Toward Botnet Mesocosms

Paul Barford and Mike Blodgett
University of Wisconsin-Madison
pb@cs.wisc.edu, mblodget@cs.wisc.edu

***Abstract*— An in-depth understanding of botnet behavior is a precursor to building effective defenses against this serious and growing threat. In this paper we describe our initial steps toward building a flexible and scalable laboratory testbed for experiments with bots and botnets. Our Botnet Evaluation Environment (BEE) is designed to enable individual bots or networks of up to thousands of bots to be tested in a secure, self-contained framework. BEE is being developed as a toolkit for Emulab-enabled network testbeds; a design choice made to obviate the need for building user/experiment management functions and to enable access to collections of computing hosts. The focus of our implementation efforts has been on building a library of OS/Bot images that can be run on individual systems or on virtual machines. The library currently includes images generated from source code of four well known bots (Agobot, GTbot, Spybot, SDbot) and from binary code for several unknown bots, and a number of Windows OS variants. BEE also includes a set of services that are required for botnets including DHCP, DynDNS, and IRC, as well as other tools that are useful for botnet measurement and evaluation such as VM monitors and honeypots. To demonstrate the utility of BEE, we describe a simple set of tests that characterizes command and control traffic from three different botnet configurations.**

I. Introduction

> ***mes·o·cosm****: n. a medium size, representative system that matches as directly as possible a larger system in constitution, configuration or development.*

Over the past several years, *botnets* have become one of the most serious threats ever faced by the Internet. Many recent reports describing botnet structure and behavior have appeared in the technical and popular press (*e.g.,* [1], [2], [3]). While it is difficult to estimate the total number of systems that participate in botnets at any point in time, most estimates are on the order of millions [4], [5], [6], and Turing Award winner Vint Cerf recently predicted that as much as 25% of all Internet hosts could become botnet drones over the next several years [7]. Furthermore, the magnitude of the botnet threat is now widely recognized to be compounded by the emergence of an active botnet economy that is likely to be fueled by organized crime [8], [9], [10]. Developing effective counter-measures against this threat will require significant focus and innovation from the network security community.

We argue that a comprehensive testbed environment for botnet study is a critical building block in the quest to address the botnet threat. The requirements for such an environment include:

1) the ability to experiment with a variety of bot types (both known and unknown) running on a variety of standard operating systems,
2) the ability to conduct experiments in a secure fashion (*i.e.,* one that poses no threat to the greater Internet),
3) the ability to create flexible and realistic botnet topologies and configurations,
4) the ability to conduct experiments at scale and under realistic conditions.

We posit that a testbed that satisfies these requirements would enable a range of experimental study on new methods and tools for characterizing, comparing, identifying, tracking, dismantling, and preventing botnets. The key benefits of such an environment are the ability to establish *ground truth* behavior through comprehensive instrumentation, experimental *repeatability* and the ability to conduct experiments over a wide range of configurations.

In this paper we describe the design and implementation of the Botnet Evaluation Environment (BEE), with the long term goal of satisfying the requirements listed above. In brief, BEE is a set of operating system/bot images and support tool configurations that can be instantiated in secure, Emulab-enabled environments including DETER [11] and the Wisconsin Advanced Internet Lab (WAIL) [12]. Emulab is a widely used network emulation testbed technology that enables customized operating system and application images to be deployed on dedicated PCs, and for those systems to be organized into virtually any kind of topological configuration [13]. These capabilities satisfy aspects of the third and fourth BEE requirements listed above, making an Emulab-enabled testbed a natural starting point for our work.

The relatively recent emergence of malicious botnets, their dynamic nature and mechanisms for obfuscating their behavior are central challenges in creating an effective, realistic laboratory environment for botnet evaluation. While the literature is growing, there are remain many open questions on botnet structure and behavior *e.g.,* typical botnet topologies, botmaster behavior and even size and lifetimes of botnets. Other complicating factors include handling anti-virtualization capabilities, providing sufficient support services for a wide range of bot variants (*e.g.,* those that use non-IRC command and control systems), and the tussle between containment and experiments with bots that "phone home" for binary uploads. As the community develops a deeper understanding of these issues through empirical study, we argue that the utility of BEE for *botnet* study (*e.g.,* the network effects and impact of multiple bots communicating and reacting to botmaster commands) will grow. In the meantime, we argue that BEE is

a valuable platform for the study of individual *bots*.

BEE's design includes three primary components: OS/bot images, support services, and security mechanisms. The first component, as its name suggests, is a library of different versions of bot code installed on different versions of MS Windows operating systems. Our goals for this component are 1) to automate the process of generating OS/bot images, and 2) to build out a library of images that will enable experiments with wide range of individual bots [14], [15]). Building these images has been the primary focus of our initial efforts. We want to be able to experiment with bots built from bot code in either source or binary form. The former is very useful with respect to building all of the necessary support infrastructure and for establishing ground truth behavior for a particular bot variant. However, it can be difficult to find bot code in source form. In contrast, a large number of bot code binaries (or binaries suspected to be bot code) can easily be captured through the use of honeynets [16]. The challenges in building and experimenting with images built from bot binaries are security containment and the fact that a growing number of bots include anti-virtualization and anti-debugging capability.

We have developed an initial set of OS/bot images from both source code and binaries, and continue the library development effort. To improve usability and enable experiments with large numbers of bots, we have developed images that can be run on virtual machines. In our experience, this enables approximately 5-to-1 bot-to-host multiplexing on modestly configured PCs. With this capability, we have experimented with 500 bot networks – what we refer to as *botnet mesocosms*. Finally, each OS/bot image in our library includes documentation on when it was created, its name and family, and its important features such as a list of its control commands, propagation modes, and attack and information gathering capabilities.

The second component of BEE is the set of support services that are required for botnets to function. The current set of services include DHCP, DynDNS and IRC. We had to develop specific configurations for each of these services in BEE so that they will respond appropriately to a wide variety of queries. For example, BEE ignores the destination address for queries directed to the DNS service and directs them to its own DynDNS server. This server, in turn, replies to all queries with the address of the BEE IRC server. While the current set of services are, in some sense, IRC-centric, we plan to enhance this component with additional capabilities as needed to support *e.g.,* other command and control mechanisms.

The final component of BEE is it's set of security mechanisms. These are critically important since real bot code is used to build OS/bot images used in experiments. Two threat models were considered in designing the security mechanisms. The first is bot code that is not BEE-aware, but that may attempt to self-propagate or conduct other malicious activity beyond BEE. The second is users who may inadvertently disable one of the security mechanisms in the course of an experiment. To address these threats, we implemented three different security mechanisms all aimed at keeping BEE traffic contained within an experimental configuration. The mechanisms include: (i) a firewall at the BEE border that blocks all outgoing connections and UDP packets, (ii) keeping the BEE management network logically separate from the experimental network, and (iii) using only unroutable (10-net) addresses within BEE. To date, these mechanisms have only been tested in the WAIL environment, but it seems clear that DETER (a testbed designed for secure, self-contained experiments) would easily accommodate BEE, and that it could be ported to a standard Emulab environment. We discuss some of implications of these mechanisms in Section III-D.

An important aspect of BEE is how it is packaged and used. Our goal is to make experimental development by users as simple as possible. At the highest level, BEE will be available to the research community subject to approval by testbed operators, who will be conservative due to the possible dangers of experiments in BEE. Upon approval, users will be given access to the BEE OS/image library and configuration scripts that instantiate the required services. Users can then experiment with individual bots or build botnets with OS/bot images in the same way that other network topologies are built in Emulab-enabled environments. In this case, however, the images will only be able to be deployed on systems that reside behind the testbed firewall. After a topology has been built, and other mechanisms such as background traffic generators and specific instrumentation have been deployed and the service scripts have been run, the botnet can be exercised like a botnet in the live Internet *e.g.,* by sending commands through the BEE IRC server.

To demonstrate the efficacy of BEE, we created three simple botnet mesocosms in WAIL. We used these experimental configurations to generate, measure, characterize and compare botnet command and control traffic. While the results show a predictable traffic pattern through the IRC server, they also highlight randomness and differences in the traffic process which we would expect to see in the Internet. While we make no arguments for the intrinsic value of these results, we believe that the demonstration provides useful insights on the capabilities of the toolkit.

The remainder of this paper is organized as follows. We review studies related to botnet measurement and analysis in Section II. The details of BEE's implementation, the challenges that had to be overcome in its development and the challenges in its use are given in Section III. In Section IV, we report results of our case study that demonstrates how BEE can be used. We summarize our work and describe future directions for BEE in Section V.

II. Related Work

Network security research can be conducted in a variety of ways. One of the most important modalities is empirical study based on gathering data from live and/or dedicated measurement systems deployed in the Internet. There are several mechanisms that have been used effectively to gather

data specifically on botnet behavior. The first is by monitoring DNS for lookups commonly used by bots to resolve the address of the server (typically an IRC server) used for command and control. Active methods via DNS hijacking [17] and passive methods via DNS-based blackhole lists [18] have been described in prior work. Another method for tracking botnet activity is through the use of monitors deployed on unused address space (honeynets) [16], [19], [20], [21], [22]. Honeynets with active response capability enable details of attacks to be tracked and malicious binaries to be gathered [14], [23]. Two recent studies by Rajab *et al.* [24] and Freiling *et al.* [25] describe multifaceted approaches to evaluating botnets based on using honeynets to gather malicious binaries, and then to monitor systems on which these binaries execute for command and control traffic associated with botnets. Similar to that work, we anticipate that honeynets will continue to be an important source of malware that can be evaluated in BEE, but unlike that work, BEE is focused on understanding how small to medium sized botnets behave, not just individual instances. Finally, the potential of correlating data from multiple sources as a means for detection and real time tracking of botnets has been discussed in several papers including [26], [27], [28]. We believe that BEE will eventually be useful for testing this kind of distributed monitoring system with real botnets.

A second modality for network security research is through detailed examination of instances of both source code (instances of malware source code can be found by searching the Web and Usenet news groups) and executable code (executables can be gathered, *e.g.,* via honeynets). Standard tools are used to reverse engineer executables including disassemblers, debuggers and system monitors such as [29], [30], [31]. This *microcosm* approach is commonly used by network and host security vendors for signature development. However, malware authors are well aware of this approach, and are known to be developing specific deception techniques to complicate the analysis process [32]. Similarly, there are many tools available for static analysis of source code such as [33], [34]. These tools are most often focused on the problems of identifying run time errors and security vulnerabilities. However, the information they can provide including symbol tables, call graphs and parse trees could be valuable in future bot code analysis. We anticipate that a diverse repository of botnet configurations in BEE will be useful to researchers interested in developing and testing, *e.g.,* host-based mechanisms for botnet detection.

A third modality for network security research, and the one that we advocate in this paper for the study of botnets is through the use of emulation testbeds such as Emulab/DETER/WAIL [13], [11], [12]. These testbeds offer the possibility for study in controlled environments that strive to be realistic. While hundreds of papers have been published based on results from emulation infrastructures, botnet analysis presents both challenges and opportunities as described throughout this paper. One particular challenge is conducting large scale experiments, *e.g.,* approximating the size of the entire Internet. Weaver *et al.* offer an interesting approach for addressing this by describing scale-down techniques in [35]. Our objective in BEE is to enable realistic experiments with small to medium sized botnets as defined in [24]. Experiments with larger bots loom as an interesting possibility but are beyond the scope of this study. We are aware of no prior work that attempts to build a scalable, extensible and realistic testbed for botnet evaluation and analysis.

Finally, there are several efforts underway to identify and characterize malware in general and bot code in particular that are hybrids of the aforementioned analysis methods. A commercial example of this is the Norman Sandbox, which evaluates malicious binaries in a secure, emulation environment and produces reports on their behavioral characteristics including classification into malware types such as bots [36]. A similar commercial product is the Sunbelt CWSandbox [37]

III. The Botnet Evaluation Environment

In this section, we describe the implementation details of BEE including the challenges that we faced during the development process. We also provide a description of how BEE is used in an Emulab-enabled environment, and discuss its current and future limitations.

A. BEE Implementation

The objective of our work is to build a bot testbed that satisfied the requirements listed in Section I. The key components of the requirements are realism, flexibility, security, usability and scalability

The first step toward satisfying these requirements was to design and develop BEE as a toolkit for Emulab-enabled laboratories. Building on top of Emulab allowed us to take advantage of established functionality that includes user, experiment and disk image management, and straight-forward network topology generation and configuration. This choice also allowed us take advantage of installations of large numbers of dedicated general purpose computer hosts, which are required to enable BEE to be used by the research community at large and to conduct experiments with botnet mesocosms. Our local Emulab-enabled facility, WAIL [12], has over 100 PCs, and is the infrastructure on which BEE development has taken place.

With Emulab/WAIL as a starting point, BEE's design includes OS/bot disk images, a set of services required for bot operation and a set of security mechanisms that aim to ensure that no malicious packets from bots can escape into the live Internet. Each of these components is described in detail below. We also added VMware-based virtualization capability, which is the most direct method for instantiating MS Windows-based OS/bot images in Emulab-enabled testbeds. Virtualization also enables botnet size to be increased by running multiple images per PC host.

B. OS/bot Image Library

At the core of BEE is a library of OS/bot disk images that are available to authorized users through a web interface. The images are built from real bot code on top of MS Windows operating system variants, and can be deployed in an Emulab test network, enabling experiments with real bots or botnets in a matter of minutes. Our goal in developing this library is to provide users with a large number of bot variants developed from either bot source code or binary code. Learning how to build OS/bot images and then building tools to automate the process has been the primary focus of our initial efforts.

Source code for some common bots is available from the web, news groups and underground sources (although we anticipate that it will be increasingly difficult to find source instances over time). We gathered several of the most well know bot variants for the initial BEE OS/bot image development. Working with source code enabled us to bootstrap our testbed development process by providing important insights on the required network services, precluding the need for IRC passwords or channel keys, and exposing the entire command set for each bot. Perhaps most importantly, source code exposes all of the anti-debugging and anti-virtualization capabilities of the bot. Disabling these capabilities is essential to operation in a virtualized network configuration. In our current inventory of bots, only the Agobot code included anti-debugging and virtualization checks. During initialization our version of Agobot checks for the VMware "BackDoor I/O", and exits if it is found. Other bot code variants have been known to check values in various hardware registers or memory locations, and for the existence of processes or configuration elements known to exist within a guest operating system running on a VM.

The process for creating guest OS images begins by identifying a target operating system. Our initial development efforts focused on MS Windows variants (specifically Windows 2000) although our methodology can easily be extended for Unix variants. After the initial install of the guest OS, a number of changes must be made the the VMware virtual machine configuration file to allow multiple identical image instances. These changes are not OS specific, and are related to certain locking and logging features of VMware. Inside the guest OS we must set up the environment so that when the image is booted on an experimental node, it will configure node specific parameters, *e.g.*, provide a unique NetBIOS name for each image. Additionally, if the node is required to auto-infect itself with a bot binary (*i.e.*, execute a bot binary that is resident in the root directory), we must pass that information into the guest. Currently we do this through the variable passing features of VMware, but these tools also provide a detectable feature for any anti-debugging code. We are currently developing a new method for adding binaries that bypasses the standard VMware tools thereby making the overall process of running binaries in VMware environments more efficient. Finally, the build time for a new OS image from scratch can take up to one day due to the myriad of configuration tasks related to creating a specific MS Windows configuration (including service packs, patches, etc.) on VMware. We are also developing tools for streamlining this process.

Adding bots to OS images when the bot source code available is usually a straight forward task. Finding, configuring and compiling the bot are the most time consuming aspects of the process. Typically, anti-debugging or anti-virtualization checks are commented in the code and are easy to disabled. Our current implementation requires an administrator to configure the bot and place the compiled binary into the VMware virtual disk – this constitutes what we refer to as an "OS/bot image". It will eventually be possible for users to create their own OS/bot images from either source code or binaries.

Adding a binary to an OS image when the binary is either known to be or suspected to be a bot can be more complicated. The process begins by placing the binary into the VMware virtual disk. Next, the resulting OS/bot image must be tested to see if it includes anti-virtualization checks that preclude its use on VMware installations. Depending on the sophistication of the anti-virtualization mechanisms, the image may only be able to be used on a stand-alone PC which would limit the scale of experiments. While stand-alone MS Windows images are not currently supported in BEE, we are investigating how to utilize Emulab's support for this capability. Otherwise, changes to the VMware installation may very well provide a solution. We are currently working on adding some of the known counter VM detection mechanisms into our guest images [38].

We are currently developing an automated method for users to include their own bots in BEE. We believe that this is very important since it will enable expansion of the bot library (*e.g.*, by gathering binaries from honeypots) and to increase the utility of BEE (*e.g.*, by developing automated bot identification capability similar to [36], [37]). Users interested in testing their own bot code currently have to submit it to the testbed administrators who can build it into a standard image.

```
[ Network services ]
* Connects to "irc.abraracourcix.com" on port 3705 (TCP).
* Sends data stream (11 bytes) to "irc.abraracourcix.com",
  port 3705.
* Connects to IRC Server.
* IRC: Uses nickname USA|803400.
* IRC: Uses username ixllgi.
* IRC: Joins channel ##H4ck.Rb0t## with password asd.
* IRC: Sets the usermode for user USA|803400 to +i-x.
```

Fig. 1. Example output from Norman Sandbox [36] analysis of bot binary taken from Offensive Computing [39].

The final step in the process of building OS/bot images from either bot source or binaries is to generate documentation for each image. A standard template is filled out, to the extent possible, by the person creating the image. The template includes information on image creation, the OS name/version,

bot name/version,[1] whether or not the image can be used in a virtualized environment, whether or not the bot includes anti-debugging and notes on configuration and use (*e.g.*, IRC channel name and bot user name). The documentation also includes notes of the bot itself such as commands that are recognized by the bot, exploits/capabilities of the bot (*e.g.*, DoS, Spam relay, etc.), and propagation methods. Our focus is not on deep details, but to provide information sufficient to conduct a wide range of experiments with the botnets. For binaries, services such as the Norman Sandbox can be used to expose details of bots that are necessary for their use in BEE as shown in Figure 1. In this example, the IRC server name, port number and the channel name are provided, all of which are required for exercising this unknown bot.

The OS/Bot images (plus documentation) currently available in the BEE library include:

- Windows 2000 + Agobot/Phatbot (from source)
- Windows 2000 + SDBot (from source)
- Windows 2000 + GTBot (from source)
- Windows 2000 + SpyBot (from source)
- Windows XP + Agobot/Phatbot (from source)
- Windows 2000 + SpyBot (unknown binary version)

This set was developed and tested to demonstrate the ability to mix and match different OS and bot variants from both source code and binaries. It is our hope that as BEE is used by the community, that others will contribute to the image building and documentation effort.

C. Network Services

Bots in the live Internet expect a standard set of services and will not operate without them. Thus, the same set of services must be available in any test environment built from BEE OS/bot images. The set of services listed below have been configured, tested and packaged for easy instantiation and use in BEE. We anticipate expanding this set over time as new bots that require additional services are added to the library.

1) IRC Server: At present, Internet Relay Chat [41] is the primary means for bot command and control. We use IRCd-hybrid7 to provide IRC services in BEE [42]. Our default configuration disables many of the anti-flooding controls. Users can, of course, modify the configuration of the IRC daemon or completely replace it if required.

2) DNS Server: BIND can be run on any of the experimental nodes. The default configuration for BEE sets up a root zone that can be updated via DynDNS which is frequently used by bots [17]. Users specify DNS entries in the Emulab/NS configuration file, which are added to each server on startup, or entries can be manually added via the NS-update tool. A default wild card can also be used, returning a single address for all names. For security purposes, BIND is configured to only listen on experimental interfaces and does not provide recursive services. This design choice could limit the range of experiments with some bots, so we are looking into alternative configurations.

3) VMware Server: VMware-server 1.0.1 is used to provide virtualization on the testbed. We chose VMware based on its well known support for Windows images. We created a custom configuration for the VMware server that enables a user to run multiple instances of the same guest OS image on a target host. This configuration is fetched and installed on target hosts when an experiment is instantiated. The process of loading OS/bot images onto the virtualized systems is likewise automated via configuration scripts.

The current testbed nodes do not have hardware virtualization support (*e.g.*, Intel's IVT or AMD's AMD-V). We are considering possibility of adding these nodes to the WAIL testbed, which may affect the future choice of virtual machine software.

4) DHCP Server: While not used by the bots themselves, to provide addresses to the virtual machines inside an experimental network we included the ISC DHCP server and DHCP forwarders. Since the topology of the network can be varied by the user, the configuration of the DHCP daemon and DCHP forwarders must be determined at run time. At boot time, the DHCP server node(s) will extract their configuration from the Emulab/NS configuration file. For anything more than a simple LAN topology, DHCP forwarders are likely to be necessary, which at present need to be chosen manually and enabled via the Emulab/NS file. The DHCP servers are configured to only respond to MAC address prefixes corresponding to VMware virtual adapters, which otherwise cause problems with the testbed bootstrapping mechanisms.

5) Other Services: Several additional capabilities are included in BEE to enhance usability. First, packet measurement (WAIL includes systems with high performance hardware-based packet capture capability which can be used BEE experiments), link tracing and propagation delay emulation can be specified in the Emulab/NS configuration file, and examples are included in BEE/NS templates. We also included a Nepenthes honeypot image that is simple to instantiate from the Emulab/NS. A Harpoon image is also included, which enables background traffic generators to be easily included in experiments [43].

A final capability that is important in the analysis of bots in isolation is instrumentation and monitoring of the OS/bot images themselves. Dynamic profiling tools such as IDA Pro [29] can be used to provide disassembly information on the bot itself, while tools available from Sysinternals provide an array of information on MS Windows OS behavior (*e.g.*, utilities for file system, network, process and other resource usage) [31]. While not currently available in BEE, we plan to include these and other instrumentation tools in future versions of BEE.

[1]The details of malware naming conventions vary widely and we are attempting to be consistent with projects such as MITRE's Common Malware Enumeration [40].

D. BEE Security

Since real instances of malicious code are available in BEE, great care must be taken to prevent an outbreak beyond a specified testbed boundary. The security mechanisms in BEE are designed to prevent non-BEE aware bots from sending packets beyond the testbed perimeter and to prevent users from inadvertently allowing malicious traffic to escape. We believe that the mechanisms described below plus limiting BEE access to authorized users are sufficient, although we will periodically reevaluate both our policies and mechanisms.

Security in BEE is implemented in three ways beyond user authorization. First, management/control traffic from users setting up and running experiments and test traffic generated by bots and services during experiments flow over logically separate networks. We enforce this policy via filtering rules on the hosts that run OS/bot images; the primary rule being do not forward any traffic out the control interface. Similarly, we configure all network services in the experiment to generate traffic that is limited to the testbed, *e.g.,* by disabling recursive DNS. To guard against testbed hosts being compromised during experiments, the topology of the testbed control network is restricted such that the only the minimum required communication between the testbed control systems and the nodes is allowed. Emulab has built in support for creating "Secure Environments" via filtering nodes in the control network, and also isolation of the testbed infrastructure can be achieved by using the "Emulab in Emulab" features. The second security mechanism is a firewall deployed on the WAIL border and configured to block all outgoing connections and UDP traffic. This policy prevents some types of bots from being used in BEE as described below. The firewall mechanism is complemented by a separate monitoring system which enables us to know if our mechanisms have failed. Finally, an experiment-wide network address translation system can be used to limit all bot packets to RFC1918 addresses [44] that are not routable in the live Internet. Our approach is to connect all bot hosts to NAT systems that will intercept all packets from bots and translate their addresses before forwarding them to other hosts in the testbed. We are in the process of extending the NetPath delay emulator for this purpose [45]. While NAT is the default configuration, it can be changed by users to enable, *e.g.,* experiments focused on examining scanning capabilities of bots.

E. Using BEE

BEE is realized as a library of OS/bot images, a set of configuration scripts and a set of Emulab/NS templates that are available to authorized users. Upon testbed approval, users begin by creating an Emulab network topology from scratch or using a BEE template. A network can be as simple as a single node running one OS/bot image or a large, multi-node topology with complex interconnections, diverse propagation delays and realistic background traffic. Once the network has been built, users then edit the NS configuration file to specify the options they want to enable in their network including the specific OS/bot image, the specific services, etc. This network is then instantiated on the testbed (assuming the requested nodes are available), users issue a “go” command via the testbed server that starts all of the virtual machines, and then users can, for example, issue commands to the botnet via the IRC server.

F. Challenges and Limitations in Using BEE

As noted in Section I, there are several challenges that must be addressed before the vision of conducting realistic tests with botnet mesocosms can be fully realized. The nascency of malicious botnet research limits, for example, our ability to build representative botnet topologies. It also precludes study of bots that require services or configurations that we are unaware of and thus have not included in BEE.

Our security policy preventing all outgoing connections limits the diversity of bots in the BEE library. Some bot variants install a simple module on a compromised host that then “phones home” to download additional code as needed or as directed by the botmaster. In general, it is difficult (or impossible in the case of encrypted packets) to determine the intent of an outgoing connection from real malware. For obvious reasons, any packet that would do damage beyond the testbed must be blocked. Therefore, we block *all* outgoing connections. A similar challenge is faced by honeypot operators. If all of the code segments of a particular bot variant were captured by other means (*e.g.,* a honeypot), then a “phone home” server could be added to BEE.

As mentioned above, anti-debugging and anti-virtualization capability in bot code binaries presents challenges in building out the OS/bot image library. While most of the bots that we have experimented with to date do not have these capabilities, it is prudent to expect that most future bots will. In the case of anti-virtualization, we plan to continually augment out OS/bot image building methods to include the latest counter detection mechanisms (*e.g.,* [38]), although we expect there will always be some images that can only be run on a stand alone host. In the case of anti-debugging, as dynamic analysis tools that can account for these capabilities become available, we will include them in BEE.

G. Future Work on Scalability

While virtualization enables individual systems to run multiple bot images simultaneously on a single PC, the actual multiplexing factor is limited by the system resources required by each bot. The point is that users must be careful not to skew experimental results by overloading systems with too many bot images. Our own experience in this regard indicates that multiplexing factors of less than 10 to 1 are reasonable in most cases. Trial-and-error is the only way at present to make this determination. Since all Emulab-enabled testbeds are well under 1K PCs, a federation between testbeds will be necessary to conduct experiments with networks over 10K bots (which are not uncommon [3])

Beyond federating Emulab-enabled environments, another avenue we are pursuing is to enable the considerable resources available from grid computing infrastructures to be used in BEE experiments. For example, the Condor Project controls a large number of systems on our campus which are openly available for research [46]. The integration of these nodes as temporary resources could potentially provide us the ability to scale the number of bot instances into the thousands. However, the distributed nature of these resources, and the required security measures needed will make integration a challenge. We are currently investigating the feasibility of this idea.

IV. BEE Demonstration

To provide perspective on BEE capabilities, we conducted a set of simple tests that characterize command and control traffic for several different botnet mesocosms. Our intention is to highlight the process of building and exercising botnets in BEE. The results from the tests are not intended to be representative of real instances of botnet use (indeed an off-line analysis from empirical traces would be much more informative). However, one can envision using a similar environment to conduct repeatable tests in which ground truth can be established on *e.g.*, new tools for detecting bot command and control traffic, or bot scanning/propagation over a range of bot types, botnet sizes and configurations, or the scalability of honeynet systems designed to capture malware such as bots.

A. Experimental Setup and Protocol

The network topology used in all tests was built in WAIL, and is shown in Figure 2. The design goal was a network with a range of propagation delays and with sufficient link capacity so that congestion would only potentially become an issue at the single bottleneck (node A) when background traffic was introduced.

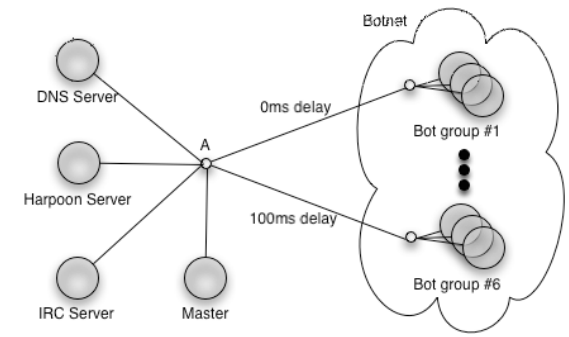

Fig. 2. Network topology used for the BEE case study.

The botnet groups 1 through 6 shown in Figure 2 consisted of 5 PCs per group. Each of these PCs ran VMware Server 1.0.1. We loaded 5 virtual machines per PC for each test. We arrived at this multiplexing factor by monitoring resource utilization (CPU, memory, network) during tests. Five images per host kept CPU utilization below 100% thereby ameliorating the impact of virtualization on results.

Using two OS/bot images from the BEE library and BEE scripts to deploy the required services (including security measures described in Section III), we conducted three different botnet tests, the details of which are given in Table I. The focus of each test was to measure, characterize and compare the basic features of bot command and control traffic. We captured all IRC traffic in each test using tcpdump running on the node represented at point A in the topology.

Each test consisted of issuing a set of $C = c_1 \ldots c_N$ commands to a 150 node botnet. A script running on the master node sent each command c_i to the IRC server five consecutive times *i.e.*, $c_{i,1} \ldots c_{i,5}$ with a 60 second spacing between command and followed by a one minute interval before issuing the next command c_{i+1}. This resulted in tests running for approximately 45 minutes. Loading the virtual machines takes about 15 minutes, and the tests run for about 30 minutes from initial command to end. In the case of test #2, Harpoon background traffic generators warmed up for 1 minute before the Master began issuing commands.

The Agobot variant used in the tests has a weak random nickname generator and with a high probability collisions will occur. Since it has no mechanism for collision detection, if names collide on start up, a bot will continually attempt to connect and eventually timeout to the IRC server. To address this, several minutes after tests #1 and #2 were started, we used a manual process to identify *connected* bots and direct them to choose a new IRC nickname. This, in turn, enabled the *unconnected* bots to join the channel.

B. Results

A simple statistical summary of the command and control traffic characteristics is given in Table II. To focus the characterization, we define a *busy period* in a test as an interval that begins with a Master sending a command $c_{i,j}$ and ending with the first zero packet per second interval after $c_{i,j}$ is sent (thus, there are 25 busy periods for each test). The Busy Period Packet per Second (BPPS) results for tests #1 and #2 reflect the command, ACK, response and confirmation packets sent by the bots. The differences in the results for tests #1 and #2 are primarily due to timing and traffic randomness in the tests and do not reflect significant underlying effects including background traffic. This inherent randomness may be a harbinger of the difficulty of developing statistical methods for identifying this traffic in the live Internet. The difference in BPPS for the Agobot tests versus the Spybot test #3 is primarily caused by the responses to the command sets (*e.g.*, "list p" generates a great deal of traffic), and does not indicate any relative inefficiency of Spybot communications.

An example of the IRC packet per second time series from test #1 is shown in Figure 3. The figure shows the predictable pattern of busy and non-busy periods due to our test protocol. A small amount of activity can also be seen in non-busy periods. This traffic is attributed to the IRC ping/pong packets from the server to the bots, which takes place throughout the tests. The fact that spikes of this ping-pong packets are visible

TABLE I

DETAILS FOR THE THREE DIFFERENT TEST CONFIGURATIONS USED IN THE BEE CASE STUDY.

Test #	Image (os/bot)	Topology	Link BW	Master Commands.	Background Traffic	Runtime
1	W2K/Agobot	150 bots (30 per 6 bot groups) as shown in Figure 1	1Gbps	".bot.secure",".bot.unsecure", ".cvar.set do_avkill false", “.bot.rndnick”, ".irc.join #botnet2"	None	46:53minutes
2	W2K/Agobot	150 bots (30 per 6 bot groups) as shown in Figure 1	1Gbps	".bot.secure",".bot.unsecure" ".cvar.set do_avkill false", “.bot.rndnick”, ".irc.join #botnet2"	Harpoon @ 200Mbps at node A	47:08 minutes
3	W2K/SpyBot	150 bots (30 per 6 bot groups) as shown in Figure 1	1Gbps	"info","list p*","makedir test" "startkeylogger","stopkeylogger"	None	39:31 minutes

TABLE II

SUMMARY STATISTICS FOR COMMAND AND CONTROL TRAFFIC FOR EACH TEST IN THE BEE CASE STUDY. BPPS = PACKETS PER SECOND DURING BUSY PERIODS. BPL = BUSY PERIOD LENGTH.

Test #	Mean BPPS	Std Dev. BPPS	Max BPPS	Mean BPL	Std. Dev. BPL	Max BPL
1	680	828	3452	12.3	6.1	25.0
2	788	693	2457	9.8	5.9	19.0
3	1384	1441	6816	8.5	4.3	23.0

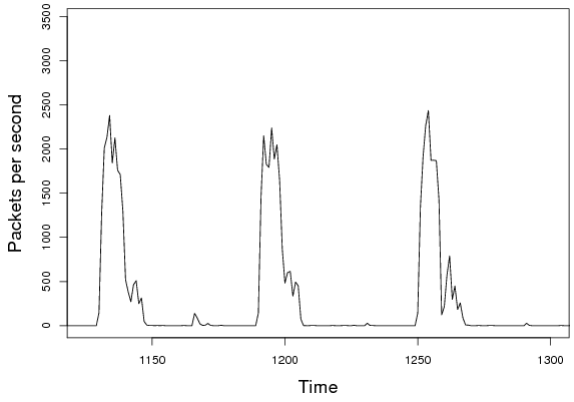

Fig. 3. Snapshot of IRC packet per second time series from test #1. Each spike begins when the bot master sends a commend.

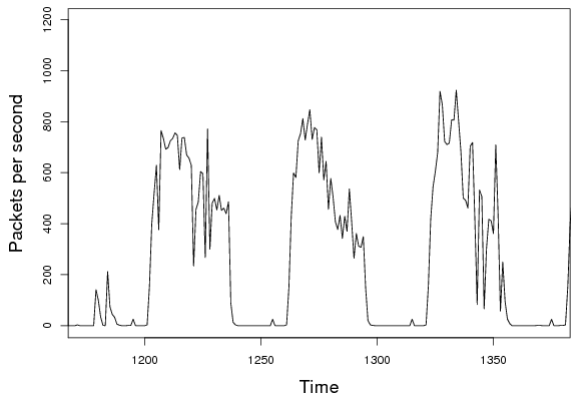

Fig. 4. Snapshot of IRC packet per second time series with 10 OS/bot images per node (150 bots total).

in the figure is due to synchronization of this traffic that arises throughout the course of the tests.

Finally, we include an IRC packet per second time series graph in Figure 4 from a test where 10 bots per PC were used on 15 systems (3 per group with no 0 delay group) for a total of 150 bots. The key observation is that the busy periods have a significantly different profile versus Figure 3 resulting from the 5 bot per node configuration. The difference is caused by CPU contention on each VMware node by running 10 bot images leading to commands taking much longer to complete. This would not be evident in the live Internet where there is a one-to-one correspondence between bots and hosts, and highlights the care that is required in creating BEE configurations.

V. CONCLUSIONS AND FUTURE WORK

The premise for our work is that realistic and secure testbed environments are important for understanding bots and botnets, and for building effective counter-measures against this threat. In this paper we describe the Botnet Evaluation Environment, which has been developed to test and evaluate bots in Emulab-enabled testbeds. The first component of BEE is a library of OS/bot images and associated meta data that have been built from bot code source and binaries. Key challenges in this effort were automating the task of constructing OS/bot images from binaries and understanding how images can be used in virtualized environments. The second component of BEE are the support services that are required for bot and botnet operation. The current set of services includes DNS, IRC and DHCP along with standard support tools such as VM/OS monitors, honeypots and background traffic generators that are commonly used in tests. A set of configuration scripts have been developed that enable all of these services to be automatically deployed in a test network. The third component of BEE are the security mechanisms that were employed to

ensure that no BEE traffic goes beyond a specified perimeter. These systems include NAT functions that aim to ensure all destination addresses in packets are RFC1918 compliant (unroutable in the Internet), a firewalled network perimeter, and a separate management network.

To demonstrate the process of building and testing botnets with BEE we created three different botnet mesocosms in the Wisconsin Advanced Internet Lab. The network topology for each consisted of 46 PCs connected with varying propagation delays to an IRC server. We exercised the botnets by issuing a series of commands via IRC, and measured the resulting command and control traffic. A simple qualitative and quantitative characterization of the traffic was presented, which we hope will inspire future creative use of BEE by the research community toward the goal of reducing the botnet threat.

Our BEE development activities are on-going. We believe that the initial utility of BEE will be primarily in the study of individual bots. To that end, one of our most important tasks is continuing to add images to the OS/bot library. The challenges in this task are in automating the image generation process, creating detailed documentation for images generated from binaries, and developing mechanisms for overcoming anti-debugging and anti-virtualization capabilities. We hope that the research community will eventually engage in the image repository building effort as our tools mature and the utility of BEE becomes clear.

An important objective in future work is to increase the utility of experiments with botnet mesocosms. On-going research by the community on botnet characteristics, capabilities and behaviors will be reflected in enhanced BEE services, configurations and documentation, enabling more detailed and realistic experiments with collections of bots. It is important to note that the DETER [11] testbed has been designed specifically for experiments with malicious code. We have recently begun communicating with DETER developers and look forward to the opportunity to streamline BEE's security mechanisms by taking advantage of built-in capabilities in DETER. Finally, we will continue to investigate methods for expanding the scale of botnet experiments.

ACKNOWLEDGMENTS

The authors thank our shepherds, Fabian Monrose and Vern Paxson, for their assistance on this paper. We also thank Joel Sommers and Vinod Yegneswaran for their input and feedback. This material is based upon work supported through the U.S. Army Research Office under the Cyber-TA Research Grant No. W911NF-06-1-0316, NSF grant No. CCR-0325653 and Homeland Security Advanced Research Projects Agency grant No NBCHC060135. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF, ARO or the Homeland Security Agency.

REFERENCES

[1] J. Markoff, "Attack of the Zombie Computers is a Growing Threat, Experts Say," New York Times, January 2007.

[2] I. Cook, "Security Matters: Beware the Enemy Within," Financial Times, February 2007.

[3] J. Kirk, "Botnets Shrinking in Size, Harder to Trace," InfoWorld.com, January 2006.

[4] A. Gostev, "Malware Evolution: January - March, 2005," http://www.viruslist.com, 2005.

[5] D. Kawamoto, "Bots Slim Down to get Tough," CNET News.com, November 2005.

[6] J. Evers, "Dutch Police Nab Suspected Bot Herders," CNET News.com, October 2005.

[7] T. Weber, "Criminals May Overwhelm the Web," http://news.bbc.co.uk/2/hi/business/6298641.stm, January 2007.

[8] "California Man Charged in Botnet Attacks," Reuters, November 2005.

[9] I. Thomson, "Hackers Fight to Create Worlds Largest Botnet," http://www.vnunet.com, August 2005.

[10] D. Verton, "Organized Crime Invades Cyberspace," http://www.computerworld.com, August 2004.

[11] ISI, "The Deterlab Network Security Testbed based on Emulab," http://www.deterlab.net, 2007.

[12] U. of Wisconsin, "The Wisconsin Advanced Internet Laboratory," http://wail.cs.wisc.edu, 2007.

[13] U. of Utah, "The Emulab Network Emulation Testbed," http://www.emulab.net, 2002.

[14] P. Bacher and T. Holz and M. Kotter and G. Wicherski, "Know Your Enemy: Tracking Botnets," http://www.honeynet.org/papers/bots, March 2005.

[15] P. Barford and V. Yegneswaran, *An Inside Look at Botnets*, ser. Advances in Information Security, Malware Detection. Springer, 2007, vol. 27.

[16] "The Honeynet Project," http://project.honeynet.org, 2003.

[17] D. Dagon, C. Zou, and W. Lee, "Modeling Botnet Propagation Using Time Zones," in *Proceedings of The Network and Distributed Systems Security Symposium (NDSS '06)*, San Diego, CA, February 2006.

[18] A. Ramachandran, N. Feamster, and D. Dagon, "Revealing Botnet Membership Using DNSBL Counter-Intelligence," in *Proceedings of The USENIX Workshop on Steps to Reducing Unwanted Traffic in the Internet (SRUTI '06)*, San Jose, CA, July 2006.

[19] M. Bailey, E. Cooke, F. Jahanian, J. Nazario, and D. Watson, "The Internet Motion Sensor: A Distributed Blackhole Monitoring System," in *Proceedings of The Network and Distributed Security Symposium (NDSS '05)*, San Diego, CA, January 2005.

[20] V. Yegneswaran, P. Barford, and D. Plonka, "On the Design and Use of Internet Sinks for Network Abuse Monitoring," in *Proceedings of Recent Advances on Intrusion Detection (RAID '04)*, Sophia, France, September 2004.

[21] R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson, "Characteristics of internet background radiation," in *Proceedings of ACM Internet Measurement Conference (IMC '04)*, Taormina, Italy, October 2004.

[22] "The Nepenthes Low Interaction Honeypot," http://nepenthes.mwcollect.org, 2007.

[23] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. Snoeren, G. Voelker, and S. Savage, "Scalability, fidelity and containment in the potemkin virtual honeyfarm," in *Proceedings of ACM Symposium on Operating Systems Principles (SOSP '05)*, Brighton, England, October 2005.

[24] M. Rajab, J. Zarfoss, F. Monrose, and A. Terzis, "A multifaceted approach to understanding the botnet phenomenon," in *Proceedings of ACM Internet Measurement Conference*, Rio de Janerio, Brazil, November 2006.

[25] F. Freiling, T. Holz, and G. Wicherski, "Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks," in *Proceedings of The 10th European Symposium on Research in Computer Security (ESORICS '05)*, September 2005.

[26] E. Cooke, F. Jahanian, and D. McPherson, "The zombie roundup: Understanding, detecting and disrupting botnets," in *Proceedings of Usenix Workshop on Stepts to Reducing Unwanted Traffic on the Internet (SRUTI '05)*, Cambridge, MA, July 2005.

[27] B. Krishnamurthy, "MoHonk: Mobile Honeypots to Trace Unwanted Traffic Early," in *Proceedings of The ACM SIGCOMM Network Troubleshooting Workshop* , Portland, OR, September 2004.

[28] M. Allman, E. Blanton, V. Paxson, and S. Shenker, "Fighting Coordinated Attackers with Cross-Organizational Information Sharing," in

Proceedings of The Fifth Workshop on Hot Topics in Networks (HotNets '06), Irvine, CA, November 2004.
[29] "The IDA Pro Disassembler and Debugger," http://www.datarescue.com, 2007.
[30] "The SoftICE Driver Suite," http://www.compuware.com, 2005.
[31] M. Russinovich and B. Cogswell, "Sysinternals," http://www.sysinternals.com, 2007.
[32] "Honeynet Scan of the Month 32," http://www.honeynet.org/scans/scan32/, 2005.
[33] Coverity, "Coverity Prevent," http://www.coverity.com, 2005.
[34] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Mine, D. Monniaux, and X. Rival, "The Astree Static Analyzer," http://www.astree.ens.fr, 2005.
[35] N. Weaver, I. Hamadeh, G. Kesidis, and V. Paxson, "Preliminary results using scale-down to explore worm dynamics," in *Proceedings of ACM Workshop on Rapid Malcode (WORM '04)*, Washington D.C., October 2004.
[36] "The Norman Sandbox," http://sandbox.norman.no, 2007.
[37] "The Sunbelt CWSandbox," http://www.sunbelt-software.com/Sunbelt-CWSandbox.cfm, 2007.
[38] P. Ferrie, "Attacks on Virtual Machines," in *Proceedings of AVAR Conference '06*, Auckland, NZ, December 2006.
[39] "Offensive Computing," http://www.offensivecomputing.net, 2007.
[40] MITRE, "Common Malware Enumeration," http://cme.mitre.org, 2007.
[41] C. Kalt, "Internet Relay Chat: Client Protocol," RFC 2812 (Informational), April 2007.
[42] "IRCD-Hybrid," http://ircd-hybrid.com, 2007.
[43] J. Sommers and P. Barford, "Self-Configuring Network Traffic Generation," in *Proceedings of ACM Internet Measurement Conference (IMC '04)*, Taormina, Italy, October 2004.
[44] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. de Groot, and E. Lear, "Address Allocation for Private Internets," Internet RFC 1918, February 1996.
[45] S. Agarwal, J. Sommers, and P. Barford, "Scalable Network Path Emulation," in *Proceedings of The IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '05)*, San Diego, CA, September 2005.
[46] M. Livny, "Condor: High Throughput Computing," http://www.cs.wisc.edu/condor, 2007.