Measurement and Analysis of Autonomous Spreading Malware in a University Environment

Jan Goebel
RWTH Aachen University
Center for Computing and Communication

Thorsten Holz and Carsten Willems
University of Mannheim
Laboratory for Dependable Distributed Systems

Abstract. Autonomous spreading malware in the form of bots or worms is a constant threat in today's Internet. In the form of *botnets*, networks of compromised machines that can be remotely controlled by an attacker, malware can cause lots of harm. In this paper, we present a measurement setup to study the spreading and prevalence of malware that propagates autonomously. We present the results when observing about 16,000 IPs within a university environment for a period of eight weeks. We collected information about 13,4 million successful exploits and study the system- and network-level behavior of the collected 2,034 valid, unique malware binaries.

Keywords: Honeypots, Malware, Invasive Software

1 Introduction

In the recent years, we see a shift in how attackers behave and how they try to compromise systems: large-scale worms which compromise tens or even hundreds of thousands of machines are rare now, mass-outbreaks like Code Red or Slammer did not appear in the last few years. This could have two main reasons. On the one hand, a worm does not offer the attacker any type of remote control. Once the worm is released and spreading in the wild, the attacker can not send any additional commands to the compromised machines or otherwise influence its behavior. On the other hand, the attacker does not have any *financial* advantages by releasing a worm. We observe more and more a change in the motivation behind attacks in cyberspace. While ten years ago most attacks were motivated by technical challenges or to prove certain vulnerabilities, today most attacks have a financial background, a real "underground economy" has developed [3].

One of the main problems in today's Internet are *botnets*. A botnet can be defined as a network of compromised machines which can be remotely controlled by an attacker. On every compromised machine a so called *bot* is installed which establishes a connection to a remote control network by which the attacker can issue arbitrary commands. Typical examples for these remote control networks are IRC networks or HTTP servers, but there have also been the first Peer-to-Peer based botnets in the last few years [4]. Botnets can be used by an attacker for many malicious activities: carrying out

Distributed Denial-of-Service (DDoS) attacks, sending out millions of spam e-mails, stealing sensitive information from the compromised machines, or seeding new malware are just a few examples. With most of these activities, an attacker can also have financial advantages. A detailed introduction to the topic of botnets is given in a paper by the Honeynet Project [16].

It is hard to measure the extent of this problem and a scientific based measurement of the number of compromised machines on the Internet is missing. There are some guesses by different people which greatly vary in size. For example, Vint Cerf estimates that up to a quarter of the 600 million machines connected to the Internet may be used by cyber criminals in botnets [10]. Other popular estimates range between 12 million [19] and 70 million [17] machines infected by bots. Moreover, it remains unclear with which metrics the size of a botnet should be measured [13].

One of the side effects of botnets is the constant "background noise" in the Internet caused by propagation attempts of bots: one of the most common commands issued by the controller of a botnet is the command to scan for other vulnerable machines in order to exploit a vulnerability found and then infect that machine. Since currently hundreds or even thousands of different bot variants are propagating in the wild, they cause a measurable amount of malicious network traffic which we can study with different techniques and tools. Another source of the constant malicious network traffic we see in the Internet is caused by different kinds of *worms* which spread even years after their first release. Worms are similar compared to bots: they also try to exploit vulnerabilities on other machines and propagate further autonomously, but lack the remote control facility. We can study both kinds of malware utilizing the same methods. Other kinds of malware like for example *spyware*, *rootkits*, or *Trojan Horses* typically do not have the ability to propagate autonomously, thus their prevalence is out of scope of our study.

In this paper, we present the results from studying autonomous spreading malware during an eight week period which lasted from December 2006 to January 2007. We studied the malware prevalence within a typical university environment, in this case the network of RWTH Aachen university, which consists of three class B networks. The study consists of several steps. In the first step, we capture a binary copy of the malware that tries to exploit a vulnerability on our sensor: we use the tool *nepenthes* to simulate 21 different vulnerabilities which are commonly exploited in the wild. Additional information collected by nepenthes allow us to study the extend of malware within our analysis environment. In the second step, we perform a system-level analysis of all collected malware binaries. We study typical system-level changes applied to infected systems and common network-level behavior of compromised machines. Furthermore, we study the reaction times of common antivirus engines, which are supposed to protect end-users against this threat. Moreover, we study the remote control networks used by bots we captured.

This paper is outlined as follows: Section 2 provides an overview of related work which studies different kinds of autonomous spreading malware. In Section 3 we describe our measurement setup and give a brief background on the tools and methods used during our study. We present the analysis results in Section 4, where we focus on four different aspects of autonomous spreading malware. Finally, we give an overview of future work and conclude the paper in Section 5.

2 Related Work

A study similar to our own was conducted by Yegneswaran et al. [21]. They analyze firewall logs collected over a four month period from over 1,600 different networks world wide by contributors of the *Internet Storm Center*. The authors study the distribution, categorization, and prevalence of exploitation attempts. The collected data shows both a large quantity and wide variety of exploitation attempts on a daily basis: the authors estimate that up to 25 billion scan attempts are performed on the whole Internet every day. Furthermore, they observe that worms like Code Red or Nimda still propagate long after their original release. Compared to our study, their study is based on a condensed summary of portscan activity obtained from various firewall and IDS solutions. They do not collect high-level information like for example the malware binary that causes the attack. In our study, we also incorporate this information and thus can study the underlying methods used by the attackers as well.

A study by Moore et al. measures the victims of one particular instance of an autonomous spreading malware [8]. They study the behavior of Code Red and conclude that 359,000 computers became infected with the Code Red worm in less than 14 hours. In a similar study, Moore et al. also analyze the effects of the Slammer worm and concludes that about 75,000 hosts were infected by this malware specimen [7]. These studies only focus on one single instance of autonomous spreading malware, whereas our analysis focuses on an overview of the overall network activity caused by this kind of malware. In total, we study the effects caused by more than 2,000 unique malware binaries collected during the analysis period of eight weeks. A binary is considered unique in this context if it has a different MD5 sum.

Saroiu et al. study the amount and distribution of spyware in a university environment [14]. They analyze four common types of spyware and derive network signatures to detect the presence of this kind of malware. By analyzing a week-long passive trace of network activity, they show that at least 5.1% of active hosts within the campus network were infected with spyware, and that many computers tend to have more than one spyware program running at the same time. A broader study is performed by Moshchuk et al. in a crawler-based study of spyware on the World Wide Web [9]. Similar to HoneyMonkey [18], they crawl large parts of the Web and analyze executables and Web pages for malicious content. During a crawl in May 2005, in which they examined about 18 million URLs, they found executable files in approximately 19% of the crawled Web sites and spyware-infected executables in about 4% of the sites. Moreover, they could identify spyware in 13.4% of the 21,200 executables downloaded. Besides spyware, they are also interested in so called *drive-by* download content, i.e., Web pages that exploit a vulnerability in the visiting browser to install a piece of malware on the victim's machine. By crawling 45,000 URLs from 1,353 domains in October 2005 with an instrumented Web browser instance, they found drive-by download attempts in 0.4% of all URLs examined and drive-by attacks that exploit browser vulnerabilities in 0.2% of the examined URLs. Both studies focus on spyware, a specific kind of malware. By default, spyware does not have the capability to autonomously propagate further. In our study, we focus on malware that has the capability to *autonomously* propagate further. We also examine the additional steps performed by an attacker after he compromised the victim's system by observing the remote control network used by common bots.

There are two studies that measure the prevalence of malware in Peer-to-Peer (P2) networks. Kalafut et al. study malware in the P2P networks Limewire and OpenFT [5], whereas Shin et al. perform a similar study for the KaZaA file-sharing network [15]. In the study by Kalafut et al., they collected data for more than one month and could show that 68% of all downloadable responses in Limewire, which are either executables or archives, contain malware. However, this is caused by only a small amount of distinct malware: the top three most prevalent malware account for 99% of all the malicious responses. In contrast to this, only about 3% of the executables or archives found in OpenFT contained some kind of malware. Again, this is caused by a small number of distinct malware samples: the top three most prevalent malware account for 75% of all the malicious responses. The study by Shin et al. results in some similar conclusions: using a light-weight crawler built for KaZaA, they gathered information about more than 500,000 files returned in response to 24 common query strings. These files were examined with 364 signatures of known malicious programs, and they found that over 15% of the crawled files were infected by 52 different viruses. In contrast to these studies, we study *active* propagation by autonomous spreading malware: the malware binaries we are interested in actively search for vulnerable machines and exploit these vulnerabilities to infect the victim. Propagation via P2P networks is just *passive*: the malware binary copies itself to the shared folder of popular P2P programs and uses promising file names in order to trick a victim to open the malicious binary. No vulnerabilities are exploited, but *social engineering* is used by these malware binaries to propagate further.

Rajab et al. use a similar measurement setup to study botnets [12]. They also use nepenthes to collect malware binaries, but their malware analysis approach is significantly different from ours: they use graybox testing to extract only the network fingerprint of the binary and in a second phase they extract IRC-specific information. In contrast to this, we study the *system-level behavior* of the binary by closely monitoring its system activity. This allows us to analyze the collected binaries in more detail. Similar to our study, they also track the remote control facility used by botnets. Rajab et al. just track IRC-based botnets, whereas we also observe botnets that use other protocols for remote control. In addition, we also study other aspects of autonomous spreading malware, e.g., the reaction time of antivirus software or the behavior of malware other than bots.

3 Measurement Setup

In this section, we describe the setup of our study and describe the individual building blocks for the measurement. We use a network-based approach based on *nepenthes* [1] to collect malware binaries. Nepenthes is a *low-interaction honeypot* which aims at capturing malicious software artifacts that spread in an automated manner, like for example worms or bots. The main focus of this application is to get hold of the malware itself, i.e., to download and store a copy of the malware binary itself for further in-depth analysis. Unlike other low-interaction honeypots, nepenthes does not emulate full services for an attacker to interact with: it offers only as much interaction as is needed to exploit a vulnerability. For this reason, nepenthes is not designed for any human interaction, as the trap would be easily detected. On the contrary, for an automated attack, just a

few general conditions have to be fulfilled, consequently, maximizing the effectiveness of this approach. These conditions usually include to display the correct banner information of an emulated service, as well as, some simulated commands. Therefore, the resulting service is only partly implemented.

In total, we use 21 different vulnerability modules, corresponding to commonly exploited network services. This is the default setup of nepenthes [1]. These modules provide a baseline for an estimation of the actual amount of autonomous spreading malware in the university network: they are common exploits being used by malware. However, this setup is not complete. There are other methods used by malware to propagate further, e.g., other exploits not emulated by nepenthes or other propagation strategies like e-mail. Therefore, we can only give a lower bound for the amount of autonomous spreading malware in our environment.

The usage of honeypots allows us to carry out a study like this without privacy issues: since the honeypot is just a network decoy which should not receive any network connections at all, any interaction with the honeypot is malicious by definition. We emulate vulnerable network services and only automated threats will successfully compromise our honeypot: a human attacker can easily spot the emulation.

In order to collect as many malware binaries as possible, we assign many IP addresses to the machine running nepenthes. This way, we can on the one hand collect autonomous spreading malware that sequentially exploits other systems. On the other hand, we also have a better chance to collect malware that tries to propagate further by randomly targeting other systems. The network of RWTH Aachen university, our analysis testbed, consists of three class B networks. We assigned more than 16.000 IP addresses from the whole network range to the sensor used in our study.

The sensor itself is running in a virtual machine based on Xen [2] on a Quad-CPU Pentium Xeon system with two virtual CPUs at 2,6 Ghz speed and 1 GB RAM assigned. The operating system used is Debian Linux with MySQL 5 as database software to store the collected information. During the measurement period, we used nepenthes in version 0.2.0 and did not perform any updates, in order to have a constant setup during the whole period. The average load is slightly above 1, depending on whether the antivirus engines are currently scanning or not. This means that the system is using most of its resources to emulate vulnerabilities and download malware binaries. Nepenthes itself uses about 80% of one CPU and an additional 2% to 4% are consumed by the database. A more detailed study on the scalability of nepenthes is given in the paper on nepenthes [1].

In addition to the default nepenthes installation, we are running a few customized modifications to gather more detailed and statistical information on the autonomous spreading malware collected. To provide a common platform for the different analysis tools in use, we have developed a custom logging module for nepenthes, which stores all gathered information in a local MySQL database. Among these information are the IP addresses of the hostile hosts, the vulnerability modules which were triggered, the download location of the malware, as well as, the binary itself. The database is used as the basis for all other analysis utilities involved.

One of these analysis utilities is *CWSandbox* [20]. CWSandbox is a tool for *automatic behavior analysis* of malware. In contrast to the traditional approach of code

analysis, the malware is viewed as a black box and only its behavior during execution is examined. This eliminates all difficulties and disadvantages of code analysis, as aspects like encryption, packing, and code obfuscation are no more relevant: the binary will decrypt and/or unpack itself and we can observe its behavior during runtime. The main disadvantage of this approach is that only one possible execution path is monitored for each execution and, therefore, analysis reports may be incomplete as not all operations of the monitored process are performed. However, our experience with CWSandbox shows that the generated analysis reports contain commonly enough details to get an overview of what a given malware binary intended to do.

In order to understand the proceeding of CWSandbox, we give a brief overview of the tool. For monitoring the behavior of a suspect file, it is executed in an instrumented Windows environment. All of its security-relevant activities are monitored, and a summarized and high-level report of the collected data is created afterwards. The monitoring, and to some degree also the controlling of the binary, is done by installing *hook functions* on several Windows API functions. Hook functions for instrumenting the following Windows objects, amongst others, exist:

- Filesystem
- Registry
- Processes and Threads
- Windows Service Applications
- Virtual Memory of running processes
- Mutexes
- Windows Shares
- COM objects
- Windows of running processes

Besides operations on these Windows objects, the following operations are hooked as well, in order to get a more complete overview of the behavior of a given binary:

- implicit loading of DLLs or retrieving of method addresses
- attempt to reboot the system
- sending of ICMP packets
- using the Winsock library to establish TCP/IP connections
- retrieving system information like computer name, name of currently logged in user, ...

Each time the malware calls one of these hooked API functions, the call parameters are examined and stored into a log file. In most cases, the hook function then calls the original Windows API function, as it would be called directly by the malware. When returning from this API, the result code is stored as well and then control is delegated back to the supervised process. This whole re-routed control flow is transparent to the malware, i.e., it should *behave* normally, as if it would be running in an uncontrolled environment. There are some exceptions to the handling of the control flow, where either the call parameters of the original API function or its return result are modified by the hook function, or where the API is not called at all. Examples for this are API function that can be used to detect the presence of CWSandbox, or particular network

functions, which can be restricted in several ways. For example, we only allow a certain number of outgoing TCP connections and block certain TCP ports completely in order to mitigate the risk involved during analysis.

One focus of the information extracted from an analysis run lies on the detection and recognition of network connections and the extraction of the relevant transmitted data. Therefore, CWSandbox maintains virtual connection objects for each separate network connection, analyzes the corresponding traffic data, and tries to determine the underlying protocol. In the current version, which was developed for this study, the protocols HTTP, FTP, IRC, SMTP, and IDENT can be detected independent of the used network port. If one of these protocols is detected, all relevant protocol-dependent data like the utilized username, password, or channel name is extracted and displayed in the analysis report.

As an additional analysis utility, we use four different antivirus scanners to check each of the downloaded binaries every hour for known malware. The resulting reports are also stored in the database. Prior to each scan, the scanners update their virus signatures, thus the results always reflect the latest available signature version. To keep track of changes in malware detection, each binary is always scanned with every antivirus engine. In case the output of a scanner varies from previous results, an additional entry for the affected binary is stored in the database. Thus, for each binary it is possible to determine whether it was detected by a scanner in the first place, how long it took until a new signature version detects the malware, and if a signature update modifies the name of the given malware sample. Therefore, we have some kind of timeline, indicating when a new virus was first recognized by which scanner or if a file, previously identified as virus **A**, is classified as virus **B** after a signature update occurred. The four antivirus scanners which are currently in use are *Avira AntiVir*, *BitDefender AntiVirus*, *Sophos Anti-Virus*, and *Clam AntiVirus*.

As the third analysis utility, we used the tool *botspy* [11]: if the binary collected by nepenthes is a bot or another program which offers a remote control facility, we also want to study this. Botspy is designed to be able to analyze the reports generated by CWSandbox and, if applicable, to track the remote control facility. This is for example accomplished by checking the network traffic section of the analysis report for outgoing IRC connections. Other information extracted from the report include the botnet server address (the so called *Command & Control* (C&C) server), the nickname, the user string, the channel name, and any passwords involved. With this information, botspy is able to connect to the IRC-based C&C server of the botnet and keep track of all instructions issued by the botnet herder. If the remote control facility is not based on IRC, but uses for example HTTP as the control mechanism, botspy periodically downloads the HTTP URL extracted by CWSandbox. Certain kinds of autonomous spreading malware like for example worms do not offer a remote control. In this case, botspy does not do anything. Botspy is designed to be able to track several hundred remote control facilities in parallel and also stores all collected information in a central database.

The whole measurement setup is depicted in Figure 1: with the help of nepenthes, we collect samples of autonomous spreading malware. These samples are then analyzed with the help of CWSandbox which results in a behavior based analysis report. In addition, several antivirus engines periodically check all collected samples and col-

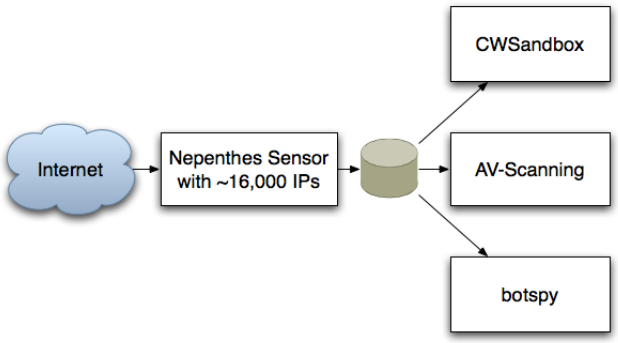

Fig. 1. Schematic Overview of Measurement Setup

lect information about the detection and analysis rates of common antivirus software. Finally, we track the remote control facility of malware (if applicable) with the help of botspy. Thus we collect information about the infrastructure behind all this autonomous spreading malware as well.

Limitation With the measurement setup outlined in this section, we are able to collect autonomous spreading malware with certain characteristics. First, the malware propagates further by exploiting common vulnerabilities on other systems. We only collect the malware binaries that actually exploit the vulnerabilities we emulate, thus we may miss certain samples. Secondly, we can only study malware that propagates autonomously, thus we miss malware that uses other techniques for propagation. While our measurement setup does not cover the whole range of malware out there, we nevertheless can study typical malware propagation from bots and worms.

4 Analysis of Autonomous Spreading Malware

In this section, we present measurements and analysis results of autonomous spreading malware activity within the network of RWTH Aachen University, Germany. With more than 40,000 computer-using people to support, this network offers us a testbed to study the effects of bots and worms. Our analysis is based on eight weeks of measurement, which took place during December 2006 and January 2007.

The network of RWTH Aachen university consists of three Class B network blocks (three /16 networks in CIDR notation). As noted in the previous section, the nepenthes sensor listens on about 16,000 IP addresses spread all across the network. Most of the IP addresses are grouped in a large block in one Class B network, but we have also taken care of evenly distributing smaller blocks of the sensor IPs all across the network to have a more distributed setup. This is achieved by routing smaller network blocks to the machine on which nepenthes emulates the vulnerabilities. We do not need to install additional software on end-hosts, but use a purely network-based approach.

4.1 Network-based Analysis Results

With the help of the MySQL logging module, we can keep track of all connections which were established to the sensor. That includes the attacker's IP address, the target IP address and port, as well as, the vulnerability module which was triggered. More than 50 million TCP connections were established during the measurement period. Since the nepenthes sensor is a *honeypot* and has no real value in the network, it should not receive any network connections at all. Thus, the vast majority of these 50 million network connections have a malicious source. On average, more than 900,000 TCP connections were established to the nepenthes sensor per day and about 240,000 known exploits were performed every single day. Thus nepenthes recognized about 27% of all incoming connection attempts and could respond with a correct reply.

The remaining 73% of network connections are mainly caused by scanning attempts: about 75% of these connections target TCP port 80 and search for common vulnerable web applications. An additional 22% contain probe requests used by attackers during the reconnaissance phase in order to identify the network service running on a given target. About 3% of network connections contain a payload that nepenthes could not understand. By manually adding support for these missed exploitation attempts, nepenthes could be enhanced. With approaches like ScriptGen [6], the detection rates could be automatically improved in the future.

A total of more than 13,400,000 times the sensor system was *hit*. This means that so many times a TCP connection was established, the vulnerability emulation was successful, and a malware binary could be downloaded by the sensor. If one host scans linearly the whole measurement range, then we count each of these connections as a separate one. Since this skews the data, we take only the unique IP addresses into account. Roughly 18,340 unique IP addresses caused this malicious network traffic. Table 1 depicts the sanitized IP addresses of the ten most active attackers together with the according country. As you can see, a small number of IP addresses are responsible for a signification amount of malicious network traffic.

IP Address:	**Country:**	**Hits:**
XXX.178.35.36	Serbia and Montenegro	216.790
XXX.211.83.142	Turkey	156.029
XXX.7.116.4	France	108.013
XXX.147.192.47	United States	107.381
XXX.92.35.23	Norway	94.974
XXX.206.128.27	United States	91.148
XXX.12.234.94	Japan	91.051
XXX.255.1.194	United States	78.455
XXX.92.35.24	Norway	78.439
XXX.29.103.225	United States	77.580

Table 1. Top Attacking Hosts with Country of Origin

An analysis revealed that the distribution of attacking hosts follows a classical long-tail distribution as depicted in Figure 2. About 9,150 IPs, corresponding to about 50%

of the total number of IPs observed, contacted the sensor system less then five times. These IPs are presumably infected with some kind of autonomous spreading malware which propagates further by scanning randomly for other victims.

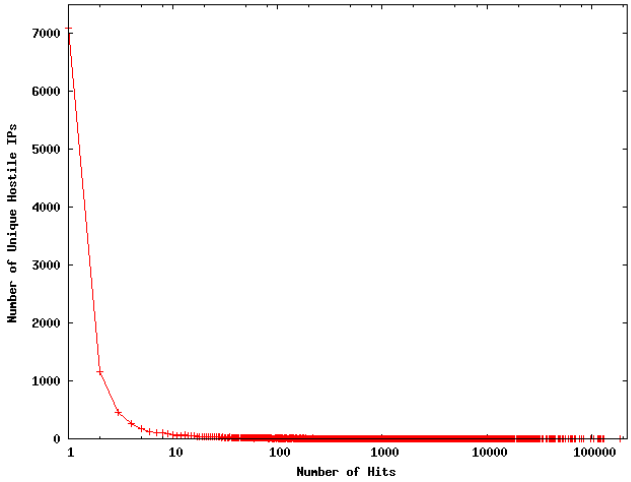

Fig. 2. Distribution of Attacking Hosts

The 18,340 unique IP addresses we monitored during the analysis period connected to different TCP ports on the sensor. The distribution of target ports is very biased, with more than 97% targeting TCP port 445. This port is commonly used by autonomous spreading malware that exploits vulnerabilities on Windows-based systems. Table 2 gives an overview of the distribution.

TCP Port Number	Number
445	57,015,106
135	184,695
3127	23,076
80	20,746
42	18,653
139	15,112
1023	14,709
5554	13,880
6129	27
1025	1

Table 2. Top Ten TCP Ports Used by Autonomous Spreading Malware

Closely related to the distribution of target TCP ports is the type of vulnerabilities exploited. This distribution is also dominated by the most common vulnerability on TCP port 445: the `Lsasrv.dll` vulnerability, commonly referred to as *LSASS*. Table 3 gives an overview of the vulnerability modules triggered and we see a heavy bias towards the Windows vulnerabilities related to network shares.

Dialogue	Number
LSASSDialogue	56,652,250
PNPDialogue	361,172
DCOMDialogue	184,696
SasserFTPDDialogue	28,589
MydoomDialogue	23,076
IISDialogue	20,746
WINSDialogue	18,655
NETDDEDialogue	15,112
SMBDialogue	2,341
DWDialogue	27

Table 3. Top Ten Vulnerabilities Detected by Nepenthes

The 13,4 million downloaded binaries turned out to be 2,558 unique samples. The uniqueness is determined by the MD5 hash of each binary: two binaries that have the same MD5 hash are considered to be the same binary. This is not foolproof due to the recent attacks on MD5, but so far we have no evidence that the attacking community has released different binaries with the same MD5 hash. On the other hand, this is also no strong indicator for uniqueness: if the malware binary is polymorphic, i.e., it changes with each iteration, we collect many samples which in fact are very similar. In the middle of December 2006, such a polymorphic bot was released in the form of *All.aple* worm. Unfortunately, we missed this particular worm since nepenthes could not analyze the payload send by this worm. In Section 4.2 we show preliminary results on how we can identify similar malware binaries based on behavior.

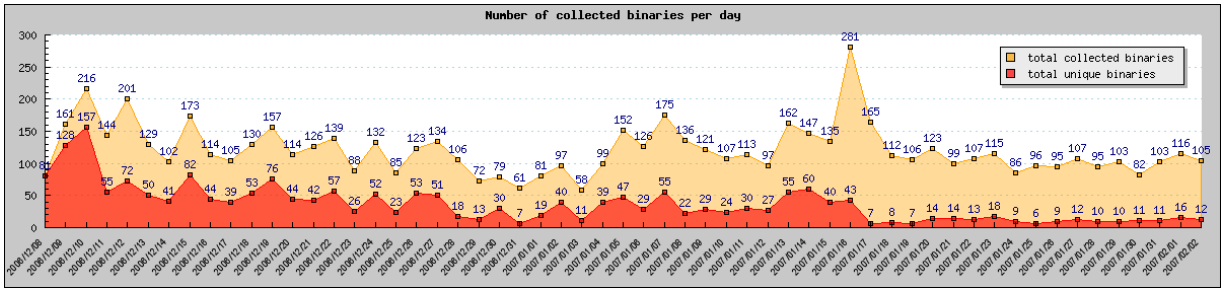

Fig. 3. Chronological Analysis of Collected Malware Binaries

The number of collected samples result in an average of one unique malware binary every 5,240 hits. Considering the number of successful exploits per day, this results in almost 46 new binaries every 24 hours. Figure 3 gives an overview of the chronological sequence for the number of collected binaries and number of unique binaries. The number of collected binary varies from day to day, ranging between 58 and 281. There are several spikes in this measurement, but no reason for these anomalies could be identified. The situation is slightly different for the number of unique binaries: in the first few days, the number of unique binaries collected per day is high, whereas this number drops after about six weeks. It seems like there is some kind of *saturation*: in the beginning, the number of unique binaries is significantly higher then in the end of the measurement period. After a certain period of time we have collected the commonly propagating malware in the measurement network and only a few new binaries are collected per day. This number varies between 6 and 16, presumably corresponding to new malware binaries released by attackers.

4.2 CWSandbox Analysis Results

In this section, we present some quantitative statistics about the analysis results of our malware collection. It should be mentioned that our collection cannot give a representative overview of current malware on the whole Internet, as on the one hand, the sample size is not large enough and, on the other hand, it contains *only* autonomous spreading applications like bots and worms. However, for this particular subset of malicious activity, our measurement setup can give us an overview of the current threat level for a typical university environment.

From the overall collected 2,454 sample files, 2,034 could be analyzed correctly by CWSandbox, 1 failed due to a crash and 419 were no valid Win32 applications. This means that in roughly 17% of the collected samples, the resulting file was not valid. This can be explained by aborted transfers or disrupted network connectivity. One additional file of the remaining 2034 had a valid PE header, but could not be correctly initialized by the Windows Loader due to an ACCESS_VIOLATION_EXCEPTION. Each successful analysis resulted in an XML analysis report, which reflects all the security-relevant operations performed by the particular file. As we are not interested in a detailed malware analysis for single file instance in this paper, we present quantitative results extracted from the 2034 valid reports. The main focus of our statistics lies on network activities, but a few other important results are presented as well.

1,993 of the 2,034 valid malware samples tried to establish some form of TCP/IP connection, either outgoing, incoming (i.e., listening connections) or both. Besides DNS requests, we have not found any single malware in our set that uses only UDP directly. 1,216 binaries were successful in the attempt to setup an outgoing TCP connection. For all the others, the remote host was not reachable or refused the connection for some other reason. Altogether, 873 different TCP remote ports have been used for outbound connection attempts, and Table 4 shows the top ten of them. Although CWSandbox is able to recognize the content of a TCP connection and infer the application protocol used, these results are not shown in this document for complexity reasons. However, it is highly probable that most connections on port 445 and 139 are aiming on further malware propagation, port 80 and 443 are used for HTTP(s) connections, 6667,

Remote TCP port	# samples
445	1312
80	821
139	582
3127	527
6667	403
6659	346
65520	143
7000	30
8888	28
443	16

Table 4. Top Ten Outgoing TCP Ports Used

66520, 7000, 6659, and 8888 are used for IRC communication, and finally, 3127 is a backdoor of *MyDoom*.

Local TCP port	# samples
113	497
3067	122
80	9
5554	7
1023	6

Table 5. Top Five Listening TCP Ports Used

Furthermore, we have found 1,297 samples that install a TCP server for incoming connections, most of them setting up an IDENT server on port 113 for supporting IRC connections. In Table 5, the top five listening local TCP ports are presented.

Since most bots rely on IRC communication, a deeper investigation of these connections is necessary. 505 samples could successfully establish a connection to an IRC server. Moreover, 352 files tried to send IRC commands over an *unestablished* connection. The reason for that is most probably bad software design. 349 of these files are variants of *GhostBot*, the other 3 are *Korgo* worms. Furthermore, we have 96 samples that try to connect to a remote host on TCP port 6667 or 7000 and fail. Adding these numbers, we have at least 953 files which try or are successful in setting up an IRC connection. The corresponding samples are most probably IRC bots. We cannot know, how many of these different binaries belong to the same bot variant or even to the same botnet. However, by taking the remote IP address, remote TCP port, IRC channel name and IRC channel password into account, we can give some estimations. This is a good indication for uniqueness: if a given binary uses the same tuple of network parameters, we can be sure that it is the same variant, although the MD5 sum of these binaries is different. Of all established IRC connections, 64 different *host:port:channel:channelpassword*-combinations have been used. As the host IP for a botnet may change, we generalize the results to the different *channel:channelpassword*-

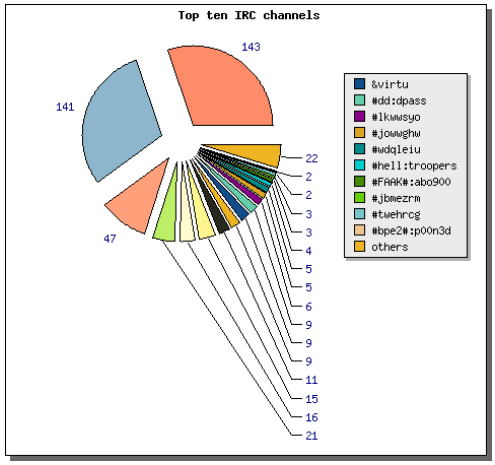

Fig. 4. Distribution of IRC Channel/Password Combinations

combinations and assume that each of those represent a different botnet or at least a different bot family. By generalizing the number of different combinations decreases down to 41. The most common channels are *&virtu (no password)* and *dd (password "dpass")* with 143 and 141 samples, respectively. These samples have a different MD5 sum, but based on their network behavior we argue that they are very similar. An overview of these results is given in Figure 4. Please note that all the combinations with only one corresponding malware sample are aggregated into *others*.

When looking at the different remote TCP ports which are used for establishing an IRC connection, we see that not only the default IRC port 6667 is used, but many more. It is interesting to see, that beside some probably random ports, a lot of well known ports of other web protocols are used, e.g., 80 (HTTP), 443 (HTTPS) or 1863 (MSN). This allows the bot to communicate through a firewall that is open for these standard protocols. Thus it is necessary to also closely observe these port when thinking about vulnerability assessment. Figure 5 shows a distribution diagram of the TCP ports observed.

As already mentioned above, a few other interesting, system-level related results can be drawn from our analysis reports as well. After infecting a new host, nearly all malware tries to install some auto-start mechanism, such that it is activated each time the infected system reboots. This is commonly done by adding some auto-start registry keys, but some malware install a *Windows Service Application* or even a *kernel driver*. By doing that, it is much harder to detect the presence of malware. Especially in the case of kernel drivers, the malware binaries can get higher security privileges on the local system. Table 6 shows that 21 of our collected files install a service application and 17 install a kernel driver. Since these binaries use the same servicename and filename for the given processes, we can learn that these were most probably installed by variants of the same malware family.

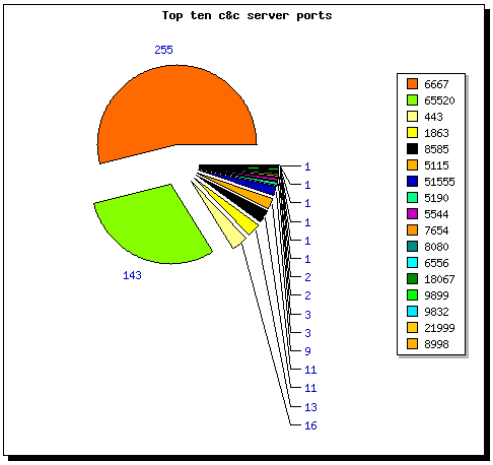

Fig. 5. TCP Ports Used for IRC Connections

The observed system-level behavior can also be used to classify malware samples: if two binaries with a different MD5 sum behave similarly, we can argue that these samples belong to the same family of malware. There are several features we can use for classification. One is the registry keys created during the analysis: 1,842 samples created a registry key of the form *HKLM\Software\Microsoft\CurrentVersion\Run* in order to have an auto-start mechanism. A closer analysis revealed that only 283 unique registry keys were created, taking into account randomly created names.

As a final statistic result, Table 7 shows a summary of the windows processes, into which the malware samples injected malicious code. It is a common approach to create a new thread (or modify an existing one) in an unsuspicious windows process, e.g., `explorer.exe` or `winlogon.exe`, and perform all malicious operations from that thread. Via this proceeding, the malware becomes more stealthy and, furthermore, circumvents local firewalls or other security software that allows network connections only for trusted applications.

4.3 Antivirus Engines Detection Rates

In order to evaluate the performance of current antivirus (AV) engines, we scanned all 2,034 binaries, which we had captured with nepenthes and successfully analyzed with CWSandbox, with four common antivirus engines. This helps us to estimate the detection rate for common autonomous spreading malware and also for vulnerability assessment. These binaries are currently spreading in the wild, exploited a vulnerability in our measurement system, and could be successfully captured. In contrast to common antivirus engine evaluation tests, which rely on artificial test sets, our test set represents malware successfully spreading in the wild.

Table 8 displays the current detection rates of each scanner with the latest signature version installed. This scan was performed one week after the measurement period, in

Servicename	**Filename** (base directory is C:\Windows\)	**Kernel driver**	**# samples**
SVKP	system32\SVKP.sys	x	15
DLLHOST32	system\dllhost.exe		8
WINHOST32	system\services.exe		2
Print Spooler	system32\spooler.exe		1
hwclock	system32\hwclock.exe		1
oreans32	system32\drivers\oreans32.sys	x	1
Windows System 32	services.exe	x	1
Windows Terminal Services	system32\vcmon.exe		1
Advanced Windows Tray	system32\vcmon.exe		1
Windows MSN	wmsnlivexp.exe		1
Windows Process Manager	system32\spoolsc.exe		1
mside	system\mside.exe		1
TCP Monitor Manager	system32\symon.exe		1
Client Disk Manager	system32\symon.exe		1
Monitor Disk Manager	system32\spoolcs.exe		1
System Restore Manager	system32\symon.exe		1

Table 6. Services and Kernel Drivers Installed by Collected Malware Samples

Injection target process (base directory is C:\Windows\)	**# samples**
explorer.exe	787
system32\winlogon.exe **and** explorer.exe	101
system32\winlogon.exe	74

Table 7. Injection Target Processes Observed for Collected Malware Samples

order to give AV vendors some additional time to develop signatures and incorporate them into their products. Nevertheless, none of the tools was able to detect all malicious files and classify them accordingly. The malware reports vary significantly from one tool to another. Figure 6 gives an overview of the detected malware binaries by the four different engines.

We focus on the ClamAV results in the following and analyze the different malware families more closely. ClamAV detected 137 different *malware variants* in the test set of 2,034 samples. In total, 27 different *families* of malware could be identified. Table 9 gives an overview of the top ten different malware variants. Two families of malware clearly dominate the result: *Padobot* and *Gobot* are the two main autonomous spreading malware families we could observe within our measurement environment.

AV software	**Absolute detection number**	**Relative detection rate**
AntiVir	2015	99.07%
ClamAV	1963	96.51%
BitDefender	1864	91.64%
Sophos	1790	88.00%

Table 8. Detection Rates for 2,034 Malware Binaries for Different AV Scanners

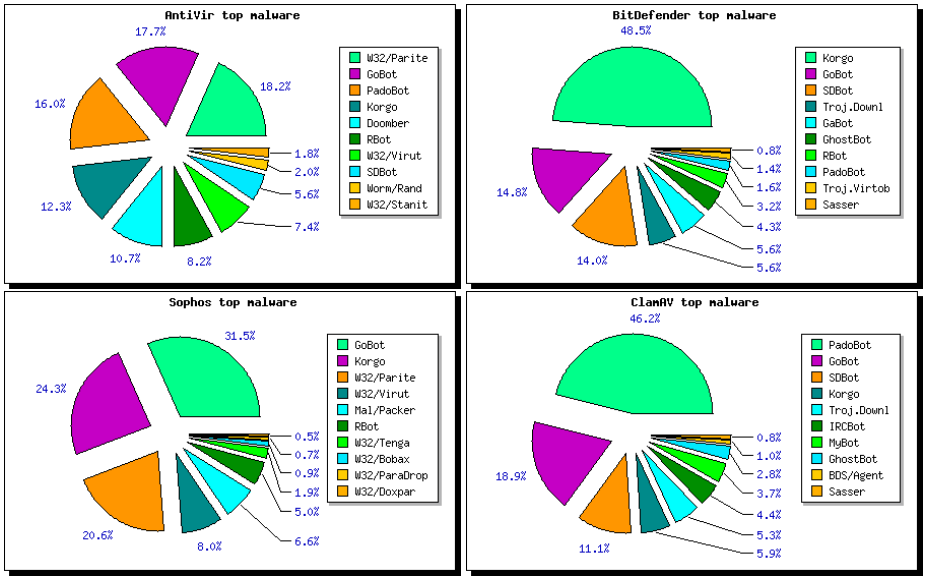

Fig. 6. Malware Variants Detected by Different Antivirus Engines

Besides these two families, also many other forms of autonomous spreading malware are currently spreading in the wild. Although Padobot and Gobot dominate the list of malware variants, the largest number of different variants was captured for *SdBot*: 35 different variants of the same family could be captured, whereas Padobot (14) and Gobot (8) had significantly less different variants.

Some of the malware binaries are already known for a long time. For example, the first variants of *Blaster* were observed in the wild in August 2003. More than three years later, we captured four different variants of Blaster which are still propagating in the Internet. Similarly, three different variants of *Sasser* were captured during the measurement period. We conclude that there are still systems on the Internet which are infected for a long time, helping "old" malware binaries to propagate further.

4.4 Botspy Analysis Results

With the help of botspy, we can study the remote control networks used by the autonomous spreading malware: in case we have captured a bot, it connects to this remote control network so that the attacker can send him commands. We just want to briefly present our results when tracking these botnets for a short amount of time.

In total, we could observe 40 different botnet Command & Control (C&C) server. Again, this behavior can be used to classify malware samples: if two binaries connect to the same C&C server, we can argue that they are similar despite the fact that they have a different MD5 sum. For example, 149 binaries connected to the IP address

Malware Variant	Number of Samples
Worm.Padobot.M	426
Worm.Padobot.P	274
Trojan.Gobot-3	118
Trojan.Gobot-4	106
Worm.Padobot.N	101
Trojan.Downloader.Delf-35	100
Trojan.IRCBot-16	76
Trojan.Gobot.A	61
Trojan.Ghostbot.A	53
Trojan.Gobot.T	37

Table 9. Top Ten Different Malware Variants

XXX.174.8.243 (`home.najd.us`). The system-level behavior of these samples is also very similar, so presumably these are just minor variants of the same malware family.

When connecting to the botnets, we could observe 33 different topics in the channel used to command the bots. The most common command used by the bot herder was related to propagation: most bots are instructed to scan for other vulnerable machines and exploit vulnerabilities on these systems. An example of such an instruction is `!asc dcom135 150 3 0 -r -b -s`, where the bots try to exploit the DCOM vulnerability on TCP port 135 and scan randomly (parameter `-r`) for vulnerable machines in their local class B (parameter `-b`) network with 150 threads in parallel. This is done for an unlimited amount of time (parameter `0`) and with a delay of three seconds (parameter `3`). More and more common are botnets that use non-standard IRC or encrypted communication mechanisms. For example, the command sent by the bot-herder to the bots could be an encrypted string. In total, we found 10 different botnets that use encrypted, IRC-based communication. Without a proper decryption routine, it is hard to study this kind of botnets. Currently it is unclear how we can efficiently study this kind of botnets.

Estimating the size of a given botnet is hard [13]. One possibility to estimate the size is to rely on the statistics reported by the C&C server upon connect: if the IRC server is not configured properly, it reports the number of connected clients. Moreover, we can query the server for the number of connected clients and various other status messages. Based on these numbers, the typical botnet size in our sample set varied between only a few hundred up to almost 10,000 bots.

Besides the IRC-based bots, we also found several samples of HTTP-based bots. These bots periodically query a given HTTP server and the response contains commands which are executed by the bot. Due to the rather stealthy communication channel of such bots, detection becomes harder. In addition, measuring the size of such a botnet is hard since we can only passively monitor the HTTP server. In total, we could identify three different botnets that use HTTP-based communication.

5 Conclusion and Future Work

In this paper, we have introduced a measurement method to study autonomous spreading malware in a university environment. We use several tools and techniques to capture

a malware binary and then analyze the sample in detail. Based on this information, we can derive an overview of the current situation of network attacks. Moreover, we can study the typical system-level behavior of malware binaries and their activity after a successful infection. Based on more than 13,4 million downloads and 2,000 malware binaries, we presented several statistics and patterns to point out the current prevalence of malware on the Internet.

Some of the tools used in this paper can also be used in other areas. As a future research, we examine how a behavior-based analysis can be used for malware analysis: the main problem in the area of automatic malware comparison and classification is that nowadays nearly all malware files are encrypted, compressed, or even use polymorphism to complicate their analysis and to fool signature based anti-virus software. Two versions of the very same malware may result in completely different binaries, when encrypted with different algorithms or encryption parameters. Therefore, one needs to decrypt and/or unpack the binaries before classification, which is not a trivial task. For that reason, the analysis reports created by CWSandbox are very helpful in comparing and classifying malware based on their *behavior* instead of their *code*. It is obvious that the behavior analysis of two different binary representations of the same application will result in the same, or at least very similar, reports. At the moment, we have no automated mechanisms to reduce CWSandbox analysis reports to such a high level of summarization, such that they can be compared directly. There are still a lot of side effects contained in the reports, that differ between several Windows operating systems or system environments. But in the next months we plan to build a system for behavior-based malware classification, in order to extend the results presented in this paper.

With our current method, we can not study malware that uses other propagation vectors like e-mails or P2P-based propagation. With different other honeypot solution like for example *client-side honeypots* or *crawler-based honeypots* it should be possible to fill this gap and also study other kinds of malware.

Acknowledgments

We would like to thank the anonymous reviewers and our shepherd Marc Dacier for helpful feedback on earlier versions of this paper.

References

1. Paul Baecher, Markus Koetter, Thorsten Holz, Maximillian Dornseif, and Felix C. Freiling. The nepenthes platform: An efficient approach to collect malware. In *Proceedings of 9th International Symposium (RAID'06)*, pages 165–184, 2006.
2. Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, 2003.
3. Team Cymru. The underground economy: Priceless. *;login:*, 31(6), 2007.
4. Julian B. Grizzard, Vikram Sharma, Chris Nunnery, Brent ByungHoon Kang, and David Dagon. Peer-to-peer botnets: Overview and case study. In *Proceedings of 1st Workshop on Hot Topics in Understanding Botnets (HotBots '07)*, 2007.

5. Andrew Kalafut, Abhinav Acharya, and Minaxi Gupta. A study of malware in peer-to-peer networks. In *IMC '06: Proceedings of the 6th ACM SIGCOMM Internet Measurement Conference*, pages 327–332, 2006.
6. Corrado Leita, Marc Dacier, and Frédéric Massicotte. Automatic handling of protocol dependencies and reaction to 0-day attacks with scriptgen based honeypots. In *Proceedings of 9th International Symposium (RAID'06)*, pages 185–205, 2006.
7. David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford, and Nicholas Weaver. Inside the slammer worm. *IEEE Security and Privacy*, 1(4):33–39, 2003.
8. David Moore, Colleen Shannon, and k claffy. Code-red: a case study on the spread and victims of an internet worm. In *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 273–284, New York, NY, USA, 2002. ACM Press.
9. Alex Moshchuk, Tanya Bragin, Steven D. Gribble, and Henry M. Levy. A crawler-based study of spyware in the web. In *Proceedings of 13th Network and Distributed System Security Symposium (NDSS'06)*, 2006.
10. BBC News. Criminals 'may overwhelm the web'. Internet: `http://news.bbc.co.uk/1/hi/business/6298641.stm`, Accessed February 2007, February 2007.
11. Claus R. F. Overbeck. Efficient Observation of Botnets. Master's thesis, RWTH Aachen University, May 2007.
12. Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose, and Andreas Terzis. A multifaceted approach to understanding the botnet phenomenon. In *IMC '06: Proceedings of the 6th ACM SIGCOMM Internet Measurement Conference*, pages 41–52, 2006.
13. Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose, and Andreas Terzis. My botnet is bigger than yours (maybe, better than yours): Why size estimates remain challenging. In *Proceedings of 1st Workshop on Hot Topics in Understanding Botnets (HotBots '07)*, 2007.
14. Stefan Saroiu, Steven D. Gribble, and Henry M. Levy. Measurement and analysis of spyware in a university environment. In *Proceedings of Networked Systems Design and Implementation (NDSI'04)*, San Francisco, California, United States, 2004.
15. Seungwon Shin, Jaeyeon Jung, and Hari Balakrishnan. Malware prevalence in the kazaa file-sharing network. In *IMC '06: Proceedings of the 6th ACM SIGCOMM Internet Measurement Conference*, pages 333–338, 2006.
16. The Honeynet Project. Know Your Enemy: Tracking Botnets, March 2005. `http://www.honeynet.org/papers/bots/`.
17. New York Times. Attack of the zombie computers is growing threat. Internet: `http://www.nytimes.com/2007/01/07/technology/07net.html`, Accessed February 2007, January 2007.
18. Yi-Min Wang, Doug Beck, Xuxian Jiang, Roussi Roussev, Chad Verbowski, Shuo Chen, and Samuel T. King. Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities. In *Proceedings of 13th Network and Distributed System Security Symposium (NDSS'06)*, 2006.
19. McAfee Whitepaper. Adware and spyware: Unraveling the financial web. Internet: `http://www.mcafee.com/us/local_content/white_papers/threat_center/wp_adware.pdf`, Accessed February 2007, August 2006.
20. Carsten Willems, Thorsten Holz, and Felix Freiling. CWSandbox: Towards automated dynamic binary analysis. *IEEE Security and Privacy*, 5(2), 2007.
21. Vinod Yegneswaran, Paul Barford, and Johannes Ullrich. Internet intrusions: Global characteristics and prevalence. In *SIGMETRICS '03: Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 138–147, New York, NY, USA, 2003. ACM Press.