2014 7th International Symposium on Telecommunications (IST'2014)

BotCatch: Botnet Detection Based on Coordinated Group Activities of Compromised Hosts

Mosa Yahyazadeh and Mahdi Abadi
Faculty of Electrical and Computer Engineering
Tarbiat Modares University
Tehran, Iran
{m.yahyazadeh, abadi}@modares.ac.ir

Abstract—**Botnets have become one of the major tools used by attackers to perform various malicious activities on the Internet, such as launching distributed denial of service attacks, sending spam, leaking personal information, and so on. In this paper, we present BotCatch, a behavior-based botnet detection system that considers multiple coordinated group activities in the monitored network to identify bot-infected hosts. To achieve this goal, it first identifies suspicious hosts participating in coordinated group activities by an online incremental clustering algorithm and then calculates a negative score for each of the hosts based on several fuzzy membership functions. It then makes an informed decision and identifies a host as bot-infected if its negative score is higher than a threshold. We demonstrate the effectiveness of BotCatch to detect various botnets including HTTP-, IRC-, and P2P-based botnets using a testbed network consisting of some bot-infected hosts. The experimental results show that BotCatch can successfully detect various botnets with a high detection rate while keeping false alarm rate significantly low.**

Keywords—botnet detection; botnet lifecycle; coordinated group activity; online incremental clustering; fuzzy membership function

I. Introduction

A bot is a malware that runs on a host typically unbeknownst to its owner(s) and carries out commands sent by a so-called botmaster. As the matter of fact, bots are just malware that infect hosts to allow remote command and control (C&C) by the botmaster [1]. A botnet is defined as a network of bot-infected hosts. The number of botnets has increased dramatically in the past few years and they have become one of the biggest malware threats, responsible for a large volume of malicious activities. However, most of the Internet users are often not aware that their hosts have been compromised and become part of a botnet [2], [3]. This is because new generation botnets often use obfuscation techniques to evade typical antivirus scanners and thus we need to develop specific botnet detection techniques to counter the botnets.

Over the last few years, various techniques have been proposed to detect botnets. However, most of them have some limitations in terms of depending on a specific C&C protocol, lacking of detection in an early stage of the botnet lifecycle, working offline, and needing to labeled data for training. To solve some of these limitations, several techniques have been proposed that generally focus on identifying botnet coordinated group activities, but they still suffer from the lack of enough attention paid to the history of these activities. This makes them to potentially have a high false alarm rate.

In this paper, we propose BotCatch, a behavior-based botnet detection system that considers multiple coordinated group activities of compromised hosts in the monitored network. BotCatch leverages intrinsic behavior of bots in a same botnet, which is that they (1) receive same commands from the botmaster in the command and control stage and (2) perform some malicious activities in response to these commands in the attack stage. These facts result in some coordinated group activities in the botnet's lifecycle. To remember the history of these activities, BotCatch calculates a negative score for each host based on several fuzzy membership functions and update it when the host participates in new coordinated group activities. The host is then identified as bot-infected if its negative score exceeds a pre-specified threshold.

The rest of this paper is organized as follows: Section II reviews some related work. Section III introduces BotCatch and Section IV reports experimental results. Finally, Section V draws some conclusions.

II. Related Work

In this section, we discuss the different botnet detection techniques in related literature.

Gu *et al.* [4] introduce BotHunter, a passive network monitoring system that focuses on recognizing the infection and coordination dialog occurring during a successful malware infection. The system works by tracking the two-way communication flows between internal and external hosts to extract an evidence trail of data exchanges that match a state-based infection model. BotSniffer [5] is a network anomaly detection technique that aims to identify botnet C&C channels. It is based on the observation that bots within the same botnet will likely demonstrate very strong synchronization/correlation in their responses/activities. Hence, it employs several correlation and similarity analysis algorithms to identify hosts that show these behaviors in the network. BotMiner [6] is an unsupervised botnet detection framework that employs group-level analysis and is independent of botnet C&C protocol and structure. It clusters similar communication traffic and similar malicious traffic, and applies cross cluster correlation to identify hosts that share both similar communication patterns and similar malicious activities. Even though BotMiner considers a history of botnet coordinated group activities by aggregating network flows during a long time interval (e.g., during one day), which makes it more accurate, but it still cannot detect botnets in an early stage and does not work online.

Castle and Buckley [7] present a technique to detect botnets that are used for sending spam. They process the headers of email messages and generate a set of synthetic headers from them. The headers are then used to obtain a set of so-called suspected botnet clusters containing a large number of email messages. This technique is simple but presents some serious drawbacks. It cannot detect botnets in an early stage and is only able to detect botnets that are capable of sending spam.

Choi *et al.* [8] present BotGAD, an unsupervised technique for online botnet detection. 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 technique when it performs DNS queries at one stage of the botnet lifecycle and never performs them again. Moreover, BotGAD is only able to detect botnets that perform group activities in DNS traffic.

Lu *et al.* [9] propose BotCop, a system for online detection and classification of botnet communication traffic. They first use the C4.5 decision tree algorithm to classify the network traffic into different application communities and then use a hierarchical clustering algorithm to find the anomalous behaviors on a specific application community. However, in reality, it is difficult to identify all network traffic (e.g., traffic with encrypted payload) into known applications on the large-scale networks.

Yahyazadeh and Abadi [10] present BotOnus, an online botnet detection technique that uses a fixed-width clustering algorithm to identify coordinated group activities in the network traffic. Despite the fact that BotOnus has a high detection rate, its false alarm rate is roughly high due to the lack of paying attention to the history of the group activities.

III. BotCatch

In this section, we present BotCatch, a behavior-based botnet detection system. The aim of BotCatch is to detect bot-infected hosts within the monitored network that are part of the same botnet. BotCatch can be deployed at the edge of the network to capture and analyze the traffic between internal and external hosts. It consists of three main steps: flow feature vector extraction, group activity detection, and negative score calculation (See Fig. 1). In the following, we discuss each of these steps in detail.

A. Flow Feature Vector Extraction

In this step, we process the captured network traffic and extract a set of feature vectors from it. Recall that a flow is a set of packets that share the same source IP address, source port, destination IP address, destination port, and protocol. A flow feature vector is a p-dimensional feature vector extracted from a flow during a particular time period [10]. As previously mentioned, bots within the same botnet receive the same commands and perform the similar activities. This causes that they exhibit a similar traffic behavior during close time periods. Thus, we expect that bot-infected hosts have similar flow feature vectors.

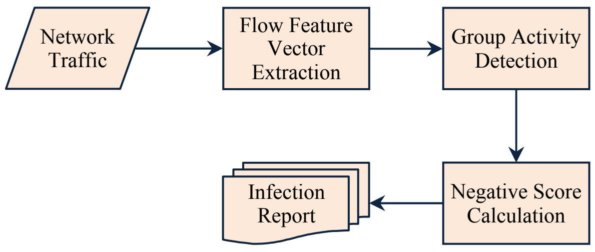


Fig. 1. Steps of BotCatch.

B. Group Activity Detection

In this step, we first divide the flow feature vectors into some flow clusters using the online fixed-width clustering (OFWC) algorithm [10]. The aim of OFWC is to form a set of fixed-width clusters such that feature vectors in the same flow cluster are more similar to each other than feature vectors in different flow clusters.

Next, we identify suspected flow clusters using the suspected cluster identification (SCI) algorithm. What makes the most difference between botnet flow clusters and other flow clusters is the closeness of feature vectors within them, indicating that there is a coordinated group activity in the network. Therefore, we examine the flow clusters and mark a flow cluster that has at least two members as suspected if it has been updated at the previous time period and its intra-cluster similarity criterion [10] is above a similarity threshold, τ_{sim}. Also, since there is a high similarity between flow feature vectors of bot-infected hosts, it is not reasonable to maintain flow clusters with a low intra-cluster similarity criterion for a long time. Therefore, flow clusters whose cluster preservation criterion [10] is below a preserve threshold, τ_{prv}, are eliminated from the set of flow clusters. Algorithm 1 shows the pseudo code of SCI.

Algorithm 1 SCI

```
input:
  $C(t-1)$: set of flow clusters
  $\tau_{sim}$: similarity threshold
  $\tau_{prv}$: preserve threshold
output:
  $\Omega(t)$: set of suspected flow clusters
  $C(t)$: set of flow clusters
begin
1:   $\Omega(t) \longleftarrow \emptyset$
2:   $C(t) \longleftarrow \emptyset$
3:   for each flow cluster $c_j \in C(t-1)$ do
4:       Calculate the intra-cluster similarity criterion $sim(c_j)$
5:       if $upd(c_j, t-1) = \mathrm{T}$ and $|c_j| \geq 2$ and $sim(c_j) > \tau_{sim}$ then
6:           $\Omega(t) \longleftarrow \Omega(t) \cup \{c_j\}$
7:       end if
8:       Calculate the cluster preservation criterion $prv(c_j)$
9:       if $prv(c_j) > \tau_{prv}$ then
10:          $C(t) \longleftarrow C(t) \cup \{c_j\}$
11:      end if
12:  end for
end
```

C. Negative Score Calculation

In this step, we use several fuzzy membership functions to calculate a negative score for each host participating in suspicious group activities. This allows us to make an informed decision about hosts which are most likely to be bot-infected. The fuzzy membership functions are built based on the concepts of observation and impression, as described below.

Definition 1 (Observation). The observation is a set of variables that reflect different aspects of a flow cluster.

Let O be the set of all observations and $\mathcal{H}$ be the set of all hosts in the network. We define two functions $h: O \to 2^{\mathcal{H}}$ and $d: O \to [0,1]$, such that for each observation $o \in O$ of a flow cluster c, $h(o)$ gives a set of hosts that have a feature vector in c and $d(o)$ gives the average intra-cluster distance of c. An observation causes that we record an impression for it.

Definition 2 (Impression). The impression is an evaluation made on certain aspects of an observation. The impression ι of an observation o is denoted as a 4-tuple:

$$\iota = (o, \tau, \eta, \nu) \ , \tag{1}$$

where τ is the time period when the impression is recorded, η is the number of different hosts in o, and ν is the closeness rating calculated through $(1 - d(o))$. In fact, η and ν are the parameters obtained from the observation that is judged.

For example, suppose a flow cluster c with 5 members and an average intra-cluster distance of 0.05 is observed during the time period 8. We record an impression $\iota = (o, 8, 5, 0.95)$ of this observation. From this impression, we conclude 5 hosts closely try to be included in a coordinated group activity.

An impression $\iota = (o, \tau, \eta, \nu)$ *covers* a host h_i iff $h_i \in h(o(\iota))$. We define $\Gamma(h_i, t)$ to be the set of all recorded impressions that cover h_i from the time period $t - m$ to t:

$$\Gamma(h_i, t) = \{\iota \mid h_i \in h(o(\iota)) \wedge (t - m) \le \tau(\iota) \le t\} \ , \tag{2}$$

where $h(o(\iota))$ is the set of hosts in $o(\iota)$ and m is a parameter specified by the user to determine the most recent impressions. If an impression has been recorded at a much earlier time period, it is considered out-of-date and not included in $\Gamma(h_i, t)$.

Definition 3 (Negative Score). The negative score is a score given to a host on a scale of 0 to 1 to show the level of its participation in coordinated group activities over different time periods. The score is calculated for each host based on the set of all recorded impressions that cover it.

When calculating the negative score of a host, it is important to know how reliable that value is. Although there are many factors and parameters that can be taken into account to increase the reliability of a negative score, we focus on two fuzzy membership functions: impression support and participant similarity.

1) Impression Support: The intuition behind this function is that in the monitored network, few observed cases of coordinated group activities recorded as different impressions are not enough to make a correct judgment about that a host is bot-infected. As the number of suspected clusters covering a host grows, the impression support of that host increases until the number reaches a maximum value, which we call the *confidence level*. Therefore, a host that has a history of coordinated group activities is more susceptible than the others.

Let $\Gamma(h_i, t)$ be the set of all impressions that cover the host h_i from the time period $t - m$ to t. The impression support of h_i at this time period, denoted as $S(h_i, t)$, is calculated as

$$S(h_i, t) = \begin{cases} \sin(\frac{\pi}{2\xi} |\Gamma(h_i, t)|) & |\Gamma(h_i, t)| \in [0, \xi] \ , \\ 1 & \text{otherwise} \ , \end{cases} \tag{3}$$

where $\xi < m$ is a parameter specified by the user to determine how many impressions should cover each host in order to reach the confidence level and $|\Gamma(h_i, t)|$ is the cardinality of $\Gamma(h_i, t)$. If the number of impressions which cover h_i is much less than the confidence level parameter ξ, the impression support of h_i will decrease.

2) Participant Similarity: The participant similarity is another factor that we can take into account to calculate the negative score of a host. If the host has continuous coordinated group activities with others, it should have the same participant hosts in different observations. Hence, the diversity of its participant hosts in the current observation and the previous ones should be very low. In other words, the greater the value of the participant diversity for a host, the greater the decrease in its negative score.

Let ι_k and ι_s be the kth and sth impressions in the set $\Gamma(h_i, t)$. The participant similarity of h_i at this time period, denoted as $D(h_i, t)$, is calculated as

$$D(h_i, t) = e^{-\delta(h_i, t)} \ , \tag{4}$$

where $\delta(h_i, t)$ is a participant diversity measure defined as

$$\delta(h_i, t) = \sum_{\iota_k \in \Gamma(h_i, t)} \sum_{\iota_s \in \Gamma(h_i, t)} \omega(\iota_k, \iota_s) \cdot |h(o(\iota_k)) - h(o(\iota_s))| \ , \tag{5}$$

where $\omega(\iota_k, \iota_s)$ is a weighting function that gives higher weights to impressions that are closer together:

$$\omega(\iota_k, \iota_s) = e^{-|\tau(\iota_k) - \tau(\iota_s)|/m} \ , \tag{6}$$

From (4), we conclude that if the sets of hosts in two observations contain h_i and the difference between the sets is very low, the participant similarity of h_i will increase.

After calculating the values of the fuzzy membership functions, we calculate the negative score of h_i at time period t as

$$\mathcal{N}(h_i, t) = w \cdot S(h_i, t) + (1 - w) \cdot D(h_i, t) \ . \tag{7}$$

IV. Experimental Results

We implemented a prototype of BotCatch to evaluate its performance on real-world network traces. In this section, we present a set of evaluations that explore the performance of different components of BotCatch.

To generate real-world network traces, we established a testbed network using a virtual network environment of several VMware [11] hosts, including Ubuntu Linux and Windows XP SP2. These hosts were intentionally infected with various botnets, including those controlled via HTTP, IRC, or P2P, in order to launch them in a controlled environment. In addition to collect botnet network traces, we also collected normal network traces from a campus network. The packets in our testbed network demonstrated a 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 bi-directional flow records by Argus [12] that is a flow monitoring tool that inspects each packet and groups together those within the same connection into a flow record. In the testbed network, we launched the botnets in three different experiments (see Table I).

TABLE I. CHARACTERISTICS OF NETWORK TRACES COLLECTED FROM THE EXPERIMENTS

Experiment Name	HTTP	IRC	P2P
Number of packets	41,258	36,197	39,741
Duration	160m	110m	120m
Number of hosts	83	94	77
Botnet type	HTTP-based	IRC-based	P2P-based

TABLE II. COMPARISON OF THE RESOURCE UTILIZATION BETWEEN BOTONUS AND BOTCATCH

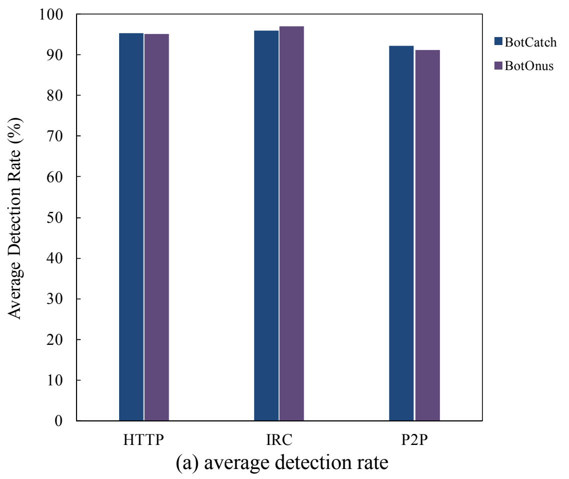
Technique	CPU Utilization (%)	RAM Utilization (%)
BotOnus	32	14
BotCatch	37	16

We used TRiAD [13] as a HTTP-based botnet, rBot [14] as an IRC-based botnet, and Immonia [14] as a P2P-based botnet. In the experiments, we commanded the bots to send basic information about their hosts, sleep for a while, and perform some malicious activities (e.g., launching denial of service attacks and port scanning). Hence, the bots received several commands during the experiments. We then collected the whole network traces (including those are generated by bot-infected and benign hosts) to evaluate the detection rate and false alarm rate of BotCatch. We extracted a set of feature vectors from the network traffic. Each feature vector consisted of values of eight different features: destination IP address, destination port, protocol, number of bytes per packet, number of bytes per second, number of packets per second, total bytes, and total packets. In our experiments, we ran BotCatch on a Linux server with an Intel Core 2 Quad 2.83 GHz CPU and 8 GB RAM, and compared it with BotOnus [10]. As shown in Table II, BotCatch was run with an average CPU and RAM utilization of 37% and 16%, respectively.

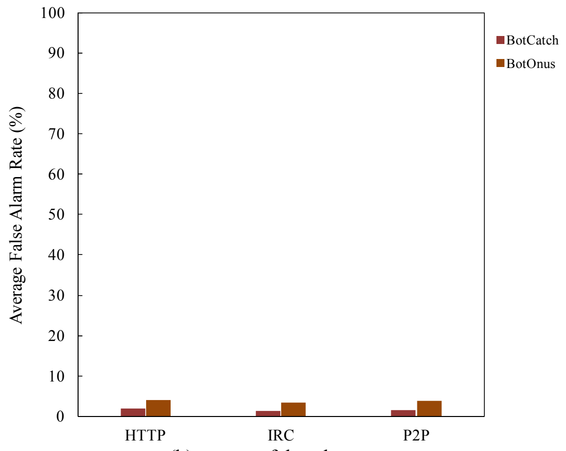
Fig. 2 compares the detection rate and false alarm rate of BotCatch with those of BotOnus [10]. Clearly, BotCatch has almost the same detection rate as BotOnus in various experiments, but much lower false alarm rate. This is because BotCatch considers the history of coordinated group activities occurred during intrinsic behavior of botnets.

It is not easy to conduct a fair comparison among various botnet detection techniques due to differences between testbed networks, volume of traffic, bot binaries used in experiments, and lack of common datasets. Therefore, instead of doing a performance comparison between BotCatch and other existing botnet detection techniques, we compare them in terms of some significant characteristics. In Table III, we give a general comparison between BotCatch and other well-known botnet detection techniques including BotSniffer [5], BotMiner [6], BotCop [9], and BotOnus [10], previously reported in the literature.

BotCatch would be able to detect unknown botnets, because it uses an unsupervised technique driven by intrinsic characteristics of botnets such as coordinated group activities, without *a priori* knowledge of them. It is also able to identify bot-infected hosts participating in some coordinated group activities in the early stages of the botnet lifecycle, even if they have not performed any malicious activities yet. In addition, it can detect various HTTP-, IRC-, and P2P-based botnets.



(a) average detection rate



(b) average false alarm rate

Fig. 2. Comparison between BotCatch and BotOnus [10].

TABLE III. COMPARISON OF BOTCATCH WITH OTHER TECHNIQUES

Botnet Detection Technique	Unknown Botnet Detection	Encrypted C&C Detection	Early Stage Detection	C&C Structure Independence
BotSniffer [5]	×	×	×	×
BotMiner [6]	✓	✓	×	✓
BotCop [9]	✓	×	✓	✓
BotOnus [10]	✓	✓	✓	✓
BotCatch	✓	✓	✓	✓

There are some techniques such as encrypting the C&C traffic, by which the botmaster may attempt to evade detection. Therefore, detection techniques that employ network packet payload analysis and rely on syntax are vulnerable to evasion [15]. BotCatch is robust to command encryption by extracting feature vectors from packet headers. It also uses an online incremental clustering to facilitate the identification of bot-infected hosts in real-time and considers the history of coordinated group activities in the network to ensure a low false alarm rate.

V. Conclusion

A botnet is a network of compromised hosts remotely controlled by a so-called botmaster. The number of botnets has increased dramatically over the past few years and they have become one of the biggest malware threats, responsible for a large volume of malicious activities. In recent years, various botnet detection techniques have been proposed, but most of them have some shortcomings in terms of depending on a specific C&C protocol, lacking of detection in an early stage of the botnet lifecycle, working offline, and needing to labeled data for training.

In this paper, we presented BotCatch, a behavior-based botnet detection system that considers multiple coordinated group activities in the network to identify bot-infected hosts. It is based on the fact that since bots in the same botnet are pre-programmed by the botmaster and run the same malicious code, they are most likely to have the same communication patterns. Therefore, this results in some coordinated group activities in the C&C stage that can be used for detecting bot-infected hosts. We discussed different components of BotCatch in detail, including flow feature vector extraction, group activity detection, and negative score calculation.

We evaluated the performance of BotCatch to detect various botnets including HTTP-, IRC-, and P2P-based botnets using a testbed network and compared with that of BotOnus. In the testbed network, we launched the botnets in three different experiments. The results of experiment showed that BotCatch can efficiently detect botnets with a high average detection rate of 94.45% and an acceptable average false alarm rate of 1.64%.

References

[1] P. Wang, S. Sparks, and C. C. Zou, “An advanced hybrid peer-to-peer botnet,” IEEE Transactions on Dependable and Secure Computing, vol. 7, no. 2, pp. 113–127, 2010.

[2] Damballa Top 10 Botnet Threat Report, http://www.damballa.com.

[3] M. Thomas and A. Mohaisen, “Kindred domains: Detecting and clustering botnet domains using DNS traffic,” in Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web Companion, Seoul, Korea, April 2014.

[4] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, “BotHunter: Detecting malware infection through IDS-driven dialog correlation,” in Proceedings of the 16th USENIX Security Symposium, Boston, MA, USA, August 2007.

[5] G. Gu, J. Zhang, and W. Lee, “BotSniffer: Detecting botnet command and control channels in network traffic,” in Proceedings of the 15th Annual Network and Distributed System Security Symposium, San Diego, CA, USA, February 2008.

[6] 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 17th USENIX Security Symposium, San Jose, CA, USA, July 2008.

[7] I. Castle and E. Buckley, “The automatic discovery, identification and measurement of botnets,” in Proceedings of the 2nd International Conference on Emerging Security Information, Systems and Technologies, Cap Esterel, France, August 2008.

[8] H. Choi, H. Lee, and H. Kim, “BotGAD: Detecting botnets by capturing group activities in network traffic,” in Proceedings of the 4th International ICST Conference on Communication System Software and Middleware, Dublin, Ireland, June 2009.

[9] W. Lu, M. Tavallaee, G. Rammidi, and A. A. Ghorbani, “BotCop: An online botnet traffic classifier,” in Proceedings of the 7th Annual Conference on Communication Networks and Services Research, Moncton, Canada, May 2009.

[10] M. Yahyazadeh and M. Abadi, “BotOnus: An online unsupervised method for botnet detection,” The ISC International Journal of Information Security, vol. 4, no. 1, pp. 51–62, 2012.

[11] VMware, Vmware Virtualization Software, http://www.vmware.com.

[12] Argus – Auditing Network Activity, http://www.qosient.com/argus.

[13] X1machine, Internet Security and Programming Related Blog, http://x1machine.blogspot.com.

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

[15] E. Stinson and J. C. Mitchell, “Characterizing bots’ remote control behavior,” in Proceedings of the 4th International Conference on Detection of Intrusions & Malware, and Vulnerability Assessment, Lucerne, Switzerland, July 2007.