The case for in-the-lab botnet experimentation: creating and taking down a 3000-node botnet

Joan Calvet, Carlton R. Davis, José M. Fernandez, Jean-Yves Marion, Pier-Luc St-Onge, Wadie Guizani, Pierre-Marc Bureau, Somayaji Anil

To cite this version:

Joan Calvet, Carlton R. Davis, José M. Fernandez, Jean-Yves Marion, Pier-Luc St-Onge, et al.. The case for in-the-lab botnet experimentation: creating and taking down a 3000-node botnet. Annual Computer Security Applications Conference, Dec 2010, Austin, Texas, United States. inria-00536706

HAL Id: inria-00536706
https://hal.inria.fr/inria-00536706

Submitted on 18 Nov 2010

The case for in-the-lab botnet experimentation: creating and taking down a 3000-node botnet

Joan Calvet[1,2], Carlton R. Davis[1], José M. Fernandez[1], Jean-Yves Marion[2], Pier-Luc St-Onge[1], Wadie Guizani[2], Pierre-Marc Bureau[3], and Anil Somayaji[4]

[1]École Polytechnique de Montréal, Montréal, QC, Canada
{joan.calvet|carlton.davis|jose.fernandez|pier-luc.st-onge}@polymtl.ca

[2]Nancy University - LORIA, Nancy, France, {marionjy|guizaniw}@loria.fr

[3]ESET, San Diego, CA, USA, pbureau@eset.com

[4]Carleton University, Ottawa, ON, Canada, soma@ccsl.carleton.ca

ABSTRACT

Botnets constitute a serious security problem. A lot of effort has been invested towards understanding them better, while developing and learning how to deploy effective counter-measures against them. Their study via various analysis, modelling and experimental methods are integral parts of the development cycle of any such botnet mitigation schemes. It also constitutes a vital part of the process of understanding present threats and predicting future ones. Currently, the most popular of these techniques are "in-the-wild" botnet studies, where researchers interact directly with real-world botnets. This approach is less than ideal, for many reasons that we discuss in this paper, including scientific validity, ethical and legal issues. Consequently, we present an alternative approach employing "in the lab" experiments involving at-scale emulated botnets. We discuss the advantages of such an approach over reverse engineering, analytical modelling, simulation and in-the-wild studies. Moreover, we discuss the requirements that facilities supporting them must have. We then describe an experiment in which we emulated a close to 3000-node, fully-featured version of the Waledac botnet, complete with a reproduced command and control (C&C) infrastructure. By observing the load characteristics and yield (rate of spamming) of such a botnet, we can draw interesting conclusions about its real-world operations and design decisions made by its creators. Furthermore, we conducted experiments where we launched sybil attacks against the botnet. We were able to verify that such an attack is, in the case of Waledac, viable. However, we were able to determine that mounting such an attack is not so simple: high resource consumption can cause havoc and partially neutralise the attack. Finally, we were able to repeat the attack with varying parameters, in an attempt to optimise it. The merits of this experimental approach is underlined by the fact that it is very difficult to obtain these results by employing other methods.

ACSAC '10 Dec. 6-10, 2010, Austin, Texas USA

1. INTRODUCTION

Botnets constitute one of the most worrying computer security threats. Practically all Internet users have experienced the ill effects of botnets, whether by receiving large volumes of spams daily, having their confidential information stolen, lost access to critical Internet services, etc. Botnets are complex and large distributed systems consisting of several thousands, and in some cases, millions of computers. In order to develop a good understanding of such a distributed system and gain insights on its vulnerabilities and weaknesses, it is necessary to study the system as a whole. To that purpose, efforts need to be made to understand how the various parts of the system interact, and in particular how the size and scale of such systems affect their performance.

While analysis by reverse engineering of botnet binaries can initially help us better understand them, it does not always provide the "big picture" in terms of botnet operations. This is because its other parts might not be visible or accessible. Beyond that, we can partially increase our understanding by observing and analysing in-the-wild botnets as a whole, giving indirect visibility of some of these inaccesible components. Studies such as [6, 15, 16, 18, 22] conducting experiments with in-the-wild botnets, have contributed to furthering understanding of botnets. Nonetheless, this method can be problematic, owing to the following: (i) In order to experiment with in-the-wild botnets, researchers need to create entities which join the botnets and perform the tasks the researchers stipulate. If a significant number of entities is added to a botnet, it is possible that the botnets operators will detect the presence of these entities, and possibly implement counter-measures to protect their botnets, and in so doing, potentially shift the botnet arms race further in their favour. On the other hand, if the number of such entities introduced constitutes only a small portion of the overall botnet, we might not be able to accurately observe or predict at-scale effects. (ii) There are legal and ethical issues involved in performing in-the-wild botnet research; for example, in some jurisdictions (particularly in Europe), it is considered unethical and even illegal to create entities that join a botnet, despite the fact that their purpose might be to disrupt the botnet. There are also potential risks involved in doing in-the-wild botnet research: some researchers who investigated botnets have reported that their domains have received distributed DoS attacks from the botnets [12, 21]. (iii) It is difficult to get statistically significant results for experiments involving in-the-wild botnets. Values that are ascertained for variables via a single experiment run—which often require several weeks or months to complete—may be outliers rather than being representative values. In principle, the only way to guarantee that the results are statistically significant is to repeat the experiment multiple times until the standard deviation of the values are within acceptable limits. As highlighted above, it may be undesirable or even counter-productive to perform an experiment on in-the-wild botnets multiple times. Nonetheless, statistical significance is very important because the changing conditions of the environment (churn of infected population, actions by humans, etc.) could give the appearance that, for example, a mitigation strategy is effective, even though the experimenter just happened to be "lucky" at the time of the experiment. (iv) Since in-the-wild botnets experiments are not controlled and (normally) cannot be repeated, they do not allow us to explore the full design space and potential choice of parameters, for example, those related to mitigation strategies being developed. Thus, the solution tested and validated in a single in-the-wild experiment could be far from optimal. For example, a failed experiment because of an unlucky bad choice of parameters could lead us to believe that a promising approach will not work, and cause us to prematurely abandon it.

Simulation studies and analytical modelling have also been employed for botnet investigations. Analytical models are often complex, and all but the more simplistic models are hard to understand and resolve. While mathematical models like Markov chains [3] and immunological equations have been used for other kinds of malware, e.g. worms, botnet-specific analytical modelling has been less common, either addressing propagation properties [8], performance of the C&C infrastructure using graph theory [11, 25], or other techniques [19]. Simulation results, on the other hand, are more accessible and can be obtained by using ready-made network simulators such as Opnet/Omnet, ns2, etc. and adapting them to specific protocols, or by home-coding special purpose discrete-event simulators tailored to model a particular botnet (e.g. the Kademlia/Storm simulator in [9, 10], and generic botnet models in [7, 23]). However, while it is easier to more precisely measure these performance criteria in simulations, this approach has the disavantadge that all aspects of the botnet must either be modelled and implemented, or simply modelled away and ignored. This is particularly problematic for two reasons. First, except for finer-grained, network-based simulators (and even then), it is hard to model and appropriately reproduce the network transmission characteristics of the Internet. Second, it is also quite hard to model and simulate the behaviour of the universe of infectable machines and users, which is particularly important in understanding the "churn" within the botnet due to infection/disinfection, power-on-power-off cycles, etc.

For these reasons, at-scale emulation studies, where conditions as close as possible to the real-world are reproduced in a controlled environment, are perhaps the best alternative to in-the-wild studies. Emulation studies allow controlled repetition of the experiments to see whether variations in environmental parameters, whether these are controlled (by experiment design) or uncontrolled (but measurable) variables, significantly affect performance results. Moreover, they are paramount in threat prediction research, in that they allow us to safely explore the botnet design space in scenarios where the botnet operating parameters have been optimised, something that would be unthinkable with in-the-wild experimentation. On the other hand, in comparison with simulation studies, where artificial models are used in lieu of real botnet entities, in emulation experiments, botnet entities that are either identical or slightly adapted versions of their real-world counterparts, are executed in controlled environments. While this at-scale approach requires large amount of system resources and experimental preparation efforts, it is worth pursuing due to its many advantages. First, as mentioned above, this approach allows researchers to have greater control over the experimental environment; consequently, more thorough investigation encompassing greater variation of experiment parameters can be undertaken. Second, botnet emulation experiments can provide information about botnets that would be very difficult or virtually impossible to ascertain via in-the-wild studies, via simulation experiments, or via reverse engineering analyses. Third, evaluating botnet mitigation schemes using emulated botnets rather than in-the-wild studies, allows researchers the privilege of hiding their ammunition from botnets operators, until the mitigation schemes are fully developed and optimised, at which point, the schemes can be made available to appropriate authorities or those who feel "the calling" and have the resources, and clout or mandate to overtly go on the offence against botnets. Fourth, in addition to facilitating more thorough evaluation of botnet mitigation schemes (as highlighted above), emulation studies can be conducted in significantly less time than in-the-wild studies. Therefore, with this approach, security researchers and practitioners can be more effective and proactive in the fight against botnets. Finally, at-scale botnet emulation provides an avenue for investigating botnets that does not present the same level of legal and ethical issues involved in actively investigating botnets in-the-wild.

For all of these reasons, we jointly endeavoured to develop a different approach for conducting such at-scale, in-the-lab botnet emulation experiments, as an alternative to these other methods of botnet analysis. In this paper, we introduce the philosophy, methodology and tools of this approach, and present a case study involving a particular botnet. The experiment described herein involved recreating in the lab an isolated version of the Waledac botnet [4, 20] consisting of approximately 3,000 nodes, and further, testing and validating a mitigation scheme against it (sybil attack), that we had theorised was possible in such previous work. The specific contributions of the paper are the following: (i) we introduce and show the feasibility of recreating and studying isolated at-scale botnet in a secure environment, (ii) we provide the first significant evidence that the Waledac botnet is vulnerable to sybil attack by demonstrating it in the lab, (iii) we illustrate how such emulated botnets can be used to validate, refine and optimise botnet takedown mechanisms, and (iv) we illustrate how at-scale experimentation of this type can be used to obtain otherwise unaccessible information by revealing previously unknown details about the non-visible components and design decisions taken by Waledac creators and operators.

The rest of the paper is structured as follows. Section 2 describes some previous work in the construction of experimental platforms supporting botnet research. We then discuss the criteria that this type of platforms should meet in order to support at-scale botnet emulation experiments in a safe and scientifically sound manner. We also describe the testbed and generic tools that we have used to conduct our 3000-node botnet emulation experiment with the Waledac botnet. Waledac itself and the experiment are described in Section 3, where we also discuss the results obtained. This includes both results about the viability of the sybil attack we described in previous work [4] and, more interestingly, some valuable insights regarding Waledac design and operations, that could not have been obtained by other methods. We discuss the relevance of these results with respect to validating this kind of experimental approach in Section 4 and conclude in Section 5 by summarizing our contributions, presenting some limitations of our work and highlighting avenues for future research.

2. BOTNET EMULATION EXPERIMENTS

2.1 Related Work

The idea of using laboratory experimentation facilities for botnet research is not new. PlanetLab [17], Emulab [24], and DETER [2] are popular network testbeds that are based on computers hosted at multiple facilities. DETER in particular is specially geared towards security research. These experimental platforms, though they have proven to be very valuable facilities for researchers, are not that suitable for high risk security experiments, such as botnets emulation, owing to the risk of malware "breaking" through logical barriers and escaping into the wild.

With regards to work related to high risk security experiments, a botnet evaluation environment is described in [1] that is a "plug-in" for Emulab-enabled network testbeds. This work is an initial step in building a scalable laboratory testbed for experiments with botnets, but one of the approaches the authors have used to contain the network traffic within the testbed is to give the nodes unroutable (private) IP addresses, which severely limits the type of experiments that can be run on the testbed. Moreover, they only managed hundreds of malicious bots, thus not allowing at-scale emulation of large modern botnets. Jackson *et al.* [13] use DETER to deploy their System for Live Investigation of Next Generation bots (SLINGbot) which, according to the authors, "enables researchers to construct benign bots for the purposes of generating and characterizing botnet command and control traffic". We took a quite different approach mainly because we wanted to run high risk experiments, e.g. involving real malware binaries, and thus we decided to totally isolate our environment from the Internet.

Finally, John *et al.* [14] created a platform named Botlab which monitors the behaviour of spam-oriented bots. Some of the goals of this work is similar to ours, they are both geared to studying botnets. However, there are significant differences. In their work, real-world in-the-wild botnets are monitored, while in ours a complete botnet is reproduced in an isolated and secure environment.

2.2 Design Criteria

The two computer security research labs involved in this work have both adopted stringent security rules and scientific criteria. This is a requirement in order to be able to conduct safe and relevant experimental security research in general, and botnet emulation experiments in particular. A full description of these facilities and the associated criteria is given in [5]. We reproduce here the criteria that we consider are specifically applicable to botnet emulation experiments.

Highly secured. Malware can be potentially highly contagious and is (by definition) developed for malicious intents. Consequently, experiments involving malware should therefore take adequate precautionary measures to ensure that it is not accidentally released into the wild during the course to the experiment. Perhaps the only way of adequately mitigating risk associated to this threat is for the experiment environment to be completely isolated from the Internet and other networks. Thus we build our emulation platform based on an isolated cluster within highly secured facilities. The physical security of the labs includes strong physical barriers (floor-to-ceiling walls, reinforced doors, etc.), surveillance systems (cameras, motion detectors), a separate access control system using multi-factor authentication. In terms of logical security, the cluster is completely isolated from other computer networks (air gapped).

Scale. We desire to have an emulation platform capable of supporting-at scale experiments; i.e. involving several thousands of bots. The choice followed was to heavily rely on virtualisation. This allowed us to have upwards of 30 virtual bots per physical machine.

Realism. An important requirement of our botnet emulation platform is that it be capable of reproducing botnets that in principle are identical (or close to identical) in functionality to those found in the wild. To achieve this, it is necessary that very few changes (or ideally none at all) be made to the bot binaries that are used to reproduce the botnets. Changes should be restricted to those that are necessary (if any are required) to overcome anti-virtualisation and anti-debugging capabilities in the bot binaries. This constraint necessitates that nodes in the emulation platform be configured with IP addresses that are hard coded in the bots binaries, and that the necessary DNS databases be setup to resolve these addresses.

Flexibility. We desire to have an emulation platform that is capable of reproducing any botnet after the necessary reverse engineering and investigative work has been done to elucidate the structure of the botnet command and control. Therefore, flexibility is an important requirement. The emulation platform should be easily configurable to adapt various overlay network topologies with for example, variable proportions of bots with private (unroutable) IP addresses versus bots with public IP addresses: proportions that mirror those observed in the in-the-wild botnet.

Sterilisability. To guarantee the integrity of the experiments, virtual machines (VM) need to be "sterilised" in order to remove any artifacts associated with the malware infection. In certain cases, this requires removal and re-installation of the VMs. Efficient mechanisms are therefore needed to accomplish these tasks.

2.3 Hardware and Tools

In order to meet these criteria, we used an isolated cluster as our emulation platform. The cluster consists of 98 blades, each having a quad-core processor, 8 Gb of RAM, dual 136 Gb SCSI disks and a network card with 4 separate gigabit Ethernet ports. The blades are contained in two 42U racks. The blades are interconnected with two separate sets of switches (each in their own 42U rack) such that two physically separated networks are created: a) a *control network* used for transmitting commands and data necessary for controlling the experiments, and b) an *experiment network* used for transmitting the experiment traffic, including in this case botnet activity (spam, C&C traffic, etc.). Having two physically separated networks helps to guarantee the integrity of the experiment, in that, commands and data necessary for controlling the experiment can be sent via a separate network. This ensures that the control traffic does not interfere with the transmission of the experiment traffic, thus preserving the validity of timing measurements made on it.

We present a brief overview of the virtualisation and configuration management tools we employed.

Virtualisation: To maximise the versatility and capability of the emulation platform, we sought a feature rich virtualisation technology that is able to emulate both Windows and Linux. Consequently, we choose the VMWare ESX product as the hypervisor for the blades, which allows good efficiency and ease of configuration.

Configuration and management: We used the Extreme Cloud Administration Toolkit (xCAT), an open-source tool, for configuring and managing the emulation platform. xCAT is particularly attractive for this purpose since it contains VMWare functionalities; for example, xCAT can create defined number of VMs with a single command, such as, `mkvm vm[001-098]`, thus creating 98 VMs which are assigned names vm01, vm02, ..., vm098. From a management point of view, xCAT operates as follows. First it requires tables containing host configuration information, including details such as machine template (i.e. location and name of the ghost image), hostname, IP address, etc. These tables can be filled manually using a text editor or they can be generated using perl or any other scripting language. When the tables are filled, xCAT can be issued commands causing the tables to be committed to the xCAT database. It incorporates powerful image deployment, configuration and control commands, that take the information from the database, and use remote boot technology such as PXE or the ESX API, to order hosts to do the required tasks. Thus, the experiment design, deployment and management process for emulated experiments is as follows. First, xCAT tables must be filled to facilitate the deployment and configuration of appropriate host images containing ESX. Following this, the researchers produce an abstract, high-level description of the desired environments, and build necessary VM templates or ghost images (e.g. a VM template for each type of bot, gateways, SMTP servers, etc.). Next, the researchers decides on a network topology, addressing plan and host naming convention. xCAT tables then need to be filled to facilitate the deployment and configuration of these entities (ESX hosts, VMs, and their configurations). Depending on the size of the experiment, xCAT tables can be filled manually or automatically using scripts, regular expressions or a combination of both.

2.4 Experiment Methodology

Generally speaking in order to prepare, design and conduct an at-scale botnet emulation experiment (some or all of) the following steps are followed:

1. Capture of botnet client code, through various methods (honeypot, collaborators, etc.).

2. Gather information on the botnet in order to understand as much as is possible about the botnet architecture and modes of operation. Examples of information that are required are (i) communications protocols and message formats; (ii) authentication process for gaining access to the botnet; (iii) categories of bots and the hierarchical relationship between them; and (iv) C& C architecture. This information can typically be obtained by reverse engineering bots and analysing their communication traffic.

3. Passively monitoring the botnet by observing infected machines and/or joining the botnet with special purpose passive botnet-like programmes (crawlers), in order to continue to gain information on its structure, in particular the C&C infrastructure, including formats of commands, location and characteristics of C&C servers, etc.

4. Construction of a surrogate C&C infrastructure complete with servers and any intermediary proxies.

5. Construction of realistic *operating environment* for the botnet in the lab, including infectable/infected machines (ideally showing human driven-like behaviour), ancillary network services (DNS, SMTP, DHCP, etc.), a realistic emulated network architecture, and, of course, counter-measures and mitigation schemes against it.

6. Determination of metrics to be measured, based on research questions that experiment must answer.

7. Implementation of methods for measuring these metrics and extracting the results in usable form for further analysis.

The main challenges in following this methodology involve:
Maintaining isolation. This means both a) maintaining spatial and logical isolation between the experimental and control components (achieved in our setup through physical separation), and b) maintaining isolation between the whole facility and the outside world (security criterion). In addition, it also means time and logical separation between successive experiment runs (sterilisability criterion).
Observing without interference. This is paramount in order to maintain the scientific soundness of the results, and also to enforce isolation.
Simulating network characteristics and user behaviour. This is a very hard problem that is not just relevant to botnet research. It is essentially a modelling problem combined with significant engineering issues.

3. THE WALEDAC EXPERIMENT

To exemplify this methodology, we now describe how we applied it in constituting an isolated Waledac botnet and launching our sybil attack against it.

3.1 Overview of Waledac

Waledac is a prominent botnet which first appeared in November 2008, shortly after the Storm botnet became inactive. Waledac employs a "home grown" peer-to-peer (P2P) network infrastructure for its C&C. Other researchers and members of our group [4, 20] had previously reversed engineered the Waledac binary and obtained details about its mode of operation. Here, we will limit our description of Waledac to the aspects relevant to the goals of this research, i.e. disruption of its C&C through sybil attacks.

Waledac botnet uses a four layered C&C architecture. The first layer contains bots that are referred to as *spammers*. These are machines with private IP addresses residing behind Network Address Translator (NAT) devices. Spammers are essentially the "worker" bots and constitute approximately 80% of the botnet. Their principal role is to send spam, harvest email addresses from files stored on the infected machines, and harvest confidential information (e.g. usernames and passwords) from the network traffic that traverses the infected machines.

Waledac binaries are hardcoded with a list consisting of 100 to 500 contact information of *repeaters*. This list—which is referred to as a *RList*—is stored in XML format in a registry key. An *RList* has a global Unix timestamp and between 100 to 500 records that contain the following fields: a 16-byte ID, an IP address, a port number, and a Unix timestamp. The list is sorted in descending order of timestamp (oldest at the top of the list). The *RLists* play a key role in facilitating Waledac operations and maintaining the P2P infrastructure, in the following ways: (i) The *Rlist* allows each node to "know" a small subset of the botnet nodes. A spammer, for example, contacts the repeaters in its *RList*, at frequent intervals, and request "jobs" (i.e. tasks to perform). The repeater will in turn, forward the job request to a *protector*, which will subsequently forward the request to the C&C server, and relay the response (the work order) to the repeater, which will issue the work order to the spammer. (ii) The *RList* provides a means of propagating identification information of repeaters which recently join the botnet. This process is facilitated as follows. All bots regularly send update messages to their known repeaters. For a bot S that wishes to send an update message we have two possibilities: a) if S is a spammer, it extracts 100 records from its *RList* and sends this extract to a randomly selected repeater; and b) if S is a repeater, it selects 99 records from its *RList*, adds its own record (containing its identification information and the current timestamp) to the top of the list, and sends the list to the selected repeater. When the recipient bot R receives the list, it reciprocates the process, by sending a list containing 100 records of repeaters it knows of, back to bot S. The recipient of an update list uses the list to update its *RList*, as indicated below in Section 3.3.

In addition to an *RList*, each repeater also has a cryptographically-signed protector list, containing identification information on the protectors. The repeaters regularly exchange signed protector lists.

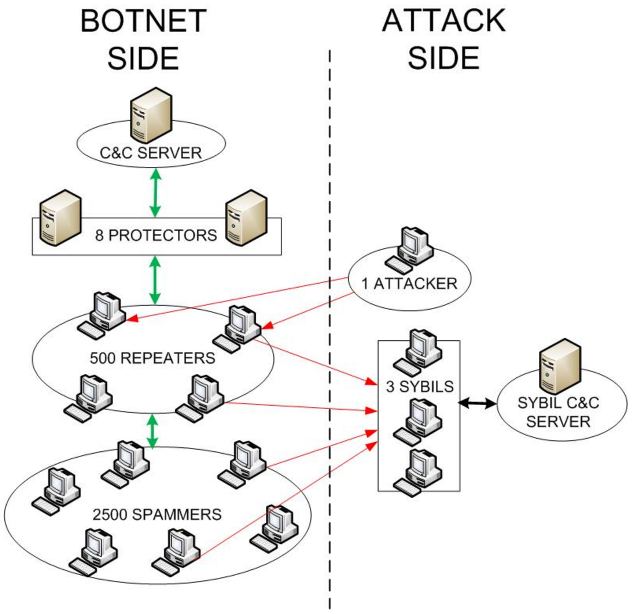


Figure 1: Experimental setup

The private key for signing such lists is known only to the C&C server, and the public key certificate for verifying them is embedded in the bot code. Note that it would be nonsensical to sign *RLists* since any bot (even if infiltrated) must be able to provide them. More interestingly, commands from the server are not signed either, something that could provide some level of protection against a sybil attack. However, the traffic between the server and the bots is encrypted with AES-128, using a key that is chosen by the server (and was probably meant to be a session key).

Waledac also provide a failback mechanism that allows bots to maintain connection to the botnet even if the repeaters listed in its *RList* are not reachable. The failback mechanism works as follows: if a bot makes 10 consecutive unsuccessful attempts to contact a repeater, the bot connects to a HTTP server (the URL for the servers are hardcoded in the Waledac binary) and download an updated *RList*. These lists are updated every 10 minutes on the server, so that they contain the most recently "heard of" repeaters.

3.2 Waledac emulation

The overall setup and architecture for our emulation experiment involving a contained Waledac botnet is depicted in Figure 1. The process we employed to constitute it is as follows:

(*i*) *Create VM templates*. First, we installed the binaries on Windows XP VMs and created xCat VM templates associated with them. We created separate templates for spammer and repeaters.

(*ii*) *Add the IP addresses of 500 repeaters to the RLists*. We deleted the entries in the original *RLists*, and added the identification information of the 500 repeaters we used for the experiment.

(*iii*) *Add script to issue commands to the VMs*. We created a Python script and added it to the VM template. This script allows us to issue commands to the VM, for example, to start and stop execution of the Waledac binaries, to clean the VMs, to restore the *RList* to its initial state, and delete the *RList* dumps (see Section 3.4).

(*iv*) *Deploy the VM templates*. Next, we utilised xCAT to install the VM templates on the blades (approximately 30 VMs per blade).

(*v*) *Setup C&C server*. Through our in-the-wild investigation of Waledac, we were able to determine the type and the format of the messages the C&C server sends to the bots in response to those it receives from them. The server code is a Python script that is capable of responding to all such requests in a similar manner as the original C&C server. For example, we observed that spam orders issued to bots contain between 500 and 1,000 email addresses. We mimic this functionality by creating five different spam order messages, each containing between 500 and 1,000 addresses and programmed the C&C to send them to bots requesting spam jobs. We also implemented the failback scheme as follows: every 10 minutes the script creates an *RList* containing the identification information of the most active repeaters; this list is placed on a HTTP host. All HTTP requests from the Waledac binaries to the hardcoded failback domains are directed to this host. As is the case for Waledac C&C server, our C&C server utilises 1024-bit RSA and 128-bit AES keys to provide confidentially services for the messages the C&C server and the bots exchange. The server runs on a VM that is the only one to run on that blade.

(*vi*) *Constitute the botnet*. Finally, we issue commands to the VMs to start running the Waledac binaries. We can similarly stop and re-start the experiment at will. The botnet we constituted for our experiment consisted of 500 repeaters, 2,300 spammer, 8 protectors and the C&C server (for a total of 2,809 nodes in the botnet). This proportion is close to that which is observed in the wild for Waledac. It should be noted that the protectors we created are actually components of the C&C server, and not separate machines: we assigned 8 network interfaces—each with a different IP address—to the C&C server for this purpose. These addresses are set to be identical to those of the real Waledac protectors. This was necessary because the protectors identification information hardcoded in the bots binaries is signed and we have no knowledge of the corresponding private key.

(*vii*) *Setup environment*. In addition to the blades in the cluster, we used standalone Linux machines to setup the ancillary infrastructure needed for the botnet to run. These standalone machines provide services, such as DNS, SMTP and DHCP, that would normally be present in the Internet. They constitute a simple reproduction of part of the "environment" within which the real botnet would operate. These machines were of course connected to the experiment network of the cluster.

3.3 Mitigation scheme and implementation

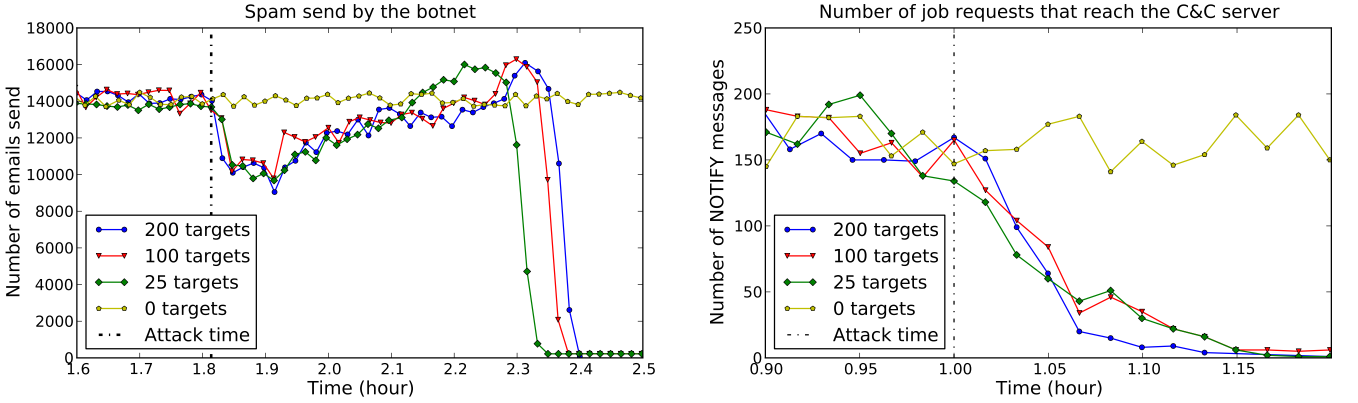
By reverse engineering the Waledac code and analysing its network traffic, we had previously conjectured [4] that Waledac was vulnerable to sybil attacks, due to characteristics of the home-made P2P protocol it uses for C&C. In addition, because the IP address of a bot needs not be unique (bots are primarily identified by their 16-byte ID), it is possible to generate large number of sybils—with unique IDs but with the same IP address—whilst using few machines, thus making this attack relatively easy to mount.

We indicated in Section 3.1 that bots use the update messages they receive, containing a 100-entry extract of the sender's *RList*, to update their *RList*. For each entry i in the update list, the recipient computes a new timestamp:

$$\mathit{NewTS}_i = \mathit{CurrentTS} - |\mathit{UpdateTS} - \mathit{TS}_i|$$

where *CurrentTS* is the current timestamp, *UpdateTS* is the list's global timestamp and TS_i is the timestamp of the entry. The recipient then replaces the timestamp and inserts the entry in its *RList* at the correct location: recall that an *RList* is sorted in descending order of timestamp. All entries beyond position 500 are deleted.

By analysing the binarires, we also discovered that Waledac bots do not check the update lists they receive to determine if they contain more than 100 records. It is therefore possible to craft special



(a) Spam output before and after sybil attack

(b) Job requests arriving at the C&C server before and after attack

Figure 2: Botnet activities before and after sybil attack being launched.

update messages that will cause all the entries in the *RLists* of the recipients to be entirely replaced by the sybil records. This can easily be accomplished by placing 500 sybil records in the update list and set the timestamp of all the sybil entries, to a value that is identical to the list's global timestamp. This guarantees that the *NewTS* for all the sybil entries will be equal to the current timestamp; consequently these 500 sybil entries will be placed at the top of the recipient's *RList* and all the others will be deleted.

The extent to which a bot can be controlled and isolated when it receives such a message, depends on the type of bot.

If the bot is a repeater. After receiving the message from the sybil, a race condition situation arises. Since the repeater's identity information is likely to be in other bots' *RLists*, they are likely to send the repeater update messages whose entries could replace some of the sybil entries in its *RList*. To maximise the chances that the repeater remains completely isolated, the sybils need to continue sending update messages to the repeater at short time intervals.

If the bot is a spammer. In this case the result of the sybil attack is more effective, since the spammer cannot be contacted directly. When it receives an update from a sybil, and the entries in its *RList* are consequently completely replaced by sybil entries, the spammer will become completely isolated from other bots. It should be noted though, that in order to infiltrate a spammer's *RList*, the sybils first need to infiltrate the *RLists* of repeaters whose identity information are in the *RList* of the spammer. Also, in order to mantain isolation, the sybils need to remain active until all the Waledac domains that are encoded in the affected bots are disactivated, otherwise the bots will resort to the failback mechanism we described in Section 3.1, and download a "clean" *RList* from a Waledac HTTP server.

For the sybil attack implementation, we used three separate entities: (a) fake repeaters (sybils), (b) attackers, and (c) a sybil C&C server. The role of the sybils is to passively respond to update messages—sent to sybils—with responses containing the specially crafted update message we described above, whereas the attackers' role is to target specified numbers of repeaters and send them the specially crafted update messages. The records in the update messages all contain sybil ID information. The attackers send these messages to the targeted repeaters once every minute. This rate was utilised because we observed that the Waledac bots in-the-wild send between 2 to 5 update messages during a two-minute time period. The role of the sybil C&C server is to prevent the "turned" bots from resorting to the failback mechanism. We indicated in Section 3.1 that if a bot makes 10 consecutive unsuccessful attempts to contact repeaters, the bot will connect to a Waledac Web server and download an updated *RList*. In order to prevent this from happening, the sybils are programmed to relay all messages from the turned bots to the sybil C&C server. The sybil C&C server will in turn send harmless spam orders to the turned bots. We use the following feature of Waledac to issue these harmless orders: spammers are supplied with a special SMTP server IP address that they are required to use to test if they are capable of sending spams. Before sending spam, spammer try to connect to this special SMTP server and send spam only if they succeed. We therefore send the bots the address for an "SMTP server" that is unreachable. In so doing, we can ensure that the bots the sybils control, do not send spam.

We employed three VM to host the sybils, one VM to launch the attackers and another VM to run the sybil C&C server.

3.4 Experiment results

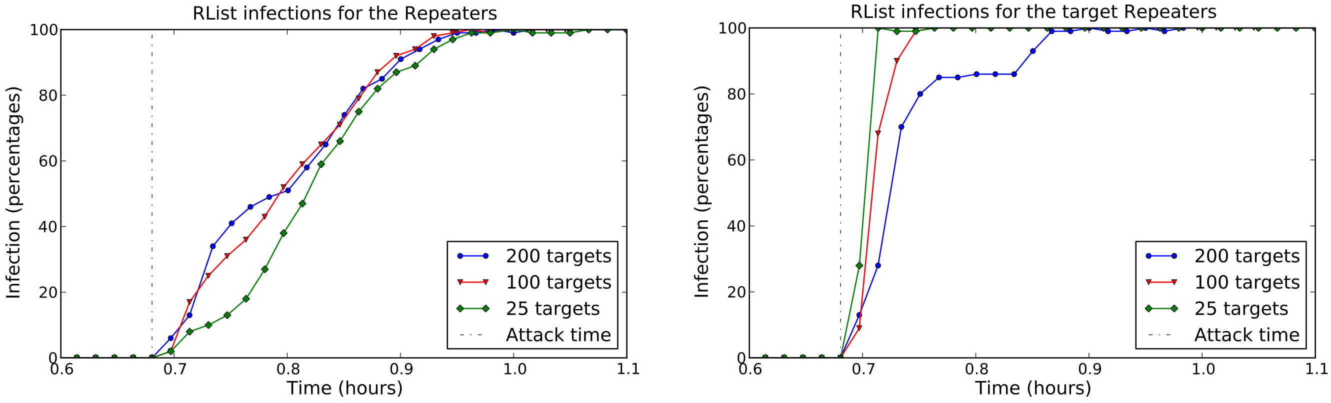
We utilised the following metrics for assessing the effectiveness of the sybil attack mitigation scheme.

Spam output. We measure the spam output of the botnet, over a fixed time period, before and after we launch the attack. To facilitate spam output measurement, we programmed our botnet C&C server to send spam orders with email addresses belonging to the same domain. This allows us to more easily count the number of spam sent by counting DNS requests to that domain.

Connectivity of the botnet. In order to determine the extent to which the sybil attack affects the connectivity of the botnet, we measure the number of NOTIFY messages the C&C server receives over a fixed time period, before and after we launch the attack. NOTIFY messages are the second message that a bot sends when it dialogues with the C&C server. Counting only NOTIFY messages allows us to filter out noise due to failed connection requests.

Percentage of sybils in *RList*. The goal of the attack is to replace the entries in *RLists* with sybil records, which will utlimately isolate the bots. This parameter is thus an intermediary indicator of effectiveness. We measure it by way of a Python running on the bots, that dumps *RList* to a file each time it is modified, and send these files to an FTP server via the control network. We then analyse these files to determine the percentages of sybils in the *RList*.

In addition to measuring the above metrics, we also wish to determine the degree of success of the attack when subsets of known



(a) Percentage *RList* infection for all repeaters

(b) Percentage *RList* infection for targeted repeaters

Figure 3: Percentage *RList* infections by sybils.

repeaters are targeted, vice targeting all known repeaters. We consequently performed 3 sets of experiments, targeting 200, 100 and 25 repeaters, respectively.

The first set of experiments was used to benchmark the botnet. We performed 3 experiment runs; for each run, we allow the botnet activities to reach a steady state, then we measure spam output and botnet connectivity. The average values—for the 3 experiment runs—for these measures were 13,200 emails per minutes and 120 NOTIFY messages per minute, respectively.

The next set of experiments assessed the efficacy of the sybil attack by determining the steady state values of the above metrics, before and after the attack begins. We performed 3 experiment runs for each set of experiments and compute the average values for the the runs. As indicated in Table 1, the standard deviation values obtained are relatively small, and we therefore believe that these results are statistically significant.

Figure 2(a) shows the spam output of the botnet before and after the sybil attack begins. The dotted vertical line indicates the time that the attack begins. The graph shows that the attack is a success and that the spam output drastically falls after less than an hour. We also observe that after the sybil attack begins, there is an initial decrease in the spam output, with a subsequent gradual return to its original level, and even rising significantly above it, before the final fall. Furthermore, we can see that the time taken to reach the final fall is longer with 200 and 100 targets than it is with 25, which is also not so intuitive.

We think the main cause of these two surprising and interesting effects is the load on the C&C servers, both the sybil and the malicious one. The C&C servers are overloaded and cannot keep up with the computing time required for cryptographic operations. As mentioned in [4], the Waledac botnet uses RSA with 1024 bits keypairs and AES-128. Through our observation of the Waledac botnet in-the-wild, we discovered that the C&C server used the same AES session key for all bots, for approximately 10 months. We initially thought that this was a design error made by the botnet creators, but when implementing the Waledac C&C server we discovered that it was not impossible to generate a session key for each bot, because it overloads the server with cryptographic computation. Waledac bots are too verbose and if a good availability of the C&C server is desired, there is no choice but to keep the same session key for all active connections (at least for several minutes) and give bots the same set of encrypted orders. Hence it is likely that this was no mistake, but rather a conscious design choice by Waledac creators.

However, as the sybil attack progresses, the sybil C&C server has fewer cryptographic operations to perform because we use exactly the same strategy as the Waledac C&C server in-the-wild: we use the same session key and pre-encrypt the work orders in batch mode before they need to be sent. Thus as the attack progresses the sybil C&C server availability does not decrease too significantly since it does not have to encrypt any orders, hence allowing it to control an increasing number of bots, and adequately handle their requests.

It is important to note that there is a delay between the moment we completely infect a spammer's *RList* and the moment it stops sending spam: a bot will not contact the C&C server until it has finished its current task.

As the attack progresses and we gain a hold within the botnet, we decrease the load on the real C&C server, which becomes more available for the non-poisoned bots. Thus, these bots receive orders every time they ask (which is not a normal situation, even in-the-wild) and continue to spam in a more efficient way than in a normal state. During that short interim time period, the botnet is more efficient under attack than in its normal state. It should be noted that if we had given more resources to the C&C server to start with, this effect would probably be less important, but as we attributed 4 processors and 8 Gb RAM for its VM, we think it is realistic to assume that this effect would also be observed in-the-wild.

Figure 2(b) shows the number of NOTIFY messages the C&C server receives before and after the sybil attack begins. The number of NOTIFY messages the C&C server receives is essentially a measure of the connectivity of the botnet. The figure indicates that, as expected, there was a gradual decrease in the number of messages that arrive at the C&C server, after the sybil attack commences. After the attack begins, the more efficient attack is the one with 25 targets. This is also a consequence of the load on the sybil C&C server. Because it is overwhelmed with the more aggressive attacks, it refuses connections. After a transition period, the 100 and 200 targets attacks become eventually more effective as the sybil C&C server has fewer cryptographic computation to perform.

Figure 3(a) shows the percentage of sybils entries in all repeaters *RList* (targets and non-targets). After an initial transitory phase where the more aggressive attacks (more targets) seem more efficient, we reach a stage of equivalent linear growth in the number of sybil-controlled machines in the *RLists*, This is due to the fact that propagation of sybil records in the *RList* of non-sybil, non-targeted bots is dependent on the rate of *RList* updates between non-targeted real bots, which is the same for all attacks.

Figure 3(b) shows the percentage of poisoning on the targeted repeaters only. We can observe that it is quicker to fully control 25 direct targets than 100 or 200, because of the race condition faced by these direct targets: the more bots we target, the higher the chance that we lose some races and have sybil records replaced by real ones.

4. DISCUSSION

As previously discussed, at-scale botnet emulation in the lab displays several advantages with respect to other analysis methods. The Waledac experiments that we have conducted exemplifies the viability of this approach and provides clear indications of some of these advantages.

First, we were able to assess the efficacy of the mitigation scheme directly by measuring the following three parameters: (i) the number of NOTIFY messages arriving at the C&C server within a given time period, (ii) the number of spam sent within a given time period; and (iii) the penetration ratio of sybil identification within the bots peer lists (*RLists*). These parameters provide the most effective means of measuring the connectivity and productivity of the botnet. Whereas we were easily able to measure these parameters via such botnet emulation experiments, the value of these parameters are virtually impossible to ascertain—particularly items (i) and (iii)—via in-the-wild botnet studies.

Second, we were able to address and answer questions about attack optimisation. In particular, in our attack the role of fake repeaters (the sybils) is to target a specified subset of repeaters by sending them specially crafted update messages. An important question that needed to be answered regarding the implementation of the mitigation scheme is, what is the ideal number of repeaters to target? It is very difficult to design in-the-wild botnet experiments to find answer to this question. Moreover, even if it were possible to do so, these experiments would likely take several weeks or even months to complete, whereas botnet emulation experiments supply the answer to this question within a few hours.

Moreover, some of the experiment results seem counter-intuitive. They indicate, for example, that targeting higher number of repeaters does not necessarily cause the efficacy of the mitigation scheme to increase. By performing the experiment multiple times and observing the same trend, it became clear that the larger the number of repeaters that are targeted, the more update requests will be sent to the fake repeaters (sybils) that respond to update messages sent to sybils; and if the number of update messages sent to the sybils increase beyond a given threshold, many of these messages will be dropped and consequently will not be serviced. This leads to higher number of repeaters that are under the control of the sybils resorting to the failback mechanism (as outlined in Section 3.1) and download "clean" *RList*, and in so doing, breaking free of the control of the sybils. Becoming aware of this fact is important for a couple of reasons. Firstly, it provides pointers as to how the mitigation scheme can be made more stealthy, since in targeting smaller number of repeaters it is likely that the probability of the botnet operators detecting the presence of the counter-botnet agents in the botnet will decrease. Secondly, this awareness provides indicators as to how the counter-botnet agents can allocate their resources to maximise the efficacy of the counter-botnet operations. In essence, we were able to discover this phenomenon (repeaters re-joining the real botnet because of sybil overload) by running an at-scale botnet emulation experiment where we could observe and note the behaviour of bot clients. Again, it would have been very difficult to notice this by passive in-the-wild botnet observation, unless researchers have machines that join the botnet and play an active role in them (i.e. send spam and support criminal activities), something that many would consider dangerous and questionable. Thus, this constitutes a third example of why at-scale botnet emulation are a necessary tool in botnet research.

Since fake repeaters responding to update messages sent to targeted repeaters were easily overloaded, we can deduce that this task requires more resources than the attackers, whose role it is to send specially crafted update messages to the targeted repeaters in order to place fake repeaters in their *RList*. We did not directly address the question of what would have been the optimal value for the ratio of fake repeaters to attackers, i.e. how to best allocate sybil machines to these roles. However, by running the experiments multiple times, it became clear that the resources we allocated for responding to update messages sent to the fake repeaters were inadequate, since large number of messages that were addressed to the fake repeaters were dropped, simply because the service queue was being filled. Whereas this observation could also have been made via in-the-wild botnet experiments, it is much easier to verify via in-lab experiments such as this.

Finally, the last example has nothing to do not with the attack, but rather with the botnet and the botmasters themselves. By not only directly observing the bot clients but also the reconstructed botnet C&C server, we were able to "wear the shoes" of the botmaster and were thus able to identify some of the performance and design challenges that botnet creators and botmasters must face. In particular, what would have been a textbook solution to ensure data confidentiality and integrity, i.e. the use of unique symmetric keys for each session between a bot and the C&C server, turned out to be non viable due to the size of the botnet. Without at-scale experimentation in the scale of several thousand bots, we would never have discovered this fact. This illustrates that, surprising to some (including some of us!), one can indeed learn a lot about the bad guys even in the lab. In other words, field work is not by itself the end-all of botnet and cyber criminality research.

5. CONCLUSIONS

In this paper we presented an alternative approach for conducting botnet research: at-scale botnet emulation in laboratory conditions. We have discussed its generic advantages with respect to other approaches like analytical modelling, simulation studies, and in-the-wild botnet experimentation. In a nutshell, it provides a greater verifiable realism than analytical models, simulation methods or small-scale emulations, while providing greater levels of control and safety, and presenting fewer ethical and legal problems than in-the-wild experimentation.

In order to deliver such advantages, however, botnet emulations must be run on platforms or testbeds that meet certain criteria. We have postulated and described such necessary criteria. Namely: i) security, to mitigate risks of accidental or unauthorised release of botnet code or information about them; ii) scalability, in order to be able to emulate botnets of large enough size so that similar phenomena as those in a real botnet can be observed; iii) realism, for the same reason; iv) flexibility, so that experiments can easily be repeated, under varying controlled conditions and for different types of botnets and/or mitigation schemes, and v) sterilisability, so that results from previous experiments do not affect that of future ones.

Using the isolated security testbeds based on virtualisation [5], we were able to mount a set of at-scale emulation experiments of the Waledac botnet involving close to 3,000 bots. The controlled conditions of the lab and the full visibility on the botnet and the ancillary infrastructure (the botnet's "operating environment") allowed us to measure performance metrics for both the botnet and

(a) Average (30 min period before attack) and Std deviation values for spam output

Experiment	Average	Standard deviation for the spam output each 2 minutes after the start of the attack														
25 targets	13219	403.27	389.29	102.75	149.51	101.22	528.81	182.02	121.48	211.20	230.79	274.75	169.00	445.75	325.85	395.44
100 targets	13433	180.07	546.72	195.69	327.22	22.61	326.74	271.37	250.24	339.77	338.79	511.07	187.33	171.66	315.68	462.08
200 targets	13582	506.29	348.47	144.76	312.90	769.54	56.01	493.12	239.83	450.14	160.21	662.21	378.59	154.85	406.08	562.96

(b) Average (30 min period before attack) and Std deviation values for job request reaching the C&C

Experiment	Average	Standard deviation for the number of job requests that reach the C&C server														
25 targets	185	23.33	19.09	7.78	3.54	1.41	0.71	7.78	5.66	4.95	0.71	0.71	1.41	0.71	0.00	0.00
100 targets	176	19.22	39.59	21.63	5.03	2.52	3.00	6.03	0.00	0.58	0.00	0.58	0.58	0.00	0.00	0.58
200 targets	172	25.70	12.22	5.51	10.02	16.07	4.58	7.00	2.31	4.04	1.73	0.58	0.58	0.00	0.00	0.58

(c) Std deviation values for percentage *RList* infection for all the repeaters

Experiment	Standard deviation for percentages of *RList* infection each 2 minutes during the transition state														
25 targets	0.00	1.00	2.52	0.58	2.31	2.31	1.53	2.65	2.65	0.00	0.58	0.58	0.00	0.00	0.00
100 targets	0.00	0.50	0.82	2.22	1.50	2.38	1.63	2.63	2.00	1.26	1.00	0.50	0.00	0.00	0.00
200 targets	0.00	1.65	2.08	1.73	1.53	5.00	0.58	2.08	2.65	1.73	1.00	0.58	0.00	0.00	0.58

(d) Std deviation values for percentage *RList* infection for the target repeaters

Experiment	Standard deviation for percentages of *RList* infection each 2 minutes during the transition state														
25 targets	0.00	1.97	2.90	0.71	0.71	1.54	0.00	2.83	0.00	2.83	0.00	2.12	2.12	0.00	1.41
100 targets	0.00	1.53	0.58	2.00	0.58	0.58	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
200 targets	0.00	4.78	3.49	2.87	2.36	2.12	0.00	0.00	0.00	0.00	1.24	1.83	0.00	0.00	0.71

Table 1: Standard deviation values for the experiment runs

attacks agains it that would have been very hard to measure in in-the-wild botnet, such as i) spam yield (i.e. number of spams per minute sent by the bots), ii) botnet activity (i.e. number of NOTIFY messages per minute), and iii) penetration of sybils into the botnet (i.e. percentage of sybils in bot peer lists). The results obtained by measurement of these quantities can be summarised as follows. With respect to the efficacy and viability of the sybil attack, we can conclude that:

1. The sybil attack as implemented is indeed effective and achieves full disruption within an hour.
2. Workload on the sybil C&C server is an important factor to consider, as it creates a transient "window of detection" that could allow the botmaster to detect the attack before he completely loses control of the botnet.
3. It is not necessary to poison the *RList* of all or even many repeaters for the attack to succeed. In fact, targeting smaller numbers of repeaters (as few as 5% of them) yields essentially the same disruption results as wider attacks (targeting up to 40% of the repeaters), while somewhat mitigating the workload problem of the sybil attack C&C server.

With respect to the actual botnet, we were able to deduce the following facts from our results:

4. Workload on the actual C&C server is also a problem. We suspect that this is the real reason why common session keys started to be used 10 months into the botnet deployment, and not due to a programming or design mistake, as was initially suspected. This is also the probable reason why server commands are not signed.

It is very important to note that it would have been very difficult, and in some cases impossible, to reach these same conclusions by resorting to other methods of botnet analysis. While the efficacy of the sybil attack on the real botnet could have been measured by continuous monitoring the attacked botnet, it is unlikely that the attack designer would have had a chance to run several experiments to find out that limited targeted attacks are a better option. In addition, and as mentioned above, testing counter-measures in-the-wild could bave several negative side-effects (retaliation, premature disclosure of mitigation strategies, premature beginning of an arms race, etc.) that could easily outweigh the benefits of such research. In addition, without running an actual C&C server in the lab for an at-scale reproduction of the botnet, it would not have been possible to confirm that the design choices made by the botnet creators were due to performance issues. This cannot be deduced from real-world botnet observation, unless one has gained access to the actual C&C server, a very unlikely proposition.

Thus, we hope to have made a strong case for the use of at-scale botnet emulation as a fundamental tool in botnet research, complementary at least, and superior in many respects to other botnet and counter-measure study techniques. Nonetheless, there are some important limitations to this approach.

First, they require access to testing facilities that meet the above-mentioned criteria; this is unfortunately not the case today for many good and well-established botnet researchers. National and international collaborative efforts like those in which the authors are involved, or the US DETER project are one way to address this. However, even though it is understandable that actual usage of the facilities might be restricted, more collaboration and sharing of procedures, tools and standards would greatly benefit the community as a whole and encourage researchers and research funders to follow that path.

Second, while we were very careful in the fidelity of the botnet emulation portion of our experimental setup, the emulation of the operating environment of the botnet is somewhat simplistic. Aspects of the environment that can be included in that category are: i) a more realistic model and emulation of the Internet (including Layer 3 and below characteristics such as topology, latency, adressing, etc.) as it interconnects the bots, the C&C server and the ancillary infrastructure, ii) a more realistic model describing the natural oscillations in botnet population —also referred to as *churn* or *birth-death process*— due to user action such as infection/disinfection, powering on and off, diurnal usage patterns, etc.

Internet networks and user modelling is another field of research all on its own, and a very hard one at that. Nonetheless, we are currently working on ways to easily and transparently port and implement such given models to our security testbeds, which would allow us to test the impact of changes in network configuration and user behaviour on botnet and counter-measure efficacy. This, we hope, will lead to very fruitful research, as we, the good guys, do in principle control the network and can positively affect user behaviour through education or regulation.

Finally, none of our experiments emulate the behaviour of an important part of the botnet: the botmaster, who deploys and operates the botnet and that has, in principle, clearly defined objectives for doing so (e.g. profit). While it is not as easy to capture the "botmaster code" into the lab as it was for the bot code itself, it would be relatively easy to adapt our botnet emulation to allow for interactive "gaming" of a botmaster vs. botnet attacker scenario, where both are played by security researchers in real time or off-line by surrogate "game" engines that play out pre-defined strategies. This approach would allow us to quantify the typical payoff matrices that are used in game theory to try to predict the ultimate outcome of such scenarios.

Acknowledgments

This research was partially funded by Canada's Natural Sciences and Engineering Research Council (NSERC) strategic research network on Internetworked System Security Network (ISSNet). We are also very grateful for the valuable input and feedback we received from Patrick McDaniel on previous versions of this manuscript.

6. REFERENCES

[1] P. Barford and M. Blodgett. Toward botnet mesocosms. In *Proc. 1^{st} Work. on Hot Topics in Understanding Botnets (HotBots)*, Apr. 2007.

[2] T. Benzel, R. Braden, D. Kim, C. Neuman, A. Joseph, K. Sklower, R. Ostrenga, and S. Schwab. Experience with DETER: A testbed for security research. In *Proc. IEEE Conf. on Testbeds and Research Infrastructures for the Dev. of Networks and Communities (TridentCom)*, Mar. 2006.

[3] P.-M. Bureau and J. Fernandez. Optimising networks against malware. In *Proc. Int. Swarm Intelligence and Other Forms of Malware Work. (MALWARE)*, Apr. 2007.

[4] J. Calvet, C. Davis, and P.-M. Bureau. Malware authors don't learn, and that's good! In *Proc. Int. Conf. on Malicious and Unwanted Software (MALWARE)*, Oct. 2009.

[5] J. Calvet, C. Davis, J. Fernandez, W. Guizani, M. Kaczmarek, J.-Y. Marion, and P.-L. St-Onge. Isolated virtualised clusters: testbeds for high-security experimentation and training. In *Proc. 3^{rd} USENIX Work. on Cyber Sec. Experimentation and Test (CSET)*, Aug. 2010.

[6] E. Cooke, F. Jahanian, and D. McPherson. The zombie roundup: Understanding, detecting, and disrupting botnets. In *Proc. Work. on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, July 2005.

[7] D. Dagon, G.Gu, C.Zou, J. Grizzard, S. Dwivedi, W. Lee, and R. Lipton. A taxonomy of botnets. In *Proc. of CAIDA DNS-OARC Work.*, July 2005.

[8] D. Dagon, C. Zou, and W. Lee. Modeling botnet propagation using time zones. In *Proc. 13^{th} Network and Distributed System Security Symp. (NDSS)*, Feb. 2006.

[9] C. Davis, J. Fernandez, and S. Neville. Optimising sybil attacks against P2P-based botnets. Proc. 4^{th} Int. Conf. on Malicious and Unwanted Software (MALWARE), Oct. 2009.

[10] C. Davis, J. Fernandez, S. Neville, and J. McHugh. Sybil attacks as a mitigation strategy against the storm botnet. In *Proc. 3^{rd} Int. Conf. on Malicious and Unwanted Software (MALWARE)*, Oct. 2008.

[11] C. Davis, S. Neville, J. Fernandez, J.-M. Robert, and J. McHugh. Structured peer-to-peer overlay networks: Ideal botnets command and control infrastructures? In *Proc. 13^{th} European Symp. on Research in Computer Security (ESORICS)*, Oct. 2008.

[12] S. Gaudin. Storm botnet puts up defenses and starts attacking back. `http://informationweek.com`, Aug. 2007.

[13] A. Jackson, D. Lapsley, C. Jones, M. Zatko, C. Golubitsky, and W. Strayer. Slingbot: A system for live investigation of next generation botnets. In *Proc. of IEEE Conf. for Homeland Security, Cybersecurity Applications and Technology (CATCH '09)*, Mar. 2009.

[14] J. John, A. Moshchuk, S. Gribble, and A. Krishnamurthy. Studying spamming botnets using botlab. In *Proc. 6^{th} USENIX Symp. on Networked Systems Designs and Implementation (NSDI)*, Apr. 2009.

[15] C. Kanich, C. Kreibich, K. Levchenko, B. Enright, G. Voelker, V. Paxson, and S. Savage. Spamalytics: an empirical analysis of spam marketing conversion. In *Proc. 15^{th} ACM Conf. Comp. & Comm. Security (CCS)*, Oct. 2008.

[16] C. Kanich, K. Levchenko, B. Enright, G. Voelker, and S. Savage. The Heisenbot uncertainty problem: Challenges in separating bots from chaff. In *Proc. 1^{st} USENIX Work. Large-Scale Exploits & Emergent Threats (LEET)*, Apr. 2008.

[17] L. Peterson and T. Roscoe. The design principles of PlanetLab. *ACM SIGOPS Operating Systems Review*, 40:11–16, Jan. 2006.

[18] M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A multifaceted approach to understanding the botnet phenomenon. In *Proc. 6^{th} ACM SIGCOMM Conf. on Internet measurement (IMC)*, Oct. 2006.

[19] E. Ruitenbeek and W. Sanders. Modeling peer-to-peer botnets. In *Proc. 5^{th} Int. Conf. on Quantitative Evaluation of Systems (QuEST)*, pages 307–316, Sept. 2008.

[20] G. Sinclair, C. Nunnery, and B. Kang. The Waledac protocol: The how and why. In *Proc. 4^{th} Int. Conf. on Malicious and Unwanted Software (MALWARE)*, Oct. 2009.

[21] J. Stewart. Storm worm DDoS attack. `http://www.secureworks.com/research/threats/storm-worm`, Feb. 2007.

[22] B. Stock, J. Goebel, M. Engelberth, F. Freiling, and T. Holz. Walowdac analysis of a peer-to-peer botnet. In *Proc. Europ. Conf. Computer Network Defense (EC2ND)*, Nov. 2009.

[23] P. Wang, S. Sparks, and C. C. Zou. An advanced hybrid peer-to-peer botnet. In *Proc. 1^{st} Work. on Hot Topics in Understanding Botnets (HotBots)*, Apr. 2007.

[24] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. of 5^{th} Symp. on Operating systems design and implementation (OSDI)*, pages 255–270, 2002.

[25] Y. Zhao, Y. Xie, F. Yu, Q. Ke, Y. Yu, Y. Chen, and E. Gillum. Botgraph: Large scale spamming botnet detection. In *Proc. 6^{th} USENIX Symp. on Networked Systems Designs and Implementation (NSDI)*, 2009.