

# Detecting Malware’s Failover C&C Strategies with SQUEEZE

Matthias Neugschwandtner<sup>1</sup>, Paolo Milani Comparetti<sup>1</sup>, and Christian Platzer<sup>1</sup>

<sup>1</sup>Vienna University of Technology, {mneug,pmilani,cplatzer}@seclab.tuwien.ac.at

## ABSTRACT

The ability to remote-control infected PCs is a fundamental component of modern malware campaigns. At the same time, the command and control (C&C) infrastructure that provides this capability is an attractive target for mitigation. In recent years, more or less successful takedown operations have been conducted against botnets employing both client-server and peer-to-peer C&C architectures. To improve their robustness against such disruptions of their illegal business, botnet operators routinely deploy redundant C&C infrastructure and implement failover C&C strategies.

In this paper, we propose techniques based on multi-path exploration [1] to discover how malware behaves when faced with the simulated take-down of some of the network endpoints it communicates with. We implement these techniques in a tool called SQUEEZE, and show that it allows us to detect backup C&C servers, increasing the coverage of an automatically generated C&C blacklist by 19.7%, and can trigger domain generation algorithms that malware implements for disaster-recovery.

## 1 Introduction

Malicious code, also known as malware, is an essential component of criminal activity on the internet. Miscreants use a variety of strategies to infect computers with malware and organize them into networks of remote-controlled *bots*. These *botnets* can then be used for a variety of harmful activities. These include identity theft (such as stealing a user’s credit card number or online banking credentials), sending out unwanted email (SPAM), performing distributed denial of service attacks (DDoS), tricking the user into purchasing fake anti-virus products, or generating advertisement revenue by producing fake “clicks” on advertisement links. In fact, internet criminals are always looking for new ways to profit from the computers they control at the expense of their legitimate users or of the internet at large.

To be successful and maximize their profits, botnet operators need to be able to dynamically control and update their malware installations. This allows them to adapt a botnet’s behavior to the dynamic, adversarial environment in which they operate. For instance, a successful SPAM operation has to frequently modify the structure and content of the messages being sent, not only because the goals of the campaign may change rapidly over time (from advertising online pharmacy web-sites, to distributing attachments

that contain an exploit) but also because they need to remain one step ahead of anti-SPAM efforts.

The Command and Control (C&C) infrastructure that botmasters use to control and update their bots is thus a critical component of their operations. At the same time, C&C is an attractive target for those who wish to mitigate the damage caused by malware. Disabling a botnet’s C&C infrastructure can effectively *take down* the botnet. If botmasters lose control of their bots, they are prevented from using them to cause further mischief, even though the malware infections of the individual bots may not themselves have been remediated.

In recent years, botnet operators have therefore deployed a variety of client-server or peer-to-peer (P2P) C&C architectures, designed to be robust against take-down attempts. Recent client-server botnets include Pushdo/Cutwail [2], Torpig [3], Rustock [4] and Zeus (strictly speaking, Zeus is a toolkit for building botnets [5]). Examples of P2P botnets include Storm [6, 7], Nugache [8] and Waledac [9]. A few botnets, such as later versions of Conficker, combine both C&C approaches [10]. Modern client-server botnets do not naively rely on a single C&C server. Instead, they try to achieve robustness using domain flux [3], frequent updates of the C&C endpoints, and a high level of redundancy. For instance, the Koobface botnet uses about one hundred C&C servers running on compromised hosts [11]. In many cases, the botnets are also partitioned such that a single bot installation does not contain the coordinates of the entire C&C infrastructure.

Nonetheless, both client-server and P2P botnets have been the target of more or less successful node enumeration [12], infiltration [7, 3, 13, 11, 14] or take-down [15, 16, 17, 18] operations. Most recently Rustock, one of the largest SPAM botnets, was successfully taken down as a result of legal action led by Microsoft [19]. From a technical point of view, this involved identifying all of the C&C servers employed by the botnet, and taking them down simultaneously. Furthermore, the botmasters had to be prevented from registering specific domains in China that they could have used to recover control of their bots. In some cases it may not be feasible to take down identified C&C servers in a timely manner. However, even in such cases a blacklist of C&C servers, such as the ones provided by FIRE [20] or Zeus Tracker<sup>1</sup>, can be extremely useful. By deploying such a blacklist, network administrators are able to detect infected machines in their network and to prevent them from receiving further commands from the botmasters.

A common way of building a C&C blacklist is to run malware samples in an analysis sandbox such as Anubis [21] or CWSandbox [22] and detect the C&C traffic they generate [20, 23, 24, 25, 26]. This approach, however, suffers from the familiar limitations of dynamic analysis: incomplete coverage. That is, a single execution of a malware sample is unlikely to reveal several, let alone all, of the redundant C&C servers that the bot is able to contact. Furthermore, so long as the botnet’s main C&C servers are available,

<sup>1</sup><https://zeustracker.abuse.ch/>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ACSAC '11 Dec. 5-9, 2011, Orlando, Florida USA  
Copyright 2011 ACM 978-1-4503-0672-0/11/12 ...\$10.00.

the bot will not reveal any fallback strategies it may be provided with to recover from C&C takedown. For instance, after all of its C&C servers were taken down in 2008, the Srzbi botnet was able to get back online because it implemented a domain generation algorithm (DGA) as a recovery mechanism [16].

The goal of this work is to improve the amount of C&C behavior that can be observed during malware execution. Specifically, our aim is to increase the number of C&C servers that are contacted during an analysis run and can therefore be detected, and to trick malware into revealing backup C&C strategies such as DGA algorithms. For this, we use a specialized, targeted form of multi-path exploration [1]. The basic idea is that, if a bot is blocked from contacting a specific C&C server, it may try to contact alternative C&C endpoints.

When a bot attempts to contact an endpoint, an analysis system can either allow the communication to proceed, or block it. By making such a decision for each endpoint, we are essentially exploring a binary tree of execution paths. This tree grows in size exponentially with the number of endpoints that a sample may contact. During its execution, a malware sample frequently contacts a significant number of endpoints, most of which are not related to C&C, and are, therefore, uninteresting for our purposes. Therefore, it is not feasible in practice to explore the entire execution tree. Furthermore, before C&C behavior can be observed, we may need to allow the bot to contact specific endpoints, that it uses to test the state of its internet connection or to download additional code. Therefore, intelligent strategies are needed to explore the execution tree and discover the largest amount of C&C endpoints within a reasonable time frame. In Section 3, we introduce two such strategies. Furthermore, fast exploration of the execution tree requires the ability to revert the state of the analysis environment to the moment before a connection was blocked or allowed, to immediately explore the alternative branch.

In this work, we introduce a system called SQUEEZE, that is able to increase the amount of C&C endpoints and strategies that are revealed during dynamic analysis. We evaluate our tool on over 8000 malware samples. Results show that, even with a relatively short run-time, SQUEEZE can reveal hundreds of domains and IPs of C&C servers that were never observed in a normal run of our Anubis sandbox. As a result, we are able to increase the number of entries in an automatically-generated C&C blacklist by 19.7%. Furthermore, we show that our tool can reveal malware’s ability to use failover strategies such as DGA algorithms.

In summary, our contributions are the following:

- We introduce two effective strategies for exploring the tree of executions generated by a malware sample when faced with the availability or absence of contacted network endpoints.
- We design and implement a system that is able to efficiently execute multiple paths within this tree. For this, we employ techniques for reverting the execution state of the analysis sandbox and dynamically re-configuring its network environment.
- We evaluate the proposed techniques on a diverse and representative collection of real-world malware samples, and show that they lead to the detection of hundreds of additional C&C endpoints. This allows us to increase the size of an automatically-generated C&C blacklist by 19.7%. Furthermore, we show that SQUEEZE can reveal alternative C&C strategies that are used by malware to recover from the takedown of its primary C&C infrastructure.

## 2 Approach

To discover backup C&C servers and fallback strategies, we use a dynamic approach. That is, we run malware samples in a con-

trolled environment and observe their C&C communication. Our approach is targeted against client-server C&C architectures. The basic idea is to simulate the takedown of a malware’s primary C&C servers to trick it into revealing the servers and algorithms it is provided with for failover. For this, the obvious approach would be to simply block all traffic originating from the sandbox environment. There are, however, several reasons why this does not work in practice. First of all, a bot binary may be delivered by a dropper, as is the case in pay-per-install affiliate programs. In such cases, a naive approach would not allow the interesting bot binary to be downloaded in the first place. Furthermore, malware authors frequently contact a popular server to check their internet connectivity. If it is not available, the sample quits or idles until the network becomes available. Simply dropping all traffic would cause exactly this undesired behavior. Finally, unless we allow traffic to potential C&C endpoints, we are unable to verify if the traffic is indeed related to C&C activity.

These considerations provide us with a starting point for the design of a system for triggering malware’s failover C&C. First of all, we need to treat C&C traffic differently than other network traffic. Traffic that is not related to C&C communication should be allowed through, to the extent that this is possible without allowing the malware to cause harm. On the other hand, C&C traffic should be blocked to trigger backup behavior. For this, we require some knowledge about C&C communication, such as signatures for C&C traffic, that can be used to detect new C&C endpoints. Finally, to trick the malware into communicating with several of its redundant C&C servers, we need to be able to block C&C traffic *after* having allowed through enough of it for our models of C&C communication to detect it. For this, we need to “rewind” the malware execution to the moment before the C&C connection was successfully established. This can be achieved by reverting to a previously taken snapshot in a virtual machine.

Our approach is essentially a specialized form of multi-path exploration [1]. For this, it relies on a knowledge base on malware C&C communication, and on snapshotting functionality that allows it to explore multiple execution paths within a single analysis run. Whenever an analyzed sample attempts to contact a new endpoint, we take a snapshot before deciding whether to allow this traffic. Later in the analysis, we are able to revert to this snapshot to explore the alternative branch. The C&C knowledgebase provides some domain knowledge to help us decide which execution branches to explore. This knowledge could come in the form of network-based signatures for C&C traffic [24, 25] or of network-based [23] or host-based [26] behavioral models.

## 3 Exploration Strategies

To describe possible strategies for exploring a malware sample’s network behavior on different execution paths, we will use a running example. Figure 1 shows the different components of a malware, the endpoints it connects to and the effect that connectivity to these endpoints has on its network behavior. In this example a dropper is used to distribute the malware. As a first step, the dropper will connect to a remote server (*File-Server A*) and download the actual bot component. If the server is not accessible, the dropper employs a simple failover strategy and will try to contact backup server *File-Server B*. If either download is successful, the downloaded payload is executed, launching the actual bot. In this case, the dropper does not actually save the payload to disk before running it. Instead, it injects the downloaded code into another process and starts a remote thread [2]. Before executing any malicious behavior, the bot performs a connectivity check by attempting to access a popular web site, in this case the Yahoo home page. If the connectivity check fails, no further action is taken. Otherwise,

the bot registers itself at the primary C&C server and receives further instructions. These commands will cause the bot to launch a spam engine or a port scan, connecting to even more endpoints. The endpoints that are of primary interest to us, however, are those associated with C&C communication. If the primary C&C server is unreachable, our example tries to connect to a different hard-coded endpoint. Only if this backup is also unavailable, the bot falls back to a Domain Generation Algorithm. The DGA generates large numbers of domains based on the current date (obtained by parsing the Yahoo home page, rather than from the system clock). In case the two main C&C servers were taken offline, the botmasters would register a few of these domains each day, and use them to provide their bots with an updated list of C&C servers.

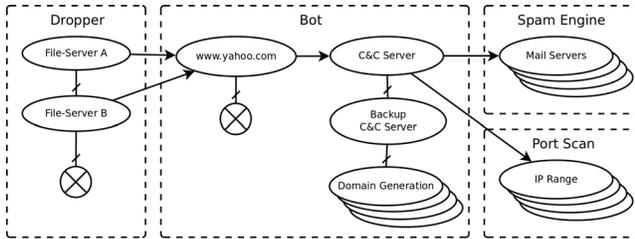


Figure 1: Running example: network endpoints contacted by a malware sample. A line between two endpoints indicates that the malware attempts to contact the endpoints one after the other. Lines with an arrow represent behavior following a successful connection, while crossed lines originate from unreachable endpoints. A crossed circle indicates no further network activity.

Every time SQUEEZE encounters an endpoint a decision has to be taken on whether to block this endpoint or allow the communication to succeed, influencing the behavior of the malware and the further endpoints it will or will not contact as a result. By representing the endpoints as nodes, the decision whether or not to block a connection as node expansion and the result as edges, the process of analyzing a malware’s network behavior can be modeled as the exploration of a binary tree. The observant reader will notice that Figure 1 is in fact a directed acyclic graph, rather than a tree, since the node `www.yahoo.com` can be reached from two different paths. However, malware may exhibit different behavior depending on the sequence of endpoints it was able to contact. For instance, the two file servers in Figure 1 may in fact deliver different payloads. Strictly speaking, each node in the execution tree is therefore identified by the sequence of endpoints that the malware has attempted to contact so far, rather than by the latest endpoint only. Note that an endpoint may be identified by its domain name or by its IP address. We use the IP address only in cases where the IP was not previously obtained by the analyzed sample through DNS resolution.

Since we can analyze each malware sample for a limited amount of time, it is unlikely that we can explore all possible branches of the execution tree during an analysis run. In fact, some malicious behavior (most obviously, scanning) involves attempting to contact a practically unlimited number of endpoints. Therefore we need to develop a strategy for exploring the execution tree that will reveal the largest amount of C&C activities, given the time constraints. Standard approaches for traversing a tree include depth-first and breadth-first search algorithms. As we will see, neither approach is directly suitable to the problem at hand. Instead, we will need to develop exploration strategies that make use of domain knowledge on C&C communication.

In a breadth-first search, each node is completely expanded before proceeding to the next node. This would allow us to explore

many branches of a sample’s initial network behavior, but would not lead us very deep within the execution tree. Since C&C endpoints, tried one after the other by the malware, typically form a deep branch in the tree, depth is more important to us than breadth. A depth-first search would therefore seem better suited to our needs, but in fact it also has its pitfalls. With a depth-first search, exploration can easily get “stuck” in a branch where a lot of uninteresting network activity is occurring (such as port scanning or click fraud).

A solution to this problem is to enrich the search strategy with additional knowledge on C&C communication. Such knowledge can contribute to the decision whether a branch is of interest and its nodes should be expanded or not. Ideally, we would exactly know which nodes of the tree correspond to C&C endpoints. In practice, however, we may not be able to automatically identify all C&C communication. Nonetheless, since our goal is to automatically build a blacklist of C&C endpoints, it is necessary to have some means of identifying C&C traffic. For SQUEEZE, we use domain knowledge in the form of a set of network-based C&C signatures that were provided to us by a security company. These signatures have been vetted by human experts, so we have high confidence that they can identify C&C communication without generating false positives (though they may not cover all forms of C&C communication, leading to false negatives). Furthermore, by matching these signatures against the traffic observed during malware execution in our Anubis sandbox in the past, we can identify a set of known C&C endpoints. The C&C signatures and the set of known C&C endpoints constitute our initial knowledgebase on malware Command and Control.

There is a significant difference between these two types of information: A known C&C endpoint can be blocked *before* a connection is actually established, while a C&C signature can only be matched *after* traffic has already been transmitted. Hence, the former can be used with a block-first strategy while the latter requires an allow-first strategy. Another aspect to take into account is that each component of the malware – bot, spam engine, etc. – is typically executed in a separate process. Using our C&C knowledgebase, we are able to identify the process or processes that are carrying out C&C communication. We will call these processes the *bot component*. With this information, we can focus our efforts on exploring the parts of the execution tree that represent endpoints contacted by the bot component.

For SQUEEZE, we developed two alternative strategies for exploring the execution tree. Both can be seen as depth-first search algorithms, but they differ in the type of C&C knowledge that they leverage. Figure 2 provides an example for both approaches.

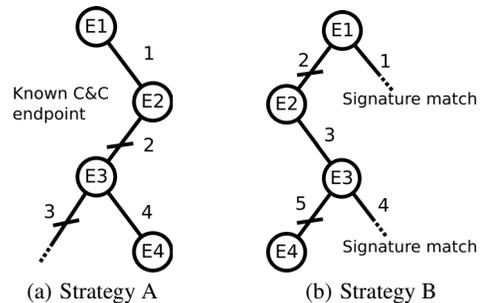


Figure 2: The two exploration strategies. A node represents an endpoint. A solid line represents an allowed connection, a crossed line a blocked connection and dotted lines backtracking before another endpoint is discovered.

**Strategy A** takes advantage of the set of known C&C endpoints. When using this strategy, SQUEEZE initially allows all connections. Once the analyzed malware attempts to contact a known

C&C endpoint, SQUEEZE blocks the connection, and switches to a block-first, depth-first search. After each decision to block or allow an endpoint, a timeout is restarted. If no further endpoints are contacted within the timeout, the search backtracks. Furthermore, whenever the tool detects a connection to a known C&C endpoint, it identifies the responsible process, and adds it to the bot component. The exploration strategy only takes into account endpoints contacted by the bot component, and always allows traffic to other endpoints. The idea behind this strategy is to allow the malware to perform connectivity checks or other network behavior that needs to be successfully completed before C&C communication is triggered. Once the malware attempts a first C&C connection, it is blocked, leading the bot to attempt to contact its backup C&C servers, one after the other. Note that this strategy will fail if no known C&C endpoints are observed.

Figure 2(a) shows an example of Strategy A in action: After allowing the connection to E1, the system recognizes the known C&C endpoint E2. Blocking E2 causes the malware to perform an intermediate connection check on endpoint E3. The connection is initially blocked, and the sample performs no further network behavior. After a timeout, we track back and unblock E3. Following the now successful connection check, the malware contacts its backup C&C server at endpoint E4.

**Strategy B** takes advantage of the set of C&C signatures. When using this strategy, SQUEEZE performs a depth-first, allow-first search. However, the C&C signatures are matched against all observed network traffic. As soon as a signature matches the traffic going to a specific endpoint, SQUEEZE backtracks, blocking communication with this C&C endpoint. This strategy is aimed at efficiently discovering endpoints that can be detected with the available C&C signatures.

Figure 2(b) shows an example of Strategy B: When a signature match is detected in traffic to E1, the system tracks back and blocks this endpoint. Since the following connection check on endpoint E2 is successful, the malware immediately proceeds to the backup C&C server E3. Traffic from E3 also matches a C&C signature. Blocking this endpoint reveals E4.

The two strategies have different advantages and disadvantages. By using a block-first approach, strategy A tends to reveal more endpoints overall. However, unless the exploration backtracks far enough to allow the malware to connect to an initially blocked endpoint, we do not observe traffic to it and are therefore unable to confirm if it is in fact a C&C server. Strategy B has the advantage that it will trigger on a C&C connection even if the contacted C&C endpoint is not known. Finally, an advantage of strategy A is that it is more flexible, and can leverage knowledge of C&C endpoints regardless of how they were detected. Strategy B, on the other hand, relies on the ability to detect C&C communication on the fly. It cannot be as easily combined with C&C detection approaches that rely on observing the behavior of a bot *after* it receives a command, such as those from Wurzinger et al. [23].

## 4 System Description

Figure 3 shows an overview of SQUEEZE’s architecture. SQUEEZE takes as input a malware sample and a knowledgebase on C&C communication. Depending on the exploration strategy used, this knowledgebase can contain C&C network signatures or C&C server endpoints. While the sample is executing in the sandbox, its network activity is constantly monitored and compared with the C&C information. Whenever a new endpoint is encountered, we take a snapshot of the current state of the system. The exploration strategy then determines when to backtrack in the execution tree by reverting to a previous snapshot. Furthermore, the sandbox’s network environment is dynamically modified by re-configuring a DNS re-

cursor and a firewall. Finally, a delay analysis module within the sandbox tries to avoid delays between the malware’s network connections by manipulating time (as observed from within the sandbox), to “fast-forward” malware execution.

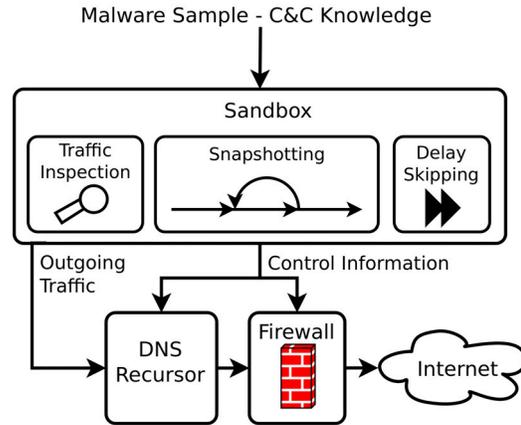


Figure 3: SQUEEZE Architecture

### 4.1 Sandbox

To build our analysis sandbox we leveraged our Anubis dynamic malware analysis system [21, 27]. Anubis is a dynamic malware analysis service that has been operating since early 2007, and has since analyzed over ten million malware samples. At its core is an instrumented full system emulator for the Windows XP operating system [21] and is built upon Qemu [28].

**Snapshotting.** To be able to backtrack in our exploration of the execution tree, we need to be able to revert the state not only of the malware being analyzed, but also of the operating system and analysis environment that it runs in. While it would in principle be possible to simply restart a sample’s analysis with a different network configuration every time we need to backtrack, this would be extremely inefficient. Furthermore, in the new execution the malware might exhibit completely different behavior, confusing our exploration algorithms.

Since Anubis is a full system emulator, it is possible to take a snapshot of the state of the entire system (disk, memory and processor state), and to restore such a snapshot to revert the execution to the previous state. In fact, Qemu provides such a snapshotting mechanism. However, the Anubis instrumentation also keeps some execution state. For instance, it tracks which processes are under analysis and what resources operating system handles correspond to. To use Qemu snapshotting in Anubis, we therefore extend it to also save and restore the state of the Anubis instrumentation.

To reduce the time spent saving and restoring snapshots, we keep all generated snapshots in main memory, rather than save them to disk. Since we only have a limited amount of memory available, this imposes an upper bound on the number of snapshots that can be stored simultaneously. Since we use depth-first exploration strategies, this essentially corresponds to a limit to the depth of the execution tree that we can explore. With a suitable amount of memory this limit is seldom reached during SQUEEZE analysis.

**Delay skipping.** A general problem of dynamic analysis is that, for practical reasons, we are only able to execute each binary for a limited amount of time. For instance, in its default configuration, Anubis executes each sample for four minutes. As a consequence, malware could evade dynamic analysis by simply waiting a certain amount of time before performing any interesting behavior. For

SQUEEZE, this problem is exacerbated by the fact that malware frequently waits for some time before trying to contact a backup C&C server or resorting to a failover C&C strategy.

While this problem is hard to solve in general, in practice most malware samples can be tricked into skipping such delays. For this, we modify the behavior of system calls that can be used to delay a process' execution. As a first countermeasure, we intercept the undocumented `NtDelayExecution` native Windows API function, which returns after a specified time interval, and limit the delay to a maximum of half a second. Furthermore, we tamper with instructions and system calls, such as `GetTickCount`, that can serve as local sources of timing information, and could be used to busy-wait until a certain time has elapsed. Specifically, we send time forward by an interval that grows exponentially with the number of invocations of `GetTickCount`, up to a maximum value (currently set to five hours).

**Traffic Interception.** To be able to trigger the generation of snapshots and the dynamic re-configuration of the network environment required by our exploration strategy, SQUEEZE needs to match the network traffic generated by the malware against C&C endpoints and signatures on the fly. For this, we hook into the appropriate operating system interfaces and inspect their parameters *before* the actual functions are executed. Note that in Anubis, hooking happens from "outside the box" by instrumenting the emulator, rather than by modifying the operating system "inside the box". Specifically, we hook the `NtDeviceIoControlFile` function which is used to send data to device drivers. The driver used to transmit data over sockets is `afd.sys`, which is the Ancillary Function Driver for WinSock. Of the various I/O control codes used to handle connections, two are important for us: `AFD_CONNECT` which is used to establish a connection to a port at an IP address and `AFD_SEND` which sends input data out over an established link. To keep track of and be able to react to DNS queries, we additionally intercept the Windows DNS client's `DnsQuery` function that is implemented in `dnsapi.dll`.

## 4.2 Network Environment

SQUEEZE needs to dynamically adapt the network environment to block or allow connections to specific endpoints in accordance with the selected exploration strategy. For endpoints identified by a hostname, we want to block the hostname at the DNS level rather than the IP addresses associated with it. To this end we redirect all DNS queries to a local recursor. We use the popular PowerDNS recursor<sup>2</sup>, because it allows fine-grained dynamic configuration using LUA scripts. This enables us to create an `NXDOMAIN` reply for DNS queries for hostnames we wish to block. To block IP addresses that are directly contacted by the malware (without a DNS lookup), we leverage the netfilter firewall<sup>3</sup> and configure it to reject the connection attempt and reply to a TCP SYN packet with a TCP reset, and to a UDP packet with an ICMP destination unreachable packet. This is more efficient than simply dropping the packet, because it avoids unnecessary delays caused by waiting for connection time-outs on the client side.

In addition to the dynamic network blocking required for SQUEEZE, we also need to prevent the analyzed samples from causing harm to the internet at large. For this, we deploy the same countermeasures employed by Anubis. These include preventing SPAM by redirecting SMTP connections to a local mail server (that does not actually send the mail), limiting a malware sample's contribution to denial of service attacks by throttling network throughput and number of connections, and blocking frequent attack vectors by redirecting traffic on a number of ports (such as TCP ports 139 and 445) to a

<sup>2</sup><http://www.powerdns.com>

<sup>3</sup><http://www.netfilter.org>

local honeypot. These measures cannot completely guarantee that the malware will not cause any harm (0-day attacks in particular are hard to block). However, experience from the Anubis deployment shows that they are in practice sufficient for the safe operation of a malware analysis sandbox. Over four years running Anubis, we have received only a single abuse complaint, which was promptly addressed.

## 5 Evaluation

To evaluate SQUEEZE, we tested it on several thousand malware samples that had generated C&C traffic in Anubis. As we will show, analyzing these samples with SQUEEZE revealed a significant number of C&C servers that were not observed in Anubis.

To detect C&C traffic, we use a set of 192 network signatures provided to us by a security company. These signatures are similar in expressiveness to Snort rules [29], and include regular expressions that are to be matched against network flows. These signatures have been manually vetted for accuracy, and do not in our experience lead to false positives when matched against Anubis traffic. However, they may not cover the full spectrum of current malware C&C communication.

### 5.1 Dataset

By matching the C&C signatures against traffic dumps generated by Anubis analysis runs, we identify a set of samples that generated C&C traffic, together with the C&C endpoints they connected to. For our evaluation, we then selected a subset of these samples based on two criteria: First of all, we required recent samples, since older samples are often no longer functional because they do not contain up-to-date information for contacting C&C infrastructure. Second, we would like to test our tool on a dataset involving the largest possible set of C&C endpoints. Thus, we tried to contact each C&C endpoint and discarded those that were no longer reachable. This includes domains that failed to resolve to an IP address and servers that were no longer reachable on the port used for C&C. For each of the remaining endpoints, we selected the most recent samples that communicated with it.

In total, we selected 8,346 malware samples. To assess the diversity of this dataset, we scanned the samples with the Kaspersky anti-virus engine. Kaspersky provided a total of 2,225 different AV labels for these samples. Kaspersky virus labels have a loosely structured format, such as "Trojan-Spy.Win32.Zbot.aebi". By discarding the last part of these labels (indicating a specific variant), we can obtain a more coarse-grained classification of malware binaries into families. Our dataset contained 213 malware families according to this classification, with the most represented ten families accounting for only 21% of the dataset. Comparing these labels with a recent list of the most prevalent malware families [30] shows that fourteen of the twenty top malware families are represented in our dataset.<sup>4</sup> This confirms that we are testing our tool on a diverse and representative malware dataset. Notable exceptions (top 20 malware families not seen in our dataset) are Conficker and Storm 2.0, which are P2P botnets and are therefore not covered by our signatures for client-server C&C activity.

**Setup.** To run SQUEEZE we used four virtual machine instances on a host with 32GB memory and an Intel XEON E5420 CPU. 4GB of memory was assigned to each machine of which 3GB were dedicated to storing snapshots. Since snapshots are on average 130MB in size, this allows for approximately 21 snapshots to be taken. We run each malware sample for up to 6 minutes. Our setup can therefore analyze about one thousand samples a day. We ran our eval-

<sup>4</sup>A fifteenth family, the Bredolab botnet, is absent from our dataset because it was successfully taken down shortly after the report was published [31].

uation for 10 days in March 2011. Since our setup did not allow us to deploy the two strategies in parallel, the system was configured to use exploration strategy A during the first five days, and strategy B during the last five days. Our evaluation dataset is therefore split into Dataset A, consisting of 4,013 samples analyzed with SQUEEZE using strategy A, and Dataset B, consisting of 4,333 samples analyzed using strategy B. We used a new dataset for strategy B, rather than repeat analysis on Dataset A, to ensure we were testing SQUEEZE against a “fresh” selection of malware samples.

## 5.2 Malware Behavior

	Strategy A	Strategy B
Samples analyzed	4013	4333
Initial C&C knowledge match	54%	58%
No further activity	44%	43%
Substantial delay skipping	34%	31%
New endpoints	25%	23%
New endpoints in bot component	19%	13%
New endpoints with signature match	9%	8%

Table 1: Malware behavior in SQUEEZE

Table 1 shows an overview of the behavior of malware samples when analyzed with SQUEEZE using the two proposed exploration strategies. The first thing we notice is that a significant fraction of the analyzed samples never trigger SQUEEZE’s blocking mechanisms. With strategy A, only 54% of analyzed samples ever try to contact a known C&C endpoint. Likewise, with strategy B only 58% of analyzed samples generate traffic that matches our C&C signatures. This is despite the fact that all of the samples in our dataset, when originally analyzed in Anubis, had performed signature-matching traffic that led us to detect an endpoint in the first place. We have found that in some cases the original binary is actually a downloader: In the Anubis execution, it dropped a remote-controlled bot, but in the SQUEEZE execution it dropped a completely different payload. In other cases some of the endpoints contacted by the sample were no longer available. One reason for this is that, during this experiment, the delay between executing a sample in Anubis and in SQUEEZE was too high (several days). Results from our deployment (Section 5.5) show that this result can be improved by reducing this delay.

A large share of the samples, 44% and 43% respectively for strategies A and B, do not employ a backup strategy at all. That is, after a C&C endpoint is blocked they show no further network activity. The difference between these first two rows of Table 1 leaves 10% and 15% of samples, respectively, where SQUEEZE may have been able to trigger additional C&C activity. As can be seen in the last row, 9% and 8% of samples respectively reveal new C&C endpoints when ran in SQUEEZE compared to those they revealed in Anubis. These endpoints are confirmed by a match against a C&C signature.

There may be several reasons why, for the remaining samples, no C&C traffic can be detected. First of all, with the block-first strategy of approach A, C&C traffic can only occur if the endpoint is unblocked before the end of the analysis. Then, backup C&C endpoints may not have been active during analysis. Finally, communication with a backup C&C server might use a slightly different format that cannot be detected by our C&C signatures. Therefore, 9% and 8% mark the lower bound of samples that revealed valuable endpoint information. A reasonable upper bound can be given by

also including other endpoints that the bot component connected to. Recall that the bot component is defined as the set of processes that have performed C&C activity that matches our signatures or known endpoints. This provides an upper bound of 19% and 13% of samples respectively.

Furthermore, the results show that allowing connectivity checks to go through is important for SQUEEZE’s effectiveness. Over ten percent of the samples in our dataset performed connectivity checks by contacting a variety of popular sites such as google.com, microsoft.com, or weather.yahoo.com. Some samples perform such checks only at startup, while others test their connectivity again whenever they fail to contact a C&C server.

Finally, our delay skipping techniques also play a significant role in SQUEEZE’s effectiveness. For around one third of samples, SQUEEZE fast-forwards time in the sandbox by at least one minute. In Section 5.4 we will discuss an example of a malware family that reveals a significant number of C&C endpoints, but only after skipping a delay of over half an hour.

## 5.3 Endpoints

Table 2 shows the total number of distinct endpoints contacted when executing the samples in our datasets in an unmodified Anubis sandbox, as well as the additional endpoints contacted when using SQUEEZE with the two alternative exploration strategies.

The most important results are the numbers of C&C endpoints detected shown in the rows labeled “Endpoints with signature match”. SQUEEZE is able to detect a significant number of C&C endpoints that were not contacted during the execution of any of the samples in datasets A and B. SQUEEZE reveals 201 and 185 new C&C endpoints respectively using strategies A and B. This corresponds to an increase in the number of C&C endpoints that can be extracted from our two malware datasets of 12.6% and 19.7% respectively. The number of IPs is significantly lower than the number of domains: Recall that we take into consideration an IP address only if it is accessed directly rather than obtained through DNS resolution. While some malware samples contain hard-coded C&C IP addresses, most rely on the DNS infrastructure to resolve the domain names of their backup C&C servers.

As discussed in the previous section, the number of endpoints with a signature match is only a lower bound for the amount of C&C endpoints we may have observed. This is chiefly because our C&C signatures may not match on all of a sample’s C&C connections, particularly if the format of messages to the primary C&C server is not identical to the format used when communicating with a backup server. As a rough upper bound for C&C endpoints we can use all the endpoints contacted by the processes involved in C&C communication (the bot component). SQUEEZE may therefore have revealed up to 714 and 506 new C&C endpoints respectively using strategies A and B. This corresponds to an increase of 29.4% and 32.8% compared to unmodified Anubis.

To further investigate how many of the endpoints revealed by SQUEEZE are actually malicious, we used nine publicly available blacklists. For this, we queried these blacklists repeatedly until two months after running the malware samples. The results are shown in the row “Endpoints in blacklists”. Table 3 shows a breakdown of these results for new endpoints (that is, all endpoints that were not contacted during the baseline Anubis analysis). Of the blacklists considered, only AMaDa focuses specifically on malware C&C endpoints. Several of the other blacklists include C&C endpoints as well as other malicious servers, while Google Safe Browsing and Norton Safe Web focus on malicious web sites. We nonetheless include these blacklists because miscreants may use endpoints for multiple malicious purposes. Note that all of these blacklists are meant to be deployed to block traffic to or from malicious hosts, and therefore strive to have extremely low false positive rates. The

Dataset A	Baseline			Strategy A		
	Domains	IPs	Total	Domains	IPs	Total
Endpoints	3562	767	4329	661	942	1603
Endpoints in bot component	2080	362	2432	454	260	714
Endpoints in blacklists	1970	111	2081	391	36	427
Endpoints with signature match	1489	110	1599	195	6	201
Dataset B	Baseline			Strategy B		
	Domains	IPs	Total	Domains	IPs	Total
Endpoints	2627	364	2991	534	325	859
Endpoints in bot component	1330	211	1541	293	213	506
Endpoints in blacklists	1336	81	1417	353	15	368
Endpoints with signature match	885	53	938	184	1	185

Table 2: New endpoints revealed by SQUEEZE compared to baseline (execution in Anubis sandbox)

Blacklist	Strategy A			Strategy B		
	Domains	IPs	Total	Domains	IPs	Total
Google Safe Browsing (Malware)	95	11	106	53	5	58
Norton Safe Web	47	N/A	47	80	N/A	80
Spamhaus (SBL,XBL,DBL)	2	20	22	2	7	9
ClearCloud DNS	281	N/A	281	226	N/A	226
DNS-BH Malware Domain Blocklist	101	N/A	101	86	N/A	86
Malware Domain List (MDL)	11	5	16	15	0	15
malc0de DNS Blackhole	80	3	83	70	1	71
Emerging Threats fwip rules	N/A	6	6	N/A	3	3
abuse.ch Malware Database (AMaDa)	32	2	34	40	2	42
Total Blacklisted	391	36	427	353	15	368
Total Endpoints	661	942	1603	534	325	859

Table 3: New endpoints revealed by SQUEEZE listed in publicly available blacklists

results are very different for domains and IP addresses: 62.5% of domains are listed in at least one blacklist, while the same is the case for only 4% of IPs. Overall, however, SQUEEZE has contacted 427 and 368 additional blacklisted endpoints using strategies A and B. Compared to the blacklisted endpoints revealed by Anubis, this is an increase of 20.5% and 26.0% respectively.

In conclusion, an estimate of the percentage of additional C&C servers revealed by SQUEEZE ranges between a minimum of 12.6% (confirmed C&C endpoints for strategy A) and a maximum of 32.8% (endpoints in bot component for strategy B). For automatically-generated C&C blacklists, such as the one provided by FIRE [20], these results have practical implications. The reason is that they demonstrate that, by complementing a malware analysis system with SQUEEZE, we can generate blacklists that provide a significantly improved coverage of malware C&C servers.

#### 5.4 Qualitative Results

In this section, we will highlight some interesting aspects of the behavior of samples analyzed with SQUEEZE, with particular emphasis on those belonging to the top twenty malware families from the FireEye report [30].

Palevo/Butterfly is a botnet toolkit that, according to [30], is behind the currently most prevalent malware family. With SQUEEZE, we were able to observe the backup C&C strategies employed by Palevo samples. For this, strategy B was particularly effective. These samples initially attempt to contact a static IP address. This traffic matches our C&C signatures. After this endpoint is blocked, they attempt to contact another static IP (once again matching our signatures). Finally, they fall back to a domain generation algorithm. One sample in particular tried to resolve 42 generated domains during the analysis run, none of which were actually active.

Samples of the Pakes malware family fall back to a number of backup C&C endpoints when their primary C&C is blocked. Be-

fore attempting to contact each successive endpoint, however, these samples idle for several minutes. This delayed execution behavior is implemented with repeated calls to GetTickCount. Therefore, SQUEEZE was able to skip these delays. For one sample, analysis with strategy A revealed 20 additional endpoints compared to the original Anubis run, but only after sending time forward by over half an hour. As soon as Pakes is able to contact a C&C server, it receives instructions to send spam.

Koobface is another interesting malware family, because it straddles the line between client-server and P2P botnet C&C. As reported by Thomas et al. [11], Koobface sets up HTTP-based C&C servers on machines it has exploited. According to Thomas et al., close to one hundred such parasitic C&C servers are active on a given day. During analysis, Koobface samples initially performed a connection check by contacting the Google main page. Then they tried to contact a number of hardcoded C&C servers. By blocking them one after the other, SQUEEZE was able to trigger connections to up to fifty C&C endpoints before the six minute timeout. In this case, simply increasing the timeout would presumably allow us to detect further C&C endpoints.

The Piptea malware family (also known as Harnig) is a trojan downloader. During normal execution, samples contact the primary C&C server and query it via HTTP to retrieve encrypted download instructions. If the primary C&C server is unreachable, a second one serves as a fallback. However, the Piptea botnet was put offline, presumably by its operators, shortly after the Rustock take-down on March 17th, 2011 [32]. Since this was the first day of our evaluation period, we are able to observe this in our results. A few Piptea samples analyzed on March 17th successfully contacted their C&C servers. Those analyzed on a later date, however, could reach neither their primary nor their backup C&C servers.

Another interesting sample was also a downloader, most likely related to a pay-per-install affiliate program. This dropper queries

a C&C server using a proprietary, plain-text protocol for which we do not have a signature. The server then provides a list of URLs from which to download additional executables. In our analysis, the dropper downloaded and executed three additional binaries one after the other. All three binaries are recognized by anti-virus engines as malicious, but interestingly do not seem to be related to each other: The dropper is thus installing multiple malware strains on the same machine. The third executable is a bot, and our signatures were able to detect its C&C traffic and the corresponding endpoint. When strategy B blocked this endpoint, the bot attempted to contact a backup C&C domain that had not yet been registered.

While SQUEEZE was able to reveal the backup C&C strategies employed by some malware families, there are also families that displayed no such behavior: After a C&C server was blocked, these samples simply idled or performed a default behavior such as network- or file-based propagation. We further investigated two such families, and concluded that indeed these malware samples do not employ a backup C&C strategy.

The first such family we investigated is Virut. Virut spreads by file infection, and its bot component is rather basic: The samples in our dataset use only a single IRC C&C server, which they contact to get download information on further binaries. If this server is unavailable, Virut's activity is limited to spreading.

The second family is Zbot. Zbot samples stem from the Zeus toolkit [5], which allows users to create their own, customized botnet. None of the samples we analyzed with SQUEEZE displayed a backup strategy when their first C&C server was blocked; In fact they remained completely idle. This seems surprising, considering that Zeus is still widely successful despite efforts from projects such as Zeustracker to identify and blacklist its C&C servers. The recently published source code of the Zeus toolkit (Version 2.0.8.9), however, allowed us to confirm this observation: Although zBot can use various C&C servers, it first relies on a primary C&C server to provide the bot configuration. This server is compiled into the binary and can thus only be changed by means of a binary update. To increase resilience, operators of Zeus-based botnets presumably rely on frequent updates to their bot binaries. If this is the case, a malware analysis system could reveal more Zbot C&C servers if it were to capture updated binaries downloaded during analysis, and periodically re-analyze the latest version. We leave this for future work.

## 5.5 Deployment

Based on the encouraging evaluation results we set up a stable deployment of SQUEEZE, integrated with the Anubis infrastructure. This deployment uses strategy B, and is configured to re-analyze all samples that matched a C&C signature in the original Anubis run. From the beginning of June until the end of August 2011, over 32,000 samples were re-analyzed by SQUEEZE. During this period, plain Anubis found a total of 4355 distinct endpoints in the bot component and 2338 distinct endpoints with a signature match. Squeeze improved this by revealing an additional 1706 (39%) and 278 (12%) endpoints respectively.

We tested the statistical significance of these results using a single-tailed Wilcoxon signed-rank test [33]. The research hypothesis states that an analysis with SQUEEZE reveals more endpoints than a plain Anubis run. The null hypothesis is rejected with a probability of error below 0.001% when applying the test to either endpoints in the bot component or endpoints with a signature match.

The deployment also backed our assumption that decreasing the delay until re-analysis would have a positive effect on the share of samples that re-connect to a C&C endpoint. While in Table 1 only 58% of samples re-analyzed with SQUEEZE matched a C&C signature, this number rises to over 90% in our live deployment, where the average delay between analysis in Anubis and re-analysis

in SQUEEZE is reduced to eleven hours.

As a consequence, the FIRE service<sup>5</sup>, which makes use of Anubis as a source of C&C information, directly benefits from SQUEEZE. This improves the coverage of both the FIRE blacklist and of its ranking of "malicious networks".

## 6 Limitations and Future Work

In this section we briefly discuss the main limitations of our approach. The first challenge is posed by the communication method used by botnets. First of all, our approach is of limited utility against botnets that use a fully decentralized P2P architecture. While SQUEEZE could still investigate the traffic and even block connections, it would simply produce a list of infected peers. Thus, mitigating P2P botnets clearly requires alternative approaches. However, as was discussed in Section 5.1, the majority of today's most prevalent malware families make use of client-server C&C. Thus, SQUEEZE can provide useful intelligence in the current threat landscape. Another issue is that botnets could use encrypted C&C protocols that are harder to detect at the network level. However, the specific mechanism used to model and detect C&C communication is largely orthogonal to our work. SQUEEZE could be combined with C&C detection techniques that do not suffer from this limitation, such as those based on behavioral detection at the network level [23], or on host-based analysis of information flows [26]. Strategy A, in particular, can be trivially combined with arbitrary C&C detection mechanisms.

A further limitation is that our tool can currently provide limited information on DGAs it triggers. That is, SQUEEZE can observe a number of requests going to generated domains, however it can neither detect that these are C&C servers (since they are typically offline) nor reveal how the DGA works. This limitation can be overcome by applying slicing and gadget extraction techniques [34] that can extract the DGA as a functional, self-contained piece of code that can be used to generate new domains. Finally, like most techniques based on the dynamic analysis of malware, SQUEEZE may be thwarted by malware that detects it is running in an instrumented environment and refuses to run. In recent years, a number of techniques have been proposed that attempt to mitigate this problem [35, 36, 37, 38, 39, 40].

## 7 Related Work

Botnets have been the subject of a significant amount of research. A survey of this topic has been performed by Bailey et al.[41]. Several studies have been performed with the goal of measuring and understanding the botnet phenomenon [4, 5, 6, 8, 9, 11, 12]. In other cases, researchers went a step further and actively infiltrated a botnet with the goal of obtaining further insight, taking down or even taking over the botnet [7, 3, 13, 14]. Another well-explored topic is the network-level detection of botnets, based on the bots' crowd-like behavior [42], on their reaction to C&C commands at the network [23] or host level [26], or on signatures generated by detecting recurrent patterns in botnet traffic [24, 25]. These approaches can be used either to create detection models that can be deployed to protect a network and detect which machines are already infected, or to detect and blacklist C&C servers. FIRE [20] identifies malware C&C servers and tracks their uptime, to identify networks that persistently host malicious activities. This has already led some ISPs to take an interest in the issue and improve their security practices [43]. SQUEEZE can be readily adapted to take advantage of any of these techniques for detecting C&C communication.

Limited coverage is a general problem of dynamic program analysis. Techniques to overcome this limitation have been proposed in

<sup>5</sup><http://maliciousnetworks.org>

a number of fields such as software testings [44] and vulnerability discovery [45]. In the context of malware, Limbo [46] aims to recognize malicious device drivers (rootkits) by forcing execution to traverse the driver's control flow graph. Limbo however cannot ensure that the program state (that is, the values of registers and memory) is consistent with the program paths it is forcing. Other approaches [1, 47] overcome this limitation. Multi-path exploration [1] is the approach most closely related to our work, since it explores possible execution paths by modifying the execution environment before reverting to a snapshot. Multi-path exploration uses a constraint solver to find out how to modify the environment to lead execution down an alternative branch. MineSweeper [47] similarly aims to uncover trigger-based malware behavior using a combination of concrete and symbolic execution. While useful, these techniques suffer from the path explosion problem: The overall number of program paths that need to be explored grows exponentially. The reason is that, at each interesting branch in the program, the analysis has to follow two successor paths. Furthermore, code obfuscation can make the constraint solving step required by such tools provably hard [48]. SQUEEZE does not attempt to discover triggers for hidden behavior: Instead, it assumes that the behavior of interest (backup C&C communication) is triggered by the unavailability of network resources. This reduces the execution space that has to be explored to the (much smaller) endpoints tree, such as the one shown in Figure 2.

A different approach for increasing coverage is used by Reanimator [49]. By automatically identifying and modeling the code responsible for an observed behavior, Reanimator is able to recognize the same capability in other programs even if it is not observed at run-time. A limitation is that a behavior has to be triggered in at least one malware execution before it can be modeled.

## 8 Conclusion

Modern botnets are complex distributed systems designed to be resilient in the face of takedown attempts. To avoid losing control over their infected computers, botmasters use redundant C&C servers and failover C&C strategies such as domain generation algorithms. While it is possible to detect malware C&C servers by monitoring the execution of bots in a controlled environment, this approach suffers from limited coverage and will not reveal all C&C servers. Connections to backup C&C servers will not be triggered until the primary servers are taken down. In this paper we have introduced SQUEEZE, a system that uses a specialized form of multi-path exploration to trick bot binaries into revealing additional C&C endpoints and failover strategies. We introduced two alternative strategies for making use of domain knowledge on C&C communication to efficiently explore the execution paths that are revealed when communication with a contacted endpoint is allowed or blocked. By testing SQUEEZE on a diverse and representative malware dataset and comparing it against an ordinary analysis sandbox, we showed that it can reveal hundreds of additional active C&C servers and trigger DGA algorithms that bots use as a failover strategy.

## 9 Acknowledgements

The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement n. 257007 (SysSec), from the Prevention, Preparedness and Consequence Management of Terrorism and other Security-related Risks Programme European Commission - Directorate-General Home Affairs (project i-Code), and from the Austrian Research Promotion Agency (FFG) under grant 820854 (TRUDIE). This publication reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

## 10 References

- [1] Moser, A., Kruegel, C., Kirda, E.: Exploring Multiple Execution Paths for Malware Analysis. In: IEEE Symposium on Security and Privacy. (2007)
- [2] Decker, A., Sancho, D., Kharouni, L., Goncharov, M., McArdle, R.: A study of the Pushdo / Cutwail Botnet. [http://us.trendmicro.com/imperia/md/content/us/pdf/threats/securitylibrary/Study\\_of\\_pushdo.pdf](http://us.trendmicro.com/imperia/md/content/us/pdf/threats/securitylibrary/Study_of_pushdo.pdf) (2009)
- [3] Stone-Gross, B., Cova, M., Cavallaro, L., Gilbert, B., Szydowski, M., Kemmerer, R., Kruegel, C., Vigna, G.: Your botnet is my botnet: analysis of a botnet takeover. In: Proceedings of the 16th ACM conference on Computer and communications security (CCS). (2009)
- [4] Chiang, K., Lloyd, L.: A case study of the rustock rootkit and spam bot. In: Proceedings of the First Workshop on Hot Topics in Understanding Botnets. (2007)
- [5] Binsalleeh, H., Ormerod, T., Boukhtouta, A., Sinha, P., Youssef, A., Debbabi, M., Wang, L.: On the analysis of the zeus botnet crimeware toolkit. In: International Conference on Privacy, Security and Trust. (2010)
- [6] Grizzard, J.B., Sharma, V., Nunnery, C., Kang, B.B.H., Dagon, D.: Peer-to-Peer Botnets: Overview and Case Study. In: Proceedings of the First Workshop on Hot Topics in Understanding Botnets table of contents. (2007)
- [7] Holz, T., Steiner, M., Dahl, F., Biersack, E., Freiling, F.: 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. (2008)
- [8] Dittrich, D., Dietrich, S.: P2p as botnet command and control: a deeper insight. In: 3rd International Conference On Malicious and Unwanted Software (Malware). (2008)
- [9] Stock, B., Goebel, J., Engelberth, M., Freiling, F.C., Holz, T.: Walowdac - analysis of a peer-to-peer botnet. In: Proceedings of the 2009 European Conference on Computer Network Defense (EC2ND). (2009)
- [10] Fitzgibbon, N., Wood, M.: Conficker. C: A technical analysis. [http://www.sophos.com/sophos/docs/eng/marketing\\_material/conficker-analysis.pdf](http://www.sophos.com/sophos/docs/eng/marketing_material/conficker-analysis.pdf) (2009)
- [11] Thomas, K., Nicol, D.: The koobface botnet and the rise of social malware. In: 5th International Conference On Malicious and Unwanted Software (Malware). (2010)
- [12] Kang, B.B., Chan-Tin, E., Lee, C.P., Tyra, J., Kang, H.J., Nunnery, C., Wadler, Z., Sinclair, G., Hopper, N., Dagon, D., Kim, Y.: Towards complete node enumeration in a peer-to-peer botnet. In: Proceedings of the 4th International Symposium on Information, Computer, and Communications Security (ASIACCS). (2009)
- [13] Cho, C.Y., Caballero, J., Grier, C., Paxson, V., Song, D.: Insights from the inside: a view of botnet management from infiltration. In: USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET). (2010)
- [14] Stone-Gross, B., Holz, T., Stringhini, G., Vigna, G.: The Underground Economy of Spam: A Botmaster's Perspective of Coordinating Large-Scale Spam Campaigns. In: USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET). (2011)
- [15] Mushtaq, A.: Smashing the Mega-d/Ozdok botnet in 24 hours. <http://blog.fireeye.com/research/2009/11/smashing-the-ozdok.html> (2009)

- [16] Krebs, B.: Takedowns: The Shuns and Stuns That Take the Fight to the Enemy. *McAfee Security Journal* (6) (2010)
- [17] Cranton, T.: Cracking Down on Botnets. [http://blogs.technet.com/b/microsoft\\_blog/archive/2010/02/25/cracking-down-on-botnets.aspx](http://blogs.technet.com/b/microsoft_blog/archive/2010/02/25/cracking-down-on-botnets.aspx) (2011)
- [18] Krebs, B.: Researchers Kneecap Pushdo Spam Botnet. <http://krebsonsecurity.com/2010/08/researchers-kneecap-pushdo-spam-botnet/> (2010)
- [19] Boscovich, R.: Taking Down Botnets: Microsoft and the Rustock Botnet. [http://blogs.technet.com/b/microsoft\\_blog/archive/2011/03/17/taking-down-botnets-microsoft-and-the-rustock-botnet.aspx](http://blogs.technet.com/b/microsoft_blog/archive/2011/03/17/taking-down-botnets-microsoft-and-the-rustock-botnet.aspx) (2011)
- [20] Stone-Gross, B., Moser, A., Kruegel, C., Almaroth, K., Kirda, E.: FIRE: Finding Rogue Networks. In: Annual Computer Security Applications Conference (ACSAC). (2009)
- [21] Bayer, U., Kruegel, C., Kirda, E.: TTAalyze: A Tool for Analyzing Malware. In: Proceedings of the 15th European Institute for Computer Antivirus Research (EICAR 2006) Annual Conference. (2006)
- [22] Willems, C., Holz, T., Freiling, F.: Toward Automated Dynamic Malware Analysis Using CWSandbox. *IEEE Symposium on Security and Privacy* 5(2) (2007)
- [23] Wurzinger, P., Bilge, L., Holz, T., Goebel, J., Kruegel, C., Kirda, E.: Automatically generating models for botnet detection. In: Proceedings of the 14th European conference on Research in computer security (ESORICS). (2009)
- [24] Perdisci, R., Lee, W., Feamster, N.: Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces. In: USENIX conference on Networked Systems Design and Implementation (NSDI). (2010)
- [25] Rieck, K., Schwenk, G., Limmer, T., Holz, T., Laskov, P.: Botzilla: detecting the "phoning home" of malicious software. In: Proceedings of the 2010 ACM Symposium on Applied Computing (SAC). (2010)
- [26] Jacob, G., Hund, R., Holz, T., Kruegel, C.: JACKSTRAWs: Picking Command and Control Connections from Bot Traffic. In: USENIX Security Symposium. (2011)
- [27] Bayer, U., Milani Comparetti, P., Kruegel, C., Kirda, E.: Scalable, Behavior-Based Malware Clustering. In: Network and Distributed System Security (NDSS). (2009)
- [28] Bellard, F.: QEMU, a Fast and Portable Dynamic Translator. In: USENIX Annual Technical Conference. (2005)
- [29] Roesch, M.: Snort: lightweight intrusion detection for networks. In: USENIX Systems Administration Conference (LISA). (1999)
- [30] Mushtaq, A.: World's Top Malware. [http://blog.fireeye.com/research/2010/07/worlds\\_top\\_modern\\_malware.html](http://blog.fireeye.com/research/2010/07/worlds_top_modern_malware.html) (2010)
- [31] Williams, J.: Bredolab Takedown, Another Win for Collaboration. <http://blogs.technet.com/b/mmpc/archive/2010/10/26/bredolab-takedown-another-win-for-collaboration.aspx> (2010)
- [32] Rashid, F.Y.: Harnig Botnet Goes Offline After Rustock Raid. <http://www.eweekurope.co.uk/news/harnig-botnet-goes-offline-after-rustock-raid-25588> (2011)
- [33] Wilcoxon, F.: Individual Comparisons by Ranking Methods. *Biometrics Bulletin* 1(6) (1945) 80–83
- [34] Kolbitsch, C., Holz, T., Kruegel, C., Kirda, E.: Inspector Gadget: Automated Extraction of Proprietary Gadgets from Malware Binaries. In: IEEE Symposium on Security and Privacy. (2010)
- [35] Chen, X., Andersen, J., Mao, Z.M., Bailey, M., Nazario, J.: Towards an Understanding of Anti-Virtualization and Anti-Debugging Behavior in Modern Malware. In: IEEE International Conference on Dependable Systems and Networks (DSN). (2008)
- [36] Dinaburg, A., Royal, P., Sharif, M., Lee, W.: Ether: Malware Analysis via Hardware Virtualization Extensions. In: ACM Conference on Computer and Communications Security (CCS). (2008)
- [37] Kang, M.G., Yin, H., Hanna, S., McCamant, S., Song, D.: Emulating Emulation-Resistant Malware. In: Proceedings of the 2nd Workshop on Virtual Machine Security (VMSec). (2009)
- [38] Balzarotti, D., Cova, M., Karlberger, C., Kruegel, C., Kirda, E., Vigna, G.: Efficient Detection of Split Personalities in Malware. In: Network and Distributed System Security Symposium (NDSS). (2010)
- [39] Johnson, N.M., Caballero, J., Chen, K.Z., McCamant, S., Pooankam, P., Reynaud, D., Song, D.: Differential Slicing: Identifying Causal Execution Differences for Security Applications. In: IEEE Symposium on Security and Privacy. (2011)
- [40] Lindorfer, M., Kolbitsch, C., Milani Comparetti, P.: Detecting Environment-Sensitive Malware. In: Recent Advances in Intrusion Detection (RAID). (2011)
- [41] Bailey, M., Cooke, E., Jahanian, F., Xu, Y., Karir, M.: A survey of botnet technology and defenses. Conference For Homeland Security, Cybersecurity Applications and Technology (2009) 299–304
- [42] Gu, G., Zhang, J., Lee, W.: Botsniffer: Detecting botnet command and control channels in network traffic. In: Proceedings of The 15th Annual Network and Distributed System Security Symposium (NDSS 2008). (2008)
- [43] Krebs, B.: Naming and Shaming 'Bad' ISPs. <http://krebsonsecurity.com/2010/03/naming-and-shaming-bad-isps/> (2010)
- [44] Godefroid, P., Klarlund, N., Sen, K.: Dart: directed automated random testing. In: Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation. PLDI 05 (2005) 213–223
- [45] Cadar, C., Ganesh, V., Pawlowski, P.M., Dill, D.L., Engler, D.R.: Exe: automatically generating inputs of death. In: Proceedings of the 13th ACM conference on Computer and communications security. CCS 06 (2006) 322–335
- [46] Wilhelm, J., Chiueh, T.: A Forced Sampled Execution Approach to Kernel Rootkit Identification. In: Symp. on Recent Advances in Intrusion Detection (RAID). (2007)
- [47] Brumley, D., Hartwig, C., Liang, Z., Newsome, J., Pooankam, P., Song, D., Yin, H.: Automatically identifying trigger-based behavior in malware. In: Botnet Detection. (2008)
- [48] Sharif, M.I., Lanzi, A., Giffin, J.T., Lee, W.: Impeding malware analysis using conditional code obfuscation. In: Network and Distributed System Security (NDSS). (2008)
- [49] Milani Comparetti, P., Salvaneschi, G., Kirda, E., Kolbitsch, C., Kruegel, C., Zanero, S.: Identifying Dormant Functionality in Malware Programs. In: IEEE Symposium on Security and Privacy. (2010)