VRIJE UNIVERSITEIT

Using Malware Analysis to Evaluate Botnet Resilience

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. L.M. Bouter
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de Faculteit der Exacte Wetenschappen
op dinsdag 23 april 2013 om 11.45 uur
in de aula van de universiteit,
De Boelelaan 1105

door

Christian Rossow

geboren te Herten, Duitsland

promotoren:	prof.dr.ir. H.J. Bos
	prof.dr.ir. M.R. van Steen

Front page illustration: The graph shows a snapshot of the Zeus P2P botnet, drawn after a first sinkholing initiative in April 2012. Gray nodes represent Zeus P2P bots and the blue nodes in the middle of the graph are the sinkholing servers that we used during our experiments. Although most bots know the sinkholes, not all the bots are perfectly isolated, illustrating the difficulties in sinkholing peer-to-peer botnets (see Chapter 6).

"A man should look for what is, and not for what he thinks should be."
Albert Einstein (1879 – 1955)

Contents

Acknowledgements

This work would not have been possible without the support of many people, who I gratefully thank for their help.

First, I thank my supervisors Maarten van Steen and Herbert Bos for their steady support during my PhD studies. You transformed me to a real researcher by teaching me the academic life, and you served as a career mentor multiple times. Also thank you for all your trust in my sometimes spontaneous and transient research ideas, and for participating in our – as I think very successful – geographically distributed PhD experiment.

Maarten, although you are not from the IT security domain, I have always appreciated your supervision. Your open mind made me forget about your distributed systems background and you have taught me many important academic lessons. I highly appreciate that you enabled me to become a PhD student at the VU even though I did not to move to Amsterdam for my studies.

Herbert, also thanks for your supervision. Not only you greatly helped me on the subject, but you also opened numerous new contacts for me, so that I enjoyed great internships and collaborations during my PhD studies. Also thanks for your trust in me to supervise many of your Master students, with whom I worked on awesome projects.

I also thank the members of my PhD committee, who patiently reviewed this manifest and sent me very constructive feedback. Your thoughts, questions and recommendations brought this PhD thesis to this final state.

Being a close colleague for many years, I would also like to thank Christian J. Dietrich for the fruitful collaborations, our inspiring discussions, his honest feedback and mental support. It was a really enjoyable time to have you as a PhD fellow and I will never forget it.

My dual-homed research was only possible as prof. Norbert Pohlmann granted me a large degree of freedom at the Institute for Internet Security in Gelsenkirchen. Your contacts made it possible to acquire the numerous malware projects I have worked on. Thank you for your trust and support during all these years (not to forget the enjoyable foosball table).

A big thank you to all co-authors that worked with me on papers, in particular, Chris Grier, Christian Kreibich, Tillmann Werner, Dennis Andriesse, Daniel

Plohmann and Brett Stone-Gross – it was really a pleasure to work with you.

Christian J. Dietrich, Thomas Czerwinski, Dennis Fricke, Christian Nordlohne, Arne Welzel, thanks for your contributions to Sandnet. Since Sandnet is worth nothing without the malware feeds, my sincere thanks to all the anonymous malware sample providers. Thanks to SURFnet and The Shadowserver Foundation for their friendly support in our peer-to-peer botnet work. And one part that is often forgotten, thanks to the open-source community for all the programs and support, particularly for PostgreSQL and VirtualBox. Thank you, Arno, for sharing your desk at the VU. Asia, your advice and especially the LaTeX template was very helpful. Jacopo, I hope that your warming words will turn out to become true in the end. Remco, thanks for your help with the Samenvatting that unfortunately nobody will ever read. All other colleagues, thanks for your support and input for discussion.

I enjoyed internships at two great places during my PhD studies. Paolo Milani Comparetti accepted me as an intern at the TU Vienna, inspiring some of my ideas for the Sandnet project – thank you! Thank you, Vern Paxson, Christian Kreibich and Chris Grier, for hosting me at ICSI in Berkeley during the summer 2011. It was a totally awesome and successful time (not only climbing-wise) and I will never forget your lovely city and the awesome food and beers.

But my biggest thank you goes to my girlfriend Kerstin. You have always tolerated my physical or psychological absence during many weekends and evenings. It must have been hard sometimes to believe that you are more important to me than the fight against malware, but trust me, you are. You cheered me up in times where I lost hope and, luckily, you often kidnapped me from work whenever I just had my research in mind. I am very proud of you.

Finally, I thank all my friends, my and Kerstin's family, particularly for their mental motivation. It was motivating to know so many people that are proud of me, and at the same time, people that care about my private life. Although you may not have succeeded in all occasions, you often reminded me that there are other important things in life.

Publications

The research presented in this thesis was also published in the following peer-reviewed conference or workshop proceedings:

- Christian Rossow, Dennis Andriesse, Tillmann Werner, Brett Stone-Gross, Daniel Plohmann, Christian J. Dietrich, Herbert Bos. “P2PWNED: Modeling and Evaluating the Resilience of Peer-to-Peer Botnets”. In Proceedings of the 34th IEEE Symposium on Security and Privacy, IEEE S&P 2013, San Francisco, CA. (Chapter 6)
- Christian Rossow, Christian J. Dietrich, Herbert Bos. “Large-Scale Analysis of Malware Downloaders”. In Proceedings of the 9th Conference on Detection of Intrusions and Malware & Vulnerability Assessment, DIMVA 2012, Heraklion, Greece. (Chapter 5)
- Christian Rossow, Christian J. Dietrich, Christian Kreibich, Chris Grier, Vern Paxson, Norbert Pohlmann, Herbert Bos, Maarten van Steen. “Prudent Practices for Designing Malware Experiments: Status Quo and Outlook”. In Proceedings of the 33rd IEEE Symposium on Security and Privacy, IEEE S&P 2012, San Francisco, CA. (Chapter 3)
- Christian Rossow, Christian J. Dietrich, Herbert Bos, Lorenzo Cavallaro, Maarten van Steen, Felix C. Freiling, Norbert Pohlmann. “SANDNET: Network Traffic Analysis of Malicious Software”. In Proceedings of the ACM Workshop on Building Analysis Datasets and Gathering Experience Returns for Security, ACM BADGERS 2011, Salzburg, Austria. (Chapter 4)

The work presented in this thesis also fostered other lines of research, such as malware detection and software exploit analysis. The following publications base on the systems and results established in this thesis:

- Christian J. Dietrich, Christian Rossow, Norbert Pohlmann. "Exploiting Visual Appearance to Cluster and Detect Rogue Software". In Proceedings of the 28th Symposium On Applied Computing, ACM SAC 2013, Coimbra, Portugal.
- Chris Grier, Lucas Ballard, Juan Caballero, Neha Chachra, Christian J. Dietrich, Kirill Levchenko, Panayiotis Mavrommatis, Damon McCoy, Antonio Nappa, Andreas Pitsillidis, Niels Provos, Zubair Rafique, Moheeb Abu Rajab, Christian Rossow, Kurt Thomas, Vern Paxson, Stefan Savage, Geoffrey M. Voelker (alphabetical order). "Manufacturing Compromise: The Emergence of Exploit-as-a-Service". In Proceedings of the 19th ACM Conference on Computer and Communications Security, ACM CCS 2012, Raleigh, NC, USA.
- Christian J. Dietrich, Christian Rossow, Felix C. Freiling, Herbert Bos, Maarten van Steen, Norbert Pohlmann. "On Botnets that use DNS for Command and Control". In Proceedings of the European Conference on Computer Network Defense, EC2ND 2011, Gothenburg, Sweden.

1 Introduction

1.1 Problem Definition

In 1986, the first-ever PC virus (c)Brain appeared, spreading via FAT-formatted floppy disks. Since then, the motivation behind developing malicious software (*malware*) shifted from fun/honor to a profit-oriented market organized in underground communities. The monetization techniques of today's malware are manifold, ranging from illegitimate product advertisements, extortion, personal data theft, system resources abuse, and markets for selling or renting infected machines. Independent from the actual techniques used, malware evolved far beyond simplistic and isolated pieces of software. As a side-effect of this development, today's malware requires, or at least strongly depends on, *infrastructures* to *command and control* (C&C) the malware.

From a security perspective, these C&C infrastructures represent a possible attack vector to mitigate the damage of malware. Consider, for example, a spambot that receives a regular feed of spam templates from a central C&C server. In such a case, the *resilience* of the spamming botnet heavily depends on the availability of this particular C&C server. If defenders could control or terminate this particular server, the operation of the botnet would cease. Although the PCs would actually remain infected, in this trivialized scenario, the infection effects are minimized.

As such, defenders have taken efforts to identify and publish lists of C&C end points. Some C&C servers were taken offline due to public pressure of this kind. Similarly, defenders disrupted individual botnets in immensely tedious initiatives spanning both technical and legal entities. Despite these small successes, though, the botnet problem remains and there are botnets that have been operational for many years. While anecdotes about the resilience of botnets exist, only little research has enlightened this problem domain. As a consequence, our community lacks a profound understanding of the *reasons* for botnet resilience. In this thesis, we focus on two particularly problematic fields that need further research.

First, it is largely unknown how attackers install malicious binaries on freshly infected systems. This process, however, is an important part in the lifecycle of botnets: it represents a major way of adding new bots to an existing botnet. A separation of tasks in the underground community fostered specialized and well-organized groups that take care of the malware installation business. Therefore, to understand the resilience of malware, we also have to analyze the programs, techniques and infrastructures used for the malware installation process. Only then can we understand how counter-measures can effectively and efficiently thwart the continuous problem of botnets.

Second, once botnets are operating, botmasters do not want their networks to be disrupted. To avoid single point of failures in botnet infrastructure, attackers started using resilience schemes such as distributing redundant C&C servers via DNS. These schemes significantly harden a botnet's C&C infrastructures, but still defenders can attack such botnets by shutting down C&C domains. Unfortunately, these defensive techniques generally do not work for peer-to-peer botnets. With only a few exceptions (such as the Storm botnet), little is known about the actual resilience of peer-to-peer botnets, and we lack a terminology for and measurements of peer-to-peer botnet resilience.

One way to tackle the two aforementioned problems is the use of static and dynamic malware analysis. Malware analysis has provided valuable insights into malware behavior. However, when using malware analysis techniques, one needs to take care of some idiosyncrasies. For example, when dynamically analyzing malware, we must ensure a realistic analysis environment that attackers cannot easily circumvent. At the same time, to mitigate harm to others, we must not allow unrestricted Internet access to the malware under analysis. A third problem and one we discuss first in this thesis, is the lack of best practices or guidelines for prudent malware experimentation.

1.2 Research Questions

This work seeks to explore the resilience of botnets. However, we cannot analyze resilience without the ability to perform malware experiments. Unfortunately, performing *sound* malware experimentation itself is challenging and little guidance exists for such experiments. There is a trade-off between realistic malware analysis and the risk of potential harm during malware analysis. Thus, as a first research question, we explore how we can best use dynamic malware analysis to monitor and analyze malware resilience:

Research Question 1. *How can a dynamic malware analysis platform be designed that allows for realistic and safe experiments? How can we ensure scientifically sound use of malware analysis datasets?*

When this question is answered, we can use the resulting malware analysis system to measure and evaluate malware resilience. To do so, we first explore the support infrastructures that are used to install malware. Botnets using these infrastructures are responsible for a large fraction of the real-world malware installations. Resilient malware installation infrastructures enable attackers to install and operate botnets that have even weak resilience designs. Therefore, our next research question addresses the root cause for many malware families:

Research Question 2. *Which techniques and infrastructures do attackers use for installing malware on systems that were recently exploited? What causes the resilience of these malware networks?*

Finally, we turn to the resilience of the botnets themselves. Once the malware is installed on the system, botmasters will use their C&C infrastructures to control the bots. From an attacker's perspective, one way to mitigate the resilience deficits of centralized botnet designs is the use of peer-to-peer mechanisms. Our third research question addresses the analysis of botnets with distributed C&C mechanisms:

Research Question 3. *How are botnets designed that are largely independent from centralized C&C infrastructures? Can we systematically describe the defensive methods to disrupt peer-to-peer botnets, and how effective are such techniques?*

1.3 Contributions

The contributions of this thesis can be summarized as follows.

- We develop guidelines that help to use malware analysis in realistic, correct, transparent and safe research experiments. We propose a dynamic analysis system called Sandnet, which suits our needs for analyzing botnet resilience following these guidelines. In addition, we evaluate the importance of our guidelines by running real-world experiments and surveying experimental descriptions in the literature.

- We perform the first long-term and large-scale analysis of 23 malware downloader families, shedding light onto the techniques, protocols and infrastructures that attackers use during the malware installation process. We extend Sandnet to re-analyze these malware downloaders periodically and propose two generic techniques to acquire malware binaries by abusing malware downloaders.

- We analyze the resilience of both historic and existing peer-to-peer (P2P) botnets. We propose a systematization for P2P botnet resilience analysis. We discuss the P2P botnet resilience against reconnaissance measures, such as identifying all infected hosts. Similarly, we analyze the mitigation resilience of existing P2P botnets by prototyping sinkholing strategies. Finally, we discuss trends towards highly resilient P2P botnets, such as reputation schemes and self-healing P2P protocols.

1.4 Thesis Outline

The remainder of this thesis is structured as follows. Chapter 2 provides detailed background information about botnets and discusses recent developments that are important in the context of this work. Part I then addresses the complexity of establishing a dynamic analysis system that suits the task of analyzing the resilience of botnets. Specifically, Chapter 3 proposes and discusses best practices for creating and using malware datasets in scientific experiments. In Chapter 4, we propose Sandnet, a dynamic malware analysis environment implementing many of the proposed guidelines. Part II analyzes the resilience of botnets. In particular, Chapter 5 includes our large-scale analysis of malware downloaders. Chapter 6 characterizes and discusses the resilience of peer-to-peer botnets. We conclude this thesis in Chapter 7.

2 Background

In this chapter, we detail background information required to understand our analysis on botnet resilience. We first propose a model for the malware lifecycle. Then we discuss the different botnet C&C architectures. Third, we define the term *resilience* in the context of botnets and give examples of botnet resilience evaluations in the past. Last, we describe the most common malware analysis techniques, as we will often use such methods in this thesis.

2.1 The Malware Lifecycle

Malware often follows a systematic lifecycle that helps to understand its resilience. We propose a malware lifecycle model in Figure 2.1, spanning five *phases* that malware typically undergoes.

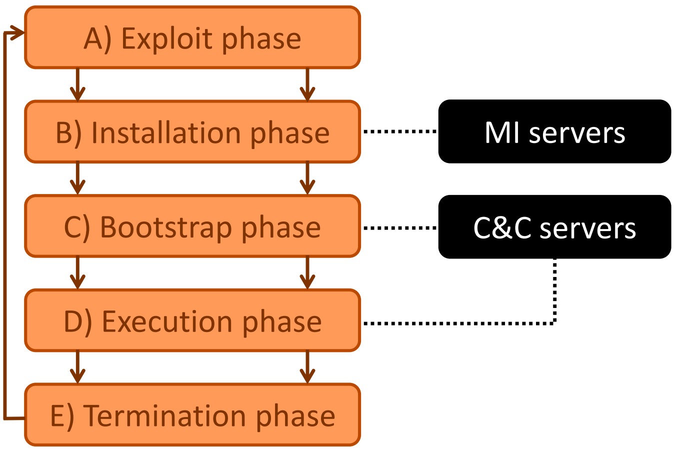


Figure 2.1: The malware lifecycle and required infrastructures in each phase

We describe the five phases in the following:

A) In the *exploit phase*, a vulnerable system is exploited by attackers, gaining basic access on the victim's PC. Techniques to exploit systems are client-side vulnerabilities such as drive-by downloads, where browser vulnerabilities are exploited by malicious code embedded in websites [75]. The exploits typically carry *shellcode* as payload, a small piece of software that is executed right after the exploit [37, 53]. Alternatively, malware can actively spread via spam attachments or file infections on, for example, memory sticks.

B) Independently from the infection technique, in the *installation phase*, malware carrying the full functionality (as opposed to shellcode) is downloaded from malware installation (MI) servers and installed on the victim system. Consequently, MI servers represent a critical part of the malware infrastructure, as they are responsible for serving malicious binaries.

C) Once these binaries are launched, the *bootstrap phase* begins, in which the malware initializes by contacting the command and control (C&C) infrastructure. By connecting to C&C infrastructures, malware specimen become *bots*, that is, malware that can be remotely controlled by the *botmaster*. During bootstrapping, a bot sends information of the infected system to the C&C server, and it may check for available updates.

D) Shortly thereafter, the C&C infrastructure serves commands to the bot, which is then in the *execution phase*. In this phase, the botmasters monetize their attacks in a variety of ways, such as spamming, extorting victims of DDoS attacks, offering proxy services, stealing personal data, or selling further malware infections.

E) If malware is upgraded to a new version, if the system is disinfected, or if the C&C infrastructure is not available anymore, the malware enters the *termination phase*.

The lifecycle model helps to understand how important the malware infrastructures are. If these infrastructures fail, attackers cannot install malware, nor can botmasters command their bots. As a consequence, attackers have invented several strategies, techniques and network architectures to improve the resilience of their botnets. Section 2.2 will explain the common botnet architectures, and Section 2.3 will discuss strategies that attackers use to harden botnet resilience.

2.2 Botnet Architectures

When bots organize in groups that can be remotely controlled by attackers, we term the originating network a *botnet*. As with normal networks, botnets are also structured in various architectures. One inherent property of all botnet architectures is that the network allows a botmaster to send commands to the bots in some way. Similarly, although not a strict requirement in every botnet, most botnet designs also allow the bots to send feedback to the botmaster. But apart from that, botnet architectures pre-define unique characteristics that are important when analyzing botnet resiliences.

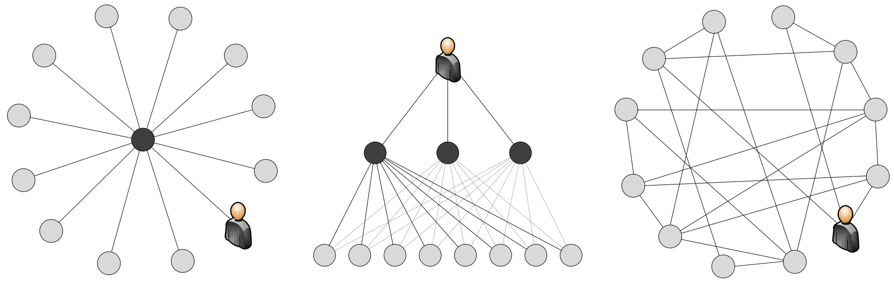

(a) Centralized (b) Semi-Distributed (c) Peer-to-Peer

Figure 2.2: Comparison of botnet architectures. Light nodes represent bots, dark nodes represent C&C servers, and the person symbols represent botmasters.

Figure 2.2 shows the three botnet architectures commonly used by botmasters: centralized, semi-distributed and peer-to-peer (P2P) botnets. We will describe these three architectures in the following subsections.

Centralized Botnets

In centralized botnets, as shown in Figure 2.2(a), all bots connect to a single C&C server. From an attacker's perspective, this client-server architecture is trivial to implement. The endpoint address of the C&C server is usually a hard-coded string within malware binaries, and bots communicate with the servers over IRC, HTTP or proprietary protocols. The main resilience bottleneck of a centralized botnet is its centralized server. If defenders take control over or disrupt the C&C server, the clients can neither request commands, nor send feedback to the servers anymore. Defenders often aim to *sinkhole* a botnet, in that they deploy infrastructures that mimic the existing C&C endpoints, for example, by redirecting a C&C domain to a server controlled by the defender (the sinkhole). When sinkholing a C&C endpoint, the defenders do not serve commands to the bots, rendering them useless for the botmasters. At the same time, the defenders can identify infected systems by looking at the sinkhole log files. We will analyze the resilience of existing centralized botnets in depth in Chapter 5.

Semi-Distributed Botnets

Over the years, attackers noticed the drawbacks of centralized botnets and developed various strategies to distribute C&C servers. The goal was to mitigate the single point of failure of centralized botnets, while retaining the client-server model of centralized botnets. Figure 2.2(b) shows that in semi-distributed botnets, the bots contact multiple different C&C servers.

The semi-distributed design significantly improved botnet resilience and thus raised the bar for botnet takedowns. Botmasters invented different schemes to achieve a higher degree of distribution. Most trivial, bot binaries contain multiple server addresses, and in case one C&C is not reachable anymore, the bot communicates with another. A special and more advanced kind of such distribution is achieved by using domain name generation algorithms (DGAs). DGAs precompute C&C server hostnames by algorithms that accept deterministic seeds (such as the current date). The botmaster knows the DGA and needs to host only the C&C server under one of the generated domains. A defender would need to register up to hundreds of frequently changing C&C domains to fully disrupt the botnet. The Torpig botnet is an example that DGAs do not guarantee bulletproof resilience, though. Stone-Gross et al. disrupted Torpig three consecutive weeks by registering DGA-generated domains [86].

Attackers use DNS fast-flux networks as a complementary technique to increase botnet resilience. In fast-flux, the set of C&C server IP addresses is rapidly changed via DNS by either the bots or the botmaster. As opposed to DGAs, if the botmaster controls the IP address fluctuation, defenders cannot compute the future end points of C&C servers. In double fast-flux botnets, also the C&C domain authoritative name servers are served round-robin, adding another layer of fluctuation.

Summarizing, semi-distributed botnets are significantly more resilient than centralized botnets, although they still have (redundant) centralized servers.

Peer-to-peer Botnets

Peer-to-peer (P2P) botnets are fully distributed botnets, in which the bots retrieve their commands from other bots via the P2P network. As Figure 2.2(c) shows, P2P bots keep track of other bots in the botnet, following an architecture without central servers. The lack of central components makes the resilience of P2P architectures attractive for botmasters. In particular, the botnet continues to operate even if a large number of bots are removed from it, and the P2P network quickly heals itself from sudden network changes. On the other hand, P2P networks are prone to other mitigation techniques, ranging from enumerating all infected bots to P2P-based sinkholing or P2P network partitioning. We characterize P2P botnets and evaluate their resilience in Chapter 6.

2.3 Botnet Resilience

In this thesis, we use *resilience* as a measure for how difficult it is for defenders to disrupt a botnet. Resilient botnets typically operate for a considerable time, that is, spanning several years. Such botnets are responsible for large fractions of the total amount of cybercrime nowadays, such as spam, DDoS attacks, data theft, extortion, to name but a few. Understanding the *reasons* for botnet resilience will help to mitigate these issues. Naming the factors for botnet resilience is difficult, though, and is not purely technical. For example, journalists such as Joseph Menn documented that although procedures for disrupting botnets were known, and the botmasters were identified, botnets kept operating because of social factors such as bureaucratic burdens or political influences [64].

Still, as we will show, the design of a botmaster significantly influences the resilience of botnets. In particular, we focus on analyzing the resilience of what lets malware become a bot: its command and control architecture. Botnets are heavily dependent on the C&C channels, given that it represents the way how to control and thus monetize bots. Botnet takedowns in the past have shown that a resilient C&C architecture is of utmost importance for botmasters. Similarly, from a defender's perspective, we need to understand the strategies and techniques botmasters use to stealthen their botnets.

Srizbi, for example, is one of the largest existing botnets which has been responsible for massively sending spam. In November 2008, defenders analyzed Srizbi's semi-distributed C&C architecture. At that time, two active Srizbi C&C servers were hosted by the former web hosting provider McColo. The detailed and public C&C resilience analysis pressurized McColo to an extent that was never before observed in the context of botnets. As a response, McColo's BGP peers stopped routing traffic for the provider responsible for such large amounts of spam, effectively disconnecting Srizbi's C&C architecture. Unfortunately, due to its semi-distributed architecture, Srizbi returned to operation only a few weeks after the takedowns. Although defenders were aware of the domain-name generation algorithm in place, they stopped registering the generated domains. In turn, botmasters could register these C&C domains, re-gaining access to most of the until then largely isolated bots.

Similarly, the peer-to-peer based spam bot Waledac was disrupted by Microsoft in *Operation b49* in February 2010. Prior analysis of researchers showed vulnerabilities in Waledac's peer-to-peer component [84]. Specifically, by adding fake P2P nodes to the network, the bots' peerlists could be manipulated such that bots would communicate with the defenders only. This way, and by shutting down the additional layer of centralized C&C servers, Microsoft was able to disrupt the botnet, causing another significant decrease in spam.

These examples show the importance of detailed botnet resilience analysis. Only complete knowledge of the C&C architecture will allow for successful counter-measures against botnets. Without resilience analysis, and just by terminating single C&C servers, most botnets would not be disrupted and possibly not even be disturbed in their operation. Then again, if successful, botnet takedowns severely mitigate the damage that malware causes, like the massive decrease in spam after the disruption of Srizbi or Storm.

2.4 Malware Analysis

We use two well-explored mechanisms to analyze botnet resilience: dynamic and static malware analysis. This section shortly describes both techniques.

Static malware analysis refers to the analysis of the malware's program logic (i.e., its source code). Tools like The Interactive Disassembler (IDA) transform bytecode into assembly or even high-level language source code. An analyst then reads, annotates and interprets the code, often manually searching for important parts in the malware binary. The term *static* reflects that the malware is not executed for analysis.

Dynamic malware analysis is a complementary technique and describes the analysis of the malware behavior. To observe the behavior, a malware binary is executed in an analysis environment. For manual dynamic analysis, analysts use debuggers such as WinDbg or OllyDbg, which allow one to set breakpoints on interesting program parts or single-step through the program. To cope with the daily abundance of new malware binaries, researchers proposed systems to automate dynamic analysis. In such automated dynamic analysis environments, the malware behavior is recorded via, for example, hypervisors, system hooks, or network taps.

In the context of botnet resilience analysis, we typically focus on analyzing the C&C server architecture. For our resilience analyses, we will combine dynamic and static malware analysis to get an accurate view of the C&C architectures. While we will use off-the-shelf tools for the static analysis, we have specific requirements for dynamic malware analysis. For example, the analysis should reveal the C&C communication, but at the same time transparently redirect harmful network traffic to local servers. Similarly, in some situations we demand special network policies, such as blocking DNS traffic to trigger the malware's backup C&C channels. In addition, the C&C resilience analysis demands for parsers of typical C&C network protocols such as DNS, IRC, and particularly HTTP.

When starting this thesis, a few automated malware analysis systems existed already, though none of them completely satisfied our needs. Anubis [17], for example, offers a public interface to upload unknown binaries that are executed using a modified version of Qemu. ThreatExpert [5] offers similar services, particularly also detailing many of the host-level events such as file or Windows registry modification. With SANDNET, we propose a complementary system of this kind, with a special focus on C&C analysis.

In the next part, we will discuss malware analysis peculiarities that motivated the development of SANDNET. We propose best practices on how to establish correct, transparent and realistic malware datasets in a safe manner. With many of the guidelines being implemented, we will use SANDNET to analyze botnet resilience throughout this work.

Part I

Designing Sound Malware Experiments

Experimentation with Malware

Before we can analyze botnet resilience, we need a solid methodology for analyzing malware. Malware experimentation is challenging for a number of reasons. When executed, malware exposes malicious behavior that must not leave lab environments unfiltered. For example, we have to ensure that spam and DDoS attacks are properly contained. At the same time, resilience analysis requires realistic malware datasets that, for example, monitor the malware's C&C communication. As in other scientific fields, real-world experiments should possibly validate measurements. Similarly, datasets must not be biased, for example, by some malware families being significantly more present than others. These problems are not only important to us, but are also vital to other malware research fields, such as malware clustering or botnet detection. In the first chapter of this part, we therefore propose guidelines that the malware research community can use for sound malware experimentation.

To monitor botnet resilience, we propose SANDNET, a dynamic malware analysis system in Chapter 4. The system implements most of the guidelines proposed in Chapter 3. SANDNET is designed such that it can reliably monitor the resilience of botnets over multiple years. Since we launched SANDNET in February 2010, it automatically analyzes daily feeds of malware samples. In the long run, the continuous malware execution allows us to monitor change in malware behavior and botnet infrastructures.

Summarizing, the following two chapters provide the basis for the botnet resilience analysis in this thesis.

3 Guidelines for Malware Experimentation

Malware researchers rely on the observation of malicious code in execution to collect datasets for a wide array of experiments, including generation of detection models, study of longitudinal behavior, and validation of prior research. For such research to reflect prudent science, the work needs to address a number of concerns relating to the correct and representative use of the datasets, presentation of methodology in a fashion sufficiently transparent to enable reproducibility, and due consideration of the need not to harm others.

In this chapter we study the methodological rigor and prudence in 36 academic publications from 2006–2011 that rely on malware execution. 40% of these papers appeared in the 6 highest-ranked academic security conferences. We find frequent shortcomings, including problematic assumptions regarding the use of execution-driven datasets (25% of the papers), absence of description of security precautions taken during experiments (71% of the articles), and oftentimes insufficient description of the experimental setup. Deficiencies occur in top-tier venues and elsewhere alike, highlighting a need for the community to improve its handling of malware datasets. In the hope of aiding authors, reviewers, and readers, we frame guidelines regarding transparency, realism, correctness, and safety for collecting and using malware datasets. We will use these guidelines ourselves throughout this thesis to perform prudent botnet resilience analyses in succeeding chapters.

3.1 Introduction

Observing the host- or network-level behavior of malware as it executes constitutes an essential technique for researchers seeking to understand malicious code. Dynamic malware analysis systems like Anubis [17], CWSandbox [94] and others [32, 46, 54, 72, 79] have proven invaluable in generating ground truth characterizations of malware behavior. The anti-malware community regularly applies these ground truths in scientific experiments, for example to evaluate malware detection technologies [12, 22, 38, 40, 50, 51, 57, 61, 82, 88, 99–101], to disseminate the results of large-scale malware experiments [15, 25, 79], to identify new groups of malware [12, 18, 74, 78], or as training datasets for machine learning approaches [44, 62, 68, 74, 77, 78, 87, 102]. However, while analysis of malware execution clearly holds importance for the community, the data collection and subsequent analysis processes face numerous potential pitfalls.

In this chapter we explore issues relating to prudent experimental evaluation for projects that use malware-execution datasets, such as botnet resilience analyses. Our interest in the topic arose while analyzing malware and researching detection approaches ourselves, during which we discovered that well-working lab experiments could perform much worse in real-world evaluations, i.e., the detection rates in real networks were significantly lower than the rates stated in the experimental evaluations included in the publications. Investigating these difficulties led us to identify and explore the pitfalls that caused them. For example, we observed that even a slight artifact in a malware dataset can inadvertently lead to unforeseen performance degradation in practice. Similarly, we noticed that monitoring botnet resilience demands for rigorously correct and sound datasets.

Thus, we highlight that performing prudent experiments involving such malware analysis is harder than it seems. Related to this, we have found that many efforts frequently fall short of fully addressing existing pitfalls. Some of the shortcomings have to do with presentation of scientific work, that is, authors remaining silent about information that they could likely add with ease. Other problems, however, go more deeply, and bring into question the basic representativeness of experimental results.

As in any science, it is desirable to ensure we undertake prudent experimental evaluations. We define experiments reported in this chapter as *prudent* if they are correct, realistic, transparent, and do not harm others. Such prudence provides a foundation to objectively judge an experiment's results, and only well-framed experiments enable comparison with related work. As we will see, however, many reported experiments could often have been improved in terms of transparency, for example, by adding and explaining simple but important aspects of the experiment setup. These additions render the papers more understandable, and enable others to reproduce results.

In addition, we find that published work frequently lacks sufficient consideration of experimental design and empirical assessment to enable translation from proposed methodologies to viable, practical solutions. In the worst case, papers can validate techniques with experimental results that suggest the authors have solved a given problem, but the solution will prove inadequate in real use. In contrast, well-designed experiments significantly raise the quality of science. Consequently, we argue that it is important to have guidelines regarding both experimental design and presentation of research results.

We aim in this chapter to frame a set of guidelines for describing and designing experiments that incorporate such prudence.To do so, we define goals that we regard as vital for prudent malware experimentation: *transparency*, *realism*, *correctness*, and *safety*. We then translate these goals to guidelines that researchers in our field can use.

We apply these guidelines to 36 recent papers that make use of malware

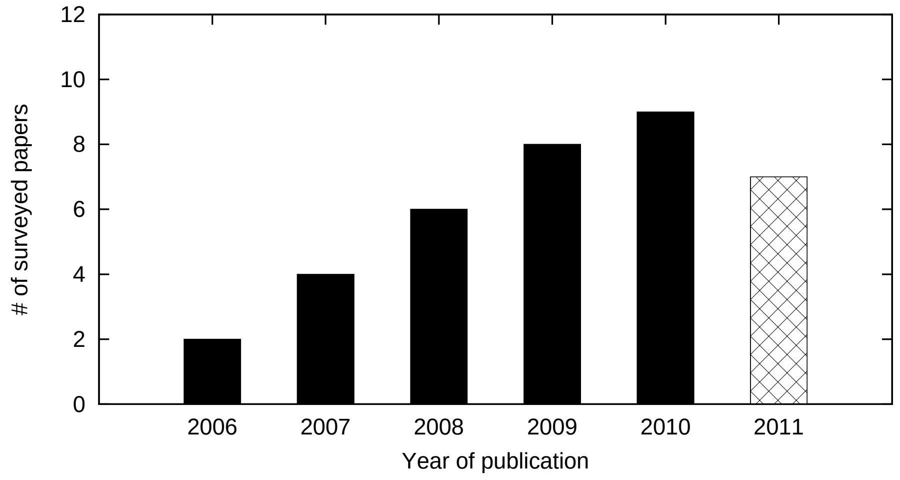


Figure 3.1: Surveyed papers using malware execution, per year.

execution data, 40% from top-tier venues such as ACM CCS, IEEE S&P, NDSS and USENIX Security, to demonstrate the importance of considering the criteria. Figure 3.1 shows the number of papers we reviewed by publishing year, indicating that usage of such datasets has steadily increased. Table 3.2 (on page 25) lists the full set of papers. We find that almost all of the surveyed papers would have significantly benefited from considering the guidelines we frame, indicating, we argue, a clear need for more emphasis on rigor in methodology and presentation in the subfield. We also back up our assessment of the significance of some of these concerns by a set of conceptually simple experiments performed using publicly available datasets.

We acknowledge that fully following the proposed guidelines can be difficult in certain cases, and indeed this chapter comes up short in some of these regards itself. For example, we do not fully transparently detail our survey datasets, as we thought that doing so might prove more of a distraction from our overall themes than a benefit. Still, the proposed guidelines can—when applicable—help with working towards scientifically rigorous experiments when using malware datasets.

3.2 Designing Prudent Experiments

We begin by discussing characteristics important for prudent experimentation with malware datasets. In formulating these criteria, we draw inspiration from extensive experience with malware analysis and malware detection, as well as from lessons we have learned when trying to assess papers in the field and—in some cases—reproducing their results.

We group the pitfalls that arise when relying on data gathered from malware execution into four categories. Needless to say, compiling *correct datasets* forms a crucial part of any experiment. We further experienced how difficult it proves to ensure *realism* in malware execution experiments. In addition, we must provide *transparency* when detailing the experiments to render them both repeatable and comprehensible. Moreover, it is our opinion that legal and ethical considerations mandate discussion of how to conduct such experiments *safely*, mitigating harm to others. For each of these four "cornerstones of prudent experimentation," we now outline more specific aspects and describe guidelines to ensure prudence. As we will show later, the following guidelines can be used to overcome common shortcomings in existing experiments.

3.2.1 Correct Datasets

A) **Check if goodware samples should be removed from datasets.** Whereas *goodware* (legitimate software) has to be present for example in experiments to measure false alarms, it is typically not desirable to have goodware samples in datasets to estimate false negative rates. However, malware execution systems open to public sample submission lack control over whether specimens submitted to the system in fact consist of malware; the behavior of such samples remains initially *unknown* rather than *malicious* per se. (We explore this concern as one of our illustrative experiments in Section 3.5.2.) We advocate that researchers use sources of malware specimens gathered via means that avoid the possible presence of goodware; explicitly remove goodware samples from their datasets; or compile sample subsets based on malware family labels.

B) **Balance datasets over malware families.** In unbalanced datasets, aggressively polymorphic malware families will often dominate datasets filtered by sample uniqueness (e.g., MD5 hashes). Authors should discuss if such imbalances biased their experiments, and, if so, balance the datasets to the degree possible.

C) **Check whether training and evaluation datasets should have distinct families.** When splitting datasets based on sample-uniqueness, two distinct malware samples of one family can potentially appear in both the training and validation dataset. Appearing in both may prove desirable for experiments that derive generic detection models for malware families by training on sample subsets. In contrast, authors designing experiments to evaluate on previously unseen malware types should *separate* the sets based on families.

D) **Perform analysis with higher privileges than the malware's.** Malware with rootkit functionality can interfere with the OS data structures that kernel-based sensors modify. Such malware can readily influence monitoring components, thus authors ought to report on the extent to which malware samples and monitoring mechanisms collide. For example, kernel-based sensors could monitor whenever malware gains equal privileges by observing if it is loading a kernel driver. Ideally, sensors are placed at a level where they cannot be modified, such as monitoring system calls with a system emulator or in a VM.

E) **Discuss and if necessary mitigate analysis artifacts and biases.** Execution environment artifacts, such as the presence of specific strings (e.g., user names or OS serial keys) or the software configuration of an analysis environment, can manifest in the specifics of the behavior recorded for a given execution. Particularly when deriving models to detect malware, papers should explain the particular facets of the execution traces that a given model leverages. Similarly, biases arise if the malware behavior in an analysis environment differs from that manifest in an infected real system, for example due to containment policies.

F) **Use caution when blending malware activity traces into benign background activity.** The behavior exhibited by malware samples executing in dynamic analysis environments differs in a number of ways from that which would manifest in victim machines in the wild. Consequently, environment-specific performance aspects may poorly match those of the background activity with which experimenters combine them. The resulting idiosyncrasies may lead to seemingly excellent evaluation results, even though the system will perform worse in real-world settings. Authors should consider these issues, and discuss them explicitly *if* they decide to blend malicious traces with benign background activity.

3.2.2 Transparency

A) **State family names of employed malware samples.** Consistent malware naming remains a thorny issue, but labeling the employed malware families in some form helps the reader identify for which malware a methodology works. As we illustrate in Section 3.5.3, employing a large number of unique malware *samples* does not imply family diversity, due to the potential presence of binary-level polymorphism. If page-size limitations do not allow for such verbose information, authors can outsource this information to websites and add references to their paper accordingly.

B) **List which malware was analyzed when.** To understand and repeat experiments the reader requires a summary, perhaps provided externally to the paper, that fully describes the malware samples in the datasets. Given the ephemeral nature of some malware, it helps to capture the dates on which a given sample executed to put the observed behavior in context, say of a botnet's lifespan that went through a number of versions or ended via a take-down effort.

C) **Explain the malware sample selection.** Researchers often study only a subset of all malware specimens at their disposal. For instance, for statistically valid experiments, evaluating only a *random selection* of malware samples may prove necessary. Focusing on more recent analysis results and ignoring year-old data may increase relevance. In either case, authors should describe how they selected the malware subsets, and if not obvious, discuss any potential bias this induces. Note that random sample selections still may have imbalances that potentially need to be further addressed (see guideline A.2).

D) **Mention the system used during execution.** Malware may execute differently (if at all) across various systems, software configurations and versions. Explicit description of the particular system(s) used (e.g., "Windows XP SP3 32bit without additional software installations") renders experiments more transparent, especially as presumptions about the "standard" OS change with time. When relevant, authors should also include version information of installed software.

E) **Describe the network connectivity of the analysis environment.** Malware families assign different roles of activity depending on a system's connectivity, which can significantly influence the recorded behavior. For example, in the Waledac botnet [84], PCs connected via NAT primarily sent spam, while systems with public IP addresses acted as fast-flux "repeaters."

F) **Analyze the reasons for false positives and false negatives.** False classification rates alone provide little *clarification* regarding a system's performance. To reveal fully the limitations and potential of a given approach in other environments, we advocate thoughtful exploration of what led to the observed errors. Sommer and Paxson explored this particular issue in the context of anomaly detection systems [80].

G) **Analyze the nature/diversity of true positives.** Similarly, true positive rates alone often do not adequately reflect the potential of a methodology [80]. For example, a malware detector flagging hundreds of infected hosts may sound promising, but not if it detects only a single malware family or leverages an environmental artifact. Papers should evaluate the diversity manifest in correct detections to understand to what degree a system has general discriminative power.

3.2.3 Realism

A) **Evaluate relevant malware families.** Using significant numbers of popular malware families bolsters the impact of experiments. Given the ongoing evolution of malware, exclusively using older or sinkholed specimens can undermine relevance.

B) **Perform real-world evaluations.** We define a real-world experiment as an evaluation scenario that incorporates the behavior of a significant number of hosts in active use by people other than the authors. Real-world experiments play a vital role in evaluating the gap between a method and its application in practice.

C) **Exercise caution generalizing from a single OS configuration.** For example, by limiting analysis to a single OS version, experiments may fail with malware families that solely run or exhibit different behavior on disregarded OS versions. For studies that strive to develop results that generalize across OS versions, papers should consider to what degree we can generalize results based on one specific OS version.

D) **Choose appropriate malware stimuli.** Malware classes such as keyloggers require triggering by specific stimuli such as keypresses or user interaction in general. In addition, malware often exposes additional behavior when allowed to execute for more than a short period [79]. Authors should therefore describe why the analysis duration they chose suffices for their experiments. Experiments focusing on the initialization behavior of malware presumably require shorter runtimes than experiments that aim to detect damage functionality such as DoS attacks.

E) **Consider allowing Internet access to malware.** Deferring legal and ethical considerations for a moment, we argue that experiments become significantly more realistic if the malware has Internet access. Malware often requires connectivity to communicate with command-and-control (C&C) servers and thus to expose its malicious behavior. In exceptional cases where experiments in simulated Internet environments are appropriate, authors need to describe the resulting limitations.

3.2.4 Safety

A) **Deploy and describe containment policies.** Well-designed containment policies facilitate realistic experiments while mitigating the potential harm malware causes to others over time. Experiments should at a minimum employ basic containment policies such as redirecting spam and infection attempts, and identifying and suppressing DoS attacks. Authors should discuss the containment policies and their implications on the fidelity of the experiments. Ideally, authors also monitor and discuss security breaches in their containment.

3.3 Methodology for Assessing the Guidelines

The previous section described guidelines for designing and presenting scientifically prudent malware-driven experiments. As an approach to verify if our guidelines are in fact useful, we analyzed in which cases they would have significantly improved experiments in existing literature. This section describes our methodology for surveying relevant publications with criteria derived from our guidelines.

3.3.1 Assessment Criteria

Initially, we establish a set of criteria for assessing the degree to which experiments presented in our community adhere to our guidelines. We aim to frame these assessments with considerations of the constraints the reviewer of a paper generally faces, because we ultimately wish to gauge how well the subfield develops its research output. Consequently, we decided not to attempt to review source code or specific datasets, and refrained from contacting individual authors to clarify details of the presented approaches. Instead, our goal is to assess the prudence of experiments given all the information available in a paper or its referenced related work, but no more. We employed these constraints since they in fact reflect the situation that a reviewer faces. A reviewer typically is not supposed to clarify missing details with the authors (and in the case of double-blind submissions, lacks the means to do so). That said, we advocate that readers facing different constraints should contact authors to clarify lacking details whenever possible.

Table 3.1 lists the guideline criteria we used to evaluate the papers. We translate each aspect addressed in Section 3.2 into at least one concrete check that we can perform when reading a given paper.[1] We defined the assessment criteria in an objective manner such that each item can be answered without ambiguity. We also assign a three-level qualitative importance rating to each check, based on our experience with malware execution analysis. Later on, this rating allows us to weigh the interpretation of the survey results according to the criteria's criticality levels.

Criterion	Gdl.	Imp.	Description
Correct Datasets			
Removed goodware	A.1)	●	Removed legitimate binaries from datasets
Avoided overlays	A.6)	●	Avoided comparison of execution output with real system output
Balanced families	A.2)	◐	Training datasets balanced in terms of malware families
Separated datasets	A.3)	◐	Appropriately separated training and evaluation datasets based on families
Mitigated artifacts/biases	A.5)	◐	Discussed and if necessary mitigated analysis artifacts or biases
Higher privileges	A.4)	◐	Performed analysis with higher privileges than the malware
Transparency			
Interpreted FPs	B.6)	●	Analyzed when and why the evaluation produced false positives
Interpreted FNs	B.6)	●	Analyzed when and why the evaluation produced false negatives
Interpreted TPs	B.7)	●	Anal. the nature/diversity of true positives
Listed malware families	B.2)	◐	Listed the malware family names
Identified environment	B.4)	◐	Named or descr. the execution env.
Mentioned OS	B.4)	◐	Named the OS used during analysis

[1]Although the guideline "Choose appropriate malware stimuli" is in the *Realism* section, we added the criterion "Mentioned trace duration" (as one possible criterion for this guideline) to the *Transparency* category.

Criterion	Gdl.	Imp.	Description
Described naming	B.1)	◐	Described the methodology of how malware family names were determined
Described sampling	B.3)	○	Detailed malware selection mechanism
Listed malware	B.1)	○	Listed which malware was when analyzed
Described NAT	B.5)	○	Described if NAT was used
Mentioned trace duration	C.4)	○	Described for how long malware traces were recorded.
Realism			
Removed moot samples	C.1)	●	Explicitly removed outdated or sinkholed samples from dataset
Real-world FP exp.	C.2)	●	Performed real-world evaluation measuring wrong alarms or classifications
Real-world TP exp.	C.2)	●	Performed real-world evaluation measuring true positives
Used many families	C.1)	●	Evaluated against a significant number of malware families
Allowed Internet	C.5)	◐	Allowed Internet access to malware
Added user interaction	C.4)	○	Explicitly employed user interaction to trigger malware behavior
Used multiple OSes	C.3)	○	Analyzed malware on multiple OSes
Safety			
Deployed containment	D.1)	●	Deployed containment policies to mitigate attacks during malware execution

Table 3.1: List of criteria assessed during our survey. The second column names the guideline from which we derived this criterion. The third column denotes the importance that we devote to this subject: ● is a must, ◐ should be done, and ○ is nice to have.

For an informal assessment of our approach, we asked the authors of two papers to apply our criteria. The researchers were asked if the criteria were applicable, and if so, if the criteria were met in their own work. During this calibration process, we broadened the check to determine coverage of false positives and false negatives, to allow us to perform a generic assessment. In addition, as we will discuss later, we realized that not all criteria can be applied to all papers.

3.3.2 Surveyed Publications

We assessed each of the guideline criteria against the 36 scientific contributions ("papers") in Table 3.2. We obtained this list of papers by systematically going through all of the proceedings of the top-6 computer- and network-security conferences from 2006–2011. [2] We added a paper to our list if any of its experiments make use of PC malware execution-driven datasets. We then also added an arbitrary selection of relevant papers from other, less-prestigious venues, such that in total about two fifth (39%) of the 36 surveyed papers were taken from the top-6 security conferences. We selected papers in August 2011, missing a few papers that appeared in 2011, but were not public as of then. As Figure 3.1 shows, we see increasing use of malware execution during recent years.

#	1st Author Paper Title	Venue	Top
1	Lanzi [57] *AccessMiner: Using System-Centric Models for Malware Protection*	ACM CCS 2010	✓
2	Morales [68] *Analyzing and Exploiting Network Behaviors of Malware*	IEEE SecComm '10	
3	Rieck [78] *Automatic Analysis of Malware Behavior using Machine Learning*	Journal of CompSec.	
4	Bailey [12] *Automated Classification and Analysis of Internet Malware*	RAID 2007	
5	Wurzing. [95] *Automatically Generating Models for Botnet Detection*	ESORICS 2009	
6	Bayer [15] *A View on Current Malware Behaviors*	USENIX LEET 2009	
7	Perdisci [74] *Behavioral Clustering of HTTP-Based Malware and Signature Generation [...]*	NSDI 2010	
8	Kirda [50] *Behavior-based spyware detection*	USENIX Sec. 2006	✓
9	Jang [45] *BitShred: Feature Hashing Malware for Scalable Triage and Semantic Analysis*	ACM CCS 2011	✓
10	Zhang [101] *Boosting the Scalability of Botnet Detection Using Adaptive Traffic Sampling*	ASIACCS 2011	
11	Gu [38] *BotMiner: Clustering Analysis of Network Traffic for [...] Botnet Detection*	USENIX Sec. 2008	✓
12	Strayer [88] *Botnet Detection Based on Network Behavior*	Adv. Info. Sec. 2008	
13	Gu [40] *BotSniffer: Detecting Botnet C&C Channels in Network Traffic*	NDSS 2008	✓

[2]We determined the top-6 conferences based on three conference-ranking websites: (1) Microsoft Academic Search - Top Conferences in Security & Privacy (`http://academic.research.microsoft.com/RankList?entitytype=3&topdomainid=2&subdomainid=2`), (2) Guofei Gu's Computer Security Conference Ranking and Statistic (`http://faculty.cs.tamu.edu/guofei/sec_conf_stat.htm`), and (3) Jianying Zhou's Top Crypto and Security Conferences Ranking (`http://icsd.i2r.a-star.edu.sg/staff/jianying/conference-ranking.html`). As all rankings agreed on the top 6, we chose those as constituting top-tier conferences: ACM CCS, IEEE S&P, NDSS, USENIX Security, and two conferences (Crypto and Eurocrypt) without publications in our focus. We defined this list of top-venues prior to assembling the list of papers in our survey.

#	1st Author / Paper Title	Venue	Top
14	Bowen [22]	RAID 2010	
	BotSwindler: Tamper Resistant Injection of Believable Decoys [...]		
15	Liu [61]	ISC 2008	
	BotTracer : Execution-based Bot-like Malware Detection		
16	Rieck [77]	ACM SAC 2010	
	Botzilla: Detecting the "Phoning Home" of Malicious Software		
17	Stinson [82]	DIMVA 2007	
	Characterizing Bots' Remote Control Behavior		
18	Lindorfer [60]	RAID 2011	
	Detecting Environment-Sensitive Malware		
19	Gu [39]	USENIX Sec. 2007	✓
	Detecting Malware Infection Through IDS-Driven Dialog Correlation		
20	Caballero [26]	ACM CCS 2009	✓
	Dispatcher: Enabling Active Botnet Infiltration [...]		
21	Kolbitsch [52]	USENIX Sec. 2009	✓
	Effective and Efficient Malware Detection at the End Host		
22	Balzarotti [13]	NDSS 2010	✓
	Efficient Detection of Split Personalities in Malware		
23	Stone-Gr. [87]	ACSAC 2009	
	FIRE: FInding Rogue nEtworks		
24	Bayer [16]	ACM SAC 2010	
	Improving the Efficiency of Dynamic Malware Analysis		
25	Kolbitsch [51]	IEEE S&P 2010	✓
	Inspector Gadget: Automated Extraction of Proprietary Gadgets [...]		
26	Jacob [44]	USENIX Sec. 2011	✓
	JACKSTRAWS: Picking Command and Control Connections from Bot Traffic		
27	Rieck [76]	DIMVA 2008	
	Learning and Classification of Malware Behavior		
28	Caballero [25]	USENIX Sec. 2011	✓
	Measuring Pay-per-Install: The Commoditization of Malware Distribution		
29	Yu [100]	Journ. of Netwks 2010	
	Online Botnet Detection Based on Incremental Discrete Fourier Transform		
30	Milani C. [67]	IEEE S&P 2009	✓
	Prospex: Protocol Specification Extraction		
31	Rossow [79]	ACM BADGERS 2011	
	Sandnet: Network Traffic Analysis of Malicious Software		
32	Bayer [18]	NDSS 2009	✓
	Scalable, Behavior-Based Malware Clustering		
33	Barford [14]	USENIX HotBots 2007	
	Toward Botnet Mesocosms		
34	Yen [99]	DIMVA 2008	
	Traffic Aggregation for Malware Detection		
35	Zhu [102]	SecureComm 2009	
	Using Failure Information Analysis to Detect Enterprise Zombies		
36	Livadas [62]	IEEE LCN 2006	
	Using Machine Learning Techniques to Identify Botnet Traffic		

Table 3.2: List of surveyed papers ordered by title. We shorten some titles with "[...]" due to space limitations.

The surveyed papers use malware datasets for diverse purposes. A significant number used dynamic analysis results as input for a *training process* of malware detection methods. For example, Botzilla [77] and Wurzinger et al. [95] use malicious network traffic to automatically generate payload signatures of malware. Similarly, Perdisci et al. [74] propose a methodology to derive signatures from malicious HTTP request patterns. Livadas et al. [62] identify IRC-based C&C channels by applying machine-learning techniques to malware execution results. Zhu et al. [102] train SVMs to model the abnormally high network failure rates of malware. Morales et al. [68] manually derive characteristics from malware observed during execution to create detection signatures. Malheur [76, 78] can cluster and classify malware based on ordered behavioral instructions as observed in CWSandbox. Kolbitsch et al. [52] present a host-based malware detection mechanism relying on system call slices as observed in Anubis.

In addition, we have surveyed papers that used malware execution solely to *evaluate methodologies*. Most of these papers leverage malware traces to measure true positive rates of malware detection mechanisms [22, 38–40, 50, 57, 61, 82, 88, 99–101]. Typically, the authors executed malware samples in a contained environment and used the recorded behavior as ground truth for malicious behavior, either via network traces (for assessing network-based IDSs) or via host behavior such as system call traces (for system-level approaches). Similarly, researchers have used malware execution traces for evaluating methodologies to understand protocol semantics [26, 67], to extract isolated code parts from malware binaries [51], to detect if malware evades contained environments [13], or to improve the efficiency of dynamic analysis [16].

A third group of papers used malware traces to *obtain a better understanding of malware behavior*. For example, JACKSTRAWS [44] leverages Anubis to identify botnet C&C channels. Similarly, FIRE [87] identifies rogue networks by analyzing malware communication end points. Caballero et al. [25] execute malware to measure the commoditization of pay-per-install networks. DISARM [60] measures how different malware behaves in virtualized environments compared to Anubis. Bayer et al. [18] and Jang et al. [45] present efficient clustering techniques for malware behavior. Bailey et al. [12] label malware based on its behavior over time. Finally, Bayer et al. [15] and Rossow et al. [79] analyze the behavioral profiles of malware samples as observed in Anubis and SANDNET. Also the botnet resilience analysis in this thesis falls into this group.

3.3.3 Survey Methodology

To ensure consistency and accuracy in our survey results, two persons conducted an initial survey of the full set of papers. Employing a fixed pair of reviewers helps to ensure that all papers received the same interpretation of the guideline criteria. When the two reviewers did not agree, a third person decided on the specific case. In general, if in doubt or when encountering vague decisions, we classified the paper as conforming with the guideline (“benefit of the doubt”). Note that our assessments of the papers contain considerably more detail than the simple statistic summaries presented here. If a paper lacked detail regarding experimental methodology, we further reviewed other papers or technical reports describing the particular malware execution environment. We mark criteria results as "unknown" if after doing so the experimental setup remained unclear.

We carefully defined subsets of *applicable papers* for all criteria. For instance, executions of malware recompiled to control network access do not require containment policies. Similarly, analyzing the diversity of false positives applies only to methodologies that have false positives, while removing goodware samples matters only when relying on unfiltered datasets with unknown (rather than guaranteed malicious) binaries. Also, removing outdated or sinkholed samples might not apply if the authors manually assembled their datasets. Balancing malware families is applicable only for papers that use datasets in classification experiments and if authors average classification performances over the (imbalanced) malware samples. Moreover, we see a need to separate datasets in terms of families only if authors suggest that a methodology performs well on previously unseen malware types. We further define real-world experiments to be applicable only for malware detection methodologies. These examples show that building subsets of applicable papers is vital to avoid skew in our survey results. Consequently, we note for all criteria the number of papers to which we deemed they applied.

We also sometimes found it necessary to interpret criteria selectively to papers. For example, whereas true-positive analysis is well-defined for assessing malware detection approaches, we needed to consider how to translate the term to other methodologies (e.g., malware clustering or protocol extraction). Doing so enabled us to survey as many applicable papers as possible, while keeping the criteria fairly generic and manageable. In the case of malware clustering techniques, we translated *recall* and *precision* to true positive and false positive rate, respectively. This highlights the difficulty of arriving at an agreed-upon set of guidelines for designing prudent experiments.

3.4 Survey Observations

We divide our survey interpretation into three parts. First, in a *per-guideline analysis*, we discuss to which extent specific guidelines were met. The subsequent *per-paper analysis* assesses whether only a small fraction of all papers accounts for the results, or if our findings hold more generally across all of the papers. Finally, a *top-venue analysis* details how papers appearing in more competitive research venues (as previously defined) compare with those appearing in other venues.

Criterion Importance	All Papers			Top-Venue Papers		
Correctness	App	Ukwn	OK	App	Ukwn	OK
Removed goodware	9	0%	44%	4	0%	50%
●	More than half potentially include experiments with goodware samples in the datasets. In these cases, authors seem to have mistakenly presumed binaries from public binary execution environments as malicious.					

Criterion Importance	All Papers			Top-Venue Papers		
Correctness	App	Ukwn	OK	App	Ukwn	OK
Avoided overlays	7	0%	29%	4	0%	0%
●	Five of the seven papers that perform real-world experiments to measure true positives merged traces from execution environments into real-world ones.					
Balanced families	13	0%	54%	2	0%	50%
◐	Only half of the papers considered balancing training datasets based on malware families rather than individual specimens, possibly biasing the detection models or testing datasets towards polymorphic families.					
Separated datasets	8	0%	0%	1	0%	0%
◐	No paper discussed issues regarding separating training and testing datasets in terms of malware families. This may invalidate experiments testing if a methodology is generic.					
Mitigated artifacts/bias	36	0%	28%	14	0%	50%
◐	Less than a third discussed or removed artifacts/biases from the datasets. If present, such artifacts/biases could significantly influence experimental validity; only real-world assessment can prove otherwise.					
Higher Privileges	36	6%	75%	14	0%	86%
◐	The quarter of papers that use data recorded at a privilege level equal to that of the malware execution risk increased evasion.					
Transparency	App	Ukwn	OK	App	Ukwn	OK
Interpreted FPs	25	n/a	64%	9	n/a	89%
●	Of the papers that present false positive rates, a third lacks details beyond the plain numbers.					
Interpreted FNs	21	n/a	48%	7	n/a	57%
●	In more than half of the cases, readers have to speculate why false negatives occur.					
Interpreted TPs	30	n/a	60%	11	n/a	55%
●	Two out of five applicable papers do not interpret true positives. This can hide vital information on the *basis* and *diversity* of classifications.					
Listed malw. families	36	n/a	81%	14	n/a	86%
◐	Most papers adequately name the malware families in their datasets. Seven papers rely on high numbers of distinct samples instead, hiding on which families experiments are based.					

Criterion Importance	All Papers			Top-Venue Papers		
Correctness	App	Ukwn	OK	App	Ukwn	OK
Identified environment	36	n/a	81%	14	n/a	79%
◐	A minority of papers fail to name or describe the execution environment used to capture malware traces used during experiments.					
Mentioned OS	36	n/a	64%	14	n/a	64%
◐	A third do not mention the OS used during their experiments.					
Described naming	32	n/a	50%	12	n/a	58%
◐	Only half described how the family labels for malware samples were obtained.					
Described sampling	16	n/a	81%	5	n/a	60%
○	A fifth of the papers using bulks of malware samples do not motivate how the subsets from all available reports of a dynamic analysis environment were chosen.					
Listed malware	36	n/a	11%	14	n/a	7%
○	Almost all papers lack details on which particular malware samples (e.g., distinct MD5 hashes) were analyzed.					
Described NAT	30	n/a	10%	11	n/a	9%
○	Only three papers mention whether the execution environment used NAT or if the infected machine was assigned a public IP addresses.					
Mentioned trace dur.	36	n/a	64%	14	n/a	57%
○	A third do not mention for how long malware executed when capturing traces.					
Realism	App	Ukwn	OK	App	Ukwn	OK
Removed moot sampl.	16	0%	0%	5	0%	0%
●	No paper discussed excluding execution of outdated malware binaries or those with sinkholed communications. As we illustrate in § 3.5.4, such traces can make up a significant fraction of recorded malware behavior.					
Real-world FP exp.	20	0%	50%	6	0%	67%
●	Only half of the malware detection papers include real-world false positive experiments, vital for prudently evaluating the overhead of wrong alarms.					
Real-world TP exp.	20	0%	35%	6	0%	67%
●	Most of the malware detection papers lack real-world true positive experiments.					
Used many families	36	1/8/745		14	1/8/745	
●	Minimum/median/maximum number of malware families used in experiments.					

Criterion Importance	All Papers			Top-Venue Papers		
Correctness	App	Ukwn	OK	App	Ukwn	OK
Allowed Internet	36	6%	75%	14	0%	79%
◐	A fifth of the papers either simulated the Internet or modified bot source code to run without it, raising concerns of the realism of experiments.					
Added user interaction	36	0%	3%	14	0%	0%
○	In only one case the authors explicitly deployed sophisticated user interactions to trigger certain malware behavior. The lack of such mention in other papers may indicate that experiments lack user-triggered behaviors such as keylogging.					
Used multiple OSes	36	22%	19%	14	21%	29%
○	Only about a fifth seemed to deploy their experiments on multiple OSes.					
Safety	App	Ukwn	OK	App	Ukwn	OK
Deployed containment	28	71%	21%	11	64%	27%
●	The majority of papers did not explicitly mention containment policies, and 77% lack a policy description. This compromises transparency, and hinders readers to judge if authors gave sufficient consideration to mitigating malware attacks.					

Table 3.3: Overview and short interpretation of survey results

3.4.1 Per-Guideline Analysis

Table 3.3 lists the results of our assessment methodology ordered by theme and importance. The second major column includes statistics on all surveyed papers, while the third major column represents data from publication at top-tier venues only. *App* specifies the number of papers for which the criterion applied. *OK* states the proportion of those applicable papers that adhered to the guideline, whereas *Ukwn* specifies the proportion for which we could not assess the guideline due to lack of experimental description.

Correctness

In this section we highlight instances of criteria that potentially call into question the basic correctness of a paper's results.

In five cases, we find papers that mix behavioral traces taken from malware execution with traces from real systems. We find it difficult to gauge the degree of realism in such practices, since malware behavior recorded in an execution environment may deviate from the behavior exhibited on systems infected in the wild. For instance, Celik et al. [28] have pointed out that time-sensitive features such as frames per hour exhibit great sensitivity to the local network's bandwidth and connectivity latency; blending malware flows into other traces thus requires great care in order to avoid unnatural heterogeneity in those features. Another difference is generally the lack of user interaction in malware execution traces, which typically exists in real system traces. Consequently, we argue that researchers should not base real-world evaluations on mixed (*overlay*) datasets. On the positive side, two papers avoided overlay datasets and instead deployed sensors to large networks for real-world evaluations [74, 77].

In two papers, the authors present new findings on malware behavior derived from datasets of public dynamic analysis environments, but did not remove goodware from such datasets. Another two malware detection papers include potentially biased false negative experiments, as the datasets used for these false negative evaluations presumably contain goodware samples. We illustrate in § 3.5.2 that a significant ratio of samples submitted to public execution environments consists of goodware. Other than these four papers, all others filtered malware samples using anti-virus labels. However, no author discussed removing outdated or sinkholed malware families from the datasets, which has significant side-effects in at least one such case.

Summarizing, at least nine (25%) distinct papers appear to suffer from *clearly significant* problems relating to our three most basic *correctness* criteria. In addition, observing the range of further potential pitfalls and the survey results, we speculate that more papers may suffer from other significant biases. For example, in another 15 cases, the authors did not explicitly discuss the presence/absence of sinkholed or inactive malware samples. In addition, three malware detection papers do not name malware families, but instead use a diverse set of malware binaries during experiments. We illustrate in § 3.5.3 that such datasets are typically biased and potentially miss significant numbers of malware families. We further observed seven papers with experiments based on machine learning that did not employ cross-validation and thus potentially failed to generalize the evaluation to other datasets. To name good examples, the authors in [52, 60, 76, 76, 78] chose a subset of malware families and balanced the number of samples per family prior to the training process. Similarly, we observed authors performing cross-validation to avoid overfitting detection models [57, 76, 78, 95].

Lastly, nearly all of the papers omitted discussion of possible biases introduced by malware execution, such as malware behavior that significantly differs if binaries execute in a virtual machine [13, 60]. Typically, further artifacts or biases, for example, due to containment policies exist when executing malware as illustrated in § 3.5.5. We highlight the importance of real-world scenarios, as they favor methodologies which evaluate against realistic and correct datasets.

Transparency

We observed two basic problems regarding transparent experiment descriptions in our community. First, descriptions of experimental setups lack sufficient detail to ensure repeatability. For example, 20% of the papers do not name or describe the execution environment. For a third of the papers it remains unclear on which OS the authors tested the proposed approach, and about a fifth do not name the malware families contained in the datasets. Consequently, in the majority of cases the reader cannot adequately understand the experimental setup, nor can fellow researchers hope to repeat the experiments. In addition, 75% do not describe containment policies.

Second, we find the majority of papers do not provide complete descriptions of experimental results. That is, papers frequently fail to interpret the numeric results they present, though doing so is vital for effectively understanding the importance of the findings. Consider the simple case of presenting detection rates. In which exact cases do false positives occur? Why do some malware families raise false negatives while others do not? Do the true positives cover sufficient behavioral diversity?

Realism

Our survey reveals that only a minority of papers includes real-world evaluations, and very few papers offer significant sample sizes (e.g., in numbers of hosts) for such experiments. The lack of real-world experiments makes it hard to judge whether a proposed methodology will also work in practice. We find that authors who *do* run real-world experiments often use locally accessible networks (e.g., a university campus, or a research lab). Doing so does not constitute a problem *per se*, but authors often base such experiments on the untenable assumption that these environments do not contain malware activity. In eight cases, authors used university networks for a false positive analysis only, although their methodology should also detect malware in such traces.

We noted a further eight papers that model malicious behavior on malware samples controlled by the authors themselves. Without justification, it seems unlikely that such malware samples behave similarly to the same samples when infecting victim machines in the wild. The malware execution environment may introduce further biases, for example, via author-controlled servers that may exhibit unrealistically deterministic communication patterns. All of these cases lack representative real-world evaluations, which could have potentially offset these criteria.

We find that the typical paper evaluates its methodology against eight (median) distinct malware families, and five (14%) evaluated using only a single family. Similarly, two thirds of the surveyed malware detection methodologies evaluated against eight or fewer families. There may be a good reason for not taking into account further families, for example, if no other malware families are applicable for a specific experiment. In general, however, we find it difficult to gauge whether such experiments provide statistically sound results that can generalize.

Safety

Most papers did not deploy or adequately describe containment. More than two thirds (71%) completely omit treatment of any containment potentially used during the experiments. The reasons for this may be that authors rely on referencing to technical reports for details on their containment solution. We found, however, that only few such reports detail the containment policies in place. Two papers state that the authors explicitly refrained from deploying containment policies.

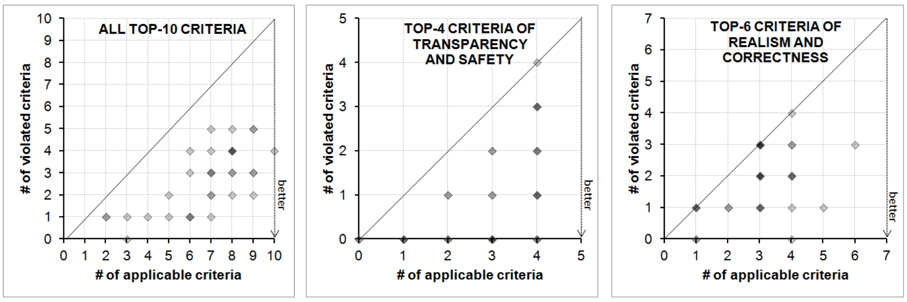


Figure 3.2: Guideline violations related to applicable criteria, separated into (1) all 10 most important criteria (left), (2) transparency/safety (middle), and (3) correctness/realism (right). Each dot represents one paper, darker dots cover more papers.

3.4.2 Per-Paper Analysis

The preceding discussion has shown the high potential of our guidelines for improving specific prudence criteria. As a next step, we analyze how many papers can in total benefit from significant improvements.

To do so, Figure 3.2 details how many of the most important criteria (● in Table 3.1)[3] a paper violated. The fewer criteria a paper met, the more its experiments could have been improved by using our guidelines. The figure shows that only a single paper fulfilled all of the applicable guidelines. More than half (58%) of the papers violate three or more criteria. In general, the plot shows a correlation between the number of violated criteria and the number of applicable criteria. This means that our guidelines become increasingly important when designing more complex experiments.

We then separate the results into presentation and safety issues (middle graph) and incorrect or unrealistic experiments (right graph). We find that lacking transparency and safety constitutes a problem in half of the cases. Far more papers (92%) have deficiencies in establishing correct datasets and realistic experiments. Note that this does not imply that the experiments suffer from heavy flaws. It does flag, however, that many papers remain silent about important experimental descriptions. In addition, this analysis shows that experiments in applicable papers could be significantly improved in terms of correct datasets and realistic experiments.

In some cases, malware datasets were reused in related papers (such as [45]), often inheriting problems from the original experiments. In such cases, issues are mostly with the original paper. However, we knowingly did not remove such papers, as we wanted to survey the *use* instead of the *creation* of malware datasets.

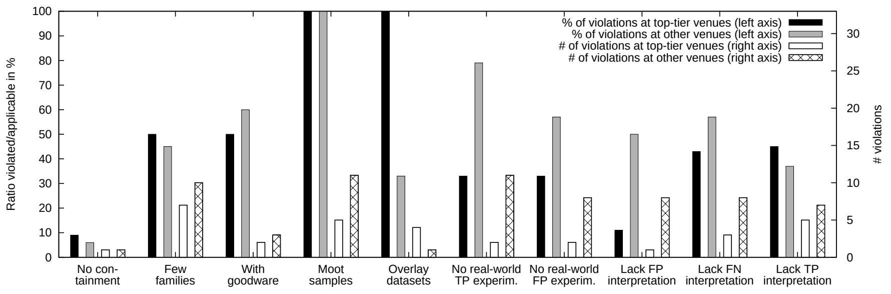


Figure 3.3: Violations at top-tier venues compared with other venues.

3.4.3 Top-Venue Analysis

We now ask ourselves if experiments presented at top-tier conferences appear to be more prudent than others. To measure this, Figure 3.3 compares results for the ten most important guidelines (● in Table 3.1). We do not observe any obvious prudence tendency towards top-tier conferences or other venues. The first strong difference regards the prevalence of real-world experiments: while more papers presented at top-tier venues include real-world scenarios, authors base these on potentially skewed overlay datasets (e.g., mixing malware traces in real traces). Second, we observed more papers interpreting false positives at top-tier conferences than at other venues. However, while the number and ratios of violations slightly differ across the criteria, the violations generally remain comparable. We therefore conclude that research published in top-tier conferences would equally benefit from our guidelines as papers presented at other venues. Thus, these shortcomings appear endemic to our field, rather than emerging as a property of less stringent peer review or the quality of submitted works.

3.5 Experiments

We now conduct four experiments that test four hypotheses we mentioned in previous sections. In particular, we will analyze the presence of (1) goodware, (2) malware family imbalances, (3) inactive and sinkholed samples, and (4) artifacts in malware datasets taken from contained environments that accept public submissions. Similar datasets were used in many surveyed experiments, raising the significance of understanding pitfalls with using such datasets. As we will show, our illustrative experiments underline the importance of proper experiment design and careful use of malware datasets. At the same time, these experiments show how we can partially mitigate some of the associated concerns.

[3]We note that we devised the importance ranking prior to conducting the analyses in this section.

3.5.1 Experimental Setup

We conducted all malware execution experiments in SANDNET [79] (Chapter 4), using a Windows XP SP3 32bit virtual machine connected to the Internet via NAT. We deploy containment policies that redirect harmful traffic (e.g., spam, infections) to local honeypots. We further limit the number of concurrent connections and the network bandwidth to mitigate DoS activities. An in-path honeywall NIDS watched for security breaches during our experiments. Other protocols (e.g., IRC, DNS or HTTP) were allowed 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. As Windows XP malware was most prevalent among the surveyed papers, we did not deploy other OS versions during dynamic analysis.

We base experiments 3.5.2 and 3.5.3 on 44,958 MD5 distinct malware samples and a diverse set of more than 100 malware families. We received these samples as a snapshot of samples submitted to a large public dynamic analysis environment during Jan.1–30, 2011. The samples originated from a diverse set of contributors, including security companies, honeypot infrastructures, and spam traps. To analyze the dynamic malware behavior in experiments 3.5.5 and 3.5.4, we randomly chose 10,670 of these 44,958 samples. We executed this subset of samples and recorded the malware's network traces at the Internet gateway. An execution typically lasts for at least one hour, but for reasons of scale we stopped execution if malware did not show network activity in the first 15 minutes. The reader can find the data regarding execution date, trace duration, MD5 hashes, and family names of the malware samples used in the experiments at our website.[4] As we use the following experiments to measure the presence of imbalances, goodware, sinkholing and artifacts, we explicitly did not clean up our dataset in this regard.

3.5.2 Legitimate Samples Under Analysis

Experiments that erroneously consider legitimate software samples as malware suffer from bias. For example, when evaluating detection accuracies, legitimate software may cause false positives. Similarly, surveys of malicious behavior will exhibit bias if the underlying dataset contains legitimate software. Thus, in this experiment, we test our hypothesis that *goodware* is significantly present in datasets from dynamic analysis systems that offer public sample submission routines.

To give lower bounds for the ratio of goodware, we queried the MD5 hash sum of all 44,958 binaries in two whitelists during the first week in November 2011. First, we query Shadowserver.org's *bin-test* [91] for known software. Second, we consulted *Bit9 Fileadvisor* [20], a file reputation mechanism also used by anti-spam vendors. *bin-test* revealed 176 (0.4%) goodware samples. The *Bit9 Fileadvisor* recognized 2,025 (4.5%) samples. In combination, both lists revealed 2,027 unique binaries as potentially being benign. As *Bit9* also includes malicious software in their database, we inspected a small sample of the 2,027 known

[4]See `http://christian-rossow.de/publications/datasets/ieee12.htm`

binaries to estimate the ratio of goodware in the hits. In particular, we manually analyzed a subset of 100 randomly selected matches and found 78 to be legitimate software. Similarly, we cross-checked the 2,027 binaries via VirusTotal and found that 67.5% did not trigger any anti-virus detection. Estimating more conservatively, we use the minimum ratio of goodware samples (67.5%) to extrapolate the number of goodware samples within the 2,027 "whitelisted" samples. This translates to a *lower bound* of 1,366 (3.0%) goodware samples in our total dataset. We can also approximate an *upper bound* estimate regarding the prevalence of nonmalicious samples by observing that 33% of the samples that were scanned by VirusTotal were not detected by any of the 44 vendors listed at VirusTotal. We therefore conclude that the ratio of legitimate binaries (3.0%–33%) may bias experiments.

3.5.3 Distribution of Malware Families

In this experiment we test our hypothesis stating that polymorphic malware manifests in an unduly large portion in randomly collected sets of malware samples. We used the VirusTotal labels obtained in Experiment 3.5.2 and counted the occurrences of malware families for each anti-virus vendor. To obtain the malware family names, we parsed the naming schemes of three anti-virus vendors (Avira, Kaspersky and Symantec) commonly used to assign malware labels.

The CDF in Figure 3.4 shows the relationship of malware families to prevalences of families in our dataset. Ideally, a multi-family malware corpus stems from a uniform distribution, that is, each malware family contributes the same number of samples. In our dataset, randomly collected from a dynamic analysis environment that publicly offers a sample submission routine, we find this goal clearly violated: some malware families far dominate others. For example, when relying on Kaspersky, almost 80% of the malware samples belong to merely 10% of the families. In the worst case, this would mean that experiments performing well with 4/5's of the samples may not work with 90% of the remaining malware families. In summary, unless researchers take corresponding precautions, polymorphic malware families can disproportionately dominate randomly drawn corpora of malware samples.

3.5.4 Inactive or Sinkholed Samples

The identification of correctly functioning malware samples poses one of the major challenges of automated dynamic analysis. Large fractions of analyzed samples do not exhibit any behavior [79]. Further complicating things, even if network communication manifests, it remains unclear whether it constitutes successful operation and representative behavior. During recent years, Shadowserver.org, Spamhaus, and other individuals/organizations have exercised take-overs of botnet infrastructure or botnet-employed domains. Such achievements can perturb empirical measurement as a side effect: takedowns introduce "unnatural" activity in collected datasets [48].

To assess the magnitude of these issues, we analyzed which of the 10,670 executed samples showed network activity, but apparently failed to bootstrap

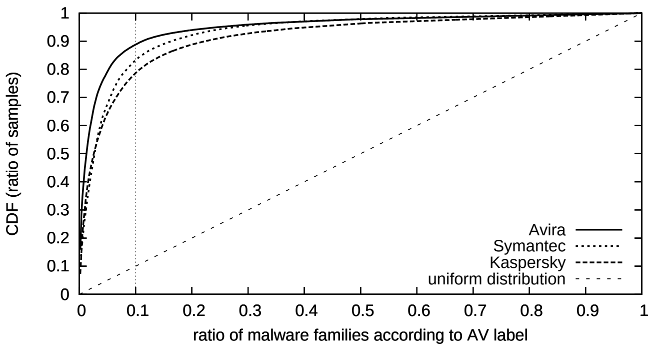


Figure 3.4: Family imbalances in randomly chosen malware samples

malicious activities. We used Avira to identify the malware families. Only 4,235 (39.7%) of the 10,670 samples showed any network activity. Of these samples, we found that of the 22 families with at least 5 distinct samples showing any HTTP activity, 14 (63%) included samples that had only failing HTTP communication (HTTP response codes 4XX/5XX). Similarly, of the most prevalent 33 families that used DNS, eight (24%) contained samples that did not have any other communication than the (typically failed) DNS lookups. We observed such inactive samples to be more prevalent in some families (e.g., Hupigon 85%, Buzus 75%), while other families (e.g., Allaple 0%) were less affected.

Next, we tried to quantify the effects of sinkholed malware infrastructure. We contacted various security organizations to obtain information about sinkholed C&C servers. These contacts enabled us to obtain sinkholing patterns of four different organizations operating sinkholing infrastructure. We then searched for these patterns for sinkholed samples among the 4,235 samples showing network activity. Most significantly, we found that during 59 of the 394 Sality executions (15%) and 27 of the 548 Virut executions (5%), at least one sinkholed domain was contacted. Although we are aware of additional malware families in our dataset to have sinkholed domains (e.g., Renos/Artro, Gozi, TDSS, SpyEye, ZeuS, Carperb, Vobfus/Changeup, Ramnit, Cycbot), we could not spot sinkholing of these in our sample dataset. Combining this data, this translates to the observation that *at least* eleven of the 126 active families (8.7%) in our dataset are potentially affected by sinkholing.

In summary, execution of inactive or sinkholed samples will not yield representative activity, highlighting the need for authors to consider and quantify their impact.

3.5.5 Artifacts

Due to the specific setups of malware execution environments, the artifacts introduced into recorded malware traces can be manifold. For example, network traffic contains specific values such as the environment's IP address or the Windows user name. We found such easy-to-spot artifacts widespread across many malware families. Specifically, we analyzed which of the recorded network traces contain the contained environment's IP address, Windows user name, or OS version. For instance, more than 10% of all Virut samples that we executed transmitted the bot's public IP address in plaintext. Similarly, one in five Katusha samples sent the Windows user name to the C&C server. The use of "`Windows NT 5.1.2600`" as HTTP User-Agent, as for example by Swizzor (57%) or Sality (52%), likewise occurs frequently. These illustrative examples of payload artifacts are incomplete, yet already more than a third (34.7%) of the active malware families in our dataset communicated either SANDNET's external IP address, our VM's MAC address, the VM's Windows username, or the exact Windows version string in plaintext in at least one case.

More dangerous types of biases may hide in such datasets, unbeknownst to researchers. For instance, methodologies relying on time-based features should consider artifacts introduced by specific network configurations, such as limited bandwidth during malware execution. Similarly, containment policies may bias the analysis results. For example, we have observed spambots that cease running if a containment policy redirects their spam delivery to a local server that simply accepts all incoming mail.

In general, it is hard to measure the exact ratio of malware families generating any artifact. Some artifacts, such as limited bandwidth or particular system configurations such as installed software, are inherent to all malware families. Consequently, authors need to carefully consider artifacts for each experiment. The best advice to preclude artifacts is to either carefully and manually assemble a dataset, or to perform representative real-world experiments.

3.6 Related Work

Prudent experiments Kurkowski et al.'s survey [56] of the technical quality of publications in the Mobile Ad Hoc Networking community inspired our methodology. As their survey's verification strategies do not immediately apply to our community's work, we needed to establish our own review criteria. Krishnamurthy and Willinger [55] have identified common methodological pitfalls in a similar fashion to ours, but regarding Internet measurements. They established a set of standard questions authors ought to consider, and illustrate their applicability in a number of measurement scenarios. Closer to our research, Aviv and Haeberlen have discussed a set of challenges in evaluating botnet detectors in trace-driven settings [10], and proposed distributed platforms such as PlanetLab as a potential enabler for more collaborative experimentation and evaluation in this space. Moreover, Li et al. [59] explored difficulties in evaluating malware clustering approaches. Supporting our observations, they observed that using balanced and well-designed datasets have significant effects on evaluation results. They then show the importance of creating ground truths in malware datasets, broaching concerns related to some guidelines in this chapter.

Perhaps most closely related to our effort is Sommer and Paxson's approach to explaining the gap between success in academia and actual deployments of anomaly-based intrusion detection systems [80]. The authors find five reasons:

(1) a very high cost of errors; (2) lack of training data; (3) a semantic gap between results and their operational interpretation; (4) enormous variability in input data; and (5) fundamental difficulties for conducting prudent evaluations. In fact, anomaly detection research has suffered from these problems for decades, whereas experiments with malware datasets are being increasingly applied. Consequently, our work complements theirs in that we shift the focus from anomaly detection to malware experiments in general.

Dynamic analysis evasion Malware datasets typically stem from dynamic analysis in specially prepared environments [17, 32, 46, 72, 79, 94]. To ensure diverse datasets, malware must not evade dynamic analysis. Others have studied the extent to which malware can detect and evade dynamic analysis [29, 60, 73]. Chen et al. present a taxonomy of dynamic analysis fingerprinting methods and perform an analysis to which extend these are used [29]. Paleari et al. present methods to automatically generate tests that effectively detect a variety of CPU emulators [73]. Most recently, Lindorfer et al. [60] analyzed how and to which extent malware samples evade Anubis.

Survey on malware detection systems Stinson and Mitchell [83] presented a first approach to evaluate existing botnet detection methodologies. They focus on possible evasion methods by evaluating six specific botnet detection methodologies. Their survey is orthogonal to ours, as we explore how authors design experiments with malware datasets. Further, we provide guidelines how to define prudent experiments that evaluate methodologies in absence of any evasion techniques. In addition, we assist researchers in designing experiments in general rather than evaluating specific methodologies.

3.7 Conclusion and Discussion

In this chapter we have devised guidelines to aid in designing prudent malware-based experiments. We assessed these guidelines by surveying 36 papers on malware analysis. We identified shortcomings in most papers from both top-tier and less prominent venues. Consequently, we argue that our guidelines could have significantly improved the prudence of most of the experiments we surveyed.

The observed shortcomings in experimental evaluation likely arise from several causes. Researchers may not have developed a methodical approach for presenting their experiments, or may not see the importance of detailing various aspects of the setup. Deadline pressures may lead to a focus on presenting novel technical content as opposed to the broader evaluation context. Similarly, detailed analyses of experimental results are often not given sufficient emphasis. In addition, page-length limits might hamper the introduction of important aspects in final copies. Finally, researchers may simply overlook some of the presented hidden pitfalls of using malware datasets.

Many of these issues can be addressed by devoting more effort to presentation, as our transparency guidelines suggest. Improving the correctness and realism of experiments is harder than it seems, though. For instance, while real-world scenarios are vital for realistic experiments, conducting such experiments can prove time-consuming and may raise significant privacy concerns for system or network administrators. Furthermore, it is not always obvious that certain practices can lead to incorrect datasets or lead to unrealistic scenarios. For example, it requires great caution to carefully think of artifacts introduced by malware execution environments, and it is hard to understand that, for example, experiments on overlay datasets may be biased. The significance of imprudent experiments becomes even more important in those instances where current practices inspire others to perform similar experiments—a phenomenon we observed in our survey.

While many of our guidelines are not new, we witnessed possible improvements to experiments for every one of the criteria. We believe this approach holds promise both for authors, by providing a methodical means to contemplate the prudence and transparent description of their malware experiments, and for readers/reviewers, by providing more information by which to understand and assess such experiments. Given their positive impact, we will do a best-effort to follow the guidelines in this thesis, striving for realistic analyses on botnet resilience.

4 Dynamic Malware Analysis with Sandnet

Dynamic analysis of malware is widely used to obtain a better understanding of unknown software. Existing dynamic-analysis systems mainly analyze host-level activities of malware and limit the analysis period to a few minutes. However, to analyze the resilience of C&C infrastructure, we need detailed analysis of a malware's network behavior. Moreover, as the previous chapter has shown, it requires great care to create correct malware datasets in a safe manner. In this chapter, we propose a dynamic analysis environment called Sandnet. With Sandnet, we provide a comprehensive overview of typical malware network behavior by discussing the results we obtained during the analysis of more than 100,000 malware samples. This chapter will provide an in-depth analysis of the two protocols that are most popular among malware authors, DNS and HTTP, providing first insights into botnet C&C channels.

4.1 Introduction

Dynamic analysis has proven to be a well-established and effective tool to understand the workings of malicious software [17, 78, 94]. Understanding the behavior of malware provides insights into damage functionalities of malware, upcoming techniques, and underground economy trends. In addition, it gives the opportunity to develop novel countermeasures specifically built on top of that understanding. Current analysis systems are mainly specialized in monitoring system-level activities, such as manipulation of Windows registry keys and accesses to the file system. Little effort has been devoted to understanding the network behavior exposed by malware. In fact, similarly to system-level activities, network-level activities also show very distinct behaviors that can back up the insights provided by system-level analysis. Moreover, detailed analyses of C&C channels help to understand their resilience. Lastly, understanding the network behavior of malware helps to develop novel approaches to collect, classify and eventually mitigate malicious software. Driven by these observations, we focus our research on dissecting, analyzing, and understanding the behavior of malicious software as observed at the network level.

As we will show later, the observed malware behavior highly depends on the duration of the dynamic analysis. Current systems try to analyze as many malware samples as possible in a given period of time. This results in very short analysis periods, usually lasting only a few minutes, which makes it difficult to observe malicious network behavior that goes beyond the bootstrapping process. From a network behavior point of view, however, the post-bootstrap behavior is often more interesting than what happens in the first few minutes. A thorough analysis is key to understanding the highly dynamic workings of malware, which is frequently observed to be modular and often undergoes behavior updates in a pay-for-service model.

This chapter presents an in-depth analysis of malware network behavior that we gathered with a new system called Sandnet from February 2010 to January 2011. Sandnet [3] is an analysis environment for malware that complements existing systems by a highly detailed analysis of malicious network traffic. With Sandnet, we try to address limitations we see in publicly available dynamic analysis systems. While existing systems usually execute a malware sample for a few minutes, we analyze each sample for at least one hour. Moreover, Sandnet implements policies that can trigger specific C&C behavior important for analyzing botnet resilience. For example, Sandnet can block DNS requests of bots to trigger backup C&C channels. Using the data collected through Sandnet, we provide a comprehensive overview of network activities of current malware.

This chapter is structured as follows. In Section 4.2, we will give an overview of Sandnet. Section 4.3 describes the dataset our analysis is based on. We will then provide a general malware network traffic overview in Section 4.4. In Section 4.5, we will analyze the usage of the DNS protocol by malware. Section 4.6 describes the usage of the HTTP protocol by malware. We will discuss related work in Section 4.7 and show future work in Section 4.9.

4.2 System Overview

In Sandnet, malware is analyzed in execution environments known as *sandpuppets* consisting of a (virtualized) hardware and a software stack. Currently, we use VMs with Windows XP SP3 based on VirtualBox as sandpuppets. The machines are infected immediately after booting using batch scripts and gracefully shut down after a configurable time interval, which is typically one hour. Each sandpuppet is configured to have a local IPv4 address and a NATed Internet connection. A local DNS resolver is preconfigured.

Figure 4.1 shows the system overview of Sandnet. The *sandherder* is a Linux system hosting the sandpuppet virtual machines. Besides virtualization, the sandherder also records, controls and proxies network traffic to the Internet. We limit the potential damage of running malware samples by transparently redirecting certain traffic (e.g., spam, infections) to local sinkholes or honeypots. For example, SMTP traffic is captured at a local mail server that forges the IP address and even SMTP welcome banner of the original MX server. In addition, we limit the number of concurrent connections, the network bandwidth and the packet rates per sandpuppet to mitigate DoS activities. Internet connectivity parameters such as bandwidth and packet rate are shared fairly among the sandpuppets to avoid inter-execution artifacts. The current SANDNET setup comprises five bot sandherders with four sandpuppets each, resulting in twenty sandpuppets dedicated to malware analysis. Herders and sandpuppets can easily be added due to a flexible and distributed design.

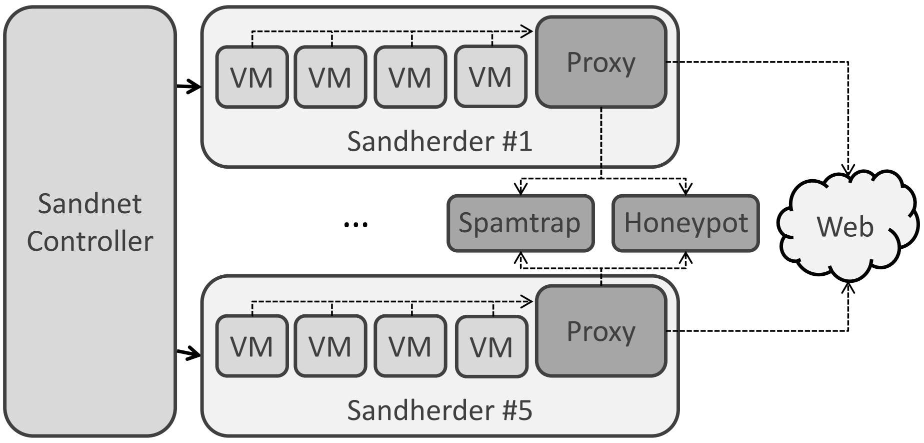


Figure 4.1: System overview of SANDNET

After executing a malware binary, we dissect the recorded network traffic for further analysis. A *flow extractor* converts raw .pcap-files into UDP/TCP *flows*. A flow is a network stream identified by the usual 5-tuple (layer 4 protocol, source IP addr., destination IP addr., source port, destination port). For TCP, a flow corresponds to a reassembled TCP connection. For UDP, a flow is considered to be a stream of packets terminated by an inactivity period of 5 minutes. Our experience shows that this timeout length is a reasonable mechanism to compensate the lack of UDP flow termination frames. Additionally, we use payload-based protocol detection in order to determine the application-level protocol of a flow. We define a flow to be *empty*, if no UDP/TCP payload is transmitted in it.

Automated execution of malware raises some ethical concerns, though. Given unrestricted network connectivity, malware could potentially harm others on the Internet. Possible attack scenarios are, but not limited to, Denial-of-Service attacks, spam or infection of other hosts. We tried to find the right balance between ethical concerns when designing SANDNET and restricting the Internet connectivity. Technically, we integrated certain honeywall techniques. The harm of DoS attacks is limited by network level rate-limiting, spam is transparently redirected to local mail servers and protocols known to be used for infection are redirected to local honeypots. SANDNET is closely monitored during execution. Admittedly, it is technically impossible to completely prevent all possible attacks. However, we are convinced that within the bounds of what is possible we implemented a huge part of mitigation techniques and that the value of SANDNET strongly outweighs the reasonably limited attack potential.

4.3 Dataset

In order to study malicious network traffic, we analyzed malware samples that were provided to us by partner research institutions. For each sample we acquire A/V scan results from VirusTotal [2]. 85% of the samples that we executed had at least one scan result indicating malware (see Figure 4.2). In order to avoid accidental benign samples we collated our set of samples with a list of known software applications using Shadowserver's bintest [91]. We randomly chose the samples from a broad distribution of all malware families. We tried to mitigate side-effects of polymorphism by extracting the family name of a given malware sample's A/V labels and limit the number of analyses per malware family. For our analysis we defined the following set of samples. We analyzed a total of 104,345 distinct samples (in terms of MD5 hashes) from February 2010 to January 2011. A complete list of all samples can be found at the URL `http://christian-rossow.de/files/dataset-sandnet-chapter.txt`, including MD5 checksums, malware families based on anti-virus scans and dates of malware analysis. Samples were executed with regard to their age. On average, the samples were executed about 7.8 days after submission. We prefer newer samples for execution in SANDNET, but re-scheduling of older samples (see Chapter 5) increases the average sample age during execution.

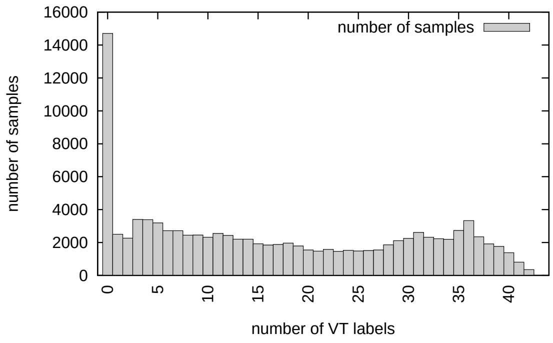


Figure 4.2: Histogram on the number of VirusTotal labels per sample in SANDNET

4.4 Network Statistics Overview

Of the 104,345 samples, the subset S_{Net} of 45,651 (43.8%) samples exhibited some kind of network activity.

The network traffic caused by these samples sums up to more than 70 million flows and a volume of 207 GB. It remains an open issue to understand why a majority of the samples did not show any network activity. We suspect that most of the inactive samples a) are invalid PE files, b) operate on a local system only (e.g., disk encryption), c) are active only if there is user activity, or d) detected that they are being analyzed and stopped working.

Table 4.1 is a complete list of the ISO/OSI layer-7 protocol distribution observed in SANDNET. Protocol inspection reveals that a typical sample in S_{Net} uses DNS (92.3%) and HTTP (58.6%). IRC is still quite popular: 8% of the samples exposed IRC. Interestingly, SMTP only occurred in 3.8% of the samples in S_{Net}, indicating that traditional SMTP spam is a less popular monetization technique.

Protocol	**Samples**	**Flows**	**Volume**	**IP addr.**	**Dst Domains**
DNS	42,143	11,845,193	3.7 GB	241,126	14,732
HTTP	26,738	13,492,189	110 GB	36,921	55,032
Unknown	18,349	32,265,514	24 GB	9,145,625	86,523
Flash	5881	299986	32 GB	2955	2205
SSL	5104	79344	1.9 GB	2278	1622
SMB	4275	8,602,414	6.1 GB	7,253,975	10
IRC	3657	169,833	0.1 GB	564	554
SMTP	1715	3,155,014	20 GB	282,401	118,959
MPEG	1162	2200	0.2 GB	58	44
SSDP	885	1861	<0.1 GB	2	0
Quicktime	389	1222	8.3 GB	62	41
FTP	243	7523	<0.1 GB	159	121
NetBIOS	184	134,600	<0.1 GB	108,909	0
TDS	163	1086	<0.1 GB	44	36
NTP	102	2950	<0.1 GB	13	5
STUN	68	276	<0.1 GB	19	8
TFTP	48	12,492	0.6 GB	19	0
PPLIVE	37	1481	0.1 GB	1321	0
Gnutella	32	20,545	0.2 GB	15,640	0
DDL	28	277	<0.1 GB	52	35
Bittorrent	26	1180	0.1 GB	588	32
Mysql	21	33	<0.1 GB	12	7

Table 4.1: SANDNET: Distribution of ISO/OSI layer 7 protocols

As DNS and HTTP are by far the most widely used protocols in SANDNET traffic, we will inspect these in more detail in Table 4.2. Table 4.2 also compares our protocol statistics with data based on Anubis provided by Bayer et al. [15] in 2009. When comparing the results, the samples we analyzed showed increased usage of all protocols. However, the ranking and the proportion of the protocols remain similar. We suspect this increase is a) due to a relatively long execution of malware samples and b) caused by a growing usage of different application-level protocols by malware.

30.1% of the flows were empty (no payload was transmitted). We verified that the vast majority of these flows originate from TCP port scans, most being redirected to our local honeypot. Already 90% of the empty flows targeted Net-

Protocol	**Reference**	**Sandnet**
DNS	44.5 %	92.3 %
HTTP	37.6 %	58.6 %
IRC	2.3 %	8.0 %
SMTP	1.6 %	3.8 %

Table 4.2: Sandnet: Layer-7 protocol distribution compared with Bayer et al. [15]

BIOS/SMB services. The remaining empty flows are normally distributed over lots of different ports.

Of the remaining flows with payload (69.9%), for 22.8% no well-known protocol could be determined. Over 60% of these flows account for NetBIOS or SMB-related communication (mostly scanning) according to the destination port. Again, the remaining flows with failed protocol detection are normally distributed across many destination ports.

Payload-based protocol detection is a big advantage if protocols are used over other than their well-known ports. We found that 12.8% of S_{Net} use protocols over other than the well-known ports. We speculate that in these cases malware tries to communicate via ports opened in the firewall, independent from the actual communication protocol. For instance, we regularly found IRC bots connecting to IRC servers listening on TCP port 80. Thus, nonstandard port usage might serve as a malware classification or detection feature. The top-3 affected protocols are listed in Table 4.3.

Protocol	S_{Net} **Samples**	**Distinct Ports**
HTTP	8.17 %	303
IRC	7.13 %	174
Flash	0.91 %	9

Table 4.3: Top 3 protocols over nonstandard ports

As additional analysis, we found out that a longer analysis period is indeed helpful for a better understanding of malware behavior. To judge on this, we performed two measurements each after an analysis period of 5 minutes and after 1 hour. First, we found out that only 23.6% of the communication endpoints that we have seen samples connecting to were contacted in the first 5 minutes of analysis. We then calculated that only a minor fraction (6.1%) of all flows started within the first 5 minutes. Lastly, we found that 4.8% of the samples started using a new protocol after 5 minutes that they have not used in the first minutes.

4.5 DNS

DNS is by far the most prevalent layer-7 protocol in Sandnet network traffic and gives an interesting insight into malware activity. The subset of samples using DNS is denoted by S_{DNS}.

4.5.1 DNS Resolution

Although all sandpuppets have their Windows stub resolver point to a working DNS resolver, we observed malware that used a different resolver or even carried its own iterative resolver. We developed the following heuristic in order to detect executions that carry an iterative resolver. An execution is considered as carrying an iterative resolver if there is an incoming DNS response from a server other than the preconfigured DNS resolver with a referral concerning a TLD (a resource record of type NS in the authority section) and the Recursion Available flag set to 0. We cross checked the resulting executions whether at least one of the DNS root servers had been contacted via DNS.

We can only speculate on the reasons why the preconfigured local DNS resolver is avoided. Using one's own resolver has advantages, though. Resolution of certain domains might be blocked at the preconfigured resolvers in some environments (e.g., corporate ones). Additionally, using custom resolvers avoids leaving traces in logs or caches of the preconfigured resolver. Moreover, if the Windows stub resolver is configured to use custom resolvers, local queries can be modified at will. This could be used for phishing attacks (redirect to a proxy) or to prevent A/V software from updating. Furthermore, preconfigured resolvers might be rate-limited.

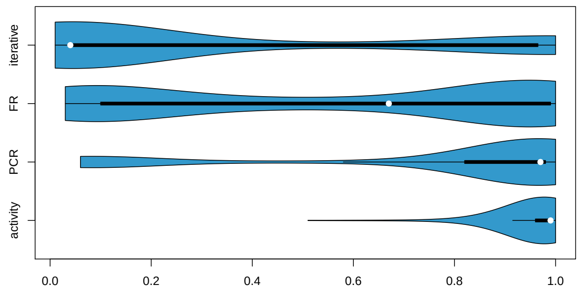


Figure 4.3: Violin plot of DNS activity end distribution

We found that 99% of the samples in S_{DNS} use the preconfigured resolver. Given this high ratio, a DNS resolver indeed turns out to be an interesting source for network-based malware detection. In fact, Bilge et al. [19] and Antonakakis et al. [8] proposed DNS-based malware detection approaches. However, 3% of S_{DNS} perform recursive DNS resolution with other resolvers than the preconfigured one (termed foreign resolvers in the following). Only 2% of S_{DNS} expose iterative DNS resolution. Note that the sets are not disjunct, as an execution may exhibit multiple resolution methods or resolvers. We speculate that this is due to the fact that malware occasionally downloads and executes multiple binaries, each of which might have different resolution methods. The foreign resolvers used include Google's Public DNS (used by 0.38%) as well as OpenDNS (0.25%). However, there is a large number of foreign resolvers that are used less frequently. One resolver that was located in China got our attention because queries for well-known popular domains such as facebook.com and youtube.com resolved into arbitrary IP addresses with no recognizable relation to the domain. We consider this to be an artifact of the so-called Great Firewall of China [30]. In total, 932 of 5092 (18.3%) distinct DNS servers were used recursively at least once and thus can be regarded as publicly available recursive DNS resolvers.

Furthermore, we looked into the activity distribution of the different resolution methods (see Figure 4.3). The preconfigured resolver (PCR) was typically used throughout the whole analysis period. The end of the usage of foreign resolvers (FR) is widespread over time, leaning toward the end of the analysis. Interestingly, iterative resolution appears to end much sooner compared to the other resolution methods.

4.5.2 DNS TTL Analysis

The Time-To-Live parameter was of special interest to us, as it could be an indicator of fast flux usage. Fast flux is used as a means to provide flexibility among the C&C infrastructure of bots [41].

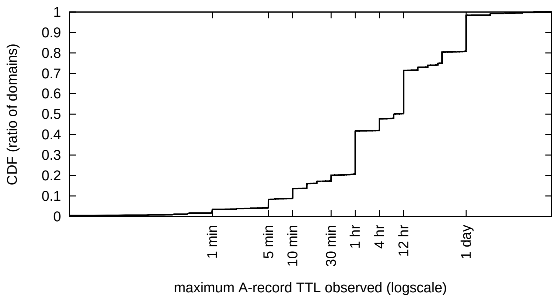


Figure 4.4: CDF of DNS TTL per domain

Figure 4.4 shows that 10% of all domains have a maximum TTL of 5 minutes or less. As spotted elsewhere [41], we expected domains with a small TTL and a large set of distinct answer records to be fast-flux candidates. However, when inspected manually, we found many domains of content distribution networks and large web sites. Using small TTLs seems to have become common among web hosters. As a result, the distinction between malicious fast-flux networks and legitimate hosting services becomes much more difficult. A couple of responses with a TTL of zero looked themselves like C&C communication. This way, we found Feederbot [31], the first publicly discussed botnet that used DNS as C&C channel. Feederbot's responses can be characterized by atypically TXT record values. The TTL of zero prevents caching of these responses, effectively causing the resolver to always fetch the newest response from the authoritative DNS server. All in all, DNS suits well as a low-profile, low-bandwidth C&C channel in heavily firewalled environments. Only our detailed malware network behavior understanding has enabled for such an in-depth analysis of Feederbot.

4.5.3 DNS Message Error Rate

In order to measure DNS-transaction failure, we defined the *DNS request error rate* as the number of DNS requests that were not successfully resolved over the total number of DNS requests. When aggregating the DNS message error rate per sample, we realized that for 10.1% of the samples in S_{Net} all of their DNS resolution attempts fail. However, the majority of the samples in S_{Net} (60.3%) have all DNS queries successfully resolved. The complete CDF is provided in Figure 4.5. Zhu et al. suggested the high rate of NXDOMAIN as one feature to detect malware samples [102]. Our results indicates that this is possible only for the minority of malware samples. However, the high DNS message error rates of a few samples may indicate the usage of domain name generation algorithms (DGAs). Malware using DGAs often fails to find randomly chosen DGA generated C&C domains, as botmasters typically register only a small fraction of all generated domain names. The fact that 60% of the samples never fail in name lookups indicates that most malware families either do not use DGAs at all, or only fall back to this mode when the primary C&C channel is not reachable anymore. However, it also shows that the high DNS-failure rates of particular malware families may in fact identify DGA variants, as shown by Antonakakis et al. [9].

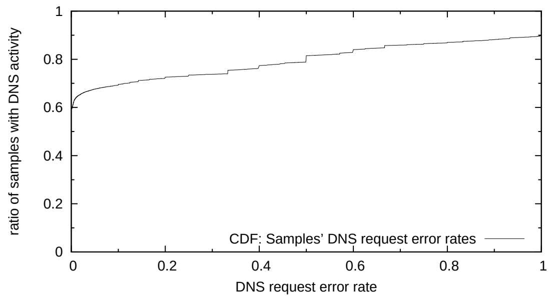


Figure 4.5: CDF of DNS message error rate

4.5.4 Resource Record Type Distribution

Figure 4.6 shows the distribution of the Resource Record types of the query section. Obviously, A records dominate DNS queries in SANDNET traffic, followed by queries for MX records. All samples in S_{DNS} have queried for an A record at least once. The high prevalence of A records is expected as A records are used to translate domain names into IP addresses. Furthermore, 2.3% of the samples in S_{DNS} queried blacklists. MX records have been queried by far fewer samples (8%). Interestingly, when comparing the MX query rate with SMTP activity, we have seen both: samples that performed MX lookups but had no SMTP activity and samples that successfully used SMTP but showed no MX queries at all. We assume that in the latter case, the required information on the MX destinations is provided via other means, for example, the C&C channel.

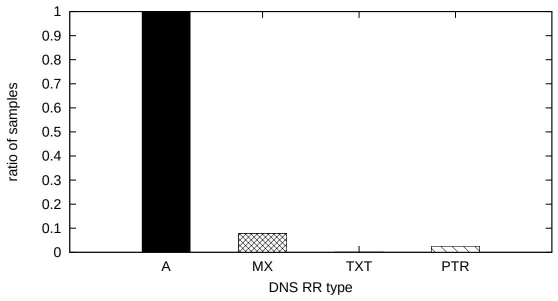


Figure 4.6: DNS Resource Record distribution among samples

4.5.5 Resolution for Other Protocols

DNS, though itself a layer-7 protocol, plays a special role as it provides resolution service to all other layer-7 protocols. We analyzed how malware uses DNS before connecting to certain destinations. 23% of the samples in S_{DNS} show at least one flow without prior DNS resolution of the destination (DNS flows and scans excluded). In such a case either the destination's IP address is known (e.g., hard-coded in the binary) or resolution takes place via some other mechanism than DNS. Furthermore, 2.3% of the samples in S_{DNS} queried blacklists (26% of these also sent spam).

4.6 HTTP

HTTP traffic sums up to 88 GB inbound and 21 GB outbound, which makes HTTP by far the most prevalent protocol in SANDNET measured by traffic. The subset of samples using HTTP is denoted by S_{HTTP}. Given the high detail of the OpenDPI protocol classification, additional protocols that are carried in HTTP traffic are treated separately and thus contribute additional traffic: The Macromedia Flash protocol sums up to an additional 32 GB, video streams like MPEG and Apple Quicktime sum up to an additional 9 GB. We observed that the protocols carried in HTTP are usually caused by embedded objects included in websites that are visited by samples.

The immense potential abuse of HTTP-driven services motivated us to perform an in-depth analysis of typical malware HTTP traffic. Not only botnets started using HTTP as C&C structures. To name but a few, click fraud (i.e., the abuse of advertising services), mail address harvesting, drive-by downloads and DoS attacks on web servers are malicious activities of a wide range of malware authors. Of all samples with network activity (S_{Net}), the majority of 58.6% exposed HTTP activity. This section provides details to which extent, how, and why malware typically utilizes the HTTP protocol.

4.6.1 HTTP Requests

The analyzed samples typically act as HTTP clients and contact HTTP servers, mainly because the Sandnet communication is behind a NAT gateway. Figure 4.7 gives a general overview of how many HTTP requests malware typically made during the analysis period. The number of requests gives us a lead for which role malware has. Click fraud campaigns or DoS activities cause a large number of requests. Malware update functionality and C&C channels potentially need little HTTP activity only. About 65% of the samples in S_{HTTP} made more than 5 HTTP requests. 16.3% of the samples in S_{HTTP} made only one HTTP request and then stopped their HTTP activity, although 70% of these samples continued with other network activity. We manually checked a fraction of these cases and found that many samples use HTTP to load second-stage binaries and continue with non-HTTP based damage functionality.

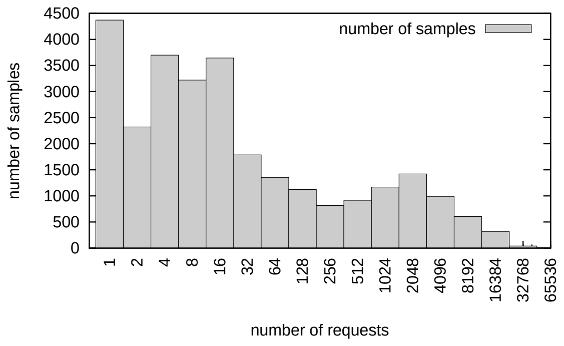


Figure 4.7: Histogram of HTTP request distribution

The GET request method was used by 89.5% of the samples in S_{HTTP}. We observed that 72% of the samples in S_{HTTP} additionally included GET parameters. Analyzing just the fraction of GET requests with parameters, GET requests have on average 4.3 GET parameters. The average size of a GET parameter was 12 characters for the key and 33.3 characters for the value. Although other means (such as steganography) allow to pass data to the sever, GET parameters seem to remain a popular method. On average, we have observed 1966 GET requests per sample with at least one request parameter. Interestingly, the number of unique GET parameter keys used by a sample is significantly lower than the total number of GET parameters per sample. This trend is particularly strong for samples with many parametrized GET requests and indicates that parameter keys are reused for follow-up requests. On average, the ratio between the number of distinct GET parameter keys and the total number of GET parameters is merely 1:16. Perdisci et al. have shown that such detailed analyses help to detect malicious HTTP requests [74].

The POST request method was used by 56.3% of the samples in S_{HTTP}. The average body size of POST requests is 739 bytes. We manually inspected a randomly chosen fraction of POST bodies to find out for what purpose malware uses POST requests. A large fraction of the inspected POST requests was used within C&C communication with a botnet server. We commonly observed that data passed to the server was base64-encoded and usually additionally obfuscated/encrypted. In addition, we frequently saw POST requests directed to search engines.

42% of the samples in S_{HTTP} used both POST and GET requests. Only 0.9% of the samples in S_{HTTP} showed HEAD requests at all. All other HTTP methods were used by less than 0.1% of the samples in S_{HTTP} and seem insignificant.

4.6.2 HTTP Request Headers

Table 4.4 lists the 30 most popular HTTP request headers as observed in SANDNET. These HTTP headers include common headers usually used by benign web browsers. In total, we have observed 144 unique HTTP request headers. At a closer look at these, we identified a significant amount of misspelled or custom headers (excluding all extension headers, that is, those starting with *'X-'*). Manual inspection shows that many of the less-frequently used headers look suspicious. Merely 5.7% of all samples in S_{HTTP} sent an HTTP request without any header at all. Consequently, analyzing HTTP request headers could be a promising angle for network-based malware detection.

User-Agent

In benign applications, the HTTP *User-Agent* header specifies which exact web browser (including its version number) is requesting web content. However, HTTP clients and thus also malware can customize the User-Agent values in the header. Table 4.5 gives a detailed list of the 30 most popular raw User-Agent strings observed in SANDNET. Most samples (98.6% of S_{HTTP}) specified a User-Agent header at least once.

In an approach to get an overview of *actual* user agents we developed heuristics to filter the User-Agent list. First, we observed that 29.9% of the samples in S_{HTTP} specified wrong operating systems or Windows versions in their forged HTTP User-Agent headers. Next, we identified that at least 13.4% of the samples in S_{HTTP} claim to use nonexisting browser versions (e.g., *wget 3.0* or *Mozilla 6.0*). In addition, we saw that 37.8% of the samples in S_{HTTP} specified malformed or very short and to us unknown User-Agent values. In total, 67.5% of the samples in S_{HTTP} transmitted at least once a suspicious User-Agent string. Over the whole analysis period, only 31% of the samples in S_{HTTP} specified apparently correct User-Agent strings.

This result suggests that most samples have their own HTTP components which are bad in forging real web browsers. Interestingly, about half (50.6%) of the samples in S_{HTTP} change or alternate the User-Agent header during their analysis period. We hypothesize that this is due to the modular architecture of

HTTP Header	**Samples**	**HTTP Requests**
Host	27,771	21,054,208
User-Agent	26,359	20,923,840
Connection	21,205	20,570,434
Cache-Control	18,529	1,346,260
Accept	18,483	20,554,040
Content-Length	14,811	977,547
Accept-Encoding	14,406	19,424,065
Content-Type	14,135	1,033,111
Accept-Language	11,382	18,319,897
Referer	10,079	18,311,670
Cookie	10,075	10,939,127
If-Modified-Since	5462	3,044,837
If-None-Match	4696	1,005,364
x-flash-version	4386	464,334
Pragma	4290	73,427
x-requested-with	2079	14,329
Range	1597	15,451
If-Range	1006	3882
Unless-Modified-Since	962	3868
Accept-Charset	922	69,908
X-Agent	658	36,302
Keep-Alive	642	87,149
X-Moz	511	517

Table 4.4: SANDNET: HTTP headers

malware, where the modules have inconsistent User-Agent strings. Furthermore, based on this observation, we suspect that malware adapts the User-Agent header (and possibly other headers) depending on the target website.

Localization Headers

HTTP requests typically include headers that tell the server which languages and character sets the client requests (*Accept-Language* and *Accept-Charset*). We inspected these two localization headers and compared them with the locale setting of the sandpuppets (German). While the Accept-Charset header was used by 0.35% of the samples in S_{HTTP}, the Accept-Language values are more interesting to analyze: In total, 44.3% of the samples in S_{HTTP} included Accept-Language as an HTTP request header. Of these samples, 24.1% did not respect the locale setting and specified a non-German language. Chinese (zh) and English

User Agent	**Requests**	**Samples**
Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident)	17,193,201	11,168
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)	861,353	5628
Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident)	1,937,020	5376
Microsoft-CryptoAPI/5.131.2600.5512	17,581	3485
Mozilla/6.0 (Windows; wget 3.0)	12,851	3242
Download	5022	2042
Mozilla/4.0 (compatible; MSIE 8.0.6001; Windows NT 5.1)	23,022	1802
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)	12,569	1546
ClickAdsByIE 0.7.3	34,615	1208
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)	69,078	992
XML	3403	891
Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1)	1714	849
PinballCorp-BSAI/VER_STR_COMMA	3454	771
Mozilla/3.0 (compatible; Indy Library)	71,971	761
Microsoft Internet Explorer	8652	750
gbot/2.3	22,791	694
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET)	23,772	608
_	5327	589
NSISDL/1.2 (Mozilla)	692	535
Microsoft-ATL-Native/9.00	3827	524
Mozilla/5.0 (Windows; U; Windows NT 5.1; de; rv:1.9.2.6) Gecko	31,078	514
Mozilla/4.0 (compatible)	6004	487
Mozilla/4.0 (compatible; MSIE 8.0; 10.1.53.64; Windows NT 5.1)	884	426
NSIS_Inetc (Mozilla)	515	403
wget 3.0	3917	339
Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET)	946	311
opera	946	300
Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9) Gecko	6764	300

Table 4.5: Sandnet: HTTP User-Agent header

(en) are the foreign languages specified most frequently, followed by Russian (ru). We speculate that in these cases malware authors forge HTTP headers either as observed at their own local systems or with respect to the target website. This would depict yet another indicator that malware carries its own (possibly self-made) HTTP implementation. Another reason could be that malware authors explicitly specify foreign languages to hoax web servers.

4.6.3 HTTP Responses

In Sandnet, all HTTP responses observed originated from HTTP servers on the Internet that were contacted by a sample. Therefore, the following analysis is not an analysis of the samples themselves, but may give indications to which type of servers malware communicates.

We observed that 97.8% of the HTTP requests were answered with an HTTP response. We define the *HTTP error rate* as the ratio between failed responses (HTTP status codes 4XX and 5XX) and all responses. Figure 4.8 shows a distribution of the sample-wise HTTP error rate. Only a small fraction (less than 10%) of samples never succeed in obtaining web content. Most samples have a relatively small error ratio, indicating the web sites requested by the samples are still in place. We will give an overview of the requested servers in Section 4.6.6.

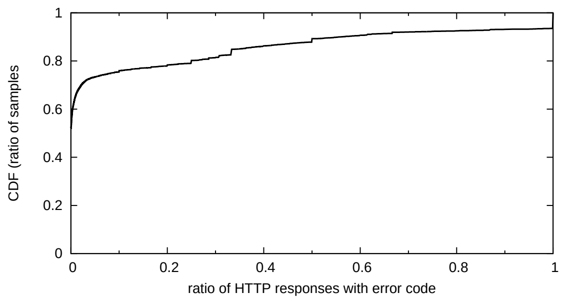


Figure 4.8: Distribution of HTTP error rates among samples

4.6.4 HTTP Response Headers

As opposed to HTTP request headers, response headers are set by servers and are not chosen by the malware samples. Analyzing the headers helps us to understand which servers are contacted by malware samples and gives information about the type of the retrieved content.

Content-Type

The Content-Type header shows which type of web content was retrieved by the samples. Figure 4.9 shows that most samples at least retrieve web sites with Content-Type *text/**. By far the most popular content-type of textual responses is *text/html*. However, only about half of all samples retrieved rich documents with Content-Type set to *images/** (48%) or *application/** (59.4%). 23.9% of the HTTP active samples with more than a single request got textual responses only. We see two reasons for such presumably light HTTP clients: First, spidering web sites without loading images is much more efficient. Second, we hypothesize that a considerable number of samples lacks a full-blown HTTP implementation that can recursively fetch objects embedded in web sites.

Server

The *Server* HTTP response header indicates which type of web server is responding to the malware's HTTP request. Note that the content of this header can again be forged. Moreover, the majority of contacted web servers is presumably benign. However, when manually inspecting the HTTP Server response header,

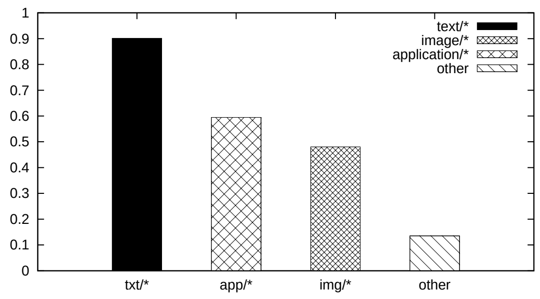


Figure 4.9: Ratio of samples using given Content-Type

we spotted servers that presented suspicious banner strings. Table 4.6 summarizes the list of the 30 most popular server types observed in SANDNET.

4.6.5 HTTP Responses with PE Binaries

After compromising a system with minimized exploits, attackers usually load so-called second-stage binaries. These binaries carry the actual malware functionality rather than just the exploit with minimized shell-code. We will discuss malware downloaders in more detail in Chapter 5. In this chapter, we will analyze HTTP plaintext malware binary downloads only.

We identified unencrypted downloads of Windows executable files by searching for valid headers of Microsoft's Portable Executable (PE) file format. All Windows executables are in the PE format, which can be identified by two constant bytes in the beginning of the header. We extracted all binaries downloaded via HTTP by matching HTTP response bodies against these two bytes. This straightforward extraction of PE binaries already discovered that 16.7% of the samples in S_{Net} loaded additional PE files. We observed that 19% of these samples load binaries for multiple times - occasionally even more than 100 times. We verified that the five binaries downloaded most often were not corrupt and lack reasonable explanations why the binaries were downloaded that often. In total, we detected 42,295 PE headers, resulting in 17,676 unique PE files.

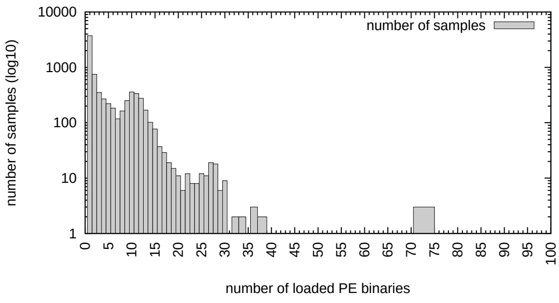


Figure 4.10: Distribution of # of PE binaries loaded

Server	Ratio (%)	Servers
Apache	68.4	326,237
Microsoft-IIS	49.4	102,652
nginx	40.9	108,104
Golfe	21.4	20,534
lighttpd	21.4	32,934
YTS	20.0	28,320
sffe	19.4	15,128
GFE	18.3	21,089
Apache-Coyote	17.6	41,875
QS	15.4	6906
PWS	14.7	16,297
DCLK-AdSvr	13.9	6782
cafe	13.7	11,399
AmazonS3	13.7	17,203
ADITIONSERVER 1.0	10.9	6092
AkamaiGHost	10.3	3520
Cookie Matcher	10.1	4011
gws	9.5	6075
VM_BANNERSERVER 1.0	9.4	2620
CS	9.1	3987
Adtech Adserver	8.9	4842
CacheFlyServe v26b	8.0	2242
RSI	7.8	2196
yesup httpd 89	7.8	2160
yesup httpd 103	7.7	2151
Resin	7.7	4375
ECS (fra	6.7	7615
Oversee Turing v1.0.0	6.1	2579
JBird	6.0	1987
TRP Apache-Coyote	5.7	1966

Table 4.6: Sandnet: HTTP server banners

Figure 4.10 shows that most of the samples load more than a single PE binary. For readability of the graph we precluded 21 samples that loaded more than 100 and up to 1080 unique PE binaries.

The maximum size of a downloaded binary was 978 kB, the average size is 144 kB. Table 4.7 summarizes the Content-Type values of all HTTP responses that contain PE binaries. Most samples retrieve reasonable Content-Type values from the server. However, a significant number of servers tries to camouflage PE binary downloads as text, HTML, JavaScript or image files. We will analyze this behavior and discuss PE file downloads via encrypted communication in Chapter 5.

Content-Type	**# Binaries**	**# Samples**
application/octet-stream	6468	5908
text/plain	356	1716
application/x-msdownload	732	1082
application/x-msdos-program	550	786
image/gif	177	402
image/jpeg	390	365
text/plain; charset=UTF-8	166	344
text/html	776	326
application/x-javascript	190	78
image/png	68	55

Table 4.7: Sandnet: HTTP Content-Type header values of PE downloads

4.6.6 HTTP Servers

We next analyze which HTTP servers are visited by malware. Table 4.8 lists the 40 most popular domains ordered by the number of different samples visiting a server. Many HTTP requests were put to presumably benign web sites. The next sections briefly discuss why malware contacts these services.

Ad Services

We identified a significant number of ad service networks in the list of popular domains. Of the top-40 domains in Table 4.8, we manually identified 32 domains that are related to ads. Thousands of different malware samples use these services. A possible reason for this is that ads are also included in benign websites and crawlers follow the ads. However, after manually exploring the HTTP traffic of particular samples we assume that the reason for the popularity of ad services is vicious: click fraud. We leave it up to future work to analyze and mitigate the abuse of ad services by malware samples in greater detail.

Public Web APIs

Similarly to its popularity among benign users, Yahoo's and particularly Google's public Web APIs are present in Sandnet traffic, too. We suspect there are two reasons behind the popularity of these or similar services. First, some of these

HTTP domain	**# Samples**
`www.google-analytics.com`	5286
`ad.yieldmanager.com`	5046
`cookex.amp.yahoo.com`	4716
`content.yieldmanager.com`	4655
`ak1.abmr.net`	4288
`pixel.quantserve.com`	4050
`content.yieldmanager.edgesuite.net`	4009
`edge.quantserve.com`	3957
`ad.doubleclick.net`	3677
`ad.harrenmedianetwork.com`	3470
`ad.103092804.com`	3458
`s0.2mdn.net`	3370
`ib.adnxs.com`	3280
`pixer.meaningtool.com`	3219
`ad-emea.doubleclick.net`	2972
`www.google.com`	2940
`ad.harrenmedia.com`	2920
`www.mupimg.de`	2823
`imagesrv.adition.com`	2770
`www.mupads.de`	2759
`view.atdmt.com`	2754
`ad.xtendmedia.com`	2726
`cm.g.doubleclick.net`	2669
`googleads.g.doubleclick.net`	2657
`fpdownload2.macromedia.com`	2619
`www.myroitracking.com`	2573
`serw.clicksor.com`	2489
`ad.adition.net`	2468
`ads.clicksor.com`	2466
`ad.tlvmedia.com`	2449
`ad.adserverplus.com`	2414
`b.scorecardresearch.com`	2376
`pub.clicksor.net`	2375
`ajax.googleapis.com`	2335
`img.billiger.de`	2308

Table 4.8: SANDNET: HTTP servers contacted most frequently

services are ubiquitous on the Internet. For example, a wide variety of web sites include Google Analytics to record statistics on the visitor behavior. Each time a sample visits such a web site and follows the embedded links, it will contact Google. As most of such services are open to anyone, we also suspect malicious usage of Google's and Yahoo's services by malware samples to be a reason for their popularity. A typical scenario that we observed was the abuse of search engines as a kind of C&C engine. In this case the malware searched for specific keywords and fetched the web sites suggested from the search results. Moreover, we have observed malware using the search engines to harvest new e-mail addresses for spamming campaigns. In general, benign human interaction with these services is particularly hard to be distinguished from abuse, especially from the client-perspective. We assume this is one of the main reasons malware authors use these HTTP-based services.

PE File Hosters

Based on the set of PE files that malware samples downloaded, we analyzed the file hosting servers. Table 4.9 lists the most popular of all 1823 PE file hosters that we identified. 42.3% of the samples that downloaded PE files contacted the PE host directly without prior DNS resolution. This proves that still a significant number of malware samples include hard-coded IP addresses to download binaries. We further observed that a significant fraction of the URIs requested from file servers are dynamic, although frequently only the parameters change. This observation may be important for blacklists trying to block entire URIs instead of IP addresses or domains.

HTTP C&C Servers

HTTP based botnets such as Torpig [86] switched from the classical IRC protocol to using HTTP. While manually inspecting SANDNET HTTP traffic, we occasionally encounter C&C traffic. What we see most is that samples include infection status information in their GET request parameters. Whereas some samples include clear-text status information, we have observed many samples started encoding and encrypting the data exchanged with the server. However, we found it difficult to automatically spot C&C servers without knowing the command syntax of specific botnets. The big difference to IRC is that HTTP is a prevalent protocol on clean, noninfected systems and is thus harder to spot in the volume of HTTP data. Encouraged by the results reported by Cavallaro et al. [27] and Perdisci et al. [74], we believe that clustering the network behaviors of malware may help us in spotting generic C&C communication channels.

4.7 Related Work

The malware phenomenon has been considerably studied over the last years by researchers and security practitioners. Numerous techniques have been proposed to collect malware via honeypots like Nepenthes [11], analyze malware [17, 27, 35, 72, 74, 94], or detect malware [27, 74].

PE File Server	**#S**	**#B**
64.79.86.26	775	1340
66.96.221.102	681	1063
ku1.installstorm.com	487	944
origin-ics.hotbar.com	483	483
64.191.44.9	480	727
img.ub8.net	458	460
pic.iwillhavesexygirls.com	437	478
64.120.232.147	431	747
origin-ics.clickpotato.tv	390	390
p2pshares.org	389	391
sky.installstorm.com	363	363
208.43.146.98	331	531
file0129.iwillhavesexygirls.com	323	853
173.45.70.226	315	437
173.45.70.227	313	444
dl.ghura.pl	302	302
122.224.6.48	300	553

#S *= number of samples contacting file hoster*
#B *= number of binaries downloaded*

Table 4.9: Sandnet: HTTP servers hosting PE binaries

For instance, Perdisci et al. [74] present an interesting system to cluster network-level behavior of malware by focusing on similarities among malicious HTTP traffic traces. Similarly, Cavallaro et al. [27] present cluster-based analyses aimed at inferring interesting payload-agnostic network behaviors of malicious software. While Sandnet is currently limited to analyzing a large corpus of network protocols, it is clear how the adoption of similar cluster-level analyses can provide better understandings of the network behaviors of unknown software.

Anubis [15, 17] and CWSandbox [94] are probably the closest works related to our research. Although they both provide interesting—but basic—network statistics, their main goal is to provide insights about the host behaviors of unknown—potentially malicious—software. In this context, Sandnet complements Anubis and CWSandbox results by providing an in-depth analysis of the network behaviors of the analyzed samples. We will use the network behavior analysis that we obtain from Sandnet to understand botnet resilience in this thesis.

4.8 Discussion

We have designed Sandnet such that it mostly conforms to the guidelines in Chapter 3. For example, to ensure realistic experiments, we aim to maximize the diversity of malware families that we analyze by using multiple sources of malware samples. Consequently, we saw the need to filter legitimate software from our dataset and used a variety of checks to exclude benign samples. To mitigate the damage of malware executed in Sandnet, we deployed containment policies that filter harmful traffic such as spam or DDoS attacks. But at the same time Sandnet remains a realistic malware-analysis environment. Bots can still access their C&C servers as if they were executed on real systems.

However, a few guidelines were not yet explicitly addressed in the design of Sandnet. For example, this chapter analyzed malware samples using a Windows XP VM behind a NAT gateway only, such that we cannot give an overview of how malware for Windows Vista/7 with public Internet access would behave. Similarly, the statistics in this chapter are not balanced over malware families.

Parts of these decisions were made to remain comparable to similar work, such as data derived from Anubis by Bayer et al. [15]. In addition, the results presented in this chapter predate the publication of our work presented in Chapter 3. This does by no means invalidate the results presented in this chapter. In fact, the experiences we made during the work presented in this chapter significantly influenced the development of the guidelines for malware experimentation. Moreover, dynamic analysis environments and the succeeding data analysis undergo steady developments. For example, we added support for Windows Vista/7 and Windows XP 64bit to Sandnet in October 2012, and we integrated sandpuppets with public IP addresses in Sandnet. Having said this, some of the guidelines presented in Chapter 3 do not apply to analyzing Sandnet data itself. For example, an evaluation of false positives or true positives was not relevant in this chapter. In the following chapters, we will use Sandnet for analyzing botnet resilience, imposing specific requirements on the analysis datasets. Assembling balanced, correct and realistic datasets is a laborious task, and we need to take such experimental specifics into account. We will therefore discuss how we assemble our datasets in each of the subsequent chapters individually.

4.9 Conclusion

In this chapter, we have presented Sandnet, a system to dynamically analyze the network behavior of malware. We gave a comprehensive overview of network traffic exposed by more than 100k malware samples that we analyzed between Februar 2010 and January 2011. Our in-depth analysis of DNS and HTTP traffic has shown novel malware trends and led to numerous inspirations to combat and analyze malware. The provided data complements related work that is either outdated, analyzes particular malware families only, or focuses mainly on the host behavior of malware.

In subsequent chapters, we will use Sandnet to analyze the resilience of malware families. The design of Sandnet, particularly its in-depth analysis of C&C communication, allows us to easily identify and monitor malware families. In Sandnet, we classify unknown malware samples by, for example, applying payload signatures. Similarly, we can automatically execute malware samples of a specific family on a regular basis, observing its changes over time. These fine-grained analyses sets Sandnet apart from other malware analysis environments, which makes it suitable for the following botnet resilience research.

Part II

Botnet Resilience

Botnet Resilience Analysis

In Part I, we have established a solid methodology for malware analysis. We will now use these insights to analyze the resilience of botnets.

In Chapter 5, we explore the resilience of malware installations networks. *Malware downloaders* are one of the root causes for botnets, as they offer other botmasters an easy way of buying new bots for their botnet. Hence, the resilience of botnets also largely depends on the availability of malware downloaders. Related work by Caballero et al. details the monetization of malware downloaders in pay-per-install markets [25]. However, little research was done to understand if and why these networks are resilient. To close this gap, we monitor 23 malware downloaders over multiple years. Our observations will detail some techniques that botmasters use to keep their malware downloader networks operational on long term.

In Chapter 6, we will then analyze the resilience of peer-to-peer (P2P) botnets. Feeling the pressure by takedowns of centralized botnets, botmasters designed C&C architectures that are largely independent from C&C servers. Consequently, P2P botnets can be highly resilient, as they lack a single point of failure. However, with Waledac and Storm, two P2P botnets were successfully disrupted by abusing weaknesses in the P2P protocols. Following these first real P2P botnets, a number of academic, supposedly more resilient P2P botnet designs were proposed. Since then, more than ten new P2P botnets appeared in the wild, with a variety of proprietary P2P protocols. We will analyze the resilience of these fairly undocumented P2P botnets, showing that they are more resilient than preliminary P2P botnet designs.

5 Resilience Analysis of Malware Downloaders

Malware downloaders are malicious programs with the goal to subversively download and install malware (*eggs*) on a victim's machine. When reflecting the malware lifecycle model (Section 2.1 on page 5), malware downloaders and the corresponding malware installation infrastructure manifest the installation phase of many malware families. Therefore, in this chapter, we will analyze the resilience of malware downloaders and their infrastructures. In particular, we analyze and characterize 23 Windows-based malware downloaders. We first show a high diversity in downloaders' communication architectures (e.g., P2P), carrier protocols and encryption schemes. Using dynamic malware analysis traces from over two years, we observe that 11 of these downloaders actively operated for at least one year, and identify 18 downloaders to be still active. We then describe how attackers chose resilient server infrastructures. For example, we reveal that 20% of the C&C servers remain operable on long term. Moreover, we observe steady migrations between different domains and domain registrars, and notice attackers to deploy critical infrastructures redundantly across providers. After revealing the complexity of possible counter-measures against downloaders, we present two generic techniques enabling defenders to actively acquire malware samples. To do so, we leverage the publicly accessible downloader infrastructures by replaying download dialogs or observing a downloader's process activities from within the Windows kernel. With these two techniques, we successfully milk and analyze a diverse set of eggs from downloaders with both plain and encrypted communication channels.

5.1 Introduction

A crucial part in a malware's lifecycle is to spread, for example, via spam, drive-by downloads or exploiting vulnerabilities. Whereas malware such as worms spreads on its own, attackers have begun to separate the task of infecting victim systems and the exploitation or "monetization" of the infected systems. Recent investigations to this business, known as "Pay-per-Install" (PPI), have shown the vast potential of this kind of malware distribution model. Caballero et al. [25] analyzed PPI networks by infiltrating and by becoming member of a handful of PPI programs. The authors showed that PPI networks are responsible for installing a diverse set of malware on infected systems.

Technically, the PPI scheme is only a subset of the malware type that we term a *downloader*. A downloader is a malicious program with the purpose to subversively download and install malware on a victim's machine. The specifics of PPI networks allow attackers to get paid on a per-system and per-affiliate basis, but the effect of PPI or, more generally, downloaders is comparable: Once a downloader is executed, and no matter if related to a PPI network or not, the running system will typically be compromised with additional malware families. Thus, downloaders represent a simple yet widely used way to spread new malware, typically as part of a service model within the underground community.

In this chapter, we outline and analyze the landscape of what we think represents a snapshot of prevalent and current downloaders. We identified 23 downloaders, of which many – to the best of our knowledge – have not yet been documented. We characterize these downloaders concerning their communication model. For example, we discuss the communication architectures of downloaders and outline the techniques used to encrypt or even camouflage the malicious activities. We then use dynamic analysis traces to provide a long-term monitoring analysis on these 23 downloaders, identifying 18 downloaders that are still operational as of writing this paper. In addition, we show that eleven downloader families are actively distributing malware for more than a year.

Motivated by this observation, we investigate how attackers ensure the resilience of downloader infrastructures. Contrary to our expectation that IP address blacklists would force attackers to change their infrastructure frequently, we show that 219 C&C servers (20%) were operational for more than four weeks. For the remaining servers, we analyze how attackers use DNS and IP address fluxing to operate their downloaders, suggesting that isolating downloader infrastructures is much harder than it seems.

As a third part of our analysis, we propose two automated methods to extract the downloaded malware (*eggs*) in a generic and scalable fashion. We hope that these techniques will support future efforts in analyzing downloaders without the manual effort of reverse engineering particular downloader families. We evaluate these two techniques both on downloaders with plaintext and encrypted communication, acquiring a diverse set of malware in the wild.

5.2 Malware Downloaders

Malware defense mechanisms, especially anti-virus, have forced attackers to develop increasingly complex malware. This complexity has motivated attackers to specialize and separate duties. For example, services to stealthily install malware on computers may be provided by one group, while other fraudsters specialize in sending spam, and a third group could focus on keylogging. In this work, we focus on the service of installing new malware on systems via *downloaders*.

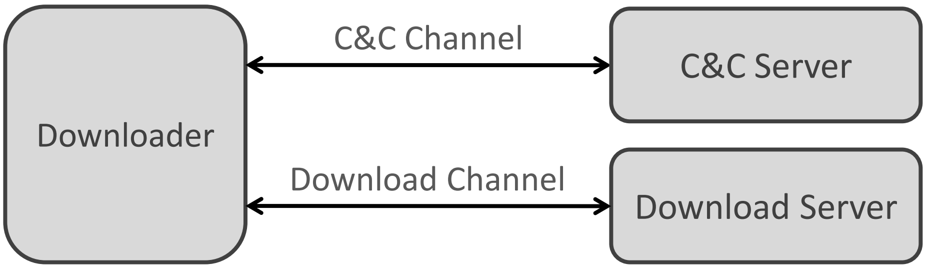


Figure 5.1: Downloader architecture: Seperation between C&C and download channel.

Downloaders are malicious programs that are instrumented to load additional malware via the Internet, which is in turn executed on the victim's system.

5.2.1 Downloader Architectures

Figure 5.1 illustrates the architecture of a downloader. Once executed, a downloader contacts its command-and-control (C&C) server(s) via *C&C channels*. After receiving download instructions, it then establishes at least one *download channel* to load malware (*eggs*) via the network.

C&C Channels

A downloader's C&C channel is used to get lists of URLs (or similar address information) where eggs can be downloaded from. Next to download instructions, the C&C channel can be used to report back to the C&C server if the download succeeded. In addition, as shown by Caballero et al. [25], C&C channels may exchange affiliate IDs in the economical model of pay-per-install downloaders. Moreover, downloaders send details about the infected host using the C&C channel, such as the OS version, user name or device IDs.

We characterize C&C channels by the *carrier protocol* they use to transfer commands. Typical examples for carrier protocols are IRC, HTTP (e.g., if C&C messages are in the HTTP body) or plain TCP/UDP. The information exchanged on C&C channels is critical and highly subjective to counter-measures such as signature-based IDSs, and thus more advanced downloaders encrypt their C&C channel. During our investigations, we also observed downloaders that carry hard coded download URLs in their binaries. We excluded such downloaders from our analysis because of their simplicity and transitory nature.

Download Channels

We found the C&C and download channels to be typically well separated. A typically distinguishing characteristic between C&C and download channels is the number of bytes transferred. C&C commands tend to be small, while eggs – no matter if encrypted or not – have significantly larger file sizes. In addition, download channels may have different carrier protocols and encryption schemes than the C&C channel of the same family. For example, as we will show, some downloaders camouflage their downloads in seemingly legitimate web traffic.

5.2.2 Related Work

First steps to analyze specific downloaders were made by Caballero et al. by analyzing four pay-per-install (PPI) programs [25]. PPI downloaders, a subset of downloaders in general, are based on an economical model cashing out attackers for installing malware on a freshly infected system. Caballero et al. implemented so called *milkers* to download eggs from these four PPI networks, and systematically analyze the ecosystem behind these networks. They show in depth how egg families relate to download programs, and identified that kinds of malware (e.g., DDoS) were distributed in download campaigns.

Our work was inspired by Caballero et al., and we seek for a broader characterization of downloaders. In fact, we found ourselves at a position not knowing the magnitude and different types of downloaders currently active in the wild. We identify that the number and kinds of downloaders is significantly higher than expected. To the best of our knowledge, we are the first to approach a characterization of downloaders. We then also seek to answer the fundamental but yet unanswered question of how attackers build up infrastructures that are sufficiently resilient for long-term operations of downloaders. We expand a malware acquisition technique as proposed in Botlab [46] with replaying network dialogs as proposed by Newsome et al. [71]. While Botlab fetches malware from URLs found in spamfeeds, our technique repeatedly acquires malware from downloader URLs. Existing systems like ThreatExpert [5] or Anubis [17] can already analyze malware in general, but we are the first to analyze the behavior and infrastructures of downloaders over multiple executions and on long-term.

5.3 Analysis of the Downloader Landscape

In this section, we characterize and describe the 23 downloaders identified as part of this work, which we will then further analyze later in this chapter.

5.3.1 Dataset Description

We base our analysis on malware reports from SANDNET (Chapter 4). We assembled our dataset such that they conform to most guidelines presented in Chapter 3. The following description of the dataset covers these guidelines in detail. SANDNET executes and dynamically analyzes malware using Windows XP SP3 32bit virtual machines connected to the Internet via NAT. The data obtained from this specific setting can thus not be used to generalize to the download landscape on operating systems, although we manually verified that some downloaders also run on alternative operating systems (e.g., Windows Vista/7) and on 64bit architectures. During malware execution, we deploy containment policies that redirect harmful traffic (e.g., spam, infections) to local honeypots. We further limit the number of concurrent connections and the network bandwidth to mitigate DoS activities. An in-path honeywall NIDS watched for security breaches during our experiments. Other protocols (e.g., IRC, DNS or HTTP) were allowed to enable C&C communication. We consider the biases affecting the following experiments due to containment to be negligible. Specifically, given our long measurement period of one hour per malware sample, we did not observe any incomplete download behavior in our trace. Assuming that downloaders silently operate without the user's consent, we did not deploy user interaction during our experiments.

Our dataset consists of 243,000 MD5 unique malware samples analyzed in Sandnet at least once between February 2010 and February 2012. We gratefully received these samples from a variety of sources, including samples submitted to public dynamic analysis environments, feeds by security companies, our own honeypot infrastructures and spamtraps. While we cannot prove that this dataset covers all relevant malware families, it shows a diversity of 38,000 unique malware labels (according to Kaspersky). We extracted the malware family names from these labels and found over 1800 malware families in our dataset, likely covering all relevant malware families.

From this diverse set of samples, we scheduled a random selection on a daily basis, without giving any emphasis to particular malware families. To trigger the malware behavior, we then executed these samples for at least one hour. Obviously, only a minor fraction of these malware samples are in fact downloaders. To build up a dataset covering the most relevant downloaders we followed a threefold approach. First, we consulted literature research and asked A/V vendors for their expert knowledge on recent and prevalent downloaders. Second, in our dataset covering millions of malware samples, we searched for prevalent A/V labels suggesting the malware is a downloader. Third, we manually inspected a random subset of the Sandnet analysis reports for downloader behavior. We manually filtered legitimate programs in our dataset of potential downloaders, such as Windows Update, Google Updater or programs to update system drivers.

For each identified downloader, we systematically searched for related analysis reports in Sandnet. Typically, we used payload or behavioral signatures to classify and recognize a particular downloader. In rare cases, where a downloader family did not expose any signature, we carefully assembled sets of domains and IP addresses to recognize downloader traffic. Using these techniques, we are able to detect all previous and upcoming executions of a particular downloader family. A list of MD5 binary checksums for each downloader family can be found at `http://christian-rossow.de/files/dataset-downloader-chapter.txt`.

5.3.2 Downloaders Overview

The resulting dataset provides an empirical overview of existing downloaders. Table 5.1 lists the downloaders that we monitor as part of this work. While this is not necessarily complete, it shows a large diversity in terms of different downloader characteristics. The attributes in Table 5.1 form two groups: The left-hand attributes characterize the C&C channel, while the right-hand columns characterize the download channel. We labeled three downloaders with generic names (dldr-#1 to dldr-#3), as anti-virus vendors either assigned too generic or contradictory labels for those.

		C&C Channel			*Download Channel*		
Family	*arch*	*Pl?*	*Prot.*	*DNS*	*Pl?*	*Prot.*	*DNS*
Renos/Artro	cent	✗	HTTP	✓	✗	HTTP	✓
Sality	cent	✗	HTTP	✓	✗	HTTP	✗
dldr-#1	cent	✗	HTTP	✓	✗	HTTP	✓
Cycbot/Gbot	cent	✗	HTTP	✓	✗	HTTP-inl	✓
Karagany	cent	✗	HTTP	✓	✗	HTTP-inl	✓
Gamarue	cent	✗	HTTP	✓	✓	HTTP-inl	✓
Dofoil	cent	✗	HTTP	✓	✓	HTTP	✓
Emit	cent	✗	HTTP	✓	✓	HTTP	✓
GoldInstall	cent	✗	HTTP	✓	✓	HTTP	✓
Rodecap	cent	✗	HTTP	✓	✓	HTTP	✓
Virut (crypt C&C)	cent	✗	TCP	✓	✓	HTTP	✓
TDSS	cent	✗	TLS	✓	✗	HTTP	✓
Winwebsec	cent	✓	HTTP	✗	✓	HTTP	✗
Dabvegi	cent	✓	HTTP	✓	✗	HTTP	✓
Buzus	cent	✓	HTTP	✓	✗	HTTP	✓
dldr-#3	cent	✓	HTTP	✓	✓	HTTP	✓
Zwangi	cent	✓	HTTP	✓	✓	HTTP	✓
Harnig/LoaderAdv	cent	✓	HTTP	✓	✓	HTTP-inl	✓
dldr-#2	cent	✓	HTTP	✓	✓	HTTP-inl	✓
Virut (plain C&C)	cent	✓	IRC	✓	✓	HTTP	✓
Vobfus/Changeup	cent	✓	TCP	✓	✓	HTTP	✓
Sality P2P	P2P	✗	UDP	✗	✗	TCP	✗
Zeus P2P	P2P	✗	UDP	✗	✗	TCP	✗

Table 5.1: Overview of downloaders under our analysis. Columns 2–5 characterize the C&C channel, columns 6–9 characterize the download channel. “Pl?” shows if the communication channel was in plain text, and “DNS?” shows if names of the communication end points were resolved via DNS prior to contacting them.

Carrier Protocols

A first distinction between the downloaders can be made in terms of the *carrier protocol*, that is, the protocol used to communicate with C&C or download servers. To understand and also classify downloaders, we had to reassemble and parse numerous carrier protocols (UDP, TCP, DNS, HTTP, IRC, TLS). For obfuscated protocols, we define the carrier protocol to be the underlying protocol of the C&C protocol, for example, “HTTP” for GoldInstall or Renos/Artro and “TCP” for the encrypted variant of Virut C&C. Interestingly, Table 5.1 shows that C&C channels are not necessarily designed in the same way as download channels. For example, five downloaders use obfuscated or encrypted C&C channels, but at the same time have plaintext HTTP download channels. Another five downloaders do not separate between C&C and download channels, abbreviated by “inl” to show that malware is served inline with the C&C protocol.

Communication Architectures

As Table 5.1 suggests, almost all downloaders deploy a centralized C&C architecture. Two exceptions are Sality P2P and Zeus P2P. Sality uses a hybrid C&C architecture, that is, some samples use a centralized HTTP-based C&C channel while others receive their commands via a peer-to-peer network. Zeus P2P is a pure P2P based bot with download functionality. Such distributed networks are attractive to attackers, as the C&C infrastructure cannot be disrupted by taking offline single C&C servers. Both Sality P2P and Zeus P2P initialize their C&C channel by contacting tens to hundreds of P2P bootstrapping nodes. While Zeus is primarily used for ID theft, we also observed it to download and execute other types of malware.

DNS

Table 5.1 reveals that most downloaders use DNS to resolve the names of their C&C and/or download servers. However, downloader families such as Winwebsec and the P2P-driven downloaders avoid DNS resolution for both C&C and download servers. We speculate that such downloaders either have no technical need for DNS, such as for the P2P architectures, or want to foil malware domain blacklists. From the attacker's point of view, another disadvantage of using DNS is that taking down domains exposes an additional point of failure in the communication chain. However, on the other hand, DNS would allow to quickly redirect to different IP addresses of download servers. This dilemma basically boils down to: Who is more resilient, the hoster (IP) or the DNS provider (domain)? We try to shed light onto different resilience strategies in Section 5.4. The fact that most downloaders use DNS resolution shows that developing mitigation techniques based on DNS is promising. However, although we see downloaders using DNS, they may also have a backup communication channel, for example, using hard-coded IP addresses [70].

Intuitively, one may think that downloaders use DNS to quickly react on server takedowns. Fast flux [69], domain flux and the business of bullet-proof DNS hosting would support this intuition. As we figured, however, some downloaders do not (need to) change DNS records of particular C&C domains. Consequently, while the usage of domains evolved over time, the IP addresses resolved by these domains were relatively static. We will further analyze these observations in Section 5.4.1.

Communication Encryption

Defense mechanisms such as network-based intrusion detection systems or antivirus scanners scan for URLs and file contents downloaded from the Internet. As a consequence, downloaders deploy a wide set of schemes to obfuscate or encrypt their communication channels. A distinction can be made between deploying well-known or custom encryption techniques. For example, the TDSS downloader relies on TLS within its C&C channel, thus preventing from eavesdropping on C&C communication [36]. Similarly, we observed that Renos/Artro encrypts using RC4 with a key hard-coded in the samples.

In contrast, other downloaders use custom encryption/obfuscation algorithms. To give insights, we reverse engineered specific downloader families. For example, Emit deploys an XOR shifting technique to obfuscate traffic. Similarly, Virut picks a random session key and the C&C servers derive these session keys by performing a known-plaintext attack on the ciphertext of the first message sent from the bot to the server. The session key itself is thus never transmitted. Independent from the cryptographic strength of a particular algorithm, understanding and possibly decrypting the ciphertexts often requires tremendous reverse engineering efforts.

Steganography

Attackers further disguise the egg downloads with steganography. While encryption prevents eavesdroppers to read exchanged data, steganography tries to hide the existence of egg downloads. We have spotted camouflage techniques used by downloaders that could be interpreted as first steps towards steganography. For example, Renos/Artro hides its eggs in valid GIF files. Although these files look like regular legitimate pictures, eggs are carried as part of the files. Using custom routines, the downloader transforms these files to correct PE binaries.

Downloaders Using Public Services

Most downloaders rely on their own infrastructure for hosting malicious software. However, we also observed that particular downloaders make use of publicly accessible services. For example, dldr-#1 retrieves its malicious files from a large public file clouding provider. From a defender's perspective, it is much harder to block access to legitimate services, as a distinction between legitimate or malicious downloads from such sources raises big challenges.

Tracking Mechanisms

Among the plaintext downloaders, we could observe downloaders that are client-aware. That is, attackers derive pseudo-unique IDs per system, such as its MAC address, the gateway's public IP address or the Windows serial. The C&C servers can then keep track of which clients contacted them, and serve binaries accordingly. Similarly to, for example, Torpig [86], the downloader victims are presumably identified to track the number of infections, either to keep an overview or to use this data for payment (e.g., PPI). Another reason would be to observe and defend against potential abuses of the downloader infrastructures (see Section 5.5). To work around this in our setup, we are modifying fixed strings such as the MAC address for every malware execution in SANDNET since ever.

5.3.3 Downloader Lifetime

With our understanding that downloaders are a fundamental part of the malware lifecycle, we now analyze the lifetimes of the downloaders. For this lifetime analysis, we are not interested in a particular downloader *binary* (identified by the MD5 hash sum). Instead, we analyze when a particular downloader *family* appears in our dataset, and how long its C&C or download activities continue.

As a first step, we used the mechanisms described in Section 5.3.1 to identify downloaders of a particular family in our dataset. We specifically designed our signatures to match evolutions of particular downloaders. For example, GoldInstall, a PPI program with diverse affiliation programs [25], was covered by a single signature. We then had to filter C&C flows that reached the C&C server, but the C&C server responded with non-C&C data (e.g., HTTP 404 responses). We enhanced our signatures with heuristics verifying that an end point shows active C&C communication, filtering out a significant amount of sinkholed communication.

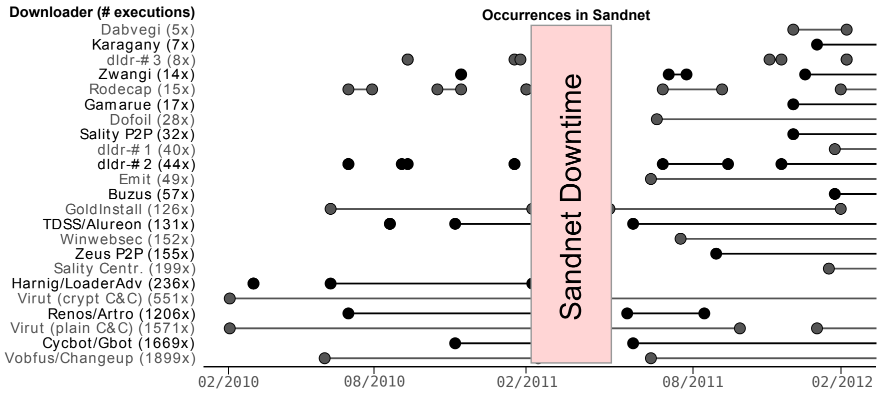


Figure 5.2: Lifetime of downloaders, as observed in Sandnet, from Feb 2010 until Feb 2012. The numbers in brackets represent the number of active executions of this downloader in Sandnet.

Figure 5.2 shows the resulting activity plot. To increase readability, we connected two markers if the gap between these two downloader occurrences in our dataset was less than four weeks. Due to a maintenance period in Sandnet, the graph lacks activity measures between 03/02/2011 and 08/04/2011. Overall, however, the graph shows that at least 11 of the 23 downloaders (48%) have been operational for more than a year. In addition, 18 downloaders (78%) were still active as of the analysis in this chapter (04/2012). Given that some downloaders are more present in our sample feeds than others, and given that our measurement period started in Feb 2010, the resulting data represents lower bounds of the actual downloader lifetimes. We even noticed that some downloader families were discussed by the community prior to our measurement period, indicating that the lifetimes of some downloaders is significantly longer than two years. We therefore speculate that in fact even more downloaders were successfully operated in the long term. This poses a long-lasting threat to our community, as apparently downloaders are largely and continuously used to infect PCs.

A few downloaders, such as Dofoil or Gamarue appeared first in our dataset in 2011, underlining active developments in the malware scene. The reasons why other downloaders ceased operation during the measurement period are twofold. First, in case of GoldInstall, C&C servers were not responsive for weeks, potentially indicating a downloader was abandoned or undergoes a major evolution. Second, as of August 2011, all specimen of Renos/Artro in our dataset were sinkholed by Shadowserver or Spamhaus.

5.4 Downloader Resilience

Seeing the significant lifetimes of downloaders, and knowing that defenders try to mitigate the threats of malware in general, we asked ourselves: How do attackers ensure such a high and long-term resilience of their downloader infrastructures? In this section, we will investigate the critical infrastructures used by attackers to operate their downloaders, that is, C&C servers and download servers, respectively.

5.4.1 C&C Infrastructure

C&C servers are vital to instrument the downloaders with new download instructions, and thus represent a sensitive part in the architecture of downloaders. From a downloader's perspective, two infrastructural services are crucial. First, most downloaders depend on DNS resolution prior to contacting their C&C server. Second, C&C servers obviously need to be reachable and service correctly. From a defender's perspective, both hosts (IP addresses) and domains represent vantage points to detect and/or disrupt downloaders.

Downloader	*IPs*		*ASes*		*Domains*		*TLDs*		*Timespan*
Family	#	*LL*	#	*LL*	#	*LL*	#	*LL*	M/Y - M/Y
Buzus	2	1	1	1	3	2	2	1	01/12 - 02/12
Cycbot/Gbot	145	48	56	36	2347	57	6	6	10/10 - 02/12
Dabvegi	5	4	4	3	5	4	3	3	11/11 - 01/12
dldr-#1	69	19	4	2	5	2	3	2	01/12 - 02/12
dldr-#2	41	11	21	5	45	12	7	4	06/10 - 02/12
dldr-#3	10	1	2	1	10	2	4	2	08/10 - 01/12
Dofoil	12	2	7	2	16	0	3	0	06/11 - 02/12
Emit	7	2	2	1	9	4	1	1	06/11 - 02/12
Gamarue	80	3	57	3	12	1	4	1	11/11 - 02/12
GoldInstall	12	5	7	3	13	8	3	2	05/10 - 01/12
Harnig/LoaderAdv	24	11	6	1	42	32	1	1	03/10 - 01/11
Karagany	2	0	1	0	7	0	2	0	12/11 - 02/12
Renos/Artro	27	5	12	3	75	0	3	0	06/10 - 02/12
Rodecap	8	4	2	2	5	4	3	3	06/10 - 02/12
Sality Centr.	239	62	125	47	243	59	31	18	06/11 - 02/12
Sality P2P	9849	1457	900	424	0	0	0	0	11/11 - 02/12
TDSS/Alureon	28	8	21	8	28	3	1	1	08/10 - 02/12
Virut (crypt C&C)	20	6	11	6	44	10	3	2	02/10 - 02/12
Virut (plain C&C)	14	4	9	4	3	3	1	1	02/10 - 02/12
Vobfus/Changeup	19	8	14	7	17	13	3	3	05/10 - 02/12
Winwebsec	5	2	4	2	0	0	0	0	10/10 - 02/12
Zeus P2P	2140	31	446	21	0	0	0	0	08/11 - 02/12
Zwangi	97	7	4	1	10	1	1	1	10/10 - 02/12

Table 5.2: Statistics on the C&C server distribution infrastructure per downloader family. LL=Long Lasting, i.e., IP addresses/domains had an uptime of more than 4 weeks. In each such case, we increase the corresponding AS/TLD counter by one also.

We use the data obtained in Section 5.3.3 for further analyzing the C&C infrastructure. In particular, we aggregate the number of domains and IP addresses used by a downloader as observed in SANDNET. While this does not necessarily give a complete view on the IP addresses and domains used by a downloader, the numbers can serve as lower bounds. Table 5.2 shows that the resilience strategies differ between the downloaders. In the second major column, we summarize statistics on the specific C&C server IP addresses of a downloader, plus its Autonomous System (AS). In the third major column, Table 5.2 lists the number of C&C domains per downloader. We highlighted domains or IP addresses that we have seen in active use for at least four consecutive weeks in Table 5.2 in the columns annotated with "LL" (long lasting).

Table 5.2 reveals that most downloaders use multiple C&C server hosts, and tend to distribute their servers across network boundaries. For example, Virut has been in operation during our entire analysis period with about 20 IP addresses in eleven ASes. We speculate that spreading server locations among multiple ASes is a strategic decision by the attackers. The more responsible parties and different national regulations are in place, the higher the complexity for defenders to take actions against specific downloaders. In that sense, Cycbot/Gbot stands out with 146 servers, hosted in more than 50 different networks. Observing such a large diversity may indicate that Cycbot/Gbot is in fact a malware toolkit with downloader functionality, which results in many smaller infrastructures independent from each other. We verified that the Cycbot/Gbot instances in our dataset used different IP addresses at approximately the same time. Another interesting case is dldr-#1, which appears to operate many C&C servers on its own. But instead it uses a large public file sharing company and this hoster's load balancing techniques, hiding eggs in seemingly benign Bitmap image files. As we are interested in all C&C activities of a downloader, we manually inspected all cases where possibly benign IP addresses or domains (e.g., image hosters) were involved and we explicitly did not exclude them from Table 5.2 if we also detected C&C.

Outstanding are the P2P variants of Zeus and Sality, with more than thousands of different C&C "server" hosts each. For these families, we consider P2P neighbors that respond to P2P-related UDP requests as active. The large number of ASes involved, 900 for Sality P2P and 446 for Zeus P2P, show that provider-driven initiatives against these P2P networks cannot disrupt the C&C infrastructures of these families. Interestingly, and particularly for Sality P2P, we saw a large fraction of P2P nodes to be lasting for more than four weeks. We first thought this may indicate that defenders joined this particular P2P network, but the high number of long-lasting ASes speaks against this. We will analyze P2P bots in Chapter 6 more extensively.

The analysis on the domains used by downloaders provides further interesting insights. Zwangi, for example, heavily rotates its C&C IP addresses typically within four /22 networks. Similarly, Gamarue deploys one particular domain pointing to highly fluctuating IP addresses in over 50 different ASes. In both cases the IP addresses are typically reused, that is, DNS is used to steer downloaders towards a rotating set of C&C servers. On the other hand, we observed downloaders for which the set of IP addresses was relatively constant, but the domains to resolve these IP addresses changed over time. For example, Virut used 45 domains to resolve to its 20 C&C servers, and Renos/Artro pointed its

136 domains to 38 IP addresses. Related to the previous observation that attackers settle their C&C servers in multiple networks, we also show that – for most downloaders – a diverse set of Top-Level Domains (TLDs) is chosen. Usually, these C&C domains are even registered across many continents, mostly including European, South-/North-American, and Asian registrars. Again, involving multiple domain registrars is presumably a strategic decision in order to complicate sinkholing operations.

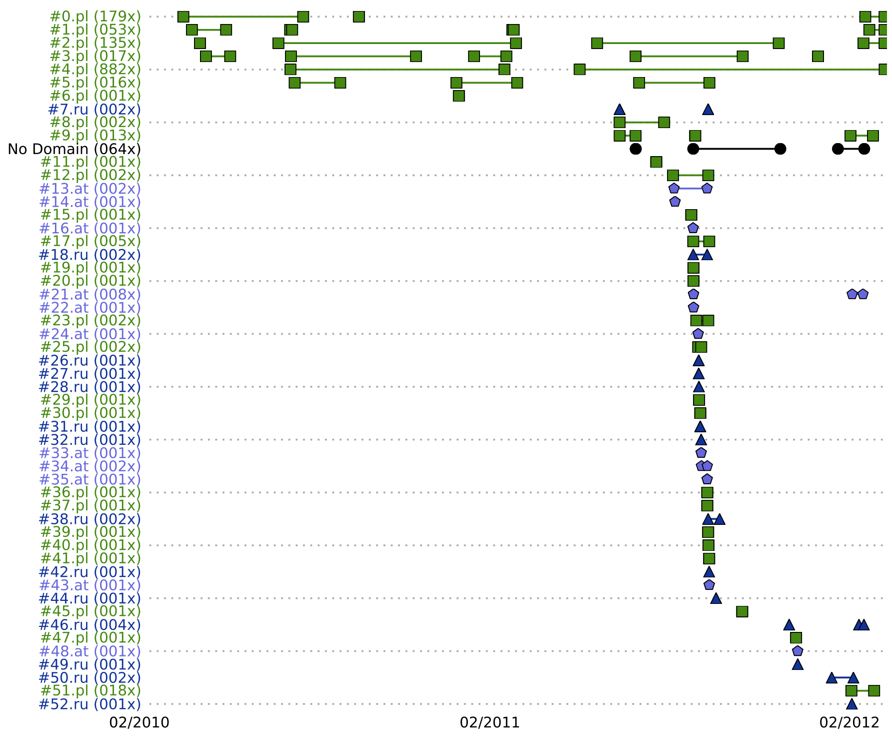
It can be seen that a large fraction of C&C servers (20%) remains operable for more than four weeks. Similarly, 217 domains pointing to active C&C servers (7%) remain in active use for at least four weeks. The observed long-levity enables defenders to take actions against downloaders, such as using domain or IP-address blacklists. On the other hand, the involvement of numerous registrars and providers shows how complex takedown efforts can be.

As a case study, we compared the usage time spans of Virut's C&C server domains (Figure 5.3(a)) with the usage time spans of the egg download server domains (Figure 5.3(b)). Both figures reveal that Virut seems to have a subset of stable domains that have been used throughout the last two years and that are still in active use, for both C&C and egg servers. In addition, several domains have been used only for certain periods. However, the sets of domains for C&C and egg distribution are distinct, that is, we have not witnessed domains being used for both, C&C and egg distribution. Interestingly, we observed a churn of Virut C&C server domain names between June 2011 and January 2012. Our initial hypothesis that these domains were used as backup C&C domains was proven wrong, as many other domains have been actively in use during that period. In addition, our passive DNS database in SANDNET revealed that not a single DNS resolution request for a Virut C&C domain resulted in NXDOMAIN or an empty answer section. Despite its technical simplicity, Virut thus exhibits a remarkable C&C- and egg-server resilience.

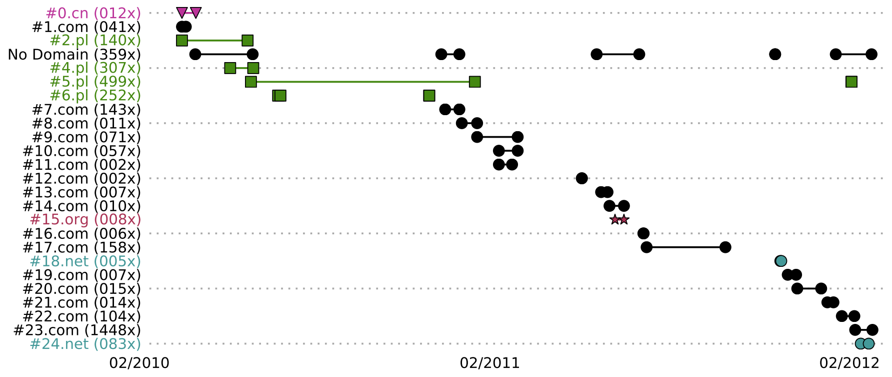
5.4.2 Download Server Infrastructure

The second pillar of a downloader's infrastructure is the resilience of download servers. We will now analyze the infrastructures of downloaders with plaintext download channels. We focus on plaintext downloaders, as we could map download channels to downloaders in these cases with high accuracy. Table 5.3 shows statistics on the egg distribution infrastructure for these downloader families. On purpose, we do not consider the C&C infrastructure here, except – unavoidably – in cases where the egg sample download is part of the C&C channel (see "inl" marker in Table 5.1). Two thirds of the observed plaintext downloader families exhibit more than ten distinct IP addresses for their download servers. A similar trend is observed concerning the domain names – only the Winwebsec family does not use DNS in the egg download process.

Per downloader family, the maximum uptime expresses the maximum time span where one single egg-server IP address has been witnessed as serving egg samples. Note that, in comparison to Table 5.2, the measurement in Table 5.3 is restricted to a family's egg servers and omits its C&C infrastructure. We consider an IP address or domain as long lasting if it serves eggs for at least four weeks.



(a) Virut's C&C server domains



(b) Virut's egg servers

Figure 5.3: Virut's C&C (above) and egg (bottom) server usage by domain over time. Colors/markers denote top-level domains.

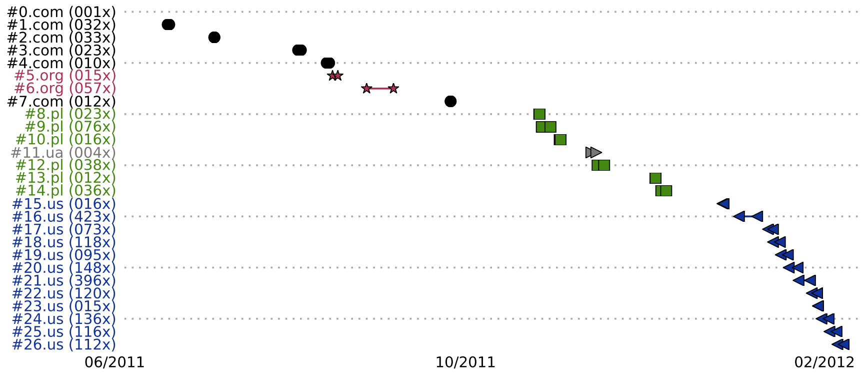
Downloader	*IPs*		*Domains*		*Eggs*		*Max.*	*Packers*
Family	*#*	*LL*	*#*	*LL*	*#*	*#MD5s*	*Uptime*	*Detected*
dldr-#2	26	7	44	10	1029	110	561 d.	b,t,u,f,h,y
dldr-#3	8	1	9	1	648	158	114 d.	u,s,d,n,p,c,e
Dofoil	14	1	29	0	103	93	96 d.	u,c,Y
Emit	6	2	27	0	5938	698	183 d.	u,p
GoldInstall	70	25	63	16	13155	971	592 d.	u,b,s,v,p,w,n,N
Harnig/LoaderAdv	31	12	46	23	1731	735	185 d.	u,o,f,p,d,N,b,n,M
Rodecap	2	2	8	2	286	23	445 d.	u,a
Virut (crypt C&C)	30	8	25	6	3852	293	459 d.	u,x,n,s,d
Vobfus/Changeup	15	7	34	3	2005	424	77 d.	u
Winwebsec	6	1	1	1	80	22	58 d.	n/a
Zwangi	86	2	8	1	263	138	49 d.	n/a

Table 5.3: Statistics on the egg sample distribution infrastructure per downloader family. LL=Long Lasting, i.e., uptime of more than 4 weeks. Packers: u=UPX, t=Themida, p=PECompact, e=PEtite, b=BobPack/Bobsoft, a=Armadillo, s=ASPack/ASProtect, x=EXECryptor, h=Thinstall, n=NsPack, f=FSG, d=D1S1G, v=Upack, c=CrypKey, o=ProActivate, y=XtremeProtector, w=WinUpack, N=NET MS, M=MoleBox, Y=y0dasCrypter

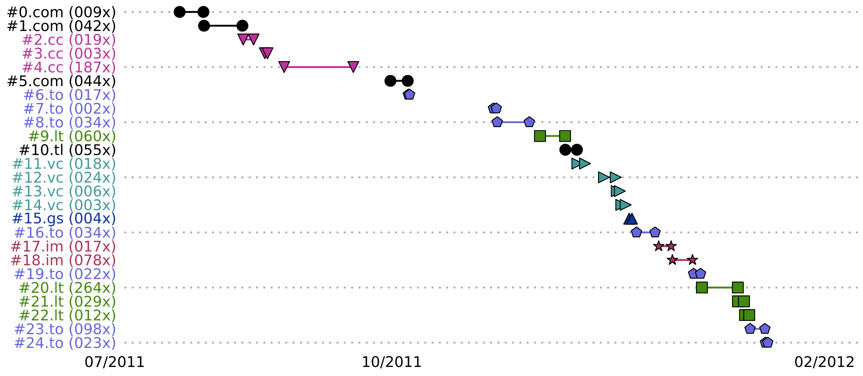
Table 5.3 shows that over the whole monitoring period, only a small fraction of the IP addresses is actually long lasting. In the cases that we manually inspected, we observed that downloaders typically move their download servers from time to time. For each downloader family of Table 5.3, we manually inspected the egg-server usage over time for both, domains and IP addresses. Interestingly, all downloader families exhibit similar egg-server usage patterns where the migration from one domain to another is clearly visible. The same applies to the IP addresses of egg servers, however, egg-server domains typically change more often than IP addresses. Some of the highly resilient download servers even have been serving eggs for more than a year.

For the downloader family Emit, Figure 5.4(a) shows each egg-server domain on the y-axis and the associated usage time spans. Note that the domain names have been pseudonymized. The egg-server domains show hardly any overlap in their usage time spans. In addition to the usage time span, the marker and the color denote the top-level domain. We observe that not only does the egg server move from one domain to another, as indicated by the changing pseudonyms. Download servers also migrate from one top-level domain to another, initially from .com to .org, then to .pl, and finally to .us. This pattern shows that – in order to strive for a takedown of this downloader's egg-serving infrastructure on the DNS level – many different registrars from varying time zones would be required to cooperate.

Figure 5.4(b) shows Vobfus/Changeup, which exhibits a strong domain migration pattern for its download servers. In this case, the domain names are typically used only for a couple of days, and never reused. Not as consistent as Emit, but still, Vobfus exhibits sequential top-level domain migration, too, although a few top-level domains are used in parallel.



(a) Emit: Gradually migrating to new domains and sometimes new TLDs



(b) Vobfus/Changeup: Migrating to new, previously unseen TLDs

Figure 5.4: Download server usage by domain over time for two downloaders. Domain names have been pseudonymized. Marker styles and colors distinguish the download server's top-level domain.

5.5 Egg Acquisition and Analysis

After investigating the downloader infrastructure, we will now analyze the downloaded eggs. Such an analysis allows us to draw conclusions on how attackers operate the egg infrastructure, for example, by using polymorphism and aggressively repacking served samples. We will begin with presenting two techniques how to acquire eggs from both plaintext and encrypted downloaders. The resulting dataset of actively acquired eggs will then serve to give first insights into evasive techniques used by downloaders.

5.5.1 Egg Acquisition Techniques

All downloader infrastructures have one necessity in common: these services must be publicly accessible, as (with the exception of targeted attacks) fraudsters aim for large-scale deployment of their malware. Consequently, attackers cannot easily deploy client authentication mechanisms that prevent their infrastructures from being "abused," raising the difficulty for attackers to control who is accessing the infrastructures. We exploit these necessities to obtain eggs for the downloaders under our analysis. We present two techniques that enable us to acquire the downloaded eggs for downloaders with plaintext and encrypted communication, respectively. Previous efforts analyzing a few specific downloaders [25] did not require automated egg acquisition techniques. However, given our significantly larger sample set, we seek for a more scalable solution to analyze downloaders. Our techniques may be a potential enabler for future research on malware acquisition methods, as they require only little manual effort compared with reverse engineering.

Plaintext Downloaders

For plaintext downloaders, we exploit the fact that eggs are downloaded without disguising or encrypting the communication. Methodically, we replay the egg-download dialog towards each download server and require new egg samples this way. For example, in case of HTTP, once the download server and the egg's URI is known to defenders, downloads can be repeated regularly. We implemented a *dialog repeater* that takes pairs of HTTP request and communication end point as input, that is, payload bytes with a destination IP address and port. For each such pair, the repeater replays the dialog towards the specified destination once an hour, typically resulting in HTTP responses. We feed the repeater with input pairs by searching for requests by downloaders in our dataset that led to egg downloads. Given the prevalence of HTTP in our dataset, we did not incorporate further network protocols to the dialog repeater.

In order to avoid such mechanisms, fraudsters could potentially use blacklists of IP addresses of known malware analysis systems [1]. For an attacker, it is straightforward to block all requests from systems as ours. Consequently, instead of using a single Internet outbreak and IP address, we established a proxy network to route the traffic through our home DSL lines. In contrast to well-known proxies such as Tor or open proxies, end-user IP addresses seem to stem from realistic end users and – in our case – even change daily. We made sure that our ISPs did to interfere with our measurements by comparing outputs of multiple proxy hosts. Despite its simplicity, as we will show, the repeater is a well-working mechanism to acquire new eggs.

Encrypted Downloads

A drawback of the dialog repeater is that it cannot download eggs via encrypted download channels. Even if the download succeeds, we could not make use of the encrypted egg. Therefore, as a complementary technique, we leverage the actual downloader to acquire eggs. The intuition behind this method is simple: whenever a downloader is executed, it will download and execute previously unknown malware samples. We instrumented our SANDNET VMs with a kernel-based Windows system driver that records the file images whenever new processes are forked or system drivers are loaded. For each potential egg being executed, the kernel driver computes the MD5 checksum and records the new processes' image.

However, monitoring new processes results in a large number of legitimate system files to be interpreted as potential egg. To filter legitimate system files, we built a whitelist of trusted system files by scanning all files of a clean SANDNET VM. In addition, as a further filter to catch only actually dropped and not modified system files, we manually assembled patterns for the file paths where each downloader is storing its eggs. We specifically discard eggs that we identify as exact or repacked/modified copies of the downloader program itself. To do so, we correlate the time when data was received from the network with the time when the new process was forked. After adding the kernel driver to SANDNET, we additionally scheduled the execution of downloader families with encrypted download channels on a daily basis for seven weeks starting in January 2012.

5.5.2 Egg Sample Distribution

In addition to our passive SANDNET database, we use both active techniques described to obtain a comprehensive egg dataset. Thus, for the plaintext downloaders, we identified the download channels and describe the downloader infrastructures and their uptime. Table 5.3 (page 82) shows the egg distribution per downloader family that exhibit plaintext egg downloads. The number of successful egg downloads as well as the number of MD5-unique egg samples differs widely among the plaintext downloader families. Whereas for GoldInstall more than 13,000 egg downloads completed successfully, the number of unique egg samples is much smaller. Other families such as Dofoil show that a significant fraction of the successful egg downloads expose differing MD5s. This indicates that not all downloaders aggressively repack the served eggs.

Table 5.4 summarizes our experiments of actively milking encrypted downloaders in SANDNET. For each downloader, we name the number of executions in SANDNET and show the number of eggs and unique eggs, respectively. For nine of ten downloaders, our technique was able to trace eggs. Despite its short runtime and the relatively small number of execution per downloader, we were able to acquire a high diversity of eggs. For example, although Zeus P2P is well known as a banking trojan, we can confirm Symantec's recent observation [58]

Family	*Execs*	*Eggs*	*MD5s*	*Packers*
Buzus	316	1898	329	b,u,p
Cycbot/Gbot	181	1030	374	u,c
Dabvegi	278	271	8	unknown
dldr-#1	14	10	3	t
Karagany	256	242	178	z
Renos/Artro	320	2454	23	u
Sality	261	241	59	u
Sality P2P	250	0	0	n/a
TDSS/Alureon	226	652	79	n/a
Zeus P2P	224	221	101	n/a

Table 5.4: Downloaded egg samples from the encrypted downloaders from Dec '11 to Feb '12. Packers: u=UPX, t=Themida, p=PECompact, b=BobPack/Bobsoft, c=CrypKey, z=StealthPE.

that it also downloads non-Zeus samples. For Sality P2P, we have observed active downloads, but our kernel-level monitor did not observe these eggs to be executed. Renos/Artro drops malware, although our Renos samples were effectively sinkholed since August 2011. The low number of executions of dldr-#1 is due to scheduling this downloader only recently. As a particularly interesting case, TDSS/Alureon dropped all recorded executables by extracting the original sample. In addition to the loaded eggs, however, we recorded that in about half of the executions a kernel driver was loaded, showing that our kernel-based analysis techniques may also work with downloaders that carry rootkits capability themselves.

5.5.3 Polymorphism

Malware is well known for polymorphism in order to evade antivirus signatures. An interesting question in the context of downloaders is if and how polymorphic code is used. We approached this aspect in two ways. First, we classified all egg samples using yara [6] and packer-identification rules in order to assign which packer was used to (re)pack an egg sample. In addition, we submitted the egg samples to our sample sharing partners and Virustotal. In turn, querying Virustotal, we assigned A/V labels to the egg samples in order to see how over 40 A/V vendors name the eggs.

Sample Packing

A large fraction of the egg samples were successfully classified using yara packer rules. Tables 5.3 and 5.4 show the number of distinct packers for the eggs of each downloader family. The dominating packers are based on UPX. However, many different packers can be found, such as Armadillo, Themida, ASPack, ASProtect, NsPack and PECompact. In addition, some eggs, such as those of Winwebsec, were packed with unknown packers. The fact that the egg packers vary throughout one downloader family, supports the assumption that there are multiple “clients” per downloader and that it is likely not the download server that repacks the eggs. Instead, we assume that the clients make packed eggs available to the downloaders. In this context, we consider a client to be an attacker willing to distribute malware via downloaders.

Repacking

Eggs are repacked to successfully evade signature-based A/V. For those families that have plaintext egg downloads, based on SANDNET and dialog repeater traces, we estimate lower bounds on which downloader families distribute polymorphic eggs. In this context, we define an egg to be repacked if different content – in terms of MD5 hash – is served for what can be considered the same egg sample – based on (approximate) file size and A/V label. Thus, for each downloader family, we consider an egg to be repacked if we observe egg downloads with at least eight distinct MD5 egg hashes all having (nearly) the same file size and the same A/V label, within a time span of one month. On average, our filter criteria translate to a repacked egg sample at least once every four days. Furthermore, to ensure statistical significance, we limit our dataset for this experiment to families with at least 90 distinct eggs. Of those nine families, we observed eight to exhibit repacked samples. Note that we do not consider repacking to be a property of the downloader family. Instead, we assume that the clients of these downloaders take care of the repacking of their eggs. Wheres at least one client of Emit reached a maximum repacking rate of once every 17 minutes, dldr-#3 repacked only up to once every 2.5 days. For GoldInstall, we measured repacking once a day, and one of the Dofoil clients repacked its eggs once every hour.

This confirms similar analyses by Cabellero et al. [25], only that two downloaders in our dataset (Emit, Dofoil) deploy overly aggressive repacking. Employing our dialog repeater, we looked for server-side polymorphism where the egg sample is repacked upon *each* request. In particular, we tried to measure whether the repacking of Emit eggs takes place via on-the-fly server-side polymorphism, but unfortunately the egg servers were not been reachable anymore during this experiment.

5.6 Discussion

Our analysis provides detailed, novel and important insights into the resilience of malware downloaders. Revealing the possibility to monitor downloaders may motivate attackers to switch to more advanced techniques. However, given the large numbers of long-term operating downloaders, we see the need to raise attention to this problem domain. Our work also aims to highlight relevant downloader families, such as P2P- or rootkit-driven downloaders, fostering future research on potentially previously unknown malware families.

Obviously, our techniques to automatically milk downloaders are evadable by attackers. While it is straightforward to evade our dialog repeater, evading our kernel-based driver requires more thoughts, though. For example, we face the risk that our current setup may fail for kernel-level rootkits such as TDSS. Similarly, we had to exclude one particular downloader (Wintrim) from our analysis, as it detects virtual machines before unpacking itself. However, hardened dynamic analysis as with Ether [32], hardware-based hosts [49], or developing resilient kernel drivers would be effective against attackers' moves.

During dynamic analysis, and particularly when allowing network access to malware, we potentially risk to harm others. However, in a best effort to drop all harmful traffic, we strictly control and monitor SANDNET's activity. As a consequence, we have not observed a single abuse complaint concerning SANDNET so far. Furthermore, a particular risk of executing downloaders is to – unintentionally – financially support PPI downloaders, in that attackers are paid for installations. However, the cash flow in this case is that attacker A (whose downloader is executed in SANDNET) is paid by attacker B (who asked for his malware being dropped), not causing harm to any innocent uninvolved individual. In fact, analyzing and abusing these PPI cash flows is an interesting future work topic itself.

5.7 Conclusion

We identified and characterized 23 downloader families, showing that the downloader landscape is diverse in terms of architectural design, communication protocols and encryption schemes being used. We observed that many downloaders – albeit sometimes simple – have been actively operated for more than a year. Motivated by this observation, we analyze how attackers ensure the resilient operation of their downloader infrastructure. For example, we show that downloaders migrate their C&C servers aggressively among different Autonomous Systems, often involving multiple countries. Similarly, we observed downloaders not only to alter the C&C domains frequently, but also to involve diverse domain registrars. As a byproduct of our analysis, we revealed further details on the workings of downloaders, such as server-side polymorphism. These observations show that mitigating the problem of downloaders is more difficult than it might seem. To foster future research in this area, and as an automated mechanism to acquire previously unseen malware samples, we present two generic techniques which extract downloaded eggs from any downloader.

6 Resilience Analysis of Peer-to-Peer Botnets

The previous chapter has shown the resilience of malware installation infrastructures, which are typically used to create or to enlarge botnets. This chapter will focus on the resilience of botnets themselves[1]. In particular, we discuss botnets that are explicitly designed to be highly resilient: peer-to-peer (P2P) botnets. The resilience of centralized botnets depends on the availability of a few C&C domains or C&C servers. P2P botnets, on the other hand, do not rely on central servers that are present in centralized or semi-distributed botnet architectures. In this chapter, we provide an overview of ten P2P botnet variants, of which six are still being operated. We assess the resilience of these P2P botnets by prototyping mitigation strategies such as sinkholing. The systematic resilience analysis aims to assist security researchers in evaluating mitigation strategies against future P2P botnets.

6.1 Introduction

Botnets have since many years been used by fraudsters with financial motivations. In the past, defenders could often disrupt these mostly centralized botnets by terminating command and control (C&C) servers or C&C domains. As a response, attackers have developed new and supposedly more resilient botnet topologies, such as highly distributed networks lacking central components. These topologies remove the single point of failure, and botnet takedowns require new strategies. One type of distributed botnets are peer-to-peer (P2P) botnets, where each bot stays in contact with other bots to exchange commands, lacking central servers.

This redundancy advantage over centralized botnets may suggest clear trends towards P2P botnets. In a few cases, such as with Storm/Peacomm and Waledac, defenders could infiltrate the P2P botnets and disrupt their operation. Yet, largely unnoticed in academia, a number of new P2P botnets such as Zeus P2P,

[1]Some analyses in this chapter base on reverse engineering efforts by Dennis A. Andriesse and Tillmann Werner, who we thank for their contributions.

Sality, ZeroAccess, Hlux or Miner, have emerged. These botnets have been operational for as long as five years, deploying a variety of custom P2P protocols. In most cases, the population sizes of these botnets are unknown, nor does any evidence about the resilience of these botnet designs exist. To fill these gaps, we characterize four historic and six existing P2P botnet protocols. With this characterization, we provide an overview of P2P botnets and compare their design evolution. We classify the known P2P botnets into *structured* and *unstructured* networks, and describe advantages of both design choices.

We then analyze the *resilience* of these P2P botnets. To guide future research in the area of P2P botnet mitigations, we discuss four different aspects of P2P botnet resilience: (1) We evaluate the *reconnaissance resilience*, that is, to which degree botnets can deter defenders from enumerating bots in their networks. (2) We characterize *C&C resilience*, describing how the C&C layer of the P2P botnet is secured against abuse by others. (3) We show the *sinkholing resilience*, by prototyping and successfully testing attacks to mitigate existing P2P botnets. (4) We discuss if defenders can split the botnet into small unusable subnetworks by evaluating the *partitioning resilience* of P2P botnets.

Our practical resilience assessments give detailed insights into the proprietary C&C protocols of P2P botnets. With this C&C knowledge, we measure the population of seven P2P botnets, and identify botnet families spanning more than a million of infections. Motivated by this, we test and elaborate how defenders can sinkhole existing P2P botnets. In fact, we develop attack prototypes that helped to prepare sinkholing operations against two bot families and in total eight disjoint botnets. Our systematic resilience analyses aim to foster future efforts to mitigate the threats of P2P botnets.

However, we also observe trends towards highly resilient P2P botnets that have significant improvements over historic P2P botnet protocols. For example, we analyze the peer reputation scheme of Sality, and discuss the effects of self-healing P2P protocols as exhibited by ZeroAccess. Seeing these innovations, it is unclear if sinkholing can be successful in the future. To inspire future research, we will discuss alternative P2P botnet mitigation strategies.

The remainder of this chapter is structured as follows: Section 6.2 gives background information about P2P botnets and Section 6.3 describes existing P2P botnets. In Section 6.4, we measure the botnet populations to show their magnitudes. Section 6.5 provides a structured resilience analysis of historic and current P2P botnets. Section 6.6 discusses implications of our results. We outline related work in Section 6.7 and conclude our work in Section 6.8.

6.2 Preliminaries

Traditionally, botnets were purely centralized and had a single, non-redundant C&C server. Once defenders shut down this C&C server, the operation of the botnet was ceased immediately, as the bots could not be commanded anymore. To achieve better robustness, botmasters followed two distinct strategies. First, botmasters designed what we term *semi-distributed botnets*. Such botnets remain centralized, but improve the robustness of the C&C infrastructure with redundancy. For example, botmasters started to announce redundant C&C servers by fluctuating DNS records (fast-flux) [69], or introduced domain-name generation algorithm (DGA) to have changing C&C server end points over time [9]. Although these techniques require small modification to the C&C channels, the typical client-server patterns are similar to centralized botnets.

6.2.1 P2P Botnet Architecture

As an alternative botnet architecture, malware authors started to deploy *peer-to-peer (P2P) botnets.* A P2P botnet does not (entirely) rely on centralized servers for its primary C&C channel, but has ways to spread C&C commands via client-to-client communication. As there is no single point of failure in P2P networks, defenders cannot disrupt the botnet by terminating a few hosts only. In P2P botnets, a bot stores end points of a subset of all other bots (*peers*) in a *peerlist.* A bot then contacts end points in this peerlist, for example, to pull C&C commands or to update its peerlist. This typically results in a directed graph.

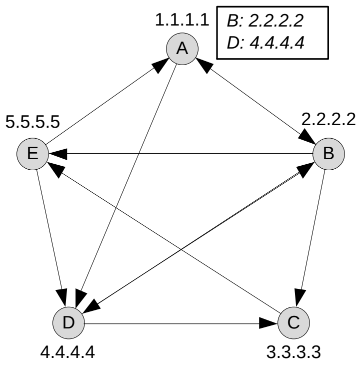


Figure 6.1: P2P Botnet Graph Example

Figure 6.1 shows an example P2P botnet consisting of five peers with IDs A – E. The peerlist of peer A lists the *peer IDs* (unique identifier of a peer) and end point addresses (IPv4 addresses and – optionally – ports) of peers B and D. In this example, peer D does not store A in its peerlist, rendering the network in a directed graph.

Graph theory helps to analyze the resilience of the P2P networks. For example, peer A has in-degree two, as peers B and E know about it. Similarly, peer A has an out-degree of two, as it has peers B and D in its peerlist. Whereas the out-degree shows how well-connected a bot is to other bots, the in-degree measures the popularity of a bot. For example, peers with an out-degree of zero could not pull commands anymore. Similarly, peers with an in-degree of zero will never receive messages from others.

6.2.2 P2P Botnet Taxonomy

Although the basic design principle of peerlists is inherent to all P2P botnets, a clear distinction can be made between *structured* and *unstructured* P2P botnets. Both designs have unique strategies for exchanging C&C commands, and thus require different angles for resilience analysis.

Structured P2P Botnets

Structured P2P botnets are similar to existing P2P networks, like many popular file-sharing networks. Structured botnets rely on a Distributed Hash Table (DHT) to route requests through the network. Structured botnets are typically *pull-based*, that is, the botmaster stores commands in the DHT and the bots regularly pull information from peers. Bots can also store, for example, stolen data in the DHT, removing the need for centralized dropzones. The details differ per DHT implementation, but the basic concept is as follows.

A bot chooses a unique identifier in the DHT space and publishes itself on the DHT network. To store data in the DHT, a unique identifier ID_{data} in the DHT space is assigned to each piece of data. Then n nodes with IDs close to ID_{data} are responsible for hosting this data. To request this data, a peer recursively asks neighbors that are close to ID_{data}, eventually converging to peers hosting the data. A P2P botnet can leverage this concept for exchanging C&C commands via the DHT. For example, the botmaster can include an algorithm to compute data IDs dependent on the current time in the bot binaries. Whenever a bot searches for an ID, the botmaster can publish a command in the DHT with the matching ID. The bot can then retrieve the command by searching for IDs derived from the ID generation algorithm.

Unstructured P2P Botnets

Unstructured P2P botnets, on the other hand, do not store information in a DHT. Instead, unstructured networks rely on sending or receiving commands via gossiping. In *push-based* unstructured P2P botnets, bots forward a command that they received from a neighbor node to other peers in their peerlist. To prevent message loops in such broadcasts, time-to-live (TTL) values can be included in the gossip messages. Only peers with a positive in-degree will receive such commands. In *pull-based* unstructured P2P botnets, bots regularly ask peers for new commands, indicating the need for high out-degrees to retrieve new commands. As commands need to be requested to propagate through the network, commands on pull-based network typically spread slower than on push-based networks.

6.3 Overview of P2P Botnets

This section provides an overview of P2P botnet families that arose between begin-2007 and mid-2012. We carefully assemble this set of P2P bots by both reading literature and searching for P2P-related behavior in SANDNET (see description in Chapter 4). We used traffic analysis to identify P2P bots in SANDNET traffic. For example, we search for malware with a large set of contacted hosts, or inspect malware samples that have many outgoing connection without prior DNS resolution. This way, we identified all publicly known P2P botnets that arose since 2010. From our dataset of P2P bots, we explicitly exclude bots that use P2P communication as their backup C&C channel, such as TDL-4 [36] and Conficker.C [34, 90]. As we will not use dynamic malware analysis to analyze P2P bots in the following sections, we will not discuss the relevance of the guidelines presented in Chapter 6 here.

6.3.1 P2P Botnet Characteristics

In this work, we closely analyze ten P2P botnet variants, of which six were still active as of September 2012. Figure 6.2 shows the life span of the botnets under analysis. The start of a new botnet is derived from the date a botnet was publicly discussed for the first time. A botnet's lifetime ends when the botnet was successfully sinkholed or when the botmasters terminated the botnet. Figure 6.2 shows that the minimum P2P botnet lifetime is eight months. Some P2P bots, such as Sality or ZeroAccess, have been operational for up to five years. We are not aware of sinkholing attempts against the four active P2P botnet families: Zeus, ZeroAccess, Sality, and the most recent Hlux version.

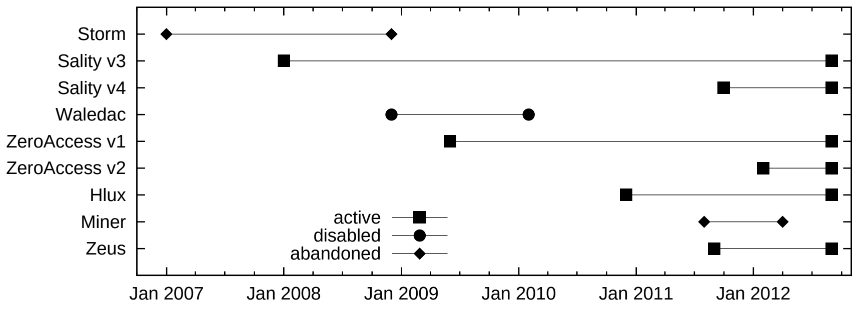


Figure 6.2: History and recent developments of new P2P botnets

Figure 6.2 also shows the evolution of a particular botnet crew. The code similarity in Storm, Waledac and Hlux strongly indicates that all bots were developed by the same botnet crew [24]. Waledac was introduced by the Storm authors about two years after the appearance of Storm to address resilience vulnerabilities [84, 89]. About two years after its introduction, the Waledac botnet was disrupted and succeeded by Hlux. Despite successful sinkholing efforts against the first two Hlux variants, a third Hlux variant is still operational [92].

Table 6.1 helps to understand the nature of these P2P botnets. For example, ZeroAccess and Sality are malware downloaders and thus drop "only" other kinds of malware. We speculate that this explains why ZeroAccess and Sality largely stay under the radar, as they do not cause direct harm to infected clients. On the other hand, Hlux and particularly Zeus directly harm their victims by, for example, stealing and abusing banking accounts or spamming. Table 6.1 also details the botnets' P2P command routing layer. Storm used Overnet, a Kademlia-based network for its communication and was the only P2P botnet that ever purely relied on existing P2P networks. All other P2P botnets under analysis use proprietary unstructured P2P protocols.

Botnet	**From**	**To**	**Proto**	**Routing**	**Monetization**
Hlux	12/10	-	Propr.	Router peers	Spam, DDoS
Miner	08/11	03/12	Propr.	Gossiping	Bitcoins, spam
Sality v3	01/09	-	Propr.	Gossiping	PPI
Sality v4	11/10	-	Propr.	Gossiping	PPI
Storm	01/07	12/08	Overnet	DHT searches	Spam
Waledac	12/08	02/10	Propr.	Router peers	Spam
ZeroAccess v1	06/09	-	Propr.	Gossiping	PPI
ZeroAccess v2	02/12	-	Propr.	Gossiping	PPI
Zeus	09/11	-	Propr.	Gossiping	Banking trojan

Table 6.1: Overview of P2P botnets, their protocol and monetization

6.3.2 Botnet Descriptions

We will now describe the most important details of the individual P2P botnets. For more extensive analyses of individual botnets, we will refer the interested reader to technical reports.

Storm

Storm (a.k.a. Peacomm) was a structured P2P botnet using the Overnet protocol, a Kademlia implementation. In fact, the first version of Storm used an existing Overnet network, and the bots added themselves to the existing DHT. In Storm, botmasters stored spam templates at deterministically computable IDs in the DHT. In turn, the bots requested these commands by looking up the computable IDs. Storm was significantly disrupted in 2008, when Holz et al. continuously replaced Storm's commands with bogus commands [42].

Sality P2P

Sality P2P appeared with its second version in 2008 and is a variant of the centralized Sality malware downloader. Sality uses an unstructured P2P network in a pull-based manner. Peers regularly check if their neighbors promote previously unseen files, and if so, they download and install these files [33]. We distinguish between two separate Sality networks, consisting of peers only with version three or version four, respectively. Both networks share the same P2P protocol and differ mainly in the file downloading mechanism.

Waledac

Waledac is assumed to be the successor of Storm [84]. Waledac had centralized peers in its upper layer, which served spam templates to peers in the lower hierarchy. These lower peers form the majority in the network and were connected via an unstructured P2P network. The lower peers exchanged lists of peers in the upper layer using pull-based communication. In February 2010, Waledac was sinkholed via manipulated peerlists by Stock et al. [84].

ZeroAccess

ZeroAccess (a.k.a. Sirefef) is a malware downloader with an unstructured P2P C&C architecture. It exists in at least two variants and is organized in at least seven disjoint networks. The older variant, ZeroAccess v1, is pull-based, and bots regularly consult their neighbors for malware installation instructions. The newer variant, ZeroAccess v2, uses both push- and pull-based communication. New peers are broadcast to the network (push), whereas the malware download commands are requested from other peers (pull) [63, 96].

Hlux

Hlux (a.k.a. Kelihos) is an unstructured P2P botnet mainly used for spamming and ID theft [24], but its entire malicious functionality was never fully analyzed. Hlux is assumed to be the successor of Waledac. Kaspersky successfully mitigated the first two variants of Hlux by manipulating the peerlists. The two variants were sinkholed in September 2011 and March 2012 respectively, each time abusing the pull-based P2P protocol. A succeeding third Hlux variant is still operated [92].

Miner

Miner was an unstructured P2P botnet that included three libraries for mining Bitcoins (a digital currency). The Miner botnet consisted of two almost disjoint networks with about 38,000 non-NATed peers according to Kaspersky [93]. According to Crowdstrike, Miner ceased to operate around 03/2012, presumably due to insufficient monetization of mining bitcoins.

Zeus P2P

The C&C servers of most centralized Zeus botnets were increasingly being tracked and taken down by defenders [7]. As a consequence, attackers adapted the leaked Zeus source code to launch a P2P-based Zeus (a.k.a. Gameover bot) in August 2011. Zeus P2P is an unstructured P2P network having a pull- and push-based command architecture. Zeus's configuration files are pulled from peers with more recent versions, containing, for example, browser hooks used to steal personal data. Dropzone locations for receiving the stolen data are pushed via gossiping.

6.4 P2P Botnet Populations

From a defender's perspective, the ability to enumerate individual P2P bots is a significant advantage over centralized botnets. This P2P exploration procedure, also known as *crawling*, is exceedingly helpful for many reasons. First, crawling measures the number of infected systems in a P2P botnet. Size estimations help defenders to shift focus to more prevalent botnets, or observe significant changes in the botnet dimensions over time. Second, several organizations are interested in lists of infected clients. For example, ISPs or CERTs can alert infected customers. Similarly, in case of banking trojans such as Zeus, banks or credit-card companies can verify their transactions by checking the client's IP address. Lastly, crawling is typically required for P2P botnet mitigation techniques.

6.4.1 Crawling Peculiarities

As most P2P botnets deploy proprietary P2P protocols (Section 6.3), crawling P2P networks requires a deep understanding of the protocol peculiarities. We therefore manually reverse engineered bot binaries in depth, thankfully assisted by early-on reversing results [23, 24, 33, 58, 63, 89, 96]. We summarize our reverse engineering insights that are relevant in the context of crawling in Table 6.2.

Botnet	**Bot IDs**	**NATed**	**Peers shared**
Hlux	pseudo-rnd 16b	No	newest 250
Miner	none	No	all
Sality v3	not shared	No	1 random active
Sality v4	not shared	No	1 random active
Storm	16 bytes	Yes	10 close to req. ID
Waledac	20 bytes	No	newest 100
ZeroAccess v1	not shared	Yes	all 256
ZeroAccess v2	not shared	Yes	newest 16
Zeus	20 bytes	Yes	10 close to req. ID

Table 6.2: P2P botnet characteristics relevant for crawling. Legend: NATed = peers behind NAT are maintained in peerlists.

Ideally, botnet population measurements count unique bot identifiers. However, Table 6.2 shows that only Waledac, Zeus and Hlux use truly unique identifiers. ZeroAccess does not keep peer IDs in its peerlists, and a bot communicates its own peer ID only in peerlist requests. Similarly, Sality deploys deterministic ID calculations that cause many ID collisions. In these cases, we have to rely on counting unique IP addresses for estimating network sizes. Second, the peer selection algorithm for peerlist responses influences the way we can crawl the network. For example, in Zeus, the peers whose IDs are closest to an arbitrary search key are returned, asking for either random exploration or binary searches. In ZeroAccess, the most recent peers are returned, meaning that it is better to deploy breadth-first rather than depth-first crawls. Third, a nontrivial issue for estimating botnet sizes is that seven botnets do not maintain peers that are behind a firewall or NAT gateway in their peerlist. However, these NATed hosts participate in the P2P botnet, that is, they also receive commands. Similarly, even if NATed peers are shared, crawlers cannot reliably determine if a peer is active, or if the shared peerlist entry is outdated. The next section describes a crawling strategy tackling these peculiarities.

6.4.2 Crawler Strategy

Algorithm 1 sketches our crawler design. Simple yet effective, it allows to count all public peers in the network and keeps track of peers that actively responded to our requests. The algorithm is initialized with a set of bootstrapping peers, for example, extracted during dynamic analysis of a P2P bot. We then continuously send requests to random peers to learn about other peers. In the unstructured P2P botnets under analysis, we directly ask known bots for (parts of) their peerlists, iteratively expanding knowledge of the entire network. In structured networks, we would send random route requests, learning about nodes that are close to the search hash. We mark peers that have ever responded to our requests as *active*. This way, we can later distinguish between potential and verified bots.

```
$peers \leftarrow$ bootstrapping peers
while $true$ do
    if $packet\ arrived\ at\ socket$ then
        $peerlist_{new} \leftarrow$ read from socket
        find sender $s$ in $peers$ and mark $s$ as active
        $peers \leftarrow peers \cup peerlist_{new}$
    else
        send request to random peer $p$ from $peers$
    end
end
```

Algorithm 1: P2P Botnet Crawling Algorithm.

However desirable, performing *accurate* crawls of P2P botnets is not straightforward. Already the early-on experiences by crawling Storm in 2007 have shown that the crawling results need to be carefully interpreted [48]. The following list enumerates potential biases and counter-measures that we take to achieve correct crawling results:

A) **Infected dial-up users may frequently get new IP addresses, a common practice observed, for example, at European ISPs.** To avoid double-counting nodes, we observe population developments over time. Intuitively, the bias introduced by churn is negligible in the early crawling phase. Similarly, if botnets use unique identifiers, we provide the number of crawled IDs.

B) **It is unknown when crawling converges towards the full list of peers.** P2P botnets undergo steady change, as peers continuously enter or leave the network. For example, infected PCs are switched on or turned off during the crawl, or new infections join the botnet. To give accurate numbers, we monitor if the change in botnet population stagnates to a steady level, which may indicate that crawling has converged.

C) **Other researchers or defenders may have injected fake peers to the P2P network.** While it is practically hard to identify fake nodes in the network, we do our best to search for anomalies in the crawling results. For example, we noticed that the Polish CERT was well-connected in Zeus, and removed their fake peers from our counts after contacting them. Similarly, we count only peers that follow the P2P protocol and respond to our peerlist requests with valid replies.

D) **Peers behind a firewall or NAT gateway cannot be contacted.** Although NATed peers do not respond to our requests, they may in fact be active. To reliably count active peers, we count only peers that respond to our peerlist requests. We ignore peers that are potentially online, but are protected by a firewall or NAT gateway. Instead, we will extrapolate the total botnet population by assuming ratios of NATed hosts.

6.4.3 Crawling Results

We implemented crawlers for eight disjoint P2P botnets that were active as of September 2012. We seeded the initial crawler bootstrapping peers with data from Sandnet. Per bot variant, we extracted IP addresses (and ports, if applicable) of up to 100 recently active bots via dynamic analysis in Sandnet. We launched the crawlers from a university network on a typical weekday (06/09/2012) and let all crawlers run for at least 24 hours. As the crawled P2P botnets use UDP for peerlist exchanges, our crawler risks packet losses if it requests for peerlists too aggressively. For that reason, we rate-limited the request rate such that we did not face packet loss when monitoring the 150 MBit/s uplink during the crawls. Figure 6.3 shows our crawling results for the four botnet variants. The numbers represent unique IP addresses of verified peers, that is, peers that responded to our peerlist requests. Although this already excludes most NATed peers from our measurements, we also explicitly removed active peers that did not initiate communication from typical (bot type dependent) port ranges. Consequently, our crawls set lower bounds for the botnet population.

Figure 6.3 depicts the population convergence over time. As expected, the number of unique IP addresses increases even after 24 hours of crawling. In fact, we included two lines for Zeus in Figure 6.3, one counting the number of IP addresses (upper line), and another counting the number of unique node IDs (lower line). While at the beginning of the crawl the discrepancy is negligible, IP address churn significantly biases the results after 24 hours. This also confirms observations that Stone-Gross et al. [85] made in a similar experiment, in which they found that a day-long IP address count approximated the total botnet size for the Torpig botnet. In Figure 6.4, we show which ratio of unique peers is found in each hour, that is, we draw the derivative on the number of peers. Obviously,

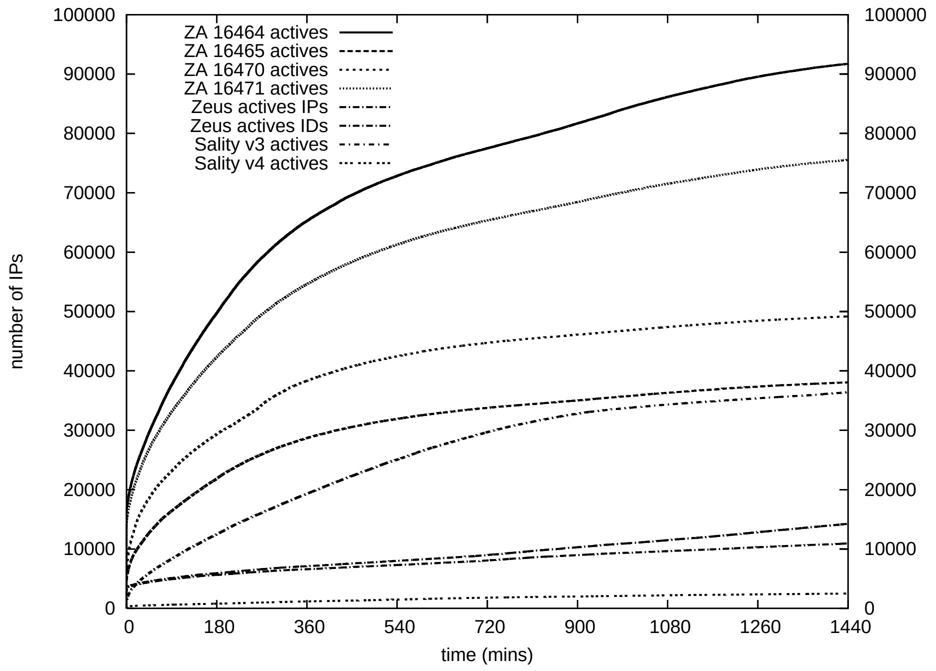


Figure 6.3: P2P Botnets Crawling Results

the largest increase of new peers happens in the first few hours. The number of new peers found then stagnates after about 12 hours. We speculate that the stagnated level of increase (about 2%–4%) is caused by both, IP address churn and new infections joining the botnets.

We summarize our crawling results in Table 6.3. The first four columns show the number of active peers (i.e., unique IP addresses) that we crawled after one hour, 12 hours and 24 hours, respectively. To estimate the overall botnet populations, we also account for the number of NATed bots, which our crawlers did not find (or explicitly filtered them out). Stone-Gross et al. have shown that – in the case of the Torpig botnet – around 80% of the bots were NATed [85]. We thus conservatively estimate the minimum population pop_{min} by multiplying the number of peers found after 12 hours by five (assuming that 80% of all bots are NATed). Stone-Gross et al. have also shown that NAT gateways are often shared by multiple bot-infected PCs and thus state that IP address-based estimates would underestimate the infection count by a factor of more than three times. We estimate an upper bound pop_{max} by multiplying the number of bots found after 24 hours by ten (accounting for the fact that more than 80% of the bots are NATed, that there are multiple bots behind one NATed IP address, and that the 24-hour-long crawling process did not fully converge yet).

Table 6.3 that ZeroAccess v2 is by far the largest botnet, spanning up to 2.5 million infections in its four subnetworks. Of these peers, 1.67 million (65%) are on 32-bit systems and 0.87 million (35%) are on 64-bit systems. Sality v4 seems

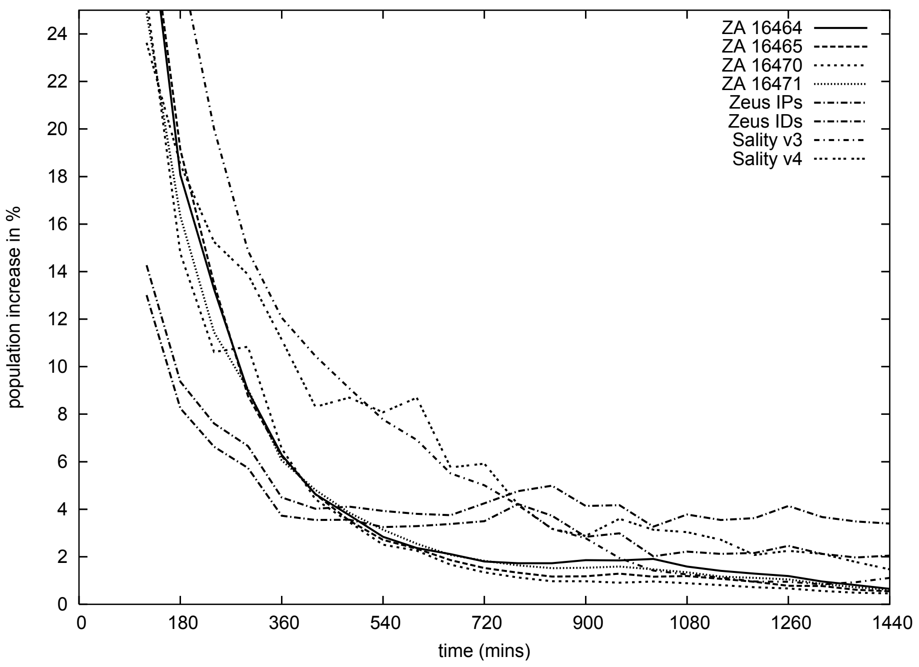


Figure 6.4: P2P Botnets Crawling Convergence

to be the smallest P2P network with about 25,000 bots. This is quite surprising, because – as we will show – the resilience of Sality v4 is significantly higher than in the case of ZeroAccess. Similarly, for some reason, the Sality botmasters decided not to migrate bots from the old protocol (v3) to the new protocol (v4). Consequently, although it has significant improvements over version three, Sality v4 remained relatively small over the years. Last, we estimate the population of only the P2P variant of Zeus to about 145,000 infections.

6.5 P2P Botnet Resilience

The P2P botnet designs offer unique defensive measures that differ from countermeasures against centralized or semi-distributed botnets. For example, researchers can estimate P2P botnet sizes, ISPs can identify infected customers, or authorities can potentially sinkhole the botnet. In this section, we evaluate the *resilience* of P2P botnets against such countermeasures.

6.5.1 Reconnaissance Resilience

P2P botnet reconnaissance allows defenders to enumerate infected PCs, which is typically also required to perform other mitigation strategies. Thus, from an attacker's perspective, it is desirable to prevent P2P botnet reconnaissance to the highest degree possible. Table 6.2 (page 96) has shown that it is not always

Botnet	**1 hr**	**12 hrs**	**24 hrs**	pop_{min}	pop_{max}
Sality v3	6768	29,688	36,415	150k	365k
Sality v4	546	1808	2490	10k	25k
ZeroAccess v2 16464	32,432	77,506	91,732	385k	920k
ZeroAccess v2 16465	14,116	33,766	38,067	170k	380k
ZeroAccess v2 16470	20,367	44,726	49,178	225k	490k
ZeroAccess v2 16471	29,155	65,332	75,553	325k	750k
Zeus	4611	8987	14,247	45k	140k

Table 6.3: Crawling results after 1 / 12 / 24 hours and botnet population estimations

possible to enumerate all bots in the existing P2P botnets using crawling. In particular, botnets like Sality or ZeroAccess do not include NATed peers in their peerlist. Consequently, crawling without further peerlist manipulations cannot enumerate all active peers. Unfortunately, not disclosing NATed peers effectively avoids reconnaissance for the majority of all bots. In these cases, defenders can propagate a fake peer in the botnet. When such a fake peer gains popularity (high in-degree) in the network, it will also be contacted by NATed peers. We leave such implementations for future work. We further found that Zeus implements an IP-address blacklist to exclude crawlers from the network. Peers are automatically blacklisted if they request peerlists too frequently. In addition, Sality deploys a so called "purity control" routine that verifies if a peer is adhering to the protocol and comparing the version number of the remote peer. Such approaches can significantly increase reconnaissance resilience in future P2P botnets.

However, the intrinsic requirements of P2P bots generally foster reconnaissance measures. For example, new bots need to get neighbor information to bootstrap the network. Thus, P2P protocols typically deploy peerlist exchange messages. Similarly, P2P botnets need to cope with changing end point addresses, such as IP address churn. Consequently, P2P bots usually integrate or update peers to their peerlist at some point. Depending on the strategy chosen to share peerlist entries, it requires specific strategies to fully explore the peerlists of bots. For example, when receiving peerlist requests, the Zeus bot returns peers close to a search key. A binary search helps to exhaustively crawl peerlists of bots. Summarizing, although technically possible, current P2P botnets only modestly defend against crawling.

6.5.2 C&C Resilience

Once the C&C protocol of a P2P botnet is known, defenders or other fraudsters may want to *abuse* the C&C for their purposes. For example, defenders may want to send a command to stop spamming, or may want to launch disinfection routines. Similarly, fraudsters could potentially abuse a P2P botnet (that they do not "own") by injecting own malicious commands.

Table 6.4 summarizes the C&C robustness of P2P botnets. All bots but Miner

Botnet	**Crypto**	**Signed**	**Replay?**
Hlux	BF3D	RSA2048	No
Miner	None	No	Yes
Sality v3	RC4	RSA1024	No
Sality v4	RC4	RSA2048	No
Storm	XOR	No	Yes
Waledac	AES	No	Yes
ZeroAccess v1	RC4	RSA512	Yes
ZeroAccess v2	XOR	RSA1024	No
Zeus	chXOR	RSA2048	Yes

Table 6.4: P2P-based C&C resilience characteristics. Legend: chXOR = chained XOR; BF3D = Blowfish+3DES.

encrypt their C&C communication. Zeus and Sality encrypt in a way that even identical C&C messages look arbitrary after encryption, effectively evading IDSs that base on C&C payload signatures. In addition, we found that three botnets, Miner, Storm and Waledac, did not sign the commands. In theory, given the symmetric encryption in these networks, defenders (or other attackers) could inject custom commands. Last, although their commands are signed, Zeus and ZeroAccess v1 allow defenders to replay old commands. Depending on the recorded commands, replaying C&C commands may help to disturb a P2P botnet.

6.5.3 Sinkholing Resilience

Sinkholing is a widely applied botnet mitigation strategy that can also be applied to P2P botnets. With a P2P botnet sinkhole, bots communicate only to end points controlled by defenders. Sinkholing is efficient, as it assumes that peerlists can be manipulated via the botnet's C&C protocol. Sinkholing neither requires physical access to the infected hosts, nor entities such as A/V vendors or ISPs to actively participate in the sinkholing efforts. The basic principle of sinkholing is to manipulate the peerlists of bots such that all entries point to special servers. Typically, to learn about new nodes in the P2P network, P2P botnets foresee mechanisms to *add of new entries to the peerlist*. Moreover, to handle IP address or port churn, most P2P-based C&C protocols include a mechanism to *update existing peerlist entries*. Depending on the P2P protocol implementation, these two techniques allow one to sinkhole a P2P botnet.

Table 6.5 summarizes the peerlist management strategies chosen by historic (upper part) and existing (lower part) unstructured botnets. In particular, Table 6.5 shows in which situations sinkhole entries can be *added* to the peerlists and which existing peerlists entries can be *overwritten*. For all existing P2P botnets, we can manipulate the peerlists of bots such that entries point to sinkhole addresses. To practically verify the sinkholing resilience, we developed attack prototypes for all existing unstructured P2P botnets.

Botnet	Add peers	Overwrite peers	Backup
Waledac	if more recent	all (500-1000)	fast-flux
Miner	never	no	Centr. C&C
Hlux	if more recent	half (250)	fast-flux
Sality v3	if higher rep.	1 (IP spoofing)	None
Sality v4	if higher rep.	1 (IP spoofing)	None
ZeroAccess v1	if more recent	all (256)	None
ZeroAccess v2	if more recent	16 per message	None
Zeus	if $\lvert PL\rvert < 50$	10 per message	DGA

Table 6.5: Peerlist management strategies of P2P botnets

Zeus

When receiving a request, the Zeus bot adds the sender to its peerlist if it knows fewer than 50 peers. Once a peerlist is saturated with 50 entries, Zeus does not add or replace entries. However, when receiving a request from a peer whose unique ID is in a bot's peerlist, a recipient would update the end-point address for the peer ID of the sender. By spoofing node IDs, it is possible to redirect existing peerlist entries to a sinkhole. In addition, bots also check the current peerlist size, and request new nodes from the sinkhole if the size is below 25 entries. Summarizing, we see that one can introduce the sinkhole to a peerlist.

However, the Zeus P2P command layer remains active as long as peers have a high out-degree. To successfully sinkhole Zeus, it is vital to remove or invalidate all peerlist entries that do not point a sinkhole. We achieve this by invalidating existing entries. In particular, when communicating with Zeus bots, we fake the source bot ID in Zeus messages to overwrite Zeus' peerlist entries with sinkhole IP addresses. Whenever a Zeus message is from a source ID existing in the recipients peerlist, the recipient will update the end point information to the sender's address. In addition, Zeus frequently tries to contact and validate existing entries, and deletes manipulated entries that are not reachable anymore. Our prototype has shown that we can remove all entries from the Zeus peerlists this way.

Sality

Like with Zeus, it is straightforward to add a sinkhole address to the peerlist of Sality bots. Unlike any other P2P botnet, Sality deploys a reputation scheme per entry in its peerlist. Peers gain reputation if they are reachable and strictly follow the most recent Sality P2P protocol dialect, or lose reputation otherwise. This reputation scheme becomes important when observing how Sality adds peers to its peerlist. In particular, if the list is saturated (1000 entries), Sality replaces the peer with the lowest reputation with the new peer. This way we can replace only peers with low reputation scores.

The main sinkholing challenge is to remove entries with a high reputation from the peerlist. To replace these peers, we use Sality's peer announcement mechanism to update the port of existing peerlist entries. IP spoofing enables us to invalidate existing peerlists entries as follows: when Sality receives an announcement message from an IP address that is in its peerlist, but the source port differs, Sality verifies if the announced bot is reachable. Once the verification process succeeds, the bot updates the peer's port to an incorrect value, invalidating the entry for future reputation checks. This way, eventually, the reputation of manipulated peerlist entries will decrease. Similarly, this attack effectively cuts off Sality's communication channel to other bots, as the only valid entries point to sinkholes. However, from a practical point of view, this process may take years to finish, making it unlikely that the attack succeeds.

ZeroAccess

ZeroAccess allows defenders to add new peers to the peerlist as long as the new peers are more recent than existing entries. As a peerlist response also contains the age of peerlist entries, in our attack scenario, we set the timestamp to the maximum value. These fake entries then have a high likelihood to be merged into the recipient's peerlist. The two ZeroAccess variants differ in the way in which fake peerlists can be sent to the bots. In the earlier ZeroAccess version, peerlists can be served only to peers when they request a peerlist from a sinkhole. In the more recent ZeroAccess version, peerlist messages will be accepted by peers even though they did not request them. Either way, our prototype efficiently succeeded to replace entire peerlists of ZeroAccess bots.

Unfortunately, sinkholing ZeroAccess also requires a minimization of the in-degree of all peers. Both ZeroAccess variants deploy a self-healing protocol, that is, peerlists are restored by broadcasting non-NATed peers to all neighbors. When receiving a propagation message, ZeroAccess merges both the sender and the propagated peer into its peerlist. As a consequence, nodes receiving such broadcasts, that is, all non-NATed peers, effectively restore their peerlists. A complete ZeroAccess sinkholing evaluation thus requires manipulation of many peerlists simultaneously, which we did not integrate in our prototype sinkhole due to possible collateral damage. Still, we have shown that all NATed peers, the vast majority of the bots, can be sinkholed.

Hlux

Hlux, in all its three versions, has major resilience weaknesses. Similar to ZeroAccess, Hlux merges new peers to its peerlist, favoring more recent peers. To sinkhole Hlux, we can connect to any existing bot and push an arbitrary peerlist, replacing half of the remote peer's peerlist. Kaspersky's sinkholing attempts against the first two Hlux versions support these claims [24]. In the third version, the Hlux botmasters introduced a multi-layer obfuscation of the P2P messages. However, the P2P protocol and architecture remained the same, enabling for similar attacks as with the first two Hlux versions.

For all sinkholing operations, it is typically infeasible to directly manipulate peerlists of NATed peers. A NAT gateway typically does not accept any incoming packet if the stream was not initiated by the local system. However, NATed peers will soon learn about sinkholes via peerlist requests issued towards non-NATed peers. When NATed peers contact a sinkhole, the sinkhole can interact with the NATed peer from this moment on. If the P2P protocol allows it, the sinkhole can then manipulate the NATed peers' peerlists.

Similarly, it is important to consider backup C&C channels of P2P botnets. For example, if a Zeus bot has an empty peerlist, it tries to receive new peerlists from DGA-generated domains. Similarly, Miner and Hlux contained mechanisms to contact (hard coded) centralized C&C servers in case their out-degree is zero. However, our sinkholing prototypes can take these backup channels into account and adapt the strategies accordingly.

Summarizing, sinkholing is a complex but effective mitigation technique against unstructured P2P botnets. Our attack prototypes have shown that none of the existing botnets are designed sufficiently resilient to withstand sinkholing. However, we can recognize a clear tendency towards more resilient P2P botnets that, for example, deploy reputation schemes or avoid modifying saturated peerlists. In particular, it requires only minor changes to both the Zeus and Sality protocols to circumvent sinkholing. We will discuss possible P2P botnet evolutions in Section 6.6.

6.5.4 Partitioning Resilience

Partitioning is a technique to split P2P botnets into many small, separately connected networks. In contrast to sinkholing, partitioning attacks do not insert fake entries to peerlists. Instead, partitioning is based on the fact that the botmaster has to inject commands in all network partitions to reach all peers. If the botmaster does not track all bots, it loses contact to a significant fraction of the bot population. Technically, like sinkholing, partitioning also requires peerlist manipulations. A defender can manipulate peerlists such that bots, for example, only know peers in their bot-ID or IP-address proximity. Assume, for simplicity, a uniform distribution of bots across the IP-address space. A defender can then, for example, create $2^{12} = 4096$ logical subnetworks of a botnet by pointing bots only to peers in the same /12 IP-address network.

However, it is significantly harder to evaluate the success of partitioning than for sinkholing. Partitioning does not produce direct feedback such as bots contacting the sinkhole. For this reason, we did not practically evaluate whether partitioning the botnets is feasible. Similarly, partitioning is less efficient, as no sinkhole participates in peerlist manipulations. On the other hand, partitioning has important benefits. First, partitioning does not require resources for sinkholes, such as bandwidth or IP addresses. Moreover, partitioning is less conspicuous than sinkholing, as all peerlist entries point to normal peers. For the same reason, botmasters cannot easily deploy reactive measures such as IP-address blacklists. Lastly, it is possible to combine sinkholing and partitioning. For example, one could sinkhole only a subset of peers, and partition all other bots such that they peer only with this sinkholed subset.

6.6 Discussion

We have shown prototypes for crawling, sinkholing and partitioning existing P2P botnets. Given these neutral facts, this section discusses how to proceed with this knowledge. For example, is P2P botnet sinkholing an ethical measure that advances security research? And, for the future, which alternatives exists for mitigating resilient P2P botnets?

Discussions on botnet takedowns often argue that attacks on P2P botnets provoke the creation of newer and more advanced botnets. In fact, we have observed that Storm was succeeded by Waledac, and later Hlux succeeded Waledac. Similarly, after sinkholing the first two Hlux variants, shortly thereafter new botnets with almost equal architecture arose. Two lessons can be learned from these observations. First, botnet successors typically have similar (nonresilient) architectures, although the attacks were discussed publicly and attackers could have learned from them. This shows that, so far, the strategies for botnet takedowns need to be only slightly changed to cope with adapted architectures of new bot variants. Consequently, a thorough sinkholing strategy that is adapted for future bot variants may eventually cease a botnet operation. Our systematic P2P botnet resilience evaluation aims to assist in future mitigation strategies.

Second, as long as malware-installation markets such as pay-per-install exists, attackers can spawn new botnets in a short time. Most botnets attacked so far, P2P-based or not, were botnets with *direct* harm. For example, Storm / Waledac / Hlux were spambots, and Torpig / Zeus are banking trojans. Malware downloaders such as Sality / ZeroAccess only cause *indirect* harm, in that they install other malware (e.g., bots with direct harm) on infected systems. Research-wise, sinkholing a combination of both types of botnets could deliver important insights. This way it could be analyzed if botmasters can still easily acquire new infections, as some parts of their typical installation chain (malware downloaders) are disrupted.

Moreover, it is unclear if botnet sinkholing is ethical and legal. On the one hand, sinkholing can mitigate the harm of a botnet affecting millions of victims. Still, how far may defenders go in their activities? Is peerlist manipulation already too intrusive? Are defenders allowed to impersonate IP addresses (i.e., spoof IP packets) for the good? Who has the authority to give defenders legal backup when performing sinkhole operations? A major difference to centralized botnets is the lack of responsibility in P2P botnets. In centralized botnets, C&C-domain registries or C&C-server hosting providers may be held responsible for operating the C&C infrastructure, while a central responsibility for distributed botnets simply does not exist. We clearly need further assistance to cope with these responsibility and legal issues for future botnets.

Lastly, we would like to foster a discussion about the resilience of future P2P botnets. By no means do our results prove that any future P2P botnet will offer angles for mitigation attempts. In fact, some botnets could improve their resilience with only minor tweaks. We would then face botnets that cannot be sinkholed, asking for alternative approaches to mitigate such networks. While controversial, a new field of research could analyze weaknesses in bot command signatures or malware binary vulnerabilities to disinfect clients remotely.

However, despite their weak resilience schemes, current P2P botnets have been operational for longer than a year, and their revenues probably exceed their costs. We thus cannot foresee if botmasters see even a need to design more resilient P2P botnets. In any case, our attack categorization aims to assist in complex future P2P botnet mitigation efforts. If eventually a bullet-proof P2P botnet arises, we at least provide a systematic approach to evaluate the botnet resilience.

6.7 Related Work

Most related to this work are very detailed malware analysis reports published by others [23, 24, 33, 34, 58, 63, 89, 96]. We thankfully used initial reverse engineering insights from these reports, largely assisting our manual code analysis. In this work, we focus on the resilience of the P2P architectures. Thus, we extend the current insights from initial reports with P2P protocol analyses and assess the resilience of these architectures.

Other researchers have already shown that sinkholing operations can be successful. Holz et al. presented their crawling and sinkholing results from the Storm bot [42]. In addition, they discussed general resilience weaknesses of structured P2P botnets. We showed in Section 6.3 that Storm was in fact the last structured P2P botnet, prompting for more research about unstructured P2P botnets. Stock et al. presented their sinkholing results from Waledac, the first-ever attacked unstructured P2P botnet [84]. We were motivated by the success of these two cases and provided an overview of further existing P2P botnets. In addition, we analyze if similar mitigation techniques can be applied to other existing botnets. Furthermore, we compare the botnet population of current botnets with P2P botnets in the past.

The problem of crawling P2P botnets was first addressed by Kanich et al. [48], who present advice for future enumeration attempts based on the lessons learned by crawling the Storm botnet. An alternative approach to enumerate all infected PCs (included NATed hosts) for structured P2P botnets was proposed by Kang et al. [47]. In their system, the authors introduce many fake nodes that listen for search requests, which could be adapted to unstructured P2P botnets. However, to not threaten future sinkhole attempts, we chose to enumerate hosts with less intrusive crawls that do not require peerlist manipulations. In contrast to the system proposed by Kang et al. [47], our botnet population estimations in Section 6.4 are immune to such poisoning attempts, as we verify whether peers respond to our requests.

Another branch of research analyzes if and how highly resilient P2P botnets can be designed. For example, Starnberger et al. propose Overbot [81], a structured P2P botnet design trying to hide the identities of bots in existing P2P networks. Similarly, Yan et al. propose the structured P2P botnet design RatBot [97]. In RatBot, bots in the structured P2P botnet randomly send command requests with spoofed IP addresses, injecting noise to the number of crawled peers. Yan et al. propose AntBot [98], a structured P2P botnet which relies on trustful communication based on a secret key shared between the botmaster and the bots. We are not aware of existing P2P botnets that base on ideas from these academic proposals, though. About a year after Sality appeared with its reputation scheme, Hund et al. introduced Rambot [43], an unstructured P2P botnet with proof-of-work and peer reputation scheme.

6.8 Conclusion

P2P botnet architectures have the potential to be much more resilient than centralized botnets, as they lack single point of failures. In fact, we have shown how popular P2P botnets have become over the years: as of September 2012, we have observed 12 disjoint active P2P botnets, using six distinct proprietary P2P protocols. When monitoring only eight of these botnets on a normal weekday, we counted multiple millions of infections.

Motivated by these insights, we evaluated the resilience of these P2P botnets against reconnaissance, C&C abuse, sinkholing and partitioning. We reverse engineered the bots and prototyped sinkholing attacks against all active P2P botnets. Some of these prototypes are currently integrated to actual sinkholing attempts. However, we also found that P2P botnets increased their resilience by introducing self-healing P2P protocols and reputation schemes. Although bulletproof P2P botnet designs do not exist yet, future P2P botnets may demand for new mitigation techniques.

Part III

Epilog

7 Conclusions and Future Work

7.1 Conclusions

Botnets have been a major security threat since many years, but until now little research was done to understand the resilience of these malicious networks. However, botnet resilience causes botnets to remain operable in the long term, continuously harming end users by attacks like DDoS, spam, identity theft or extortion. With the contributions in this thesis, we improve the current state of botnet resilience research. We have estimated the resilience of botnets, and we have explored techniques that botmasters use to improve botnet resilience.

In Chapter 3, we showed how to perform malware analysis for sound scientific experimentation. We developed guidelines that enabled us to analyze botnet resilience in a safe, transparent, realistic and scientifically correct manner. Our survey on 36 academic publications highlighted the importance of such guidelines, as most of the surveyed papers would also have benefited from similar best practices. We used these guidelines to design a dynamic malware analysis system called Sandnet. Since we launched Sandnet in February 2010, it greatly assisted us in analyzing botnet resilience, as described in Chapter 4.

When we started analyzing botnet resilience, we soon observed that not only the botnet resilience determines how long and successful botnets are operated. Instead, as long as botmasters can buy new installs on the underground market, the root cause and the problem of botnets will persist. We thus separated the actual botnet resilience analysis into two chapters.

In Chapter 5, we described the methodology of 23 malware downloader families. Malware downloaders are one of the main sources for new malware installations on the underground market. We showed that malware downloaders are used to persistently drop thousands of malware samples. The resilience of these downloader networks is achieved both via technical and organizational means. Technically, attackers encrypt and try to hide C&C channels, often separating C&C infrastructures from malware hosting infrastructures. Next to technical measures, malware downloaders also systematically fluctuate their C&C hosting providers and C&C domain registrars. Consequently, malware downloaders can reliably feed new installations to botmasters, remaining a root cause for botnets.

But even without such continuous feeds of new infections, Chapter 6 has shown that botnets themselves can be highly resilient. We compared the resilience of six existing peer-to-peer (P2P) botnets with historic P2P botnets. We successfully prototyped mitigation techniques for each of the botnets, preparing sinkholing operations against botnets such as Zeus and ZeroAccess. Despite these successes, though, we observed a tendency towards more resilience P2P botnets. Botmasters increasingly deploy mitigation defenses such as reputation schemes or P2P protocols with self-healing peerlists. In addition, we found P2P botnets that use secondary C&C channels as backup if their P2P C&C component is mitigated by defenders. Overall, while current P2P botnets still offer resilience weaknesses, only minor changes in P2P protocols would render such networks highly resilient.

Summarizing, we have analyzed resilience techniques that botmasters use to successfully operate their botnets for many years. As we have shown, botnets rarely show single point of failures. Even if C&C server takedowns cause a botnet to disrupt temporarily, eventually the botmaster will use its remaining infrastructures to reactivate the network, buying new malware installs if necessary. Consequently, operations to successfully disrupt botnets must mitigate all communication means simultaneously. However, C&C servers and C&C domains are hosted by many different institutions, often spanning multiple time zones and varying legislations. Given these complex designs, the resilience analyses in this thesis can greatly help to assist in understanding the overall botnet resilience for future mitigation operations.

7.2 Future Work

One goal of this work has been to understand the resilience of botnets. After we fulfilled this goal for a large number of botnets, a possible next step is to collaborate in botnet mitigation operations that abuse the resilience weaknesses that we found. Although botnet takedowns were analyzed in the past, research-wise, we can still learn from such operations. For example, it is yet unclear how and how fast botnets are rebuilt once they are taken down. We plan to measure this in the future, in particular, by disrupting two botnets simultaneously: one with direct harm, and at the same time, disrupting the botnet that helps to spread installations to the former one. We could then measure the effect by observing evolutions of either botnet. Similarly, as of now, little data exists on takedowns of malware downloader botnets such as Sality or ZeroAccess. A possible research project could analyze the economy behind such takedowns, as losing and buying bot installations financially affects botmasters.

In this thesis, we focused our P2P resilience analysis on P2P botnets. However, numerous other legitimate systems are based on P2P architectures, such as the VoIP software Skype [66] or file sharing networks like BitTorrent [21]. Applying our P2P resilience analysis to these networks is an interesting research question itself. A research project could investigate whether such large-scale P2P systems have similar resilience weaknesses.

As we have shown, we may encounter future botnets that cannot be disrupted at the C&C network layer anymore. It is an open research question how to mitigate the harm of such botnets. One possible research direction is exploring ways how to remotely disinfect victims' systems. Until now, malware removals are performed on the host by anti-virus products or tools like Microsoft's Malicious Software Removal Tool (MSRT) [65]. However, particularly bots with server functionalities such as P2P bots may have vulnerabilities that allow for remote disinfection. Similarly, by refactoring cryptographic keys or finding weaknesses in the command layer of botnets, it may be possible to inject disinfection commands to the bots. Although controversially discussed, these methods may be the only way to enforce disruptions of future botnets. We see the exploration of such proactive malware disinfection routines as another research project.

Lastly, botnet resilience is also determined by the human factor. Botnets can persist only because most users are not aware of the malware that is hidden on their systems. So far, only little research explored how to reliably inform infected users. Our P2P botnet crawling has identified millions of infections, and we have shared this data with security organizations such as CERTs and Shadowserver [4]. Yet a large fraction of users remain unnoticed, not knowing they risk losing data or participating in attacks like spam or DDoS. Until now, there is no viable solution to inform infected users. Designing a trustful, secure, and reliable infection alert system remains an open research question.

Bibliography

[1] Antivirus tracker. `http://avtracker.info/`.

[2] Hispasec Sistemas - VirusTotal. `http://www.virustotal.com/`.

[3] Sandnet. `http://www.if-is.net/sandnet/`.

[4] Shadowserver Foundation. `http://www.shadowserver.org`.

[5] ThreatExpert. `http://www.threatexpert.com/`.

[6] yara-project - A malware identification and classification tool. `http://code.google.com/p/yara-project/`.

[7] Abuse.ch. ZeuS Tracker. `https://zeustracker.abuse.ch/`.

[8] M. Antonakakis, R. Perdisci, W. Lee, N. Vasiloglou, and D. Dagon. Detecting Malware Domains at the Upper DNS Hierarchy. In *Proceedings of the 20th USENIX Security Symposium, San Francisco, CA*, August 2011.

[9] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon. From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware. In *Proceedings of the 21st USENIX Security Symposium, Bellevue, WA*, August 2012.

[10] A. J. Aviv and A. Haeberlen. Challenges in experimenting with botnet detection systems. In *Proceedings of the 4th USENIX Workshop on Cyber Security Experimentation and Test (CSET), San Francisco, CA*, August 2011.

[11] P. Baecher, M. Koetter, M. Dornseif, and F. Freiling. The Nepenthes Platform: An Efficient Approach to Collect Malware. In *Proceedings of the 9th International Symposium On Recent Advances In Intrusion Detection (RAID), Hamburg, Germany*, September 2006.

[12] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario. Automated Classification and Analysis of Internet Malware.

In *Proceedings of the 10th International Symposium on Recent Advances in Intrusion Detection (RAID), Queensland, Australia*, September 2007.

[13] D. Balzarotti, M. Cova, C. Karlberger, C. Kruegel, E. Kirda, and G. Vigna. Efficient Detection of Split Personalities in Malware. In *Proceedings of the 17th Annual Network and Distributed Systems Security Symposium (NDSS), San Diego, CA*, February 2010.

[14] P. Barford and M. Blodgett. Toward Botnet Mesocosms. In *1st USENIX Workshop on Hot Topics in Understanding Botnets (HotBots), Cambridge, MA*, April 2007.

[15] U. Bayer, I. Habibi, D. Balzarotti, E. Kirda, and C. Kruegel. A View on Current Malware Behaviors. In *Proceedings of the 2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET), Boston, MA*, April 2009.

[16] U. Bayer, E. Kirda, and C. Kruegel. Improving the Efficiency of Dynamic Malware Analysis. In *Proceedings of the 25th ACM Symposium On Applied Computing (SAC), Sierre, Switzerland*, March 2010.

[17] U. Bayer, C. Kruegel, and E. Kirda. TTAnalyze: A Tool for Analyzing Malware. In *Proceedings of the 16th Annual EICAR Conference, Hamburg, Germany*, April 2006.

[18] U. Bayer, P. Milani Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda. Scalable, Behavior-Based Malware Clustering. In *Proceedings of the 16th Annual Network & Distributed System Security Symposium (NDSS), San Diego, CA*, February 2009.

[19] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. EXPOSURE : Finding Malicious Domains Using Passive DNS Analysis. In *Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA*, February 2011.

[20] Bit9. FileAdvisor. `http://fileadvisor.bit9.com`.

[21] BitTorrent, Inc. BitTorrent. `http://en.wikipedia.org/wiki/Bittorrent`.

[22] B. Bowen, P. Prabhu, V. Kemerlis, S. Sidiroglou, A. Keromytis, and S. Stolfo. BotSwindler: Tamper Resistant Injection of Believable Decoys in VM-Based Hosts for Crimeware Detection. In *Proceedings of the 13th International Symposium on Recent Advances in Intrusion Detection (RAID), Ottawa, Canada*, September 2010.

[23] T. Bukowski. ZeuS v3 P2P Network Monitoring. Technical Report by CERT.PL, 2012.

[24] P.-M. Bureau. Same Botnet, Same Guys, New Code: Win32/Kelihos. In *VirusBulletin 2011.*

[25] J. Caballero, C. Grier, C. Kreibich, and V. Paxson. Measuring Pay-per-Install: The Commoditization of Malware Distribution. In *Proceedings of the 20th USENIX Security Symposium, San Francisco, CA, August 2011*, San Francisco, CA, August 2011.

[26] J. Caballero, P. Poosankam, C. Kreibich, and D. Song. Dispatcher: Enabling Active Botnet Infiltration using Automatic Protocol Reverse-Engineering. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS), Chicago, IL*, November 2009.

[27] L. Cavallaro, C. Kruegel, and G. Vigna. Mining the Network Behavior of Bots, Technical Report, 2009.

[28] Z. B. Celik, J. Raghuram, G. Kesidis, and D. J. Miller. Salting public traces with attack traffic to test flow classifiers. In *Proceedings of the 4th USENIX Workshop on Cyber Security Experimentation and Test (CSET), San Francisco, CA*, August 2011.

[29] X. Chen, J. Andersen, Z. M. Mao, M. Bailey, and J. Nazario. Towards an Understanding of Anti-virtualization and Anti-debugging Behavior in Modern Malware. In *The 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Achorage, AK*, June 2008.

[30] R. Clayton, S. J. Murdoch, and R. N. M. Watson. Ignoring the Great Firewall of China. In *Privacy Enhancing Technologies*, pages 20–35, 2006.

[31] C. J. Dietrich, C. Rossow, F. C. Freiling, H. Bos, M. van Steen, and N. Pohlmann. On Botnets that use DNS for Command and Control. In *Proceedings of the European Conference on Computer Network Defense (EC2ND), Gothenburg, Sweden*, September 2011.

[32] A. Dinaburg, P. Royal, M. Sharif, and W. Lee. Ether: Malware Analysis via Hardware Virtualization Extensions. In *15th ACM Computer and Communications Security Conference (CCS), Alexandria, VA*, October 2008.

[33] N. Falliere. Sality: Story of a Peer-to-Peer Viral Network. Technical Report by Symantec, July 2011.

[34] N. Fitzgibbon and M. Wood. Conficker.C - A Technical Analysis. Technical Report by SophosLabs, Sophos Inc., April 2009.

[35] J. Goebel and T. Holz. Rishi: Identify Bot Contaminated Hosts by IRC Nickname Evaluation. In *Proceedings of the 1st USENIX Workshop on Hot Topics in Understanding Botnets (HotBots), Cambridge, MA*, April 2007.

[36] S. Golovanov and V. Rusakov. TDSS. `http://www.securelist.com/en/analysis/204792131/TDSS`.

[37] C. Grier, L. Ballard, J. Caballero, N. Chachra, C. J. Dietrich, K. Levchenko, P. Mavrommatis, D. McCoy, A. Nappa, A. Pitsillidis, N. Provos, M. Z. Rafique, M. A. Rajab, C. Rossow, K. Thomas, V. Paxson, S. Savage, and

G. M. Voelker. Manufacturing Compromise: The Emergence of Exploit-as-a-Service. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, October 2012.

[38] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. In *Proceedings of the 17th USENIX Security Symposium, San Jose, CA*, August 2008.

[39] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation. In *Proceedings of the 16th USENIX Security Symposium, Boston, MA*, August 2007.

[40] G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. In *Proceedings of the 16th Annual Network & Distributed System Security Symposium (NDSS), San Diego, CA*, February 2008.

[41] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling. Measuring and detecting fast-flux service networks. In *Proceedings of the 16th Annual Network & Distributed System Security Symposium (NDSS), San Diego, CA*, February 2008.

[42] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling. Measurements and Mitigation of Peer-to-Peer-based Botnets: A Case Study on Storm Worm. In *Proceedings of the 1st USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET), San Francisco, CA*, April 2008.

[43] R. Hund, M. Hamann, and T. Holz. Towards Next-Generation Botnets. In *Proceedings of the European Conference on Computer Network Defense (EC2ND), Dublin, Ireland*, December 2008.

[44] G. Jacob, R. Hund, C. Kruegel, and T. Holz. JACKSTRAWS: Picking Command and Control Connections from Bot Traffic. In *Proceedings of the 20th USENIX Security Symposium, San Francisco, CA*, August 2011.

[45] J. Jang, D. Brumley, and S. Venkataraman. Bitshred: feature hashing malware for scalable triage and semantic analysis. In *18th ACM Conference on Computer and Communications Security (CCS), Chicago, IL*, October 2011.

[46] J. P. John, A. Moshchuk, S. D. Gribble, and A. Krishnamurthy. Studying Spamming Botnets Using Botlab. In *6th USENIX Symposium on Networked Systems Design and Implementation (NSDI), Boston, MA*, April 2009.

[47] B. B. H. Kang, E. Chan-Tin, C. P. Lee, J. Tyra, H. J. Kang, C. Nunnery, Z. Wadler, G. Sinclair, N. Hopper, D. Dagon, and Y. Kim. Towards Complete Node Enumeration in a Peer-to-Peer Botnet. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security (ASIACCS), Sydney, Australia*, March 2009.

[48] C. Kanich, K. Levchenko, B. Enright, G. M. Voelker, and S. Savage. The Heisenbot Uncertainty Problem: Challenges in Separating Bots from Chaff. In *Proceedings of the 1st USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET), San Francisco, CA*, April 2008.

[49] D. Kirat, G. Vigna, and C. Kruegel. BareBox: Efficient Malware Analysis on Bare-Metal. In *Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC), Orlando, FL*, December 2011.

[50] E. Kirda, C. Kruegel, G. Banks, and G. Vigna. Behavior-based Spyware Detection. In *Proceedings of the 15th USENIX Security Symposium, Vancouver, Canada*, August 2006.

[51] C. Kolbitsch, T. Holz, C. Kruegel, and E. Kirda. Inspector Gadget: Automated Extraction of Proprietary Gadgets from Malware Binaries. In *Proceedings of the 30th IEEE Symposium on Security & Privacy (S&P), Oakland, CA*, May 2009.

[52] C. Kolbitsch, P. Milani Comparetti, C. Kruegel, E. Kirda, X. Zhou, X. Wang, U. C. S. Barbara, and S. Antipolis. Effective and Efficient Malware Detection at the End Host. In *Proceedings of the 18th USENIX Security Symposium, Montreal, Canada*, August 2009.

[53] J. Koziol, D. Litchfield, D. Aitel, C. Anley, S. Eren, N. Mehta, and R. Hassell. *The Shellcoder's Handbook: Discovering and Exploiting Security Holes.* John Wiley & Sons, 2004.

[54] C. Kreibich, N. Weaver, C. Kanich, W. Cui, and V. Paxson. GQ: Practical Containment for Measuring Modern Malware Systems. In *ACM Internet Measurement Conference (IMC), Berlin, Germany*, November 2011.

[55] B. Krishnamurthy and W. Willinger. What are our standards for validation of measurement-based networking research? In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), Annapolis, MD*, June 2008.

[56] S. Kurkowski, T. Camp, and M. Colagrosso. MANET Simulation Studies: The Incredibles. *SIGMOBILE Mobile Computing and Communications Review*, 9:50–61, October 2005.

[57] A. Lanzi, D. Balzarotti, C. Kruegel, M. Christoderescu, and E. Kirda. AccessMiner: Using System-Centric Models for Malware Protection. In *17th ACM Conference on Computer and Communications Security (CCS), Chicago, IL*, 2010.

[58] A. Lelli. Zeusbot/Spyeye P2P Updated, Fortifying the Botnet.

[59] P. Li, L. Liu, D. Gao, and M. K. Reiter. On Challenges in Evaluating Malware Clustering. In *Proceedings of the 13th International Symposium on Recent Advances in Intrusion Detection (RAID), Ottawa, Canada*, September 2010.

[60] M. Lindorfer, C. Kolbitsch, and P. Milani Comparetti. Detecting Environment-Sensitive Malware. In *14th International Symposium on Recent Advances in Intrusion Detection (RAID), Menlo Park, CA*, September 2011.

[61] L. Liu, S. Chen, G. Yan, and Z. Zhang. BotTracer: Execution-based Bot-like Malware Detection. In *Proceedings of the 11th Information Security Conference (ISC), Teipei, Taiwan*, September 2008.

[62] C. Livadas, B. Walsh, D. Lapsley, and T. Strayer. Using Machine Learning Techniques to Identify Botnet Traffic. In *Proceedings of the 31st IEEE Conference on Local Computer Networks (LCN), Tampa, FL*, November 2006.

[63] K. McNamee. Malware Analysis Report: ZeroAccess/Sirefef. Technical Report by Kindsight Security Labs, February 2012.

[64] J. Menn. *Fatal System Error: The Hunt for the New Crime Lords Who Are Bringing Down the Internet.* 2010.

[65] Microsoft Corporation. Malicious Software Removal Tool. `http://www.microsoft.com/security/pc-security/malware-removal.aspx`.

[66] Microsoft Skype Division. Skype. `http://en.wikipedia.org/wiki/Skype`.

[67] P. Milani Comparetti, G. Wondracek, C. Kruegel, and E. Kirda. Prospex: Protocol Specification Extraction. In *Proceedings of the 30th IEEE Symposium on Security & Privacy (S&P), Oakland, CA*, May 2009.

[68] J. A. Morales, A. Al-bataineh, S. Xu, and R. Sandhu. Analyzing and Exploiting Network Behaviors of Malware. In *Proceedings of the 6th International ICST Conference on Security and Privacy in Communication Networks (SecureComm), Singapore*, September 2010.

[69] J. Nazario and T. Holz. As the Net Churns: Fast-Flux Botnet Observations Tracking Fast-Flux Domains. In *Proceedings of the 3rd International Conference on Malicious and Unwanted Software (Malware), Alexandria, VA*, October 2008.

[70] M. Neugschwandtner, P. Milani Comparetti, and C. Platzer. Detecting Malware's Failover C&C Strategies with SQUEEZE. In *Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC), Orlando, FL*, December 2011.

[71] J. Newsome, D. Brumley, J. Franklin, and D. Song. Replayer: Automatic Protocol Replay by Binary Analysis. In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS), Alexandria, VA*, October 2006.

[72] Norman ASA. Norman Sandbox. `http://www.norman.com/security_center/security_tools/`.

[73] R. Paleari, L. Martignoni, G. Fresi Roglia, and D. Bruschi. A Fistful of Red-Pills: How to Automatically Generate Procedures to Detect CPU Emulators. In *Proceedings of the 3rd USENIX Workshop on Offensive Technologies (WOOT), Montreal, Canada*, August 2009.

[74] R. Perdisci, W. Lee, and N. Feamster. Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces. In *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI), Boston, MA*, March 2010.

[75] M. Polychronakis, P. Mavrommatis, and N. Provos. Ghost turns Zombie: Exploring the Life Cycle of Web-based Malware. In *Proceedings of the 1st USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET), San Francisco, CA*, April 2008.

[76] K. Rieck, T. Holz, C. Willems, and P. D. Learning and Classification of Malware Behavior. In *Proceedings of the 5th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA), Paris, France*, July 2008.

[77] K. Rieck, G. Schwenk, T. Limmer, T. Holz, and P. Laskov. Botzilla: Detecting the Phoning Home of Malicious Software. In *Proceedings of the 25th ACM Symposium on Applied Computing (SAC), Sierre, Switzerland*, March 2010.

[78] K. Rieck, P. Trinius, C. Willems, and T. Holz. Automatic Analysis of Malware Behavior using Machine Learning. In *Journal of Computer Security 2011.*

[79] C. Rossow, C. J. Dietrich, H. Bos, L. Cavallaro, M. van Steen, F. C. Freiling, and N. Pohlmann. Sandnet: Network Traffic Analysis of Malicious Software. In *Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS), Salzburg, Switzerland*, April 2011.

[80] R. Sommer and V. Paxson. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *Proceedings of the 31st IEEE Symposium on Security & Privacy (S&P), Berkeley, CA*, May 2010.

[81] G. Starnberger, C. Kruegel, and E. Kirda. Overbot - A botnet protocol based on Kademlia. In *Proceedings of the 4th International ICST Conference on Security and Privacy in Communication Networks (SecureComm), Istanbul, Turkey*, September 2008.

[82] E. Stinson and J. C. Mitchell. Characterizing Bots Remote Control Behavior. July 2007.

[83] E. Stinson and J. C. Mitchell. Towards Systematic Evaluation of the Evadability of Bot / Botnet Detection Methods. In *Proceedings of the 2nd USENIX Workshop on Offensive Technologies (WOOT), San Jose, CA*, July 2008.

[84] B. Stock, M. Engelberth, F. C. Freiling, and T. Holz. Walowdac Analysis of a Peer-to-Peer Botnet. In *European Conference on Computer Network Defense (EC2ND), Milan, Italy*, November 2009.

[85] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna. Your Botnet is My Botnet: Analysis of a Botnet Takeover. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS '09), Chicago, IL*, November 2009.

[86] B. Stone-Gross, M. Cova, B. Gilbert, L. Cavallaro, M. Szydlowski, C. Kruegel, G. Vigna, and R. Kemmerer. Your Botnet is My Botnet: Analysis of a Botnet Takeover. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS), Chicago, IL*, November 2009.

[87] B. Stone-Gross, C. Kruegel, K. Almeroth, A. Moser, and E. Kirda. FIRE: FInding Rogue nEtworks. In *Proceedings of the 25th Annual Computer Security Applications Conference (ACSAC), Honolulu, HI*, December 2009.

[88] W. T. Strayer, D. Lapsely, R. Walsh, and C. Livadas. Botnet Detection Based on Network Behavior. *Advances in Information Security, 2008*, 36:1–24, 2008.

[89] G. Tenebro. W32.Waledac: Threat Analysis. Technical Report by Symantec, August 2010.

[90] The Rendon Group. Conficker: Lessons Learned. Technical Report by the Conficker Working Group.

[91] The Shadowserver Foundation. bin-test. `http://bin-test.shadowserver.org/`.

[92] T. Werner. Kelihos.C: Same Code, New Botnet. Blog post by CrowdStrike: `http://blog.crowdstrike.com/2012/03/kelihosc-same-code-new-botnet.html`.

[93] T. Werner. The Miner Botnet: Bitcoin Mining Goes Peer-To-Peer. Blog article by Kaspersky: `http://www.securelist.com/en/blog/208193084/The_Miner_Botnet_Bitcoin_Mining_Goes_Peer_To_Peer`.

[94] C. Willems, T. Holz, and F. Freiling. Toward Automated Dynamic Malware Analysis Using CWSandbox. In *Proceedings of the 31st IEEE Security and Privacy Magazine*, volume 5, pages 32–39, March 2007.

[95] P. Wurzinger, L. Bilge, T. Holz, J. Goebel, C. Kruegel, and E. Kirda. Automatically Generating Models for Botnet Detection. In *Proceedings of the 14th European Symposium on Research in Computer Security (ESORICS), Saint Malo, France*, September 2009.

[96] J. Wyke. ZeroAccess. Technical Report by SophosLabs UK, 2012.

[97] G. Yan, S. Chen, and S. Eidenbenz. RatBot: Anti-enumeration Peer-to-Peer Botnets. In *Lecture Notes in Computer Science, 2011, Volume 7001/2011.*

[98] G. Yan, D. T. Ha, and S. Eidenbenz. AntBot: Anti-pollution peer-to-peer botnets. In *Journal of Computer Networks, Vol. 55, 2011.*

[99] T.-f. Yen and M. K. Reiter. Traffic Aggregation for Malware Detection. In *Proceedings of the 5th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA), Paris, France*, July 2008.

[100] X. Yu, X. Dong, G. Yu, Y. Qin, D. Yue, and Y. Zhao. Online Botnet Detection Based on Incremental Discrete Fourier Transform. *Journal of Networks*, 5(5):568–576, 2010.

[101] J. Zhang, X. Luo, R. Perdisci, G. Gu, W. Lee, and N. Feamster. Boosting the Scalability of Botnet Detection Using Adaptive Traffic Sampling. March 2011.

[102] Z. Zhu, V. Yegneswaran, and Y. Chen. Using Failure Information Analysis to Detect Enterprise Zombies. In *Proceedings of the 5th International ICST Conference on Security and Privacy in Communication Networks (SecureComm), Athens, Greece*, September 2009.

Glossary

A/V Anti-virus: An A/V vendor offers security solutions against viruses, often by searching for known A/V signatures in binaries. 43, 71, 83, 84, 100

AS Autonomous system: A collection of connected Internet Protocol routing prefixes, often similar but not limited to an ISP. 76

CERT Computer Emergency Response Team: A team handling computer security incidents for specific networks. 93, 95, 111

DDoS Distributed Denial of Service (attack): An attack, by which multiple remote-controlled systems simultaneously add a high load (in terms of bandwidth, connections, HTTP requests, etc.) towards one target, with the goal to disrupt its service. 6, 13, 70, 109, 111

DGA Domain Generation Algorithm: A botnet can use a DGA to compute its C&C domains dynamically, for example, based on a function that takes the current date as seeding input. 7, 8, 48, 102

DHT Distributed Hash Table: Decentralized system providing lookup services similar to a hash table. 90, 92

ISP Internet Service Provider: An organization that provides access to the Internet, for example, to end users. 82, 93, 95, 98, 100, 121

MD5 Message-Digest Algorithm 5: A typical cryptographic hash function, transforming a file to a 128bit checksum. 18, 34, 35, 43, 71, 74, 82, 84

NAT Network Address Translation: The process of modifying IP address information in IP packet headers while in transit across a traffic routing device. 19, 22, 34, 50, 70, 94–98, 101, 102, 104

NIDS Network intrusion detection system: An independent system that identifies intrusions or attacks by examining network traffic. 34, 70

OS Operating System, for example, Windows XP or Red Hat Linux. 18, 20, 34, 69

PE Portable Executable: File format for executable files on Microsoft Windows operating systems. 55, 58

PPI Pay-per-install: An underground service, in which an affiliate is paid for malware installations on exploited systems. 67, 68, 70, 85

SVM Support Vector Machine: A supervised learning model used for classification or regression analysis. 26

TLD Top Level Domain: One of the domains at the highest level in the DNS hierarchy, for example, `.com` or `.nl`. 77

TTL Time to live: In the context of DNS, TTL refers to the time a DNS response record may be cached. In the context of the IP protocol and P2P botnets, TTL refers to the maximum number of hops a packet is routed before it is dropped. 48, 90

VM Virtual Machine, for example, based on VirtualBox or VMware. 18, 37, 42, 82

Summary

Botnets, networks of remotely controllable malware-infected PC systems, impose a threat to millions of users by attacks such as spam, identity theft, or denial of service. While we aware of botnets in general, there is no solid understanding of the resilience of these malicious networks. For example, it is unclear how botmasters ensure that botnets remain operational for many years. Similarly, our community does not know which botnets operate for how long and how many infected PCs are part of these botnets.

In this thesis, we aim to improve the current state of botnet resilience research. To bootstrap our botnet resilience analysis, Chapter 3 discusses how to perform malware analysis for sound scientific experimentation. We propose guidelines that enable us to analyze botnet resilience in a safe, transparent, realistic and scientifically correct manner. By surveying 36 academic publications that perform malware experimentation, we highlighted the importance of such guidelines. Most of the surveyed papers would also have benefited from similar best practices. We then used these best practices to propose a dynamic malware analysis system called SANDNET in Chapter 4. We launched SANDNET in February 2010 as a supportive tool to analyze botnet resilience in this thesis.

A complete botnet resilience analysis has to cover two sides of the botnet problem. In particular, botnet resilience is also determined by the infrastructures attackers use to create or enlarge such networks. This thesis has shown that botnets rarely spread themselves anymore, but use malware installation infrastructures to obtain new infected PCs. Thus, next to botnet resilience itself, we also have to analyze the resilience of malware installation networks. This thesis separates the botnet resilience analysis into two parts.

First, in Chapter 5, we outline the workings of 23 malware downloader families. These malware downloaders, as we have shown, persistently drop thousands of malware samples. Attackers use technical and organizational means to improve the resilience of these networks. Technically, attackers encrypt and try to hide C&C channels, often separating C&C infrastructures from malware hosting infrastructures. Next to technical measures, malware downloaders also systematically fluctuate their C&C hosting providers and C&C domain registrars. With these techniques, malware downloaders remain a persistent root cause for suc-

cessful botnet operations. Levering the necessity of malware downloaders to host their infrastructures publicly, we propose two new techniques to automatically acquire malware samples from these infrastructures.

In Chapter 6, we show that botnet architectures themselves can be highly resilient. In particular, we highlight recent trends in the development of peer-to-peer (P2P) botnets, which are explicitly designed to be highly resilient. We compare the resilience of six existing P2P botnets with historic P2P botnets. To test the botnet resilience, we prototype mitigation techniques for each of the botnets. These prototypes have helped to prepare sinkholing operations against P2P botnets such as Zeus and ZeroAccess. Overall, though, we observed trends towards highly resilient P2P botnets that cannot easily be attacked anymore. For example, botmasters use reputation schemes or deploy P2P protocols with self-healing peerlists that mitigate sinkholing attempts. In addition, P2P botnets back off to using secondary C&C backup channels if their P2P C&C component fails. We find that only minor changes in P2P protocols would render such networks highly resilient. As a consequence, we expect further P2P botnets in the near future.

Overall, we analyzed techniques that botmasters use to successfully operate their botnets for many years. Our observations imply that botnet mitigations strategies have to go far beyond disruptions of single C&C servers or C&C domains. Such one-off initiatives would only cause temporary botnet disruptions, and botmasters will use their remaining infrastructures to reactivate the botnets. However, botnet resilience is also determined by relatively easy organizational decisions by the botmasters. For example, if C&C end points are hosted at multiple sites, mitigation efforts have to involve institutions in varying time zones and legislations. Our resilience analyses assists in the complex problem of understanding the overall botnet resilience. Such resilience analyses are, for example, helpful for future botnet mitigation operations. Our discussion raises awareness of these resilient botnets, fostering research to explore alternative counter-measures against these networks.

Samenvatting

Titel: Een evaluatie van de robuustheid van botnets door middel van malware analyse

Botnets, netwerken van op afstand bedienbare met malware-geïnfecteerde PC systemen, vormen een bedreiging voor miljoenen gebruikers door het versturen van spam, het stelen van gevoelige gegevens of het onbereikbaar maken van essentile services. De onderzoeksgemeenschap is zich bewust van de mogelijkheden van botnets in het algemeen, echter is er geen voldoende inzicht in de veerkracht van deze kwaadaardige netwerken. Zo is het onduidelijk hoe botmasters ervoor zorgen dat botnets gedurende vele jaren operationeel blijven, hoe lang botnets operationeel zijn en hoeveel geïnfecteerde PCs deel uitmaken van deze netwerken.

In dit proefschrift willen wij de manier van onderzoek naar de veerkracht van botnets verbeteren. Als eerste hebben we onderzocht hoe experimenten met botnets kunnen worden uitgevoerd op een wetenschappelijk verantwoorde manier. In hoofdstuk 3 bespreken wij richtlijnen die ons in staat stellen botnets te analyseren in een veilige, realistische omgeving en op een transparante wetenschappelijk correcte manier. Door middel van een literatuurstudie, bestaande uit 36 wetenschappelijke publicaties waarin malware experimenten worden beschreven, onderbouwen wij het belang van dergelijke richtlijnen. Uit dit onderzoek concluderen wij dat het merendeel van de bestudeerde publicaties profijt zouden hebben gehad bij het volgen van de voorgestelde richtlijnen. Vervolgens hebben we gebruik gemaakt van deze richtlijnen om Sandnet, een dynamische malware analyse systeem dat wordt besproken in hoofdstuk 4, te ontwikkelen. We lanceerden Sandnet in februari 2010 als een ondersteunende tool bij het uitvoeren van analyses naar botnets.

Een compleet onderzoek naar de veerkracht van botnets dient twee kanten van het botnet probleem te belichten. De veerkracht van botnets wordt namelijk mede bepaald door de infrastructuur waarvan de aanvallers gebruik maken of waarmee huidige kwaadaardige netwerken worden uitgebreid. In dit proefschrift laten wij zien dat botnets zelden zelf verspreiden, maar gebruik maken van een malware installatie infrastructuur om nieuwe PC systemen te infecteren. Dus om een compleet beeld te schetsen, moeten we ook de veerkracht van malware installatie infrastructuren analyseren. In dit proefschrift wordt het onderzoek naar de veerkracht van botnets daarom in twee delen behandeld.

In het eerste deel, behandeld in hoofdstuk 5, bespreken we de werking van 23 malware downloaders. Deze malware downloaders installeren voortdurend massaal malafide programma's op al geïnfecteerde PC systemen. Om deze malware installatie infrastructuren te beschermen combineren aanvallers zowel technische als organisatorische maatregelen. Technische maatregelen zijn het coderen en verborgen houden van C&C-kanalen, maar vaak scheiden aanvallers ook de C&C-infrastructuur van de malware installatie infrastructuren. Naast technische maatregelen zorgen malware downloaders ervoor dat hun C&C hosting providers en C&C domein registrars systematisch veranderen. Het toepassen van deze technieken door malware downloaders is een van de redenen dat botnets zo succesvol blijven. Wij stellen twee nieuwe technieken voor die gebruik maken van de publieke aard van de malware installatie infrastructuren om automatisch malware samples te verzamelen voor verdere analyse.

In hoofdstuk 6 laten we zien dat botnet architecturen zeer veerkrachtig kunnen zijn. In het bijzonder behandelen wij de huidige trend in de ontwikkeling van peer-to-peer (P2P) botnets, die speciaal zijn ontworpen om zeer veerkrachtig te zijn. We vergelijken het herstelvermogen van zes bestaande P2P botnets met historische P2P botnets als deze doelbewust worden verstoord. Om de veerkracht van deze P2P botnets te testen hebben wij prototypes ontwikkeld die de botnets verstoren. Deze prototypes hebben geholpen om sinkholing operaties voor te bereiden tegen succesvolle P2P botnets zoals Zeus en ZeroAccess. Uit onderzoek, met gebruik van deze prototypes, blijkt dat deze trend van zeer veerkrachtige P2P botnets leidt tot kwaadaardige netwerken die niet meer gemakkelijk kunnen worden verstoord. Botmasters gebruiken bijvoorbeeld reputatie regelingen of implementeren P2P protocollen met zelfherstellende peerlists wat sinkholing pogingen beperkt. Daarnaast vallen P2P botnets terug op secundaire C&C back-up kanalen bij het falen van hun P2P C&C component. Uit ons onderzoek blijkt dat met slechts kleine aanpassingen van de P2P-protocollen dergelijke netwerken zeer veerkrachtig worden. Als gevolg hiervan verwachten wij een opkomst van P2P botnets in de nabije toekomst.

Wij hebben met name technieken onderzocht die botmasters gebruiken om succesvol hun botnets operationeel te houden voor vele jaren. Onze waarnemingen impliceren dat botnet ontmantel strategien veel verder moeten gaan dan het buitenwerking stellen van enkele C&C servers of C&C domeinen. Zulke initiatieven leiden alleen tot tijdelijke verstoring van een botnet en botmasters zullen gebruik maken van hun resterende infrastructuur om de botnets weer operationeel te maken. Deze veerkracht van botnets wordt mede bepaald door relatief eenvoudig organisatorische maatregelingen van de botmasters. Bijvoorbeeld, als C&C eindpunten worden gehost op meerdere plaatsen, dan moeten meerdere partijen in verschillende tijdzones en wetgevingen samenwerken om de botnet te ontmantelen. Onze analyses geven inzicht in de algehele veerkracht van botnets. Dergelijke analyses zijn bijvoorbeeld nuttig bij toekomstige pogingen om botnets te ontmantelen. Onze discussie maakt mensen bewust van de veerkracht van botnets en stimuleert het onderzoek naar alternatieve methoden om deze kwaadaardige netwerken te ontmantelen.

List of Figures

List of Tables