und das resultierende Verzeichnis via SCP oder SFTP kopieren.

Damit alle gesammelten Protokolldaten miteinander vergleichbar waren, wurde die Systemzeit bewusst auf UTC[65] belassen und mit der Installation eines NTP-Clients[66] die Synchronisation sichergestellt. Zuletzt wurde noch das Skript `changeip` geschrieben, das während der Installation des Sensors verwendet wurde, um die IP-Adresse des Referenzssystems durch die externe IP des Ziel-PlanetLab-Slivers zu ersetzen. Dies war notwendig, weil die *SurfCERT IDS*-Module die Sensoren anhand ihrer IP-Adresse unterschieden. Zur entfernten Administration wurde außerdem ein *OpenSSH*-Server auf Port 22222 aufgesetzt, da Port 22 bereits durch den SSH-Dienst des PlanetLab-Knotens belegt war.

Nachdem das virtuelle System zufriedenstellend lief, wurden alle überflüssigen temporären Daten und Logdateien entfernt, um das Abbild so klein wie möglich zu halten. Anschließend wurde der freie Speicherplatz der Partition mit Nullen vollgeschrieben, was der späteren Kompression des Installationspakets dienlich war.

Sensoren - Automatische Installation Nachdem die UML-Instanz mit Kernel und Festplattenimage vorbereitet war, wurde eine Reihe von Skripten geschrieben, die die Installation neuer PlanetLab-Knoten automatisiert durchführen konnten. Zunächst musste jedoch die Netzwerkanbindung der virtuellen UML-Instanzen sichergestellt werden. Beim Analysieren der PlanetLab-Slivers wurde festgestellt, dass es mangels Zugriffsberechtigungen auf die virtuelle Gerätedatei `/dev/net/tun` nicht möglich war, virtuelle Netzwerkschnittstellen zu erstellen, die mit dem externen Interface `eth0` zu einer Netzwerkbrücke verbunden worden wären. Folglich war die Einrichtung eines VPN-Tunnels auf dem PlanetLab-Sliver selbst nicht möglich, da dieses zwingend eine virtuelle Netzwerkschnittstelle voraussetzt. Da `tun`/`tap`-

[65] *Coordinated Universal Time*, Weltzeit

[66] Das *Network Time Protocol* ist ein Standard zur Synchronisierung von Uhren in Computersystemen (Quelle: Wikipedia)

Devices auch die bevorzugte Variante sind, um eine UML-Instanz in ein Netzwerk einzuhängen, musste eine Alternative gefunden werden. Die Recherche ergab, dass sich das Tool `slirp`[67] für diesen recht speziellen Fall eignete. Die Software startete zusammen mit der UML-Instanz als zusätzlicher Prozess auf dem PlanetLab-Sliver und übernahm jeglichen IP-Datenverkehr für ein virtuelles Netzwerkinterface. Es konnte ohne Root-Rechte ausgeführt werden, benötigte keine zusätzlichen virtuellen Schnittstellen, lief komplett im Userspace und war somit aus Sicht des Kernels ein normaler Prozess, der mit dem Netzwerk kommunizierte. `Slirp` brachte zudem eine entscheidende zusätzliche Funktionalität mit: Es konnte Ports auf dem Hostsystem abhören und jeglichen Datenverkehr, der an diese gesendet wurde, an die UML-Instanz weiterleiten. Diese Funktionalität ähnelte herkömmlichem Masquerading und erlaubte es, die im virtuellen System laufenden Honeypots nach außen hin sichtbar zu machen. Leider war die Anwendung in keinem *Fedora*-Repository vorhanden und musste deshalb auf allen Knoten aus den Quellen kompiliert werden. Im Auslieferungszustand limitierte `slirp` außerdem die maximale Bandbreite, was durch Rekonfiguration vor dem Kompilieren jedoch behoben werden konnte.

Die automatische Installation eines neuen Sensors konnte durch das Skript
`remote_setup.sh <hostname>` gestartet werden. Sie lief in folgenden Schritten ab:

1. In der **Stage 1** wurde die Datei `setup.ssh` zusammen mit einem öffentlichen Login-Key auf den PlanetLab-Sliver kopiert und ausgeführt. Obwohl für diese Schritte bereits der vom PlanetLab zur Verfügung gestellte SSH-Server genutzt wurde, installierte dieses Skript einen weiteren *OpenSSH*-Dienst aus der Fedora-Paketverwaltung und startete ihn auf Port 2223. Der mitgelieferte Schlüssel erlaubte das passwortlose Login für den weiteren Installationsverlauf. Der Grund für diesen Schritt war, dass es über den bereits vorhandenen SSH-Zugang nicht möglich war, *GNU screen* zu benutzen. Dieses Tool war allerdings essentieller Bestandteil des UML-Systems.
2. **Phase 2** war am Umfangreichsten und beinhaltete die Installation der UML-Instanz. Das entsprechende Skript `setup.ssh` wurde via SSH auf den PlanetLab-Sliver kopiert und dort anschließend mit dem Hostnamen und der IP-Adresse des PlanetLab-Slivers als Argumente ausgeführt. Es unternahm die folgenden Schritte:
   (a) Installation von *GNU screen* über das Fedora-Softwaremanagement, das virtuelle Konsolensitzungen erzeugen konnte, in denen die UML-Instanzen liefen.
   (b) Installation der Softwaregruppen *Development Tools* und *Development Libraries* des Fedora-Softwaremanagements. Dies erfüllte die Voraussetzungen, um *slirp* im Anschluss kompilieren zu können.
   (c) Herunterladen und Entpacken des Installationsarchivs `hpsensor-0.2.tar.bz2`. In diesem befand sich der mit bzip2 komprimierte UML-Kernel, das Festplattenabbild, der *slirp*-Quellcode und eine Reihe von Skripten zur Installation und zum Betrieb des virtuellen Systems. Das Archiv zum Herunterladen wurde hierfür auf dem *Logging Server* platziert und via HTTP zugänglich gemacht.
   (d) Das Kompilieren von *slirp* übernahm ein weiteres gesondertes Skript, das den im Archiv mitgelieferten, bereits gepatchten Quellcode konfigurierte und kompilierte. Zuvor wurde in der Datei `config.h` noch `FULL_BOLD` aktiviert, womit *slirp* die maximale Bandbreite des Netzwerkinterfaces nutzen konnte.
   (e) Hinzufügen eines *cron*-Jobs zum periodischen Ausführen des Skriptes `start.sh`. Da UML nicht auf allen PlanetLab-Knoten stabil lief und eine ganze Reihe von Problemen auftreten konnten (auch slirp lief nicht immer perfekt und leitete unter Umständen Pakete nicht mehr weiter), überprüfte dieses Skript eine Reihe von Bedingungen und startete die virtuelle In-

[67] `http://slirp.sourceforge.net/`

stanz, wenn notwendig, neu.

(f) *Cron* war auf den PlanetLab-Slivers vorinstalliert, aber nicht aktiviert. Deswegen wurde der Dienst nun in den Runleveln 2 bis 5 gestartet.

(g) Der Sensor wurde erstmalig gebootet und es wurden 180 Sekunden abgewartet, um sicherzustellen, dass der Bootvorgang abgeschlossen war. Einige wenige PlanetLab-Slivers hatten sehr wenig CPU-Kapazitäten, in diesen Fällen musste die Wartezeit noch erhöht werden.

(h) Ausführen des Skriptes `sethostname.sh`, das sich via SSH mit der UML-Instanz verband und den Hostnamen des Systems auf den Hostnamen des PlanetLab-Slivers änderte. Zusätzlich wurde die IP-Adresse der virtuellen Instanz mit Hilfe des zuvor beschriebenen `changeip.sh`-Skriptes auf die externe IP-Adresse des PlanetLab-Slivers geändert. Dies war nötig, weil die Honeypot-Plugins, die Daten zum *Logging Server* sendeten, als „ID“ des Sensors dessen IP-Adresse verwendeten. Damit auf Serverseite also eine eindeutige Zuordnung möglich war, musste diese einzigartig sein.

3. In **Phase 3** wurde der neu eingerichtete Sensor beim *Logging Server* registriert. Die *PostgreSQL*-Datenbank speicherte nur ankommende Daten von zuvor registrierten Sensoren. Falls die von den Entwicklern vorgesehene Architektur mit OpenVPN-Tunneln verwendet worden wäre, hätten sich die Sensoren über diesen automatisch registriert. Da diese Komponente im Experiment nicht verwendet wurde, war jedoch eine manuelle Registrierung aller Sensoren nötig, um zu verhindern, dass andere Quellen die gesammelten Daten verfälschten. Zu diesem Zweck wurde auf dem *Logging Server* das mitgelieferte Skript `localsensor.pl` verwendet, das Sensoren anhand ihres Hostnamens und ihrer ID (in diesem Fall der IP-Adresse) identifizierte. Um Spoofing zu vermeiden, war die *PostgreSQL*-Datenbank noch mit einer zusätzlichen Benutzername/Passwort-Authentifizierung versehen.

Nachdem all diese Schritte automatisch ausgeführt wurden, war der neue Sensor sofort einsatzbereit, verband sich mit dem IRC-Server und sammelte ab diesem Moment Daten. Um bei den einzelnen Schritten nicht manuell Passwörter eingeben zu müssen, wurde auf allen Knoten auf *Public-Key-Authentifizierung* gesetzt.

Nach der Fertigstellung dieser Skripte wurden 50 freie Nodes zum PlanetLab-Slice dieses Experiments hinzugefügt. Es wurden nicht, wie ursprünglich geplant, 100 Sensoren verwendet, da beim Aufsetzen der ersten Instanzen bereits verschiedenste Probleme auftraten und der zusätzliche Aufwand der Verwaltung von dieser Menge an Knoten in keinem Verhältnis zum Zeitaufwand und dem Umfang dieses Experiments stand. Insgesamt wurden über 60 Sensoren mit der Installationsroutine aufgesetzt, von denen aber einige wieder abgeschalten werden mussten, da verschiedene Probleme auftraten. So gab es einige Hosts, die ein- und ausgehende Verbindungen massiv limitierten und für eine solche Architektur ungeeignet waren. Auf wieder anderen Slivers war UML aus unbekannten Gründen überhaupt nicht lauffähig (`Segmentation Fault`) oder unbenutzbar langsam. Bei der Auswahl der Sensoren wurde insbesondere drauf geachtet, in möglichst vielen verschiedenen Ländern Sensoren zu platzieren. Das Experiment startete schließlich mit den in Tabelle 5 aufgelisteten Knoten.

4.3.2 Betrieb

Nachdem alle Sensoren aufgesetzt und online waren wurde das Experiment offiziell gestartet und für sieben aufeinanderfolgende Tage Daten gesammelt. Der Betrieb verlief im Wesentlichen reibungslos. Mit Hilfe des IRC-Kontrollkanals wurde die Festplattenbelegung der Sensoren mehrmals überprüft und

Hostname	Standort	Hostname	Standort
planetlab02.just.edu.jo	Jordanien	csplanetlab4.kaist.ac.kr	Südkorea
planetlab3.inf.ethz.ch	Schweiz	planetlab6.csee.usf.edu	Florida, USA
planetlab1.informatik.uni-goettingen.de	Deutschland	planetlab2.mnlab.cti.depaul.edu	Chicago, USA
planetlab1.tmit.bme.hu	Ungarn	planetlab4.cnds.jhu.edu	Maryland, USA
plab2.ple.silweb.pl	Polen	planetlab2.rutgers.edu	New Jersey, USA
plab2.cs.ust.hk	Hong Kong, China	planetlab2.diku.dk	Dänemark
planetlab1.cesnet.cz	Tschechoslowakei	planetlab1.tau.ac.il	Israel
aguila1.lsi.upc.edu	Spanien	planetlab2.buaa.edu.cn	China
orval.infonet.fundp.ac.be	Belgien	pl2.cis.uab.edu	Alabama, USA
onelab6.iet.unipi.it	Italien	planetlab1.cnis.nyit.edu	New York, USA
planetlab-1.ida.liu.se	Schweden	planet1.dsp.ac.cn	China
planetlab1.nrl.eecs.qmul.ac.uk	UK	pl1.eng.monash.edu.au	Australia
roam2.cs.ou.edu	Oklahoma, USA	planetlab1.temple.edu	Pennsylvania, USA
planetlab-node-01.ucd.ie	Irland	planetlab03.cs.washington.edu	Washington, USA
plab1.engr.sjsu.edu	USA	pl2.bell-labs.fr	Frankreich
plab2.create-net.org	Italien	planetlab2.iis.sinica.edu.tw	Taiwan
ple2.dmcs.p.lodz.pl	Polen	plgmu2.ite.gmu.edu	Virginia, USA
ple2.ait.ac.th	Thailand	pl-node-1.csl.sri.com	Kanada
planetlab1.fct.ualg.pt	Portugal	planetlab1.virtues.fi	Finnland
planetlab2.fct.ualg.pt	Portugal	planetlab-1.cs.ucy.ac.cy	Zypern
planetlab1.upm.ro	Rumänien	planetlab6.cs.duke.edu	England
planet2.cc.gt.atl.ga.us	USA	cs-planetlab4.cs.surrey.sfu.ca	Kanada
planetlab-tea.ait.ie	Irland	planetlab05.mpi-sws.mpg.de	Deutschland
utet.ii.uam.es	Spanien	planet1.ku.edu.tr	Türkei
planetlab1.ifi.uio.no	Norwegen	planetlab2.pop-pa.rnp.br	Brasilien

Tabelle 5: Die im Experiment verwendeten Sensoren

von einigen wenigen Instanzen, die sehr viele Logdaten generiert hatten und in Platzmangel gerieten, wurden im Vorfeld die Daten manuell eingesammelt.

Da im IRC-Kontrollkanal konstant ca. 50 Bots anwesend waren, die periodisch auf die vom „Boss“ gesendeten Ping-Requests antworteten (eine Eigenheit des *irckit*), wurde die *Freenode*-Administration auf diese Situation aufmerksam. Zwei Administratoren des Netzwerks betraten den Kanal und erkundigten sich, was der Zweck dieses offensichtlichen Botnets sei und forderten dazu auf, die Systeme abzuschalten. Nach einer ausführlichen Erklärung des Experiments und des Forschungsgegenstandes waren sie jedoch mit dem Weiterbetrieb für die Dauer des Experiments einverstanden.

Bei einer manuellen Routineüberprüfung der Sensoren wurde festgestellt, dass einige nicht mehr auf Requests reagierten, obwohl aus Sicht des PlanetLab-Slivers kein Problem vorlag. Es stellte sich nach Konsultation der Protokolle des virtuellen Systeme heraus, dass diese zu viele geöffnete Verbindungen aufrechtzuerhalten versuchten. Es könnte sich hierbei um *SYN-Flood-Angriffe* gehandelt haben, dies war jedoch anhand der gesammelten Daten nicht eindeutig verifizierbar. Leider unterstützte der UML-Kernel eine unter Linux-Systemen häufig verwendete Schutzmaßnahme vor derartigen Angriffen, *TCP SYN-Cookies*, nicht. Es half in dieser Situation schließlich nur noch ein Zurücksetzen der UML-Instanz. Um dies in Zukunft automatisch zu gewährleisten, wurde auf dem *Logging Server* schließlich ein weiteres Skript implementiert, das periodisch die Liste aller bekannten Sensoren via SSH abfragte und diejenigen, die nicht mehr reagierten, über den auf dem PlanetLab-Sliver laufenden SSH-Dienst (auf Port 2223) reaktivierte.

Nach Ablauf des Testzeitraums wurden alle von den Sensoren gesammelten Daten auf einem zentralen Host zur Auswertung gesammelt. Das hierfür zuständige Skript `collectdata.sh` erhielt als Parameter eine Datei, in der die zu besuchenden Hosts aufgelistet waren. Es besuchte der Reihe nach jeden Knoten über den auf dem TCP-Port 22222 laufenden SSH-Server, führte das an früherer Stelle bereits beschriebene Skript `collectsensordata`, das in jeder UML-Instanz vorhanden war, aus und kopierte anschließend die gewonnenen Daten in eine Verzeichnisstruktur
`logs/<hostname>/amun|kippo|dionaea`. Das Abschalten des Sensornetzwerks erfolgte abschließend über die IRC-Kommandoschnittstelle.

5 Auswertung der gesammelten Daten

In diesem Kapitel werden die im Laufe des Experiments gesammelten Daten präsentiert und ausgewertet. Die Daten wurden sowohl über das *SurfCERT IDS* Webinterface, als auch aus den gesammelten Logdateien aller Sensoren gewonnen. Während die Webanwendung umfassende Filtermöglichkeiten für den Datenbestand anbot, wurden die Logfiles mit Linux-Shellkommandos wie `grep`, `sort`, `sed` und `awk` ausgewertet. Da eine individuelle Auswertung eines jeden Sensors an dieser Stelle zu umfangreich wäre, wurden die gesammelten Daten aller Knoten der Übersichtlichkeit halber aggregiert und in den meisten der nachfolgenden Diagramme und Tabellen aufsummiert.

5.1 Aggregierte Verbindungsdaten

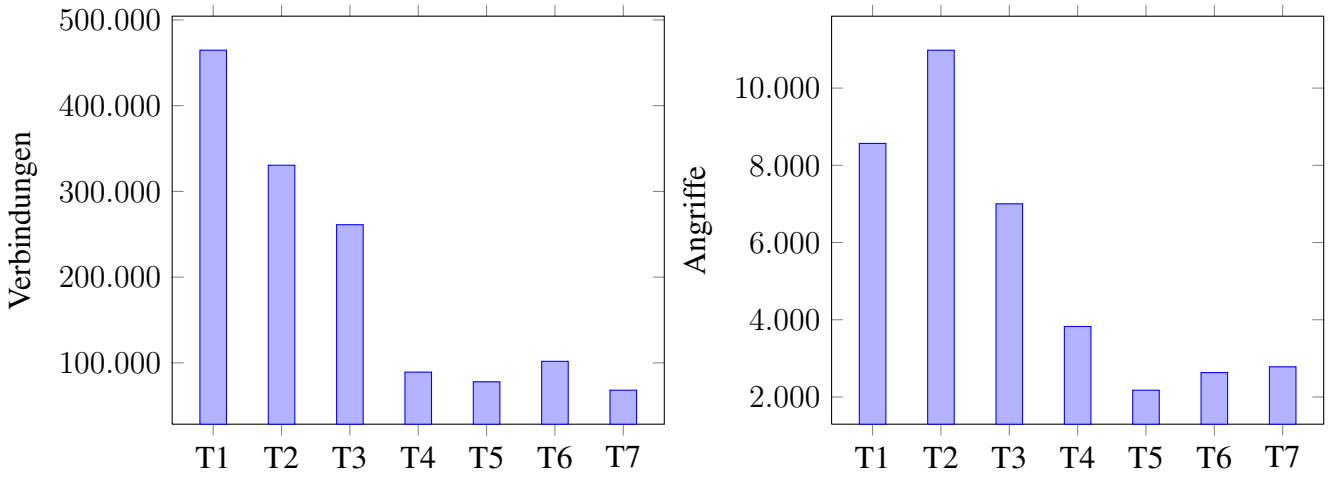

Abbildung 4: Verbindungen und Angriffe im Verlauf des Experiments

Sensoren, die Verbindungsprotokolle an das *SurfCERT IDS* sendeten, ordneten diese entweder als **Possible Malicious Attack** oder als **Malicious Attack** ein. In die erste Kategorie fielen jegliche Verbindungen, die nicht eindeutig als bösartig eingestuft werden konnten, darunter auch Portscans oder nicht identifizierbare Angriffe. Diese Kategorisierung erfolgte innerhalb der Plugins, die die Verbindungsdaten zum *Logging Server* sendeten. Die Einordnung einiger Verbindungen hat sich in diesem Zusammenhang als fragwürdig herausgestellt, wie im Anschluss noch gezeigt werden wird.

Im sieben Tage umfassenden Testzeitraum wurden insgesamt **1.543.527 TCP- und UDP-Verbindungen** zu den 50 weltweit verteilten Sensoren gezählt. **40.369** dieser Verbindungen, etwa $2,6\%$, wurden als **Malicious Attack** eingestuft. Abbildung 4 zeigt die Verteilung aller Verbindungen und aller Angriffd (Malicious Attacks) über den Testzeitraum. *T1* bis *T7* bezeichnen dabei die jeweiligen Tage des Experiments. Es fällt auf, dass $75,6\%$ aller zu den Honeypots aufgebauten Verbindungen in den ersten drei Tagen des Experiments stattfanden, mit den Angriffen verhielt es sich analog. Zudem ist im Allgemeinen ein Abwärtstrend, sowohl hinsichtlich aller gezählten Verbindungen, als auch der als Angriff klassifizierten Pakete im Verlauf der Woche zu erkennen. Diese Beobachtung lässt sich zu einem kleinen Teil damit erklären, dass im Verlauf des Experiments einige Sensoren, bzw. PlanetLab-Slivers inaktiv wurden und

keine Daten mehr sammelten. So wurden zwei der Knoten vollständig zurückgesetzt, da der Speicherverbrauch der UML-Instanz für die verwaltende Institution des Server zu hoch war. Andere Slices wiederum wurden einfach abgeschalten und waren über Tage nicht mehr erreichbar, während wieder andere Slivers noch auf TCP-Port 22 (SSH) antworteten, allerdings aus unerfindlichen Gründen keinen Zugang mehr gewährten. Es ist möglich, dass die Sensoren im Laufe des Testzeiraumes von Angreifern als Honeypots identifiziert und von weiteren Angriffen ausgeschlossen wurden, was den deutlichen Rückgang der Verbindungen und Angriffe erklären könnte. Ein Beweis für diesen Umstand, wie beispielsweise ein „offizielles“ Blacklisting der IP-Adresse einiger Knoten, wurde jedoch nicht gefunden.

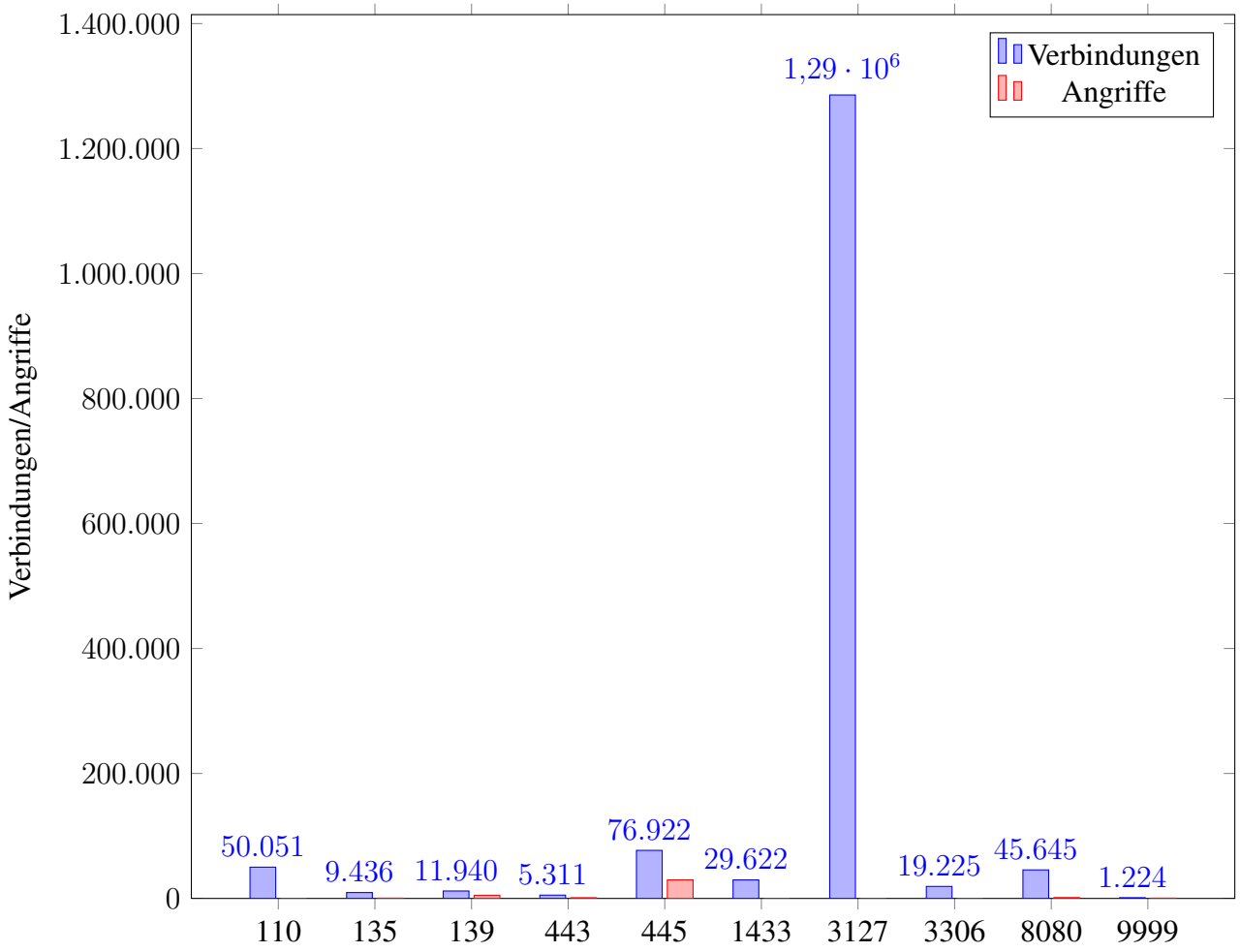

Abbildung 5: Die TCP-Ports mit den meisten aufgezeichneten Verbindungen

Zunächst soll ein Blick auf den Datenverkehr sortiert nach angebotenem Service, bzw. geöffnetem Port geworfen werden, ohne dabei eine Klassifizierung vorzunehmen. Abbildung 5 zeigt die zehn Ports, die den meisten Traffic verursacht haben. Nicht alle Honeypot-Module konnten den ankommenden Datenverkehr klassifizieren, weshalb nur für einen Teil der Daten eine Unterscheidung in Angriff/Sonstiges möglich war. Es sticht heraus, dass 83% aller aufgezeichneten Verbindungen zum Port 3127 der Honeypots aufgebaut wurden. Auf diesem Port lief das zum Honeypot `amun` gehörige Verwundbarkeitsmodul `vuln-mydoom`. Die Recherche hat ergeben, dass **W32.Mydoom** ein Computerwurm ist, der insbesondere als Anhang von Spam-E-Mails vorkommt [19]. Bei einer Infektion eines Windows-Systems installiert die Malware eine Hintertür, indem sie die TCP-Ports 3127 bis 3198 öffnet. Port 3127 kann

hierbei von Angreifern als Proxy genutzt werden. Die Schadsoftware führte zusätzlich in einem begrenzten Zeitraum im Jahr 2004 eine *Denial of Service*-Angriffe aus. Anschließend verblieb nur noch die Proxy-Komponente auf dem System, auf die `vuln-mydoom` abzielt. Der `amun`-Honeypot klassifizierte Versuche, diesen Port als Proxy zu benutzen, nicht als Angriff, sondern nur als *Possible Malicious Attack*. Ob es sich dabei um eine Fehlklassifikation oder ein bewusstes Vorgehen handelt, da in diesem speziellen Fall keine Sicherheitslücke in einem normalerweise auf diesem Port angebotenen Dienst ausgenutzt wird, ist jedoch unklar. Die Tatsache, dass eine solche Menge an Proxy-Anfragen an diesen Port geschickt wurden, obwohl es sich nicht um einen typischen von Proxy-Servern verwendeten Port handelt, lässt den Schluss zu, dass die Schadsoftware trotz ihres Alters noch immer weit verbreitet ist.

Die ebenfalls noch häufig auftretenden TCP-Ports 1433 und 3306 wurden vom Honeypot `dionaea` verwaltet und simulierten einen *MSSQL*-, sowie einen *MySQL*-Server. Leider konnte der Honeypot zum Zeitpunkt des Experiments keine Klassifikation der an diesen Ports auftretenden Daten vornehmen. Diese müsste im Anschluss manuell für jeden einzelnen aufgezeichneten Datenstrom durchgeführt werden, was im Rahmen dieses Projekts jedoch zu aufwändig gewesen wäre. Auf der Emulation des SMB/CIFS-Dienstes (Windows-Dateifreigabe) auf Port 445 lag der Fokus von `dionaea`. Der Honeypot konnte bei diesen Verbindungen eine Klassifikation vornehmen: 29.659 Angreifer ($27,8\%$ der Verbindungen) versuchten, durch dem Honeypot bekannte Exploits Sicherheitslücken auf dem vermeintlichen Windows-System auszunutzen. Details über die von diesen Angreifern gesammelte Malware werden im Anschluss vorgestellt.

Die Diagramme in den Abbildungen 6 und 7 schlüsseln die Verbindungen zu den zehn häufigsten Ports noch einmal genauer über den Zeitraum des Versuches auf.

Es fällt auf, dass die Proxy-Anfragen an Port 3127 (mydoom) insbesondere in den ersten 4 Tagen des Experiments stattfanden und dann zurückgingen. Angesichts des kurzen Aufzeichnungszeitraumes dieser Daten kann nicht sicher gesagt werden, ob es sich hier nur um eine normale Schwankung handelte, oder ob die Sensoren (die die Proxy-Anfragen nicht beantworteten) als Täuschung identifiziert wurden. Die an Port 110 (POP3-Service) gerichteten Anfragen erreichten am Tag 4 ihr lokales Maximum. Es ist dabei anzumerken, dass dieser Ausschlag im Wesentlichen von einem einzelnen Angreifer mit einer schwedischen IP-Adresse verursacht wurde, der etwa 45.000 Verbindungen über den ganzen Tag hinweg zu einem einzigen Sensor, `planetlab02.just.edu.jo`, aufbaute.

Zudem soll noch angemerkt werden, dass die Anfragen an die simulierten Datenbankserver auf den Ports 1433 und 3306 während des Testzeitraums nicht dem auf anderen Ports verzeichneten Rückgang an Verbindungen ausgesetzt waren. Da beide Server auch mit normalen SQL-Clients genutzt werden konnten und eine SQLite-Datenbank als Basis verwendeten, um Anfragen auch glaubhaft zu beantworten, könnten diese Dienste realistisch genug ausgesehen haben, um bei Angreifern keinen Verdacht auszulösen. Die Emulation dieser Dienste durch `dionaea` unterschied sich stark von den Verwundbarkeitsmodulen des `amun`-Honeypots, die ausschließlich spezielle Sicherheitslücken vortäuschten und darüber hinaus keinerlei Interaktion im Rahmen der entsprechenden Protokolle erlaubten.

Abbildung 8 zeigt, wie oft die simulierten Sicherheitslücken der Honeypots `amun` und `dionaea` ausgenutzt wurden. Die Namen der „Verwundbarkeiten“ oder auch Sicherheitslücken leitet sich aus dem Namen der verwendeten Module im Honeypot her, eine eindeutige Zuordnung zu einer bekannten Sicherheitslücke war anhand dieses Namens jedoch nicht immer möglich. Es fällt auf, dass 29.442, also $73,2\%$ aller „erfolgreichen“ Exploits, auf die von Microsot als MS08-67 bezeichnete Sicherheitslücke zurückzuführen waren. Dabei handelt es sich um einen 2008 gefundenen Fehler im *Windows Server Service*,

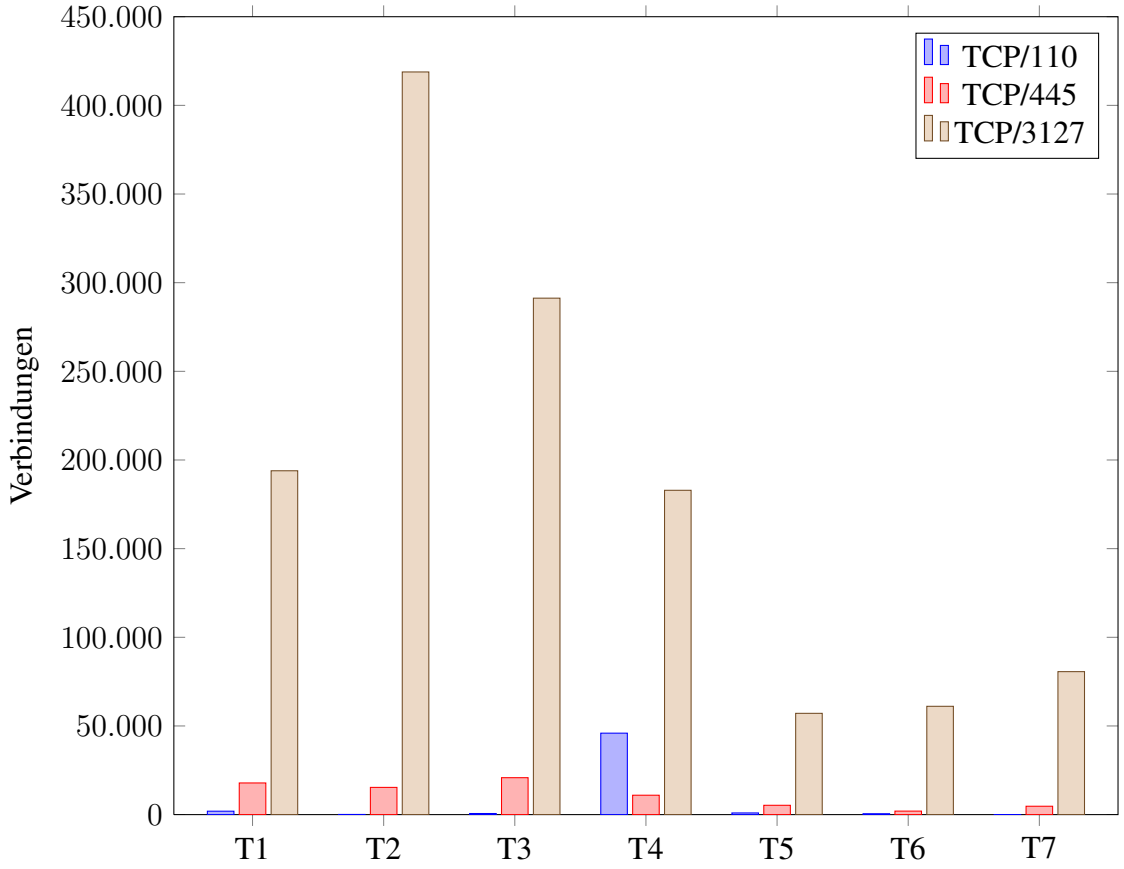

Abbildung 6: Gesammelte Verbindungsdaten im Testzeitraum

der bei speziellen RPC-Anfragen auftrat und das Einschleusen von Schadcode ermöglichte. Trotz dass für diese Lücke noch im selben Jahr ein Update veröffentlicht wurde, sind anfällige Systeme noch immer verbreitet, wie in einem Artikel von *SecureState* im Herbst 2012 festgestellt wurde [18] und was sich im Experiment auch bestätigt hat. Die mit auf diese Sicherheitslücke abzielenden Exploits mitgelieferten Payloads haben im Laufe des Experiments in der Regel Malware von externen Quellen heruntergeladen, die im Kapitel 5.2 näher betrachtet wird. Da `amun` nur wenig Dokumentation anbietet und der Quellcode der Plugins kaum kommentiert ist, konnte nicht herausgefunden werden, was für eine Schwachstelle das Modul `vuln-netbiosname` emuliert. Sie wurde auf dem TCP-Port 139 simuliert, dem *NETBIOS Session Service*, bei den mitgelieferten Payloads handelte es sich ebenfalls um Malware-Downloadverfahren.

5.2 Malware

Dieser Abschnitt gibt einen Überblick über gesammelte Malware, die nach einem erfolgreich durchgeführten Exploit gegen einen der Sensoren heruntergeladen und lokal gespeichert wurde. Es ist abhängig von der jeweiligen Payload, welches Verfahren beim Herunterladen zum Einsatz kam. Tabelle 6 gibt einen Überblick über die zu diesem Zweck genutzten Protokolle. Das Webinterface des *Logging Servers* hat die Proxy-Anfragen an den zuvor bereits beschriebenen mydoom-Port 3127 in diese Statistik mit aufgenommen, obwohl über diesen keine Malware-Binaries bezogen wurden. Das Ausmaß dieser

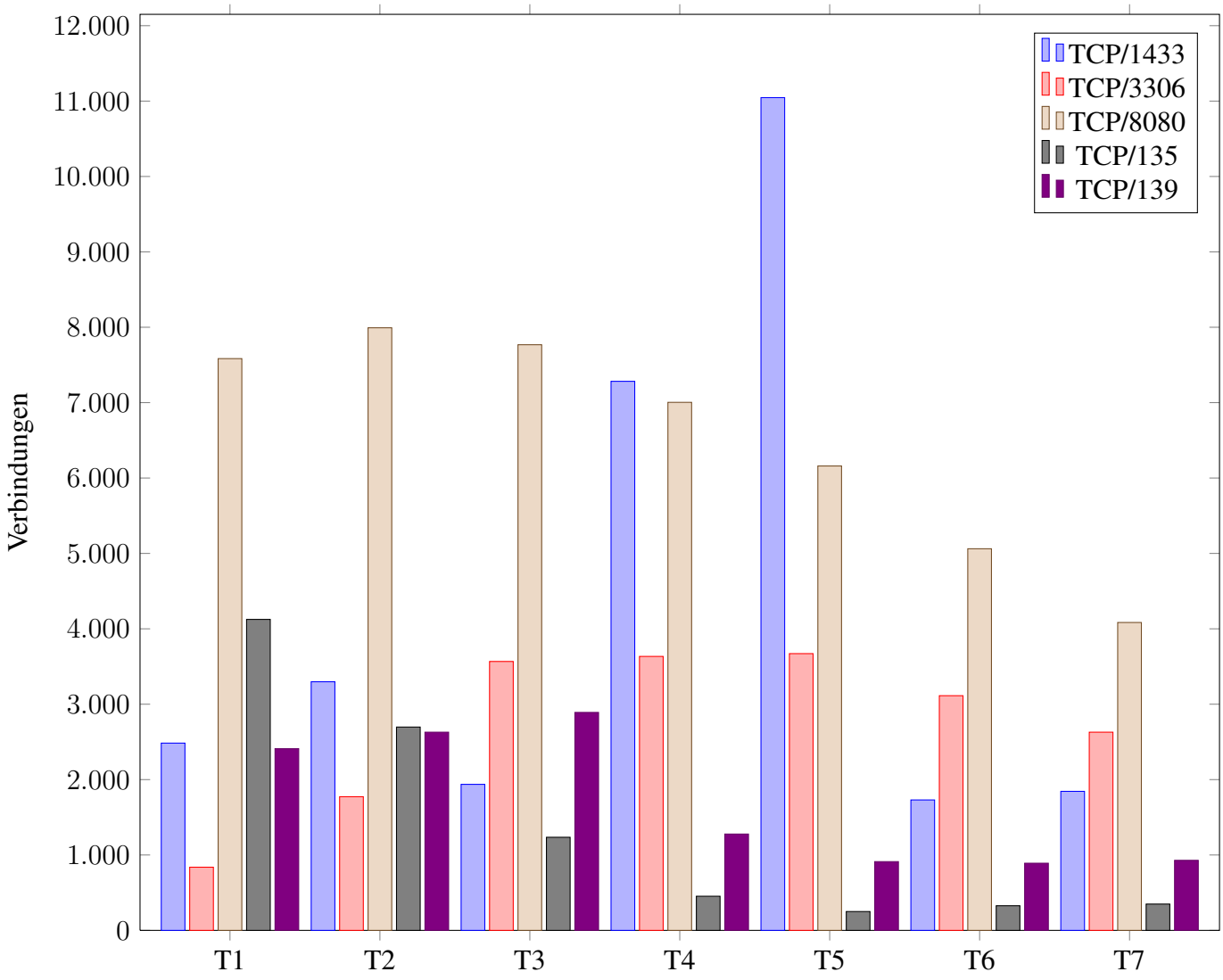

Abbildung 7: Gesammelte Verbindungsdaten im Testzeitraum

Proxy-Anfragen wird auch in dieser Statistik mit 73% aller „Downloads“ wieder gut deutlich. Malware-Binaries wurden schließlich primär über das `HTTP`-Protokoll auf TCP-Port 80 bezogen (22%), die für diesen Zweck ebenfalls häufig verwendeten Protokolle `FTP` und `SMB` [33] wurden hingegen insgesamt nur 490, bzw. 58 mal gezählt. In die Kategorie „Sonstige“ fallen noch seltener aufgetretene Protokolle wie `TFTP` oder solche, die nicht über einen der standardmäßig verwendeten Ports liefen (beispielsweise `HTTP` über TCP-Port 30123).

Tabelle 7 gibt schließlich einen Überblick über die zehn häufigsten heruntergeladenen Malware-Exemplare. Diese sind anhand ihres MD5-Hashes leicht miteinander vergleichbar und es war festzustellen, dass Angreifer überwiegend Variationen bekannter Malware verwendeten, die sich eindeutig einer Familie von Schadsoftware zuordnen ließ. Wie die Tabelle zeigt, waren die zehn am häufigsten angetroffenen, einzigartigen Malware-Exemplare alle Variationen des sog. *Conficker*-Wurms, der 2008 erstmalig entdeckt wurde und verschiedenste Verbreitungswege wie Exploits, USB-Wechselmedien oder Dateifreigaben im Netzwerk zur Weiterverbreitung nutzt. Obwohl die Schadsoftware von aktueller Antiviren-Software pro-

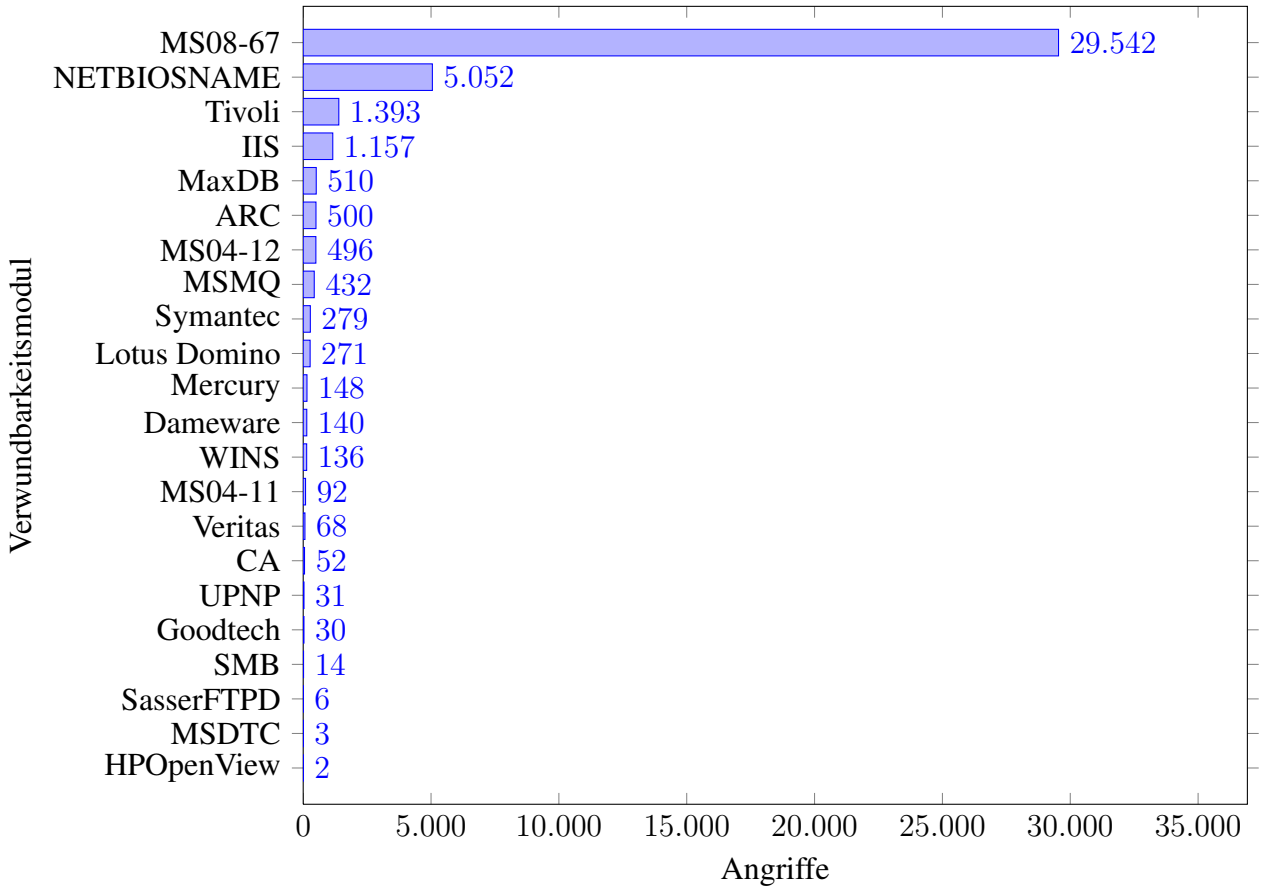

Abbildung 8: Exploits

blemlos erkannt wird, wie anhand der *VirtusTotal-Score*[68] deutlich wird, wird sie von den automatisierten Angreifern noch immer verbreitet.

Malware wurde nicht immer direkt von der angreifenden IP-Adresse selbst angeboten. Die Tabelle 8 listet die zehn Hosts mit zugehörigen Standort auf, von denen die meisten Malware-Exemplare heruntergeladen wurden. Was es mit diesen Hosts auf sich hat und warum ausgerechnet diese so häufig als Quelle dienten, ist nicht bekannt. Eine Suche nach diesen IP-Adressen mit Hilfe bekannter Suchmaschinen hat keine zusätzlichen Informationen über diese Hosts geliefert, abgesehen vom Standort.

5.3 Individuelle Sensoren

Zuletzt soll noch ein Blick auf die individuellen Sensoren geworfen werden. Abbildung 9 zeigt die zehn Sensoren, die am häufigsten von Angriffen (nach der Klassifizierung des *SurfCERT IDS*) betroffen waren. Etwa $31,7$ aller aufgezeichneten Exploits geschahen auf dem in Virginia gelegenen PlanetLab-Sliver `plgmu2.ite.gmu.edu`. Weitere $24,7\%$ waren auf dem in Deutschland gelegenen Knoten `planetlab05.mpi-sws.mpg.de` zu verzeichnen und noch knapp 15% der Exploits betrafen den in der Tschechoslowakei gelegenen Host `planetlab1.cesnet.cz`. Insgesamt spielten sich über 85%

[68] Anteil der getesteten Antiviren-Lösungen, die die Malware laut `http://virustotal.com` identifizieren konnten

Protokoll	Downloads	Anteil
mydoom (TCP/ 3127)	175654	74%
HTTP (TCP/80)	52276	22%
Sonstige	8113	3%
FTP (TCP/21)	490	0%
SMB (TCP/445)	58	0%

Tabelle 6: Genutzte Protokolle/Ports von Payloads

MD5-Hash	Malware-Kategorie	VirusTotal-Score	Vorkommen
393e2e61ff08a8f7439e3d2cfcb8056f	Worm/Conficker	45/47	4814
2e8da5a55865a091864a4338ef4d2e44	Worm/Conficker	45/47	3075
8c9367b7dc43dadaa3ec9da767c586cf	Worm/Conficker	46/47	1987
78c9042bbcefd65beaa0d40386da9f89	Worm/Conficker	46/47	1242
95ad430abca3da496600f764c120683c	Worm/Conficker	45/47	829
c3852074ee50da92c2857d24471747d9	Worm/Conficker	45/47	701
0c059b0d1d5a03f69a21185987c17d5c	Worm/Conficker	45/47	628
4385e19f4dc5ebbcf97c438cc780600c	Worm/Conficker	42/47	411
690ad58f214d3df1fb93fcb9c2a516c1	Worm/Conficker	44/47	360
94e689d7d6bc7c769d09a59066727497	Worm/Conficker	46/47	183

Tabelle 7: Die zehn häufigsten Malware-Exemplare im Experiment

aller Exploits auf nur fünf von insgesamt 50 Sensoren ab. Ein Zusammenhang zwischen dem geographischen Standort des Sensors und der Häufigkeit der Angriffe konnte allerdings nicht hergestellt werden, so verzeichnete beispielsweise der ebenfalls in Deutschland gelegene Sensor `pl1.informatik.uni-goettingen.de` im gesamten Testzeitraum lediglich 6 erfolgreiche Angriffe.

Die Tabelle 9 listet erneut alle Sensoren, die Teil des Experiments waren und Daten gesammelt haben sortiert nach der Anzahl der von ihnen aufgezeichneten Verbindungen auf. Als „Verbindung“ zählt in diesem Zusammenhang jeder Datenstrom, der nicht als „Angriff“ (*Malicious Attack*) klassifiziert wurde, da diese in einer eigenen Spalte aufgeführt sind. Die Spalte „Up“ kennzeichnet die *Uptime* des jeweiligen Sensors, also der Zeitraum in Tagen, in dem erfolgreich Daten gesammelt wurden. Mit Ausnahme des Knotens `planetlab2.iis.sinica.ed.tw`, der in nur zwei Tagen mehr als 98.000 Verbindungsdaten gesammelt hat, wurden alle Knoten, von denen keine oder nur Daten von drei oder weniger Tagen vorhanden war, aus der Auswertung ausgeschlossen. Beim Betrachten der Sensordaten fällt auf, dass es keinen Zusammenhang zwischen der Summe der als Angriff klassifizierten und den als **Possible Malicious Attack** eingeschätzen Verbindungen gibt. Der Host, der im Testzeitraum die meisten eingehenden Verbindungen verzeichnete, `planetlab2.diku.dk`, hat lediglich 312 Angriffe (oder erfolgreiche Exploits) aufgezeichnet. Im Kontrast dazu verzeichnete der Host `plgmu2.ite.gmu.edu` die drittmeisten Verbindungen und darunter über 12.700 Angriffe. Beim Betrachten dieser Statistik muss allerdings erneut berücksichtigt werden, dass die Proxy-Requests an den mydoom-Port 3127 nicht als Angriff klassifiziert wurden. Zudem wurden im Experiment ausschließlich Low-Interaction-Honeypots eingesetzt, die lediglich ihnen bekannte Angriffe auch als solche erkennen konnten. Wie viele der in der Tabelle nur als „Verbindungen“ gekennzeichneten Datenströme schließlich tatsächlich Angriffe waren, ist nicht ohne erheblichen Aufwand, das heißt Einzelanalyse der individuellen Verbindungen, festzustel-

Host	Ursprungsland	Vorkommen
176.58.18.75	Polen	4833
82.211.142.25	Georgien	2336
46.98.140.165	Ukraine	1780
193.231.43.9	Rumänien	1738
46.98.67.168	Ukraine	1645
tw.myblog.yahoo.com	Taiwan	1127
201.188.45.60	Chile	801
201.189.128.164	Chile	777
95.26.93.222	Russland	744
201.189.92.66	Chile	635

Tabelle 8: Die zehn häufigsten Bezugsquellen beim Malware-Download

len.

Während des Experiments waren zu keinem Zeitpunkt alle 50 Sensoren gleichzeitig aktiv. Wie die Spalte „Up“ in der vorherigen Tabelle bereits angedeutet hat, wurden die PlanetLab-Slices und damit auch die auf ihnen laufenden virtuellen Slivers teilweise von den zugehörigen Institutionen heruntergefahren und erst später während des Testzeitraums wieder reaktiviert. Andere Knoten, wie beispielsweise `pl2.bell-labs.fr`, sammelten für zwei Tage Daten und waren anschließend nie wieder erreichbar. Die Administrationsoberfläche des PlanetLab listet keine Details zu den einzelnen Hosts auf, weswegen es nicht möglich war, die Ursache für ausgefallene Sensoren herauszufinden. Weitere aufgetretene Probleme waren netzwerkbedingt, so gab es Sensoren, die zwar aktiv waren, allerdings den *Logging Server* aus unerfindlichen Gründen nicht erreichen konnten und somit keine brauchbaren Daten lieferten. Wieder andere Knoten verweigerten den Zugang zum Sliver, nachdem sie einmal neu gestartet wurden. Die Gründe für diese Beobachtungen mögen vielfältig sein und wurden im Rahmen des Experiments aus Zeitgründen nicht weiter untersucht. Die Anzahl der Sensoren wurde groß genug gewählt, um auch bei ausfallenden Slivers genügend verwertbare Daten von den verbleibenden Honeypots sammeln zu können.

Ein separates Skript auf dem *Logging Server* hat alle 30 Minuten die zu jedem Zeitpunkt aktiven Sensoren gezählt und diese Information aufgezeichnet. Abbildung 10 zeigt die Entwicklung der Durchschnittswerte für die einzelnen Messtage. Es muss an dieser Stelle darauf hingewiesen werden, dass der letzte Messtag ein Sonderfall darstellte, an dem aus unerfindlichen Gründen sogar mehr als zehn Sensoren dauerhaft fehlten. Die meisten dieser Knoten waren am Folgetag, der allerdings nicht mehr im Testzeitraum lag, wieder aktiv.

Zusammengefasst hat die Auswertung der gesammelten Daten eine unerwartet hohe Aktivität auf dem Port 3127 vieler Knoten ergeben, die sich auf Proxy-Requests an vermeintlich mit der Malware W32.Mydoom infizierte Systeme zurückführen lässt. Die vom Honeypot *dionaea* sehr umfassend emulierten Protokolle SMB/CIFS, *MySQL* und *MSSQL* verzeichneten ebenfalls viel Datenverkehr, was darauf hindeutet, dass Low-Interaction-Honeypots, die den Fokus auf die Qualität der Emulation legen, sich für den praktischen Einsatz sehr gut eignen. Der Honeypot `amun` hat insbesondere mit dem Verwundbarkeitsmodul `vuln-netbiosname` viele Angreifer angezogen, war jedoch beim Interpretieren der zugehörigen Payload und beim Beziehen einer eventuellen Malware nicht immer erfolgreich. Überraschend war außerdem, dass der `kippo`-Honeypot auf Port 2222 (SSH) über alle Sensoren hinweg im Testzeitraum nicht eine einzige Verbindung verzeichnet hat. Dass eine solche aufgezeichnet worden wä-

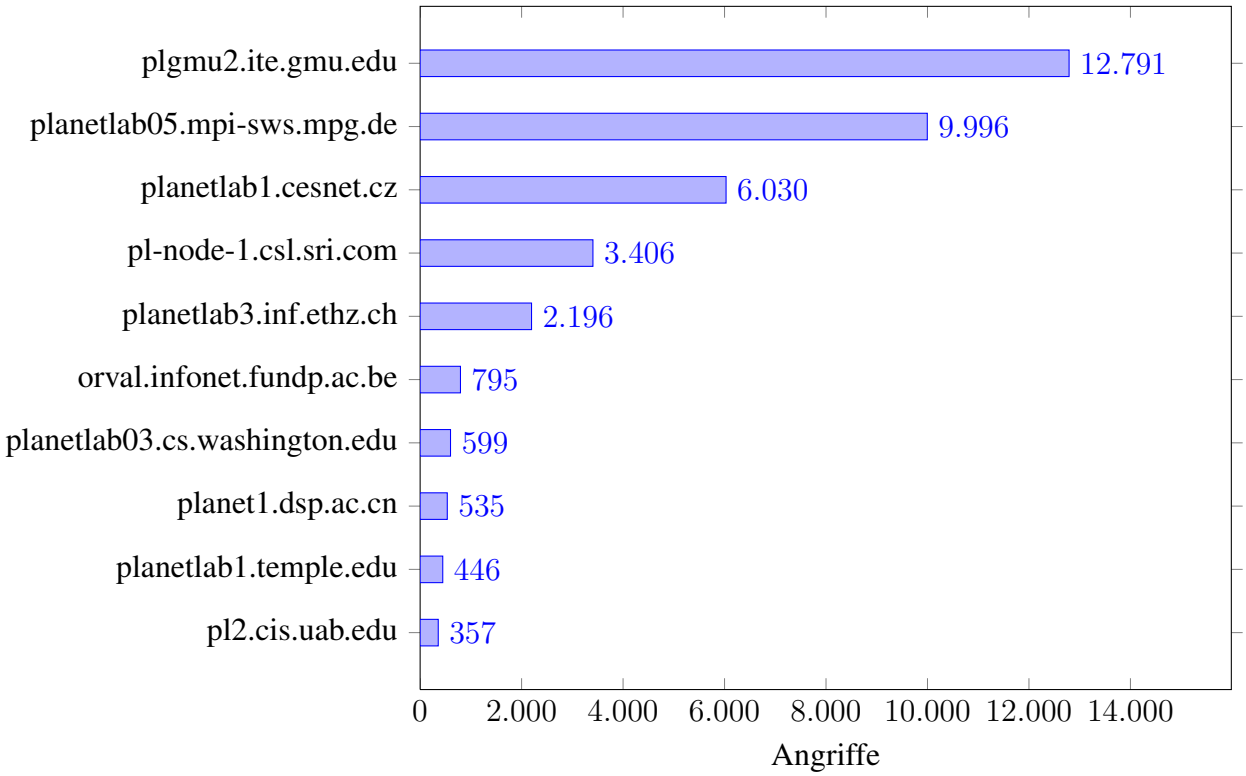

Abbildung 9: Die zehn am häufigsten von Angriffen betroffenen Sensoren

re, wurde vor dem Experiment verifiziert, weswegen Konfigurationsfehler auszuschließen sind. Viel mehr lässt diese Tatsache und die bereits präsentierten Fakten den Verdacht zu, dass die im Rahmen des Versuchs aufgezeichneten Angriffe im Wesentlichen automatisierter Natur waren und von Bots durchgeführt wurden. Ein Angreifer, der „manuell“ vorgegangen wäre, hätte als eine der ersten Maßnahmen einen Portscan unternommen und dabei den SSH-Server auf TCP-Port 2222 schnell entdeckt.

Hostname	Verb.	Angriffe	Up/d	Hostname	Verb.	Angriffe	Up/d
planetlab2.diku.dk	648334	312	7	planet1.ku.edu.tr	1955	299	7
planetlab1.fct.ualg.pt	159155	119	7	planetlab-tea.ait.ie	1797	15	7
plgmu2.ite.gmu.edu	123320	12791	7	pl1.informatik.uni-goettingen.de	1691	6	7
planetlab2.iis.sinica.ed.tw	98084	113	2	aguila1.lsi.upc.edu	1680	62	7
planetlab-1.cs.ucy.ac.cy	88811	302	7	planetlab1.cnis.nyit.edu	1665	1	7
planetlab05.mpi-sws.mpg.de	76740	9996	5	csplanetlab4.kaist.ac.kr	1587	5	7
planetlab3.inf.ethz.ch	63911	2196	7	plab1.engr.sjsu.edu	1537	68	7
planetlab2.fct.ualg.pt	45600	116	7	plab2.ple.silweb.pl	1506	3	7
planetlab1.cesnet.cz	18556	6030	7	pl2.cis.uab.edu	1440	357	4
planetlab03.cs.washington.edu	24287	599	7	planetlab1.virtues.fi	1463	6	7
planetlab02.just.edu.jo	24229	223	7	planetlab1.temple.edu	1223	446	6
planetlab1.ifi.uio.no	24220	14	7	planetlab6.cs.duke.edu	1206	65	7
pl-node-1.csl.sri.com	16454	3406	7	planetlab1.nrl.eecs.qmul.ac.ul	1161	0	7
planetlab1.tau.ac.il	8736	23	7	planetlab6.csee.usf.edu	1109	4	7
planetlab2.pop-pa.rnp.br	8023	250	6	pl1.eng.monash.edu.au	811	18	6
planet2.cc.gt.atl.ga.us	6385	7	7	plab2.cs.ust.hk	800	35	7
planetlab2.mnlab.cti.depaul.edu	4253	66	7	planetlab-1.ida.liu.se	718	18	7
planetlab1.upm.ro	4111	42	7	planetlab2.buaa.edu.cn	597	0	5
planetlab2.rutgers.edu	3844	188	7	utet.ii.uam.es	336	2	7
cs-planetlab4.cs.surrey.sfu.ca	3354	199	7	onelab6.iet.unipi.it	-	-	-
planet1.dsp.ac.cn	3221	535	7	roam2.cs.ou.edu	-	-	-
ple2.dmcs.p.lodz.pl	2799	52	7	planetlab-node-01.ucd.ie	-	-	-
ple2.ait.ac.th	2506	137	4	planetlab4.cnds.jhu.edu	-	-	-
orval.infonet.fundp.ac.be	2197	795	7	pl2.bell-labs.fr	-	-	-
plab2.create-net.org	2030	314	7	planetlab1.tmit.bme.hu	-	-	-

Tabelle 9: Alle Sensoren mit ihren aufgezeichneten Verbindungen, Angriffen und Uptimes (Up)

6 Zusammenfassung

Im Rahmen dieser Arbeit wurden aktuelle Honeypot-Technologien untersucht und einige davon im praktischen Einsatz erprobt. Kapitel 2 gab eine Einführung in den Sachverhalt und stellte die häufig anzutreffende Klassifizierung in Low- und High-Interaction-Honeypots, sowie in Client- und Server-Side-Honeypots vor. Im Anschluss wurden Möglichkeiten präsentiert, mit denen ein solches System in ein bestehendes Netzwerk integriert und abgesichert werden kann. Es wurde zudem gezeigt, welche rechtlichen Aspekte beim praktischen Einsatz von Honeypots zum Tragen kommen, einschließlich datenschutzrechtlicher Gesichtspunkte. Hybride Verfahren kombinieren bewährte Honeypot-Technologien miteinander, um komplexe Honeypots- und Honeynetze mit tausenden von Systemen aufzubauen. Diese Systeme haben das Potential, besonders komplexe Setups beherrschbar zu machen und den Honeypot-Einsatz in Zukunft durch fertige, benutzbare Lösungen zu vereinfachen.

Es wurde aktuelle Software aus den zuvor vorgestellten Kategorien ausgewählt, die sich im praktischen Einsatz bereits bewährt und gute Ergebnisse geliefert hat. Die individuellen Stärken und Schwächen dieser Projekte wurden hervorgehoben und dabei auch auf die Erweiterbarkeit um zusätzliche Funktionalität geachtet, um die Zukunftstauglichkeit dieser Lösungen abschätzen zu können. Zusätzlich wurden Honeypot-Konzepte vorgestellt, die sich derzeit noch in der Entwicklung befinden und auf Themenfelder konzentrieren, die erst in jüngster Vergangenheit verstärkt in Erscheinung getreten sind: SCADA-Honeypots, die verwundbare, über das Netzwerk erreichbare Industriesteuerungsanlagen vortäuschen und „Mobile Honeypots“, deren Fokus besonders auf der Simulation von Mobilgeräten wie Smartphones und

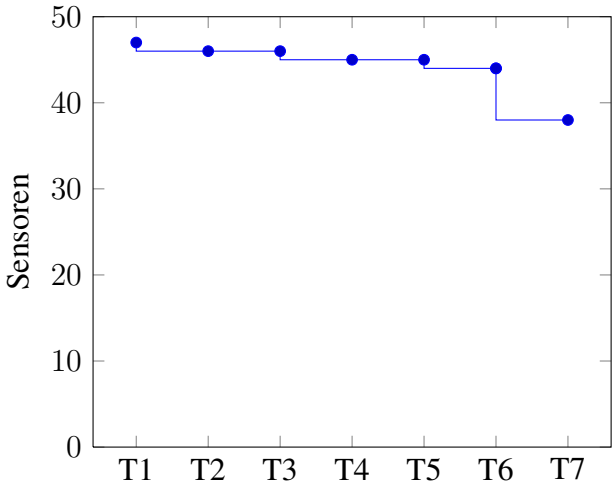

Abbildung 10: An den einzelnen Tagen im Messzeitraum aktive Sensoren

Tablet-Computern liegt.

Im praktischen Teil der Arbeit wurde ein Netzwerk aus 50 weltweit verteilten Sensoren errichtet, die alle mit drei verschiedenen Low-Interaction-Honeypots ausgestattet waren und deren in einem einwöchtigen Testzeitraum gesammelte Daten an zentraler Stelle ausgewertet werden konnten. Es wurde gezeigt, welche Schritte zur Absicherung und Kontrolle des Netzwerks, sowie zur vollautomatischen Installation neuer Sensoren unternommen wurden. Die Auswertung der Protokolle hat große Unterschiede in der Menge der gezählten und als gefährlich eingestuften Netzwerkverbindungen der jeweiligen Sensoren offenbart, sowie gezeigt, dass die Computerwürmer *Conficker* und *W32.Mydoom* trotz ihres Alters noch immer weit verbreitet sind.

Ausblick Es existieren noch weitere Aspekte auf dem Themenfeld der Honeypots, die im Rahmen dieser Arbeit nicht oder nur grob beleuchtet wurden. So wäre es interessant herauszufinden, wie ein Angreifer etablierte Honeypots als solche zu identifizieren versucht und mit welchen Mitteln die Täuschung schließlich weiter verbessert werden kann. Inwieweit sich High-Interaction-Honeypots als Sandbox zur Verhaltensanalyse von Malware einsetzen lassen, wäre ein weiteres zu untersuchendes Thema. Leider war es nicht möglich, im Rahmen dieser Arbeit Statistiken über den praktischen Einsatz von Honeypots in Unternehmen oder Institutionen zu gewinnen. Dies ist insofern verständlich, als dass diese Einrichtungen potentiellen Angreifern keine Informationen über ihre Sicherheitsstruktur preisgeben möchten. Es wäre trotzdem interessant, auf anonymer Basis Praxiserfahrungen mit dieser Technologie zu sammeln und als Feedback zur Verbesserung existierender Lösungen einzusetzen. Hybride Honeypots wurden im Rahmen dieser Ausarbeitung nur am Rande erwähnt, bieten jedoch genügend Raum für zusätzliche Forschung zur Optimierung der Interaktion und Kombination verschiedenster Honeypot-Technologien.

Literatur

[1] Reto Baumann and Christian Plattner. White Paper: Honeypots. 2002.

[2] Leyla Bilge and Tudor Dumitras. Before We Knew It. 2012.

[3] Ronald Eikenberg. Kritische Schwachstelle in hunderten Industrieanlagen. http://www.heise.de/security/meldung/Kritische-Schwachstelle-in-hunderten-Industrieanlagen-1854385.html. Abgerufen am 28.05.2013.

[4] Ronald Eikenberg. Mit dem virtuellen Atomkraftwerk auf Hackerfang. http://www.heise.de/newsticker/meldung/Mit-dem-virtuellen-Atomkraftwerk-auf-Hackerfang-1864586.html. Abgerufen am 28.05.2013.

[5] Julie Tate Ellen Nakashima, Greg Miller. U.S., Israel developed Flame computer virus to slow Iranian nuclear efforts, officials say. http://articles.washingtonpost.com/2012-06-19/world/35460741_1_stuxnet-computer-virus-malware. Abgerufen am 28.05.2013.

[6] Jan Göbel. *Amun: A Python Honeypot*. Universität Mannheim/Institut für Informatik, 2009.

[7] Juszczyk Kijewski Pawlinski Grudziecki, Jacewicz. Proactive Detection of Security Incidents. 2012.

[8] Ralf Hofstetter. *Honeypots mit dynamisch erzeugten Angriffszielen zur Täuschung von potentiellen Angreifern*. PhD thesis, Universität Zürich, 2004.

[9] Xuxian Jiang, Dongyan Xu, and Yi-Min Wang. Collapsar: a VM-based honeyfarm and reverse honeyfarm architecture for network attack capture and detention. *Journal of Parallel and Distributed Computing*, 66(9):1165–1180, 2006.

[10] ju. Smartphone-Honeypots im Mobilfunknetz der Telekom. http://www.heise.de/security/meldung/Smartphone-Honeypots-im-Mobilfunknetz-der-Telekom-1630359.html. Abgerufen am 28.05.2013.

[11] kbe. ENISA wirbt für digitale Hackerfallen. http://www.heise.de/security/meldung/ENISA-wirbt-fuer-digitale-Hackerfallen-1758563.html. Abgerufen am 28.05.2013.

[12] David Moore, Colleen Shannon, Geoffrey M Voelker, and Stefan Savage. *Network Telescopes: Technical Report*. Department of Computer Science and Engineering, University of California, San Diego, 2004.

[13] Collin Mulliner, Steffen Liebergeld, and Matthias Lange. Poster: Honeydroid-creating a smartphone honeypot. In *IEEE Symposium on Security and Privacy*, 2011.

[14] N.N. Genii Honeynets. http://old.honeynet.org/papers/gen2/. Abgerufen am 28.05.2013.

[15] N.N. ILOM (Integrated Lights Out Manager) for x86. http://www.googlux.com/ilom_x86.html. Abgerufen am 28.05.2013.

[16] N.N. IP-Adressen - personenbezogene Daten. http://www.datenschutzbeauftragter-info.de/fachbeitraege/ip-adressen-personenbezogene-daten/. Abgerufen am 28.05.2013.

[17] N.N. kippo Honeypot Website. http://code.google.com/kippo/. Abgerufen am 28.05.2013.

[18] N.N. MS08-067 Still Alive and Kicking. http://blog.securestate.com/ms08-067-still-alive-and-kicking/. Abgerufen am 28.05.2013.

[19] N.N. MyDoom A Worm Proxy. http://www.symantec.com/security_response/attacksignatures/detail.jsp?asid=20400. Abgerufen am 28.05.2013.

[20] N.N. Nepenthes Documentation. http://nepenthes.carnivore.it/documentation. Abgerufen am 28.05.2013.

[21] N.N. Path Traversal vulnerability in VMware's shared folders implementation. http://www.coresecurity.com/content/advisory-vmware. Abgerufen am 28.05.2013.

[22] N.N. PlanetLab Europe Basics. https://www.planet-lab.eu/files/PlanetLab_Basics.pdf. Abgerufen am 28.05.2013.

[23] N.N. SCADA Honeynet. http://www.digitalbond.com/tools/scada-honeynet/.

[24] N.N. Norton Cybercrime Report. 2012.

[25] N.N. Symantec Intelligence Report, Februar 2012. 2012.

[26] N.N. SOPHOS Security Threat Report 2013. 2013.

[27] N.N. Symantec Intelligence Report, Februar 2013. 2013.

[28] James P Owens Jr. *A study of passwords and methods used in brute-force SSH attacks*. PhD thesis, Clarkson University, 2008.

[29] Sebastian Poeplau and Jan Gassen. A honeypot for arbitrary malware on USB storage devices. In *Risk and Security of Internet and Systems (CRiSIS), 2012 7th International Conference on*, pages 1–8. IEEE, 2012.

[30] Georgios Portokalidis and Herbert Bos. SweetBait: Zero-Hour Worm Detection and Containment Using Low- and High-Interaction Honeypots. *Elsevier Computer Networks, Special Issue 'From Intrusion Detection to Self-Protection'*, 51(5):1239–1255, April 2007.

[31] Georgios Portokalidis, Asia Slowinska, and Herbert Bos. Argos: an Emulator for Fingerprinting Zero-Day Attacks. In *Proc. ACM SIGOPS EUROSYS'2006*, Leuven, Belgium, April 2006.

[32] Christian Wolf Prof. Dr. Dr. Eric Hilgendorf. Internetstrafrecht - Grundlagen und aktuelle Fragestellungen.

[33] Niels Provos and Thorsten Holz. *Virtual Honeypots: From Botnet Tracking to Intrusion Detection*. Addison-Wesley Professional, first edition, 2007.

[34] Goaletsa Rammidi. Survey on Current Honeynet Research.

[35] Sebastian Reitenbach. Planung und Aufbau eines virtuellen Honeynetzwerkes. 2004.

[36] J Rocaspana. SHELIA: a client honeypot for client-side attack detection. *http://www.cs.vu.nl/~{}herbertb/misc/shelia*, 2009.

[37] Christian Seifert, Ian Welch, and Peter Komisarczuk. Taxonomy of Honeypots. 2006.

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

[39] Lance Spitzner. Honeytokens: The other Honeypot, 2003.

[40] Lanz Spitzner. Know Your Enemy: Honeywall CDROM roo 3rd Generation Technology. http://old.honeynet.org/papers/cdrom/roo/, 2005.

[41] Stefan Vömel. *Using Honeypots to Capture and Analyze Malicious Activities on the Internet*. PhD thesis, Universität Mannheim, 2009.

[42] Matthias Wählisch, Sebastian Trapp, Christian Keil, Jochen Schönfelder, Jochen Schiller, et al. First Insights from a Mobile Honeypot. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 305–306. ACM, 2012.

[43] Yi-Min Wang, Doug Beck, Xuxian Jiang, Roussi Roussev, Chad Verbowski, Shuo Chen, and Sam King. Automated Web Patrol with Strider Honeymonkeys. In *Proceedings of the 2006 Network and Distributed System Security Symposium*, pages 35–49, 2006.

[44] Georg Wicherski. Medium Interaction Honeypots. *German Honeynet Project*, 2006.

[45] Kyle Wilhoit. Wer steckt tatsächlich hinter den Angriffen auf ICS-Ausrüstung? 2013.