The ISC Int'l Journal of Information Security

ISeCure

January 2012, Volume 4, Number 1 (pp. 51–62)

http://www.isecure-journal.org

# BotOnus: An Online Unsupervised Method for Botnet Detection☆

Mosa Yahyazadeh [1] and Mahdi Abadi [1,*]

[1] *Faculty of Electrical and Computer Engineering, Tarbiat Modares University, Tehran, Iran*

ARTICLE INFO.

*Article history:*
**Received:** 9 November 2011
**Revised:** 24 January 2012
**Accepted:** 25 January 2012
**Published Online:** 30 May 2012

*Keywords:*
Botnet Detection, Botnet Lifecycle, Command and Control Channel, Online Clustering.

ABSTRACT

Botnets are recognized as one of the most dangerous threats to the Internet infrastructure. They are used for malicious activities such as launching distributed denial of service attacks, sending spam, and leaking personal information. Existing botnet detection methods produce a number of good ideas, but they are far from complete yet, since most of them cannot detect botnets in an early stage of their lifecycle; moreover, they depend on a particular command and control (C&C) protocol. In this paper, we address these issues and propose an online unsupervised method, called **BotOnus**, for botnet detection that does not require *a priori* knowledge of botnets. It extracts a set of flow feature vectors from the network traffic at the end of each time period, and then groups them to some flow clusters by a novel online fixed-width clustering algorithm. Flow clusters that have at least two members, and their intra-cluster similarity is above a similarity threshold, are identified as suspicious botnet clusters, and all hosts in such clusters are identified as bot infected. We demonstrate the effectiveness of BotOnus to detect various botnets including HTTP-, IRC-, and P2P-based botnets using a testbed network. The results of experiments show that it can successfully detect various botnets with an average detection rate of 94.33% and an average false alarm rate of 3.74%.



## 1 Introduction

Malware threats can mount a variety of attacks and give the attacker the opportunity to conduct different malicious activities. The significant increase of these threats on the Internet has attracted renewed interest in much of the recent research. Over the past few years, and contrary to the old-fashioned threats, malware attacks have evolved into better organized and more profit-centered endeavors [1]. This introduces a new type of threats that endangers thousands of network infrastructures around the world. At the heart of these threats lies a network of vulnerable hosts compromised and remotely controlled by an attacker. These hosts form a botnet.

The term "bot" is short for robot. A bot, also known as *zombie* or *drone*, is a malware that runs on a host typically unbeknownst to its owners, and carries out commands, sent by an attacker also called the *botmaster*. For all intents and purposes, bots are just viruses or worms that infect hosts to allow remote command and control by the botmaster. A *botnet* is a network of hosts on which the botmaster has somehow installed bots.

The value of a botnet to the botmaster depends on

☆ This article is an extended/revised version of an ISCISC'11 paper.

* Corresponding author.
Email addresses: m.yahyazadeh@modares.ac.ir (M. Yahyazadeh), abadi@modares.ac.ir (M. Abadi).

ISSN: 2008-2045 

its size. Not only it provides the botmaster with more processing power and Internet bandwidth for free, but also it allows the botmaster to launch coordinated attacks of unprecedented scale and complexity. According to a recent research report [2], botnets have become one of the biggest malware threats, responsible for a large volume of malicious activities.

Most Internet users are often not aware that their hosts have been compromised and become parts of a botnet. Moreover, the bots which take control of compromised hosts can evade typical antivirus scanners using obfuscation techniques. Therefore, there is a need to develop a specific method for detecting botnets, and the best policies can be taken only if the behavior of each stage of the botnet lifecycle is clearly understood. Generally, the lifecycle of botnets can be divided into three stages: formation, command and control, and attack.

In the formation stage, the botmaster spreads his bots via a propagation mechanism, such as exploiting propagation, email propagation, web browser propagation, and file sharing propagation [3], to compromise numerous hosts and install bots on them. The bots then try to join into the botnet via a rallying mechanism and form a botnet which meets the needs of the botmaster. Each bot is programmed so that it can connect to the botmaster via a *command and control* (C&C) channel, and update itself with new instructions and codes [4] to launch new attacks. In the command and control stage, the botmaster sends commands via C&C channels to remotely control his bots. Finally, in the attack stage, the bots perform different types of malicious activities such as launching distributed denial of service attacks, sending spam, leaking personal information, and defrauding pay-per-click advertisers [5].

Various methods have been proposed for detection of botnets, but most of them have some shortcomings, including (1) dependence on a specific C&C protocol, (2) lack of detection in an early stage of the lifecycle, (3) working offline, and (4) requiring labeled data for training. In practice, automated botnet detection in the command and control stage can be a difficult problem, because botnets often use existing common protocols (e.g., IRC, HTTP) for communication. Therefore, their traffic tends to resemble that of hosts not under control of the botmaster.

To tackle these shortcomings, we propose an online unsupervised method, called **BotOnus**, to detect botnets in the command and control stage. It extracts a set of flow feature vectors from the network traffic at the end of each time period and then groups them to some flow clusters by an online fixed-width clustering algorithm. Flow clusters that have at least two members, and their intra-cluster similarity is above a similarity threshold, are marked as suspicious botnet clusters. All hosts in these clusters are identified as bot infected. It also chooses some flow clusters to be eliminated from the flow cluster set, based on a cluster removal criterion.

The rest of this paper is organized as follows: Section 2 briefly reviews some related works. An overview of botnet C&C channels is provided in Section 3. Section 4 presents BotOnus. Experimental results are reported in Section 5. Finally, Section 6 sums up the discussion and draws the conclusions.

## 2 Related Works

In the recent years, various botnet detection methods have been proposed that can be classified according to two criteria: the stage of the lifecycle in which botnets are detected and the technique of learning involved. According to the learning technique, botnet detection methods can be classified into two categories [6]: supervised and unsupervised. Supervised botnet detection methods use a labeled dataset for training, which makes the process error-prone. By removing the need of labeling, unsupervised botnet detection methods facilitate online learning, and thus provide a higher potential to identify novel botnet activities. In this section, we discuss the different botnet detection methods in the related literature.

Livadas *et al.* [7] proposed a supervised botnet detection method to identify C&C traffic of IRC-based botnets. They first use some classification techniques like C4.5, Naive Bayes, and Bayesian network to distinguish between IRC and non-IRC traffic, and then incorporate these techniques to distinguish between botnet and real IRC traffic. The main drawbacks of this method are its dependence on a particular C&C protocol, and the need to labeled data for training.

Goebel *et al.* [8] proposed a method, called *Rishi*, which uses $n$-gram analysis with a scoring function and black/white lists to detect IRC-based botnets based on characteristics of their C&C channels. They first extract details of TCP packets that contain common IRC keywords (e.g., NICK) and then give packet nicknames to a scoring function. A nickname with a score higher than the set threshold will trigger an alarm. Rishi is limited to only detecting IRC-based botnets. Also, it has a relatively high false alarm rate, due to depending on regular expressions as signatures to automatically identify bot infected hosts. Therefore, if innocent people accidentally use a nickname containing suspicious strings, it will trigger a false alarm. Moreover, in the case where bots utilize nicknames composed out of random characters for which

a regular expression does not exist, it is unable to detect them. Wang *et al.* [9] presented another method to detect IRC-based botnets in an early stage. Based on their observations, bots that belong to the same botnet have nicknames, composed of a constant string and a random string, which basically follow the same pattern. Therefore, they defined the channel distance criterion to calculate the similarity of nicknames in one channel and used it to detect botnet channels in which nicknames have the same structure. The limitations of this method are similar to Rishi; however, it can detect unknown IRC-based botnets.

Gu *et al.* [10] proposed an unsupervised botnet detection method, called *BotMiner*, which is independent of botnet C&C protocol and structure. They first cluster similar communication traffic and similar malicious traffic, and then apply cross cluster correlation to detect hosts that share both similar communication patterns and similar malicious activities. The main drawbacks of this method are that it cannot detect botnets in an early stage, and does not work online.

Castle *et al.* [11] presented a method to detect botnets used for sending spam. They first extract the header information from an email message and enhance it with information from the envelope to generate a set of synthetic headers. They then replace the values of the synthetic headers with a limited range of tokens to generate a normalized template representation. Finally, they group the templates to obtain few clusters with a large number of email messages associated with them. These clusters are identified as suspicious botnet clusters. This method is simple but presents some serious drawbacks. It cannot detect botnets in an early stage, and is only able to detect botnets that are used for spamming.

Lee *et al.* [12] presented a method for detecting HTTP-based botnets using the degree of periodic repeatability. They found that if the degree of periodic repeatability for a client is low, it can be an HTTP bot that repeatedly connects to a C&C server at regular intervals to obtain new commands. This method can achieve a high false-alarm rate in case users employ automatic programs to connect to HTTP servers.

Choi *et al.* [13] proposed an online unsupervised botnet detection method, called *BotGAD*. They define a group activity as a key feature of botnets, and present a metric to detect botnets by monitoring group activities in DNS traffic. A botnet can evade this method when it performs DNS queries at one stage of the botnet lifecycle and never performs them again. Hence, it cannot detect botnets in an early stage and is only able to detect botnets that perform group activities in DNS traffic.

Xiaocong *et al.* [14] presented an unsupervised method that can detect centralized botnets in an online fashion. They first transform the captured network traffic into multi-dimensional feature streams and then use a data-adaptive clustering algorithm to group feature streams with high similarities. A correlation analysis is used as a similarity measure to detect suspicious botnet clusters.

Lu *et al.* [6] proposed an unsupervised method for detecting and clustering botnet communication traffic on network application communities. They first identify the network traffic into existing known applications using a C4.5 decision tree model, and then cluster the network traffic to find anomalous behavior on a specific application community based on the $n$-gram features extracted from the content of network flows. In real world, it is very difficult to identify all network traffic (e.g., traffic with the encrypted payload) into known applications on the large scale networks.

## 3 Command and Control

Unlike other malware which work separately, a botnet requires a form of communication infrastructure to be used by the botmaster to send out commands to the bots, and receive responses from them [6]. This communication infrastructure is known as the *command and control* (C&C) channel.

Botnets can be classified into three categories based on the structure of their C&C channels: *centralized*, *distributed*, and *hybrid*. In centralized botnets, the botmaster usually designates a host with high bandwidth Internet access as C&C server and then uses IRC or HTTP protocols to communicate with his bots. This structure is easy to construct and efficient in distributing the botmaster's commands. In an HTTP-based botnet, the botmaster sets up a web server to post his commands on, and the bots of this botnet poll this server periodically in order to obtain the most recent commands (Figure 1). In an IRC-based botnet, the botmaster creates a channel in an IRC server to post commands on, and then bots subscribe to this channel in order to obtain his commands. However, this structure presents a major weakness: the C&C server is a single point of failure. Taking down the C&C servers would cause all bots lose their communications with the botmaster.

In order to avoid the weakness of having a single point of failure, in distributed botnets, the communication infrastructure does not completely depend on only one C&C server, and it cannot be destroyed even by detecting a number of bots. In these botnets, the botmaster uses various P2P protocols to communicate with his bots (Figure 2). In a P2P-based botnet, a new

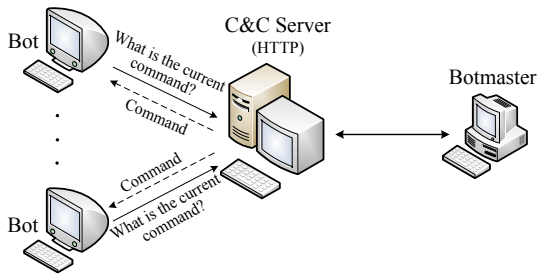



**Figure 1**. HTTP-based C&C channel

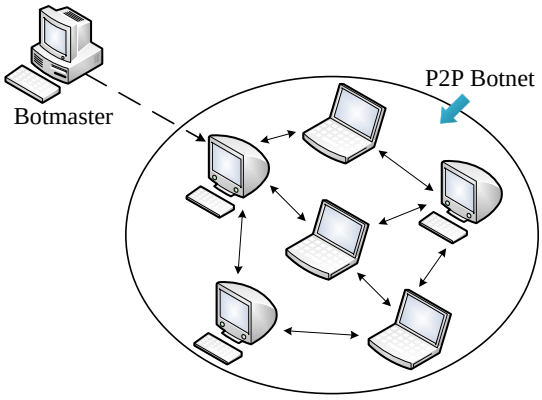



**Figure 2**. P2P-based C&C channel

bot-infected host needs an initial procedure, usually known as *bootstrap*, for finding and joining the botnet. There is two general ways for bootstrapping: using an initial peer list or using rendezvous server hard-coded in each bot to obtain the IP address of peer bots for joining the botnet. Instead of connecting with one C&C server, each bot connects with its peer bots and acts as both client and server. Therefore, if some bots in the botnet are detected, the botnet can still continue to operate under the control of the botmaster. The botmaster then injects his commands to some bots and they distribute these commands to all peer bots connected to them. Since distributed botnets are more resilient to defense countermeasures than centralized botnets, more botnets will move to use P2P protocols for their communication infrastructure.

Finally, hybrid botnets combine both centralized and distributed structures together or use a random structure to make themselves harder to be discovered and destroyed. For example, in Figure 3, there are two groups of bots including servent and worker bots. Servents (SERVer and cliENT) are bots with static non-private IP addresses, which act as both servers and clients, and are accessible from the global Internet. They use P2P connections and relay the botmaster's commands to worker bots. Workers are bots with dynamic or private IP addresses, which are behind firewalls or NAT devices such that they cannot accept incoming Internet connections [1]. They always connect to servent bots to obtain most recent commands. Hybrid botnets have more network stability and will become more important in the future as a larger proportion of hosts will reside behind firewalls or use private IP addresses.

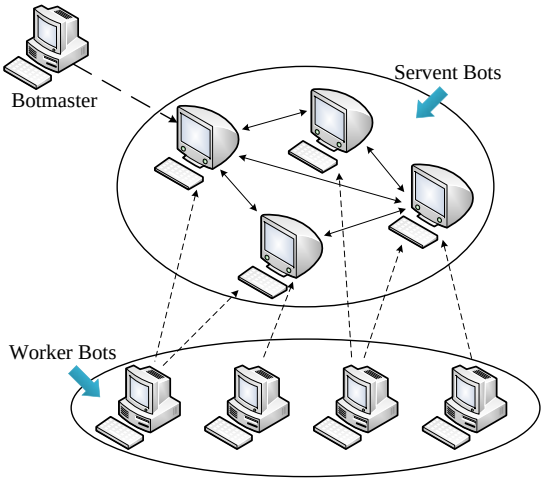



**Figure 3**. Hybrid C&C channel

## 4 Online Unsupervised Botnet Detection

In this section, we present an online unsupervised method, called BotOnus, for detecting botnets in the command and control stage. The aim of BotOnus is to detect a group of compromised hosts within a monitored network that are parts of a botnet. In the following, we first review the problem definition and then describe different steps of BotOnus.

### 4.1 Problem Definition

In order to present a general unsupervised botnet detection method, we need to study the basic characteristics of botnets. We thus start with the definition of a botnet.

**Definition 1 (Botnet).** A botnet is defined as a coordinated group of bots that are controlled via C&C channels and perform similar malicious activities [10].

The term "coordinated group" means that multiple (at least two) bots within a botnet receive the same commands and perform similar activities [10]. If the botmaster sends different commands to all bots separately via C&C channels, these bots are not considered as a botnet and are beyond the scope of this work.

### 4.2 Steps of BotOnus

BotOnus consists of four main steps: whitelist filtering, flow stream extraction, online clustering, and botnet detection (see Figure 4).

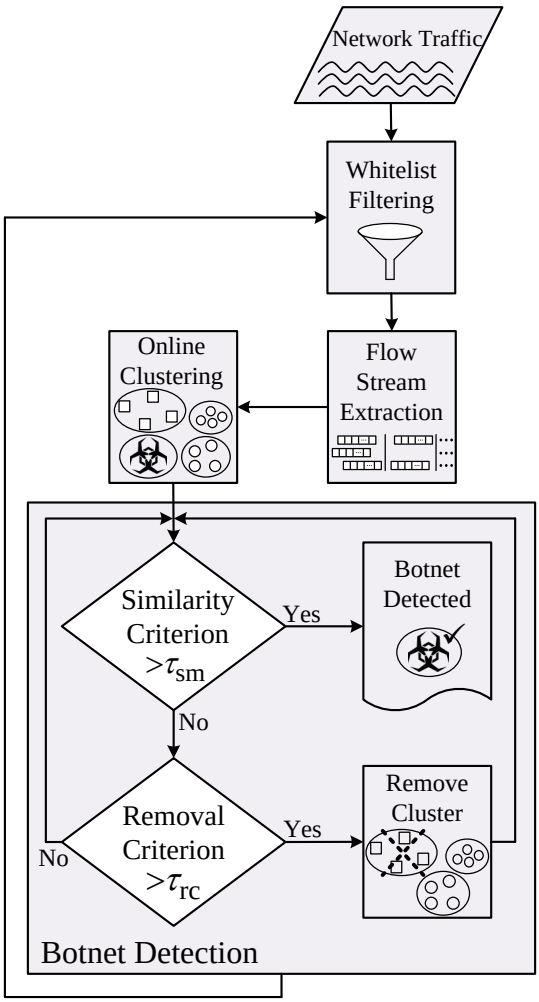



**Figure 4**. Steps of BotOnus

We first capture the network traffic and then apply whitelist filtering on it to filter out packets that are safe. Whitelist filtering relies upon defining a set of trusted servers (e.g., Google, Yahoo, and Facebook) as "safe" for a given network and eliminating packets sent to or received from these servers. The remaining traffic is delivered to the next step. It should be noted that whitelist filtering is not essential to the detection rate of BotOnus but can reduce the storage space of the captured traffic. Moreover, it can be effective in decreasing the false-alarm rate, since if multiple hosts in a monitored network send the same requests at the same time to a *safe* server, it is highly probable that they will be incorrectly considered as a coordinated group of bots. At the end of each time period, the traffic received from the previous step is processed and a set of flow feature vectors is extracted from it. Then, these flow feature vectors are divided into some flow clusters by the online fixed-width clustering (OFWC) algorithm. Finally, flow clusters that have at least two members and their intra-cluster similarity is above a similarity threshold $\tau_{sm}$, are marked as suspicious botnet clusters and all hosts that have at least a flow feature vector in these clusters are identified as bot infected. Also, flow clusters with a removal criterion above a remove threshold $\tau_{rc}$, are eliminated from the flow cluster set. Algorithm 1 shows the pseudo code of BotOnus.

**Algorithm 1** BotOnus

**Input:**
$\tau_{sm}$: similarity threshold
$\tau_{rc}$: remove threshold
$\sigma$: cluster width

1: **for** each time period $t$ **do**
2: Extract a set $S(t)$ of flow feature vectors from the network traffic
3: $C(t) \leftarrow ofwc(S(t), C(t-1), \sigma)$
4: **for** each flow cluster $c_j \in C(t)$ **do**
5: Calculate similarity criterion $sm(c_j)$
6: **if** $(|c_j| >= 2$ **and** $sm(c_j) > \tau_{sm})$ **then**
7: Identify all hosts that have at least a flow feature vector in $c_j$ as bot infected
8: **end if**
9: Calculate removal criterion $rc(c_j)$
10: **if** $rc(c_j) > \tau_{rc}$ **then**
11: $C(t) \leftarrow C(t) - \{c_j\}$
12: **end if**
13: **end for**
14: **end for**

### 4.3 Online Fixed-Width Clustering

BotOnus detects botnets through network-flow analysis of their C&C communication traffic. A flow is a set of packets that share the same source IP address, source port, destination IP address, destination port, and protocol. The flow is active as long as packets that meet its specification arrive continuously. Hence, statistical features of each active flow are continuously changed over time and it is impossible to cluster active flows unless we consider each of them as a flow stream over different time periods.

**Definition 2 (Flow feature vector).** Let $f_i$ be a flow. Each flow feature vector $f_i(t_j)$ is a $p$-dimensional feature vector extracted from $f_i$ during the time period $t_j$:

$$f_i(t_j) = (x_1, x_2, \ldots, x_p), \quad (1)$$

where $p$ is the number of features. $x_k$ is the $k^{\text{th}}$ feature value of $f_i(t_j)$ and can be denoted as $f_i^k(t_j)$.

**Definition 3 (Flow stream).** The flow stream of a flow $f_i$, denoted as $F_i$, is defined as a sequence of flow feature vectors extracted from $f_i$ during consecutive time periods of the same length:

$$F_i = \langle f_i(t_1), f_i(t_2), f_i(t_3), \ldots, f_i(t_j), \ldots \rangle, \quad (2)$$

where $f_i(t_j)$ is a flow feature vector.

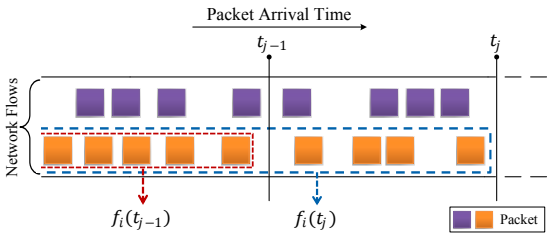



**Figure 5**. Flow feature vectors at two consecutive time periods

**Definition 4 (Flow set).** Let $F_i$ be the $i$th flow stream. The flow set $S$ is the infinite collection of flow streams in the network traffic:

$$S = \{\ldots, F_{i-1}, F_i, F_{i+1}, \ldots\}. \tag{3}$$

The set of flow feature vectors at a time period $t$ is denoted as $S(t) \subseteq S$:

$$S(t) = \{f_i(t), f_{i+1}(t), \ldots, f_{i+n}(t)\}. \tag{4}$$

As mentioned before, bots within a botnet receive the same commands and perform similar activities. Hence, they exhibit a similar traffic behavior during close time periods. This is largely due to the fact that bots are pre-programmed to perform the same routine activities as coordinated by the same botmaster [15]. Therefore, we can use a clustering technique to group bots in the same clusters.

We use a distance function to calculate the similarity between two flow feature vectors.

**Definition 5 (Flow distance).** Let $f_i(t)$ and $f_j(t)$ be two flow feature vectors at a time period $t$. The distance of $f_i(t)$ from $f_j(t)$ is defined as the average of differences between their corresponding features:

$$d(f_i(t), f_j(t)) = \frac{\sum_{k=1}^{p} \delta_{i,j}^k \Delta(f_i^k(t), f_j^k(t))}{\sum_{k=1}^{p} \delta_{i,j}^k}, \tag{5}$$

where $\delta_{i,j}^k$ is an indicator and is defined as

$$\delta_{i,j}^k = \begin{cases} 0 & \text{if } f_i^k(t) \text{ or } f_j^k(t) \text{ is missed,} \\ 1 & \text{otherwise.} \end{cases} \tag{6}$$

$\Delta(f_i^k(t), f_j^k(t))$ is the difference between the $k^{\text{th}}$ feature value of $f_i(t)$ and $f_j(t)$. If the $k^{\text{th}}$ feature is a categorical feature, the difference $\Delta(f_i^k(t), f_j^k(t))$ is calculated as

$$\Delta(f_i^k(t), f_j^k(t)) = \begin{cases} 0 & \text{if } f_i^k(t) = f_j^k(t), \\ 1 & \text{otherwise.} \end{cases} \tag{7}$$

Otherwise, it is calculated as

$$\Delta(f_i^k(t), f_j^k(t)) = \frac{|f_i^k(t) - f_j^k(t)|}{f_{max}^k(t-1) - f_{min}^k(t-1)}, \tag{8}$$

where $f_{min}^k(t-1)$ and $f_{max}^k(t-1)$ are the minimum and maximum of the $k^{\text{th}}$ feature values of flow feature vectors at the previous time period $t-1$, respectively. The main purpose of using them is to normalize feature values when we are calculating the distance of two flow feature vectors.

**Definition 6 (Flow cluster).** A flow cluster is a set of flow feature vectors such that the distance of each one from the cluster centroid is less than a fix width $\sigma$. Each flow cluster $c_j$ is defined as a pair $(\mu_j, \beta_j)$, where $\mu_j$ is the centroid and $\beta_j$ is the birth date of $c_j$.

The set of flow clusters at a time period $t$ is denoted as $C(t)$. For each flow cluster $c_j \in C(t)$, the centroid $\mu_j$ is represented as a $p$-dimensional feature vector. Let $c_j^k$ be the set of $k^{\text{th}}$ feature values of flow feature vectors in $c_j$:

$$c_j^k = \{f_i^k(t) | f_i(t) \in c_j\}. \tag{9}$$

Each feature value $\mu_j^k \in \mu_j$ is calculated as the average of $c_j^k$ if it is numerical, otherwise it is set to a value with the highest frequency in $c_j^k$.

Our proposed online fixed-width clustering (OFWC) is similar to fixed-width clustering (FWC), in which it forms a set of clusters of a fixed-width $\sigma$, but the main difference is that data samples arrive continuously over time. Let $S(t)$ be a set of flow feature vectors at the current time period $t$ and $C(t-1)$ be the set of flow clusters at the previous time period $t-1$. OFWC first receives $S(t)$ and $C(t-1)$ as input. It then updates flow feature vectors in each flow cluster $c_j \in C(t-1)$ with their new values in $S(t)$. If the flow distance of a flow feature vector $f_i(t) \in c_j$ from the centroid $\mu_j$ is less than $\sigma$, then $\mu_j$ is updated. Otherwise, $f_i(t)$ is removed from $c_j$ and added to the nearest flow cluster $c_l \in C(t-1)$ whose flow distance is less than $\sigma$. Algorithm 2 shows the pseudo code of OFWC.

### 4.4 Intra-cluster Similarity

OFWC partitions a set of botnet and non-botnet flow feature vectors into flow clusters such that the botnet flow feature vectors are in the same flow clusters. What makes the most difference between botnet flow clusters and other flow clusters is the closeness of flow feature vectors within them (See Figure 6).

The bots belonging to the same botnet are pre-

**Algorithm 2** OFWC

**Input:**
$S(t)$: set of flow feature vectors
$C(t-1)$: set of flow clusters
$\sigma$: cluster width
**Output:**
$C(t)$: set of flow clusters

**for** $k = 1 \rightarrow p$ **do**
  Calculate $f^k_{min}(t-1)$ and $f^k_{max}(t-1)$
**end for**
**for** each flow feature vector $f_i(t) \in S(t)$ **do**
  **if** $f_i(t-1) \in c_j$ for some $c_j \in C(t-1)$ **then**
    **if** $d(f_i(t), \mu_j) < \sigma$ **then**
      $c_j \leftarrow (c_j - \{f_i(t-1)\}) \cup f_i(t)$
      Update centroid $\mu_j$
    **else**
      $c_j \leftarrow c_j - \{f_i(t-1)\}$
      Update centroid $\mu_j$
    **end if**
  **end if**
  **if** $f_i(t) \notin c_j$ for all $c_j \in C(t-1)$ **then**
    Find the nearest flow cluster $c_l \in C(t-1)$ to $f_i(t)$
    **if** $d(f_i(t), \mu_l) < \sigma$ **then**
      $c_l \leftarrow c_l \cup \{f_i(t)\}$
      Update centroid $\mu_l$
    **else**
      Make a new flow cluster $c_k$ with centroid $\mu_k$
      $C(t-1) \leftarrow C(t-1) \cup \{c_k\}$
      $\mu_k \leftarrow \{f_i(t)\}$
      $\beta_k \leftarrow t$
    **end if**
  **end if**
**end for**
$C(t) \leftarrow C(t-1)$
**return** $C(t)$

programmed to respond to different commands from the botmaster. Hence, they behave almost deterministically on the given commands. This implies botnet flow feature vectors within the same flow cluster are very similar to each other and there is a high proximity between them. In order to detect botnet flow clusters, BotOnus uses the intra-cluster similarity criterion.

**Definition 7 (Intra-cluster similarity criterion).** Let $c_j$ be a flow cluster with the centroid $\mu_j$. Intra-cluster similarity criterion of $c_j$ is calculated as

$$sm(c_j) = e^{-\frac{\overline{d}_j}{1+o_j}}, \tag{10}$$

where $\overline{d}_j$ is the average distance of all flow feature vectors in $c_j$ from $\mu_j$ and called the average intra-cluster distance:

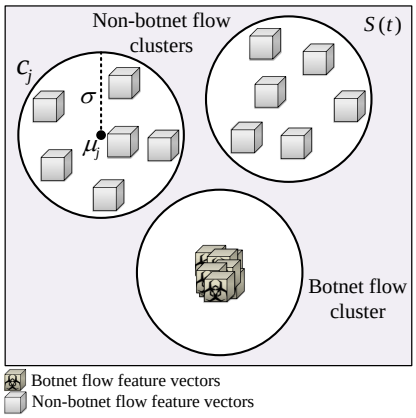



**Figure 6**. Botnet and non-botnet flow clusters

$$\overline{d}_j = \frac{1}{m}\sum_{i=1}^{m} d(f_i(t), \mu_j), \tag{11}$$

and $o_j$ is the lifetime of $c_j$:

$$o_j = t - \beta_j, \tag{12}$$

where $t$ and $\beta_j$ are the current time period and the birth date of $c_j$, respectively.

### 4.5 Cluster Removal

As long as new flow streams are arriving into the network over different time periods, the increase in the number of flow clusters appears to take place. This makes a significant increase in the amount of storage space and calculation time. To address this problem, at the end of each time period, BotOnus eliminates flow clusters that their removal criterion is above a remove threshold.

**Definition 8 (Cluster removal criterion).** Let $\overline{d}_j$ be the average intra-cluster distance and $o_j$ be the lifetime of the flow cluster $c_j$. The cluster removal criterion of $c_j$ is calculated as

$$rc(c_j) = \overline{d}_j \times o_j. \tag{13}$$

Since botnet flow streams are generated in a short period of time and there is a high similarity between them, it is not reasonable to maintain flow clusters with the low intra-cluster similarity for a long time. Hence, parameters $\overline{d}_j$ and $o_j$ in the cluster removal criterion cause flow clusters with the high intra-cluster distance and lifetime to be good candidates for elimination from the flow cluster set.

## 5 Experimental Results

To evaluate the performance of BotOnus, we used a testbed network consisting of five bot-infected hosts and connected it to the campus network (Figure 7). In addition to campus network traffic, we produced

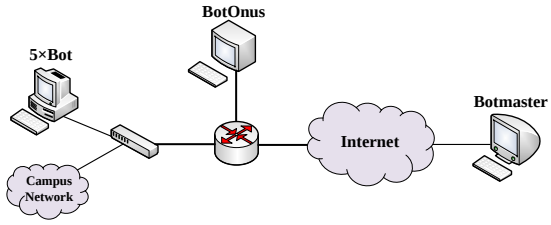



**Figure 7**. The testbed network used in the experiments

different network packets using a network traffic generator tool to ensure the existence of various types of network traffic. It was run throughout the experiments. The packets in our testbed network demonstrated wide diversity in popular protocols such as HTTP, FTP, SSH, DNS, and SNMP, and collaborative applications such as IM, P2P, and IRC. They were organized into bidirectional flow records by Argus [16]. It is a flow monitoring tool that inspects each packet and groups together those within the same connection into a flow record. BotOnus was run on a Linux server with an Intel Core 2 Quad 2.83 GHz CPU and 8 GB RAM. The system was run with average CPU and RAM utilization of 32% and 14%, respectively.

Since BotOnus is a four-step method, in the whitelist filtering step, we created a whitelist of the top 100 most popular websites (as reported by Alexa [17]) to filter out packets that are safe. However, as we discussed previously, this step is not essential for BotOnus and mainly aims to improve its efficiency. In the flow stream extraction step, at the end of each time period, we extracted a set of flow feature vectors from the network traffic. Each flow feature vector consisted of values of six different features: destination IP address, destination port, protocol, total bytes, total packets, and flow status. The value of the flow status can be one of three values: *initiated*, *ongoing*, or *completed*. The bots were launched during the fifth time period.

We used two measures of detection rate (DR) and false alarm rate (FAR) to evaluate the performance of BotOnus to detect various botnets including HTTP-, IRC-, and P2P-based botnets using the described testbed network:

$$DR = \frac{TP}{TP + FN}, \tag{14}$$

$$FAR = \frac{FP}{FP + TN}, \tag{15}$$

where $TP$ is the number of bot infected hosts that are correctly identified and $FN$ is the number of bot infected hosts incorrectly identified as uninfected. $FP$ is the number of uninfected hosts incorrectly identified as bot infected and $TN$ is the number of hosts that are correctly identified as uninfected.

**Table 1**. Parameter settings

| Parameter | Value |
|---|---|
| Length of time period | 5(s) |
| Cluster width ($\sigma$) | 0.5 |
| Similarity threshold ($\tau_{sm}$) | 0.95 |
| Remove threshold ($\tau_{rc}$) | 10 |
| Number of runs | 30 |
| Average number of hosts | 68 |
| Number of infected hosts | 5 |

**Table 2**. Average detection and false alarm rates of BotOnus for various botnets

| Botnet | Average Detection Rate | Average False Alarm Rate |
|---|---|---|
| HTTP-based | 0.95 | 0.041 |
| IRC-based | 0.96 | 0.033 |
| P2P-based | 0.91 | 0.037 |

We used TRiAD [18] as an HTTP-based botnet, RBot [19] as an IRC-based botnet, and Immonia [19] as a P2P-based botnet.

In our experiments, the length of time period was set to 5(s), the cluster width to $\sigma = 0.5$, and the similarity and remove thresholds to $\tau_{sm} = 0.95$ and $\tau_{rc} = 10$, respectively. We repeated each experiment 30 times and reported the average results. Table 1 shows the parameter settings.

Table 2 shows the average detection and false alarm rates of BotOnus for each type of botnets.

We performed experiments to analyze the effect of different settings of parameters on the performance of BotOnus. Figure 8 shows the effect of the parameter $\sigma$ on the average number of flow clusters during the online clustering step in the experiments for detecting various botnets. As shown, a decrease in $\sigma$ would lead to an increase in the number of flow clusters at each time period; thus, it incurs a significant additional amount of calculation time in subsequent steps.

As mentioned before, BotOnus is based on the intuition that since bots in the same botnet run the same program code, they have similar communication patterns and generate similar flow feature vectors. Therefore, there is a high proximity between flow feature vectors in each botnet flow cluster. Figure 9 demonstrates the intra-cluster distance of flow feature vectors of the IRC-based botnet.

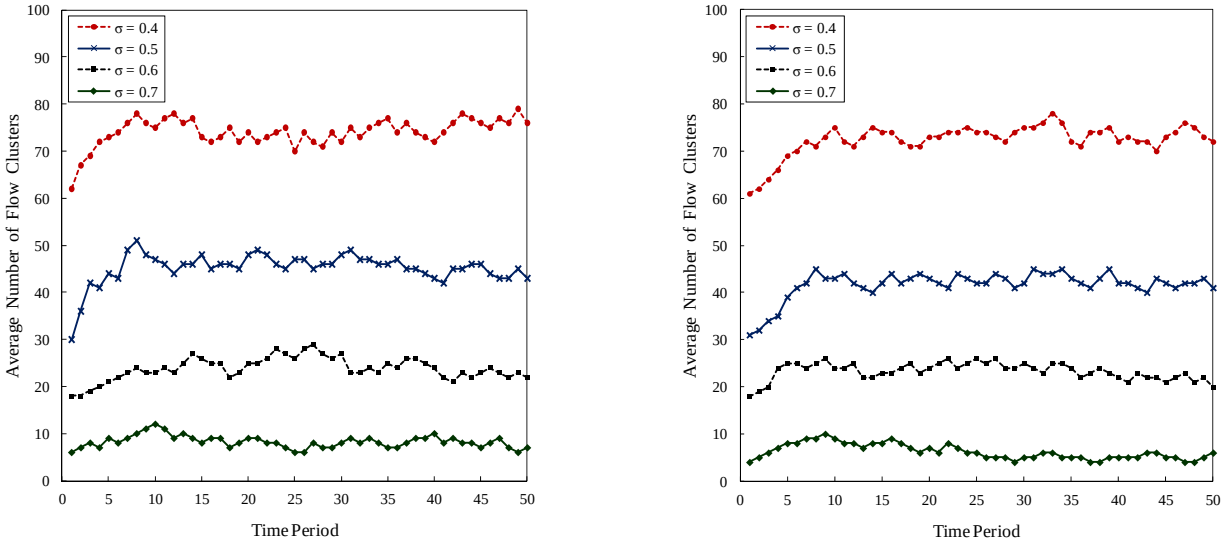



**Figure 8**. Effect of $\sigma$ on the average number of flow clusters in the experiments for detecting (a) IRC-based botnet, and (b) P2P-based botnet

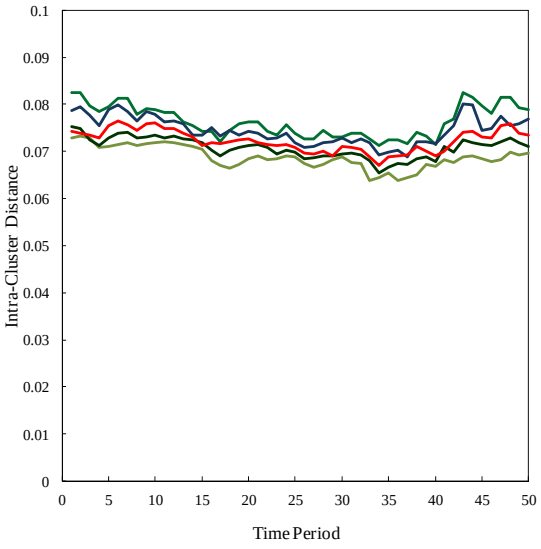
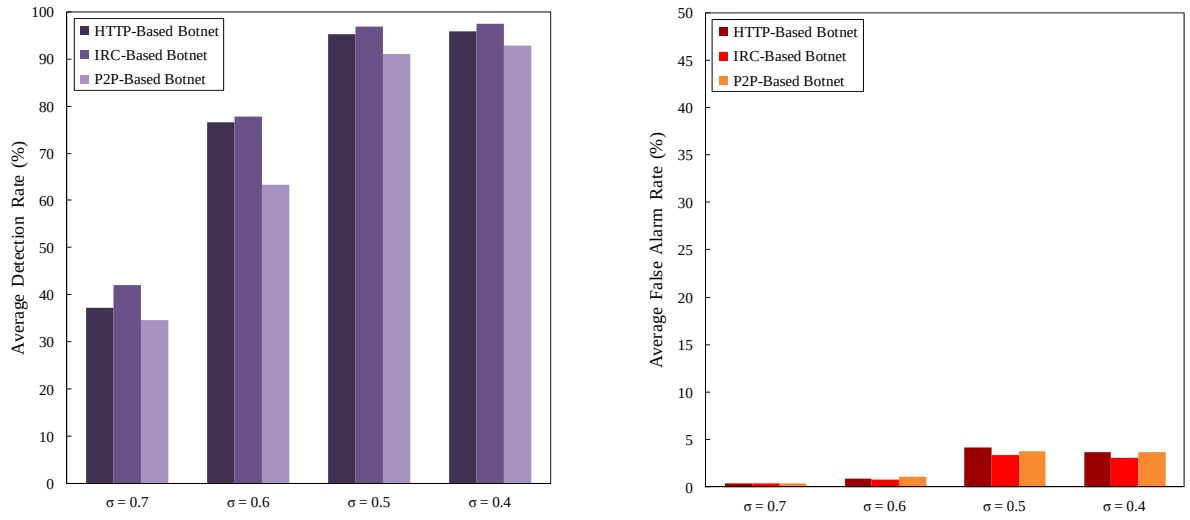



**Figure 9**. Intra-cluster distances of flow feature vectors of the IRC-based botnet

Figure 10 shows the average detection and false alarm rates of BotOnus for different values of the parameter $\sigma$, ranging from 0.7 to 0.4. As demonstrated before, there is a high proximity between flow feature vectors within a botnet flow cluster. Therefore, with an increase in $\sigma$, non-botnet flow feature vectors may be added to botnet flow clusters and reduce their intra-cluster similarity. As a result, this leads to a decrease in the detection and false alarm rates. According to Figure 8 and Figure 10, we can make a trade-off between the number of flow clusters, detection rate and false alarm rate by choosing $\sigma = 0.5$.

Figure 11 compares the average detection and false alarm rates of BotOnus for different values of the parameter $\tau_{sm}$, ranging from 0.8 to 0.98. As can be seen, with an increase in $\tau_{sm}$, the false alarm rate decreases, but the detection rate remains almost unchanged until $\tau_{sm} = 0.98$. This is because for a low value of $\tau_{sm}$, some non-botnet flow clusters may be incorrectly identified as botnet flow clusters. Also, by choosing $\tau_{sm}$ closely near to 1, the detection rate decreases, because flow feature vectors in a botnet flow cluster may not be exactly the same and thus it cannot satisfy the high value of $\tau_{sm}$.

Figure 12 shows the effect of different values of the parameter $\tau_{rc}$ on the average number of flow clusters during the online clustering step, where $\tau_{rc} = \infty$ shows BotOnus without cluster removal criterion. As the figure shows, maintaining old flow clusters with the high average intra-cluster distance leads to an increase in the number of flow clusters dramatically, and thus it incurs a significant additional amount of storage space and calculation time in subsequent steps. The selected value of $\tau_{rc}$ depends on the degree of synchronization in group activities of bots. In other words, if $\tau_{rc}$ is set to a low value and bots in the same botnet do not communicate with the botmaster at the relatively same time, a new formed botnet flow cluster may be removed before it includes flow feature vectors of all bots. On the other hand, selecting an appropriate value of $\tau_{rc}$ gives an opportunity to botnet flow clusters with insufficient flow feature vectors to remain in the flow cluster set and to increase their intra-cluster similarity during subsequent time periods.




(a) Detection Rate (b) False Alarm Rate

**Figure 10**. Performance of BotOnus for different values of $\sigma$

## 6 Conclusion and Discussion

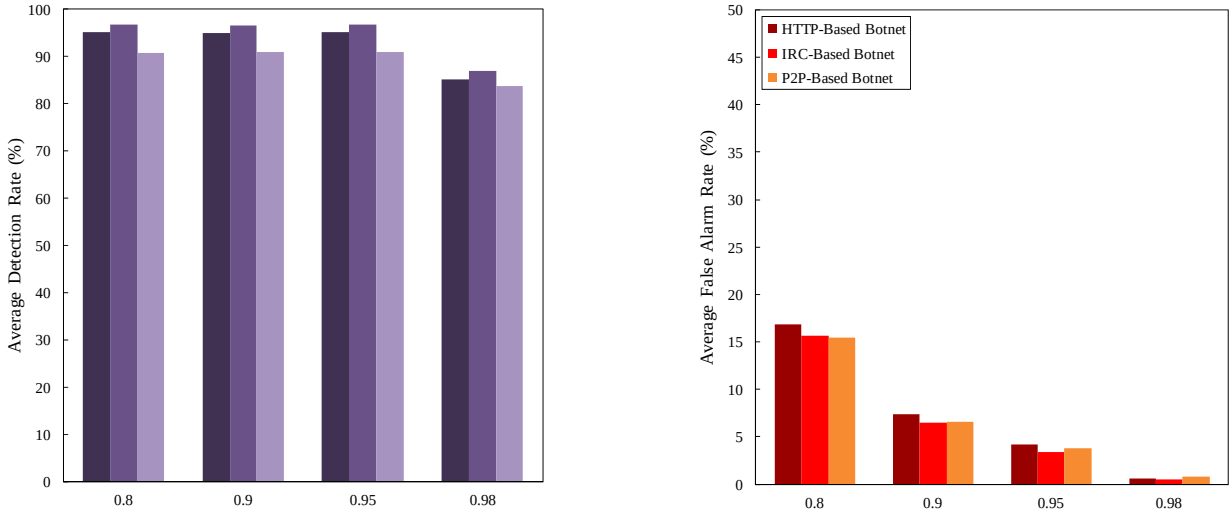
Botnet detection is a relatively novel and a very challenging research area. In the recent years, several botnet detection methods have been proposed, but most of them have some shortcomings, such as depending on a specific C&C protocol, lack of detection in an early stage of the lifecycle, working offline, and requiring labeled data for training. In this paper, we addressed these issues by proposing an unsupervised method, called BotOnus, that can be used for online botnet detection at the command and control stage. It is based on the fact that since bots in the same botnet are preprogrammed by the botmaster and run the same binaries, they are most likely to have the same communication pattern. The aim of BotOnus is to identify a group of bot-infected hosts within a monitored network that are parts of a same botnet.

We discussed different steps of BotOnus in detail, including whitelist filtering, flow stream extraction, online clustering, and botnet detection. At the end of each time period, BotOnus first extracts a set of flow feature vectors from the network traffic and then groups them to some flow clusters by the OFWC algorithm. Flow clusters that have at least two members and their intra-cluster similarity is above a similarity threshold, are identified as suspicious botnet clusters and all hosts in these clusters are identified as bot infected. It also chooses some flow clusters to be eliminated from the flow cluster set based on a cluster removal criterion. We evaluated the performance of BotOnus to detect various botnets including HTTP-, IRC-, and P2P-based botnets using a testbed network and investigated the impact of different parameters on its performance. The experiment results have shown that BotOnus can efficiently detect botnets with a high detection rate and an acceptable low false-alarm rate.

**Table 3**. Comparison of BotOnus with other methods

| Detection Method | Unknown Botnets | Low Dependency | Command Encrypted | Online Detection | Early Stage Detection |
|---|---|---|---|---|---|
| Livadas [7] | No | No | Yes | No | Yes |
| Rishi [8] | No | No | No | Yes | Yes |
| BotMiner [10] | Yes | Yes | Yes | No | No |
| Castle [11] | No | Yes | No | No | No |
| Lee [12] | No | No | Yes | Yes | Yes |
| Xiaocong [14] | Yes | No | Yes | Yes | Yes |
| BotSniffer [15] | No | No | No | Yes | No |
| BotProbe [20] | Yes | No | Yes | Yes | Yes |
| BotOnus | Yes | Yes | Yes | Yes | Yes |

It is not easy to conduct a fair comparison among various botnet detection methods due to differences between testbed networks, volume of traffic, bot binaries used in the experiments, and lack of common datasets. Therefore, instead of doing a performance comparison between BotOnus and other existing botnet detection methods, we compare them in terms of some significant characteristics. In Table 3, we give a general comparison between BotOnus and other well-




(a) Detection Rate (b) False Alarm Rate

**Figure 11**. Performance of BotOnus for different values of $\tau_{sm}$

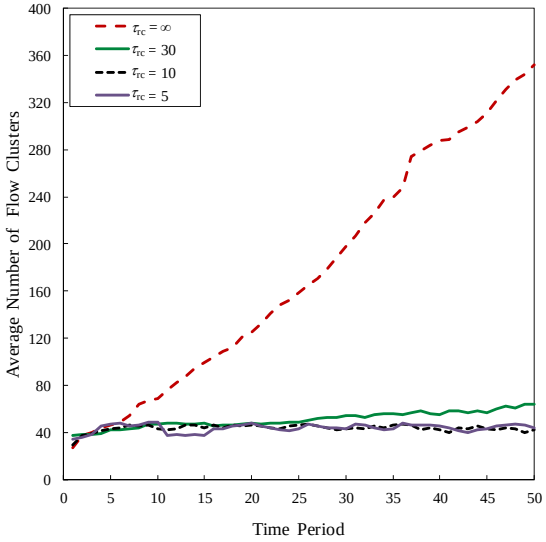



**Figure 12**. Effect of $\tau_{rc}$ on the average number of flow clusters

known botnet detection methods including Livadas [7], Rishi [8], BotMiner [10], Castle [11], Lee [12], Xiaocong [14], BotSniffer [15], and BotProbe [20] previously reported in the literature.

There are some techniques, such as encrypting the C&C traffic, by which the botmaster may attempt to evade detection. Hence, detection methods, which employ network packet payload analysis will be ineffective. To accommodate encryption, BotOnus uses flow feature vectors extracted from packet headers. It is also able to detect unknown botnets, since it uses an unsupervised technique driven by intrinsic characteristics of botnets such as group activities, without *a priori* knowledge of them.

We found that BotOnus has a low dependency on a specific botnet C&C protocol and can successfully detect botnets with a high detection rate and an acceptable low false alarm rate.

We are interested in further development of BotOnus toward a comprehensive tool for tracking botnet malicious activities, in conjunction with monitoring of botnet communications, to further reduce the false-alarm rate.

## Acknowledgment

This work was supported in part by the Iran Telecommunication Research Center (ITRC) under contract 90-01-04.

## References

[1] P. Wang, S. Sparks, and C. Zou, "An Advanced Hybrid Peer-to-Peer Botnet", *IEEE Transactions on Dependable and Secure Computing*, 7(2):113–127, 2010.

[2] Damballa Top 10 Botnet Threat Report – 2010, http://www.damballa.com/downloads/r_pubs/Damballa_2010_Top_10_Botnets_Report.pdf

[3] X. Li, H. Duan, W. Liu, and J. Wu, "Understanding the Construction Mechanism of Botnets", in *Proceedings of the* $6^{th}$ *International Conference on Ubiquitous Intelligence and Computing*, Brisbane, Australia, July 2009.

[4] M. Rajab, J. Zarfoss, F. Monrose, and A. Terzis, "A Multifaceted Approach to Understanding the Botnet Phenomenon", in *Proceedings of the* $6^{th}$ *ACM SIGCOMM Conference on Internet Mea-*

*surement*, Rio de Janeriro, Brazil, October 2006.

[5] B. Jansen,"Click Fraud", *Computer*, 40(7):85–86, 2007.

[6] W. Lu, G. Rammidi, and A. Ghorbani, "Clustering Botnet Communication Traffic Based on N-gram Feature Selection", *Computer Communications*, 34(3):502–514, 2011.

[7] C. Livadas, R. Walsh, D. Lapsley, and W. Strayer, "Using Machine Learning Techniques to Identify Botnet Traffic", in *Proceedings of the* $31^{st}$ *Annual IEEE Conference on Local Computer Networks*, Florida, USA, November 2006.

[8] J. Goebel and T. Holz, "Rishi: Identify Bot Contaminated Hosts by IRC Nickname Evaluation", in *Proceedings of* $1^{st}$ *Workshop on Hot Topics in Understanding Botnets*, Cambridge, MA, USA, April 2007.

[9] W. Wang, B. Fang, Z. Zhang, and C. Li, "A Novel Approach to Detect IRC-Based Botnets", in *Proceedings of the International Conference on Networks Security, Wireless Communications and Trusted Computing*, Wuhan, Hubei, China, April 2009.

[10] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection", in *Proceedings of the* $17^{th}$ *USENIX Security Symposium*, San Jose, CA, USA, July 2008.

[11] I. Castle and E. Buckley, "The Automatic Discovery, Identification and Measurement of Botnets", in *Proceedings of the* $2^{nd}$ *International Conference on Emerging Security Information, Systems and Technologies*, Cap Esterel, France, August 2008.

[12] J. Lee, H. Jeong, J. Park, M. Kim, and B. Noh, "The Activity Analysis of Malicious HTTP-Based Botnets Using Degree of Periodic Repeatability", in *Proceedings of the International Conference on Security Technology*, Sanya, Hainan Island, China, December 2008.

[13] H. Choi, H. Lee, and H. Kim, "BotGAD: Detecting Botnets by Capturing Group Activities in Network Traffic", in *Proceedings of the* $4^{th}$ *International ICST Conference on Communication System Software and Middleware*, Dublin, Ireland, June 2009.

[14] Y. Xiaocong, D. Xiaomei, Y. Ge, Q. Yuhai, and Y. Dejun, "Data-Adaptive Clustering Analysis for Online Botnet Detection", in *Proceedings of the* $3^{th}$ *IEEE International Joint Conference on Computational Science and Optimization*, Anhui, China, May 2010.

[15] G. Gu, J. Zhang, and W. Lee, "BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic", in *Proceedings of the* $15^{th}$ *Annual Network and Distributed System Security Symposium*, San Diego, CA, USA, February 2008.

[16] Argus—Auditing Network Activity, http://www.qosient.com/argus

[17] Alexa—The Web Information Company, http://www.alexa.com

[18] X1machine—Internet security and programming related blog, http://x1machine.blogspot.com

[19] Hack Forums, http://www.hackforums.net

[20] G. Gu, V. Yegneswaran, P. Porras, J. Stoll, and W. Lee, "Active Botnet Probing to Identify Obscure Command and Control Channels", in *Proceedings of the* $25^{th}$ *Annual Computer Security Applications Conference*, Honolulu, HI, USA, December 2009.

**Mosa Yahyazadeh** is an M.Sc. student of computer engineering at Tarbiat Modares University. His main research interests are network security, botnet detection and analysis, web security, intrusion detection, and anomaly detection. Currently, he works on his thesis on C&C protocol- and structure-independent botnet detection.

**Mahdi Abadi** received his B.Sc. and M.Sc. degrees in computer engineering from Ferdowsi University of Mashhad in 1998 and Tarbiat Modares University in 2001, respectively. He also received the Ph.D. degree from Tarbiat Modares University in 2008, where he worked on the network vulnerability analysis. Since 2009, he has been an assistant professor in the Faculty of Electrical and Computer Engineering at Tarbiat Modares University. His main research interests are network security, intrusion detection and prevention, malware detection and analysis, evolutionary algorithms, and data mining.