Journal of Information Assurance and Security.
ISSN 1554-1010 Volume 6 (2011) pp. 503–511

IRC BotChallenger: Creating Botnet-Resilient Networks

M. Y. Mahmoud[1] **and Ashraf Matrawy**[2]

Carleton University, Department of Systems and Computer Engineering,
Ottawa, ON, K1S 5B6, Canada
([1]mimam, [2]amatrawy)@sce.carleton.ca

Abstract: **This paper presents a new approach to eliminate the effect of botnets by disrupting their command and control (C&C) communications. The focus is on botnets that employ the IRC (Internet Relay Chat) environments as IRC is the most common platform for botnet Command and Control communication. However, we see this work extendable to other botnet communication platforms such as HTTP and Peer-to-Peer botnets. In this approach we use a system we call *BotChallenger* to intercept outgoing and/or incoming IRC traffic and subject it to user's approval.**

The novelty of this work is that it doesn't try to identify botnets, rather it actively disrupts their communication. Therefore, it is expected that this *BotChallenger* application level approach should work with various types of botnets that use IRC for their C&C communication, which makes it capable of disrupting new IRC botnets without the need to learn about or identify them.

Two user profiles are simulated based on the presence of nodes adopting the *BotChallenger* in a network. The simulation results suggest that as the percentage of nodes adopting the *BotChallenger* in a network increases, the percentage of C&C messages passing through decreases -which leads to complete disruption of a botnet even if the network is still infected. It also shows that the number of useable computers is proportional to the percentage of *BotChallenger* deployment. This gives a chance for other intervention methods to disinfect the network while the botnet is in an ineffective state.

Keywords: Botnet, IRC, command and control communications, SPAM

I. Introduction

Botnets are known to be one of the most dangerous Internet security threats. A botnet is comprised of a network of hosts (bots) waiting for a command from a controlling node (botmaster). Botmasters gain control of hosts by infecting them with robot software (botnet code). After that, botmasters can communicate with their botnet with a command and control (C&C) communication protocol [21].

As Fig. 1 shows, botnet infection follows a life-cycle. First, the botnet initial infection phase will take place when botnet node(s) exploit a victim node's vulnerabilities to infect the victim's machine. Then, once the target host is infected, it will execute a code that downloads and installs the botnet malware on the victim's machine. After that, the botnet program establishes a connection with the botnet's C&C server. Once a connection with a C&C channel is established, the victim becomes part of the botnet. Botnet nodes will listen to the C&C channels waiting for botnet updates and maintenance or other malicious commands from the botmaster [8] [10].

Theoretically, botmasters can command their botnets to do anything. They can use their botnets to gain computational power, run distributed denial of service (DDoS), attacks, and send spam emails. Furthermore, botmasters usually use their botnets for advertising, spying, online fraud, phishing and for identity theft. It is estimated that 80% of the Internet spam is generated from botnets [22]. Studies show [1] that more than fifteen percent of the computers connected to the Internet are infected and used by botnets. It has been documented that one botnet has infected and had more than 400,000 computers under its control [21] [8].

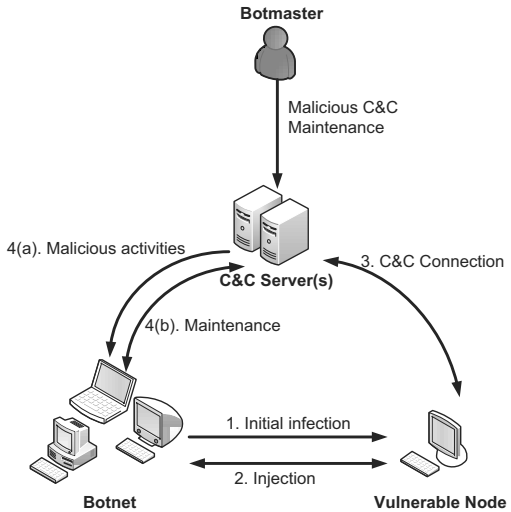

Figure. 1: Botnet Life-Cycle.

Detecting botnets is a challenging task. Therefore, 100% detection of all known and unknown botnets is not possible because botnets have very manipulative behaviors. Botnets'

hibernation and their very low activity traffic volume keep them hidden. Furthermore, botnets may use unusual destination port or encrypt their C&C traffic to stay undetected [12] [17].

The IRC [18] protocol is a simple, distributed and scalable protocol designed to facilitate a chatting environment. The IRC protocol's simplicity and reliability made it a widely available public exchange domain that facilitates almost instant communication. Therefore, the earliest and most common botnets use IRC for C&C communication [17] [6] [13] . Although botnet designers are increasingly moving towards other protocols, IRC remains to be the most used protocol for botnets' C&C messaging [6] [19].

The following classification of some botnets detection systems are given to help explaining this paper:

- **Online Vs. Offline Systems**: In online sytems the traffic is fed directly from the network to the detection system whereas in offline systems traffic traces are collected and then fed to the detection system.
- **Real-Time Vs. Non Real-Time Systems**: Real-time systems are online systems that are capable of producing results in real-time whereas non real-time systems might be online, but they need some time to produce their results.
- **Active Vs. Passive systems**: Active systems will actively intercept, block and/or alter botnet malicious traffic and does not wait for external intervention. On the other hand, passive systems detect the possibility of a botnet threat and then alert system administrator(s) or/and the IDS (Intrusion Detection System).

Most of the research on botnet detection and defense measures focuses on passive systems and algorithms. Passive monitors or systems - though might be real-time systems [11]- rely on system administrator(s) or/and IDS to take appropriate action in eliminating any threats. An active defence system, however, will actively affect the network traffic. Therefore, active defence systems could be very disruptive if they have many false positives.

In this research, the focus is on IRC-based botnets because they are the most common botnets. Peer-to-Peer (P2P) and HTTP-based botnets' detection is a work in progress. The *BotChallenger* concept will be extended to other types of botnets (including those which use UDP for their C&C communication) in our ongoing work. The idea of *BotChallenger* is presented to actively intercept and challenge any IRC messages then blocks botnets C&C messages. The performance of this system relies heavily on identifying the C&C communication protocol (IRC in this case). If a botnet's C&C traffic is successfully blocked by the *BotChallenger*, the botmaster would no longer be able to communicate with its bots. Therefore, no matter how smart and manipulative a botnet is, being able to disrupt its C&C communication would make it useless to the botmaster and the botnets' threat will be eliminated.

The main **contribution** of this work is that the emphasis is on C&C communication disruption not on blocking malicious botnet activities. In other words, looking at botnet lifecycle (Fig. 1), the *BotChallenger* will not try to block the first two steps (infection and injection) rather it will block the other three steps (connection, commands and maintenance). C&C communication disruption renders the botnet nodes in the network unable to communicate with their botmaster. The IRC *BotChallenger* is a distributed solution that is capable of protecting the network from new unknown IRC botnets without the need for botnet identification.

The rest of this paper is organized as follows; section II highlights some related research work on defense measures against botnets. Section III states the problem description and the intuition behind of the proposed solution. After that, section IV drives a description of the *BotChallenger* system proposed. Then, in section V, two user profiles are used to study the performance of the *BotChallenger*. Simulation assumptions, environment, and results are also illustrated. Finally, the paper's conclusion is in section VI and suggestions for future work are in section VII.

II. Related Work

Most of the literature on the defense against botnet focuses on botnet detection methods. Most botnets can be detected based on their behavior and/or by detecting their DNS (Domain Name Server) traffic.

Botnets rely heavily on DNS, and they usually use Dynamic DNS (DDNS) for their C&C servers. In [5], the authors propose an anomaly-based technique to detect botnets by monitoring their DNS traffic. The botnet DNS traffic has a unique feature that the authors define as group activity. Botnet members can be detected by using the group activity property of botnet DNS traffic while bots are connecting to their server or migrating to another server. Their algorithm is capable of detecting unknown and/or encrypted botnets regardless of botnets' protocol or structure. However, the false positive and false negative ratios depend on a threshold that can be tricky to determine in large networks.

Mazzariello [17] proposed an IRC detection method based on a model of IRC user behavior. The main idea is to find a way to differentiate human IRC traffic and automated IRC traffic using classifiers. Response times, vocabulary and language complexity can be used in this classification. The author used Support Vector Machine (SVM) [23] and J48 [20] decision trees in the experiment. Though the experiment was very successful and the author in [17] was able to separate botnet C&C traffic with almost 100% accuracy, trying to identify the correlated botnets' responses could lead to falsely identifying a legitimate node as C&C server due to large number of clients connecting to it. Moreover, the author was not sure whether these results were because of the algorithm or the dataset used to test the algorithm. Unfortunately, when it comes to the response time, botnets could be designed to adjust their response time so it mimics human response.

Akiyama et al. [2] proposed a three metric measure for botnet detection. They assumed that bots belonging to a botnet will have regularities in *relationship*, *response* and *synchronization*. In their experiment, Akiyama et al. [2] analyzed the collected traffic and found that high density structure of hosts is related to the relationship among bots. Furthermore, the response time must be a significant factor in detecting botnets. They also concluded that dynamics of measured traffic is a component of the synchronization of

bots. Though this three-metric detection looks promising, it has some limitations. Trying to identify the botnet relationship could lead to falsely identifying a legitimate node as botmaster due to a large number of clients connecting to it. When it comes to the response time, a botnet could be designed to adjust its response time so it mimics human response.

In [3], Binkley et al. suggested an anomaly-based detection algorithm for IRC botnets. Their algorithm is based on the fact that all IRC hosts will be grouped based on their channel name. Then an IRC channel will be considered "evil" if it has most of its hosts performing TCP SYN scanning. This algorithm does not need botnet signature for detection and it can detect unknown IRC botnets. However, this algorithm cannot detect encrypted botnets. Furthermore, this is not a real-time detection algorithm as it cannot detect botnets if they are not running an attack.

Chi et al. [4] proposed another metric for botnets detection by monitoring the massive malicious traffic. According to them, as botnets are based on the IRC protocol, they must wait for the botmaster to send commands. As soon as the master sends an attack command to the bots, thousands of bots prepare to attack the victim at the same time. As a result, massive malicious packets arrive at the victim simultaneously. By analyzing the flows of packets to the victim, defenders then can block the malicious traffic and stop the botnet. The authors of [4] proposed a method to detect the botmaster during an attack by starting from the victim and working backwards through the routers. This algorithm [4] is for real-time detection and does not require botnet signature for detection. It is capable of detecting unknown botnets and has low false negative rate and low computational power. On the other hand, this algorithm is incapable of detecting encrypted botnets. The detection is performed during botnet's attack, which is too late. Moreover, it requires an IDS to be installed on the victim hosts.

Goebel et al. proposed 'Rishi' [9]. It is a detection system based on the used IRC channel nickname for the detection of botnets. According to the them, the bot contact its C&C directly after an infection. Every bot connected to its IRC server has a username called the nickname. The detection method here relies on the IRC nicknames. The detection technique is based on passively monitoring the network traffic for unusual or suspicious IRC nicknames. Although Rishi is a low cost botnet detection system, it is a non-real-time passive system that can only detect un-encrypted IRC botnets.

Gu et al. proposed three systems to detect botnets. BotHunter [11] is a real time "evident-trail" approach that detects botnet infection using IDS-driven dialog correlation based on a predefined botnet infection life-cycle (i.e. target scanning, infection exploitation, botnet binary downloading, botnet code execution, C&C communication and outbound scanning). This system is capable of detecting bots regardless of the C&C structure and network protocol as long as the bot behavior follows a predefined infection cycle dialog model. BotHunter is a real-time, protocol and structure independent detection system capable of detecting unknown botnets with few false positives/negatives. However, it is a passive system that requires botnet to follow a predefined infection cycle dialog model to be detected and it is not capable of detecting encrypted botnets.

BotSniffe [12] is an anomaly-based algorithm designed to detect botnet C&C traffic in Local Area Networks (LANs.) The system monitors LAN traffic for botnet messages and activity responses. These messages carry commands from botmaster to bots, whereas activities are botnet malicious tasks. Bots usually respond the same way in a response crowd. The system then utilizes botnet's traffic spatial-temporal correlation and similarities without any prior knowledge of a botnet. Botnets need to connect to C&C server to get commands and thereby need to act upon on receiving a command, so they have much stronger synchronization and correlation than humans. This algorithm (Botsniffer) does not require prior knowledge of C&C traffic signatures and can detect encrypted C&C communication. It does not need a large number of bots to work and has low false positive and false negative rates. However, it is not a real-time system that can detect botnet in LANs only.

BotMiner [10] is a detection frame work comprised of three main phases; monitoring, clustering, and correlating. BotMiner clusters similar communication traffic and similar malicious traffic and perform cross correlation to identify the hosts that share both similar communication patters and similar malicious activities. BotMiner has low false positive rate. It is independent of botnet C&C protocol and structure, and requires no botnet signature. Therefore, it can detect encrypted, centralized (IRC and HTTP) or distributed (P2P) botnets. On the other hand, it is a passive non-real-time detection system.

All detection methods mentioned above are online algorithms in which the traffic is fed directly from the network to the detection system. As for offline detection methods - where traffic traces are collected and then fed to the detection system- the authors of [22] and [16] analyzed network traffic traces trying to isolate botnet traffic.

Strayer et al. [22] analyzed traffic traces in pipeline manner. They start by filtering out the flows those are probably not part of botnets. Then, they correlate surviving flows looking for any indication of being in same botnet. After that, flow information is further examined to find out if they share any hubs. The goal of this paper [22] is to detect C&C flows in IRC based botnet. They started by filtering the traffic to ignore all traffic that cannot be C&C. Then, they classified the traffic into IRC and non-IRC flows. Finally, the Topological Analysis stage takes place to identify C&C traffic.

The internetwork research department at BBN technologies proposes a Machine Learning (ML) technique to detect IRC-based botnet C&C traffic [16]. First, they divide the traffic into IRC and non-IRC traffic. Then, they look further into the IRC traffic trying to detect botnet C&C traffic. To do so, they isolate the flows that likely contain C&C traffic, correlate them to group flows that belong to the same botnet in order to identify the C&C host.

This method has high false positive rates, high computational overhead and it is not in real time.

III. Problem Statement

Theoretically, botmasters can command their botnets to do anything. They can use their botnets to gain computational

power, run distributed denial of service (DDoS) attacks, and send spam emails. Furthermore, botmasters usually use their botnets for advertising, spying, online fraud, phishing and for identity theft. We look at this problem from the end user's point of view (organizations or individuals). By blocking botnets, the end user would gain some of the following advantages:

- Protect your identity and financial data: By stopping spying, identity theft, E-mail address harvesting, click fraud.
- Protect yourself from trouble with the law: By stopping hosting of phishing or illegal sites, click fraud, attacks, illegal e-mails.
- Preserve bandwidth: By stopping forwarding botnet traffic (Fast-flux), e-mail spam, Adware, attacks, hosted sites.
- Preserve computer performance: By stopping the botnet from adding or deleting autostart applications, running or terminating programs, hosting sites, port scanning, Adware, attacks.

Most of the research on botnet defense mechanisms is focused on identifying and detecting *known* botnets. Obviously, these mechanisms will not work against new botnet designs. Some researchers proposed detection systems that are capable of detecting unknown new botnets [11]. However, all these systems are passive systems that are based on either anomalies of nodes' traffic or DNS queries. Smart design of botnets could reduce the efficiency of these systems. Countermeasures proposed in the literature have one or more of the following disadvantages:

- Botnet signature requirement.
- Predefined models [11].
- Passive systems [11] [12] [10] [16] [22].
- Offline systems [16] [22].
- Non real-time detection [10] [12] [16] [22].
- High false positives/nigatives rates [9].
- High computational power [10].
- Thresholds that need to be adjusted according to networks' user behavior [5].

The *BotChallenger* is designed to eliminate the threat of botnets and avoid all previously mentioned disadvantages as explained in section IV.

IV. The Design of BotChallenger

As mentioned earlier, the novelty of this approach is the ability to eliminate the effect of botnets without trying to identify them, by disrupting their C&C communication in IRC networks. In this approach, on every node, an instance of the *BotChallenger* is used to challenge any traffic associated with IRC process. Its purpose is to make sure that the user is aware of the IRC communication being processed by his/her computer. This *BotChallenger* will block all IRC traffic that is not authorized by the user. Thus effectively block C&C traffic rendering the botnet unable to function. An important part of the design of *BotChallenger* is to be able to accurately identify C&C communication protocol (IRC in this case).

A. Detecting Botnets Vs. Disturbing their C&C Traffic

Assume the scenario where we need to protect an organization network of 500 computers against IRC botnets. Looking at the countermeasures explained in section II, offline algorithms (e.g. [16] and [22]) are not practical for this scenario because traffic traces need to be collected first, then, these algorithms are applied to determine if these traffic traces contain botnet traffic. On the other hand, online algorithms (e.g. [2], [3], [9], [11], [12], [10], and [17]) are mostly passive in the sense that once a botnet activity is detected, it is up to the system administrator to take the necessary actions. This might include running cleanup tools to remove known botnets or taking infected systems off the network until cleaning software is developed/updated or until the botmaster is shutdown. These are not cost effective choices. The solution proposed in [4] is an active one where botnet's traffic is blocked router-by-router starting from the victim's computer. Unfortunately, this trace-back algorithm works only during botnet attack, which could be too late for vital organization computers. Moreover, working under attack could be very challenging if the attack is affecting the network availability.

On the other hand, deploying the *BotChallenger* renders the botnet nodes in the organization unable to communicate with their botmaster. While the network administrator is developing/updating the clean-up tool, the botmaster will not be able to use these bots to harvest information, send SPAM, start DDoS attack,... etc. The only limitation will be the end users who intentionally allow their computers to communicate with the botmaster. We say "intentionally" because if the warning system is designed according to [7], the user will be protected 100% of the time (details are given in section IV-C). Keep in mind that the *BotChallenger* will also keep all systems protected 100% of the time during business afterhours because all IRC traffic well be blocked then.

Our simulation in section V-C shows that in the scenario adopting the *BotChallenger* solution in a network, the number of useable computers in that network will be much higher than it is in the scenarios where botnet detection systems are used. In our result, we refer to these systems as generic detection (GD) systems.

B. IRC Traffic Detection Methodology

To identify IRC communication, the keywords NICK, USER, JOIN, TOPIC, PRIVMSG and NOTICE are used. The *BotChallenger* would be able to detect any request to join an IRC channel using either NICK or JOIN keyword because every user needs to use these two commands to join an IRC channel. However, for better detection and to enable users to install the *BotChallenger* on botnet infected hosts and still be able to disrupt any ongoing botnet C&C communication, the other keywords could be used. PRIVMSG and NOTICE keyword are used to detect private messages communication after joining an IRC channel. Each of these keywords has a specific syntax. Therefore, the *BotChallenger* will be able to provide the user with the IRC channel name and/or the remote user's nickname and possibly real name to make the decision more realistic to the users.

On the other hand, the system keeps an updated list of all known encrypted IRC servers. This would give the *BotChallenger* the capability to challenge the any communica-

tion with these servers. However, there is a small possibility that the botnet is using the same IRC server the user is using to conduct legitimate IRC conversation. This will limit the *BotChallenger's* ability to distinguish between these two conversations. Furthermore, in order to avoid having a too noisy system that could force users to ignore all warnings for convenience, some measures are considered and explained in the following section.

C. The IRC BotChallenger Architecture

As Fig. 2 shows, the IRC *BotChallenger* incorporates user manageable white and black lists. The user has the ability to allow/block individual messages and/or individuals. This white/black listing can be done for one message, for the current session, or forever (for IRC username). Furthermore, the user has the ability to white or black list an IRC channel. In order not to challenge every packet passing to the application layer, and to keep track of current messages, sessions and/or IRC channels, a lookup table is constructed using the following packet header fields:

- Source IP addresses.
- Destination IP addresses.
- Packet ID.
- Protocol.
- Flags.
- Fragment.

The *BotChallenger* will intercept TCP traffic only (because we are dealing with IRC for now).

If the traffic is classified as TCP, the keyword above are used to detect if this traffic is IRC. If the traffic is encrypted, the remote host will be checked against the updated list of known encrypted IRC servers. Then, IRC traffic is checked against the white and black lists. While the traffic belonging

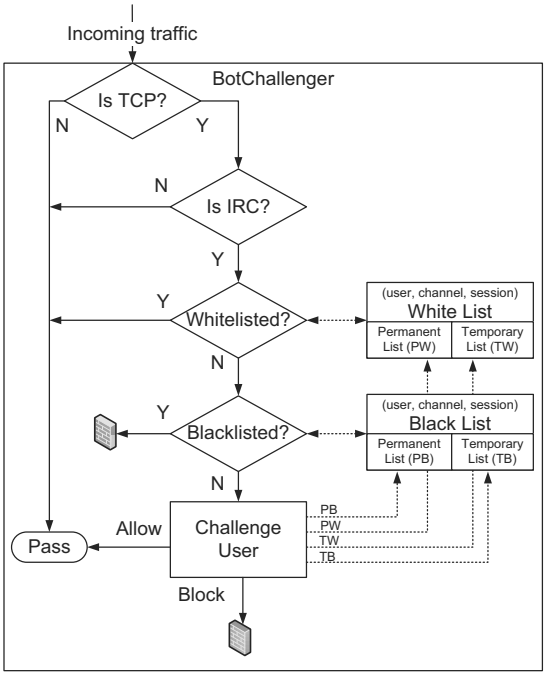

Figure. 2: IRC BotChallenger Architecture.

to whitelisted users, sessions or channels will pass, traffic associated with blacklisted users, sessions or channels will be blocked. The rest of IRC traffic (neither whitelisted not blacklisted) will be challenged by the user. The user then will decide whether to allow this traffic to pass or to block it. With the user decision, the white and black lists could be updated.

According to the study made by Egelman et al. [7], in order for a security warning to be effective, it needs to satisfy the following:

- It needs to capture the user's attention.
- It must capture the user's attention long enough to allow them to attempt comprehension.
- It should "fail safely". i.e. if the user did not read or understand the warning and clicked the default, he/she should be protected.
- It must give the user indication of danger and clearly suggest action(s).
- It should not be similar to less serious warnings.

They found that regardless of user's background, mental frame, or understanding level of the warning, 100% of them were protected by warnings that were designed to fulfill these conditions.

The *BotChallenger* is a distributed solution that does not suffer from singe point failure. It is capable of protecting the network from new unknown botnets because it disrupts the botnet's C&C communication instead of trying to detect them. We assume that the *BotChallenger* process is protected. i.e. the malicious software cannot inject their code to the *BotChallenger* process. *BotChallenger* process protection is beyond the scope of this research.

V. Simulation

A. User Profiles

According to Egelman et al. [7], an efficient design of the *BotChallenger* warning messages will protect all the users using the system 100% of the time.

The novelty of this proposal is that it fights botnets, not by trying to identify them like other algorithms, but by disturbing their C&C traffic. Therefore, even if a botnet is comprised of thousands of bot computers (bots in short), after adopting this *BotChallenger* the botmaster will no longer be able to control these bots or use them for computational power, running DDoS attacks or sending SPAM. To enhance the usability of the challenger, the default setting will be to block all IRC traffic if the user is not chatting on the IRC. Once a user starts an IRC chat, the challenger will switch to an interactive mode to challenge all IRC messages and/or sessions. To reduce the amount of unnecessary challenges, the user will have the option to trust all messages to/from a given IRC username for the current section.

Designing the *BotChallenger* to fulfill Egelman et al.'s [7] security warning conditions gives us two profiles.

- **BotChallenger**: normal users who read warning messages given from the challenger and respond accordingly. These users will block 100% of all suspicious IRC traffic and allow high percentage of legitimate IRC

to pass through. It is assumed that most of the network users fall into this category. This is reasonable even if most of the users do not know what IRC is, because the challenger messages could be very simple. For example, the challenger could send these messages to the user asking for a response;

Your computer is trying to join IRC channel $< ChannelName >$, if you are not running a chat session or trying to join an IRC channel, then most probably your computer is infected with a malware and you should block this communication.

Your computer is trying to send/receive a message to/from $< NickName >$ $(< RealName >)$, if you are not running a chat session or communicating with $< NickName >$, then most probably your computer is infected with a malware and you should block this message.

- **No BotChallenger**: Users without challenger. These users will allow all IRC traffic to pass through.

B. Simulation Setup

In this paper, we chose to simulate a botnet that is generating SPAM. Other malicious behavior such as DDoS could have been simulated easily. We do not see this as the main focus of the study. We are just trying to demonstrate the effect of the *BotChallenger* on the botnet's behavior. The Donbot [14] and Festi [15] SPAM botnets were studied to find the parameters needed for the simulation. The Donbot botnet generate an average of 8000 SPAM messages per hour per bot. That is about 134 SPAM email per minute per bot. To simulate the effect of the *BotChallenger*, we used SPAM emails as a measure for malicious traffic. The network behavior was simulated, where 500 nodes are set to send messages using the following assumptions:

- We assumed that throughout the simulation all traffic generation follows Poisson distribution instead of self-similar traffic.
- Botmaster generate SPAM C&C messages following Poisson message generation rate with mean of 1 minute (i.e. 60 simulation cycle).
- Each bot will generate 134 SPAM emails upon receiving the botmaster's C&C message (that is 8070 SPAM/hour/bot).
- Nodes are capable of sending non-IRC traffic and IRC traffic. Bots will send botnet IRC C&C traffic (bot traffic in short) in addition to the normal traffic.
- We assumed that every node has an input buffer of length 20 (messages), but no output buffer. Therefore, whenever there is a message burst, all messages will be released at the current cycle and may be buffered at the destination's input buffer if the destination buffer has enough capacity. Otherwise, messages will be dropped if there is no space in the destination's input buffer.
- User profiles assigned to nodes will stay fixed throughout the simulation. However, node's behavior will change if it gets infected with a botnet message. Hence, once a node is infected it starts to generate some botnet C&C traffic.
- We assumed no defence against infection. The main idea of the *BotChallenger* is not to identify the botnet or block the infection. We assumed an infection attempt 3% of the total simulation time. With every infection attempt 5% of the network nodes will be infected.

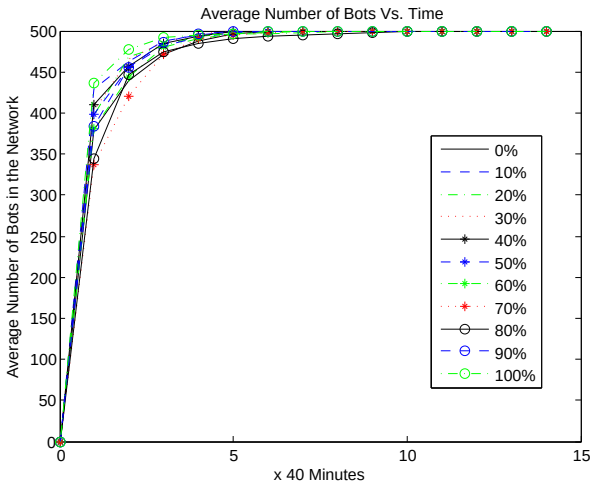

Figure. 3: Average Number of Bots Vs. Time for Different Percentages of Deployment.

At every simulation cycle all nodes will be checked. If a node has messages to send, the traffic will be challenged by its outgoing *BotChallenger*. Then, the messages will be queued into the destination's input buffer. If a node has messages in its input buffer, a message is dequeued after being challenged by the node's incoming *BotChallenger*. The main purpose of this simulation is to see if the *BotChallenger* can accurately block botnet C&C traffic and disrupt the botnet operation.

C. Simulation Results

Fig. 3 illustrates the relation between the number of botnet infected nodes versus time for different percentages of *BotChallenger* deployment. It shows that regardless of the percentage of deployment, eventually, all nodes will be infected. This is because we "intentionally" did not adopt any protection against botnet infection in our simulation.

SPAM generation in the network is used to simulate malicious traffic. Fig. 4 illustrates relation between the number of SPAM emails versus time for different percentages of *BotChallenger* deployment. It shows that as the percentages of network nodes adopting the *BotChallenger* increases, the total number of SPAM emails generated by the network decreases. Note that this is happening despite the fact that the network gets fully infected as shown in Fig. 3 because the *BotChallenger* renders the bot ineffective even while it is infected.

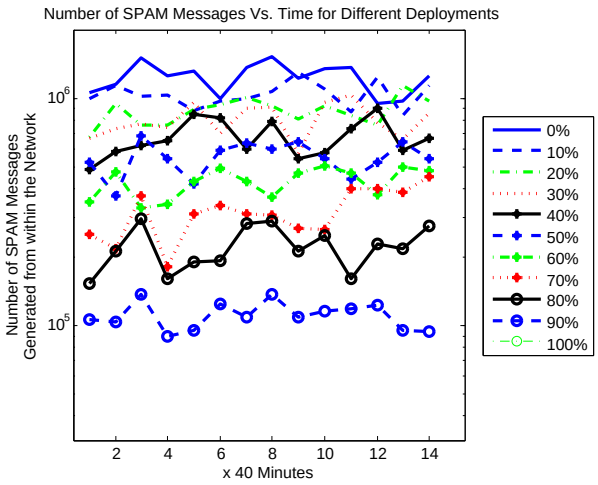

Figure. 4: Number of SPAM emails Vs. Time for Different Percentages of Deployment.

Logarithmic scale is used for this figure, therefore, the line for 100% deployment is not shown because it has value of zero. The fluctuation in the curves is due to the random distribution used to generate the SPAM. The SPAM messages burst are generated following Poisson distribution with mean of 134 message and exponential inter-arrival times with mean of 60 seconds. Fig. 5 show the same result from another perspective. It shows that the total number of SPAM emails in the network is inversely proportional to the percentage of node adopting the *BotChallenger* in the network.

To illustrate the advantage of using the IRC *BotChallenger* over other generic detections (GD) systems -where infected nodes are taken off the network for cleaning- we run the simulation comparing the *BotChallenger* against generic detection methods with different cleaning time values. We compared the number of useable computers in the network. Useable computers are those how are functional in the network without being a threat. We computed the number of useable computers according to the following formula:

$$For\ GD\ solutions: \quad nUseable = nNodes - nBots$$

$$For\ the\ BotChallenger\ solution: \quad nUseable = nNodes - nVBots$$

$$Where:$$
$$nUseable = Number\ of\ useable\ nodes.$$
$$nNodes = Total\ number\ of\ nodes.$$
$$nBots = Number\ of\ bots$$
$$nVBots = Number\ of\ bots\ without\ BotChallenger.$$

Fig. 6 shows the average number of useable computers on the network versus *BotChallenger* percentage of deployment and other generic detections systems. It shows that for the *BotChallenger*, the number of useable computers is proportional to the percentage of deployment. Note that this is happening despite the fact that the network gets fully infected as shown in Fig. 3 because we consider an infected node that is running the *BotChallenger* to be a usable node since the *BotChallenger* stops the bot' malicious activity by disabling its C&C. On the other hand, for other generic detection systems, the number of useable computers is affected by the cleaning time. It does not exceed 30% of total computers in the network in the best case scenario where the cleaning time is only 10 minutes. These values are obtained on a network simulation where 3% of the time there is an infection attempt to infect 5% of the computers on the network.

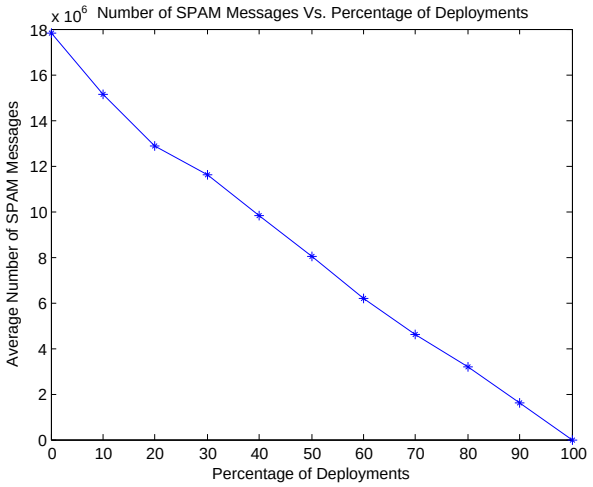

Figure. 5: Number of SPAM emails Vs. Percentage of Deployment.

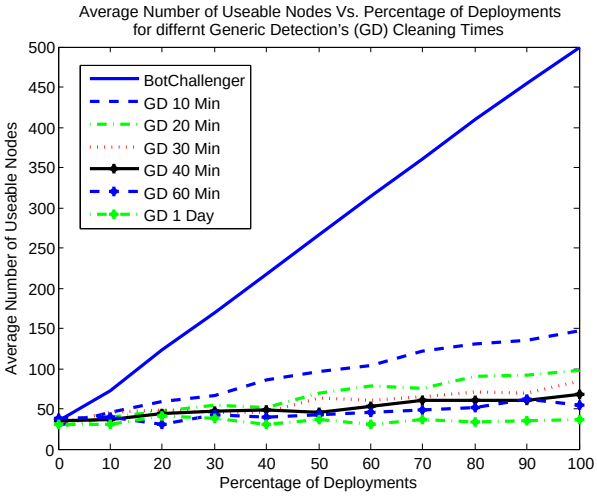

Figure. 6: Useable Nodes Vs. Percentage of Deployments

VI. Conclusion

Simulation results shows that as the percentage of nodes adopting the *BotChallenger* increases, the total number of SPAM emails generated by the network decreases. It also shows that by adopting the *BotChallenger*, infected machines will still be safely functional in the network until network admin runs cleaning utility to disinfect them. Thus, the number of useable computers on networks that adopt the *BotChallenger* is proportional the percentage of deployment. Further more, we should keep in mind that end users

who adopt the *BotChallenger* are protecting themselves from many threats that are more serious than SPAM (like protecting their identity and financial data, bandwidth, and computer performance).

The *BotChallenger* is a solution based on a simple concept. It is an application layer defense system that does not require any infrastructure changes (routers, switches, servers). It does not try to detect the presence of botnets. Rather, it makes a botnet useless to its botmaster by disturbing its C&C communication. It does so by making sure that the user is aware of any C&C communication protocol traffic (IRC in this case). The *BotChallenger* is an active reliable distributed approach that works for new unknown botnets that use the same C&C protocol.

VII. Future Work

As the simulation results demonstrate the concept of this research, the *BotChallenger* concept could be extended to cover other types of botnets. This could include HTTP C&C traffic, peer-to-peer C&C traffic, and encrypted peer-to-peer C&C traffic. Finally, the latency of the load of the *BotChallenger* needs to be evaluated. The *BotChallenger* application could be improved to enable system administrators to detect the presence of botnet(s) in their networks by collaborating all *BotChallenger* applications on network nodes with the network administrator's *BotChallenger* application.

VIII. Acknowledgment

Muhammad Mahmoud acknowledges King Fahd University of Petroleum & Minerals for its support. Ashraf Matrawy acknowledges support from Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] Emerging Cyber Threats. Report, Georgia Tech Information Security Center, October 2008.

[2] Mitsuaki Akiyama, Takanori Kawamoto, Masayoshi Shimamura, Teruaki Yokoyama, Youki Kadobayashi, and Suguru Yamaguchi. A Proposal of Metrics for Botnet Detection Based on Its Cooperative Behavior. In *Applications and the Internet Workshops, 2007. SAINT Workshops 2007. International Symposium on*, pages 82 –82, January 2007.

[3] James R. Binkley and Suresh Singh. An Algorithm for Anomaly-Based Botnet Detection. In *SRUTI'06: Proceedings of the 2nd conference on Steps to Reducing Unwanted Traffic on the Internet*, pages 7 –7, Berkeley, CA, USA, 2006. USENIX Association.

[4] Zhenhua Chi and Zixiang Zhao. Detecting and Blocking Malicious Traffic Caused by IRC Protocol Based Botnets. In *Network and Parallel Computing Workshops, 2007. NPC Workshops. IFIP International Conference on*, pages 485 –489, Dalian, China, September 2007.

[5] Hyunsang Choi, Hanwoo Lee, Heejo Lee, and Hyogon Kim. Botnet Detection by Monitoring Group Activities in DNS Traffic. In *Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on*, pages 715 –720, October 2007.

[6] Evan Cooke, Farnam Jahanian, and Danny McPherson. The Zombie Roundup: Understanding, Detecting, and Disrupting Botnets. In *SRUTI'05: Proceedings of the Steps to Reducing Unwanted Traffic on the Internet on Steps to Reducing Unwanted Traffic on the Internet Workshop*, pages 6 –6, Berkeley, CA, USA, 2005. USENIX Association.

[7] Serge Egelman, Lorrie Faith Cranor, and Jason Hong. You've Been Warned: An Empirical Study of the Effectiveness of Web Browser Phishing Warnings. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 1065 –1074, New York, NY, USA, April 2008. ACM.

[8] M. Feily, A. Shahrestani, and S. Ramadass. A Survey of Botnet and Botnet Detection. *Emerging Security Information, Systems and Technologies (SECURWARE '09). Third International Conference on*, pages 268 –273, June 2009.

[9] Jan Goebel and Thorsten Holz. Rishi: Identify Bot Contaminated Hosts by IRC Nickname Evaluation. In *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pages 8 –8, Berkeley, CA, USA, 2007. USENIX Association.

[10] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. In *SS'08: Proceedings of the 17th conference on Security symposium*, pages 139 –154, Berkeley, CA, USA, 2008. USENIX Association.

[11] Guofei Gu, Phillip Porras, Vinod Yegneswaran, Martin Fong, and Wenke Lee. BotHunter: Detecting Malware Infection Through IDS-driven Dialog Correlation. In *SS'07: Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, pages 1 –16, Berkeley, CA, USA, 2007. USENIX Association.

[12] Guofei Gu, Junjie Zhang, and Wenke Lee. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. In *In Proceedings of 16th Annual Network and Distributed System Security Symposium (NDSS'08)*, Reston, VA, USA, February 2008. The Internet Society.

[13] Anestis Karasaridis, Brian Rexroad, and David Hoeflin. Wide-Scale Botnet Detection and Characterization. In *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pages 7 –7, Berkeley, CA, USA, April 2007. USENIX Association.

[14] M86 Security Labs. Donbot, March 2009. Web Page, (http://www.m86security.com/labs/spambotitem.asp?article=899).

[15] M86 Security Labs. Festi, June 2010. Web Page, (http://www.m86security.com/labs/spambotitem.asp?article=1359).

[16] C. Livadas, R. Walsh, D. Lapsley, and W.T. Strayer. Using Machine Learning Technliques to Identify Botnet Traffic. In *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, pages 967 –974, November 2006.

[17] C. Mazzariello. IRC Traffic Analysis for Botnet Detection. In *Information Assurance and Security, 2008. ISIAS '08. Fourth International Conference on*, pages 318 –323, Naples, Italy, September 2008.

[18] J. Oikarinen and D. Reed. Internet Relay Chat Protocol, May 1993. RFC1459.

[19] Martin Overton. Bots and Botnets: Risks, Issues and Prevention. In *In Proceedings of Virus Bulletin Conference*. Virus Bulletin, October 2005.

[20] J. Ross Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[21] Daniel Ramsbrock, Xinyuan Wang, and Xuxian Jiang. A First Step towards Live Botmaster Traceback. In *Recent Advances in Intrusion Detection*, volume 5230/2008 of *Lecture Notes in Computer Science*, pages 59 –77. Springer, Berlin, Heidelberg, September 2008. Book, Chapter.

[22] W.T. Strayer, R. Walsh, C. Livadas, and D. Lapsley. Detecting Botnets with Tight Command and Control. In *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, pages 195 –202, Cambridge, MA, November 2006.

[23] Vladimir Vapnik, Steven E. Golowich, and Alex Smola. Support Vector Method for Function Approximation, Regression Estimation, and Signal Processing. In *Advances in Neural Information Processing Systems*, volume 9, pages 281 –287. MIT Press, 1996.

Author Biographies

Muhammad Mahmoud received the B.Sc. and M.Sc. degrees in computer engineering from King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia. He is currently a Ph.D. Candidate in the Department of Systems and Computer Engineering at Carleton University, Ottawa, ON, Canada. Muhammad research interests include system logic design, network security, computer networking, and Communication Network Protocols.

Ashraf Matrawy received the B.Sc. and M.Sc. degrees in computer science and automatic control from Alexandria University, Egypt, and the Ph.D. degree in electrical engineering from Carleton University, Ottawa, ON, Canada. Ashraf is currently an Associate Professor at Carleton University. He is a senior member of the IEEE, serves on the editorial board of the IEEE Communications Surveys and Tutorials journal, and has served as a technical program committee member of a number of IEEE and other international conferences. Ashrafs research interests include reliable and secure computer networking, and analysis of Internet traffic. His research has been supported by CFI/ORF, NSERC, OCE, Alcatel-Lucent Canada, and Solana Networks.