

# SLINGbot: A System for Live Investigation of Next Generation Botnets\*

Alden W. Jackson, David Lapsley, Christine Jones,  
Mudge Zatzko, Chaos Golubitsky, and W. Timothy Strayer  
BBN Technologies  
10 Moulton Street  
Cambridge, MA 01845, USA

{awjacks, cej, mudge, chaos, tstrayer}@bbn.com, dlapsley@sonusnet.com

## Abstract

*There is an urgent need for a pro-active approach to botnet detection and mitigation that will enable computer network defenders to characterize emerging and future botnet threats and design effective defense techniques before these threats materialize. To this end, we have developed a System for Live Investigation of Next Generation bots (SLINGbot). SLINGbot is an extensible, composable bot framework that enables researchers to construct benign bots for the purposes of generating and characterizing botnet Command and Control (C2) traffic. This enables researchers to simulate current and potential future botnet traffic, characterize it, and design effective defense techniques. In this paper, we describe the SLINGbot system and how it can be used for the pro-active development of botnet defenses.*

## 1. Introduction

A botnet is a collection of computers that have been taken over by attackers and can be controlled in a coordinated fashion to accomplish nefarious purposes. Botnets have evolved into highly powerful distributed computing platforms that are used for purpose such as extortion, on-line fraud, spamming, political gain, and terrorism. They have grown in size, strength, and capability. Symantec estimates that during the last 6 months of 2007, there were an average of 61,940 active bots detected per day, and 5.06 million distinct bots were active at some point during the 6 months [10].

Botnets are a distributed and rapidly evolving threat and are particularly difficult to detect and mitigate. The distributed nature of botnets obviates the current localized,

point detection strategies deployed in enterprise networks. Botnets are evolving at such a rapid rate that detection techniques are obsolete by the time they are deployed.

There is an urgent need to proactively develop defenses against current and future botnet threats. However, the challenge for researchers is how to characterize these botnet threats. We have developed a System for Live Investigation of Next Generation bots (SLINGbot), a composable botnet framework to enable researchers to construct *benign* botnets with varying command and control (C2) structures to enable researchers to generate simulated ground truth for the purposes of characterizing current and potential future botnet C2 structures in order to facilitate the proactive development of effective botnet defenses. SLINGbot is extensible and encourages the use of shared libraries of botnet modules. SLINGbot enables researchers to characterize botnet threats in a controlled and safe manner. This is fundamentally different from any other techniques currently in use.

There are currently only three alternative approaches for characterizing botnets: (1) **Testbeds** that run real botnet software [3]; (2) **Databases** that maintain traffic traces from operational data [1]; and (3) **Reverse Engineering** of captured botnet binaries [8, 4].

All of these approaches have an important role to play, however, they all suffer from the same problems. They are limited to characterising existing botnets and are not capable of investigating potential future botnet threats, and they provide limited insight/control over the experiments conducted. In contrast, SLINGbot enables the characterization of both present and future botnet threats, while also enabling full control over experimental scenarios.

In this paper we describe the SLINGbot framework. This paper is organized as follows. In Section 2 we describe an overview of the the design goals and features of SLINGbot. In Section 3 we describe the SLINGbot System Architecture. In Section 4 we describe the various Bot variants that have been implemented within the SLINGbot framework. We conclude with directions for future work.

---

\*This material is based on work supported by the United States Air Force under Contract No. FA8750-07-C-0212.

## 2. Design Overview

SLINGbot is intended to be used in and support differing research avenues, including testing the effectiveness of botnet detection and mitigation infrastructures against a variety of botnet C2 architectures, developing botnet detection and mitigation techniques for emerging and future botnet threats, and researching the effectiveness, strengths, and weaknesses of new botnet C2 architectures.

SLINGbot enables the exploration of the C2 feature space, and thus permits researchers to characterize the behavior of bots, which in turn, allows for the discovery of “tells” and vulnerabilities that can be used to develop detection and mitigation techniques.

### 2.1. Goals

In developing SLINGbot, our main goals were to design a framework with the following capabilities:

- Instantiating multiple bots within the C2 feature space defined in Section 2.3
- Providing a framework to describe, control, and repeatedly deploy botnets.
- Providing logging, monitoring, and/or verification capabilities.
- Generating realistic simulated ground truth for current and potential future botnet C2 infrastructures
- Accommodating a relatively large-scale botnet (i.e., on the order of hundreds of bots) through a combination of physical and virtual hosts

Additionally, a number of explicit restrictions were placed on the SLINGbot software:

- Only *benign* actions are implemented within the SLINGbot framework (no exploit generation nor attack features have been implemented)
- Auto-propagation is not implemented

These goals determined the overall design and features of the SLINGbot software.

### 2.2. Botnet Formation

SLINGbot mimics the behavior of real botnet C2 infrastructure. Botnets are created when vulnerable computers are identified, exploited, and have special botnet software installed that enables the botmasters to control them. There are five distinct stages of botnet formation. In the first stage,

*Reconnaissance*, the attacker or an automated attack program scans a target network to identify vulnerable target computers. In the second stage, the attacker targets these vulnerable computers with targeted *Exploits* to give the attacker access to the computer. The next stage is the *Reinforcement* stage, where the attacker downloads additional software to the vulnerable computer that will enable privilege escalation, and then downloads and activates the bot software itself. In the *Consolidation* stage, the exploited computers (now bots) connect to the botnet’s command and control infrastructure. This command and control infrastructure could be a single server, or multiple servers, depending on the type of botnet. At this point, the bots await for *Commands* from the botmaster. The final stage is *Mission Execution*.

SLINGbot focuses on the command and control communications within the botnet, represented by the Reinforcement through Command stages. Propagation to and compromise of the host and execution of the mission falls outside the scope of this work, but could be introduced (by other system users) in the future.

### 2.3. C2 Feature Space

We build upon the TrendMicro taxonomy [11] that describes a botnet C2 feature space. The taxonomy classifies the C2 feature space into the following five separate dimensions of botnet functionality:

**Topology** – The botnet hosts must be organized in some way so that the botherder can control them as a group rather than dealing with each host individually. The topology affects the amount of control the botherder has over the botnet, the speed and simplicity of sending new instructions, and the botnet’s resilience to disruption. Examples of botnet topology include: centralized, hierarchical, peer-to-peer, and random/broadcast.

**Rallying Mechanism** – The rallying mechanism is the method by which new bots locate and join the botnet. While rallying can be related to the botnet propagation mechanism, we are specifically interested in the question of how a new bot finds the botnet. Examples of botnet rallying mechanisms include: seeding, database, DNS, and IP.

**Communication Protocol** – This is the actual protocol used for the communication between the bots and the C2 server. Two important features of the protocol used are ease of programming, and difficulty of detection. Using a pre-existing protocol has advantages for both. Examples of botnet communication protocols include: IRC, AIM, HTTP, and P2P.

**Control Mechanism** – The control mechanism is the method by which the botherder sends new commands to registered bots. There are competing goals of minimizing control traffic, which could be used to detect the botnet, and

the ability to issue commands at arbitrary times that are received quickly. Examples of control mechanisms include: callback, polling, and ongoing.

**Command Authentication Mechanism.** – It is important for bots to authenticate commands received from their servers, to prevent outside parties from disabling or taking over botnets. In fact, authentication mechanisms seen so far have not been very creative—typically a simple password provided by the botherder is used—so we do not elaborate here. Examples of botnet command authentication mechanisms include: certificates, password, and none.

Multi-dimensional spider charts, as shown in Figure 1(a), help visualize the C2 feature space that a particular bot occupies. Each C2 feature is shown on a separate axis, all of which radiate from a central point. Different techniques for implementing a C2 feature are mapped to points on the corresponding axis. A bot instantiation can be mapped to an enclosed “ring” on the spider chart, where the ring passes through all the points on each axis that correspond to the techniques implemented by the bot. Figure 1(a) shows three bot variants mapped on the spider chart.

Figure 1(b) visualizes the exploration of the C2 feature space. Each level of the tree diagram represents a single dimension in C2 space, and provides various options. Multiple paths are possible between any pair of options because different variants within a bot instantiation can be created by varying the bot’s parameters. A path from the root of the tree to the bottom describes one particular bot instantiation. For example, the path in bold within Figure 1(b) represents a *centralized, IRC-based, DNS-rallying, ongoing connection* bot.

It is important to note that not all combination of features are realizable, so the explorable area of the botnet C2 space is a subset of the full tree.

## 2.4. Development Language

Our goal was to develop high functionality, robust, reusable code as rapidly as possible. There were several possible development languages available to implement SLINGbot: Java, C++, Python, . . . . Each language had its own advantages and disadvantages. Ultimately, Python was selected for its speed of implementation, platform independence, deployability, and strong library support. It provides excellent support for network protocols at both the low level (via convenient socket libraries) and at the protocol level (via high level APIs for protocols such as XML-RPC, HTTP, SSH, SMTP, FTP, etc.). The TwistedMatrix [6] Protocol Engine framework is also used to provide additional protocol support and higher-level abstractions.

## 3. System Architecture

SLINGbot is composed of the following two software modules: **Botnet Scenario Driver** and the **Composable Bot Framework**. The Botnet Scenario Driver is a control mechanism that drives botnet actions and behavior via an “out-of-band” channel that effectively separates botnet C2 traffic and scenario control traffic. Custom scenarios encapsulated within simple XML files facilitate reproducible and rapid experimentation by describing botnet setup, execution, and cleanup.

The Composable Bot Framework enables the creation of live, but benign, bots that are customizable from a library of components. The components encapsulate bot instructions and C2 feature functionalities, e.g., topology, communication protocol. Custom components may be developed by system users.

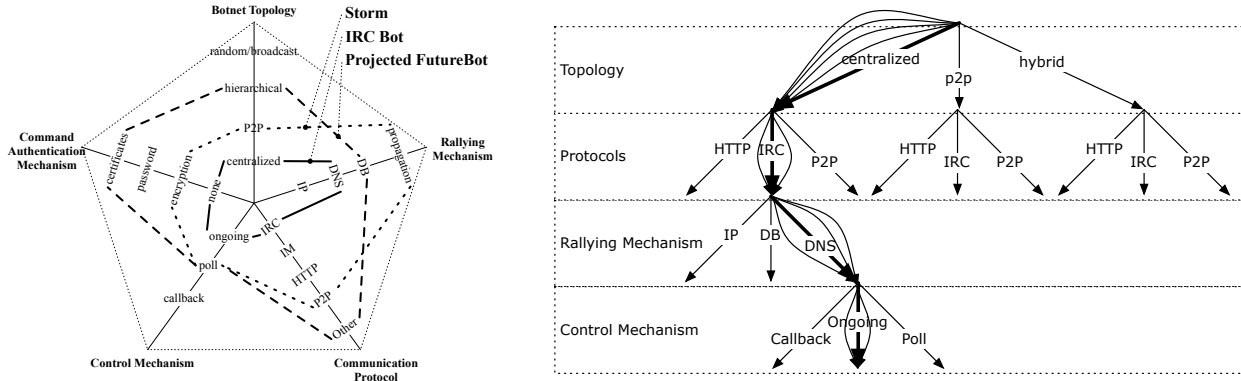
Figure 2(a) shows an overview of the SLINGbot system. A Botnet Scenario Driver acts as an out-of-band control mechanism that drives botnet activities and behavior. The actual botnet, including C2 traffic, is separate from the Scenario Driver. Individual bots (and controllers) within the botnet are implemented with the SLINGbot Composable Bot Framework.

### 3.1. Botnet Scenario Driver

The SLINGbot Scenario Driver is a meta-framework for deploying botnet experiments. This framework consists of the following applications: a **Scenario Manager**, which is deployed on a single network-visible host to direct the activities of the entire botnet by interpreting commands input by the user and communicating with various Scenario Server applications, and **Scenario Servers** on each host that a bot or controller is to be deployed. The Scenario Server acts as a middleman by receiving commands from the Scenario Manager to spawn and configure one or more local bot and/or controller processes. Also, a Scenario Server passes user instructions (via the Scenario Manager) to botnet controllers; these commands direct the activities of the botnet.

The entire flow of an experiment scenario, including all bots and controllers that are deployed and actions taken by the botnet controllers, is specified in an XML scenario file provided by the system user. The Scenario Manager parses this configuration file and uses its contents to instruct (via remote procedure call) the distributed set of Scenario Servers as to when to spawn bots, botnet controllers, and any other processes required by either the experiment infrastructure (e.g., process that models user behavior) or botnet (e.g., IRC server).

It is important to note that the Botnet Scenario Driver provides an infrastructure to manage botnet experiments—it is not an actual part of the botnet. The scenario frame-



**Figure 1. Command and Control (C2) Features: (a) C2 Feature Space – Radial axes are largely subjective; (b) C2 Feature Space Exploration**

work is used to simulate bot propagation (which is not implemented due to the restrictions described in Section 2.1) and to direct the botnet controller(s) as to how to lead and instruct the botnet. It does not simulate botnet C2 traffic, but instead controls the initiation and timing of botnet controller activities, which then results in authentic C2 traffic and behavior within the botnet.

### 3.2. Composable Bot Framework

SLINGbot includes a component-based framework with which to create benign botnets that are customizable from a library of components that encapsulate bot instruction and C2 functionality. The five SLINGbot component types are (1) instruction handler, (2) topology manager, (3) rally mechanism, (4) control mechanism, and (5) protocol manager. SLINGbot implements the components as programmable modules.

The communication components identify distinct decision points within a bot that represent code functionality and/or interfaces for which design choices significantly change the manner in which a botnet is organized, and thus, the manner in which a botnet carries out C2 communication. SLINGbot implements the components as programmable modules that can be used to experiment with and measure the effectiveness of different C2 techniques.

Included with the SLINGbot system is a base class for each of the five component types. These base classes define the common interface (i.e., API) between the components. Sub-classes must be defined that inherit the common interface and provide specific implementations. Interactions between SLINGbot components must only be via the defined common interface. The definition of and adherence to a common interface amongst SLINGbot components allows for a “mix-and-match” behavior.

Figure 2(b) provides another visualization of the Composable Bot Framework. SLINGbot (as with all bot software) is built as an application on top of the transport layer (which can include higher-layer protocols such as HTTP, IRC, AIM, etc.).

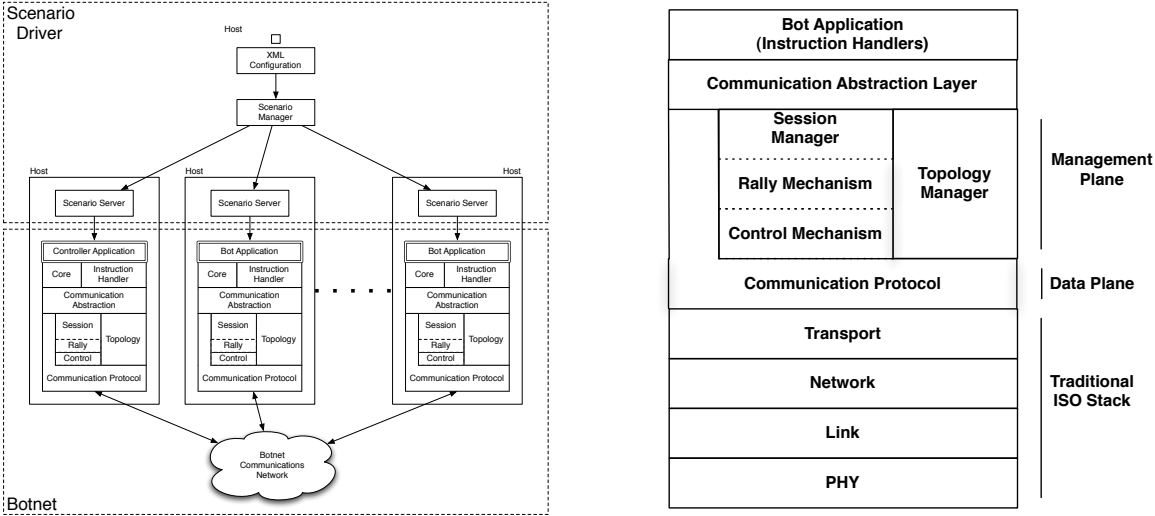
The lowest layers of the SLINGbot stack are divided into two planes: the **Management Plane**, which is responsible for maintaining connectivity to the botnet C2 infrastructure, and the **Data Plane**, which is responsible for providing services that enable the exchange of commands and responses among individual bots and controllers within the botnet.

The management plane is further subdivided into two modules: the **Topology Manager**, which manages bot peer information, performs tasks normally associated with routing, and establishes and manages any botnet communications hierarchy, and the **Session Manager**, which establishes and maintains connections to the botnet. Much of the Session Manager functionality is performed by two sub-modules—the *Rally Mechanism* (means of locating the botnet) and the *Control Mechanism* (means by which a bot retrieves commands).

The data plane consists of the **Communication Protocol** module that provides the protocol encapsulation/decapsulation capabilities.

The *Communication Abstraction Layer*, as seen in Figure 3(b), hides the details of C2 communication from the core *Bot Application*. The *Communication Abstraction Layer* defines an interface to the lower layer modules that provides a very simple set of services centered around the tasks of transmitting and receiving bot instructions and their responses.

The *Bot Application* layer encompasses the core “intelligence” of a bot, and is largely provided by a software module that contains a set of instruction handlers. These handlers are invoked upon receiving instructions (via the *Com-*



**Figure 2. System for Live Investigation of Next Generation botnets(SLINGbot): (a) SLINGbot system overview; (b) Composable Bot Framework**

communication Abstraction Layer), and perform actions based on the instruction type and parameters. The instruction handler software module may easily be updated and expanded, providing a convenient and modular means to capture the core functionality of a bot.

#### 4. SLINGbot Bot Variants

A number of bot variants have been implemented within the SLINGbot framework. In this section, we discuss the different bot variants and capabilities in more detail.

##### 4.1. C2 Topology

A botnet may consist of a very large number of distinct hosts, i.e., bots. These bots must be organized in some manner such that the botnet controller can direct them as a group rather than dealing with each bot individually. The botnet topology affects the amount of direction the controller has over the botnet, the speed and simplicity of sending new instructions, and the botnet’s resilience to disruption.

SLINGbot currently implements four bot variants that make use of three main C2 topologies:

**Centralized** – The simplest model in which all bots report to a single location, such as an IRC server or HTTP drop, for instruction.

**Distributed (or Peer-to-Peer)** – The term *peer-to-peer (P2P)* encompasses a wide range of architectures which may have centralized infrastructure components, distributed components, or no central infrastructure at all. For SLINGbot purposes, a botnet with P2P topology is one in which

bots obtain instructions by communicating directly with other similar bots, that is, in which one of the normal functions performed by bots is to serve commands to other bots.

**Hierarchical** – This model combines the scalability of distributed P2P command distribution with the control of a centralized topology. A hierarchical topology involves deploying a number of distinct servers with a central control; servers may act as bots as well. This adds a layer of indirection between the botnet controller and the bot-accessible servers.

In the following, we describe each of the bot variants contained in the SLINGbot framework.

**IRC Bot** – The IRC botnet uses a centralized topology with an IRC server at the hub facilitating the exchange of instructions and responses. Essentially, instances of the IRCBot listen for instructions generated by the IRCController on a specific IRC server and channel via the IRC PRIVMSG<sup>1</sup> command. Instruction handlers process the botnet instructions and may generate responses that are communicated back to the IRCController in the same manner.

**TinyP2P Bot** – The TinyP2P botnet uses the TinyP2P communication protocol [5]. Essentially, TinyP2P bots form a distributed, peer-to-peer, interconnected mesh network that exchanges command and response files (originated by controllers and bots, respectively) via XML-RPC requests, and use instruction handlers to process the instructions contained within those files.

A set of TinyP2P bots configured with the same pass-

<sup>1</sup>An IRC PRIVMSG is a broadcast message to a specified channel.

word is considered to be a TinyP2P botnet. The servers initiate contact and exchange remote server information with each other. In steady state, all servers know about and communicate with all other servers within the network. In other words, a TinyP2P network is an interconnected mesh network.

A TinyP2P bot polls for command files. All files located in a remote server directory that match a local command file pattern and that are not already present in the local directory are downloaded and scheduled for processing. Command processing is a simple matter of parsing the file for individual bot instructions/responses and then executing those messages.

**Kademlia bot** – Kademlia [9, 7, 12] is a peer-to-peer algorithm that provides a *Distributed Hash Table* (DHT) abstraction. Each Kademlia node is assigned a random 160-bit ID, and files are cached at the nodes whose IDs are closest to the file's key (using an XOR distance metric unique to Kademlia). Our Kademlia bot extends the *Entangled* Kademlia implementation, written in Python [2].

The Kademlia botnet uses a variation on the TinyP2P botnet's command file protocol. The Kademlia rallying mechanism simply supplies a list of IP addresses of existing Kademlia nodes to a new bot using its configuration file or the Scenario Manager. A Kademlia node needs the address of only one live node to bootstrap its connection to the network.

**Hierarchical Kademlia bot** – The Hierarchical Kademlia bot extends the base Kademlia bot. Each level in the hierarchy consists of a set of clusters or islands of bots. These clusters use Kademlia for intra-cluster communication. Each cluster has a leader node which is responsible for communicating with other leader nodes in the next level up in the hierarchy. The leader nodes thus facilitate inter-cluster communication.

Hierarchical schemes combine the benefits of robust peer-to-peer communication with the increased stealth that comes with having a single leader responsible for inter-cluster communication (thus decreasing the overall chatter between clusters).

## 5. Conclusions

We have presented our SLINGbot framework which facilitates the construction of *benign* botnets with varying command and control structures to enable researchers to generate simulated ground truth in a controlled, safe, and repeatable manner for current and potential future botnet threats. This simulated ground truth can be used to characterize various botnet command and control structures and then to develop effective defensive techniques.

SLINGbot has a number of advantages over current approaches to this problem. SLINGbot permits controllable,

repeatable experiments using current and potential future botnet command and control techniques. SLINGbot is also extensible, encouraging the use of shared libraries of botnet modules.

SLINGbot is currently being adapted for deployment on the Department of Homeland Security/National Science Foundation funded cyber-DEfense Technology Experimental Research laboratory (DETER) Testbed. SLINGbot is currently being used to characterize various botnet command and control architectures. Future work will present the results of this characterization.

## 6. Acknowledgments

The authors would like acknowledge the support from the Cyber Security Program Area of the Command, Control and Interoperability Division within the Science and Technology Directorate of the U.S. Department of Homeland Security and the U. S. Air Force Research Laboratory. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the U.S. Department of Homeland Security or the U. S. Air Force.

## References

- [1] Know your enemy: Fast-flux service networks. Technical report, July 2007.
- [2] F. Aucamp. Entangled: available at <http://entangled.sourceforge.net>.
- [3] P. Barford and M. Blodgett. Toward botnet mesocosms. In *Proceedings of First Workshop on Hot Topics in Understanding Botnets*. USENIX Association, April 2007.
- [4] D. Dagon, G. Gei, C. Zou, J. Grizzard, S. Dwivedi, W. Lee, and R. Lipton. A taxonomy of botnets. Technical report, Georgia Institute of Technology, 2004.
- [5] E. Felton. The tiny p2p protocol: available at <http://www.freedom-to-tinker.com/tinyp2p.html>.
- [6] A. Fettig. *Twisted: Network Programming Essentials*. O'Reilly, 2005.
- [7] J. Grizzard, V. Sharma, C. Nunnery, B. B. Kang, and D. Dagon. Peer-to-peer botnets: Overview and case study. Technical report, Johns Hopkins University, 2007.
- [8] E. Karim, A. Walenstein, A. Lakhota, and L. Parida. Malware phylogeny generation using permutations of code. *Journal in Computer Virology*, 2008.
- [9] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the xor metric. pages 53–65, 2002.
- [10] Symantec. Symantec global internet security threat report, April 2008.
- [11] TrendMicro. Taxonomy of botnet threats. Technical report, TrendMicro, Nov 2006.
- [12] Wikipedia. Kademlia: available at <http://en.wikipedia.org/wiki/kademlia>, Oct 2007.