UNIVERSITÄT
MANNHEIM

Identification and Recognition of Remote-Controlled Malware

Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
der Universität Mannheim

vorgelegt von

Christian Jörn Dietrich
aus Düsseldorf, Deutschland

Mannheim, 2012

Dekan:	Prof. Dr. Heinz Jürgen Müller, Universität Mannheim
Referent:	Prof. Dr. Felix Christoph Freiling, Universität Erlangen-Nürnberg
Korreferent:	Prof. Dr. Christopher Kruegel, University of California, Santa Barbara

Tag der mündlichen Prüfung: 28. März 2013

Abstract

Remote-controlled malware, organized in so-called botnets, have emerged as one of the most prolific kinds of malicious software. Although numbers vary, in extreme cases such as Conficker, Bredolab and Mariposa, one botnet can span up to several million infected computers. This way, attackers draw substantial revenue by monetizing their bot-infected computers.

This thesis encapsulates research on the detection of botnets – a required step towards the mitigation of botnets. First, we design and implement SANDNET, an observation and monitoring infrastructure to study the botnet phenomenon. Using the results of SANDNET, we evaluate detection approaches based on traffic analysis and rogue visual monetization.

While traditionally, malware authors designed their botnet command and control channels to be based on plaintext protocols such as IRC, nowadays, botnets leverage obfuscation and encryption of their C&C messages. This renders methods which use characteristic recurring payload bytes ineffective. In addition, we observe a trend towards distributed C&C architectures and nomadic behavior of C&C servers in botnets with a centralized C&C architecture, rendering blacklists infeasible. Therefore, we identify and recognize botnet C&C channels by help of traffic analysis. To a large degree, our clustering and classification leverage the sequence of message lengths per flow. As a result, our implementation, called *CoCoSpot*, proves to reliably detect active C&C communication of a variety of botnet families, even in face of fully encrypted C&C messages.

Furthermore, we observe that botmasters design their C&C channels in a more stealthy manner so that the identification of C&C channels becomes even more difficult. Indeed, with Feederbot we found a botnet that uses DNS as carrier protocol for its command and control channel. By help of statistical entropy as well as behavioral features, we design and implement a classifier that detects DNS-based C&C, even in mixed network traffic of benign users. Using our classifier, we even detect another botnet family which uses DNS as carrier protocol for its command and control.

Finally, we show that a recent trend of botnets consists in rogue visual monetization. Perceptual clustering of SANDNET screenshots enables us to group malware into rogue visual monetization campaigns and study their localization as well as monetization properties.

Zusammenfassung

Fernsteuerbare Schadsoftware, zusammengeschaltet in sog. Botnetzen, hat sich mittlerweile zu einer sehr verbreiteten Art an Schadsoftware entwickelt. Obwohl die genauen Zahlen mitunter schwanken, so zeigt sich in Extremfällen wie etwa bei Conficker, Bredolab und Mariposa, dass ein einzelnes Botnetz aus infizierten Computern mit bis zu zweistelliger Millionenanzahl besteht. Die Angreifer erwirtschaften somit erhebliche Einkommen, indem sie die infizierten Computer monetarisieren.

Diese Arbeit umfasst Forschungsarbeiten zur Erkennung von Botnetzen – ein notwendiger Schritt, um Botnetze zu entschärfen. Zunächst entwerfen und implementieren wir die Beobachtungsumgebung SANDNET, um das Botnetz-Phänomen detailliert untersuchen zu können. Mit Hilfe der Ergebnisse des SANDNET entwerfen und bewerten wir Erkennungsmechanismen, die sowohl auf Verkehrsflussanalyse des Netzwerkverkehrs als auch auf dem visuellen Eindruck der bösartigen Benutzerschnittstelle basieren.

Während Schadsoftware-Autoren in der Vergangenheit die Steuerkanäle (C&C) ihrer Botnetze häufig unter Verwendung von Klartext-Protokollen wie etwa IRC entworfen haben, so werden neuerdings fast ausschließlich verschleierte oder verschlüsselte C&C-Nachrichten verwendet. Dies verhindert Erkennungsmechanismen, die auf charakteristischen, wiederkehrenden Nutzdaten-Mustern basieren. Darüber hinaus lässt sich ein Trend hin zu verteilten C&C-Architekturen sowie ein nomadisches Umzugsverhalten der C&C-Server im Falle von Botnetzen mit zentralisierter C&C-Architektur erkennen. Auf diese Weise werden Blacklists von C&C-Endpunkten umgangen. Wir entwickeln daher einen Ansatz zur Identifikation und Wiedererkennung von Botnetz-C&C-Kanälen mit Hilfe von Verkehrsflussanalyse. Unser Ansatz basiert dabei in erster Linie auf der Sequenz von Nachrichtenlängen einer Netzwerkverbindung. In der Evaluation beweist unsere Implementierung *CoCoSpot*, dass sie auf verlässliche Art und Weise C&C-Kommunikation einer Vielzahl an verschiedenen Botnetz-Familien erkennen kann, selbst wenn die C&C-Nachrichten vollständig verschlüsselt sind.

Ferner beobachten wir, dass Botmaster ihre C&C-Kanäle unter erheblicher Berücksichtigung der Tarnung im Netzwerkverkehr entwerfen. Mit Feederbot zeigen wir, dass mittlerweile Botnetze existieren, die DNS als Trägerprotokoll für ihren C&C-Kanal verwenden. Mit Hilfe der statistischen Entropie sowie Ver-

haltenseigenheiten wird ein Klassifizierer entworfen und implementiert, der DNS-basierte C&C-Kanäle erkennen kann – selbst in gemischtem Netzwerkverkehr von legitimen Benutzern. Unter Verwendung unseres Klassifizierers entdecken wir sogar eine weitere Botnet-Familie, die DNS als Trägerprotokoll für ihren C&C benutzt.

Schließlich zeigen wir, dass ein aktueller Trend in der sog. rogue visual monetization liegt. Ein wahrnehmungsbasiertes Clustering von Screenshots des SANDNET ermöglicht es uns, Schadsoftware in Kampagnen der rogue visual monetization zu gruppieren und die Eigenschaften ihrer Lokalisierung und Monetarisierung zu studieren.

Acknowledgements

This thesis would have hardly been possible without the help and support of others. First of all, I would like to thank my supervisor Felix C. Freiling who by his guidance has significantly encouraged me and fostered my research. I have always enjoyed our inspiring discussions and appreciated your kind advice. Your positive outlook and kindness inspired me and gave me confidence.

I am deeply grateful to my colleague and friend Christian Rossow. Christian, you are — by far — the most remarkable, sharp-minded, humble and encouraging person I have ever worked with. It has been a great pleasure working with you.

Furthermore, I am thankful to Norbert Pohlmann, for supporting me over many years and giving me the opportunity to work in the inspiring environment at his Institute. I appreciate to have learned quite some lessons not limited to academic matters.

Moreover, I thank the team at the Institute for Internet Security at the University of Applied Sciences Gelsenkirchen, in particular Christian Nordlohne, who supported me with the analysis that influenced the C&C flow fingerprinting in this thesis.

Many thanks also to my second supervisor Christopher Kruegel. I really enjoyed to work with you and the people in UCSB's seclab. I am especially thankful to — alphabetically ordered — Adam Doupé, Yanick Fratantonio, Alexandros Kapravelos, Gianluca Stringhini and Ali Zand. Furthermore, I would like to thank everyone at Lastline.

A big thank you to my friend Philipp von Cube who bore with me for many discussions and always supported me in times of decision-making.

To conclude, I wish to extend a huge thank you to my parents and my brother for their help and support all the way. You always supported all the decisions I made, without asking for anything in return. Without your care and discipline as well as the ways you have paved, I would simply not be the person I am today. Thank you very much for this.

Last but not least I would like to thank my dear Julia. You have always been (and will always be) the one to make my day. I am deeply grateful not only for your inspiration and advice, but also for your love. This dissertation is dedicated to you.

Contents

Chapter 1

Introduction

1.1 Motivation

Malicious software, often referred to as malware, poses a severe problem to today's information technology. While computer viruses have been around for more than 25 years, nowadays, a prevalent subset of malware is organized in a remote-controllable fashion. An attacker merely infects computers, or more generally, IT systems of innocent victims in order to remotely execute arbitrary software. Thus, the change towards remote-controlled malware enables attackers to have maximum flexibility concerning their monetization. Typically, an attacker aggregates her infected computers in a network. Such a network of computers infected with remote-controlled malware is referred to as a botnet.

Adversaries monetize botnets in a variety of ways, e.g., by sending large amounts of unsolicited commercial email (spam), ad fraud, stealing banking credentials of the infected computers' users in order to mislead financial transactions or by luring users into buying rogue software. Some botmasters build on extortion, and if the victim does not pay, the botnet performs a distributed denial of service attack, effectively knocking the victim offline. Recent studies on the underground economy reveal potential revenues as well as the damage induced by remote-controlled malware. Botnets such as Koobface [Vil10, TN10] focus solely on pay-per-click or pay-per-install fraud, while still earning more than two million US dollars per year. Similarly, the Storm botnet is expected to have produced a yearly revenue of 3.5 million US dollars [KKL+09]. Other botnets have specialized in distributing rogue software, e.g., fake antivirus software and drew combined revenues of over 130 million US dollars a year [SGAK+11]. In November 2011, operation Ghost Click addressed the takedown and prosecution of DNSChanger, a botnet that generated more than 14 million US dollars in fraudulent advertising revenue by help of hijacked DNS resolution of the victim computers [Gal11].

But neither are botnets a problem of isolated spots, nor is it always necessarily the monetization technique alone that causes damage. For example, the Mariposa

botnet has comprised more than 13 million infected computers in more than 190 countries [SBBD10]. Even worse, the Conficker botnet variants A-D are believed to have infected between 9 and 15 million computer systems worldwide [Onl09, UPI09], some even report up to 25 million infections [SGRL12]. In addition, without ever exposing a monetization technique at all, Conficker variants A-D caused severe problems in several institutions just by collateral damage of the infections. Fighter planes of the French military were unable to take off due to Conficker infections of related computer systems [Tel09]. Likewise, British warships suffered from outages caused by Conficker infections [Reg09]. These incidents are examples of the severity of malicious, remote-controllable software.

Clearly, the damage caused by botnets has reached a substantial extent, possibly even endangering society. As a first step, we need to design accurate and reliable detection methods for botnets. Being able to detect and measure the impact of botnets on a large scale serves as a basis for subsequent actions, eventually – if legal frameworks allow – leading to disinfections or takedowns.

A key characteristic of malicious remote-controlled software lies in its ever-changing appearance. This can be observed in a variety of ways. For example, in case of malicious remote-controlled software binaries, this property is often realized by so-called packing, which refers to the process of re-coding the software binary while adding random elements, applying encryption or obfuscation. The same observation holds for a certain type of network traffic emitted by malicious remote-controlled software, namely its command and control traffic. Typically, botnet command and control channels employ some kind of encryption or obfuscation technique in order to avoid characteristic payload substring patterns, which would serve as identification and recognition attribute. As a third example, let us refer to the user interfaces of rogue visual malware, such as fake antivirus software. Although adhering to a similar user interface structure, their visual appearance employs slight variations in the details of their user interface elements.

In summary, motivated by the need to evade detection, malicious software strives for an ever-varying appearance. Even several samples of one kind of malicious remote-controlled software, e.g., what can naively be considered one bot family, expose different appearances. As a result, the classification of malicious software and its network traffic is rendered a challenge.

1.2 Contributions: Countering Deviance

This ever-changing appearance hinders the detection of both, malicious remote-controlled software as well as its network traffic. However, this thesis aims at identifying and exploiting properties of remote-controlled malicious software that – despite evasion techniques of malware – may serve for detection. This methodology builds upon two main observations.

First, the variation of malware properties is hardly ever complete. While malware authors typically address some properties specifically, such as the encoding of C&C messages by help of encryption, there is still room for detection approaches in more subtle ways. In other words, we try to identify those properties that are less frequently affected by variation, e.g., traffic analysis on botnet command and control communication. As an example, the traffic analysis approach used in *CoCoSpot* as described in Chapter 4 leverages the sequence of message lengths in botnet command and control channels. While encrypting C&C messages results in varying payloads and thus avoids to recognize botnet C&C channels by payload patterns, the traffic analysis properties still reveal characteristic recognition attributes.

Second, the variation of malware sample properties is bounded. Using an appropriate feature space, even though samples may vary, this variation is typically of limited scope, so that a large set of samples exhibits clusters of similar sample instances. From these clusters, characteristic structural templates can be extracted and used as recognition attributes. For example, this approach is used in this thesis to infer user interface templates of rogue visual software, such as fake antivirus software, by help of perceptual screenshot clustering. In this case, the variation of the user interface is bounded in the sense that only very slight changes are made to the user interface elements, while the overall perception remains the same. Using our perceptual clustering approach, we reveal the underlying structure of the user interface which can be used as a recognition feature.

Telling from the facts, the number of observed MD5-distinct malware binaries have increased for years [CDKL09]. Figure 1.1 shows the total number of MD5-distinct malware binaries per year from 1984 to 2012 as measure by AV-TEST GmbH. On the one hand, it is difficult – if at all possible – to measure whether this increase in unique malware binary hash sums relates to an increase in distinct malware families. Nevertheless, on the other hand, this trend might also be a result of the variations of malicious software binaries, such as packing. In this case, it is difficult to judge, whether the number of distinct families increases, too. However, it is desirable to understand and extract detection attributes on a per-family basis – instead of for each sample individually. Therefore, we also provide a perspective on malware families. In the scope of this thesis, we define the term *family* in each specific context where it is used. For example, in the context of *CoCoSpot*, we consider a family as the subset of malicious remote-controlled software executions which share the same command and control protocol. Using the techniques proposed in this thesis, even though no general definition of a malware family can be concluded, the notion of a malware family depending on the context will be fostered.

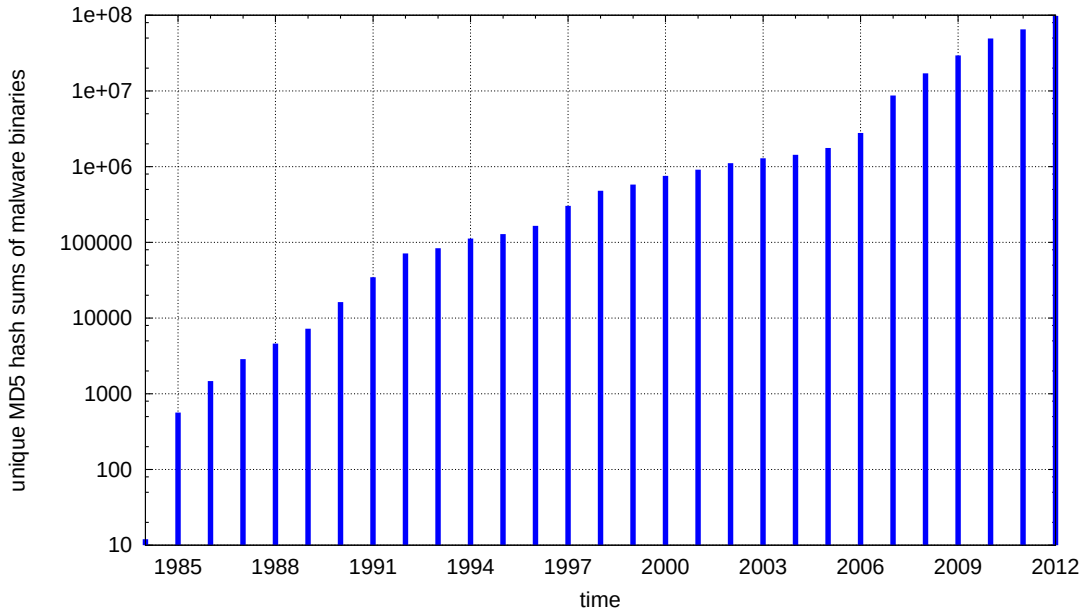


Figure 1.1: Total number of MD5-distinct malware binaries per year from 1984 to 2012 as measured by AV-TEST GmbH [Avt12]

1.2.1 Detection := Identification + Recognition

In this thesis, the detection process is separated into the two subtasks of *identifying* and *recognizing* remote-controlled malicious software and its network traffic. While the identification deals with the decision whether a certain samples is to be considered malicious, the recognition aims at finding (additional) related instances. Note that in the scope of this thesis, the term "sample" does not necessarily only refer to a malicious binary, but rather to one element of a dataset, such as a C&C network flow or a screenshot.

The identification task, i.e., the decision if a certain sample is to be considered malicious, highly depends on the definition of maliciousness, and thus on the context of the decision. By its very nature, it is thus a subjective decision, typically made by a human analyst. This thesis aims at providing means to aid a human analyst in the decision making process. For example, the clustering phase of *CoCoSpot*, a system developed as part of this thesis, exposes similar communication patterns in network traffic and aggregates similar network flows into subsets. A human analyst will only need to analyze some of the flows per subset in order to make the decision concerning maliciousness, thus effectively reducing the work load. Similarly, our screenshot clustering approach enables an analyst to quickly grasp the visual appearance of a subset of samples.

In the recognition phase, we aim at finding related samples. In the face of variation, the recognition task does not only involve exact pattern matching, but

requires fuzzy pattern matching. While the recognition features are usually extracted from a subset of samples, for example, the training samples compiled in the identification phase, this set does not represent all possible values, i.e., the training sample set is not guaranteed to be complete. As an example, the message length sequences for a subset of similar network traffic samples for the Virut family's command and control channels might expose a characteristic detection attribute. However, we will hardly ever be able to compile a complete set of training samples for Virut C&C traffic, due to variation. In other words, compared to the training samples, another instance of a Virut command and control channel, e.g., found in the wild, will most certainly again vary slightly concerning the message length sequence and thus differ from all of the training samples. As a result, not only the identification phase needs to be aware of this variation, but also the recognition phase must be designed with regard to a certain level of fuzziness. To sum up, it is exactly this deviance which renders the detection of malicious software a challenge. Thus, our separation of identification and recognition allows us to develop specifically tailored methods for each of the subtasks.

1.2.2 List of Contributions

This thesis aggregates work on the identification and recognition of malicious remote-controlled software in terms of the following contributions.

Sandnet – A contained environment for dynamic malware analysis

Before we can apply machine learning, we need to collect data to be used in our experiments. While nowadays, malware is apparently ubiquitous, gathering malicious datasets from different parties might not seem to be a challenge. However, using datasets from different sources raises concerns because they might not be comparable. In order to strive for correct and sound experiments, we aim to avoid artifacts in datasets, such as those induced by different source environments, as much as possible. Therefore, we need a dataset where a large number of diverse malware has been executed under very similar conditions and within the same environment. We solve this problem by designing Sandnet, our contained dynamic malware analysis environment. In addition, we implement an extensive post-processing chain which extracts and transforms the raw execution results into structured and easily processable form and enriches them by correlating with external sources, such as origin AS and routing paths from BGP feeds, geolocalization of IP addresses as well as reverse DNS information. Using Sandnet, we are able to compile a dataset of samples stemming from the same environment, for a given experiment. Chapter 3 describes the design, usage and workflow of Sandnet. Sandnet has been developed in joint work with Christian Rossow.

Recognition of command and control plane communication

Earlier work on detecting botnets developed means to automatically infer characteristic payload substring patterns of botnet C&C. Those substrings could then be used as payload signatures in the recognition phase. Albeit, over the last few years, botnets have evolved and nowadays the majority of botnets employ obfuscated or encrypted command and control protocols. In addition, botnets exhibit more and more nomadic C&C servers, i.e., migrate their C&C servers on a regular basis from one domain, IP address range or Autonomous System to another. As a consequence thereof, detecting C&C flows of these modern botnets is truly rendered a challenge, especially since encryption defeats payload pattern matching and a frequent migration of C&C servers turns blacklists inefficient.

It may seem unlikely to still be able to detect such C&C channels. However, in Chapter 4, we address the problem of recognizing command and control flows of botnets and show that, using traffic analysis features, we can infer a model to correctly classify C&C channels of more than 35 distinct prevalent bot families among network traffic of contained malware analysis environments. A key feature of our traffic analysis approach lies in the sequence of message lengths of C&C flows.

Detecting botnets with DNS as carrier for command and control

Traditionally, botnets designed their C&C protocols to be based on IRC and later on HTTP. Similarly, a body of related work exists on the detection of IRC- and HTTP-based command and control protocols. However, taking disguise of botnet command and control channels to the next level, we have discovered Feederbot, a botnet that uses the DNS protocol as carrier for its command and control. Being the first of this kind, we reverse engineered and investigated this botnet in detail, disclosing the techniques employed to hide their encrypted C&C traffic in regular DNS requests and responses.

Additionally, we face the challenge to design a detection approach. Although the botmasters employ DNS tunneling techniques, we show in this thesis that our specifically tailored method can still detect botnets that use DNS as carrier protocol for its C&C. Using our classifier, we have even discovered an independent second botnet that, too, builds its C&C upon the DNS protocol. Furthermore, we evaluate our approach on mixed network traffic with benign users' network traffic in order to show that this approach can even be used in real-world environments to detect DNS-based botnets.

Detection of visual monetization plane activities

Given the fact that remote-controlled malware depends on network communication, aiming to detect malware based on command and control traffic features, as described in Chapters 4 and 5, seems natural. Complementary to the command

and control plane, we can try to detect malicious software by its monetization technique. Recently, one important monetization technique has consisted in a kind of malicious software termed rogue visual malware, which aims at luring the innocent user into spending money on a rogue software product. Prevalent examples of rogue visual malware are fake antivirus software (Fake A/V) and ransomware.

In this thesis, we study essential properties of rogue visual malware and propose a method to recognize malware based on its graphical user interface. Especially in face of very low antivirus detection rates of rogue visual malware binaries – ranging as low as 10% of all binaries per campaign – our methodology points towards a new approach of detecting malware by exploiting its visual perception. In addition, our approach supports a human analyst in identifying rogue visual malware campaigns. Furthermore, using the results of our approach, we provide insights into the monetization and localization habits of rogue visual malware campaigns, based on data of more than two years.

1.3 Thesis Outline

This thesis is structured as follows. Since this work combines techniques from machine learning with the task of detecting malware, Chapter 2 provides background information on properties of malicious software as well as recurring concepts of machine learning. We will cover the general process on how to apply unsupervised and supervised machine learning to a given malware detection approach, starting with the compilation of a dataset and concluding with evaluation methods.

Subsequently, motivated to compile a correct and sound dataset, Chapter 3 describes our approach of building a contained environment for dynamic malware analysis. In addition, we provide examples on how to inspect the results of a binary execution in order to assign ground truth labels to samples of the dataset. For subsequent experiments throughout this thesis, we will always refer to the Sandnet dataset.

Recently, botnets have evolved and many bot families now employ obfuscated or encrypted command and control channels. This renders previous approaches of recognizing botnet C&C channels ineffective. For example, payload pattern matching is defeated by C&C message encryption. Furthermore, the nomadic character of modern centralized botnets, where the C&C server is migrated from one domain, Autonomous System and IP address range to another, avoids blacklist approaches. The same holds true for botnets with a distributed C&C architecture. Despite the countermeasures taken by botmasters, we will address the problem of recognizing command and control flows in Chapter 4. We will show that our methodology of exploiting traffic analysis features successfully recognizes even encrypted botnet C&C channels.

Having identified a new kind of botnet C&C, namely botnets that use DNS

as carrier for its command and control protocol, in Chapter 5, we provide a case study on such a botnet and design a detection approach. Again, we show that our classifier successfully detects DNS-based botnets, effectively revealing another botnet which uses DNS as C&C carrier protocol. Furthermore, our approach is even able to detect DNS-based botnets in network traffic mixed with that of benign users.

Chapter 6 focuses on a complementary detection approach for remote-controlled malware by exploiting the monetization visibility. Rogue visual malware is a class of malware that builds on graphical user interfaces, a key property that we exploit in our detection methodology. We show that the similarity among graphical user interfaces of one family is reflected in our perceptual clustering approach, effectively structuring a set of more than 200,000 executions of malware binaries. Concluding our work on rogue visual malware, we provide insights into and compare monetization and localization means of Fake A/V and ransomware campaigns.

Finally, Chapter 7 concludes this thesis by providing a summary and outlining directions of future research.

1.4 List of Publications

This thesis aggregates research on the detection of malicious remote-controlled software. While containing unpublished work, in large part, it consists of the following publications:

- "Sandnet: Network Traffic Analysis of Malicious Software" (co-authored with Christian Rossow) [RDB+11]
- "On Botnets that use DNS for Command and Control" [DRF+11]
- "*CoCoSpot*: Clustering and Recognizing Botnet Command and Control Channels using Traffic Analysis" [DRP12a]
- "Exploiting Visual Appearance to Cluster and Detect Rogue Software" [DRP13]

The part concerning the design and implementation of Sandnet in Chapter 3 is joint work with Rossow and has not been published before. Subsequent results have been published together with Rossow, Bos, Cavallaro, van Steen, Freiling and Pohlmann [RDB+11].

The identification and recognition of botnet command and control channels as proposed in Chapter 4 are based on a journal article together with Rossow and Pohlmann [DRP12a]. Additionally, this work has been refined and re-evaluated on a larger dataset in August 2012.

Chapter 5 provides our insights on botnets using DNS as carrier protocol for command and control. This work has been published in a paper together with Rossow, Freiling, Bos, van Steen and Pohlmann [DRF+11].

Furthermore, we contributed to the following publications, but they have not been included in this thesis:

- "eID Online Authentication Network Threat Model, Attacks and Implications" [DRP12b]
- "Large-Scale Analysis of Malware Downloaders" [RDB12]
- "Prudent Practices for Designing Malware Experiments: Status Quo and Outlook" [RDK+12]
- "Manufacturing Compromise: The Emergence of Exploit-as-a-Service" [GBC+12]

Chapter 2

Background

Before digging into the technical approaches of detecting remote-controlled malware, this chapter provides background information on recurring terms and concepts throughout this thesis. The methodologies that were developed as part of this thesis combine techniques from the domain of machine learning with the field of malware detection. Therefore, we will discuss basic aspects of both areas. First, we provide a definition and an overview over botnets or more formally, remote-controlled malware. Subsequently, we introduce machine learning techniques used throughout the remaining experiments of this thesis.

2.1 Remote-Controlled Malware

Increasingly, malware depends on network communication. To a great degree, current malicious software is programmed in a remote-controllable manner. It receives instructions over the Internet and sends back information. The property of being remote-controlled enables an attacker – in this context referred to as botmaster – to change the functionality of the malware after having infected computers in the wild. Whenever the bot master faces countermeasures, the bots can thus be updated in order to circumvent the countermeasures. Furthermore, there is a variety of ways how malware monetizes infected computers, most of which heavily depend on Internet communication. Consequently, network communication is inevitable for most modern malware.

Definition. *For the scope of this thesis,* remote-controlled malicious software *is defined as software which fulfills the following two conditions:*

- *The software is remote-controlled by means of network communication.*
- *The software employs malicious monetization, i.e., it monetizes information or resources from or via the victim user or his/her computer.*

Thus, remote-controlled malicious software requires a network-based command and control (C&C) plane. The C&C plane is used to instruct the bots and to report back to the controlling unit. For example, the C&C plane is used for the botmaster to instruct a bot to send spam and, vice versa, to report on the mail submission back to the C&C peer. In addition, the monetization plane covers the techniques to monetize on the victim. For example, such monetization might include the actual sending of spam messages, performing click fraud, denial-of-service attacks as well as to steal personal information such as online banking credentials.

Command and Control Plane

The command and control plane is an essential component of malicious remote-controlled software and enables an attacker to remotely instruct instances of its software. Figure 2.1 shows the two prevalent C&C architectures of malicious remote-controlled software. The command and control architecture is separated into centralized and distributed structures. While a centralized structure consists of a single controlling unit, possibly enhanced by one or more backup controlling units, the distributed C&C architecture exhibits an underlying distributed system such as a peer-to-peer network. In the later case, every bot can potentially act as a C&C peer by distributing commands or aggregating gathered information from other peers. A significant advantage of the distributed C&C architecture in terms of resilience is to avoid a single point of failure, which in the centralized C&C architecture translates to the dedicated C&C server entity.

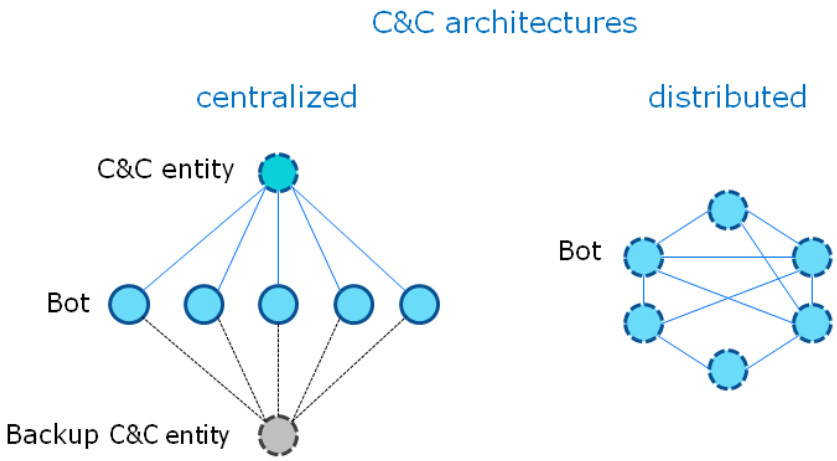


Figure 2.1: Centralized and distributed command and control architectures

As each malware author designs and defines the details of the command and control plane of her bot families, the C&C plane implementations are typically

very diverse in nature. This means that, on the one hand, the unique character of a bot family's C&C plane implementation – once grasped – might serve well as a recognition property. On the other hand, the lack of general constraints on the C&C plane allows the attacker to define the C&C plane techniques in a flexible and volatile way. Botmasters are free to change the C&C plane technique frequently, or, with steganography in mind, design it in a very similar fashion as well-known legitimate communication means.

We will use the following definition of the term *C&C protocol*.

Definition. *A C&C protocol is the protocol that is used to communicate instructions and reports, signal a bot's state and availability as well as transmit bot program updates between one or more bots and the controlling entity, the botmaster.*

In face of the detection of command and control plane communication, we observe that once the C&C plane communication is identified, it serves as a reliable recognition attribute. However, as the C&C plane communication may be encrypted or obfuscated, the identification itself can be a challenging task.

Monetization Plane

Malware authors target financial gain. Thus, the monetization plane grasps those techniques (and preparations thereof) that the attacker employs in order to have money transferred to her benefit. Malicious remote-controlled software can be separated concerning the visibility of monetization techniques on an infected computer. Sending spam email messages or logging keystrokes usually takes place in the background without the victim user noticing. Similarly, click fraud is typically performed as a background job without any interface elements noticeable to the user. This kind of monetization is thus considered covert. However, certain monetization techniques require the user to interact. For example, a recent monetization technique consists in fake antivirus software luring the user into buying a fake software product. Likewise, ransomware extorts users to pay in order to unlock their computers. In the latter cases, the malware is forced to display a user interface and its monetization technique can thus be classified as exposed.

In contrast to the C&C plane, to a certain degree, the techniques in the monetization plane have to obey outer constraints. For example, a spam bot has to comply with the general email delivery process, e.g., speaking the SMTP protocol and addressing well-defined mail servers. Similarly, a fake antivirus software has to expose a certain user interface which lures the victim into spending money.

These constraints may help in the detection process, because the malware authors are restricted, e.g., in the network protocols and their usage. Thus, the obfuscation of the monetization plane activities becomes much more limited. More precisely, with regard to detection, we observe that monetization plane activities form monetization-dependent recognition attributes. However, when

compared to C&C plane recognition, monetization plane recognition attributes can be more volatile because the duration of monetization campaigns may be shorter than the C&C plane design and not all monetization techniques expose recognizable attributes.

2.2 Machine Learning for Malware Detection

One way of addressing and possibly mitigating the ever-changing appearance of malicious software is to infer structures and dependencies among malware which are not obvious at first sight. These structures may only be discovered when analyzing a diverse set of malware and in comparison with benign counterparts. For example, even though the malware binaries of a certain bot family employ an encrypted communication channel for command and control, we might still be able to model the communication behavior that is characteristic for this malware family. Such a model could only be inferred after having analyzed lots of different samples of the malware family in question in addition to a diverse set of other families' communication. Finally, if a model can be inferred, this communication model will serve as a means to detect further instances of the same malware family.

In order to programmatically follow this approach, we turn to machine learning. Machine learning techniques deal with the modeling of relationships and dependencies among sets of data. In this thesis, we apply machine learning to the domain of malware detection. The overall process of applying a machine learning technique consists of several steps. In the following, we will provide a general overview, partly enriched with recommendations based on the guidelines developed by Rossow et al. [RDK+12].

Dataset compilation. First, a dataset covering instances of the data in question must be compiled. In general, the more diversity the dataset covers, the more likely it is to also cover the difficult and challenging cases for the chosen approach. It is exactly those challenging cases, that help to judge if the derived model can reliably be used for detection in an application context. In practice, the size of the dataset and its diversity are typically limited by outer constraints such as limited time and a limited number of sensors where to acquire instances. For the experiments in this thesis, we often turn to the Sandnet dataset. In order to maximize diversity, for example, we distribute instances over several distinct malware families and points in time. Furthermore, we maximize the number of instances to be used for the experiment. Care should be taken to avoid artifacts in the dataset, as they might influence the learning process in an undesired fashion – especially when combining datasets of distinct sources which might exhibit different artifacts. If available, the instances of the dataset should be labeled using ground truth, because these labels allow for evaluation of sub-

sequently developed models. Based on the labels the dataset can be subdivided into classes. In addition, when labeled datasets are available, we try to avoid a skewed training set, i.e., we make sure that each class is represented with roughly the same number of instances in the training dataset.

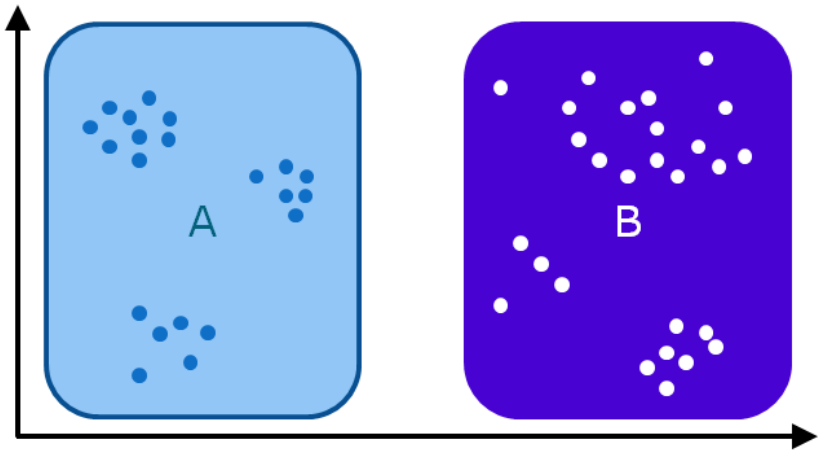
Feature Extraction. Second, features need to be defined and extracted from the dataset. The design and choice of the features depends largely on the specific goal of an experiment and its application. We will therefore provide the feature definitions in each of the subsequent chapters containing machine learning experiments. In general, having a certain experiment's goal in mind, the features should be able to express the distinction between any of the classes in the dataset involved.

Kinds of features for network traffic can be distinguished into numerical features, sequential features and syntactical features as proposed by Rieck [Rie09]. In addition, depending on the learning algorithm, features may need to be scaled and normalized. Features can be combined into a multidimensional vector. The aggregation of multiple features is referred to as a feature vector, spanning a feature space. For some learning algorithms, redundancy in the features of a feature vector may bias the learning result. As a precaution, we take care during the feature extraction process to be aware of dependencies or redundancies among features.

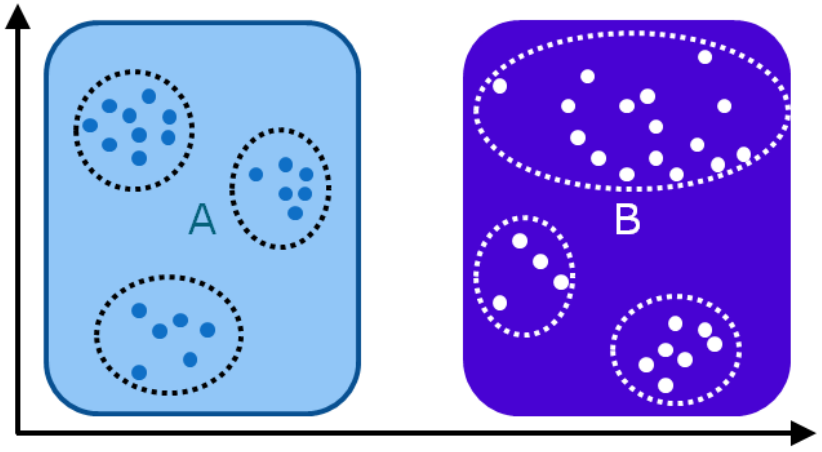
Dissimilarity or distance measure. Based on the feature space, the dissimilarity between any two instances is measured using a dissimilarity or distance function. Again, the specific distance function depends on the feature space, the context and the semantics of a certain experiment, and is thus defined in each experiment of this thesis individually. In general, a distance function is a function $d(u, v)$, which, given two feature vectors u and v, returns a value in the range $\mathbb{R}_0^+$. Similar feature vectors exhibit a low distance while dissimilar instances have a higher distance. Examples for distance functions are the Euclidean distance, the Hamming distance or the Levenshtein distance.

Furthermore, the distance function can be a function composed of several distance functions for each feature dimension. Assume a feature vector of two dimensions with one string feature and one set-of-bigram feature. While the string feature might fit the Levenshtein distance [Lev65], the bigram dimension requires a different distance function, such as the Jaccard distance [LW71].

Unsupervised learning. Machine learning techniques are divided into unsupervised and supervised learning. While unsupervised learning strives to structure a given dataset into subclasses, supervised learning targets the classification or prediction of previously unseen instances. Clustering is a prevalent example for unsupervised learning and it is particularly useful to find yet unknown properties



(a) An example dataset with two labeled classes A and B.



(b) Each class contains fine-grained clusters.

Figure 2.2: Clustering refines the labeled classes A and B (solid boxes) into fine-grained subclasses (dotted ellipses).

of the underlying data. Thus, it can be used to split an unlabeled dataset into clusters of similar instances. From the perspective of malware detection, clustering addresses the identification part where a set of samples is decomposed into clusters and each cluster can then be labeled individually.

Using a labeled dataset, clustering can be evaluated on the replicability of the classes in the dataset. In this case, the labels are not used during the clustering phase, but only in the evaluation Here, the accordance of the clusters with the labeled classes provides a measure of replicability by the clustering. Furthermore, depending on the resolution of the labels in a given labeled dataset, clustering can be used to refine the structure of the dataset, resulting in more compact subclasses. Figure 2.2(a) shows an example dataset with class labels A and B, depicted in a two-dimensional feature space. Clearly, the instances can be split into their labeled classes A and B. However, if applying clustering to this dataset, clusters that represent compact subclasses in each of the classes result. These clusters are depicted as dotted ellipses in Figure 2.2(b). This process can be helpful when the original class labels are coarse but the identification, or even later on the recognition, targets towards more fine-grained classes. One practical example is given by two coarse classes of network traffic samples, e.g., malicious and benign, while the malicious class consists of command and control flows of several distinct malware families. In this case, it might be useful to cluster the malicious class into subclasses per malware family, especially if the recognition phase aims at recognizing command and control flows on a per-family basis.

Supervised learning. Supervised learning derives a model from a set of training instances and, using this model, classifies previously unseen instances. Examples for supervised learning are classification and regression. While regression refers to prediction functions with continuous output values, classification deals with filing instances into discrete classes. In the scope of this thesis, referring to malware detection, supervised learning addresses the recognition task. Thus, we focus on classification instead of regression, because we typically deal with discrete classes.

Supervised learning can benefit from the results of a preceding unsupervised learning step, e.g., clustering. Particularly when the original dataset exhibits coarse labels, using the subclasses resulting from clustering instead can improve classification results.

Evaluation. Finally, in order to measure the accuracy of a detection approach, we turn to evaluation techniques. If possible, ground truth data is used to measure the accuracy. Evaluation is performed differently for unsupervised and supervised learning. In turn, the following sections will describe the evaluation approaches for clustering and classification.

2.2.1 Clustering Evaluation

Since hierarchical clustering is used in several experiments of this thesis, its evaluation methodology is described in a general fashion in this section. Nevertheless, we will provide a detailed evaluation procedure for each experiment. In general, clustering aims to structure a given dataset into clusters or subclasses of similar instances. Given a specific context, the clustering results can be evaluated by measuring the correspondence of clusters to previously assigned class labels, i.e., ground truth.

To evaluate how well a clustering represents the labeled dataset, we use two measures from the area of unsupervised learning and originally defined in the context of information retrieval [vR79]. First, we measure the *precision*, which represents how well our clustering separates instances of different classes into disjoint clusters. Second, we compute the *recall*, which measures if similar instances are grouped into the same cluster. Formally, let T be the set of ground truth classes in the labeled dataset, C the set of created clusters, $m = |T|$, $n = |C|$ and the total number of instances to be clustered is $s = \sum_{i=1}^{n} |C_i|$. We define precision P as

$$P = \frac{1}{s} \sum_{i=1}^{n} P_i = \frac{1}{s} \sum_{i=1}^{n} \max(|C_i \cap T_1|, |C_i \cap T_2|, ..., |C_i \cap T_m|) \tag{2.1}$$

and recall R as

$$R = \frac{1}{s} \sum_{i=1}^{m} R_i = \frac{1}{s} \sum_{i=1}^{m} \max(|C_1 \cap T_i|, |C_2 \cap T_i|, ..., |C_n \cap T_i|) \tag{2.2}$$

A high precision translates to clusters that separate well between instances of the two classes, i.e., clusters tend to only contain instances of one class. In addition, a high recall reflects that a high number of instances of one class are aggregated into one cluster. Figure 2.2.1 shows examples for the combinations of precision and recall, given a dataset with instances of two classes A (dots) and B (squares). Clusters are represented by dotted ellipses. Figure 2.2.1 a) reflects both, a low precision ($P = 0.5$, $R = 0.2$) because instances of distinct classes are not split into different clusters, as well as low recall since each cluster contains one instance per class at most. In case of Figure 2.2.1 b), the recall is high ($R = 1$) because all instances per class are grouped into the same cluster. However, precision is low ($P = 0.5$) because there is still no distinction between instances of distinct classes. Vice versa, Figure 2.2.1 c) indicates low recall ($R = 0.4$) and high precision ($P = 1$). Clearly, the best clustering performance is displayed in Figure 2.2.1 d) which exhibits high precision ($P = 1$) and high recall ($R = 1$). In this case, the clustering results reproduce the structure of the dataset as given by the class labels.

Throughout this thesis, we strive for a high precision, because instances of

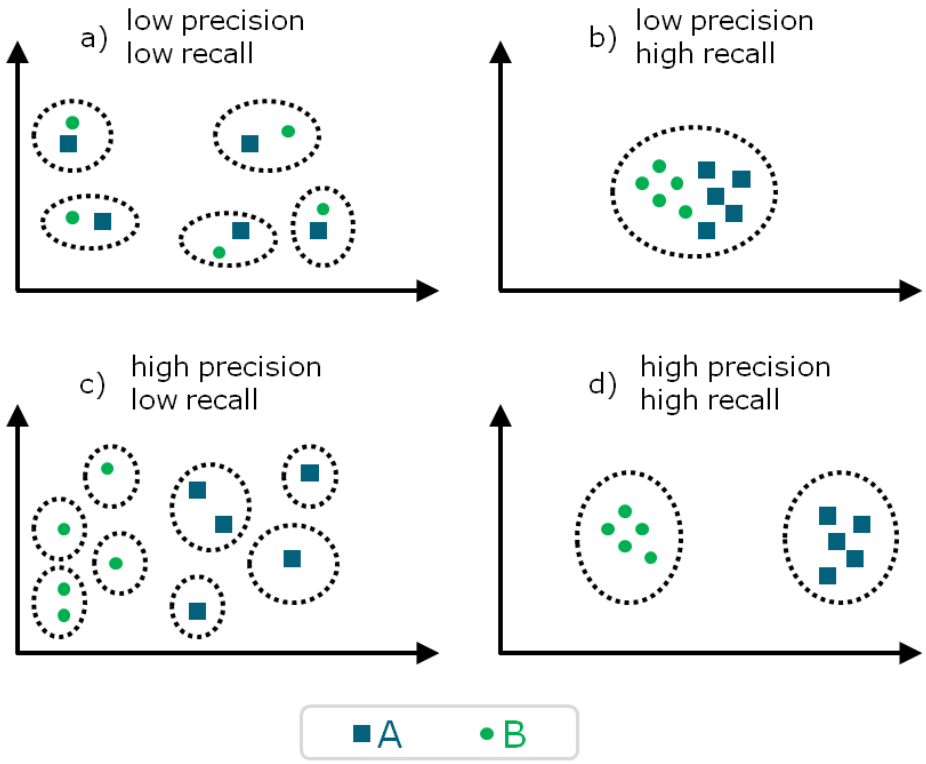


Figure 2.3: Examples for clustering results measured using precision and recall. Dotted ellipses symbolize the resulting clusters.

distinct families should be filed into different clusters. However, in practice, precision and recall form a trade-off. With higher precision, recall decreases and vice versa. In this case, we slightly favor a high precision over a high recall. In other words, under certain circumstances, it is tolerable, if the instances of one class spread over more than one cluster. One such exemplary reason could be that the clustering results in more fine-grained clusters than the resolution of the class labels reveal. For example, imagine to cluster a dataset of fruit, covering APPLES and CITRUS FRUIT. While the class labels may only provide the two fruit species APPLES and CITRUS FRUIT, the clustering might even distinguish the sort of fruit among each of the classes, e.g., *lime*, *orange* and *lemon* among CITRUS FRUIT as well as *Granny Smith* and *Cox Orange* among APPLES.

In face of clustering evaluation, we need to be able to tolerate multiple clusters for one class, but have to avoid too generic clusters by mixing different classes into the same cluster. One way to deal with this requirement is to combine precision and recall in a weighted score and prioritize precision over recall. Formally, we use the *F-measure* [vR79] to evaluate the performance of a clustering with threshold th and a weighting parameter β, with $\beta < 1$ reflecting higher weight on precision over recall:

$$\text{F-measure}_{th} \;=\; (1+\beta^2) \;\cdot\; \frac{P_{th} \;\cdot\; R_{th}}{\beta^2 \;\cdot\; P_{th} + R_{th}} \tag{2.3}$$

We will refer to F-measure in each of the subsequent chapters when dealing with clustering evaluation and provide a reasonable value for the parameter β depending on the context.

2.2.2 Classification Evaluation

While for clustering evaluation we apply precision and recall, for classification, we evaluate using false negative and false positive rates. An instance which by ground truth is assigned to class A but where the classifier erroneously predicts it not to be of class A is considered a false negative. Vice versa, a false positive occurs if an instance is not considered to be of class A (by ground truth), but the classifier incorrectly assigns it to class A.

Especially for supervised machine learning approaches, there is a general trade-off between memorization and generalization. Whereas memorization will perfectly reproduce the classification results on instances that were used during training, it will fail for yet unknown instances. In the machine learning context, this pitfall is called overfitting. In contrast, too broad of a generalization will result in less accuracy of the classification – a phenomenon typically referred to as underfitting. Ideally, we strive for the right balance between underfitting and overfitting. This methodology is called structural risk minimization and has initially been proposed by Vapnik [Vap95].

One way of measuring the fitness of a derived model and estimate its accuracy on an independent dataset is provided by cross-validation. Cross-validation

deals with a repeated classification and evaluation on different subsets of a given dataset. In this thesis, we turn to k-fold cross-validation [Sto74] which works as follows. The training and validation datasets are split into k subsets. Then, $k-1$ subsets are used for the training phase, while the remaining subset is used for the validation. This process is repeated until each subset has been used once as validation subset. The mean of the resulting false positive and false negatives rates can help to estimate the performance of a given classifier on an independent dataset.

2.3 Summary

In this chapter, we introduced the foundations of machine learning techniques as well as the required concepts and definitions on malicious remote-controlled malware. Used throughout the remainder of this thesis, these concepts form basic blocks for our detection methodologies.

In the following chapters, we will design and implement detection approaches in order to identify and recognize botnet command and control channels as well as visual monetization techniques.

Chapter 3

Malware Analysis using Sandnet

3.1 Introduction

Often, analyzing malicious software is not a straight-forward process. Since malware authors strive to evade detection, they employ a variety of means to hinder the analysis and the detection of their malicious binaries. As an example, the static analysis of PE binaries is typically hindered by custom compression and obfuscation techniques of the PE binary, an approach referred to as packing [GFC08, MCJ07, DRSL08]. Only during runtime, the PE binary unpacks itself in memory. As understanding the (un)packing algorithm of a certain malware binary is tedious, researchers turned towards executing the binary and analyzing the binary during its execution. This approach is typically referred to as dynamic malware analysis [WHF07].

In order to study the malware phenomenon, we designed a contained dynamic malware analysis environment, called Sandnet [RDB+11]. The name Sandnet is derived from "sandbox", referring to a contained execution environment for malware without harming the outside world, and "network" due to our special regard to malicious network traffic. Sandnet focuses on dynamic malware analysis, i.e., the automated execution of Windows PE binaries. Therefore, Sandnet captures the network traffic emitted during the malware binary execution and records the graphics output by taking screenshots. Furthermore, we enhanced Sandnet by means of static analysis of PE binaries as well as an extensive post-processing analysis, including payload-based protocol detection and parsers for well-known application layer protocols. The output of the post-processing is stored in a PostgreSQL database.

As a result, the Sandnet database constitutes a dataset of malicious PE binaries as well as their execution results including the network traffic and screenshots. This dataset serves as the basis for our subsequent research on detecting malicious remote-controlled software. Depending on the dedicated goal of an experiment, it is usually required to compile a subset of certain kinds of samples. For example,

as some binaries are acquired from public submission, it is not guaranteed that all binaries are indeed malicious. We have observed cases where legitimate software, such as the default Windows Notepad program have been submitted. Thus, we will discuss how the dataset for each experiment is compiled before performing the actual clustering or classification experiment. One possible way of compiling a reliable dataset for an experiment is achieved by manually inspecting (a random subset of) the samples to be included. In order to make the manual review of samples as efficient and comfortable as possible for a human analyst, we extended SANDNET with a web interface. Section 3.4 describes one example of how we designed a web interface view on the network traffic emitted by an executed binary. Throughout this chapter, we will describe the design of SANDNET and, where appropriate, provide statistics based on the resulting SANDNET data.

3.2 Sandnet Architecture

Every day, thousands of malware binaries hit the collection sensors of researchers and antivirus companies. To a certain degree, the amount of malware samples makes dynamic malware analysis a scalability problem. Thus, we designed SANDNET in a distributed, scalable manner. Figure 3.1 shows the overall SANDNET architecture.

Each binary is executed in a pre-configured virtual machine, called *Sandpuppet*, running Windows XP 32bit with Service Pack 3. Additionally, a sandpuppet has typical applications installed, such as browsers, Flash plugins, PDF readers and media players as well as language packs for foreign languages including Chinese, Japanese and Russian. Having these additional language packs installed allows us to run malware which requires support for foreign languages. The host computers for the virtual machines are termed *herders*. In this thesis, all experiments are based on malware execution using the Windows XP SP3 32bit sandpuppet.

In order to limit potential harm to others while running malware, we take care in filtering out harmful traffic. Therefore, we deploy containment policies that redirect harmful traffic, e.g., spam and infections, to local honeypots. Furthermore, we limit the number of concurrent connections and the network bandwidth to mitigate denial of service activities. An in-path network intrusion detection system (honeywall) watches for security breaches during our experiments. The filtering component operates on a per-sandpuppet basis and transparently segments the sandpuppet-local network on the data link layer. By this means, the communication between two sandpuppets can be regulated. Since malware strives to propagate, it often spreads autonomously. If communication between sandpuppets was not regulated, the binary executed in one sandpuppet might infect another sandpuppet, influencing the results of both executions. By containment policy, we can still allow communication among sandpuppets, if required in an experiment. However, usually we prevent such communication among sandpup-

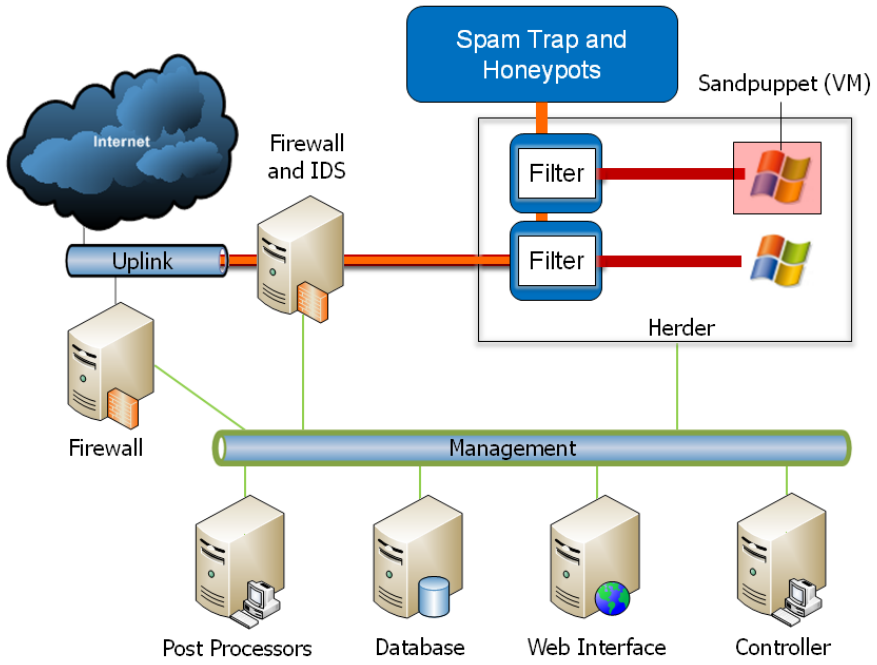


Figure 3.1: Sandnet Architecture

pets. We consider the separation of network segments per sandpuppet a very important property in order to prevent unwanted influence on execution results.

Furthermore, the fact that filtering the network traffic takes place on each herder, instead of a single entity, allows for better scalability. Each herder provides its sandpuppets with filtering instances. Thus, given enough uplink capacity, in order to increase the processing throughput of SANDNET, herders can be added. While in practice, several herders use the same spam trap and honeypots, the SANDNET architecture also allows for several spam traps and honeypots, e.g., for load balancing. However, during our SANDNET operation we never saturated the capacity of the service simulation, spam trap or honeypot farm. Similarly, the uplink capacity was never saturated because we force each sandpuppet to the maximum bandwidth and packet rate of a typical dialup DSL line.

Apart from the execution environment which consists of a set of herders, SANDNET provides management components that take care of acquiring new binaries, scheduling binary executions and post-processing the raw results of the binary executions. Based on the set of malware binaries, the SANDNET controller schedules binaries for execution in the execution queue which then serves as source for the sandpuppets. Usually, scheduling considers the age of a sample, making sure that recent binaries are executed. This is an important aspect in order to compile a set of active remote-controlled malware executions, because the older a binary, the less likely it is to be able to reach its command and control peers. Furthermore, we acquire antivirus labels for the binaries by help of VirusTotal. For the five antivirus vendors Microsoft, Symantec, Kaspersky, Avira and Eset, we parse the returned labels into type, platform, family name and variant. This way, for example, we can measure the diversity of binaries in terms of distinct families and – if needed for an experiment – schedule a diverse set of binaries. Especially in face of polymorphism, we can mitigate the prevalence of a few families by help of distributing scheduled binaries over several distinct antivirus family labels.

Once a binary has been executed, the post-processors dissect the network traffic of the execution, extract and transform the raw execution results into structured and easily processable form by storing the results in an object-relational PostgreSQL database. In addition, the results are enriched by correlating with external sources, such as origin AS and routing paths from BGP feeds, geolocalization of involved IP addresses as well as reverse DNS information. For subsequent analyses, the SANDNET database is thus the primary source of input data.

3.3 Segmentation and Dissection of Network Traffic

In order to extract features from network traffic, it is essential to develop data structures which provide access to the syntactic fields of the involved network protocols. However, it is often difficult to determine the level of detail required to

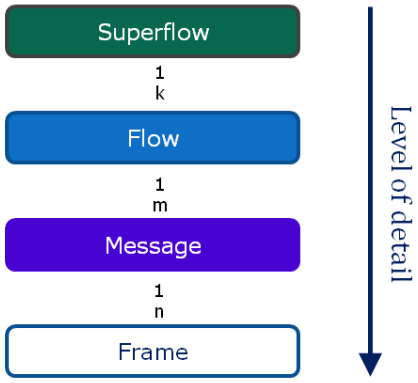


Figure 3.2: Abstract representations of network traffic as superflows, flows, messages and frames

achieve the goal of an experiment in advance. On the one hand, a high resolution in the parsed data structures provides a fine-grained access to all fields of a specific network protocol. On the other hand, today's networks have high bandwidths and large data volumes which makes analyzing network traffic as a whole in such environments infeasible. As a result, we are forced to restrict ourselves to data structures that provide an abstract view on the network traffic. Thus, with performance and efficiency in mind, it is advantageous to parse only as few structures as required. Where applicable, we therefore focus intentionally on a representation of network traffic where only a very small fraction of the whole traffic is distilled.

For SANDNET network traffic, we decided to reassemble TCP and UDP streams. Moreover, we developed parsers for the application layer protocols DNS, HTTP, SMTP, IRC, FTP and TLS. The parsers have intentionally been developed by hand so that syntax errors can be detected in detail and handled in a custom fashion. The DNS parsing results are fed to a passive DNS database. For all streams other than DNS, we assign the domain name that was used to resolve the destination IP address to the stream. This is useful in order to compare the domain name, for example, to the Host-Header in HTTP or the server name of the Server Name Indication extension in TLS.

Furthermore, we develop heuristics for the segmentation of unknown application layer protocols into messages. In general, we designed a data model for TCP- and UDP-based network traffic and its dissected protocol information, providing three layers of abstraction, namely *superflows*, *flows* and *messages*. Figure 3.2 shows the relationship between the different levels of abstraction of network traffic.

A *flow* represents the notion of a communication channel between two entities in terms of one network connection. It is uniquely identified by the 5-tuple:

transport layer protocol *l4p* (TCP or UDP), source s_{ip} and destination IP address d_{ip} as well as source port s_p and destination port d_p. Additionally, a flow contains contextual information such as start t_s and end times t_e and comprises the list of computed flow properties, such as the number of messages and the results of payload-based protocol determination. Formally, we define a flow f

$$f := \langle\ l4p, s_{ip}, d_{ip}, s_p, d_p, t_s, t_e\ \langle properties\ \rangle\ \rangle$$

Flows directed to the same destination can be aggregated into *superflows*. In this case, we do not include the source port in the unique identifier of a superflow, because in this context, we want a flow to be able to span multiple streams to the same destination IP address and port with different source ports. As an example, several HTTP connections between the same source and destination can have varying source ports whereas the transport layer protocol, the source and destination addresses as well as the destination port stay the same. A superflow is uniquely identified by the following 4-tuple (neglecting the source port): transport layer protocol *l4p* (TCP or UDP), source s_{ip} and destination IP address d_{ip} as well as destination port d_p. Formally, we define a superflow f_s

$$f_s := \langle\ l4p, s_{ip}, d_{ip}, d_p, t_s, t_e\ \langle properties\ \rangle\ \rangle$$

Just as a flow, a superflow is enhanced by a list of properties, depending on the exact experiment's use case.

Inspired by the fact that lots of application layer network protocols are designed in a dialogue-like fashion, e.g., pairs of request and response such as HTTP, SMTP and DNS, we heuristically split the packet payload of a flow into *messages* as follows. We define a message to be composed of all consecutive flow payload transmitted in one direction until the direction of the payload transmission changes, an inactivity timeout of t minutes is hit (typically with $t = 5$) or a new stream is opened. The 5-minute-timeout stems from the fact that the network egress router of the contained environment has a stream idle timeout of 5 minutes. Frames without payload, especially those carrying only signaling information such as TCP acknowledgements are ignored because they do not contribute to the message payload. Additionally, we encountered flows which do not follow the request-response rhythm, mostly because several requests are sent before a response is received. We treat such cases depending on whether the application layer protocol is known or not. If the application layer protocol is known, we try to detect if duplicate messages were sent and if so, remove the duplicates. However, if no duplicates were detected, but the protocol consists of request-response cycles and requests and responses are interleaved, we restore the request-response-cycle based on the parsed protocol information. For example, if an HTTP flow consists of two subsequent requests before any of the two responses is transmitted, we re-order the messages so that two request-response

cycles emerge. If the application layer protocol is unknown, we keep multiple subsequent messages going in the same direction.

Formally, a message m_f of the flow f is thus defined as

$$m_f := \langle\ dir, len, t_s, t_e, payload,\ \langle properties\ \rangle\ \ \rangle$$

where *dir* denotes the direction in which the message was transmitted, e.g., source to destination or vice versa, *len* is the length of the message in bytes, t_s and t_e are timestamps of the message start and end, and *payload* comprises the message's payload. In case of HTTP, a message is extended by dissected protocol-specific fields, namely the request URI and the request and response bodies. We use our custom HTTP parser to extract these fields for all streams recognized as HTTP by OpenDPI [ipo11].

Analogously, a message m_{f_s} of a superflow f_s is thus defined as

$$m_{f_s} := \langle\ dir, len, t_s, t_e, payload,\ \langle properties\ \rangle\ \ \rangle$$

In order to apply machine learning to network traffic, we need to define and extract features. The definition of the feature extraction process is presented in each of the subsequent chapters, depending on the goal of the experiment. However, the data structures for network traffic, defined in this chapter play an important in subsequent work.

3.4 Visualization

In order to evaluate our experiments, ground truth labels need to be assigned to the instances of a given dataset. Therefore, to inspect the execution results of a binary, we designed a web interface on the execution results of SANDNET. This section shows two important views of the web interface which were used throughout the subsequent experiments in this thesis.

Figure 3.3 shows the network flows over time per execution. The x-axis correlates to the relative time since the start of the execution of the binary with alternating background column coloring every five minutes. On the y-axis, starting from the top, the destinations of a superflow (or flow, respectively) are displayed in terms of IP address or domain name, destination port as well as the country code of the geolocalization result of the destination IP address. Additionally the amount of traffic transmitted in this superflow is shown. Colored bars symbolize superflows with colors denoting the application layer protocol as given by payload-based protocol detection. For example, the red bars in Figure 3.3 correlate to two IRC superflows, blue denotes superflows with DNS, green symbolizes HTTP traffic and fuchsia relates to HTTP traffic where an executable binary has been downloaded. The border color of the bars provide additional information such as whether a known C&C protocol was detected and labeled in the

superflow.

While the clustering and classification experiments rely on programmatically inferred features, the web interface assists in building a ground truth for these experiments. For example, in the recognition of command and control channels as part of *CoCoSpot*, we derived labels by inspecting executions of binaries using this web interface view in order to identify candidate command and control flows.

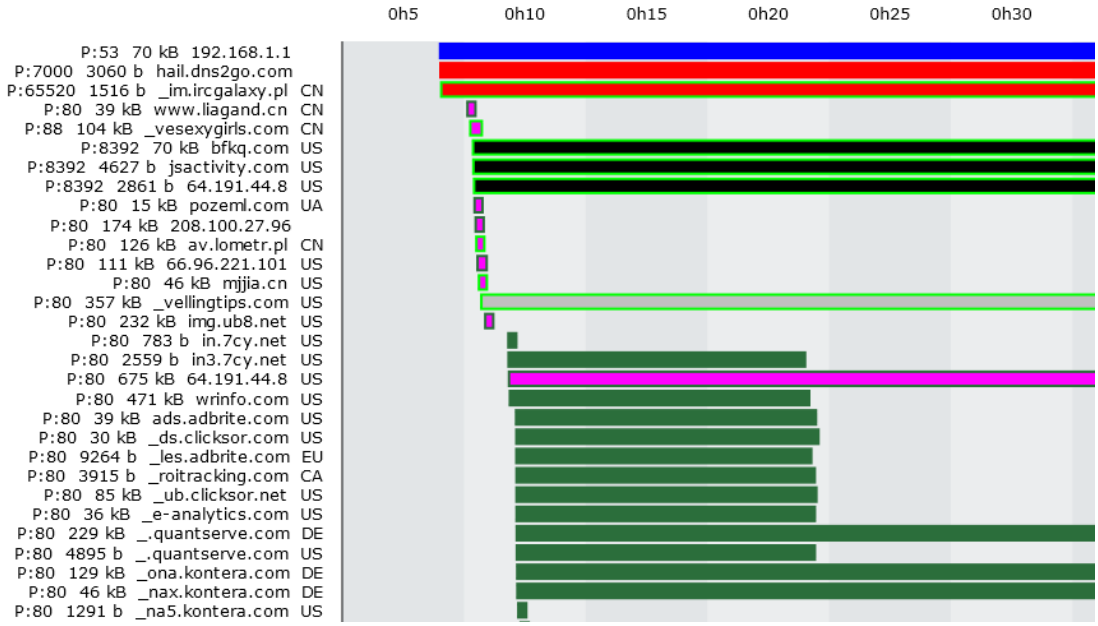


Figure 3.3: Sandnet web interface's superflows view, x-axis shows the time since the binary is launched, y-axis shows superflow destination endpoints

Once candidate (super)flows are identified, the analyst switches to the message view. Figure 3.4 displays the message view of a flow. If C&C communication is in plain text, this view typically reveals the command and control instructions. For example, in the first message in Figure 3.4, the bot reports that it runs on Windows XP Service Pack 3. In return, as can be seen in the second message, the C&C server instructs the bot to download four additional binaries from the given URLs. These binaries will subsequently be executed.

In case of encrypted C&C channels, it is much more difficult to confirm, if a given flow is command and control traffic or not. Some families appear suspicious because the distribution of file types among the HTTP communication exhibits a noticeable skew towards image types, possibly even images of only one file format. Figure 3.5 shows an example for a concealed C&C protocol where the response appears to be a bitmap image file. However, the preview in the pop-up shows that the image file does not constitute a semantically valid image, but rather consists of high-entropy contents depicted as seemingly, randomly distributed pixels. In this case, it is an encrypted binary update camouflaged as a bitmap image. In

Start Time	End Time	entropy	Payload
0:04:04.079778	0:04:04.608415	4.94839	hex6b: 4e49434b2074 NICK tkysqymn USER g0205O1 . . :-Service Pack 3 JOIN &virtu Preview in hex
0:05:07.263907	0:05:40.215677	5.19688	hex6b: 3a752e205052 :u. PRIVMSG tkysqymn :!get http://www.liagand.cn/img/evid.png :u. PRIVMSG tkysqymn :!get http://down0129.iwillhavesexygirls.com:88/erdown.txt :u. PRIVMSG tkysqymn :!get http://pozeml.com/oc/box.txt :u. PRIVMSG tkysqymn :!get http://av.lometr.pl/inst.php?lang=de&id=32&sid=0 PING :i. Preview in hex
0:05:40.216628	0:05:40.491950	4.01136	hex6b: 504f4e47203a PONG :i.JOIN &virtu Preview in hex
0:07:15.491440	0:07:15.491440	3.32193	hex6b: 50494e47203a PING :i. Preview in hex
0:07:15.492804	0:07:15.794451	4.01136	hex6b: 504f4e47203a PONG :i.JOIN &virtu Preview in hex

Figure 3.4: SANDNET web interface's message view shows the messages of a Virut plain variant's C&C flow.

addition, the mismatch of requested file type and response file type, i.e., the fact that the request indicates to retrieve a JPEG image file, but the response indicates a bitmap image file, underlines the suspicion of this HTTP transaction.

Some families strive towards steganographic C&C channels, especially if the command and control traffic is concealed more carefully, such as C&C of the Renos/Artro family, shown in Figure 3.6, where the C&C instructions are hidden in valid images, transmitted via HTTP. In these concealed cases, we then turned to manually reverse engineer the binary in order to judge and develop a decryption routine for its C&C.

If the application layer protocol is detected and we have a parser for the protocol, the message view transparently shows the parsed message contents. In case of an HTTP flow for example, each HTTP request or response is parsed transparently, such that if compression or chunking was used, the message view will instead show the decompressed body. In addition, as shown in Figure 3.6, if an image was transmitted, this image can be shown in a preview pop-up window.

3.5 Dataset Overview and Evaluation

Between February 2010 and November 2012, SANDNET analyzed more than 410,588 MD5-distinct Windows PE binaries, out of a pool of 1,549,841 MD5-distinct binaries. Since some binaries were executed multiple times, e.g., as part of an experiment to monitor the command and control infrastructure, the total

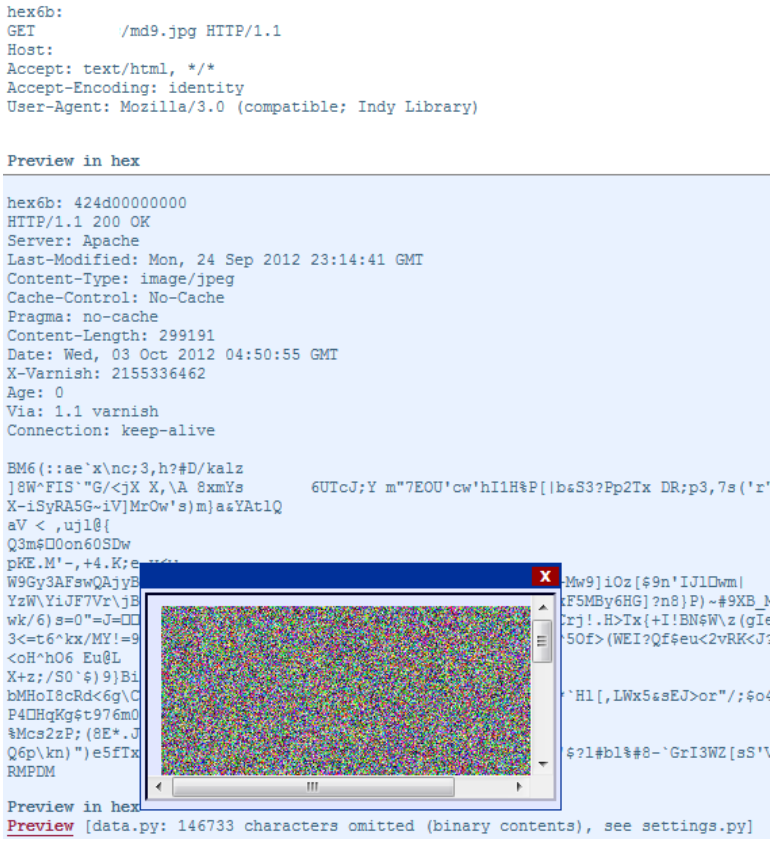


Figure 3.5: SANDNET web interface's message view shows the request and the parsed HTTP response of a C&C message concealed as a bitmap image.

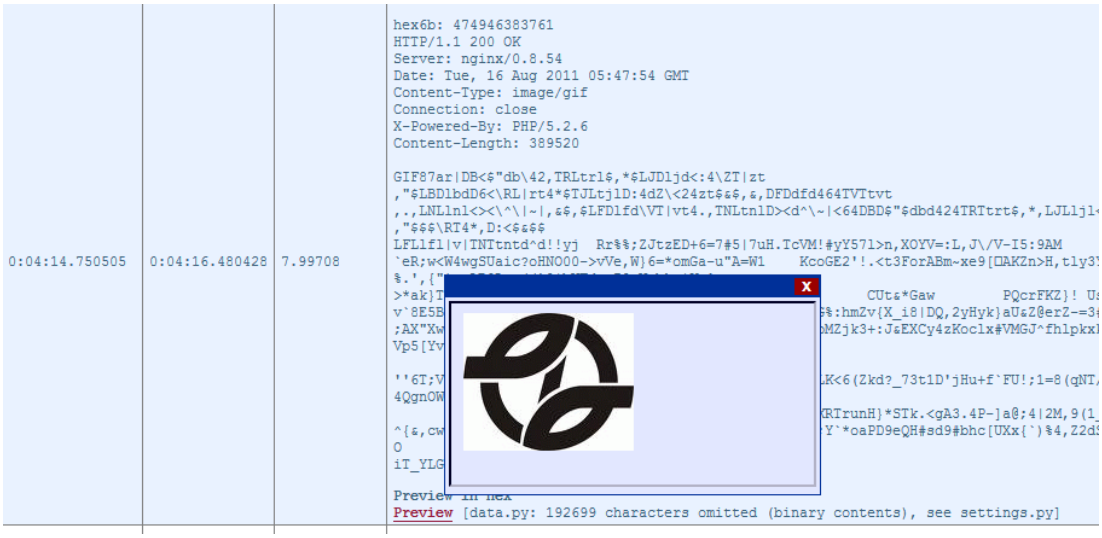


Figure 3.6: SANDNET web interface's message view shows the parsed HTTP response of a C&C message concealed by a GIF image.

number of executions is 520,166, i.e., slightly higher than the number of binaries. The total run time of all executions adds up to more than 34 years. Of the total 1,549,841 MD5-distinct binaries, 74.08% had at least one antivirus label assigned that indicated malware by the time we queried VirusTotal. Thus, we can safely assume that more than the majority of binaries is actually malicious. In total, the executed binaries cover 2858 different families measured by Microsoft A/V labels, 2371 families by Kaspersky A/V labels or 2702 different families by Avira A/V labels. Note that some binaries, although malicious, do not have any A/V labels assigned, possibly because none of the A/V scanners at VirusTotal detected the binary in question.

Malware labels as assigned by antivirus scanners have been used by researchers and analysts in numerous experiments, e.g., as a ground truth for malware detection or clustering approaches. However, because of missing labels or inconsistencies in malware naming – especially by the different vendors – it gets harder and harder to exactly determine which malware we are dealing with. As a result, A/V labels do not always fit well for evaluation purposes. Therefore, we manually developed means to recognize network traffic of certain prevalent bot families as well as decrypt and parse their C&C traffic, effectively tracking the C&C activities of the corresponding botnets.

Furthermore, one focus of this thesis deals with the detection of C&C communication. While the majority of botnets exposes a centralized C&C architecture, our dataset covers botnets with both C&C architectures, centralized as well as peer-to-peer C&C. Among botnets with a centralized C&C architecture, our tracking spans well-known botnet families such as Bredolab [dGFG12], Car-

berp, Cutwail [SGHSV11], Harnig, Koobface [Vil10], Mariposa [SBBD10], Mebroot/Sinowal [SGHSV11], Mega-D/Ozdok [CCG+10], Palevo/Rimecud, Pushdo, Renos/Artro, Rustock [CL07], Sality, Tedroo/Grum and Virut. However, next to these well-known botnets, new families have emerged, e.g., Kuluoz, Feederbot, Morto, and Chebri. In addition, also toolkit-based botnets are included in our dataset, such as Spyeye and Zeus. For peer-to-peer botnets, the dataset spans Zeus P2P, Sality P2P, Miner [Wer11b, PGP12], Hlux/Kelihos [Wer11a] as well as Sirefef/ZeroAccess P2P.

Throughout our analysis period of more than two and a half years, we have seen several botnets come and go. Some botnets have faced dedicated takedowns, such as Rustock, Mega-D and Pushdo, while others cease without further ado. When using malicious network traffic, especially command and control traffic, as a basis for detection experiments, it is advantageous to measure the activity of the C&C channels involved. In general, while we handle a diverse set of binaries, some binaries might already be outdated, i.e., their C&C peers might no longer be reachable. However, we are especially interested in those binaries that actually manage to become part of a botnet and consider the respective executions as *active*. As a result, we strive to compile a set of active executions which cover a diverse set of malware families.

By help of our tracking means, we are able to classify the activity of C&C channels. Figure 3.7 graphs the activity of the top 25 botnets in terms of consecutive C&C activity by family, as seen and covered by our tracking means in Sandnet, as well as public attention. The x-axis reflects the time period since February, 1st, 2010 until October, 31st, 2012, while the y-axis lists well-known botnet families. A star depicts a dedicated takedown action. Note that, in case of Mariposa and Mega-D, the takedown actions have taken place before the beginning of the time period in this graph. The Mariposa takedown has occurred on December 23rd, 2009 [Cor10], and Mega-D has been taken down in November 2009 [Mus09]. In these cases, the stars are placed on the start of the time period in Figure 3.7 in order to visualize the preceding takedown. A thin black line with black markers represents the time period where new binaries are acquired, but none of the binaries exhibit an active C&C channel. A thick fuchsia bar represents the time periods where active C&C communication has been observed, per botnet family. Note that the monitoring time period has been interrupted by two maintenance periods of Sandnet, from mid-February to mid-March 2011 as well as end-May to mid-July 2012. Furthermore, some families exhibit distinct botnets. For example, in case of toolkit-based families such as Zeus and SpyEye, several distinct botnets may be formed. In our C&C activity tracking, we restrict ourselves to whole families instead of individual botnets on purpose. While Figure 3.7 shows only a fraction of the tracked botnets, the total number of families covered by our tracking means is 153. This way, we can make sure to cover a diverse set of bot families without even having to rely on A/V labels.

On the one hand, as can be seen in Figure 3.7, the takedowns of the Bredolab

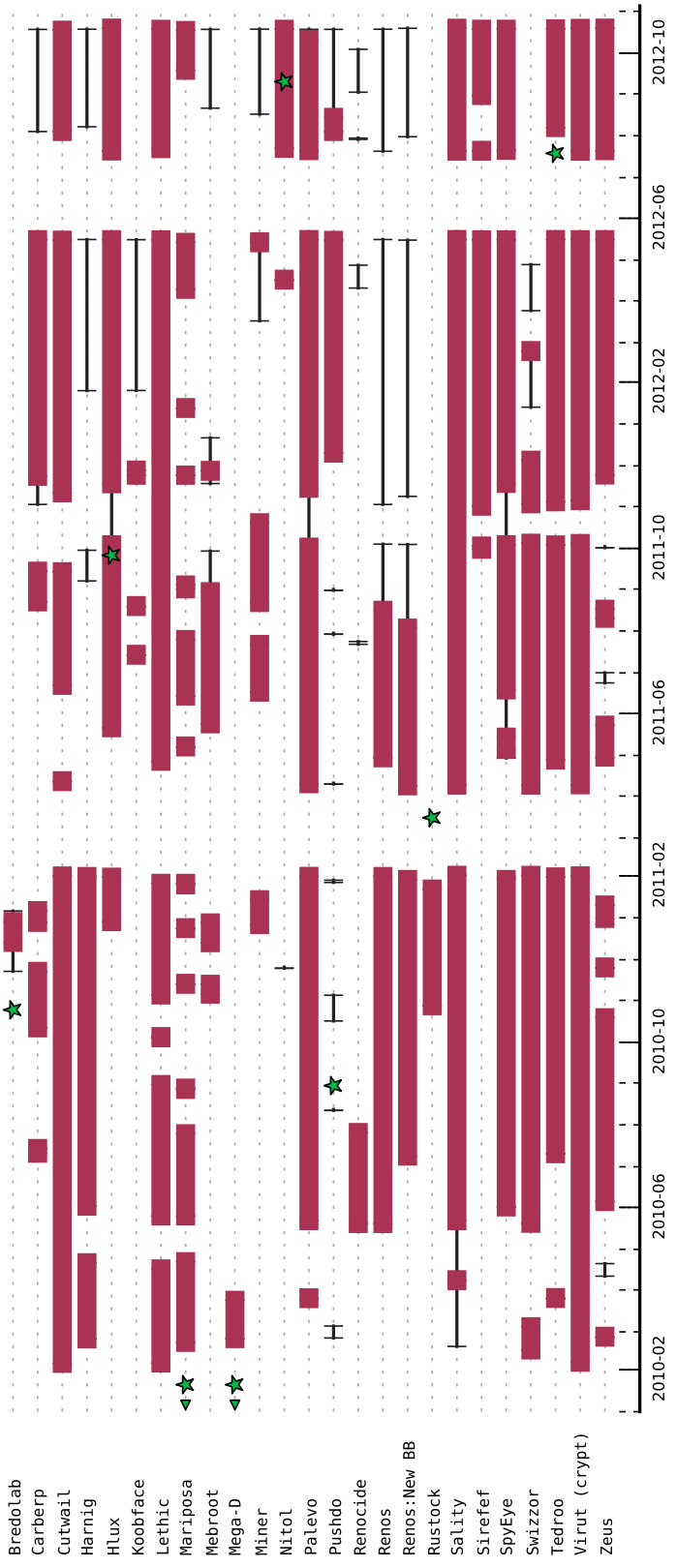


Figure 3.7: Top 25 well-known botnets tracked in SANDNET. A star represents a dedicated takedown action, a thin line represents new binaries being spread and a thick line symbolizes periods of active C&C communication.

[Wil10], Mega-D [Mus09] as well as the Rustock botnet [Wil11] have a long-lasting effect. Although for Bredolab and Mega-D we witness active C&C during up to several months after the takedowns, none of these botnets manage to achieve active C&C communication in the long run. On the other hand, we have seen quite a few botnet families resurrect from takedowns. For example, researchers initiated a takedown of the Pushdo botnet in August 2010. However, even two years after the takedown, we still observe active executions of Pushdo. The Mariposa botnet is believed to have been taken down in December 2009 [Cor10], but we have seen active Mariposa command and control traffic ever since. Similarly, although the Nitol botnets have been taken down in September 2012 [Bos12], many Nitol binaries manage to successfully bootstrap and reach a viable C&C server. In addition, the Tedroo botnet has been addressed in a takedown action on July, 19th, 2012 [Mus12]. However, again, we observed active C&C communication of the same botnet family just within days after the takedown, continuing for at least three months. From an observational point of view, the challenge of botnet family takedowns lies in their sustain.

Some botnets even cease without a dedicated takedown of the C&C infrastructure. For example, the Miner botnet has not been addressed in a dedicated operation of its peer-to-peer-based C&C, but its activity diminished significantly after October 2011. Similarly, the Renos botnet has ceased its operation, possibly after removal signatures addressing the Renos binaries have been distributed by Microsoft as part of its Removal Tool MSRT [Rad11].

Figure 3.8 shows the C&C activity of botnets that expose Fake A/V or Ransomware as monetization technique. Over time, the diversity of rogue visual malware families has increased in prevalence throughout 2011. Of the botnet families shown in Figure 3.8, Ransom and Urausy demand ransom for monetization, while all other families focus on Scareware and Fake A/V. The prevalence of active bot families using rogue visual monetization techniques underlines the importance of this trend. Thus, we will specifically deal with this kind of monetization in Chapter 6.

An interesting question is to what degree the executable binaries per botnet family manage to evade antivirus detection. Therefore, we turn to our C&C tracking means and measure the antivirus detection rates of malware binaries per botnet family. We consider all executions of binaries where the C&C tracking means trigger in the first flow of that execution. We apply this constraint in order to make sure that the binary being executed actually relates to the C&C communication observed. If this constraint had been omitted, and the C&C tracking had triggered on a subsequent flow, it would not have been guaranteed that the C&C communication stems from the executed binary, because additional malware could have been downloaded. The observed C&C communication could then have originated in the additional malware instead of the originally executed binary. In fact, we have observed this behavior, i.e., the downloading of additional malware, in at least 23 distinct bot families. We conclude that these 23 families

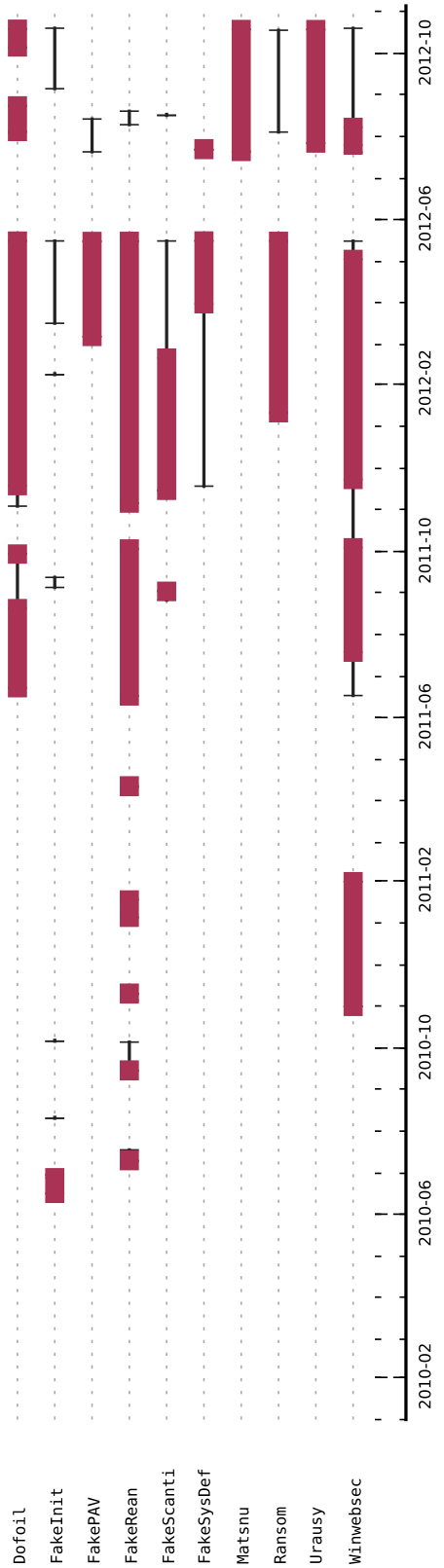


Figure 3.8: C&C activity of botnets exposing Fake A/V or ransomware in SANDNET. A thin line represents new binaries being spread and a thick line symbolizes periods of active C&C communication.

actually act as downloaders. A detailed analysis of this phenomenon is out of scope of this thesis and has been dealt with by Rossow et al. [RDB12].

Table 3.1 shows the results of measuring the antivirus detection of the executed binaries per botnet family. In this table, we included only families with at least 100 distinct binaries showing active C&C communication. Due to inconsistencies in malware naming, we restricted ourselves to only take into account whether the sample was flagged as malicious, and we do not consider the label of the antivirus scan result. The columns two and three measure the number of binaries detected as malicious by at least one A/V vendor among all binaries of that family. While the second column, entitled "Binaries (all vendors)" measures the number of binaries flagged as malicious by at least one out of all 42 vendors included in VirusTotal, the third column, entitled "Binaries (six vendors)" only considers at least one out of the six big antivirus vendors Microsoft, Kaspersky, Eset, Symantec, McAfee and TrendMicro. The results show that detection rates vary. For some families up to 15% of the binaries that showed active C&C communication have not been detected as malicious, while for other families not a single binary has been missed in antivirus detection.

While the second and third columns measure the unified detection impression per botnet family, the fourth and fifth columns aim at the detection coverage among vendors. The "Vendors" columns show the average ratio of antivirus vendors that flagged binaries, per family. The detection coverage among vendors varies, too. On average, for example, a binary of the Zeus P2P family is detected by less than three out of the six big antivirus vendors.

These results show that when labeling Sandnet executions, antivirus labels of all vendors combined typically cover up to 95% of all binaries per botnet family. However, only a fraction of vendors detect each binary of a given botnet family. Thus, on end user systems where typically only one antivirus scanner is in use, we can assume that by far not all of the binaries per family are detected.

As some binaries stem from public sample submission systems, the Sandnet dataset also spans benign binaries. By help of repeated manual inspection of a random subset of the executions, we occasionally found the following benign software in our dataset and labeled the respective executions: Norton Download Manager, Symantec Criteria Checker, Software Informer, Google Updater, PP Live TV as well as installers for various Google products such as Chrome or Google Earth. If required, we excluded these executions from the dataset to be used in a certain experiment.

3.6 Conclusion

During our research, Sandnet turned out to be a very valuable tool and source of data for our malware detection and analysis experiments. The distributed architecture has proven to scale up to 500,000 executions. In the Sandnet dataset

	Binaries (%)		Vendors (%)	
Botnet	**all vendors**	**six vendors**	**all vendors**	**six vendors**
Carberp	97.02	91.49	47.48	51.13
Cutwail	97.96	93.88	62.39	59.86
Harnig	97.54	99.65	54.78	61.29
Hlux	92.04	87.08	39.07	47.88
Lethic	92.60	97.45	63.43	66.16
Mariposa	100.00	100.00	88.71	79.78
Mebroot	100.00	93.85	53.30	58.21
Palevo	98.75	98.42	73.96	73.41
Pushdo	100.00	85.71	68.37	64.29
Renocide	100.00	92.77	59.44	53.61
Renos	99.93	97.99	87.91	77.48
Renos:New BB	99.84	100.00	93.27	82.22
Sality	98.78	99.78	84.85	80.22
Sirefef	97.69	90.95	41.80	48.30
SpyEye	98.79	93.77	55.19	59.52
Swizzor	100.00	98.53	62.07	56.33
Tedroo	100.00	94.74	53.48	51.97
Virut (crypt)	98.21	100.00	83.10	80.82
Zeus	91.27	93.01	45.00	52.33
Zeus P2P	88.55	91.67	33.39	44.92

Table 3.1: Antivirus detection of PE binaries per botnet family for families with more than 100 MD5-distinct binaries. All values in percent, either using all 42 scanners or only the top six.

evaluation we have shown how we made sure that the resulting dataset satisfies our requirements of spanning *diverse* and *active* remote-controlled malware. We maximize sample diversity by pooling binaries from several distinct malware families, while, at the same time, considering the C&C activity in order to guarantee that our dataset reflects well-functioning malware.

As a result, all of the subsequent experiments in this thesis are based on subsets of data acquired by means of Sandnet. In addition, Sandnet provides a human analyst with detailed information on the execution of a PE binary, enabling both, quickly gathering a first impression of a binary as well as deeply analyzing all network communication. However, in many cases, manual efforts were additionally involved in order to get an understanding of what a certain malware execution actually aims at, for example in terms of monetization.

To sum up, in this chapter we provide an overview of Sandnet, our contained dynamic malware analysis environment, which is used to compile a dataset of malware executions. In the following chapters, we will show how we develop malware detection approaches and evaluate using the Sandnet data.

Chapter 4

Recognition of Command and Control Flows

4.1 Introduction and Problem Statement

A defining characteristic of a bot is its ability to be remote-controlled by way of command and control (C&C). Typically, a bot receives commands from its master, performs tasks and reports back on the execution results. All communication between a C&C server and a bot is performed using a specific C&C protocol over a certain C&C channel. Consequently, in order to instruct and control their bots, botmasters – knowingly or not – have to define and use a certain command and control protocol. The C&C protocol is thus considered a bot-inherent property.

Listing 4.1: Rbot IRC-based command and control message

```
1 <:DEU|00|XP|SP3|L|440514 MODE DEU|00|XP|SP3|L|440514 :+iB
2 <:DEU|00|XP|SP3|L|440514!ciddumj@XXX JOIN :##sodoma_3
3 <:DEU|00|XP|SP3|L|440514 ##sodoma_3 :.root.start dcom135 200 5 0 -b -r -s
4 <:DEU|00|XP|SP3|L|440514 ##sodoma_3 drake 1260838540
5 <:DEU|00|XP|SP3|L|440514 @ ##sodoma_3
6 <:DEU|00|XP|SP3|L|440514 @KrOwN @drake
7 <:DEU|00|XP|SP3|L|440514 ##sodoma_3 :End of /NAMES list.
8
9 >PRIVMSG ##sodoma_3e :nzm (tftp.plg) transfer to 88.43.117.44  beginning, info:
      (C:\WINDOWS\system32\upds.exe).
```

Historically, bots used cleartext C&C protocols, such as plaintext messages transmitted using IRC or HTTP. For example, Listing 4.1 shows C&C messages sent from the C&C server to a bot, instructing the bot to start its DCOM vulnerability scanning module on destination TCP port 135 (line 3). The parameters

tell the bot to launch 200 scanning threads in parallel, while each scan operation has a timeout of 5 seconds. Scanning shall continue until either told to stop or all addresses in the same /16 IPv4 subnet have been iterated. The results shall not be posted in a channel, but instead be sent via a private message (PRIVMSG). Once, the bot finds a vulnerable target, it infects the victim and the victim connects back in order to receive the second-stage binary via TFTP. The botmaster is notified about this event via PRIVMSG in line 9. Note that the actual infection in the example as well as the downloading of second-stage binaries is transparently redirected to local honeypot systems, effectively avoiding a true infection of third parties on the Internet.

Listing 4.2: HTTP-based command and control message transmitting stolen email credentials in plaintext

```
1  POST /blog/user_id_upload.php HTTP/1.1
2  Connection: keep-alive
3  Content-Length: 196
4  Host: 2agohuxoiyr.ru
5  Accept: text/html, */*
6  Accept-Charset: UTF-8
7  Accept-Encoding: identity
8  User-Agent: Mozilla/3.0 (compatible; Indy Library)
9
10 Array
11 (
12     [email] => alina@XXXXX
13     [password] => sonne123
14     [user_name] => SYSTEM
15     [comp_name] => WORKSTATION
16     [id] => S-6788F32F-4467-4885
17     [lang_id] => 1031
18     [product] => COL
19 )
```

Similarly, Listing 4.2 depicts the plaintext C&C message of a trojan that steals credentials and reports general information of the infected system. In this case, the email address and the corresponding password have been transmitted in addition to the computer's name as well as the username that the bot runs as. However, a C&C channel relying on a plaintext protocol can be detected reliably. Methods such as payload byte signatures as shown by Rieck et al. [RSL+10] or heuristics on common C&C message elements such as IRC nicknames as proposed by Goebel and Holz in a system called Rishi [GH07] are examples for such detection techniques. To evade payload-based detection, botnets have evolved and often employ C&C protocols with obfuscated or encrypted messages as is the case with Waledac [CDB09], Zeus [BOB+10], Hlux [Wer11a], TDSS [GR10],

Virut [RDB12] and Feederbot [DRF+11], to name but a few. The change towards encrypted or obfuscated C&C messages effectively prevents detection approaches that rely on plaintext C&C message contents. For example, Listing 4.3 shows the hexdump of an encrypted Virut C&C message as sent from the bot to the C&C server. The C&C message contents is XOR-encrypted with a random four-byte key. Thus, the message exhibits no characteristic payload byte pattern. Once decrypted, as shown in Listing 4.4, the plaintext message reveals that the underlying protocol is still IRC or IRC-like. The four-byte session key can be derived by a known plaintext attack on the first four bytes of the first C&C message, which turns out to always be the string `NICK`.

Listing 4.3: An encrypted Virut command and control message, transmitted in two frames

```
0000  61 E3 B4 A1 27 31 0E 67   53 B2 AE 04 2C D7 B0 2F    a...'1.g  S...,../
0010  3D 42 74 18                                         =Bt.

0014  5C ED C6 17 EB 4F BA 3A   35 91 CB A3 FE 12 FB AD    \....O.:  5.......
0024  57 32 85 01 FE 3C E0 AB   87 72 6D 5E F9 F0 F3 30    W2...<..  .rm^...0
0034  4D A0 CD CB A8 D2 6F DE   A6 B0 29 F1 D5 7B 87       M.....o.  ..)..{.
```

Listing 4.4: The decrypted Virut command and control message of Listing 4.3

```
0000  4E 49 43 4B 20 68 79 7A   68 74 73 77 6D 0A 55 53    NICK hyz  htswm.US
0010  45 52 20 6B                                         ER k

0014  30 32 30 35 30 31 20 2E   20 2E 20 3A 23 36 63 31    020501 .   . :#6c1
0024  30 64 62 61 65 36 20 53   65 72 76 69 63 65 20 50    0dbae6 S  ervice P
0034  61 63 6B 20 33 0A 4A 4F   49 4E 20 23 2E 30 0A       ack 3.JO  IN #.0.
```

Furthermore, previous work on detecting botnet C&C channels targeted towards blacklisting the communication endpoints in terms of IP addresses and domain names that were used to locate the C&C server. This lead to a number of blacklists for C&C servers, such as [abu11a, Lis09, abu11b]. However, botnets have adapted to this approach and migrate their C&C servers from one domain to another, in order to avoid the blacklisted addresses. For example, while tracking the Belanit botnet as part of our study of downloaders [RDB12], we noticed that the domain names for the main C&C server migrate from one top level domain to another, as can be seen from Figure 4.1. Between December 2011 and January 2012, the domains were registered with `.com`, later with `.info` and subsequently with `.ru`.

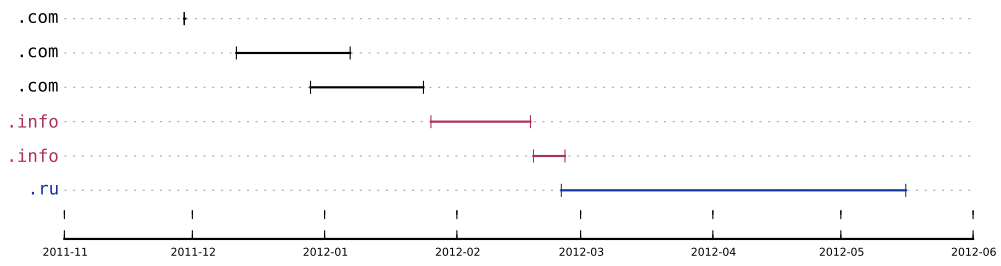


Figure 4.1: Migration of the Belanit C&C server's domain from one top level domain to another

A similar trend can be observed for the Emit botnet, shown in Figure 4.2. In this case, the botmaster migrated its C&C server domain names from `.com` via `.org`, `.pl`, `.ua` to `.us` in the period from June 2011 to April 2012.

However, the migration is not only limited to the DNS domain names. We witness the same countermeasure concerning the IP addresses of the C&C servers as well as the Autonomous System (AS) that announces the routing for the IP address origin. Figure 4.3 shows how several distinct Autonomous Systems have been used to announce the C&C servers of the Vobfus/Changeup botnet. Again, it is clearly visible that different ASes have been used over time, exhibiting the nomadic character of today's botnets.

To sum up, once a change of C&C server address is noticed, blacklists need to react quickly in order to block communication with C&C servers of centralized botnets. In addition, blacklist approaches do not work with botnets that employ a distributed C&C architecture, such as peer-to-peer botnets.

In this thesis, we take a different approach to recognize C&C channels of botnets and fingerprint botnet C&C channels based on traffic analysis properties. The rationale behind our methodology is that for a variety of botnets, characteristics of their C&C protocol manifest in the C&C communication behavior. For this reason, our recognition approach is solely based on traffic analysis.

As an example, consider a C&C protocol that defines a specific handshake – e.g., for mutual authentication – to be performed in the beginning of each C&C connection. Let each request and response exchanged during this imaginary handshake procedure conform to a predefined structure and length, which in turn leads to a characteristic sequence of message lengths. In fact, we found that in the context of botnet C&C, the sequence of message lengths is a well-working example for traffic analysis features. For example, let us consider the two prevalent botnet families Virut and Palevo[1]. Table 4.1 shows the sequence of the first 8 messages

[1] A synonym for the malware family Palevo is Rimecud (Microsoft terminology) or Pilleuz (Symantec terminology).

Figure 4.2: Migration of the Emit C&C server's domain from one top level domain to another [RDB12]

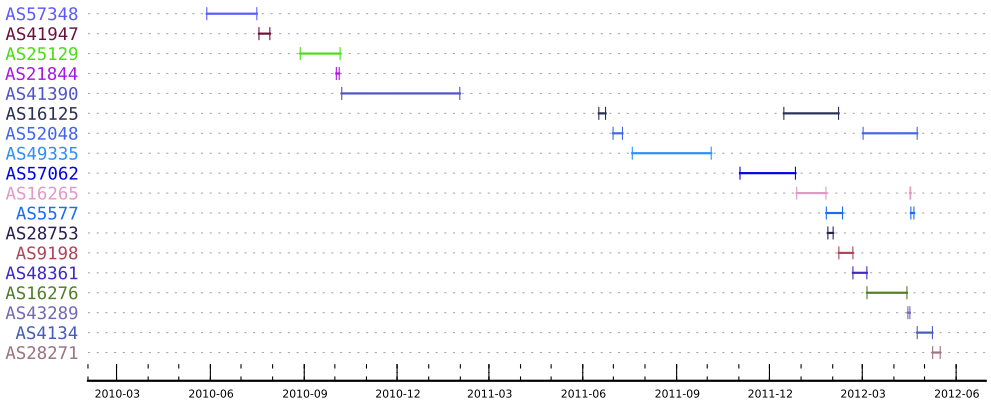


Figure 4.3: Migration of the Vobfus/Changeup C&C server from one origin Autonomous System to another

in four Virut C&C flows and two Palevo C&C flows. Whereas Virut exhibits similar message lengths for the first message (in the range 60-69) and a typical sequence of message lengths at positions five to eight, for Palevo, the first three message lengths provide a characteristic fingerprint.

ID	Family	Message length sequence							
		1	2	3	4	5	6	7	8
1	Virut	60	328	12	132	9	10	9	10
2	Virut	69	248	69	10	9	10	9	10
3	Virut	68	588	9	10	9	10	9	10
4	Virut	67	260	9	10	9	10	9	10
5	Palevo	21	21	30	197	32	10	23	10
6	Palevo	21	21	30	283	21	10	23	10

Table 4.1: Examples of message length sequences for Virut and Palevo C&C flows

Leveraging statistical protocol analysis and hierarchical clustering analysis, we develop *CoCoSpot*, a method to group similar botnet C&C channels and derive fingerprints of C&C channels based on the message length sequence, the underlying carrier protocol and encoding properties. The name *CoCoSpot* is derived from spotting command and control. Furthermore, we design a classifier that is able to recognize known C&C channels in network traffic of contained malware execution environments, such as Sandnet.

The ability to recognize botnet C&C channels serves several purposes. A bot(net)'s C&C channel is a botnet's weakest link [FHW05]. Disrupting the C&C channel renders a bot(net) ineffective. Thus, it is of high interest to develop methods that can reliably recognize botnet C&C channels. Furthermore, driven by insights of our analysis of botnet network traffic, we found that a bot's command and control protocol serves as a fingerprint for a whole bot family. Whereas for example properties of the PE binary change due to polymorphism, we witness that the C&C protocol and the corresponding communication behavior seldom undergo substantial modifications throughout the lifetime of a botnet. From an analyst's perspective, our classifier helps to detect and aggregate similar C&C channels, reducing the amount of manually inspected traffic.

To summarize, the contributions of this chapter are two-fold:

- We provide a clustering method to analyze relationships between botnet C&C flows.
- We present *CoCoSpot*, a novel approach to recognize botnet command and control channels solely based on traffic analysis features, namely carrier protocol distinction, message length sequences and encoding differences.

The remainder of this chapter is structured as follows. Section 4.2 sheds light on related work, defines the scope of this chapter and highlights innovative aspects

of our approach. Subsequently, in Section 4.3, the general methodology as well as the feature space is described. While Section 4.4 deals with the clustering phase of C&C flows, Section 4.5 outlines the classifier which is then used to classify unknown flows. In order to evaluate our approach, as described in Section 4.6, we classified arbitrary network flows emitted from our dynamic malware analysis environment SANDNET as either C&C or Non-C&C and verified the results using two datasets. Finally, we discuss limitations of our approach in Section 4.7 and conclude in Section 4.8.

4.2 Related Work

Traditionally, botnet C&C channels have mainly been identified in two ways. First, publicly available blacklists [abu11a, Spa11, Goo11, Phi11] provide lists of known botnet servers by IP addresses or domain names. The drawback of blacklists is that properties like IP addresses or domains are volatile. Botmasters can and do change these often, rendering detection methods based on blacklists infeasible. In addition, botnets that rely on a peer-to-peer C&C architecture exhibit quickly changing sets of rendez-vous points. Some botmasters design their bot's bootstrap process even more resilient by avoiding any static rendez-vous coordinates, e.g., by using domain generation algorithms where the current rendez-vous point is valid for a very limited time span such as a few hours. These volatile communication endpoints can hardly be grasped by blacklists. Second, botnet C&C channels can be detected by checking for characteristic payload substrings. For example, Botzilla [RSL+10], Rishi [GH07] and rules for the Snort IDS [Thr11] identify C&C channels in network traffic using payload byte signatures for a small set of known botnets. However, most encrypted or obfuscated C&C protocols do not exhibit characteristic payload patterns and undermine existing payload byte signatures.

Consequently, the traditional techniques are unsatisfying and have motivated research for automated and more reliable processes. In that trail of research, BotMiner [GPZL08] and BotGrep [NMH+10] provide approaches to use traffic analysis in order to find botnet C&C channels. However, while BotMiner requires detectable so-called A-plane activity such as spam, DDoS or scanning, *CoCoSpot* does not require any a priori or accompanying malicious actions and aims at the *recognition* of C&C channels. For *CoCoSpot*, in order to detect a bot, it is enough to just exhibit C&C communication. The graph-based approach of BotGrep requires botnets with distributed C&C architectures in order to detect them. However, *CoCoSpot* not only works with peer-to-peer-based botnets, but also with botnets exhibiting a centralized C&C architecture.

Jacob et al. present JACKSTRAWS [JHKH11], which exploits that certain C&C channels show recognizable system-level behavior in terms of system call traces of Anubis [BKK06]. Particularly, JACKSTRAWS dynamically analyzes

malware binaries (e.g., with Anubis) and models system call graphs for known C&C connections. New C&C channels are detected by matching unknown network connections against these graphs. As opposed to JACKSTRAWS, *CoCoSpot* does not depend on host-level analyses such as tainting, which enables our system to be applied without host access. In addition, we will show that *CoCoSpot* can detect C&C flows of numerous different bot families, while Jacob et al. do not provide insight into the diversity of the C&C channels detected by JACKSTRAWS. For example, while pure *download-and-execute* C&C channels follow strict system call patterns, more complex C&C channels spanning multiple TCP/UDP connections (e.g., typical for most modern P2P- or HTTP-based botnets) may not be detected by JACKSTRAWS. In contrast, *CoCoSpot* detects different architectural and many semantic types of C&C channels.

Concurrent to our work, Bilge proposed DISCLOSURE [BBR+12], a system to detect C&C servers in unknown network traffic based on NetFlow data. Bilge frames a number of features that characterize botnet C&C servers, which are partially similar to ours, but are applied in a different context. While we use periodicity of messages to select C&C candidates, Bilge uses periodicity of network connections towards a server as an indicator for C&C channels. Similarly, we rely on the sequence of C&C message lengths, while DISCLOSURE bases on sequences of C&C stream size lengths. In both scenarios, these features seem to work well, while the constraints are different: DISCLOSURE is bound to the strict NetFlow format, while we can rely on much finer granularity (messages vs. connections) in our setting and deploy a wider flow format. Another advantage of *CoCoSpot* over DISCLOSURE is that we have fewer assumptions that narrow the type of C&C channels. For example, DISCLOSURE models network traffic in a client-server fashion and favors centralized C&C channel architectures and thus it will presumably fail for P2P-based botnets. Moreover, *CoCoSpot* does not require *a priori* knowledge on reputation of autonomous systems, so that *CoCoSpot* even recognizes C&C servers in well-reputable networks.

In general, our approach complements existing C&C detection systems, in that we propose a technique to also recognize the *nature* (i.e., malware family) of C&C channels. While the difference may sound subtle, we see a major contribution here. In many cases, it is desirable to know which type of botnet accounts for a detected C&C channel, e.g., to automate classification of malware [SGKA+09]. In addition, especially compared to IP address (and domain) blacklists *CoCoSpot* provides a finer granularity in that it restricts the detection to specific flows, which turns out useful if legitimate and malicious activity appear on the same IP address (or domain). Moreover, our system is able to produce human-readable reports for detected C&C channels, making an analysis easy. Currently, *CoCoSpot* reports back the type of C&C channel found and provides the security analyst with examples of similar C&C channels in the training dataset.

In summary, the automatic recognition of C&C channels is a challenge and demands for a different approach. We fill this gap by designing a method to

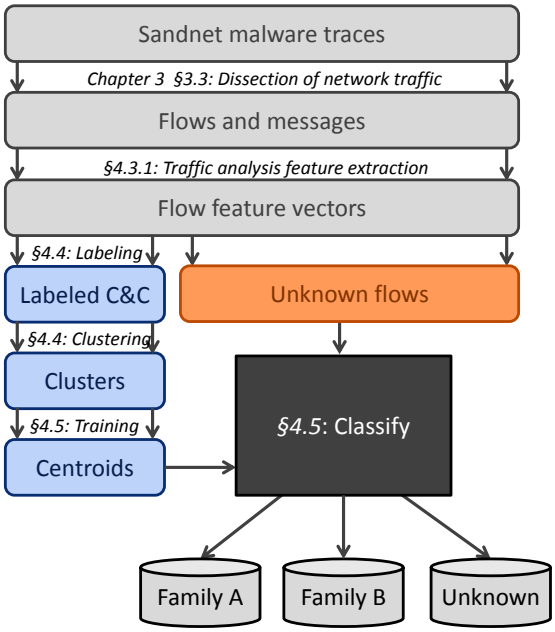


Figure 4.4: Overview of the C&C flow classification methodology

recognize known C&C channels based on traffic analysis while not relying on specific payload contents nor IP addresses or domain names.

4.3 Methodology

A coarse-grained overview of our methodology is shown in Figure 4.4. First, we dissect and aggregate TCP and UDP network traffic according to our network traffic data model. This process is described in Section 3.3 of Chapter 3. Based on this model, we design features that measure traffic analysis properties of network communication and extract these features from a set of manually analyzed C&C flows (Section 4.3.1). Using hierarchical clustering, we compile clusters of related C&C flows, and manually verify and label these C&C flows (Section 4.4). For each cluster, our method derives a centroid (Section 4.5.1) which is subsequently used during the classification of C&C candidate or even completely unknown flows of a contained execution environment such as Sandnet (Section 4.5.2).

4.3.1 Traffic Analysis Features of Botnet C&C

Based on our data model, we define features aiming to measure traffic analysis properties of network communication, especially in the context of botnets. These features will be used for the clustering analysis and the subsequent classification of flows.

Our feature space consists of three features. First, we consider the carrier protocol to be the underlying protocol of the C&C protocol and distinguish between TCP, UDP and HTTP. We assume that a C&C protocol is either designed to be used with TCP, UDP or HTTP. In fact, we have not witnessed a change of carrier protocol during the lifetime of a botnet in our analysis of botnet traffic.

As the second feature, we define the sequence of message lengths in a flow. This feature is motivated by the observation that most C&C protocols exhibit a characteristic sequence of message lengths. More precisely, we exploit the characteristic message length sequence during the beginning of a flow. We assume that the first few messages of a flow cover what could be regarded a handshake phase of the C&C protocol. In addition, we observed that C&C flows exhibit characteristic message lengths during idle phases, i.e., when only keep-alive-like messages are transmitted. We decide to take up to the first eight messages into consideration because on the one hand they cover the initial handshake and – if there is an early idle phase – the first few messages of an idle phase, too. On the other hand, eight messages is reasonably small in order to keep the computational overhead low. We evaluated different numbers of messages in a flow and while smaller numbers make the computational performance decrease, we did not experience huge improvement with longer message length sequences.

In case of TCP and UDP as carrier protocol, the length of a message is defined as the total length of the carrier protocol's payload in bytes. For HTTP, we define the message length of an HTTP request to be the sum of the body length as well as the URI query section, and for a response the body length. We omit HTTP headers because they do not contain relevant information for our method.

The third feature is specific to HTTP and counts the number of distinct byte values in the query section of the HTTP request URI, aggregated over up to the first four HTTP requests' URIs of a flow. We observed that malware authors decide for different encoding schemes such as Base64, Hex or ASCII when designing an HTTP-based C&C protocol. To a certain degree, the number of distinct bytes reflect the encoding scheme of the URI's query section. For example, for a Base64 encoded query section of the URI, the number of distinct bytes does not exceed 64. Note that we restrict ourselves to the query section of the request URIs for all requests.

Throughout one botnet family, it is possible that there are multiple C&C protocols with possibly even different C&C architectures in place, for several reasons. Malware authors might allow for a backup C&C protocol that is only activated if the primary C&C protocol fails, several variants of one bot family might operate

in parallel using two distinct C&C protocols or the bot has been designed to work with two (or more) C&C channels. As an example, we observed the latter case with New_BB, an egg downloaded by Renos/Artro which exhibits two HTTP C&C channels with two different servers. Another example is Virut where some variants exhibit a plaintext IRC-based C&C channel while other more well-known variants use a TCP-based custom-encrypted C&C protocol.

4.4 Clustering Analysis of Known C&C Flows

Clustering enables us to identify and aggregate groups of similar C&C flows in our data set. We use clustering for two main reasons. Often, the message lengths of the messages in a C&C flow are not equally static throughout several C&C flows of one botnet, but show slight deviations in a small range. Thus, we need to aggregate similar flows and learn the range of each message's length in a C&C flow. Second, grouping similar C&C flows into clusters allows us to build a centroid for each cluster which represents the fingerprint of this cluster's C&C flows. The clustering step produces efficient representations that serve as training data for the subsequent classification of flows. In addition, the clustering results provide insights into and measure the relationships between clusters of different malware families.

4.4.1 Definition of the Distance Function

Using the features described in Section 4.3.1, we define the feature vector $v(f)$ of a flow f, called *flow vector*, as:

$$v(f) \;=\; \langle\; p, ml_1, ml_2, ml_3, ..., ml_n,\; hb \;\rangle$$

where $v.p$ denotes the carrier protocol TCP, UDP or HTTP, $v.ml_k$ denotes the length of the k-th message in the flow f, and hb is the number of distinct bytes in the query section of an HTTP request URI if the flow has HTTP as carrier protocol. We use $n = 8$, i.e., up to the first eight messages per flow, as described in Section 4.3.1. Based on the resulting feature space, we define the following distance function $d(u, v)$ of the feature vectors u and v of two C&C flows:

$$d(u,v) \;=\; \frac{1}{T}\, d_p(u,v) \;+\; \frac{1}{T}\, d_{ml}(u,v) \;+\; \frac{1}{T}\, d_{hb}(u,v) \tag{4.1}$$

where

$$T \;=\; \begin{cases} 3, & u.p = \texttt{http} \wedge v.p = \texttt{http} \\ 2, & \text{else} \end{cases} \tag{4.2}$$

$$d_p(u,v) \;=\; \begin{cases} 0, & u.p = v.p \\ 1, & \text{else} \end{cases} \tag{4.3}$$

and with k being the minimum number of messages in the two flow vectors u and v:

$$d_{ml}(u,v) = \frac{1}{k}\sum_{i=1}^{k}\left(\frac{|u.ml_i - v.ml_i|}{\max(u.ml_i, v.ml_i)}\right) \tag{4.4}$$

$$d_{hb}(u,v) = \begin{cases} \frac{|u.hb - v.hb|}{\max(u.hb, v.hb)} & u.p = \texttt{http} \wedge v.p = \texttt{http} \\ 0, & \text{else} \end{cases} \tag{4.5}$$

In the distance function d, all feature distance terms d_p, d_{ml} and d_{hb} weigh equally. If both flows are HTTP flows, then the three features p, ml and hb are each weighted with 1/3, otherwise the two features p and ml are each weighted with 1/2. The main intention of introducing weights in Equation 4.1 is to limit the range of output values to $[0, 1]$. While in general, weights can also be used to fine-tune the distance computation, we decide to keep the equal weights on purpose. Fine-tuning requires a representative evaluation dataset and if applied aggressively, fine-tuning inevitably leads to overfitting. In our case, using broad evaluation datasets, we will show that using the distance function with equally weighted feature terms yields very low misclassification rates. When dealing with a very specific application or dataset, fine-tuning the weights might lead to a performance increase.

By definition, our distance function results in values between 0.0 (equal flows) and 1.0 (completely different flows). Table 4.2 consists of four flow vectors of the Virut family and two Palevo flow vectors and will be used to illustrate the distance computation. All Virut C&C flows have TCP as carrier protocol, Palevo flows have UDP as carrier protocol. The distance between the first two Virut flow vectors in Table 4.2 (IDs 1 and 2) is 0.0885. When looking at the first Virut flow vector (ID 1) and the first Palevo flow vector (ID 5), their distance is 0.4934.

ID	**Family**	**Carrier protocol**	**Message length sequence**							
			1	2	3	4	5	6	7	8
1	Virut	TCP	60	328	12	132	9	10	9	10
2	Virut	TCP	69	248	69	10	9	10	9	10
3	Virut	TCP	68	588	9	10	9	10	9	10
4	Virut	TCP	67	260	9	10	9	10	9	10
5	Palevo	UDP	21	21	30	197	32	10	23	10
6	Palevo	UDP	21	21	30	283	21	10	23	10

Table 4.2: Message length sequences for Virut and Palevo C&C flows (Table 4.1 extended with Carrier Protocol)

Even from this small subset of C&C flow vectors, it becomes obvious that the message length at a certain position is more characteristic than others. In our case, the messages at message position 2 of the Virut flows have varying lengths between 248 and 588 bytes whereas the messages at the first position vary in

ID	# Flows	Description
F_{all}	34,258,534	SANDNET flows, mixed C&C and Non-C&C
F_{Cand}	23,162	C&C candidate flows
F_C	1,137	Manually verified C&C flows

Table 4.3: Data sets of flows

length between 60 and 69. We will take this into consideration when computing the cluster centroids and explain this in detail in Section 4.5.1.

4.4.2 Dataset Overview

We compiled several datasets in order to apply and verify our C&C flow recognition methodology. As a first step, aiming to get C&C candidate flows, we apply several heuristics to a set of 34,258,534 SANDNET flows, F_{all}, and consult the Amada C&C server blacklist in order to find C&C flow candidates. The flows in F_{all} are gained from SANDNET, the contained malware execution environment presented in Chapter 3, and stem from 34,387 malware binaries of 1930 distinct families according to Microsoft antivirus labels. We conducted all malware execution experiments using the Windows XP SP3 32bit sandpuppets connected to the Internet via NAT, as described in Chapter 3. While harmful traffic such as spam and infections are redirected to simulated service endpoints, honeypots and spam traps, other protocols (e.g., IRC, DNS or HTTP) were allowed in order to enable C&C communication. The biases affecting the following experiments due to containment should thus remain limited. We did not deploy user interaction during our experiments. The flows cover a time span from February 2010 to December 2011.

As heuristics to detect C&C flow candidates, we applied periodicity detection, long-lasting flow detection as well as domain flux detection to the flows in F_{all}. Note that while these heuristics certainly do not guarantee to find all C&C flows, we use them to find a bootstrapping set of C&C flows. The resulting set of candidate C&C flows is denoted as F_{Cand}. In order to detect periodic messages, we compute the time between any two subsequent messages in a flow, the so-called message gap interval in seconds, and the relative frequencies of these message gap intervals. The message gap intervals are computed separately for requests and responses, and rounded to integer precision. A flow is considered periodic with period p if one message gap interval has a relative frequency of r_1 or two adjacent intervals add up to a cumulative relative frequency of r_2, in at least one direction. We evaluated r_1 between 45% and 60%, r_2 between 75% and 90% at a step size of 5, and finally chose $r_1 = 50\%$ and $r_2 = 80\%$.

We considered a flow as long-lasting if its duration is greater than half of the time that a malware sample was running in SANDNET. In addition, we heuris-

tically detect a sample that performs a domain generation algorithm (DGA) by looking at the ratio of distinct DNS queries that result in NXDOMAIN to successful DNS responses for a sliding window time span of m minutes. Once the ratio exceeds g – i.e., more than g DNS queries result in NXDOMAIN in m minutes – we consider the presence of a DGA. In our case, we set $g = 100$ and $m = 5$. Furthermore, we used flows to IP addresses and domains of the Amada blocklist [abu11a] as C&C candidates. In addition, we were provided with network traces of recently active botnets including Hlux [Wer11a] and Miner [Wer11b] with known C&C flows. We added these known C&C flows to the set F_{Cand}.

Of the C&C candidate flows, we manually reviewed and classified 2,691 flows as C&C and – if possible – assigned malware family labels. The resulting set of verified C&C flows is denoted as F_C.

At this stage, F_C contains 2,691 C&C flows of 43 bot families. This set is skewed, i.e., a few bot families cause the majority of C&C flows. We mitigate the imbalance in the set F_C by limiting the number of flows per family to a maximum of 50 and require a minimum of 10 flows per family. Finally, the filtered set contains 1,137 C&C flows of 43 distinct families, including e.g., Hlux/Kelihos [Wer11a], Koobface [Vil10, TN10], Mariposa [SBBD10], Miner [PGP12, Wer11b], Palevo/Rimecud, Renos/Artro, Sality P2P [Fal11], Sirefef/ZeroAccess, SpyEye [SEB12], Torpig [SCC+09], Virut and Zeus [BOB+10].

4.4.3 Hierarchical Clustering

We apply agglomerative hierarchical clustering to group the set of manually verified C&C channels F_C. In order to avoid the so-called chaining phenomenon, where the minimum distance between any two instances of adjacent clusters could cause the clusters to be merged, we decided to use average linkage hierarchical clustering, as the latter does not tend to fall for the chaining phenomenon. The clustering is performed 2-fold using two disjunct subsets to avoid overfitting, i.e., we split the dataset F_C into two halves, and cluster once using each half. Eventually, to group C&C flow vectors, a cut-off threshold determines the maximum distance for which two different flows still belong to the same C&C group (i.e., cluster). To illustrate, we again refer to the first two flow vectors in Table 4.2 (IDs 1 and 2) and their distance of 0.0885. If the cut-off threshold was greater than this distance, both instances would be aggregated in the same cluster. Otherwise they would be filed into different clusters. Finally, the most prevalent family per cluster in terms of number of flows determines the cluster's family. However, for some clusters we could not infer the malware family, because antivirus scanners did not detect the binaries at all, only heuristic or generic labels applied or labeling was inconsistent among antivirus scanners. In these cases, we tried to manually assign a malware family based on the antivirus scanning results and the IP addresses and domains used during C&C communication. Figure 4.5 provides an example dendrogram that visualizes the clustering results for one

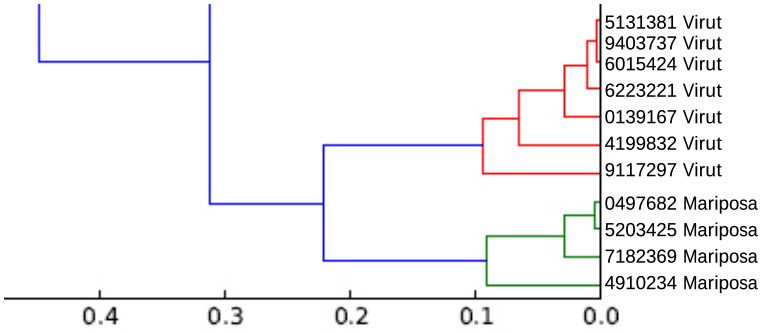


Figure 4.5: Example extract of a dendrogram that visualizes the clustering results, covering one Virut cluster and one Mariposa cluster and a cut-off threshold of 0.115.

Virut and one Mariposa cluster.

4.4.4 Cluster Evaluation

We evaluate our clustering by checking if the clustering results correspond to the labels that we assigned to our training dataset. Ideally, for each family label (i.e., C&C protocol identifier) we assigned to the dataset, a cluster with no more than all elements with this label should be created. For that, in a first step, we have to choose a clustering cut-off threshold that results in the best-possible clustering result. To evaluate how well our clustering represents the labeled dataset, we use the two measures precision and recall, as introduced in Chapter 2.2.1. First, we measure the *precision* as of Equation 2.1, which represents how well our clustering separates C&C flows of different types. Ideally, all instances of one cluster should be of the same kind of C&C flow. Second, we compute the *recall* as of Equation 2.2, which expresses if similar C&C flows are grouped into the same cluster. In general, we strive for a cluster that contains all C&C flows of one kind. However, recall is not as straight-forward as precision and will be described later in this section in more detail.

Note that we defined the overall precision and overall recall so that the precision and recall of all C&C types are equally taken into account. This mitigates imbalances that would otherwise have been introduced by skewed datasets with disparate numbers of elements per C&C type.

We aim for a clustering that both groups C&C flows of one type into one cluster (maximum recall), and that also separates between different C&C flows (maximum precision). At the later stage, a low precision translates to more false positives, while a low recall causes false negatives. In our setting, to avoid false positives, we can tolerate multiple clusters for one C&C type, but have to avoid too generic clusters by mixing different C&C types. We therefore combine

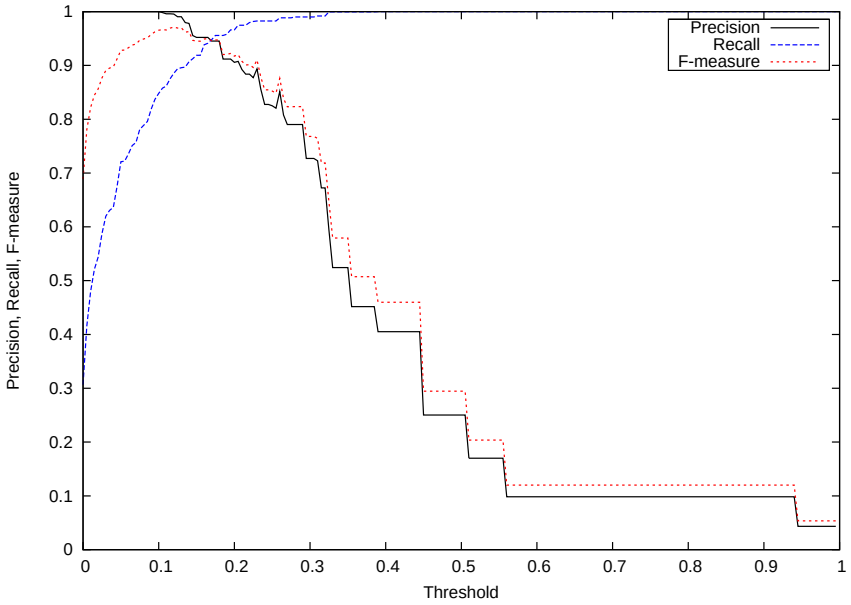


Figure 4.6: F-measure evaluation of the hierarchical clustering at different cut-off thresholds.

precision and recall in the weighted F-measure and prioritize precision twice as important as recall. Formally, we use the *F-measure* as introduced in Chapter 2.2.1 to evaluate the performance of our clustering with threshold th and $\beta = 1/2$:

$$\text{F-measure}_{th} = 1.25 \cdot \frac{P_{th} \cdot R_{th}}{0.25 \cdot P_{th} + R_{th}} \tag{4.6}$$

We then maximize this function to find the optimal clustering threshold th by iterating the cut-off-threshold in the range 0.0 to 1.0 using a step size of 0.005, as shown in Figure 4.6. The result of the F-measure evaluation leads to a cut-off threshold of 0.115.

4.4.5 Clustering results

The clustering phase results in a total number of 91 clusters, of which 13 are singletons, i.e., comprise only one C&C flow training instance. Table 4.5 provides an excerpt of the resulting clusters which includes those clusters that refer to well-known malware families. Some clusters could not reasonably be labeled with a malware family because the samples have not been detected by any antivirus scanner at all (we used VirusTotal scan results), labeling was inconsistent, or only generic and heuristic labels applied. In total, the number of clusters outweighs the number of families in the training data set F_C. This is caused by families which have multiple clusters, such as Virut. We identified three main reasons for

this. First, the bots of a certain malware family evolve over time. For Virut, two kinds of bots circulate in the wild. One variant is based on a plaintext IRC C&C channel, and another variant employs a TCP-based C&C channel which provides an intermediate layer of custom encryption. These two variants exhibit different message length sequences (and message encodings) which is reflected in two distinct Virut clusters. Second, different clusters might represent different activity states. Idle bots, i.e., where the botmaster does not have instructions for the bots, typically exhibit smaller messages (e.g., only keep-alive messages) compared to when orders (e.g., spam templates and address lists) are transmitted. Third, families that do not exhibit characteristic message lengths at all will result in a high number of clusters.

4.5 Designing the C&C Flow Classifier

During the clustering phase, we have built groups of similar C&C flows. In order to extract fingerprints, we now compute a centroid for each cluster and design a nearest-cluster classifier that can classify unknown flows based on these centroids. Finally, we evaluate our classifier on two data sets and give upper bounds of classification errors.

4.5.1 Cluster Centroids as C&C Channel Fingerprints

In the training phase, we compute a centroid z for each cluster C.

The data structure z of a centroid is based on the structure of a flow vector v as defined in Section 4.4, extended by a sequence of weights and a maximum distance. It consists of the following attributes:

$$z = < p, \quad ml_1, ml_2, \ldots, ml_n, \quad w_1, w_2, \ldots, w_n, \quad maxdist, \quad hb >$$

- $z.p$: The carrier protocol.
- $z.ml_i$: The sequence of the average length of the messages at index i of all flows in C.
- $z.w_i$: The sequence of weights for each position i in the average message length sequence $z.ml$.
- $z.maxdist$: The maximum distance between the centroid and all of the flow vectors in the cluster in order to limit the cluster's scope.
- $z.hb$: In case of a centroid for a cluster with at least one HTTP C&C flow: the average number of distinct bytes in the query section of HTTP requests.

We compute the centroid z for each cluster C as follows. First, for each cluster, we load the output of the clustering step, i.e., the set of clustered C&C flow vectors of a cluster C. The carrier protocol of the centroid, $z.p$, is set to the carrier protocol of all flow vectors in C. Note that flow vectors with two different carrier protocols will never be clustered into the same cluster, as the clustering cut-off threshold is smaller than the distance caused by two differing carrier protocols. In other words, one cluster only spans flow vectors of one carrier protocol.

For all message length sequences of the flow vectors in cluster C, we compute $z.ml_i$ by averaging the elements at the i-th position in the message length sequence. Let C contain n flow vectors and let $V(C) = \{v_k\}_{k=1..n}$ be the set of flow vectors of the cluster C. For each flow vector v_k in $V(C)$ and each index i in the message length sequence of v_k, the centroid's sequence of message lengths is computed as follows:

$$z.ml_i(C) \;=\; \frac{1}{n}\sum_{k=1}^{n} v_k.ml_i \tag{4.7}$$

Referring to the example shown in Table 4.2, some message positions of the message sequences can be considered more characteristic for a C&C protocol due to less variation at a specific message position. In order to reflect this in the cluster centroid, we introduce a weighting vector which contains a weight for each message position and indicates the relevance of the message's position. The smaller the variation of the message lengths at a given message position of all flows in a cluster, the higher the relevance of this message position. In other words, if two flow vectors' message lengths differ in a message position with low relevance, the less impact this has on the result of the classification distance function. Thus, we decide to compute the coefficient of variation (c_v) for each message position over all flow vectors in one cluster. The coefficient of variation [Dod06] is defined as the ratio of the standard deviation to the mean and fits our needs. Consequently, we define our weight as one minus the coefficient of variation, in order to reflect that a higher variation leads to a smaller weight.

The weighting vector is computed as:

$$z.w_i(C) \;=\; 1 - \min\left(c_v(v_k.ml_i), 1\right) \;=\; 1 - \min\left(\frac{stddev(v_k.ml_i)}{mean(v_k.ml_i)}, 1\right) \tag{4.8}$$

Table 4.4 shows the flow vectors of a cluster with four C&C flows and the corresponding weights for all message positions. As shown, message positions with varying lengths have a weight value that decreases as the range of message lengths at that position increases.

In order to respect the weight in the distance computation during the classification of a flow vector, we modify the distance function for the message lengths d_{ml} in Equation 4.4 by adding the weight as a factor. The resulting distance

ID	Message length sequence							
	1	2	3	4	5	6	7	8
1	301	2135	305	2169	305	2163	305	2153
2	301	2153	301	2149	305	2153	305	2123
3	301	2125	301	2153	305	2131	305	2157
4	301	2145	301	2155	305	2155	305	2115
	Average message length sequence							
	301	2139.5	302	2156.5	305	2150.5	305	2137
	Weighting sequence							
	1	0.995	0.994	0.997	1	0.994	1	0.991

Table 4.4: Examples for the average message lengths and weighting sequence of a centroid for a cluster of four C&C flows

function $d_{ml,class}$ is defined as:

$$d_{ml,class}(u,v) \;=\; \left(\sum_{i=1}^{k} z.w_i\right)^{-1} \;\cdot\; \sum_{i=1}^{k}\left(z.w_i \;\cdot\; \frac{|u.ml_i - v.ml_i|}{\max(u.ml_i, v.ml_i)}\right) \tag{4.9}$$

The complete distance function that is used during the classification is given as:

$$d_{class}(u,v) \;=\; \frac{1}{T}\, d_p(u,v) \;+\; \frac{1}{T}\, d_{ml,class}(u,v) \;+\; \frac{1}{T}\, d_{hb}(u,v) \tag{4.10}$$

where

$$T \;=\; \begin{cases} 3, & u.p = \texttt{http} \wedge v.p = \texttt{http} \\ 2, & \text{else} \end{cases} \tag{4.11}$$

In addition, we define the *quality indicator* of a cluster to be the normalized sum of all weights in the weighting vector. The quality indicator expresses the overall weight of all positions in a cluster centroid's message length sequence. As an example, if all of the message lengths in the message length sequence vary widely, this will result in a low quality indicator.

$$q(z) \;=\; \frac{1}{k} \;\cdot\; \left(\sum_{i=1}^{k} z.w_i\right) \tag{4.12}$$

The quality indicator is a means to filter clusters that do not represent characteristic message length sequences.

Family Label	*#C*	*#S*	*C&C Arch*	*CP*	*Plain*	*Avg QI*
Bifrose	4	1	centralized	TCP	no	90.48
BlackEnergy	3	0	centralized	HTTP	no	92.71
Cycbot	6	0	centralized	HTTP	no	31.28
Delf	2	1	centralized	HTTP	no	95.33
Koobface	2	1	centralized	HTTP	no	88.89
Mariposa	6	1	centralized	UDP	no	86.41
Mebroot	6	2	centralized	HTTP	no	66.50
Miner	1	0	P2P	HTTP	yes	97.35
Nimnul/Ramnit	1	0	centralized	TCP	no	67.47
Palevo	2	1	centralized	UDP	no	86.28
Renos/Artro	2	0	centralized	HTTP	no	99.87
Renos/Katusha	1	0	centralized	HTTP	no	100.00
Renos:New BB	5	1	centralized	HTTP	yes	84.42
Sality P2P	6	2	P2P	UDP	no	74.31
Sirefef/ZeroAccess	3	0	P2P	TCP	no	80.51
Small	2	1	centralized	HTTP	no	44.50
Spatet/Llac	1	0	centralized	TCP	no	92.59
TDSS/Alureon	1	0	centralized	TCP	no	81.82
Tedroo	1	0	centralized	TCP	no	97.95
Torpig	5	0	centralized	TCP	no	71.55
Virut	3	0	centralized	TCP	mixed	82.01

Table 4.5: Clustering results of some well-known botnet families. #C: number of clusters, #S: number of singleton clusters, C&C Arch denotes the C&C architecture (P2P=peer to peer), CP is the Carrier Protocol, Plain denotes whether the family uses a plaintext C&C protocol encoding; Avg QI is the average quality indicator.

4.5.2 Classification Algorithm

So far, we have developed a data structure for the cluster centroid and a distance function for the classification. The complete algorithm to classify a flow f is as follows. First, we build the corresponding feature vector v_f and compute the distance between all cluster centroids and v_f using the distance function d_{class} in Equation 4.10.

Naively, we could be tempted to assign the closest cluster to the flow f, i.e., the cluster with the minimum distance to v_f. However, the closest cluster is not necessarily correct. Especially in case f is not a C&C flow at all, our classifier requires a means to find out that f lies outside the scope of all clusters. Thus, we additionally store the maximum distance between the centroid and all training flow vectors of a corresponding cluster, computed using the distance function d of Equation 4.1. The maximum distance limits the scope of the cluster, and flows

outside of this scope are discarded by the classifier.

The label of the nearest cluster, i.e., the cluster with the minimum distance between v_f and the cluster centroid, is assigned to f, as long as the distance is below the maximum distance for the centroid. If none of the cluster centroids are in range of f, the flow is considered not to be a known C&C flow. Listing 4.5 contains the simplified Python code involved in the classification of a flow vector, given a set of centroids. For improved readability, error handling code has been omitted.

4.6 Evaluation of the C&C Flow Recognition

In this section we use our classifier to recognize C&C channels among arbitrary network traffic emitted from Sandnet, our contained malware execution environment. Due to the sheer amount of flows to be classified and the absence of rock-solid ground truth for C&C flows, validating the result of the classification is a difficult task. Therefore, we try to estimate upper bounds for the classification errors in order to provide an impression of the performance of our classification approach. We verify our classifier by help of two evaluation datasets, shown in Table 4.6.

In order to estimate the false negative rate, we compiled a set of C&C peers. For each of the 43 families in the training set F_C, we build sets of C&C server IP addresses or domain names based on manually verified C&C flows as well as publicly available C&C server blacklists [abu11d, abu11c, abu11b]. We assume all flows heading for any of the IP addresses or domains in our list of C&C peers as C&C flows. Using these C&C peer lists, we extract all matching flows from Sandnet traffic F_{all}, except those in the training set F_C. For five families, our heuristic did not find additional C&C flows, i.e., all flows of that specific family have already been used in the training phase and are thus excluded from the classification set F_{FN}.

For an initial set of centroids for classification, we discard centroids of four C&C families, as these do not show sufficiently characteristic message lengths. Among these families are SpyEye, Hlux, Zeus P2P and Carberp. Particularly, we discard families with only singleton clusters, or centroids whose average message length weights is smaller than 30%. Our idea of finding a C&C protocol's handshake could be extended to search for representative message length sequences past our current limit of eight messages per flow to mitigate these cases. The resulting data set is denoted as F_{FN} and contains C&C flows for 34 out of 43 families.

In order to estimate the false positive rate, we build a subset of all Sandnet flows F_{all}, balanced by A/V label family so that the subset contains no more than five malware samples of a certain family, determined by Kaspersky antivirus labels.

Listing 4.5: Classification algorithm in Python

```
def classify(f, centroids):
  """
  Classifies flow vector f into the class of the closest
  cluster centroid of the given set of centroids.

  Stores the class and the distance to the centroid in f and
  returns them as a tuple of (class, distance). If no centroid
  matches, class is None and distance is positive infinity.
  """
  # By default, assume no class match and a minimum distance
  # between f and any cluster centroid of positive infinity
  cur_class = None
  min_dist = float("inf")

  # Iterate all cluster centroids
  for z in centroids:
    # Compute the distance between the current cluster centroid z
    # and the flow vector f to be classified
    cur_dist = distance(f, z)
    # If the distance falls within the range of the current centroid
    # and the distance is smaller than any previously computed
    # distance, then store the class of the matching centroid in
    # cur_class and the current distance in min_dist
    if cur_dist <= z.maxdist and cur_dist <= min_dist:
      cur_class = z.get_class()
      min_dist = cur_dist

  # Store the class of the matching cluster centroid in the
  # flow vector instance
  if cur_class is not None:
    f.set_class(cur_class)
    f.set_distance(min_dist)
  return (cur_class, min_dist)
```

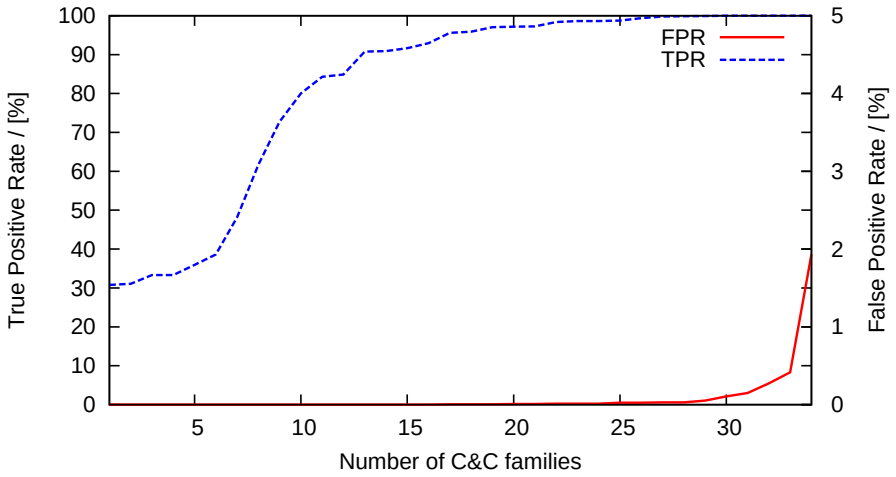


Figure 4.7: Overview of the C&C flow classification performance.

The resulting set of flows to be classified is denoted as F_S, and contains a variety of different application-layer protocols from 3245 different malware samples of 671 distinct families. From this set, we remove all flows destined to any of the IP addresses or domains of the C&C peers, so that the resulting set does not contain any C&C flows of the families known to our classifier.

The classifier is trained on the clustering output of the data set F_C of Section 4.4.4. The training subset contains up to 50 random C&C flows per family. Note that we made sure that none of the flows used during the training phase of the classifier were in either of the sets of flows to be classified.

We define two types of classifications. First, if a C&C flow of F_{FN} actually belongs to family A (based on the C&C peer list), but our classifier assigned a different family (e.g family B) or no family at all, this is considered a *false negative*. Thus, if our classifier assigned the same family as assigned by the C&C peer list, we denote this as *true positive*. Per family, the true positive rate is the ratio of correctly assigned flows over all C&C flows for a specific family. Second, if a Non-C&C flow of F_S is assigned a C&C cluster, we denote this as *false positive.* The false positive rate is the ratio of falsely classified flows for a cluster family over the total number of flows in F_S.

Figure 4.7 shows the results of the classification as a cumulative distribution function for all families that were included in both, true positive and false positive

ID	# Flows	Description
F_S	1,275,422	subset of Sandnet flows, only Non-C&C
F_{FN}	87,655	C&C flows to C&C peers

Table 4.6: Evaluation data sets

analysis. Note that the true positive rate values are given on the left y-axis, the false positive rate values on the right y-axis. More than half of all families have a true positive rate of over 95.6%. A small fraction of seven C&C families had true positives rates lower than 50%, which was caused by too specific cluster centroids. In most of these cases, we had too little training data to learn representative message length variations of a particular C&C protocol, which could be improved by adding more training data for this family.

For 88% of the families, the false positive rate is below 0.1%, and 23 cluster families do not exhibit any false positive at all. For the few families that cause false positives, we observed that the corresponding cluster centroids have high variation coefficients on many message length positions, effectively rendering the centroids being too generic and possibly matching random flow patterns. The C&C family Cycbot shows a maximum false positive rate of 1.93%, while having a true positive rate of 99.92%. At the same time, the average quality indicator of the Cycbot family is 31.28% which is the minimum of all families. The Cycbot family exhibits an HTTP-based C&C protocol and after the clustering, it spreads among six clusters, which we observed due to several different activity states. Four of the six clusters are caused by C&C server errors, such as servers returning errors in the HTTP 500-599 status code range, File not found (status code 404) and even cases where servers returned OK (status code 200), but the response body contained an error message, telling that the requested page could not be found. In summary, erroneous C&C flows in the training data for this family lead to false positives where Non-C&C flows matched which happened to exhibit similar requests and had the same response bodies returned (e.g., for status code 404 File not found or 500 Internal server error). When compared to other families, the low quality indicator for Cycbot shows that, on average, the Cycbot clusters do not tightly represent characteristic behavior.

4.7 Discussion

Whereas the previous chapters presented a method to recognize C&C channels by help of clustering and subsequent classification, this section will shed light on the strengths of our method as well as possible evasions. As is often the case, a detection approach such as ours can be mitigated by adversaries. However, we believe that today, only very few of the recent botnet C&C channels have been designed to mitigate the kind of traffic analysis proposed in this work. We provide answers to the question why our methodology works in recognizing C&C channels and we will outline the limitations of our approach.

4.7.1 Reasoning and Evasion

Why does our methodology work? We assume that, driven by the practical need to evade payload byte signatures, botnet C&C protocols have evolved from plaintext to obfuscated and encrypted message encodings. However, we speculate that in practice, it has so far hardly ever been required to address the evasion of traffic analysis from a botmaster's point of view.

The fact that our approach relies on message length sequences sets a limitation if messages would no longer exhibit characteristic lengths. During the clustering phase, we had to remove four families from the dataset because they do not have characteristic message lengths. These families include Hlux, SpyEye, Zeus P2P and Carberp. However, it is difficult to tell whether the C&C protocols of these families are designed to evade traffic analysis approaches or whether *CoCoSpot* just failed to derive centroids due to the kind of data that was transmitted over the labeled C&C flows. We manually inspected the families where our approach has failed, in order to clarify whether the message lengths are altered on purpose or by coincidence. Only in case of Zeus P2P, using reverse engineering, we found proof in that it adds random message padding to most of its C&C messages before encryption, possibly to evade message-length-based approaches such as *CoCoSpot*. Interestingly, in case of Sality P2P, while message lengths vary, certain C&C messages are not padded at all and do exhibit characteristic message lengths. Thus, even if a subset of the C&C messages exhibits characteristic message lengths, *CoCoSpot* can effectively detect and classify this family. Furthermore, telling from the evaluation results, 34 of the 43 malware families we analyzed, can successfully be detected using *CoCoSpot*. We see our approach as an innovative, additional means of classifying malicious traffic.

In addition, our methodology has some limitations concerning details of the practical implementation. As described in Section 3.3 of Chapter 3, we heuristically split messages of a flow if the direction of transmission changes or a new connection is opened. If several messages are sent in the same direction over the same connection, i.e., the C&C protocol does not follow a dialogue-like pattern, our approach will fail to correctly separate these. Indeed, after manual inspection, we found such cases to cause false negatives, especially with unreliable C&C servers that did not respond to an unusually high number of requests. For example, if a C&C server only responds to every fifth request, five subsequent requests might be aggregated into one request message until a response is received. Thus, the resulting message length sequence differs significantly from what has been learned during the training phase (where the C&C server responded to every request), exhibiting higher values for the request message lengths. However, if the carrier protocol is known, e.g., in case of HTTP, it can be parsed and does not need to be split heuristically. Retransmissions of requests without response could then be ignored and messages can be separated at well-defined boundaries. This could further reduce the possible false classification.

Additionally, care has to be taken during the training phase in order to make sure that representative C&C flows are considered. As the classification results of the Cycbot family show, erroneous C&C flows in the training data might result in high false positive rates. As such, it is important to make sure that the training set is compiled of active C&C channels.

4.7.2 C&C Blacklist Peculiarities and Delusion

In a separate experiment, we evaluated the Kuluoz malware family against a set of C&C signatures of a network intrusion detection system. When inspecting apparent false negatives of our classification results for the Kuluoz botnet C&C in comparison to those signature matches, we stumbled upon a delusion technique of that specific botnet. We observed bots directing requests with the same URI pattern as their C&C to popular legitimate destinations, such as `bing.com` and `twitter.com`, probably aiming to trigger signatures on purpose. Listing 4.6 shows three HTTP requests, where the first stems from an active C&C channel and the last two are delusion requests. Those servers have not been C&C servers and thus responded with an error message. However, the signature matches were based only on the URI request pattern and – erroneously – indicated C&C communication with the respective destinations. Instead, our method performed correctly and did not classify this communication as C&C because the error responses differed significantly from what was derived from the active C&C in the training phase. In fact, this revealed a false positive of the signature. This experience elucidates that our method is able to distinguish active C&C from inactive C&C. Furthermore, automatically compiled blacklists from signature matches will suffer from false positives, if delusions like those employed by Kuluoz occur. In this case, `bing.com` and `twitter.com` could have mistakenly been added to a blacklist due to the signature hits.

We observed a similar delusion with the domain `facebook.com`. In this case, a SpyEye version 1.3 bot directed requests with a very similar pattern as its regular C&C towards `facebook.com`, possibly to poison automatically compiled C&C server blacklists. While we have not found `facebook.com` in any of the C&C server blacklists that were used and monitored throughout our experiments, it shows the potential danger of mistreating legitimate domains as C&C servers. Especially for lesser-known benign domain names, it might not be as obvious as in the above mentioned cases, whether the respective domain acts as a C&C server.

Even worse, we have witnessed botnets that use C&C servers which are located in typical benign environments. We have observed C&C servers that were hosted on Amazon Elastic Compute Cloud EC2 [Ama12]. Blacklisting the IP address of the respective C&C server will inevitably block a variety of other services, which

Listing 4.6: Delusion of Kuluoz C&C communication

```
// This is active C&C communication with a C&C server
GET /forum/index.php?r=gate&id=6c..ae&group=2507rcm&debug=0&ips=192.168.X.Y HTTP/1.1
User-Agent: Mozilla/5.0 (Windows; U; MSIE 9.0; Windows NT 9.0; en-US)
Host: XXXXXXXXXXXXX.ru


// This is a delusion request towards twitter.com
GET /nygul/index.php?r=gate&ac=6c..ae&group=2507rcm&debug=0&ips=192.168.X.Y HTTP/1.1
User-Agent: Mozilla/5.0 (Windows; U; MSIE 9.0; Windows NT 9.0; en-US)
Host: twitter.com


// This is another delusion request towards bing.com
GET /afyu/index.php?r=gate&gh=6c..ae&group=2507rcm&debug=0&ips=192.168.X.Y HTTP/1.1
User-Agent: Mozilla/5.0 (Windows; U; MSIE 9.0; Windows NT 9.0; en-US)
Host: www.bing.com
Connection: Keep-Alive
```

Listing 4.7: Delusion of SpyEye C&C communication

```
// This is active C&C communication with a C&C server
GET /folder/gate.php?guid=Admin!COMPNAME!18273645&ver=10133&stat=ONLINE&ie=8.0.6001.18702
     &os=5.1.2600&ut=Admin&cpu=4&ccrc=F8..BF&md5=e1..b1 HTTP/1.0
User-Agent: Microsoft Internet Explorer
Host: XXXXXXXXXXX.com

// This is the delusion request towards facebook.com
GET /login.php?guid=5.1.2600!COMPNAME!18273645&ver=10325&stat=online&ie=8.0.6001.18702
     &os=5.1.2600&ut=Admin&ccrc=36..45&md5=0f..af&plg=customconnector HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0)
Host: facebook.com
```

are hosted on EC2, too, because EC2's IP addresses are pooled. Furthermore, we found that the Bancos botnet distributes bot configuration updates by help of `dropbox.com`. While `dropbox.com` is also being used for sharing benign contents, it is hardly possible to blacklist its domain or IP addresses without interfering with benign usage. In such cases, a detection approach such as *CoCoSpot* helps to fill the gap between a blacklist approach and more fine-grained filtering based on active C&C channels.

4.8 Conclusion

With *CoCoSpot*, we have shown that for a variety of recent botnets, C&C protocols can be detected using traffic analysis features, namely the message length sequence of the first 8 messages of a flow, the carrier protocol as well as differences in the encoding scheme of the URI's query section in case of HTTP messages. The huge benefit of our approach is to be independent from payload byte signatures which enables the detection of C&C protocols with obfuscated and encrypted message contents, as used by the majority of modern botnets. In addition, our C&C flow fingerprints complement existing detection approaches while allowing for finer granularity compared to IP address or domain blacklists. Especially the inherent distinction between active and inactive C&C channels renders *CoCoSpot* less prone to delusion, as shown in case of the Kuluoz botnet.

As a side-effect, our C&C flow clustering can be used to discover relationships between malware families, based on the distance of their C&C protocols. Experiments with more than 87,000 C&C flows as well as over 1.2 million Non-C&C flows have shown that our classification method can reliably detect C&C flows for a variety of recent botnets with very few false positives.

The technique presented in this chapter has focussed on the recognition of C&C channels. In order to evade detection, botmasters could design their C&C channels in a more stealthy manner so that the identification of C&C channels becomes even more difficult. As such, C&C could be performed over less common protocols for botnet C&C. For example, while most botnet C&C channels exhibit HTTP as carrier protocol, botnets could instead build on DNS. From a botmasters point of view, the DNS protocol has the advantage that most networked environments require DNS as part of the regular operation, since the domain resolution via DNS is a service protocol to most other application layer protocols. Thus, the following chapter will deal with DNS as carrier protocol for botnet command and control.

Chapter 5

Detecting Botnets with DNS as Command and Control

5.1 Introduction

Botnets, i.e., sets of computers that are infected with a specific malicious software that allows these computers to be remote controlled, have become one of the biggest security issues on the Internet imposing a variety of threats to Internet users. Therefore, organizations have keen interest to keep the number of bot infections low. Since the remote command and control channel (C&C) is a defining characteristic of botnets, techniques have been developed to *detect* bot infections by identifying the C&C network traffic.

Advances in malware research have challenged botnet operators to improve the resilience of their C&C traffic. Partly, this has been achieved by moving towards decentralized structures (like P2P) or by otherwise obfuscating and even encrypting communication [SGE+09, SCC+09, HSD+08, BHB+09, CL07, GYP+09]. This makes it harder for researchers to distinguish malicious from benign traffic, albeit not impossible.

In the previous chapter, we proposed our method of identifying and recognizing botnet C&C channels, even for encrypted C&C communication. It was only a question of time when botnet C&C channels would be designed in a way that C&C messages are hidden in common application layer protocols, striving for covert communication.

Recently, we observed a specific type of malware termed *Feederbot* that showed strange behavior in the sense that it seemingly did not use any obvious C&C channel. A significant amount of traffic generated by the malware were messages for the Domain Name System (DNS). By reverse engineering the particular sample, we found out that the bot (ab)used DNS as a communication channel for C&C traffic. Apart from this insight, we were interested in the difficulties to detect this type of seemingly "covert" and "hard to detect" traffic. Since DNS has not been

documented so far as a C&C protocol in botnets[1], such botnets benefit from the fact that currently there is no specifically tailored detection mechanism, which in turn raises the probability for the botnet to remain undetected. We achieved to detect this particular type of C&C traffic using machine learning techniques and traffic analysis.

For example, we applied the resulting method on purely malicious traffic produced using our dynamic malware analysis network SANDNET and found that in over 14 million DNS transactions of over 42,000 malware binaries we did not produce any false positive. In fact, in addition to Feederbot, we were able to identify a second class of malware that also used DNS as C&C channel.

One reason for our good results was the way in which DNS was used for communication: the botnet was using the technique of *DNS tunneling* to evade detection. DNS tunneling refers to the technique in which data is transmitted within resource record fields of a DNS message. As a bottom line, our results underline that covert communication must not necessarily be harder to detect than non-covert communication. On the contrary, the covert communication we analyzed introduced anomalies to DNS traffic that can be identified. So the difficulty was not only to detect the presence of C&C information in DNS, it was also to identify the carrier (i.e., DNS) over which covert communication takes place.

In summary, the contributions of this chapter are threefold:

- To our knowledge, we are the first to document DNS-based botnet C&C traffic.
- We present a technique that distinguishes between DNS-based C&C and regular DNS communication in real-world DNS traffic. In other words, we provide a technique for the *detection* of this particular class of malware.
- We present a classifier that can distinguish purely malicious communication into DNS-based C&C and regular DNS communication. In other words, we provide a technique for the *classification* of malware samples based on their behavior.

The remainder of this chapter is structured as follows: We present the case study of Feederbot in Section 5.2. We then describe our detection and classification approach in Section 5.3. We give a brief discussion of our findings in Section 5.4 and describe related work in Section 5.5.

5.2 Case Study: DNS as Botnet C&C

From the point of view of a botmaster, a trade-off between C&C communication visibility and the bot-inherent need to communicate arises. On the one hand, bots

[1] There is only anecdotal evidence for DNS as botnet C&C [Bro11].

must communicate with their C&C instance to receive instructions and transmit data such as stolen credentials. On the other hand, botmasters try to hide the C&C traffic in order to avoid detection. Usually, the design of a botnet's C&C results in messages being obfuscated or encrypted so that it is more difficult to detect and understand the semantics of certain types of C&C traffic.

5.2.1 DNS as Carrier for Botnet C&C

Whereas several application layer protocols have been analyzed by the research community concerning the usage as a basis for botnet command and control, to our knowledge we are the first to openly analyze the Domain Name System protocol as carrier for botnet C&C. DNS, when compared to other application layer protocols provides some advantages. Concerning its usage as botnet C&C, DNS has not been seen so far. Thus, botnets using DNS as C&C benefit from the fact that currently there is no specifically tailored detection mechanism, which in turn, raises the probability for the botnet to remain undetected. Even in environments with heavily restricted Internet access, e.g., by means of firewalling and proxying, DNS is usually one of the few protocols – if not the only one – that is allowed to pass without further ado. Furthermore, whereas for some protocols such as HTTP, there are a number of existing methods to analyze and inspect the network traffic like the one presented by Perdisci et al. [PLF10], DNS is usually served "as is". As another advantage, DNS was designed as a distributed system and as such provides advantages in terms of resilience.

Using SANDNET, our dynamic malware analysis environment, we discovered a bot that indeed uses DNS messages as carrier for command and control traffic. As DNS is a new kind of botnet C&C, we provide some insight into the inner workings of this bot named *Feederbot*. We gained insight by reverse engineering the Feederbot sample as well as analyzing the network traffic that was captured during the analysis of Feederbot in SANDNET.

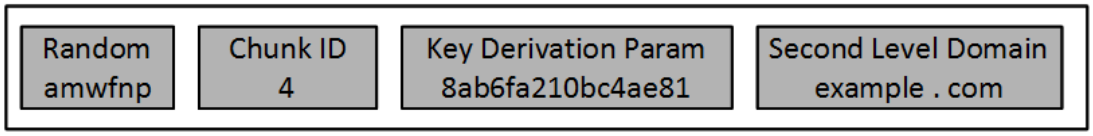


Figure 5.1: Example of Feederbot's DNS Query Domain Name

Feederbot uses valid DNS syntax. Its C&C messages consist of DNS messages with TXT resource records. Furthermore, the query domain name is used to transmit certain parameters from the bot to the C&C server such as parameters for key derivation. An example of the query domain name structure is given in Figure 5.1. Feederbot has to query the C&C servers directly, bypassing the pre-configured DNS resolver on the host because the domains that are used in

Feederbot's requests are not delegated. Manual resolution of seven domain names seen in Feederbot requests starting at the DNS root, i.e., not querying Feederbot's DNS C&C servers, results in NXDOMAIN responses.

We can only speculate as to why Feederbot avoids the pre-configured resolver and directly queries its DNS servers. One reason could be that in this way, the corresponding DNS C&C transactions leave no traces in DNS resolver logs, caches or passive DNS databases. In contrast, the fact that a different than the pre-configured DNS resolver is used, might itself be suspicious enough to catch one's eye – especially in homogeneous environments.

5.2.2 Segmentation and Encryption

Feederbot's C&C traffic is split into message chunks with a maximum length of 220 bytes per chunk. One message chunk is transmitted in the rdata field of a TXT resource record in the DNS response. The structure of a Feederbot message chunk is shown in Figure 5.2. The query domain name (Figure 5.1) contains among others the identifier for the message chunk that is to be retrieved from the C&C server.

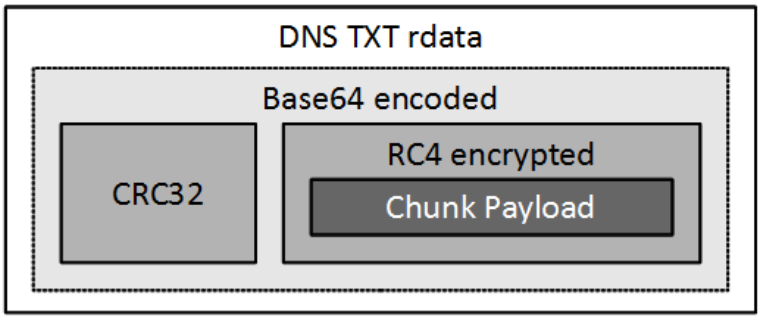


Figure 5.2: Structure of a Feederbot DNS C&C Message Chunk

In order to evade detection, most of the message chunks are encrypted using the stream cipher RC4. Feederbot uses a variety of different encryption keys. A specific part of the DNS query domain name is used to transmit parameters for key derivation. As an example, one such parametrized key derivation function takes as input a substring of the query domain name *qdparam*. This substring *qdparam* is then RC4-encrypted with the string "feedme" and the result is used to initialize the RC4 decryption of the actual C&C message chunks. The stream cipher is used in a stateful manner, so that if a message chunk gets lost, decryption of subsequent message chunks will fail. In addition, Feederbot's C&C message chunks make use of cyclic redundancy checks to verify the decryption result. The CRC32 checksum precedes message chunk payload and is not encrypted.

Using the results from the dynamic analysis in SANDNET as well as the reverse engineering efforts, we achieved the implementation of a passive decryption

utility in order to decrypt Feederbot's DNS C&C messages. Additionally, we implemented a low interaction clone of Feederbot to actively analyze its C&C.

Feederbot receives instructions from several DNS C&C servers. Initially, when Feederbot is launched, a request is sent to a very small subset of C&C servers. These servers seem to serve as bootstrapping C&C servers. During our monitoring period of nine months, we have seen only two such DNS servers (in terms of IP addresses) acting as bootstrapping C&C servers. The response to this initial bootstrapping request message is not encrypted and only Base64 encoded. It contains a pointer to at least one other C&C server as well as another domain name. Subsequent communication is encrypted.

5.3 Detecting DNS-based Botnet C&C

When facing a new kind of botnet C&C as with Feederbot, it is of concern how such a communication can be detected. Inspired by the results of the Feederbot analysis, we developed machine learning features for a DNS C&C classification method.

Our classification approach is based on several prerequisites. First, we assume that DNS-based C&C channels typically carry dense information, such as compressed or encrypted data. As botmasters strive for resilience, encryption is becoming more prevalent among botnet C&C. In addition, due to the message length limits of DNS, botmasters need to fit their commands into relatively small messages. Second, we assume that to a certain degree there is a continuous "downstream" flow of information, i.e., from the C&C server to the bot. Admittedly, this second condition may not be met for bots with certain monetization functionalities, e.g., low profile trojans. However, we believe that lots of bots actually fulfill this requirement, especially spam-sending bots or click fraudsters. In these cases, the C&C server is required to continuously provide the bot with input data, such as spam target email addresses, templates and text blocks or URLs to feed the click fraud module.

5.3.1 Classification Features

Key to our detection method are certain differences between regular DNS usage and DNS C&C, which we divide in two categories. The first category deals with differences concerning the use of the rdata field whereas the second category addresses differences concerning the communication behavior.

Rdata Features

The basic unit for these entropy-based features is the rdata field of all resource records of one DNS response. For brevity, this unit is referred to as rdata message

in the following. As an example, in the context of Feederbot this corresponds to one message chunk. Note that we do not restrict the features to certain resource record types or sections.

Shannon entropy is a measure of randomness in a string of data. Given a finite alphabet $\Sigma = \{0, 1..255\}$, the entropy estimates how randomly the characters in word w are distributed. We use the maximum likelihood estimator to calculate the *sample entropy* of a message $w \in \Sigma^*$, f_i denotes the frequency of character i:

$$\hat{H}(w) \;=\; -\sum_{i=0}^{255} f_i \cdot log_2(f_i) \tag{5.1}$$

Then, the word $w_1 = 001101$ has a lower sample entropy than the word $w_2 = 012345$. We exploit the fact that encrypted or compressed messages have a high entropy. As we assume encrypted C&C, the C&C messages exhibit a high entropy. Encrypted data composed of characters of the full 8-bit-per-byte alphabet will converge towards the theoretical maximum entropy of 8 bits per byte. In this case, entropy is typically referred to as *byte entropy*. In fact, when using DNS as C&C, certain fields of the DNS protocol such as TXT or CNAME resource records' rdata do not allow the full 8 bits to be used per byte. Thus, botmasters have to "downsample" their C&C messages to the destined alphabet, e.g., by means of Base64 or Base32 encoding. This implies that the resulting message exhibits a comparatively low *byte entropy*. We overcome this issue by estimating the destined alphabet size by counting the number of distinct characters in a given field. After that, we calculate the expected sample entropy for random data based on the estimated alphabet size.

Another issue is posed by the fact that short strings of data – even when composed of random characters – rarely reach the theoretical maximum entropy. For example, a string of 64 bytes length, based on the 8-bit per byte alphabet Σ has a theoretical maximum byte entropy of 6 bits. However, considering a string r of 64 bytes length with randomly distributed bytes of the alphabet Σ, the byte entropy is typically lower than 6 bits, e.g., around 5.8 bits. This finding is based on the birthday paradox. Basically, encrypted data is randomly distributed, but randomness does not imply a uniform distribution. Thus, if a string r is short (e.g., 64 bytes), the expected byte entropy is significantly below 8 bits, although r might be purely random.

We overcome this issue by calculating the *statistical byte entropy* for a string of a given length. This is done as follows. Empirically, we compute the average byte entropy of a set of $x \;=\; 1,000$ random words for every length $1 < N < 2^{10}$. For any word $w_1, \ldots, w_x$, we compose a random byte distribution and calculate the byte entropy. Since x was chosen sufficiently large, calculating the mean over all x byte entropies of words with length N estimates the expected *statistical byte entropy* of random data of length N. Figure 5.3 shows the maximum theoretical entropy and the expected statistical random entropy for the full 8-bit per byte

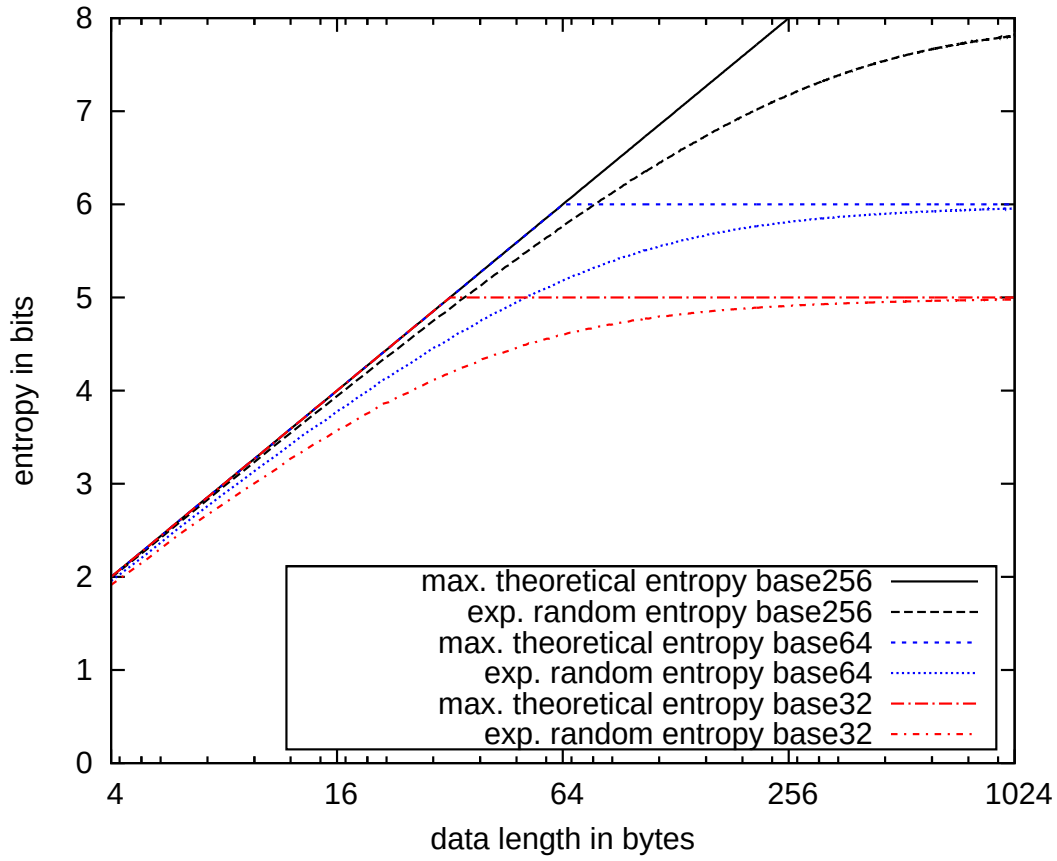


Figure 5.3: Statistical byte entropy

alphabet Σ as well as typical Base64 and Base32 alphabets. One important feature is the deviation of the actual sample entropy to the expected statistical random entropy. Effectively, this covers different alphabets in a flexible fashion, i.e., we can dynamically detect high sample entropies in data encoded with any alphabet such as Base64, Base32, Base16 or the like.

In addition, we measure the minimum and maximum byte values of a given rdata message as well as the coverage of some subsets of the ASCII character set such as capital letters and digits. The complete list of features for an rdata message m consists of:

- number of distinct byte values in m
- minimum byte value in m
- maximum byte value in m
- number of ASCII capital letters (byte values 65-90) in m

- number of ASCII digits (byte values 48-57) in m
- length of m in bytes
- absolute difference of the statistical byte entropy at given length of m and the entropy of m

Aggregated Behavioral Communication Features

Furthermore, we exploit behavioral properties of DNS C&C bots. These features address the fact that a certain amount of information has to be transmitted via the DNS C&C channel, e.g., spam templates, email addresses or click fraud URLs, and that the C&C channel exhibits a certain level of persistence. In contrast to the rdata features, the behavioral communication features do not operate on individual DNS response resource records. Instead, they operate on aggregated data. Thus, for the behavioral communication features to be computed, we need to define aggregation criteria, so that communication properties can be aggregated over time before calculating the features. When analyzing Feederbot, we observed that the DNS C&C servers were contacted directly, avoiding the pre-configured DNS resolver. In this case, in order to compute behavioral communication features (e.g., bandwidth), the requester's IP address as well as the IP address of the DNS server serve as aggregation criteria. In contrast, if DNS requests are directed towards pre-configured DNS resolvers, the requester's IP address and the query domain name (and parts thereof such as the second-level domain) are used as aggregation criteria. In both cases, we assume that the requester's IP address represents one host, i.e., no network address translation has been performed. Note that even if network address translation was performed, detecting that DNS C&C was used might still be possible. However, in such a case, it would be impossible to identify the infected host.

We define the following behavioral communication features. First, we measure the size of all rdata messages (i.e., rdata fields of DNS response resource records) and compute the corresponding aggregated bandwidth over time. We expect that the data volume transmitted between the DNS C&C server and the bot will tend to be significantly larger when compared to regular DNS usage, as DNS is typically not used for data transmission. This observation results in either larger messages and/or in an increased bandwidth consumption between the bot and the C&C server.

Second, the information flow between one bot instance and the C&C server is expected to appear more persistent. We measure the persistence as the maximum of the time between two DNS responses, as well as the communication duration calculated as the time between the first and the last message exchanged with a C&C server. Yet simple, we expect these behavioral communication features to be effective enough in order to extend a classifier based on the rdata features.

5.3.2 Clustering DNS traffic

Given these features, our goal is to develop a binary classifier that is able to detect DNS-C&C traffic. Before we can classify DNS traffic, we need to extract two sets of training data. As a first step, we extract two different kinds of DNS traffic. We define one transaction to be composed of one DNS request as well as the corresponding DNS response. Based on the network traffic caused by the Feederbot execution that we analyzed in Section 5.2, we compile a set of known Feederbot DNS C&C transactions. This set is referred to as D_D and contains 3128 DNS transactions. D_D is composed of DNS transactions fulfilling both of the following two conditions:

- The transaction was directed to any of the DNS bootstrapping C&C servers which were verified to be used as C&C during reverse engineering.
- The request has a query domain name ending in one of 7 second-level domains observed during dynamic execution analysis.

Furthermore, we extract D_N, a set of DNS transactions of 30 executions of bots that knowingly do *not* use DNS as C&C. In addition, we manually inspected 500 (1%) of the 47,433 DNS transaction of D_N. D_N contains DNS transactions that occurred as part of the monetization functionality of these bots such as spamming or click fraud. The complete list of bot families used to compile D_N is given in Table 5.1. The bot family names are based on a majority voting of up to 42 labels per sample, acquired from VirusTotal [vir12].

Bot Family	*Type of C&C*	*# Execs*	*DNS TXs*
Unknown	HTTP	3	620
Unknown	IRC	4	1951
Agobot	IRC	1	163
Koobface	HTTP	2	4119
Rbot	IRC	2	300
Sality	Custom P2P	4	5718
Sdbot	IRC	3	916
Swizzor	IRC	1	93
Virut	IRC+CE	4	17,740
Virut	IRC (plaintext)	4	15,789
Zbot	HTTP+CE	2	24

Table 5.1: Bot executions used to acquire Non-DNS-C&C transactions. CE=custom encryption; TX=transaction.

With respect to approximately equally sized training sets, the next step consists in drawing 5000 elements of D_N at random into D_{NS} so that the resulting set

D_{NS} has approximately the same size as D_D. We compose the set $D := D_D \cup D_{NS}$ as the union of D_D and D_{NS}.

At this point, we extract the rdata features described in Section 5.3.1 from all DNS transactions in D. The resulting set of feature vectors is referred to as F. Moreover, the elements of F are normalized and the normalization parameters are stored. Using k-Means clustering with $k = 2$ and Euclidean Distance function, we separated F into two clusters C_D and C_N. The cluster which contains the most known DNS C&C elements is considered as C_D, the other one as C_N.

The clustering step aims at distilling the characteristic transactions for DNS C&C into the resulting cluster C_D. Table 5.2, the classes to clusters comparison shows that only 6 elements of D_D were assigned to cluster C_N. Manual inspection revealed that each of these 6 transactions carries a Feederbot C&C message chunk with an empty payload. Thus, these are not considered as characteristic C&C messages. All of the transactions in the Non-DNS-C&C set D_{NS} were assigned to the Non-DNS-C&C cluster C_N.

	C_N	C_D
D_{NS}	5000	0
D_D	6	3122

Table 5.2: Classes to clusters comparison

Based on the clustering results, we develop a classification method. First, we aim at classifying malicious network traffic, i.e., network traffic caused by malware as it is acquired in SANDNET. This binary DNS traffic classifier is supposed to distinguish between DNS-based C&C and Non-C&C in order to find other malware executions that exhibit DNS C&C. Second, we aim at detecting DNS-based C&C channels in real-world DNS traffic.

5.3.3 Detecting Bots that use DNS C&C

As we were curious to find further malware samples using DNS C&C – apart from the Feederbot sample and its execution that we analyzed in depth in Section 5.2 – we designed a DNS C&C classifier that can be applied to the network traffic gained by our SANDNET analysis of more than 100,000 malware samples between February 2010 and April 2011. Due to the enormous amount of data and because we wanted to identify individual DNS C&C transactions, we intentionally restrict ourselves to the rdata features as described in Section 5.3.1. Therefore, we calculate the rdata features for the 14,541,721 DNS transactions of all 42,143 samples that have been executed in SANDNET and that exhibited DNS traffic. Each feature vector reflects one DNS transaction.

In order to classify DNS traffic, we calculate the mean cluster centroids of both clusters C_D and C_N built in Section 5.3.2. Each feature vector is scaled using

the normalization parameters from the training phase. Finally, as classification method, we implemented a Euclidean Distance based classifier which assigns the class of the closest cluster to the given feature vector.

All in all, 109,412 DNS transactions of SANDNET traffic are classified as DNS C&C. This procedure reveals 103 further executions of Feederbot samples. Surprisingly, our classification even discovers another bot family that uses DNS-based C&C. We term this newly found bot *Timestamper* due to the fact that it uses the Unix timestamp of the current date and time in the query domain name. Timestamper, in contrast to Feederbot, uses the pre-configured DNS resolver. Our classifier detects 53 executions of Timestamper in SANDNET data. Manual inspection verifies that all of the 156 executions which have associated DNS transactions classified as DNS C&C are either Feederbot or Timestamper executions. Thus, at this point, we draw the conclusion that, in terms of SANDNET executions, our classifier does not produce any false positive.

As part of an effort to estimate the false negative rate, we compile a regular expression for Timestamper's DNS C&C requests that matches the Unix timestamp in the query domain name. 1679 transactions where the regular expression matches the query domain name are considered as Timestamper's DNS C&C requests, the remaining 1851 DNS transactions are considered as Non-C&C DNS. Our classifier correctly classifies all of the 1679 transactions as being DNS C&C transactions, i.e., showing no false negatives among Timestamper's DNS C&C traffic.

In addition, we evaluate our classifier against 1851 Timestamper DNS transactions which are *not* part of its DNS C&C in order to estimate the false positive rate on the transaction level. Once more, our classifier correctly considers all of these 1851 transactions as not being DNS C&C transactions, i.e., showing no false positive among the Timestamper DNS transactions.

To summarize, our binary DNS C&C transaction classifier successfully reveals 103 further executions of Feederbot and discloses 53 executions of Timestamper, the newly disclosed bot that also uses DNS C&C. In addition, the results show that even though we trained only on known Feederbot DNS C&C of one execution, our classifier was able to correctly classify DNS C&C transactions of another, completely unrelated type of malware.

5.3.4 Detecting DNS C&C in mixed traffic

Furthermore, we evaluate our classifier on mixed workstation DNS traffic. Therefore, we recorded DNS network traffic at our Institute at the network router and NATting point where all traffic from workstations towards internal servers as well as arbitrary Internet destinations passes. Traffic from the internal servers heading for Internet destinations was excluded in order for recursive DNS queries caused by the DNS resolver to be avoided. All DNS traffic was recorded before source network address translation (NAT) on the router was applied. In this manner,

we are able to capture DNS traffic destined for the pre-configured DNS resolver and for remote DNS resolvers on the Internet.

Additionally, we executed one Feederbot sample and one Timestamper sample from inside the workstation net, each for one hour. Both samples were executed in virtual machines on workstation computers that were used for regular operation throughout the whole measuring period. The network access of the virtual machines was configured to use NAT on the workstation hosts. Thus, the traffic originating in each infected virtual machine and the corresponding workstation host traffic cannot be distinguished by source IP address and regarding our aggregation they represent one entity. Additionally, all network traffic caused by the virtual machines was recorded individually on the workstation host. Our goal is to detect those workstations that executed the Feederbot and Timestamper samples.

The captured network traffic contains a total of 69,820 successful DNS transactions from 49 distinct workstation IP addresses of our Institute, captured between 7 a.m. and 8 p.m. on a regular weekday. This dataset is referred to as T_{all}. The Feederbot VM caused a total of 2814 DNS transactions (T_F) among which 1092 were DNS C&C transactions (T_{FCnC}). Additionally, we observed 4334 HTTP flows during its click fraud activity. The network trace of the VM executing the Timestamp bot showed a total of 181 DNS transactions (T_T) with 102 DNS C&C transactions (T_{TCnC}). Consequently, the traffic capture contains 66,825 DNS transactions caused by the legitimate workstations during regular operation.

In order to make use of the aggregated behavioral communication features, we extended our classification method. Based on the results in Section 5.3.3, we compiled a set of 10 Feederbot executions revealed in Section 5.3.3. For these executions, we calculated the following three thresholds:

1. t_b the mean bandwidth per aggregate

2. t_{mi} the mean of the maxima of the gaps between two consecutive C&C messages for DNS C&C flows

3. t_{si} the standard deviation of the maxima of the gaps between two consecutive C&C messages for DNS C&C flows

As a first step, we applied the classifier presented in Section 5.3.3 to all of the DNS transactions in T_{all}. This results in the set of candidate DNS C&C transactions T_{cand}. Furthermore, in order to apply the behavioral communication features, we computed two kinds of aggregates for the candidate transactions in T_{cand}. First, we aggregate by each pair of source and destination IP addresses. Second, we aggregate by each pair of source IP address and second level domain of the query domain name.

Subsequently, the set of aggregates is filtered, eliminating all aggregates that do not fulfill the behavioral properties. We exclude aggregates with a computed

bandwidth smaller than t_b and a maximum time between two C&C messages greater than $d \cdot t_{si} + t_{mi}$ with $d = 3$. This filtering step makes sure that only those channels with persistence be considered as C&C channels. None of the aggregates were excluded in the filtering step.

Based on the resulting set of aggregates, we consider each source IP address to be infected with malware using DNS C&C. Indeed, only the two IP addresses of the workstations that hosted the Feederbot and Timestamper bots were classified as DNS C&C infected hosts. To sum up, we showed that our classifier can even detect DNS C&C transactions in mixed network traffic of regular workstations.

5.4 Discussion

Though achieving high true positive rates, there are certain limitations that bots could exploit to evade our detection. One such limitation is posed by the fact that botmasters could restrict their C&C messages to very small sizes. In practice, message contents could be stored in e.g., 4 bytes of an A resource record's rdata. In this case, our rdata features alone, which are currently applied to individual C&C messages, would not be able to detect these C&C messages as high entropy messages because the statistical byte entropy of such really short messages is very low and our estimate of the alphabet size by counting the number of distinct bytes is inaccurate for short messages.

In this case, a countermeasure could be to aggregate several messages and compute aggregated rdata features. Furthermore, for each aggregate, the change of entropy among subsequent messages can be measured. Additionally, for certain resource records one could compare the distribution of byte values against the expected distribution. For example, the rdata of an A resource record contains IPv4 addresses. However, the IPv4 address space is not uniformly distributed. Instead, certain IPv4 address ranges remain reserved, e.g., for private use such as 10.0.0.0/8 (RFC1918) or 224.0.0.0/4 for multicast. These might rarely show up in Internet DNS traffic whereas other addresses, e.g., popular web sites, might appear more often in DNS query results.

When looking at Feederbot, it becomes obvious that the query domain name can be chosen completely at random. In general, this is true for botnets where the DNS C&C servers are contacted directly. In order to avoid raising suspicion, the botmasters could have chosen e.g., random or even popular second-level domains. This would become a problem for our detection mechanism if only the query domain name was used for aggregation alone. However, as we also aggregate by the DNS server's IP address, our classifier can still detect this kind of DNS C&C. As a result, we suggest to aggregate by at least both, the DNS server's IP address and the query domain name, because the botmaster can only arbitrarily change one of them.

Another limitation is posed by the fact that our behavioral communication fea-

tures aim at botnets with a central C&C architecture using a limited set of C&C servers. Botmasters might exploit this by spreading the communication with C&C servers over lots of different C&C destinations so that even the aggregated behavioral features such as the aggregated bandwidth remain subliminal. Effectively, a further step in this direction would be to change for a peer-to-peer C&C architecture where each bot is part of the peer-to-peer network. However, this opens the door for a whole variety of techniques addressing peer-to-peer networks such as eclipse attacks.

5.5 Related Work

Related work can be grouped in three categories. First, several C&C techniques of botnets have been analyzed in depth [HSD+08, SGE+09, CDB09, CL07, SCC+09]. For example, Holz et al. [HSD+08] provide insights into botnets with peer-to-peer C&C architecture in a case study on the storm worm. Stock et al. and Calvet et al. [SGE+09, CDB09] analyze the Waledac peer-to-peer botnet in detail. However, to our knowledge, we are the first to analyze DNS as carrier for botnet C&C. Our work depicts how C&C messages are structured, encrypted and encoded in regular DNS syntax as is the case with Feederbot, a bot using DNS C&C we discovered during this work. Additionally, we discuss general architectural issues and limitations of DNS botnet C&C. The second group of related work contains approaches to detect botnets in network traffic. This kind of related work can be separated into application protocol specific approaches and protocol independent approaches. As HTTP is used as botnet C&C, some work has been done to develop specifically tailored detection methods for HTTP-based botnet C&C, such as Perdisci et al. [PLF10]. Goebel and Holz [GH07] present methods in order to detect IRC-based botnets. However, due to their protocol-dependent orientation, none of these approaches are able to detect DNS C&C.

Independent of the application layer protocol, Gu [GPZL08] and Strayer [SLWL08] propose botnet detection methods based on network flow characteristics. However, protocol-independent approaches will likely fail to detect DNS C&C as they expose neither chat-style characteristics nor necessarily spatial-temporal correlated behavior. For example, Feederbot does neither exhibit periodicity nor synchronized transactions among different bot executions – effectively exploiting the gap of existing detection approaches. Therefore, we provide a detection method specifically tailored to DNS C&C based on rdata and behavioral communication features – successfully filling this gap.

A special case is the work of Choi et al. [CLLK07], which – though not specifically targeting DNS C&C – addresses group activities in DNS. The authors define differences between DNS query behavior typical to any kind of botnet and legitimate DNS resolution. According to Choi et al., key features for bot-typical behavior include simultaneous DNS queries, quickly changing C&C server

addresses as well as transient domains. However, as our analysis of Feederbot discovers, none of these assumptions hold. Feederbot's C&C servers stayed up for the whole monitoring period of nine months and DNS queries are not synchronized between different bots. Instead, we exploit rdata features and persistent communication behavior to detect DNS C&C.

The third group of related work covers DNS covert communication. Bernat [Ber08] analyzed DNS as covert storage and communication medium. Born and Gustafson [BG] employ character frequency analysis in order to detect DNS tunnels. However, both approaches do not specifically address the detectability of DNS as botnet C&C. In addition, we significantly improve entropy-based features and combine them with behavioral features to target botnets.

5.6 Conclusion

Inspired by anomalous DNS behavior, we stepped into a whole new kind of botnet C&C. This shows that even though many bot families use IRC or HTTP as carrier protocol for their C&C, malware authors still find new ways of instructing their bots. It is obvious that DNS C&C moves botnet C&C one step further into the direction of covert communication. However, as shown in this chapter, the detection of such botnet C&C, even when covert, remains possible.

We combine protocol-aware information theoretical features with aggregated behavioral communication features and apply them at different levels of network traffic abstraction, i.e., DNS transactions and hosts. In this way, we detect DNS C&C in real-world DNS traffic. Furthermore, we provide means to classify malware concerning DNS C&C usage based on network traffic.

To summarize, to the best of our knowledge we are the first to not only describe a real-world botnet using DNS C&C, but also provide a mechanism to detect DNS C&C in network traffic.

Thus, the previous chapter, presenting our C&C recognition method based on traffic analysis, as well as this chapter which proposes a dedicated detection approach for covert DNS-based C&C round up our work on the detection of botnet command and control channels. The remainder of this thesis deals with the detection of one of the most prevalent monetization techniques, namely rogue visual malware.

Chapter 6

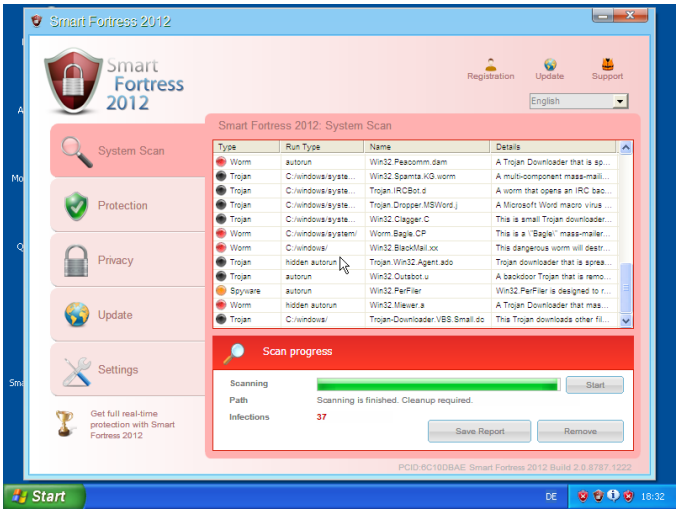
Detecting Rogue Visual Malware

6.1 Introduction

While the previous chapters deal with the detection of botnet command and control channels, in this chapter, we focus on the detection of visual monetization techniques of remote-controlled malware. Given the fact that botnets are free to design their C&C protocols in a subtle and covert way, we have observed in a few cases that botnets such as Zeus P2P even evade traffic analysis means. At that point, one way to detect such malware lies in its monetization. While malware comes in many different flavors, e.g., spam bots [SHSG+11, CL07, SGHSV11], banking trojans [ROL+10, BOB+10] or denial-of-service bots [Naz], one important monetization technique of recent years is rogue software, such as fake antivirus software (Fake A/V) [RBM+10, Mic]. In this case, the user is tricked into spending money for a rogue software which, in fact, does not aim at fulfilling the promised task. Instead, the rogue software is malicious, might not even have any legitimate functionality at all, and entices the user to pay. However, all rogue software has in common to provide a user interface, e.g., be it to scare the user, or in order to ask for banking credentials, or to carry out the payment process.

Figures 6.1(a) and 6.1(b) show example screenshots of two typical rogue software flavors. The first displays a Fake A/V user interface, mimicking a benign antivirus application, while the second exhibits a ransom screen (in German), asking the user to pay before the computer is unlocked. Especially the later category, ransomware, is considered an increasing threat with more than 120,000 new ransomware binaries in the second quarter of 2012 [McA12]. In addition, when referring to the C&C tracking of Fake A/V and ransomware botnets, we observe a significant increase in activity since June 2011, as can be seen in Figure 3.8 in Chapter 3. We conclude that, today, rogue visual software is thus one of the most prevalent techniques for the monetization of remote-controlled malware and address means to detect this kind of threat in this chapter.

As rogue software is required to provide a user interface, we aim at exploiting



(a) Smart Fortress 2012 (Winwebsec family)

(b) Ransomware asking the user to pay (in German)

Figure 6.1: Example screenshots of rogue software

its visual appearance in order to cluster and classify rogue software. We motivate our efforts by the relatively low A/V detection rates of such rogue software, and we aim to complement existing techniques to strive for better detection rates. In particular, we observed that the structure of the user interfaces of rogue software remains constant and can be used to recognize a rogue software family or campaign. Using a perceptual hash function and a hierarchical clustering approach, we propose a scalable and effective approach to cluster associated screenshot images of malware samples.

In short, the main contributions of this chapter are threefold:

- We provide a scalable method to cluster and classify rogue software based on its user interface, an inherent property of rogue visual malware.
- We applied our method to a corpus of more than 187,560 malware samples of more than 2,000 distinct families (based on Microsoft A/V labels) and revealed 25 distinct types of rogue software user interfaces. Our method successfully reduces the amount of more than 187,560 malware samples and their associated screenshot images down to a set of human-manageable size, which assists a human analyst in understanding and combating Fake A/V and ransomware.
- We provide insights into Fake A/V and ransomware campaigns as well as their payment means. More specifically, we show a clear distinction of payment methods between Fake A/V and ransomware campaigns.

The remainder of this chapter is structured as follows. In Section 6.2, we will describe the dataset our analysis is based on and outline our methodology. Subsequently, we will evaluate our method in Section 6.3. Using the clustering results, we will provide insights into the monetization and localization of rogue software, with a focus on four ransomware campaigns in Section 6.4. Section 6.5 will discuss the limitations and evasion of our approach. Finally, we will describe related work in Section 6.6 and conclude in Section 6.7.

6.2 Methodology

A coarse-grained overview of our methodology is shown in Figure 6.2. Our approach consists of three steps. First, we execute malware samples and capture the screen. Furthermore, we compute a perceptual hash of the resulting image and finally, we cluster screenshots into subsets of similar appearance. Our goal is to find subsets of images that – although slightly different in detail – exhibit a similar structure and a similar user perception.

We found that the user interfaces of Fake A/V campaigns vary concerning details such as the number of supposedly dangerous files as well as the rogue software name and logo. However, the *position* of the logo and the *sizes* of user

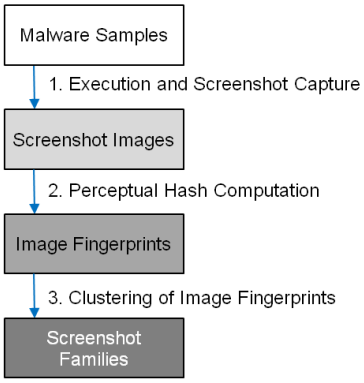


Figure 6.2: Overview of the screenshot clustering methodology

interface elements remain constant among several campaigns of one family. As an example, Figures 6.3(a) and 6.3(b) display screenshots of two malware samples of the Winwebsec family, and Figures 6.3(c) and 6.3(d) display screenshots of two FakeRean malware samples. While the two Winwebsec samples (Smart Fortress 2012 and System Tool) show different logos at the top left corner of the application window and different icons for the application menu, their overall user interface structure is very similar. The same applies to FakeRean. Again, the name differs – Internet Security vs. Spyware Protection – but the application windows and their user interface elements are positioned in the same fashion.

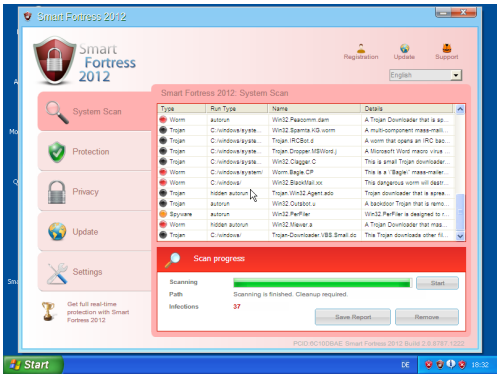
For our clustering step, we aim at separating the images of the Winwebsec family from those of the FakeRean family. Furthermore, if possible, different campaigns should be separated. In other words, we aim at recognizing the structure of the visual appearance in a screenshot, e.g., concerning the position of a program or campaign logo and the sizes of user interface elements, but at the same time ignore detailed information such as colors, the exact program name, window title or the text inside user interface elements.

In fact, we found that rogue software is required to change its name regularly – most likely because users searching for such a software name on the Internet will eventually find out that they fall prey to a rogue software campaign.

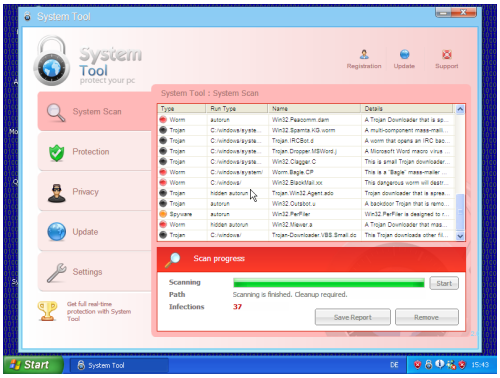
Our basic assumption can be fostered in the observation that rogue software seems to build its user interface from templates which remain constant and only the contents of certain user interface elements differs. Thus, our clustering approach specifically targets these templates.

6.2.1 Dataset

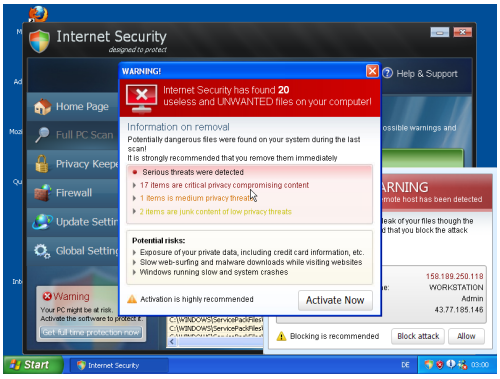
For our image clustering technique, we compiled a corpus of 213,671 screenshots that originate from executing 213,671 MD5-distinct malware binaries representing more than 2,000 malware families based on Microsoft A/V labels. The binaries



(a) Smart Fortress 2012 (Winwebsec)



(b) System Tool (Winwebsec)



(c) Internet Security (FakeRean)

(d) Spyware Protection (FakeRean)

Figure 6.3: Screenshot images of the Winwebsec and FakeRean malware families

were executed in SANDNET and span a time period of more than two years, up to May 2012. Although by far, most of the samples in our malware feeds are indeed malicious, occasionally, a sample represents legitimate software, e.g., Adobe Flash Installer. For example, this stems from the fact that some samples are gathered from public submission systems where users are allowed to upload all kinds of software, possibly even including benign software. However, for our approach, we see no need to exclude *all* legitimate software. Instead, in our clustering results, we expect benign software to be well-separated from rogue visual software because it exhibits different user interfaces. Indeed, as we will show in the clustering evaluation Section 6.3.1, benign software separates well from rogue visual malware.

6.2.2 Malware User Interfaces

In order to capture the visual appearance of a malware sample, we execute the malware sample in a virtual machine and store a screenshot of the whole virtual screen. Typically, each sample is executed for one hour. The virtual machines

use Windows XP 32bit SP3 as operating system and have limited Internet access. SMTP as well as typical infection vector ports were transparently redirected to local spam traps and honeypots. During the execution of a sample, no user interaction was performed. In order to prevent harming others and to mitigate denial-of-service attacks, we restricted outgoing traffic to the protocols IRC, DNS and HTTP and throttled outgoing traffic. As the set of executed samples covers all kinds of malware, some of the screenshot images do not show any graphical user interface at all, e.g., malware operating completely in the background, or display an error message. We will deal with the fact that not all screenshots reflect rogue visual software in subsequent sections.

6.2.3 Perceptual Hash Functions

In our case, one screenshot image consists of 800x600 pixels of 24 bit color depth, resulting in ca. 11 MB of uncompressed data per image. To reduce the amount of data, we extract a fingerprint of the visual appearance, for each image. This fingerprint is then used in the clustering phase. As we are mainly interested in the perception, we turn to so-called perceptual hash functions. While cryptographic hash functions aim at resulting in two different hash values upon the slightest difference between two input values, perceptual hash functions aim at grasping semantic differences of a certain context.

We use the perceptual hash function p based on the discrete cosine transform (DCT) as proposed by Zauner [Zau10]. First, an image is converted to grey scale and a smoothing for each 7x7 pixel subarea is performed. Furthermore, the image is resized to 32x32 pixels and the two-dimensional type-II DCT coefficients are calculated. As high frequencies might not sustain compression, 64 low-frequency coordinates (8x8) are considered – similar to the JPEG compression standard – to build up an 8-byte bitstring fingerprint. Effectively, this fingerprint provides a structural context-sensitive feature vector of the screenshot image.

An advantage of this perceptual hash function is its robustness against minor modifications to the visual appearance such as a change in color, certain levels of distortion and non-uniform scaling. Thus, even if certain elements of a rogue software user interface are changed, distorted or blurred – for example in order to result in different cryptographic hash values – the perceptual hash function will resist (to a certain degree). During the manual classification, which will be described in more detail in Section 6.2.4, we found samples that changed the desktop wallpaper and their perceptual hash value is close to the ones with the regular wallpaper. Also we found user interfaces where the colors of some user interface elements were changed, which also produced nearby perceptual hash values.

6.2.4 Intra-Fingerprint Coherence

For all of the 213,671 screenshot images, we applied the perceptual hash function, which results in a set of 17,767 distinct perceptual hash fingerprint values. We denote the full set of perceptual hash fingerprints as F. Note that not all of these fingerprints refer to user interfaces of rogue software, but also include error messages – e.g., cases where the sample to be executed was corrupt – or a blank screen, i.e., the sample did not exhibit a visual appearance at all.

Using a perceptual hash function, two images with a very high degree of structural similarity might result in the same hash value. While in general, this fact is desired, in the worst case, two images with the same hash value might not reflect the structural and perceptual similarity we aim at. In this case, we would need to extend the fingerprint either by increasing the fingerprint length or by taking additional features into account. We define the term *Intra-Fingerprint coherence* to reflect that images with the same fingerprint indeed provide the same structural and perceptual properties and thus likely reflect images of the same malware family or campaign.

In order to test our fingerprints for Intra-Fingerprint coherence, we randomly selected 345 fingerprint values (ca. 2% of 17,767) with at least 35 associated screenshot images per fingerprint, and for each fingerprint we inspected at least three random images manually. We checked whether the positions of user interface elements remain constant, especially the positions of logos, progress bars and text area, list or table elements. In all cases, the images provided the required similarity in structure.

At the same time, we classified the fingerprints and assigned a campaign label or – in case the screenshot does not show rogue software – a state label such as whether an error message, a blank screen or a browser window is displayed or describe the application window. For example, a blank screen occurs if the malware operates completely in the background, e.g., in case of Zeus, a popular banking trojan [BOB+10]. A prevalent group of fingerprints is caused by the Hotbar/Clickpotato downloader which masks as installers for various software such as VLC or xvid. Other examples for malware which show neither Fake A/V nor ransomware include Adware causing the web browser to open a specific website, cracks or serial key generator programs and Windows Explorer displaying the contents of a certain folder.

The labeled set of 345 fingerprints covers 18 different fake A/V campaigns and two ransomware campaigns, shown in Table 6.1. The second column of Table 6.1 denotes the Microsoft A/V family label, if at least one of the associated samples was detected by Microsoft.

Perceptual Hash Label	MS A/V Family
Fake A/V and Rogue	
Clean Catch	undetected
Smart Fortress 2012	Winwebsec
System Tool	Winwebsec
Personal Shield Pro	undetected
FakeScanti	FakeScanti
S.M.A.R.T Failure	FakeSysdef
Security Tool	Winwebsec
Internet Security	FakeRean
XP Home Security 2012	FakeRean
Security Monitor 2012	undetected
Antivirus Protection 2012	FakeRean
Spyware Protection	FakeRean
Antivirus Action	undetected
PC Performance and Stability	FakeSysdef
Security Shield	undetected
Security Central	undetected
Windows Trojans Sleuth	FakePAV
Microsoft Security Essentials	FakePAV
Ransomware	
Windows Security Center	Ransom
Bundespolizei	Sinmis

Table 6.1: Perceptual Hash Fingerprint Labels for 18 fake A/V and 2 ransomware campaigns and, if available, Microsoft A/V Family Labels

6.2.5 Distance Computation and Clustering

While the perceptual hash computation relaxes near-duplicates, i.e., it aggregates very similar images into one fingerprint value, a malware family or campaign typically spans multiple different fingerprint values. In other words, one fingerprint is too specific to reflect a campaign or even a whole malware family. We found several reasons for this. First, the application window might not always be at the same position on the screen or other foreground application windows might obscure parts of the rogue software's window(s). This could result in different perceptual hash values. Second, rogue software interfaces can provide several different views. For example, a fake A/V program might provide one view of a scan in progress (Figure 6.3(a)) and another one when the scan is finished and the user is prompted for action, e.g., as shown in Figure 6.3(c).

Thus, in order to aggregate several closely related fingerprints into subsets of campaigns or malware families, we add a clustering step. In this clustering phase, we use the normalized bit-wise hamming distance as distance function between two perceptual hash fingerprints. As the perceptual hashes have a fixed length of 8 bytes, the hamming distance returns the number of differing bits, with a maximum of 64. We normalize the hamming distance to the range 0.0 to 1.0 by dividing it by 64.

Since the perceptual hash is a locality-sensitive hash [KG09], we can rely on the hamming distance as a simple and fast distance function. Similar images will result in a low hamming distance (towards 0.0), while different images will exhibit higher distance values (towards 1.0). Another advantage of the hamming distance is its high performance (XOR arithmetic). The combination of the perceptual hash function and the normalized hamming distance allows us to use agglomerative hierarchical clustering despite its worst-case complexity of $\mathcal{O}(n^3)$.

Thus, we apply agglomerative hierarchical clustering to the set of fingerprint values. Furthermore, we decided to use average linkage clustering to avoid the chaining phenomenon. Finally, to aggregate similar fingerprints into one cluster, a cut-off threshold determines the maximum distance between any two different fingerprint clusters. If the distance of any two distinct fingerprint clusters is less than the cut-off threshold, the associated screenshot fingerprints will be filed in the same cluster, otherwise they would be filed into different clusters.

6.3 Evaluation

Our clustering evaluation can be divided into two parts, Intra-Fingerprint Coherence as well as Cluster Generalization. In addition, we evaluate the true A/V detection rate of Fake A/V campaigns identified by means of our clustering. As described in Section 6.2.4, we performed the Intra-Fingerprint evaluation by manually inspecting at least 3 random screenshot images for 345 random fingerprints

and assigned a campaign description to each fingerprint. This way, we made sure that there is no collision of fingerprint hashes which breaks our similarity assumptions and we achieve Intra-Fingerprint Coherence.

6.3.1 Clustering evaluation

In the next step, we aim at evaluating how well our approach generalizes. We build a subset F_m consisting of the 345 labeled fingerprints plus 700 most prevalent non-labeled fingerprints (measured by number of screenshot images per fingerprint) and clustered this subset. The subset F_m represents 187,560 screenshots and samples, respectively. Our goal is to verify the clustering by checking if the non-labeled fingerprints correspond to the labeled fingerprints of the same cluster. Therefore, for each resulting cluster, we inspect at least three of the corresponding screenshot images manually, to see if they relate to the assigned cluster's campaign.

However, before evaluation, the optimum cut-off threshold has to be determined. We use the metrics *precision* and *recall*, widely used for unsupervised machine learning techniques, to measure the clustering accuracy. More precisely, *precision* represents how well our clustering separates fingerprints of different campaigns. *Recall* denotes whether all fingerprints of a certain campaign or family are grouped into the same cluster. In our case, precision is more important than recall, i.e., it is more important to separate fingerprints of two campaigns than to have all fingerprints of one campaign in only one single cluster. In other words, for example, we prefer to have two clusters for one family (or campaign) over aggregating multiple families (or campaigns) into one cluster. For the reasons mentioned in Section 6.2.5, such as different views of the rogue software's user interface, we tolerate multiple clusters per campaign.

Precision and recall are combined into F-measure, as defined in Chapter 2, Section 2.2.1. We place twice as much emphasis on precision over recall for the reasons outlined above and thus use the following *F-measure* to evaluate our clustering with threshold th and $\beta = 1/2$:

$$\text{F-measure}_{th} = (1+\beta^2) \cdot \frac{P_{th} \cdot R_{th}}{\beta^2 \cdot P_{th} + R_{th}} = 1.25 \cdot \frac{P_{th} \cdot R_{th}}{0.25 \cdot P_{th} + R_{th}} \quad (6.1)$$

Using the labeled fingerprints, we estimate the optimum cut-off threshold by clustering with cut-off thresholds in the range 0.1 to 1.0 and a step size of 0.025. As a result, the best cut-off threshold is determined as 0.375 at a weighted F-measure of 93.32%. Figure 6.4 shows the precision, recall and F-measure ($\beta = 1/2$) over the range of evaluated thresholds.

The clustering of the set F_m with cut-off threshold 0.375 results in 51 clusters. 10 of these 51 clusters did not have any labeled fingerprints in the same cluster and were manually inspected. Of the 10 unlabeled clusters, 5 clusters represent

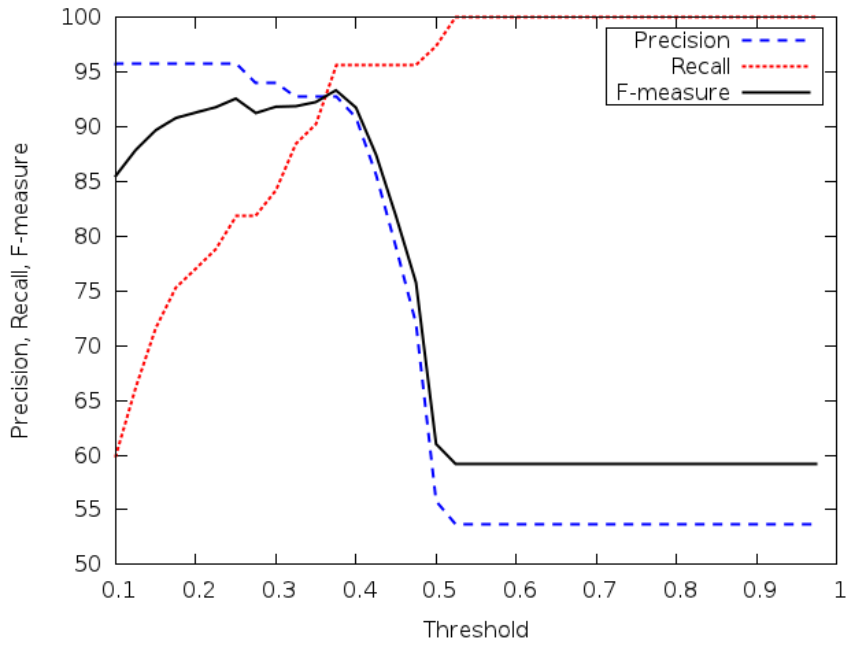


Figure 6.4: Precision, recall and F-measure ($\beta = 1/2$) evaluation of the clustering threshold

previously unseen Fake A/V and ransomware campaigns, shown in Table 6.2. The remaining 5 unlabeled clusters displayed the following distinct user interfaces:

- User Interface of a crack program in order to generate a program serial
- "Run As" dialog, waiting for the user to enter Administrator credentials
- Media Finder Installer
- Firefox, showing the website 3525.com
- Windows Photo Viewer displaying a photo

Note that the benign software, e.g., the Photo viewer as well as the browser, discovered as part of this clustering separates well from the rogue visual malware clusters.

For those clusters that have at least one labeled instance, we assign the label of the first labeled instance to the whole cluster. Note that none of the clusters with labeled instances had more than one distinct label, i.e., no cluster contained conflicts among labeled instances. In addition, for each cluster, we inspect three of the corresponding screenshot images manually, to verify that they relate to the assigned cluster's campaign label. In all cases, the cluster assigned the correct campaign label to the previously unlabeled images.

Perceptual Hash Label	MS A/V Family
Fake A/V and Rogue	
Cloud Protection	undetected
Antivirus Live	FakeSpyprot
Undecipherable	undetected
Ransomware	
GEMA Blocked	undetected
Gendarmerie nationale (FR)	undetected

Table 6.2: Previously unseen campaigns and, if available, Microsoft A/V Family Labels

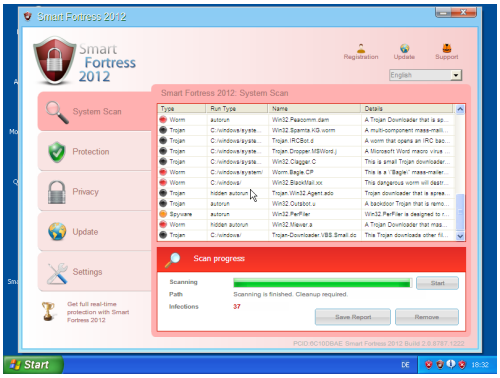
As an example and in order to underline the usefulness of our approach, we used the clusters to enumerate the campaigns that can be attributed to the Winwebsec family. Figure 6.5(a) to 6.5(h) show screenshots of eight Winwebsec campaigns.

To sum up, the clustering phase successfully grouped the set F_m consisting of 700 unlabeled fingerprints and 345 labeled fingerprints, and revealed five previously unseen campaigns. Of these five campaigns, only one (Antivirus Live) was detected by antivirus (FakeSysprot) at the time we received the samples.

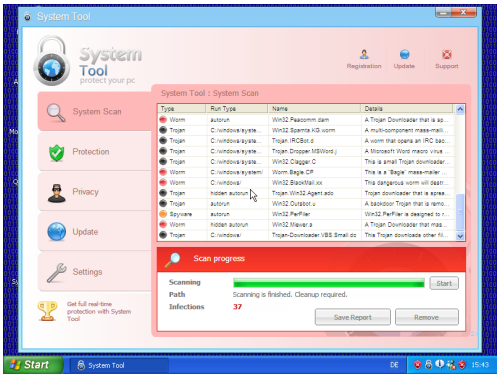
Figure 6.6 shows the dendrogram of the clustering of F_m. If space allowed, we added the campaign label for the cluster (black font color) to the dendrogram. Red font color denotes prevalent non-rogue software clusters such as those displaying an error message caused due to a malformed malware sample, missing libraries (e.g., DotNET) or language support (e.g., Chinese, Japanese and Russian), runtime errors or bluescreen crashes. Clusters labeled as "Blank screen" contain images that did not contain any significant foreground application window, but exhibit a variety of fingerprints because new desktop links have been added or the desktop link icons have been rearranged, resulting in different fingerprints. Blue font color denotes clusters that displayed malware which is not primarily considered Fake A/V or ransomware such as the Hotbar/Clickpotato downloader or installers for various other software.

6.3.2 Antivirus Detection

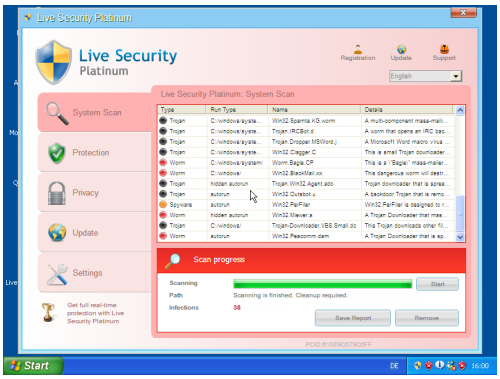
During our manual inspection of the rogue software executions, we noticed that the samples we analyzed exhibited low A/V detection rates. Based on six well-known campaigns, we computed the A/V detection rate of the samples per campaign. For each sample of these campaigns, we queried VirusTotal [vir12] for the associated A/V labels when we received the sample. Typically, samples are already known by VirusTotal when we queried for A/V labels, since we receive samples with a delay of up to one day from our sample providers. Note that even if the sample is known by VirusTotal, this does not imply that at least one



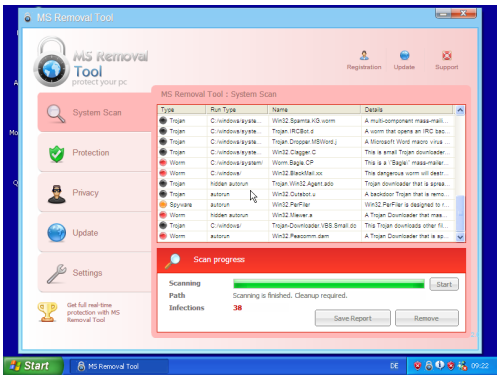
(a) Smart Fortress 2012



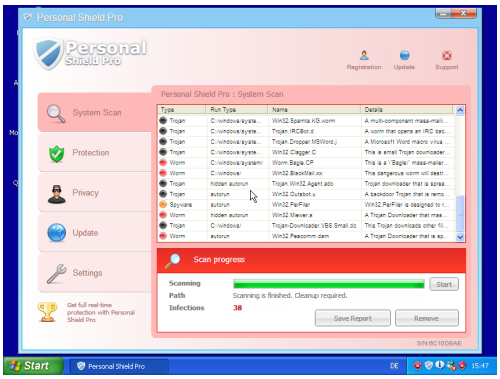
(b) System Tool



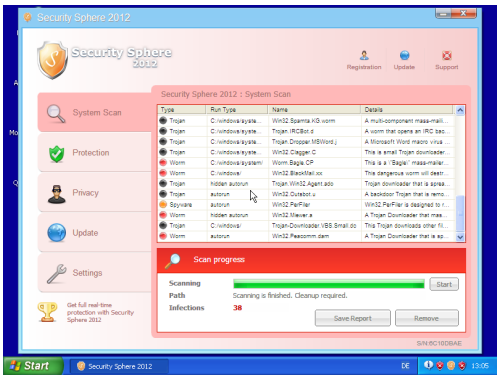
(c) Live Security Platinum



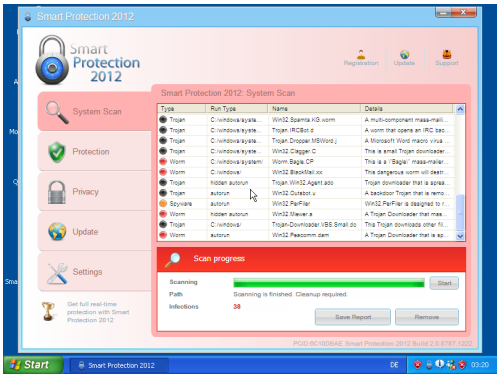
(d) MS Removal Tool



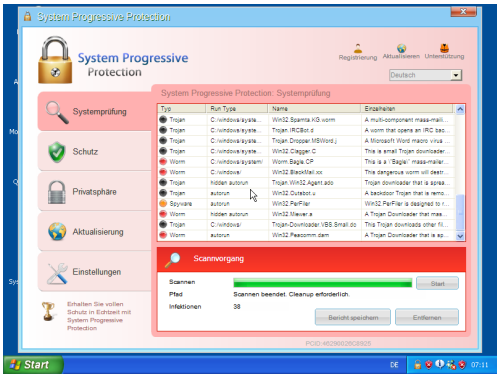
(e) Personal Shield Pro



(f) Security Sphere



(g) Smart Protection



(h) System Progressive Protection

Figure 6.5: Campaigns of the Winwebsec malware family

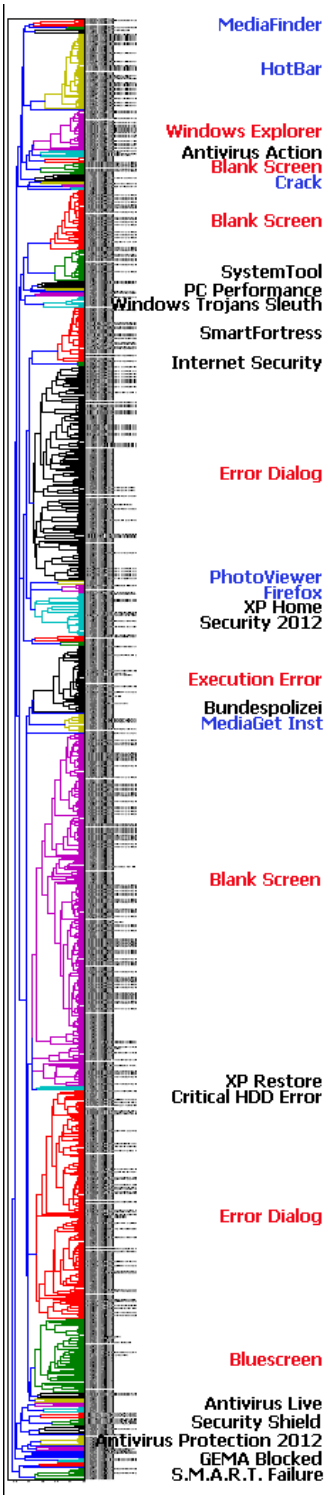


Figure 6.6: Dendrogram excerpt of the clustering of 1045 screenshot fingerprints into 51 clusters with campaign labels for some large clusters. Black font=Fake A/V or ransomware, blue=other software, red=no characteristic appearance.

Campaign Label	#MD5	no AV	Rate
Cloud Protection	737	15	97.96%
FakeRean:InternetSecurity	323	80	75.23%
FakeRean:SpywareProtect.	608	352	42.11%
Winwebsec:SmartFortress	2656	2367	10.88%
Winwebsec:SmartProtect.	139	83	40.29%
Winwebsec:SystemTool	401	175	56.36%

Table 6.3: A/V Detection of six Fake A/V campaigns

A/V vendor detects it. Table 6.3 shows the number of MD5-distinct samples per campaign, as well as the number of MD5-distinct samples without any A/V label. The detection rate is measured as number of distinct sample MD5s with an A/V label over the total number of distinct sample MD5s per campaign.

As shown in Table 6.3, A/V detection rates vary widely among Fake A/V campaigns. Of the Smart Fortress campaign, only about 11% of the samples have been detected by at least one A/V vendor. These results confirm the observation of low A/V detection rates by Rajab et al. [RBM+10] from August 2009 to January 2010, where A/V detection has ranged between 20% and 60% over all Fake A/V samples.

Our experiments have shown that perceptual clustering and classification of rogue software effectively groups associated samples of visual malware. Furthermore, we see our approach as a complementary detection method which, as a last line of defense, can be used to detect rogue software user interfaces on a consumer's computer. However, we leave a detailed analysis of our approach for desktop computer rogue software detection as future work.

6.3.3 Performance

We implemented the computation of the perceptual hash in C++ and the clustering in Python. Albeit not specifically tailored for high performance, this evaluation will give a rough impression of the processing speed of our screenshot clustering approach. The perceptual hash fingerprints were computed for all screenshot images and stored in a PostgreSQL database. Per 10,000 images, the perceptual hash computation takes 20.13 minutes on a single core, including opening and reading the uncompressed image file from disk. This equals to ca. 120 ms per image.

For clustering performance evaluation, we measured the time required to cluster the full set of 17,767 perceptual fingerprints which relate to 213,671 executed malware samples with the parameters determined in Section 6.3.1. In total, without any performance improvements, the clustering of the 17,767 fingerprints takes just over 10 minutes on a commodity computer hardware. All in all, if used

in practice, the clustering is fast enough to allow for periodic re-clustering of the whole set of screenshots, for example once a day.

6.4 Monetization and Localization

While the previous sections outlined our methodology to group Fake A/V and ransomware campaigns, using these results, we will shed some light on the monetization and localization methods.

6.4.1 Ransomware Campaigns

In the following, we restrict ourselves to the four ransomware campaigns. The results are summarized in Table 6.4. The four ransomware campaigns prevent the user from accessing its computer and motivate the user to pay in order to unlock the computer. Three of the four campaigns provided the user an input field for either ukash [uka12] or paysafecard [pay12] codes, while the German GEMA ransomware campaign provided only paysafecard as payment method. To sum up, in all cases, the campaigns' monetization is backed up by pre-paid voucher-like services. We subsume these as prepay payment method because in all cases the user has to buy a voucher before the equivalent amount can be spent by providing the voucher code. The use of prepay methods exposes a significant difference to the payment as can be observed with Fake A/V campaigns which typically offer credit card payment. Stone-Gross et al. [SGAK+11] show that Fake A/V firms even provide refunds in order to avoid credit card chargebacks, because these chargebacks and associated consumer complaints might lead to a cancellation by the credit card payment processor. From the perspective of ransomware miscreants, we speculate several advantages of these prepay payment methods over credit card payment. First, prepay payment is easy-to-use and it only requires one input field for the prepaid voucher code. Second, prepay payment avoids to be dependent on credit card payment processors which are required to act against fraud. In addition, while Fake A/V may be doubtful, ransomware is clearly considered fraud in legislation, which might even prevent ransomware campaigns from finding any cooperating credit card payment processors. Third, prepaid vouchers allow miscreants to exploit users which do not have a credit card at all.

Furthermore, we observed that three of the four campaigns urge the user to pay in a given time span, which was either three days or one day.

While all samples were executed on a virtual machine with the OS localized to Germany and an IP address of the German DFN network[1], the Gendarmerie campaign only provided French texts in the user interface. When executing this sample without Internet connectivity, the user interface remains in French. We

[1] DFN-IP Service G-WiN, AS 680, is typically the ISP for German universities.

Campaign	Language	Amount	Payment	Limit
GEMA Blocked	German	50 EUR	p	none
Gendarmerie nationale	French	200 EUR	u+p	3 days
Bundespolizei	German	100 EUR	u+p	1 day
Windows Security Center	German	100 EUR	u+p	1 day

Table 6.4: Localization and monetization methods of four ransomware campaigns; u=ukash, p=paysafecard

assume that, in this case, the language seems to be hardcoded in the malware binary.

In case of the Bundespolizei and the Windows Security Center campaigns, the sample uses the service `ip2location.com` in order to map the IP address of the infected computer to ISP, country and city. The external IP address, city, region and ISP's AS name are then displayed in the user interface. However, the sample did not adapt the user interface language when confronted with a manipulated geolocation response. In addition, as part of this campaign, successful installs of the Bundespolizei samples are reported to a C&C server using a plaintext HTTP GET request.

For the GEMA campaign, we observed that the user interface contents consists of HTML with all text and images being server-side generated and transferred via plaintext HTTP. This way, the localization of the campaign might be adapted based on the origin of the HTTP requests. Furthermore, we observed that the GEMA ransomware was accompanied by mostly SpyEye malware.

Very recently, in May 2012, a new ransomware campaign emerged [IC3]. The software claims that the Computer Crime & Intellectual Property Section of the U.S. Department of Justice had identified the user of violating United States Federal Law. Although for time reasons, we did not include this family in our analysis, we could at least confirm that this campaign is also backed up by prepay payment methods. In this case, for IP addresses geolocated in the U.S., the campaign uses paysafecard and moneypak [mon].

6.4.2 Fake A/V Campaigns

We analyzed 14 Fake A/V campaigns in order to shed light on their payment process as well as localization. The results are summarized in Table 6.5. While all of the ransomware campaigns rely on prepay payment methods, 5 of the 14 Fake A/V campaigns provided credit card payment. For 9 of the 14 Fake A/V campaigns, the payment process failed because their payment servers were no longer reachable, and thus we could not determine the payment method. However, the fact that we could not find one Fake A/V campaign using prepay payment supports our observation that there is also a clear distinction in payment

methods between ransomware and Fake A/V software. Note that, for ethical reasons, we did not perform any payments ourselves and judged on the accepted credit cards only based on the user interfaces.

When turning to the amounts of the Fake A/V programs, values range from $50 to $90 USD, depending on supposedly different versions of the program. The fact that some campaigns exhibit the same amount scheme suggests that these campaigns might be related. Indeed, in case of the SmartFortress as well as the SmartProtection campaigns, we found that both belong to the Winwebsec malware family. However, we could not find proof that the same organization is behind both campaigns.

Interestingly, while the ransomware campaigns had user interface texts translated to the locale's language, only one of the 14 Fake A/V campaigns of Table 6.5 exhibits user interface texts matching the locale of the OS (German), all others of the Fake A/V campaigns were only in English. In addition, all amounts were given in US dollars for the Fake A/V campaigns. In contrast, all ransomware campaigns had their amounts adapted to the locale's currency (Euro).

6.5 Limitations

Although our approach is based on the inherent property of rogue software to display a user interface, as always, there are some limitations and room for evasion. Targeting the image processing part, i.e., the computation of the perceptual hash function, malware authors could add random noise to the user interface, so that the resulting screenshots differ widely among samples of one campaign. However, since our perceptual hash function depends on low-frequency coordinates, random noise which results in a change in high frequencies will not significantly change the perceptual hash value. In order to modify the perceptual hash value significantly, user interface elements would need to be positioned randomly.

Another line of evasion lies in the resemblance of rogue software user interface with that of legitimate software. So far, as long as user interfaces of rogue software differ from those interfaces of legitimate software, our approach can possibly detect and exploit exactly this difference. If user interfaces no longer visually differ, e.g., because Fake A/V appearance exhibited the same user interface as one of the legitimate antivirus programs, our approach would fail. However, at some point, Fake A/V will always have to provide some kind of payment instruction and processing user interfaces which could still be used to separate from legitimate software.

In addition, when executing the malware samples, none of them was confronted with user interaction. We might have missed some samples which require the user to interact with the system before displaying their Fake A/V or ransomware user interface. However, based on current research on environment-sensitive malware [LKC11], we consider the amount of possibly missed samples to be negligible

Campaign	Language	Amounts	Payment
Antivirus Action	English	$60	n/a
Antivirus Live	English	n/a	n/a
Antivirus Protection	English	3 M: $49.45, 6 M: $59.95, LL: $69.95	n/a
Internet Security	English	n/a	n/a
Cloud Protection	English	$52	VISA / MC
MS Security Essentials	English	$50	n/a
PC Performance	English	$74.95 (light), $84.95 (Prof)	VISA / MC
Personal Shield	English	$1.50 activation + 1 Y: $59.90, 2 Y: $69.95, LL: $83	VISA / MC
Smart Fortress	English	1 Y: $59.95, 2 Y: $69.95, LL: $89.95	VISA / MC
Smart Protection	English	1 Y: $59.95, 2 Y: $69.95, LL: $89.95	VISA / MC
Smart Repair	English	$84.50	VISA / MC
Spyware Protection	English	$60	n/a
XP Antispyware	German	1 Y: $59.95, 2 Y: $69.95, LL: $79.95 + $19.95 Phone Support 24/7	n/a
XP Home Security	English	1 Y: $59.95, 2 Y: $69.95, LL: $79.95	n/a

Table 6.5: Localization and monetization methods of 14 Fake A/V campaigns; M=months, Y=years, LL=lifelong; MC=MasterCard; All amounts in USD

for the evaluation of our approach's feasibility.

6.6 Related Work

While Fake A/V software has been studied before, e.g., in [RBM+10, KZRB11, SGAK+11, CLT+10], to the best of our knowledge, we are the first to propose a clustering and classification approach that targets the visual appearance of rogue software user interfaces. Thus, we not only cover Fake A/V, but provide a much broader approach of visual malware, including ransomware. Furthermore, our approach serves as a means of classifying malware executions of a dynamic malware analysis environment and allows to identify and classify failed executions such as bluescreen crashes, error dialogs or lack of visual appearance (silent background malware).

Related work can be grouped into three trails of research. The technical aspects, such as the distribution and infrastructure of Fake A/V malware has been analyzed by Cova et al. [CLT+10], Rajab et al. [RBM+10] as well as Komili et al. [KZRB11], e.g., by measuring the lifetime and distribution of domains involved in Fake A/V software or analyzing the corresponding server geolocation, DNS and AS information. Especially two results of [RBM+10] inspired the development of our visual clustering approach. First, the infrastructure of Fake A/V software is volatile. Throughout the measurement period of one year, the median lifetime of Fake A/V domains dropped to below one hour [RBM+10]. Thus, relying on network properties such as domains and IP addresses in order to detect Fake A/V software becomes a tremendous effort. Second, Fake A/V malware suffers from decreasing A/V detection rates, even as low as 20% [RBM+10]. Our measurement confirms this, showing that in case of the SmartFortress campaign even only ca. 11% of the samples were detected by A/V (Table 6.3). This demands for a different detection approach such as ours by exploiting the visual appearance of rogue software.

Gazet has analyzed malware programming aspects of early ransomware virii and the use of cryptographic algorithms that were used when encrypting files [Gaz10] on the user's computer. However, the payment process of ransomware has not been addressed before. In addition, we observed that some of the ransomware programs included in our study no longer actually cryptographically encrypt files, but only block users from accessing the computer, typically be means of visually "locking" the full screen.

The second trail of research deals with economical aspects of Fake A/V software. Stone-Gross et al. [SGAK+11] show that more than 130 million dollars have been earned in three monitored Fake A/V businesses. Furthermore, they disclose that Fake A/V villains provide refunds in order to avoid credit card chargebacks and possibly consumer complaints which could result in termination by the credit card payment processor. We complement existing research by our

analysis of ransomware payment properties and by discovering a shift towards prepay payment methods, such as ukash and paysafecard, in all four monitored ransomware campaigns.

The third area of research covers clustering approaches using behavioral features of executed malware samples. Perdisci et al. [PLF10] develop signatures for characteristic pattern elements in HTTP requests by means of clustering. Similarly, Rieck et al. [RSL+10] propose a system to detect malware by models of recurring network traffic payload invariants. Bayer et al. [BMCH+09] provide a broad overview over malware behavior using clustering to avoid skew by aggressively polymorphic malware. Our approach complements existing approaches by mapping visual appearance of rogue software to campaign and malware family clusters. Especially if none of the existing approaches apply, e.g., if a sample does not exhibit network traffic at all, our screenshot clustering approach can still be used. Furthermore, complementary to the existing clustering methods as well as classic A/V, our approach might help in detecting Fake A/V and ransomware.

6.7 Conclusion

In recent years, rogue software has become one of the most prevalent means of monetization for malware. We propose a new approach to cluster and detect rogue software based on its inherent need to display a user interface. Using a perceptual hash function and a hierarchical clustering on a set of 187,560 screenshot images, we successfully identified 25 campaigns, spanning Fake A/V and ransomware. We observed that especially rogue software suffers from very low antivirus detection rates. Four of the five previously unseen campaigns have not been detected as malware by the time we received the samples. While malware authors seem to succeed in evading classic antivirus signatures, our approach helps to avoid undetected rogue software. Furthermore, we have shown that our approach scales to a large set of samples, effectively analyzing 213,671 screenshot images.

Using the results of our clustering approach, we show that the two prevalent classes of visual malware, Fake A/V and ransomware, exhibit distinct payment methods. While Fake A/V campaigns favor credit card payment, ransomware programs use prepay methods. From the perspective of ransomware miscreants, prepay payment methods provide a number of advantages. First, the prepay methods enable ransomware campaigns to avoid cooperation with any other payment companies, such as a credit card payment processor. We speculate that such a dependency on payment processor companies would constitute an unpredictable risk. Second, prepay payment methods are easy to use and to process as they only require one input field for the voucher code and cannot be associated to a person. Some ransomware campaigns even include detailed instructions on how and where to buy the prepay cards. Third, prepay payment allows miscreants to exploit users which do not own a credit card at all, thereby possible reaching a

larger set of victims.

To sum up, we have designed and implemented a large-scale clustering method for rogue visual malware. At the same time, we provide insights into the monetization and localization properties of current rogue visual malware campaigns. Finally, we propose to convey our detection approach to end-user systems in order to complement existing antivirus solutions.

Chapter 7

Conclusion

This thesis addresses the problem of identifying and recognizing remote-controlled malware. First, in Chapter 3 we proposed SANDNET, our dynamic malware analysis environment, which is subsequently used to generate datasets for identification and recognition experiments. Second, we designed and implemented a recognition approach of botnet C&C channels based on traffic analysis features in Chapter 4. Third, in Chapter 5, our case study on Feederbot, a bot with DNS as carrier for its C&C protocol, sheds light on a whole new class of botnets with covert C&C. Yet, our classifier of DNS traffic has proven to detect DNS-based C&C, even in mixed user traffic. Finally, in Chapter 6, we have designed and implemented a detection framework for rogue visual malware, exploiting the malware's need to expose a visual user interface for monetization.

In the following, we will briefly summarize the results of this thesis and outline future work on each of the topics.

7.1 Sandnet

With SANDNET in Chapter 3, we have shown the importance of designing a scalable and reliable contained environment in order to study the behavior of malicious remote-controlled software. Based on SANDNET, we are able to compile sound datasets representing active and diverse remote-controlled malware. These datasets enable us to design and evaluate malware detection methods.

While contained environments like SANDNET provide the basis to understand malware, the enormous growth in terms of MD5-distinct malware binaries rises concerns how researchers can deal with malware in the forthcoming years. In the future, we also expect the diversity of malware to increase which makes it much more important to respect the diversity in the analysis of malware. One trail of research directs towards a pre-selection of representative instances per family, in order to avoid the re-execution of already known malware. With Fore-

cast [NCJK11], Neugschwandtner et al. have proposed an approach of pre-filtering malware binaries based on static analysis in order to streamline dynamic malware analysis. However, an alternative method could be to combine the distribution source of malware binaries with the notion of a malware family. In this way, a family-aware tracking of the distribution of malware binaries could be evaluated, possibly leading to modeling the evolution on a per-family basis.

Furthermore, contained environments constantly face the challenge of remaining undetected for malware binaries. Otherwise, environment-sensitive malware will evade contained environments by either stopping or by exposing a completely different behavior. Therefore, future contained environments may have to try and model legitimate user environments as close as possible, including user interaction. For SANDNET, we have begun to developed herders on bare-metal, i.e., avoiding virtualization which is a possible source for the detection of contained environments for malware. However, spending bare-metal hardware for dynamic malware analysis is hardly able to scale. A more promising direction of research could examine, how hardware virtualization or dedicated hypervisors as well as over-provisioning of resources can disguise dynamic analysis environments. While today's hypervisors are mainly focussed on the execution performance, it might be worthwhile to research how they can be optimized for concealment.

7.2 *CoCoSpot* – Recognition of C&C flows

We have shown how the command and control plane of remote-controlled malware, i.e., botnets, can reliably be identified and recognized using traffic analysis features. However, even today there are already a few families, such as Zeus P2P, which manage to evade our detection approach. In the future, we need to carefully investigate how botmasters design their botnet's C&C in order to subvert our detection approaches and possibly aim for more robust detection methods.

Furthermore, using *CoCoSpot*, we developed means to track the C&C activities of several well-known and novel botnets. Future work may have to deal with a more automatic approach to monitor the C&C activities, especially if the diversity of families increases. While we managed to track 153 botnet families over the last two and a half years, some have already ceased and make room for new botnets, which means that the new C&C tracking models have to be confirmed manually before being able to monitor these botnets. In order to understand and decrypt the C&C protocols, manual effort in terms of reverse engineering is often required.

In addition, not much is known about the operators of botnets except from a few cases of successful prosecution. Future work may deal with the identification of operating groups behind botnets as there may be overlap among several malware families. Such investigations could address questions as to how botmasters choose their C&C server havens. We have shown that there is a separation of du-

ties among malware families in that especially the downloader families take care of distributing other malware. This underlines the increased professionalization in the malware underground economy. If legal conditions allow, one could think of conveying the idea of honeypots and social engineering to this field. For example, a honeypot of a potential C&C server with ideal conditions from a botmaster's perspective could help to get an understanding of how botmasters operate.

Similarly, using the botnet tracking means, the relationships and dependencies among malware families need to be studied. For example, we observed dedicated downloader families, which only aim at distributing other malware binaries. In order to comprehend the dependencies among malware families, we implemented a Flare [Fla09] visualization tool based on the dependency graph. Figure 7.1 shows a screenshot of the dependency graph visualization. In this tool, an analyst picks one of the families on the circle and the relationships among related malware families will be highlighted. A red line connects the family in question with those families that are being distributed by the picked family, while a green line refers to those families that have been witnessed as distributing the picked family. By default, due to alpha blending, blue lines indicate the popularity of the corresponding family by stroke intensity. Even in the default view, where no family is picked, those families that interact with many other families are easily visible. For example, Virut is one of the most active downloaders, as can be seen in the dependency graph visualization. Once picked, Virut's relationships reveal the diversity of families that are being distributed by Virut (red lines). In addition, it becomes obvious that Virut, although being a downloader family itself, is being distributed by several other families (green lines).

Furthermore, in contrast to Virut, the Pushdo family (Figure 7.3), has only been observed to drop the Cutwail spam bot family. However, we observed that Cutwail has also been downloaded by other families, such as Virut – possibly as an effect of the takedown efforts on Pushdo.

Along these lines, it could be target-oriented to address the disruption of the downloader families first, in order to stop the distribution of remote-controlled malware. Furthermore, a technical dependency may indicate an economical relationship between two malware families. According to the principle "follow the money", economical dependencies can be of significant value for investigations and law enforcement.

7.3 Botnets with DNS-based C&C

Taking disguise of botnet command and control channels to the next level, we have discovered Feederbot, a botnet that uses the DNS protocol as carrier for its command and control. With Feederbot being the first of this kind of botnets with covert C&C, we reverse engineered and investigated this botnet in detail, disclosing the techniques employed to hide their encrypted C&C traffic in regular

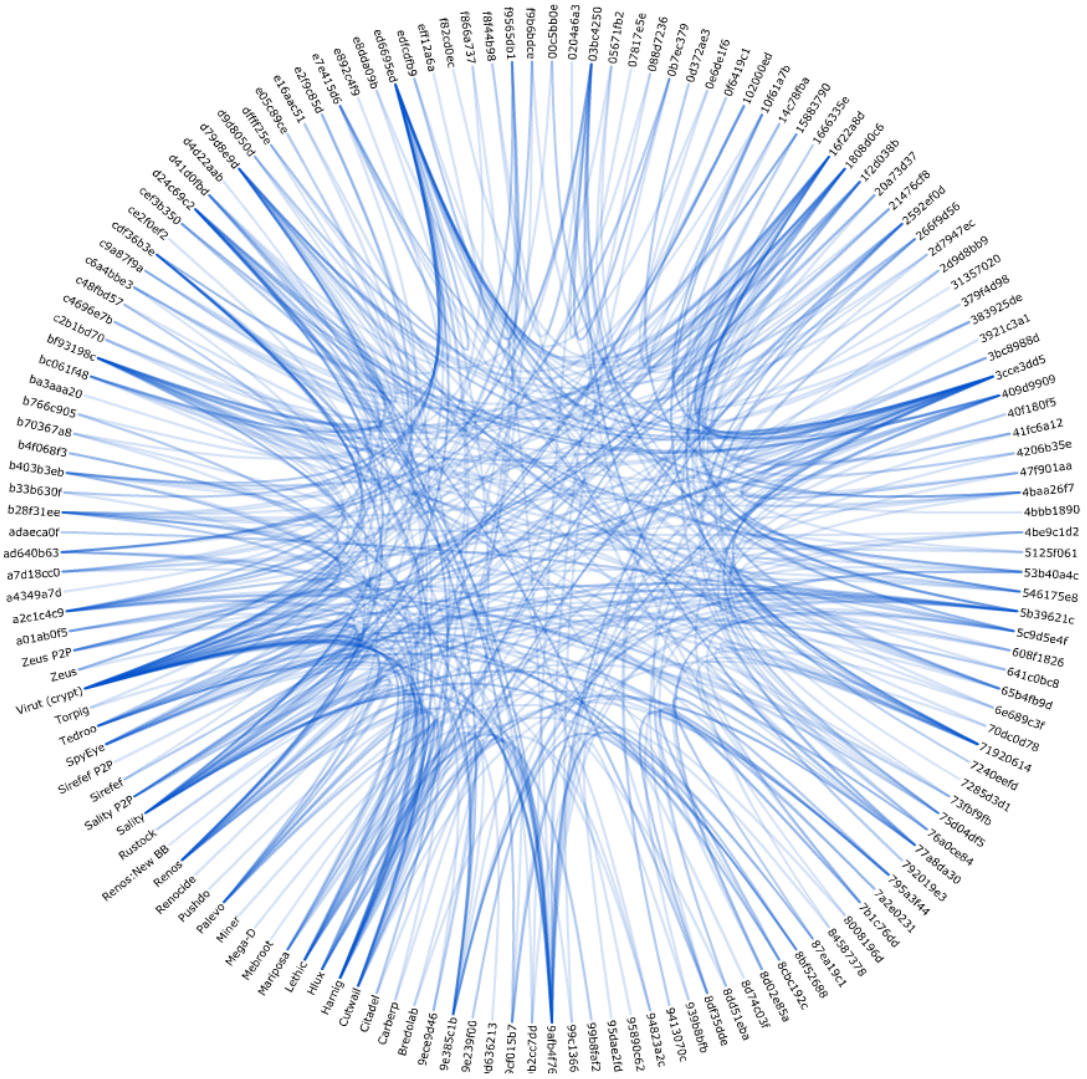


Figure 7.1: Visualization of the botnet family dependency graph

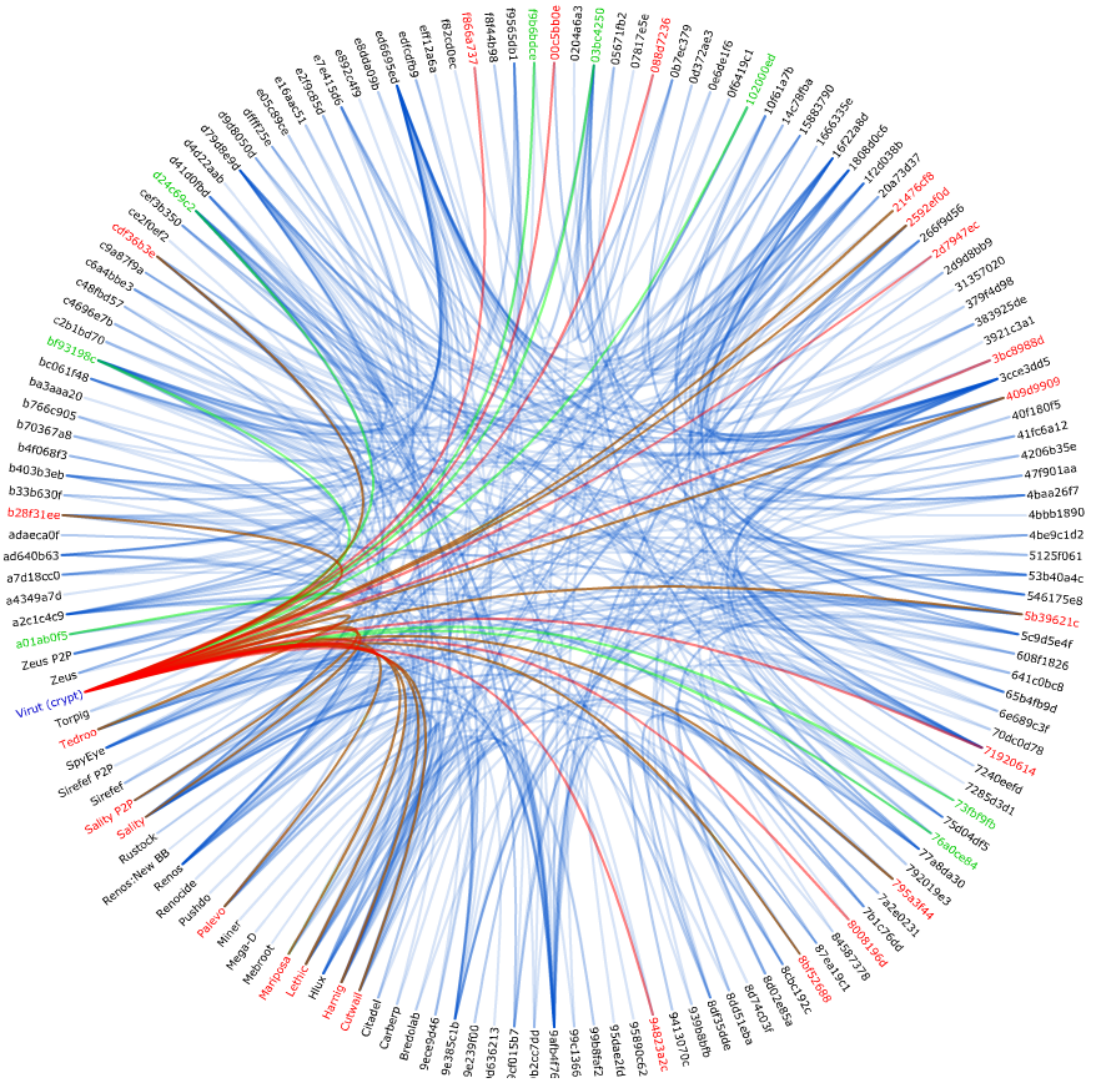


Figure 7.2: Highlighting the relationships of Virut

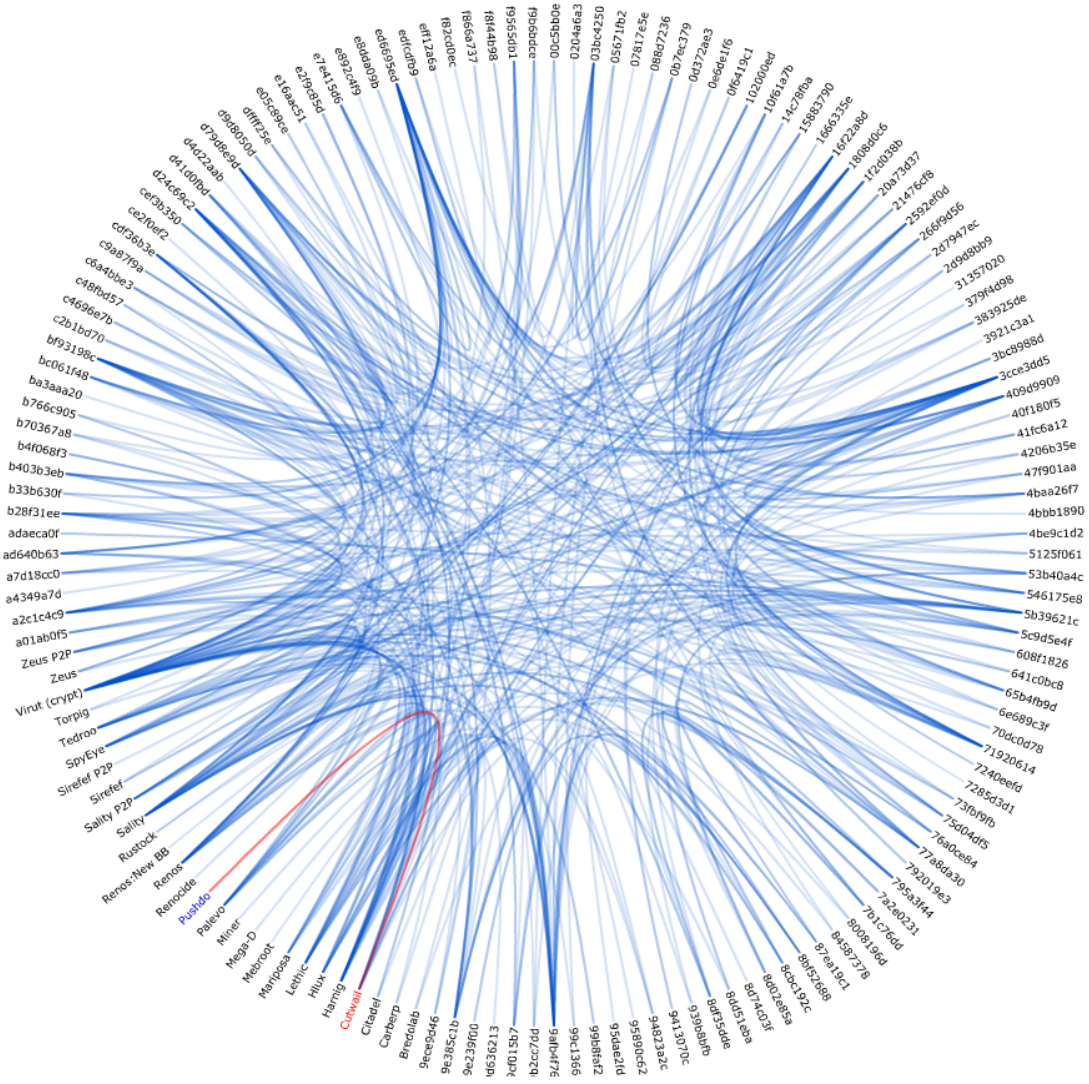


Figure 7.3: Pushdo has been observed to only drop Cutwail

DNS requests and responses.

Furthermore, we designed and implemented a detection approach for botnets with DNS as carrier for their command and control. Based on SANDNET data of a time period of more than 18 months, we show in Chapter 5 that our specifically tailored method can still detect botnets that use DNS as carrier protocol for its C&C. Using our classifier, we have even discovered an independent second botnet that, too, builds its C&C upon the DNS protocol. Finally, we evaluate our detection methodology on mixed network traffic with benign users' network traffic and prove that our classifier can even be used in real-world environments to detect DNS-based botnets.

While a whole body of research exists on detecting botnets with IRC and HTTP as carrier for its command and control, to our knowledge, we are the first to discover DNS-based botnets. In the future, we expect botmasters to increasingly design their C&C protocols to be based on different carrier protocols, in order to evade detection. Future work could examine protocols like NTP or media streaming protocols for Flash as carrier protocols for botnet C&C. Similarly, as described with Renos in Chapter 3, botnets could proceed in the direction of hiding C&C messages in benign-looking contents like images or media content in general.

While purely network-based approaches may have reached their limits concerning the detection of covert C&C channels, one approach for future work may consist in examining and modeling the data processing of media contents retrieved over the network on the system level. Possibly, benign media processing software exhibits a different behavior compared to malicious remote-controlled software. For example, from a high-level perspective, by its very nature, visual media is supposed to be displayed, i.e., destined for the graphical output. However, a software that downloads images, but never displays any of the retrieved image data, may raise suspicion.

7.4 Detection of rogue visual malware

Since botnets have turned to rogue visual monetization techniques by means of Fake A/V or ransomware, this allows for a perceptual recognition. In Chapter 6, we have shown how rogue visual malware can be clustered and recognized by help of perceptual hashing. Especially in face of the very low antivirus detection rates of the respective malware binaries, sometimes even as low as 11% per campaign, our detection methodology may complement existing antivirus detection. As part of future work, we plan to evaluate our visual detection approach in the wild in order to evaluate the detection performance on user systems.

List of Figures

List of Tables

Bibliography

[abu11a] abuse.ch. AMaDa Blocklist. http://amada.abuse.ch/blocklist.php, 2011.

[abu11b] abuse.ch. Palevo Tracker IP Address and Domain Blocklist. http://amada.abuse.ch/palevotracker.php, 2011.

[abu11c] abuse.ch. SpyEye Tracker IP Address and Domain Blocklist. https://spyeyetracker.abuse.ch/blocklist.php, 2011.

[abu11d] abuse.ch. Zeus Tracker IP Address and Domain Blocklist. https://zeustracker.abuse.ch/blocklist.php, 2011.

[Ama12] Amazon Elastic Compute Cloud EC2. http://aws.amazon.com/ec2/, 2012.

[Avt12] AV-Test GmbH: Total Malware per Year. http://www.av-test.org/statistiken/malware/, 2012. Accessed: Nov 11th, 2012.

[BBR+12] Leyla Bilge, Davide Balzarotti, William Robertson, Engin Kirda, and Christopher Kruegel. DISCLOSURE: Detecting Botnet Command and Control Servers Through Large-Scale NetFlow Analysis. In *Annual Computer Security Applications Conference*, 2012.

[Ber08] Dusan Bernát. Domain name system as a memory and communication medium. In *International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, 2008.

[BG] Kenton Born and David Gustafson. Detecting DNS Tunnels Using Character Frequency Analysis. http://arxiv.org/ftp/arxiv/papers/1004/1004.4358.pdf.

[BHB+09] Ulrich Bayer, Imam Habibi, Davide Balzarotti, Engin Kirda, and Christopher Kruegel. A View on Current Malware Behaviors. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2009.

[BKK06] Ulrich Bayer, Christopher Kruegel, and Engin Kirda. TTAnalyze: A Tool for Analyzing Malware. In *16th Annual EICAR Conference, Hamburg, Germany*, April 2006.

[BMCH+09] Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel, and Engin Kirda. Scalable, Behavior-Based Malware Clustering. In *Network and Distributed System Security Symposium (NDSS)*, 2009.

[BOB+10] Hamad Binsalleeh, Thomas Ormerod, Amine Boukhtouta, Prosenjit Sinha, Amr M. Youssef, Mourad Debbabi, and Lingyu Wang. On the analysis of the zeus botnet crimeware toolkit. In *Conference on Privacy, Security and Trust*, 2010.

[Bos12] Richard Domingues Boscovich. Microsoft Disrupts the Emerging Nitol Botnet Being Spread through an Unsecure Supply Chain. `http://blogs.technet.com/b/microsoft_blog/archive/2012/09/13/microsoft-disrupts-the-emerging-nitol-botnet-being-spread-through-an-unsecure-supply-chain.aspx`, 2012.

[Bro11] Seth Bromberger. DNS as a Covert Channel Within Protected Networks, 2011. `http://www.oe.energy.gov/DocumentsandMedia/DNS_Exfiltration_2011-01-01_v1.1.pdf`.

[CCG+10] Chia Yuan Cho, Juan Caballero, Chris Grier, Vern Paxson, and Dawn Song. Insights from the Inside: A View of Botnet Management from Infiltration. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2010.

[CDB09] Joan Calvet, Carlton R. Davis, and Pierre-Marc Bureau. Malware authors don't learn, and that's good! In *International Conference on Malicious and Unwanted Software (MALWARE)*, 2009.

[CDKL09] Julio Canto, Marc Dacier, Engin Kirda, and Corrado Leita. Large scale malware collection: lessons learned. `https://www.seclab.tuwien.ac.at/papers/srds.pdf`, 2009.

[CL07] Ken Chiang and Levi Lloyd. A case study of the rustock rootkit and spam bot. In *Workshop on Hot Topics in Understanding Botnets (HotBots)*, HotBots '07, Berkeley, CA, USA, 2007. USENIX Association.

[CLLK07] Hyunsang Choi, Hanwoo Lee, Heejo Lee, and Hyogon Kim. Botnet Detection by Monitoring Group Activities in DNS Traffic. In *IEEE International Conference on Computer and Information Technology (CIT)*, 2007.

[CLT+10] Marco Cova, Corrado Leita, Olivier Thonnard, Angelos D. Keromytis, and Marc Dacier. An Analysis of Rogue AV Campaigns. In *Recent Advances in Intrusion Detection*, 2010.

[Cor10] Luis Corrons. Mariposa botnet. `http://pandalabs.pandasecurity.com/mariposa-botnet/`, 2010.

[dGFG12] Daan de Graaf, Ahmed F.Shosha, and Pavel Gladyshev. BREDOLAB: Shopping in the Cybercrime Underworld. `http://digitalfire.ucd.ie/wp-content/uploads/2012/10/BREDOLAB-Shopping-in-the-Cybercrime-Underworld.pdf`, 2012.

[Dod06] Yadolah Dodge. *The Oxford Dictionary of Statistical Terms*. 2006.

[DRF+11] Christian J. Dietrich, Christian Rossow, Felix C. Freiling, Herbert Bos, Maarten van Steen, and Norbert Pohlmann. On Botnets that use DNS for Command and Control. In *Proceedings of European Conference on Computer Network Defense*, 2011.

[DRP12a] Christian J. Dietrich, Christian Rossow, and Norbert Pohlmann. *CoCoSpot*: Clustering and recognizing botnet command and control channels using traffic analysis. In *A Special Issue of Computer Networks On Botnet Activity: Analysis, Detection and Shutdown*, 2012.

[DRP12b] Christian J. Dietrich, Christian Rossow, and Norbert Pohlmann. eID Online Authentication Network Threat Model, Attacks and Implications. In *19th DFN Workshop 2012*, 2012.

[DRP13] Christian J. Dietrich, Christian Rossow, and Norbert Pohlmann. Exploiting Visual Appearance to Cluster and Detect Rogue Software. In *ACM Symposium On Applied Computing*, 2013.

[DRSL08] Artem Dinaburg, Paul Royal, Monirul I. Sharif, and Wenke Lee. Ether: malware analysis via hardware virtualization extensions. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM Conference on Computer and Communications Security*, pages 51–62. ACM, 2008.

[Fal11] Nicolas Falliere. Sality: Story of a Peer-to-Peer Viral Network. `http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/sality_peer_to_peer_viral_network.pdf`, 2011.

[FHW05] Felix C. Freiling, Thorsten Holz, and Georg Wicherski. Botnet tracking: Exploring a root-cause methodology to prevent distributed denial-of-service attacks. In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, *European Symposium on Research in Computer Security (ESORICS)*, volume 3679 of *Lecture Notes in Computer Science*, pages 319–335. Springer, 2005.

[Fla09] Flare. `http://flare.prefuse.org`, 2009.

[Gal11] Sean Gallagher. How the most massive botnet scam ever made millions for Estonian hackers. `http://arstechnica.com/tech-policy/2011/11/how-the-most-massive-botnet-scam-ever-made-millions-for-estonian-hackers/`, 2011.

[Gaz10] Alexandre Gazet. Comparative analysis of various ransomware virii. *Journal in Computer Virology*, 6(1):77–90, 2010.

[GBC+12] Chris Grier, Lucas Ballard, Juan Caballero, Neha Chachra, Christian J. Dietrich, Kirill Levchenko, Panayiotis Mavrommatis, Damon McCoy, Antonio Nappa, Andreas Pitsillidis, Niels Provos, M. Zubair Rafique, Moheeb Abu Rajab, Christian Rossow, Kurt Thomas, Vern Paxson, Stefan Savage, and Geoffrey M. Voelker. Manufacturing Compromise: The Emergence of Exploit-as-a-Service. In *Proceedings of the 19th ACM Conference on Computer and Communication Security*, Raleigh, NC, October 2012.

[GFC08] Fanglu Guo, Peter Ferrie, and Tzi-Cker Chiueh. A Study of the Packer Problem and Its Solutions. In Richard Lippmann, Engin Kirda, and Ari Trachtenberg, editors, *Recent Advances in Intrusion Detection (RAID)*, volume 5230 of *Lecture Notes in Computer Science*, pages 98–115. Springer, 2008.

[GH07] Jan Goebel and Thorsten Holz. Rishi: Identify Bot Contaminated Hosts by IRC Nickname Evaluation. In *USENIX Workshop on Hot Topics in Understanding Botnets (HotBots)*, 2007.

[Goo11] Google. Safe Browsing API. `http://code.google.com/apis/safebrowsing/`, 2011.

[GPZL08] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. In *17th USENIX Security Symposium, San Jose, CA*, August 2008.

[GR10] Sergey Golovanov and Vyacheslav Rusakov. TDSS. `http://www.securelist.com/en/analysis/204792131/TDSS`, 2010.

[GYP+09] Guofei Gu, Vinod Yegneswaran, Phillip A. Porras, Jennifer Stoll, and Wenke Lee. Active botnet probing to identify obscure command and control channels. In *Annual Computer Security Applications Conference (ACSAC)*. IEEE Computer Society, 2009.

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

[IC3] Internet Crime Complaint Center IC3. Citadel Malware Delivers Reveton Ransomware in Attempts to Extort Money. `http://www.ic3.gov/media/2012/120530.aspx`.

[ipo11] ipoque.com. OpenDPI. `http://www.opendpi.org/`, 2011.

[JHKH11] Gregoire Jacob, Ralf Hund, Christopher Kruegel, and Thorsten Holz. JACKSTRAWS: Picking Command and Control Connections from Bot Traffic. In *USENIX Security Symposium 2011*, 2011.

[KG09] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In *International Conference on Computer Vision (ICCV)*. IEEE, 2009.

[KKL+09] Chris Kanich, Christian Kreibich, Kirill Levchenko, Brandon Enright, Geoffrey M. Voelker, Vern Paxson, and Stefan Savage. Spamalytics: an empirical analysis of spam marketing conversion. *Communications of the ACM*, 52(9):99–107, 2009.

[KZRB11] Onur Komili, Kyle Zeeuwen, Matei Ripeanu, and Konstantin Beznosov. Stragegies for Monitoring Fake AV Distribution Networks. In *Virus Bulletin*, 2011.

[Lev65] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii Nauk SSSR*, 163, 1965.

[Lis09] Malware Domain List. `www.malwaredomainlist.com`, 2009.

[LKC11] Martina Lindorfer, Clemens Kolbitsch, and Paolo Milani Comparetti. Detecting environment-sensitive malware. In *Recent Advances in Intrusion Detection (RAID)*, 2011.

[LW71] Michael Levandowsky and David Winter. Distance between sets. *Nature*, 234, 1971.

[McA12] McAfee. Threats report: Second quarter 2012. `http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q2-2012.pdf`, 2012.

[MCJ07] Lorenzo Martignoni, Mihai Christodorescu, and Somesh Jha. Omniunpack: Fast, generic, and safe unpacking of malware. In *Annual Computer Security Applications Conference (ACSAC)*, pages 431–441. IEEE Computer Society, 2007.

[Mic] Microsoft Security - Watch out for fake virus alerts. `http://www.microsoft.com/security/pc-security/antivirus-rogue.aspx`.

[mon12] Green Dot Corporation - MoneyPak, 2012. `https://www.moneypak.com`.

[Mus09] Atif Mushtaq. Smashing the Mega-D/Ozdok botnet in 24 hours. `http://blog.fireeye.com/research/2009/11/smashing-the-ozdok.html`, 2009.

[Mus12] Atif Mushtaq. Grum, world's third-largest botnet, knocked down. `http://blog.fireeye.com/research/2012/07/grum-botnet-no-longer-safe-havens.html`, 2012.

[Naz] Jose Nazario. BlackEnergy DDoS Bot Analysis. `http://sites.google.com/site/evelynnjou/BlackEnergyDDoSBotAnalysis.pdf`.

[NCJK11] Matthias Neugschwandtner, Paolo Milani Comparetti, Grégoire Jacob, and Christopher Kruegel. Forecast: skimming off the malware cream. In *Annual Computer Security Applications Conference*, pages 11–20, 2011.

[NMH+10] Shishir Nagaraja, Prateek Mittal, Chi-Yao Hong, Matthew Caesar, and Nikita Borisov. Botgrep: Finding p2p bots with structured graph analysis. In *USENIX Security Symposium*, pages 95–110. USENIX Association, 2010.

[Onl09] BBC News Online. Clock ticking on worm attack code. `http://news.bbc.co.uk/2/hi/technology/7832652.stm`, 2009.

[pay12] paysafecard, 2012. `http://www.paysafecard.com`.

[PGP12] Daniel Plohmann and Elmar Gerhards-Padilla. A Case Study on the Miner Botnet. In *4th International Conference on Cyber Conflict (CyCon'2012), Tallinn, Estonia*, 2012.

[Phi11] PhishTank. Phish Archive. `http://www.phishtank.com/`, 2011.

[PLF10] Roberto Perdisci, Wenke Lee, and Nick Feamster. Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2010.

[Rad11] Marian Radu. Win32/Renocide, the aftermath. `http://blogs.technet.com/b/mmpc/archive/2011/03/16/win32-renocide-the-aftermath.aspx`, 2011.

[RBM+10] Moheeb Abu Rajab, Lucas Ballard, Panayiotis Mavrommatis, Niels Provos, and Xin Zhao. The Nocebo Effect on theWeb: An Analysis of Fake Anti-Virus Distribution. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2010.

[RDB+11] Christian Rossow, Christian J. Dietrich, Herbert Bos, Lorenzo Cavallaro, Maarten van Steen, Felix C. Freiling, and Norbert Pohlmann. Sandnet: Network traffic analysis of malicious software. In *Building Analysis Datasets and Gathering Experience Returns for Security*, 2011.

[RDB12] Christian Rossow, Christian J. Dietrich, and Herbert Bos. Large-Scale Analysis of Malware Downloaders . In *Proceedings of the 9th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA '12)* , Heraklion, Greece, July 2012.

[RDK+12] Christian Rossow, Christian J. Dietrich, Christian Kreibich, Chris Grier, Vern Paxson, Norbert Pohlmann, Herbert Bos, and Maarten van Steen. Prudent Practices for Designing Malware Experiments: Status Quo and Outlook. In *Proceedings of the 33rd IEEE Symposium on Security and Privacy (S&P)*, San Francisco, CA, May 2012.

[Reg09] The Register. Royal Navy warships lose email in virus infection. `http://www.theregister.co.uk/2009/01/15/royal_navy_email_virus_outage/`, 2009.

[Rie09] Konrad Rieck. *Machine learning for application-layer intrusion detection.* PhD thesis, 2009.

[ROL+10] Marco Riccardi, David Oro, Jesus Luna, Marco Cremonini, and Marc Vilanova. A Framework For Financial Botnet Analysis. In *eCrime Researchers Summit*, 2010.

[RSL+10] Konrad Rieck, Guido Schwenk, Tobias Limmer, Thorsten Holz, and Pavel Laskov. Botzilla: Detecting the "Phoning Home" of Malicious Software. In *ACM Symposium On Applied Computing*, 2010.

[SBBD10] Prosenjit Sinha, Amine Boukhtouta, Victor Heber Belarde, and Mourad Debbabi. Insights from the analysis of the mariposa botnet. In *CRiSIS*, pages 1–9. IEEE, 2010.

[SCC+09] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your Botnet is My Botnet: Analysis of a Botnet Takeover. In *ACM Conference on Computer and Communications Security*, 2009.

[SEB12] Aditya K. Sooda, Richard J. Enbodya, and Rohit Bansal. Dissecting SpyEye – Understanding the design of third generation botnets. In *A Special Issue of Computer Networks On Botnet Activity: Analysis, Detection and Shutdown*, 2012.

[SGAK+11] Brett Stone-Gross, Ryan Abman, Richard A. Kemmerer, Christopher Kruegel, Douglas G. Steigerwald, and Giovanni Vigna. The Underground Economy of Fake Antivirus Software. In *10th Workshop on Economics of Information Security (WEIS)*, 2011.

[SGE+09] Ben Stock, Jan Göbel, Markus Engelberth, Felix C. Freiling, and Thorsten Holz. Walowdac – Analysis of a Peer-to-Peer Botnet. In *European Conference on Computer Network Defense (EC2ND)*, pages 13–20, November 2009.

[SGHSV11] Brett Stone-Gross, Thorsten Holz, Gianluca Stringhini, and Giovanni Vigna. The Underground Economy of Spam: A Botmaster's Perspective of Coordinating Large-Scale Spam Campaigns. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2011.

[SGKA+09] Brett Stone-Gross, Christopher Kruegel, Kevin Almeroth, Andreas Moser, and Engin Kirda. FIRE: FInding Rogue nEtworks. In *Annual Computer Security Applications Conference (ACSAC)*, 2009.

[SGRL12] Seungwon Shin, Guofei Gu, Narasimha Reddy, and Christopher P. Lee. A large-scale empirical study of conficker. *IEEE Transactions on Information Forensics and Security*, 7(2):676–690, 2012.

[SHSG+11] Gianluca Stringhini, Thorsten Holz, Brett Stone-Gross, Christopher Kruegel, and Giovanni Vigna. BOTMAGNIFIER: Locating Spambots on the Internet. In *USENIX Security Symposium*. USENIX Association, 2011.

[SLWL08] W. Timothy Strayer, David E. Lapsley, Robert Walsh, and Carl Livadas. Botnet detection based on network behavior. In *Advances in Information Security: Botnet Detection*, pages 1–24. 2008.

[Spa11] Spamhaus. XBL. http://www.spamhaus.org/xbl/, 2011.

[Sto74] M. Stone. Cross-Validatory Choice and Assessment of Statistical Predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2):111–147, 1974.

[Tel09] The Telegraph. French fighter planes grounded by computer virus. http://www.telegraph.co.uk/news/worldnews/europe/france/4547649/French-fighter-planes-grounded-by-computer-virus.html, 2009.

[Thr11] Emerging Threats. Snort Ruleset. http://www.emergingthreats.net/, 2011.

[TN10] Kurt Thomas and David M. Nicol. The Koobface Botnet and the Rise of Social Malware. In *5th International Conference on Malicious and Unwanted Software (MALWARE) 2010*, 2010.

[uka12] uKash, 2012. http://www.ukash.com.

[UPI09] UPI. Virus strikes 19 million PCs. http://www.upi.com/Top_News/2009/01/25/Virus_strikes_15_million_PCs/UPI-19421232924206, 2009.

[Vap95] Vladimir N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[Vil10] Nart Villeneuve. Koobface: Inside a crimeware network, 2010.

[vir12] Hispasec Sistemas - VirusTotal, 2012. http://www.virustotal.com/.

[vR79] C. J. van Rijsbergen. *Information Retrieval*. Butterworth, London, 1979.

[Wer11a] Tillmann Werner. Botnet Shutdown Success Story: How Kaspersky Lab Disabled the Hlux/Kelihos Botnet. http://goo.gl/SznzF, 2011.

[Wer11b] Tillmann Werner. The Miner Botnet: Bitcoin Mining Goes Peer-To-Peer. http://goo.gl/NSXnl, 2011.

[WHF07] Carsten Willems, Thorsten Holz, and Felix C. Freiling. Toward Automated Dynamic Malware Analysis Using CWSandbox. *IEEE Security & Privacy*, 5(2):32–39, 2007.

[Wil10] Jeff Williams. Bredolab Takedown, Another Win for Collaboration. http://blogs.technet.com/b/mmpc/archive/2010/10/26/bredolab-takedown-another-win-for-collaboration.aspx, 2010.

[Wil11] Jeff Williams. Operation b107 – Rustock Botnet Takedown. http://blogs.technet.com/b/mmpc/archive/2011/03/18/operation-b107-rustock-botnet-takedown.aspx, 2011.

[Zau10] Christoph Zauner. Implementation and Benchmarking of Perceptual Hash Functions. Master's thesis, Upper Austria University of Applied Sciences, Hagenberg Campus, 2010.