TITLE

Victor Guilherme Turrisi da Costa
Department of Computer Science
of State University of Londrina
Londrina, Paraná 86057-970
Email: vt.turrisi@gmail.com

Bruno Bogaz Zarpelão
Department of Computer Science
of State University of Londrina
Londrina, Paraná 86057-970
Email: brunozarpelao@uel.br

***Abstract*—Botnets have been a serious threat to the Internet security. With the constant sophistication and resilience of them, a new trend has emerged, this being the shifting from the traditional desktop environment to the mobile environment. Like in the desktop domain, to minimize the threat imposed by mobile botnets detecting them is key. Along the diverse set of strategies applied to detect these botnets, the ones that show the best results, and generalized the best, involve discovering patterns in their anomalous behavior. On the mobile botnet field, one way to detect these anomalous pattern are by analysing the execution of these kind of applications. When comparing these kind of applications with normal ones, such as social media, music streaming and other, is possible to distinguish the behavior between the two. By applying an host-based and anomaly-based approach and utilizing machine learning algorithms this work aims to correctly find these behavior patterns in the task to detect botnets and minimize their actions. With a self-generated dataset containing multiple different types of mobile botnets and normal applications, we were able to test the performance of our approach in a very generalized and close-to-reality scenario.**

I. Introduction

Botnets have been a serious threat to the Internet security and one of the most challenging topics within the fields of computer and network security for many years [14].

In the recent years, a new trend among botnets has emerged with the grow of popularity and the capabilities of mobile devices, televisions, cars and house home appliances. These, having constant Internet access and connectivity to social networks and other applications, attracted attention from the owners of the botnets, shifting their attention from traditional desktop to these new devices [13].

As in the desktop domain, mobile botnets represent a large collections of devices, such as tablets or mobile phones, compromised with a sophisticated bot malware, putting them in control of an attacker. These devices work in conjunction to perform the fraudulent on-line activities desired by the botmasters, the ones that own and control a botnet [13].

Abotnet is formed by three main components, the bot, the botmaster and the Command and Control (C&C) infrastructure. The bot corresponds to the infected device that is under the control of a malicious user, or group of users. The botmaster is the malicious user that owns and controls all the bots. And the C&C infrastructure is the most important part of a botnet. It is used by the botmaster to send and receive information and commands to the bots [13].

To neutralize the threat imposed by the botnets an efficient approach to detect the bot malwares is needed. This approach has to have high detection rates, maintaining a low false positive rate while also minimizing the time needed to identify these malwares.

This work proposes an anomaly and host-based approach to detect mobile botnets utilizing machine learning algorithms. Our approach is based on identifying the anomaly in the execution of an application. While the main advantage of the host-based approach, is the better insight of the actions performed by the botnet in the host, including more than just the communication of the bot with the C&C infrastructure, whereas the a network-based approach would have more scalability and a lower computational cost.

To analyse the behavior of a computer program, or application, we can analyse the actions performed by it. In one of the lowest levels of abstraction, the actions performed by a program are the system calls it executes. We propose a system that works by extracting this system calls performed by the application and using them to correctly detect mobile botnets.

This work proposes, as contributions, the following:

1) To use an annomaly and host-based approach to detect mobile botnet applications against normal applications;
2) To generate a dataset containing known mobile botnets and non-botnets applications system call log files;
3) To use known machine learning algorithms to detect mobile botnets in a close-to-reality scenario;
4) To give a new insight of which features help the best in the task of identifying mobile bot malwares.

The rest of this paper is organized as follows. Section II presents an overview of the related works. Section III presents a general view of the system. This section is divided into subsection III-1, which shows the approach taken by the system, and subsection III-2, which comprehends to an insight on the features used to correctly detect mobile botnet application. The section IV is divided into subsections IV-1, IV-2 and IV-3. The first subsection comprehends to a description of the environment used for testing and the dataset generated. The second presents the results of the experiment, along with the metrics used to measure the performance of the classifier. In the last, we discussed the results achieved by our experiments. The conclusion and future work is presented in the last section, section V.

II. Related Work

So far, to our best knowledge, there are two main botnet domains, the classical desktop botnet and a new trend, the mobile botnets. There are two anomaly-based approaches to detect these, one being network-based and the other host-based, as our work is.

On the desktop botnets domain, Saad et al. [11] merged different honeypot-generated dataset, involving the Storm botnet and the Walowdac botnet, with a dataset of general traffic, from web browsing to Peer-to-Peer (P2P) gaming. In the task to detect P2P botnets, they have used flow-based features as well host-based features, focusing their attention to the network domain. Having above 90% of recall on many machine learning algorithms and 7% of false discovery rate (FDR) they achieved significant results but their work is limited by the only having two different types of botnets int their dataset. That would likely lead to not performing this well on a more botnet diverse environment. Our work tries to handle this by using 31 known mobile botnet applications across 13 different mobile botnet types.

Stevanovic et al. [14] proposed a flow-based botnet detection system utilizing features extracted from network flows. Using the dataset generated by [11], they compared multiple machine learning algorithms. Their best result being achieved by using a Random Forest, obtaining around 95% of precision and recall. While their results are very significant they are also limited by the low number of different botnets.

Sakib et al. [12] proposes a detection of HTTP-based botnets by using a unsupervised and semi-supervised approach using features extracted from the HTTP request URL and from DNS response. Applying unsupervised learning on a partial dataset from Clemson University campus network, they achieved a high recall, around 93%, but a very low precision, around 27%.

On the mobile botnet detection domain, many researches handle the detection of malwares, not focusing so much on botnets.

Burguera et al. [3] focused their research in detecting mobile malwares using a unsupervised machine learning approach and, as features, the counting of occurrences of each different system call. When testing their classifier, they used two real world applications with a very low sample size but achieved good detection rates, 92.5% in mean across both applications. The very low sample size is handled, in this work, by utilizing a diverse number of applications.

Karim et al. [8] executed mobile malware applications, botnet and non-botnet, on a sandbox and captured their behavior, but as himself says, some botnets identify the virtual environment and don't operate. By extracting features related to the malwares execution, the work achieves very high performance in the task of identifying botnet applications from other different malware applications. While this represent a very good result it lacks the normal mobile applications, included in our work, that likely lead to a more complicated environment.

While many works propose detecting botnets by utilizing machine learning, most of them are focused on desktop botnets. This work focus on correctly classifying mobile botnets and normal mobile applications. As the botnets are shifting to the mobile domain this is rapidly becoming a crescent problem. While trying to address this issue this paper proposes:

1) A host-based approach to identify mobile botnet applications against normal applications;
2) To generate a dataset containing known mobile botnets and non-botnets applications strace files;
3) To use known machine learning algorithms to detect mobile botnets in a close-to-reality scenario;
4) To give a new inside of which features help the best in the task of identifying botnets.

III. Botnet detection for mobile devices

Our solution to detect mobile botnets is host and anomaly-based. The botnet detection is done by dynamic analysing the applications and extracting features based on the system calls performed during an specific time window of it's execution.

The system is divided into three parts. The first being the monitoring and extraction module, that's responsible for collecting all the data that will be used later by the other modules. The second is the pre-processing module, responsible for extracting the information needed to generate the instances for the classifier. And the last module, the classifier, is responsible for building and training a model that will classify normal or mobile botnet activity. A top view of the system is represented in figure 1.

1) Approach: The first module works by constantly monitoring the mobile device and checking for new user owned processes, in these kind of devices, user owned processes represents all the applications that have not been granted root-access, or are not basic system processes. When it detects a new processes it automatically launches a Strace call on it. Strace is a diagnostic and debugging tool for Linux and with it is possible to monitor interactions between user space processes and the Linux kernel [4], collecting all system calls performed by the process. Redirecting the output of the Strace tool to a file, we can have a list, in chronological order, of the system calls performed by the process, which will later be used to created our features vector.

The second module of our system works by extracting the features needed for the classifier, these are generated using the output files of the Strace tool. To create the features vectors, we first group the system calls by time-window, each of these time-windows will become an instance for the classifier. Then, for each time-window, we count the number of occurrences of each different syscall. This counting process is also used by [3] during the task to detect malwares on Android devices. Unlike his paper, other features were also added to the vectors. Those are based on calls where the bytes in the operation would vary and have a meaning, for example, in a read or write call and also more generalising counting of system call occurrence.

Since the device is being monitored even during the installation of the applications, identifying these processes is a

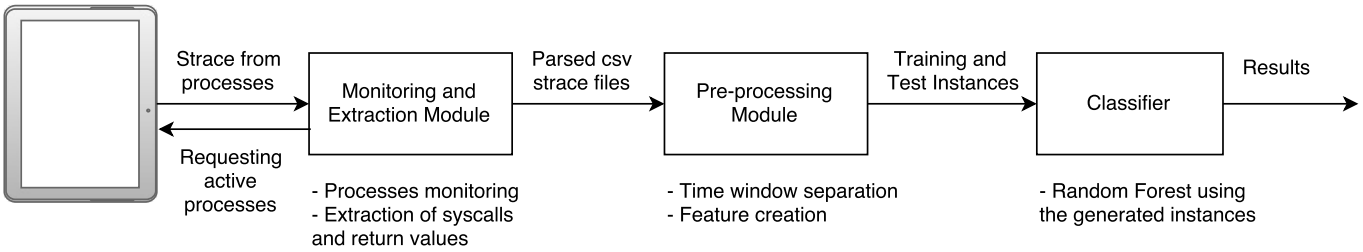

Fig. 1. Host-based mobile botnet detection system using Random Rorest.

trivial task, so, each process is manually classified as botnet or normal application.

The third part, the classifier, is a Random Forest fitted to the training data. Random Forest is a well-known machine learning algorithm. It works by combining tree predictors in a way each tree is fitted to the features and instances sampled independently, having the same distribution for all trees in the forest. Thus converging to an error limit as the number of trees grow [2]. This machine learning algorithm was chosen since it does not overfit to the data, act to reduce bias, has high accuracy and is relatively robust to outliers and noise [2]. When compared to two other machine learning algorithms, a SVM with RBF (Radial Based Function) kernel and one SVM with linear kernel, the Random Forest had the best and lowest variance, in different executions of the train-test process, metrics of the three, as seen in table V, which will be discussed on the later sections of this work.

2) Features: The features created for each instance are directly derived from the system calls performed during the time-window.

In this work, we proposed two types of features, the first being related to the occurrence of a syscall, while the other being related to the bytes involved in the call. The system calls fit, mainly, to: writing and reading calls, calls involving network operations, acquire and release lock operations and information requests by the process.

The features related to the occurrence of a syscall, correspond to:

1) The count of the occurrences of each different syscall;
2) The count of how many different calls did the process executed during the time-window;
3) The total number of calls during the period.

The other features are directly linked to the return values of the system calls, since these represent the number of bytes involved in the operation.

The operations where we identified the return values were the count of the total bytes involved in it can be seen on table I

From these system calls we derived the following byte-based features:

1) The total count of bytes in each different syscall;
2) The average of bytes of the calls during the time-window;
3) The standard deviation of bytes by syscall;
4) The total bytes along write-type system calls;
5) The total bytes along read-type system calls;
6) A write-read ratio, where the values closer to 0 represents more bytes involved in read operations, very large numbers represent the inverse and values close to 1 represents balance;
7) A send-receive ratio, that works just like the write-read ratio.

TABLE I
SYSTEM CALLS INVOLVING BYTES

Type	System Call	Action
Write	write	Write to a file descriptor
	writev	Writes to multiple file descriptors
	pwrite	Same as write, but with an offset
Read	read	Read from a file descriptor
	pread	Same as read, but with an offset
Network Operations	sendto	Send a message on a socket
	sendmsg	Same as sendto
	recvfrom	Receive a message from a socket
Directory Operations	geetdents64	Get directory entries

These were based on [6] [11] [14] byte-related features, which we extrapolate from the network-based domain to fit the host-based approach of this paper. The importance of these features will be analysed later in the discussion section.

Since the Random Forest works by selecting the features that best help to classify the instances [2] there were no worries that these byte-related features would impact the detection rates negatively. Table VII in the discussion section represents the top 15 features and as we can see 6 of them were extra to the simple occurrence of a system call, having 4 byte-related ones and 2 occurrence-related features.

IV. EVALUATION

1) Test Environment and Dataset: To generate the dataset for evaluation, we rooted a Samsung's Galaxy tablet with Android 4.1.2 and installed the Strace tool on it. This device was connected via usb to a notebook running Windows 10 and also connected via wi-fi to a network being hosted by the computer, which had Internet access.

The botnet applications were provided by the ICSX Android Botnet dataset [1], containing 14 different families of botnets. We selected 31 botnet applications, divided into 13 different families, as shown in table II.

Some legit applications were installed on the device. These include video and music streaming services, games and the

normal factory default applications, for example, e-mail, browser, calculator and note-keeper.

TABLE II
APPLICATIONS PER BOTNET FAMILY

Botnet Family	Number of Applcations
Anserverbot	4
BMaster	2
DroidDream	3
Geimini	5
MisoSMS	3
NickiSpy	2
NotCompatible	1
PJapps	2
Pletor	1
Rootsmart	1
Sandroid	4
Tigerbot	1
Zitmo	2

Using the monitoring module, we collected data from the device while running only normal applications. Then, we introduced each of the 31 botnet applications one by one, having one at a time in the device. After this, we sampled from the botnet applications and installed these in pairs and trios. In all parts of the dataset generating process we had times where we interacted with the applications and standby times.

Using the Strace files generated by the monitoring module, the pre-processing part starts. First, each file is parsed and it's system calls and return values (since in some syscalls they are the number of bytes involved in the call) are extracted. After that, it groups syscalls by time-windows. We chose a time window of 500ms, presented in table V, which will be discussed later. Then, we created a features vector for each time window, by extracting information from the parsed files.

All the modules of the system were developed in the Python programming language, and the classifier is build with scikit-sklearn's Random Forest Classifier, which is a Python library with build-in machine learning related algorithms [10].

The resulting dataset had roughly 85% of it as normal process instances and 15% as botnet instances and it's full format can be seen in table III.

2) Evaluation Metrics and Results: After our experiments with multiple different configurations, we achieved a good configuration for the Random Forest considering the trade-off in the multiple performance measures. The final configuration of the Random Forest is seen in table IV.

The metrics used to measure the performance of our classifier were:

I. $precision = \frac{TP}{TP+FP}$
II. $recall = \frac{TP}{TP+NF}$
III. $AUC = \frac{1}{2} * (recall + specificity)$
IV. $specificity = \frac{TN}{TN+FP}$
V. $FPR = 1 - specificity$

Where TP, TN, FP, FN represents the true positives, true negatives, false positives and false negatives, respectively.

Many papers that handle anomaly-based botnet detection using machine learning don't provide a good enough number of metrics to detail their classifier performance and, with these metrics used here, we can cover both sides of a good botnet classifier, it's ability to have low percentage of miss-classification.

TABLE III
DATASET DIVISION FORMAT

	Number of Instances
Total	28095
Normal Instances	23810
Botnet Instances	4285
Anserverbot	495
BMaster	1058
DroidDream	132
Geimini	923
MisoSMS	168
NickiSpy	83
NotCompatible	75
PJapps	308
Pletor	97
Rootsmart	27
Sandroid	509
Tigerbot	345
Zitmo	65

TABLE IV
RANDOM FOREST CONFIGURATION

Number of trees	Split criterion	Max Depth	Threshold
1000	Entropy	None	0.3

For the experiment we used a hold-out approach where 60% of the dataset was used for the training process and the rest, 40%, was used for testing. To minimize selecting the best, or worst, sub-datasets we iterated the experiment 50 times sampling from the dataset new train and test datasets in each iteration.

The results of our experiment are presented in table V, where we compared three popular machine learning algorithms varying the size time-window observed.

Another evaluator to compare the performance of multiple machine learning algorithms is the Receiver Operating Characteristic (ROC) curve. With it is possible to visualize the true positive rate (TPR) and the false positive rate (FPR) while varying the threshold for the algorithms. The ROC curve, figure 3, obtained in this work shows the Random Forest outperformed both SVMs.

3) Discussion: The results presented in table V show that the Random Forest classifier is able to have great performance metrics across different metrics.

Figure 2, shows the box plots of the performance metrics across 50 executions the Random Forest. The boxes are compressed, indicating a very low variance across the metrics.

The ROC curve, in figure 3, shows that the Random Forest outperformed the SVMs. This shows that the mobile botnet detection problem is non-linear and also complex.

Identifying a botnet app in a group of apps that vary from games to sms applications is not a trivial task, so a robust and complex machine learning algorithm performs best, as the Random Forest performance was the best by far.

TABLE V
MACHINE LEARNING ALGORITHM PERFORMANCE COMPARISON VARYING THE TIME-WINDOW

Time-window	ML Algorithm	Precision	Recall	AUC	Specificity	FPR
500ms	**Random Forest**	**0.866**	**0.883**	**0.929**	**0.975**	**0.024**
	SVM RBF	0.625	0.800	0.875	0.914	0.085
	SVM Linear	0.749	0.504	0.737	0.970	0.029
1s	**Random Forest**	**0.867**	**0.882**	**0.928**	**0.975**	**0.024**
	SVM RBF	0.627	0.802	0.858	0.914	0.085
	SVM Linear	0.754	0.536	0.752	0.969	0.030
5s	Random Forest	0.876	0.837	0.913	0.989	0.010
	SVM RBF	0.583	0.738	0.844	0.951	0.048
	SVM Linear	0.805	0.300	0.646	0.992	0.007
10s	Random Forest	0.869	0.812	0.902	0.991	0.008
	SVM RBF	0.541	0.706	0.833	0.960	0.039
	SVM Linear	0.857	0.284	0.640	0.996	0.003

Analysing table V we identified that the best performing time-window to analyse the process ranges from 500ms to 1 second, with the first having a slightly higher recall and the last a better precision.

This very small time-window required for a classification of a process shows that creating a system to analysed processed in real-time is a viable solution.

Since we were dealing with multiple botnet applications with different behaviors, we tried to identify which were the ones with lowest detection rate. In table VI we presented the detection rates by mobile botnet family.

TABLE VI
DETECTION RATE BY BOTNET TYPE

	Detection Rate %
Anserverbot	83.4
BMaster	97.2
DroidDream	84.9
Geimini	83.6
MisoSMS	58.0
NickiSpy	55.8
NotCompatible	99.5
PJapps	97.2
Pletor	73.4
Rootsmart	76.7
Sandroid	93.8
Tigerbot	100
Zitmo	28.4

MisoSMS had a 58% detection rate. One possible reason for this detection rate is that this kind of botnets work by sending SMSs and since our device couldn't do this kind of operation the botnet could not send any information [5].

NickiSpy tries to make the user install third-party malwares, since we didn't installed this third-party apps, the botnet couldn't complete it's function. And it reflected on it's low detection rate, 55.8%. Another action this kind of botmalwares performs is to send information from the device via SMS, which it couldn't perform, since the device had no SIM card [7].

Zitmo, the mobile equivalent of the Zeus botnet, is another botnet malware that works with SMSs. It tries to steal mobile transaction authorization numbers (mTAN), a code sent by banks to authorize transactions [9]. Since no bank operations were made, and device also didn't have any ways to send or receive SMSs, this botnet didn't perform any significant actions, directly impacting our detection rates.

One of our expected contributions with this work was to create new features that helps characterizing a botnet application, as presented in table VII we achieved this having 6 out of the 15 top features were proposed by us.

TABLE VII
TOP 15 FEATURES

Feature	Score
total_syscalls**	0.0618
clock_gettime	0.0566
gettimeofday	0.0557
ioctl	0.0530
read_bytes*	0.0484
read	0.0467
total_read*	0.0454
epoll_wait	0.0404
getpid	0.0367
getuid32	0.0366
futex	0.0323
write_read_ratio*	0.0299
different_syscalls**	0.0298
read_bytes_avg*	0.0291
gettid	0.0273

* Byte-based features;
** Call count based features extra to the counting by system call type.

From our system call occurrence-related features the total number of syscalls performed by the process during the time-window ranked first, while the total number of different system calls ranked 13th.

On the byte-related features, the ones related to the read system call ranked on the top 15, since bots may read a lot of information from the device. The ratio between write and read bytes is also interesting since it defines the proportion of write bytes and read bytes is similar across the multiple botnet families tested in this work.

V. CONCLUSION

In this work we presented a system to detect mobile botnets from normal mobile applications using an annomaly and host-based approach. By using the system calls performed by the mobile applications during a 500ms time-window and a

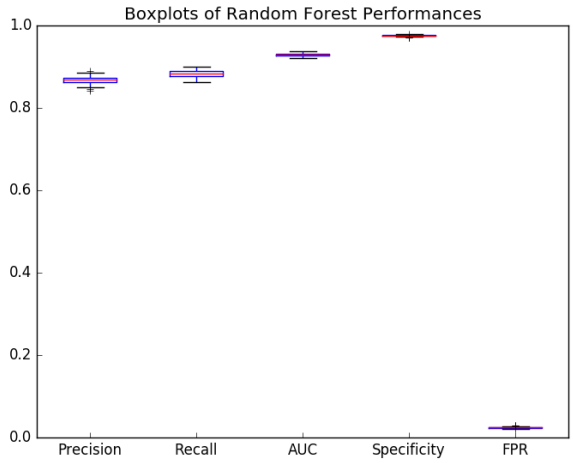

Fig. 2. Random Forest Performance.

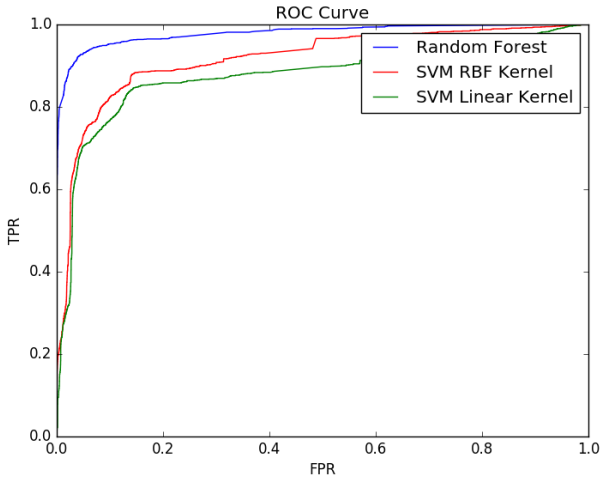

Fig. 3. ROC Curve.

Random Forest Classifier the system was capable of achieving high scores across different metrics. We also presented an insight of the most important features for the classifier and an insight of the mobile botnets that impacted the classifiers performance.

In the future, we would like to extend this work including new mobile botnets and a more diverse spectrum of mobile applications. The future work will be devoted to applying more modern machine learning techniques to increase the performance of the classifier and making the system perform real-time classifications.

REFERENCES

[1] Andi Fitriah Abdul Kadir, Natalia Stakhanova, and Ali Akbar Ghorbani. *Android Botnets: What URLs are Telling Us*, pages 78–91. Springer International Publishing, Cham, 2015.

[2] Leo Breiman. Random Forests. *Machine learning*, 45.1:5–32, 2001.

[3] Iker Burguera, Urko Zurutuza, and Simin Nadjm-Tehrani. Crowdroid: Behavior-Based Malware Detection System for Android. *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices - SPSM '11*, page 15, 2011.

[4] O. S. Community. strace. *http://sourceforge.net/projects/strace/*, 2009.

[5] FireEye. Misosms: New android malware disguises itself as a settings app, steals sms messages, 12 2013. (Accessed on 08/19/2016).

[6] Shree Garg, Ankush K. Singh, Anil K. Sarje, and Sateesh K. Peddoju. Behaviour analysis of machine learning algorithms for detecting P2P botnets. *2013 15th International Conference on Advanced Computing Technologies (ICACT)*, pages 1–4, 2013.

[7] Josh Grunzweig. Nickispy.c - android malware analysis & demo, 10 2011. (Accessed on 08/19/2016).

[8] Ahmad Karim, Rosli Salleh, and Muhammad Khurram Khan. SMARTbot: A behavioral analysis framework augmented with machine learning to identify mobile botnet applications. *PLoS ONE*, 11(3):e0150077, 2016.

[9] Kaspersky Lab. Teamwork: How the zitmo trojan bypasses online banking security, 10 2011. (Accessed on 08/19/2016).

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[11] Sherif Saad, Issa Traore, Ali Ghorbani, Bassam Sayed, David Zhao, Wei Lu, John Felix, and Payman Hakimian. Detecting P2P botnets through network behavior analysis and machine learning. *2011 9th Annual International Conference on Privacy, Security and Trust, PST 2011*, pages 174–180, 2011.

[12] Muhammad N Sakib and Chin-tser Huang. Using Anomaly Detection Based Techniques to Detect HTTP-based Botnet C & C Traffic. 2016.

[13] Sérgio S C Silva, Rodrigo M P Silva, Raquel C G Pinto, and Ronaldo M. Salles. Botnets: A survey. *Computer Networks*, 57(2):378–403, 2013.

[14] Matija Stevanovic and Jens Myrup Pedersen. An efficient flow-based botnet detection using supervised machine learning. *2014 International Conference on Computing, Networking and Communications (ICNC)*, pages 797–801, 2014.