Mobile botnets detection based on machine learning over system calls

Victor G. Turrisi da Costa* and
Sylvio Barbon Junior

Computer Science Department,
Londrina State University,
Londrina, PR, Brazil
Email: victorturrisi@uel.br
Email: barbon@uel.br
*Corresponding author

Rodrigo S. Miani

School of Computer Science,
Federal University of Uberlândia,
Uberlândia, MG, Brazil
Email: miani@ufu.br

Joel J.P.C. Rodrigues

National Institute of Telecommunications (Inatel),
Santa Rita do Sapucaí, MG, Brazil
and
Instituto de Telecomunicações,
Lisboa, Portugal
Email: joeljr@ieee.org

Bruno Bogaz Zarpelão

Computer Science Department,
Londrina State University,
Londrina, PR, Brazil
Email: brunozarpelao@uel.br

Abstract: Mobile botnets are a growing threat to the internet security field. These botnets target less secure devices with lower computational power, while sometimes taking advantage of features specific to them, e.g., SMS messages. We propose a host-based approach using machine learning techniques to detect mobile botnets with features derived from system calls. Patterns created tend to be shared among applications with similar actions. Therefore, different botnets are likely to share similar system call patterns. To measure the effectiveness of our approach, a dataset containing multiple botnets and legitimate applications was created. We carried out three experiments, namely finding out the best time-window, and performing feature selection and hyperparameter tuning. A high performance (over 84%) was achieved in multiple metrics across multiple machine learning algorithms. An in-depth analysis of the features is also presented to help future work with a solid discussion about system call-based features.

Keywords: mobile botnet detection; feature selection; host-based approach.

Reference to this paper should be made as follows: Turrisi da Costa, V.G., Barbon Jr., S., Miani, R.S., Rodrigues, J.J.P.C. and Zarpelão, B.B. (2019) 'Mobile botnets detection based on machine learning over system calls', *Int. J. Security and Networks*, Vol. 14, No. 2, pp.103–118.

Biographical notes: Victor G. Turrisi da Costa is an MSc student in the Computer Science Department at Londrina State University (UEL), Brazil. In 2017, he received his BSc in Computer Science at the same university. His research interests include deep learning, data stream mining, network security and computer vision.

Sylvio Barbon Junior is an Associate Professor and leader of the research group that studies machine learning in the Computer Science Department at State University of Londrina (UEL), Brazil. He received his Master degree in Computational Physics from University of São Paulo

(2007), degree in Computational Engineering during 2008 and PhD (2011) from IFSC/USP such as the Master degree. During 2017, he was a Visiting Researcher at the University of Modena and Reggio Emilia (Italy) and Università Degli Studi Di Milano (Italy). He is currently a Professor in postgraduate and graduate programs. His research interests include digital signal processing, pattern recognition and machine learning.

Rodrigo Sanches Miani received his BS in Mathematics from the Federal University of São Carlos, Brazil, his MSc and PhD in Electrical Engineering from the University of Campinas, Brazil. In 2011, he stayed six months as a Visiting PhD student under the supervision of Prof. Michel Cukier at the Cybersecurity Quantification Lab (CyQL), University of Maryland, USA. Since 2013, he is a Professor at the School of Computer Science of the Federal University of Uberlandia. His research interests are related to cybersecurity issues such as intrusion detection systems, security analytics and human aspects of cyber attacks.

Joel J.P.C. Rodrigues is a Professor at the National Institute of Telecommunications (Inatel), Brazil and a Senior Researcher at IT, Portugal. He is the leader of the Internet of Things Research Group (CNPq), Director for Conference Development – IEEE ComSoc Board of Governors, IEEE Distinguished Lecturer, Technical Activities Committee Chair – IEEE ComSoc Latin America Region Board, the Past-Chair of the IEEE ComSoc TCs on eHealth and on Communications Software. He is the Editor-in-Chief of the *IJEHMC*. He has authored or co-authored over 700 papers in refereed international journals and conferences, three books, and two patents.

Bruno Bogaz Zarpelão received, in 2004, his BS in Computer Science from State University of Londrina, Brazil, and, in 2010, his PhD in Electrical Engineering from University of Campinas, Brazil. In 2018, he carried out research at post-doctoral level at city, University of London, UK. He is currently an Assistant Professor at the Computer Science Department of the State University of Londrina (UEL), Brazil, which he joined in 2012. His research interests include security analytics, intrusion detection, and internet of things.

This paper is a revised and expanded version of a paper entitled 'Detecting mobile botnets through machine learning and system calls analysis' presented at The Proceedings of the 2017 IEEE International Conference on Communications (ICC), Paris, France, May 2017.

1 Introduction

Botnets are responsible for a number of complex and coordinated attacks, such as, click-fraud, distributed denial of service (DDoS) attacks, spam generation, distribution of multiple types of malware and sensitive information stealing (Stevanovic and Pedersen, 2014; Saad et al., 2011; Alparslan et al., 2012; Silva et al., 2013; Mahmoud et al., 2015). One of the most recent botnets attacks was conducted by the internet of things (IoT) botnet dubbed as Mirai, which performed a high-profile DDoS attack that showed unprecedented traffic volumes (Elzen and Heugten, 2017).

Many traditional devices, e.g., mobile phones, televisions, cars and home appliances, evolved to have constant access to the internet as well as connection to multiple applications, such as social networks and e-mail applications. The new features and the increase in computational capabilities on those devices created a realm of new opportunities for botmasters, allowing the migration of bots from traditional PC-based botnets to those devices (Silva et al., 2013). Another interesting point to make is that, with the increasing of those devices capabilities, users tend to store more sensitive information, e.g., photos and videos, SMS messages, and credit card and bank information, on their mobile devices than on their PCs (Ariyapala et al., 2016).

Botnets are composed of three main parts: the bots, the botmaster and the command and control (C&C) infrastructure. Bots are vulnerable devices compromised by malicious software called bot malware that work under the control of a malicious user, the botmaster. Lastly, the C&C infrastructure is the most critical component of a botnet. The botmaster uses it to communicate with bots, giving them instructions and receiving information from them (Silva et al., 2013).

According to Silva et al. (2013), the C&C infrastructure can have different architectures: centralised, decentralised, and hybrid. The first one is more common in older botnets and acts similar to a simple client-server network, where all bots communicate with the botmaster through one or a few servers. The main advantages of this infrastructure are the communication speed and the ease in monitoring the infected devices. On the contrary, its main problem is a central point of failure. Due to this characteristic, this kind of botnet can be easily disrupted by interrupting the communication between the bots and the centralised C&C servers. The decentralised approach uses peer-to-peer (P2P) protocols to exchange messages among the servers and the bots, meaning that discovering and neutralising some members of the P2P network does not compromise the botnet as a whole. P2P networks are designed around the idea that members of the network operate both as clients and servers, suppressing the need for central servers. The hybrid infrastructure combines features from both the centralised and decentralised architectures.

The different types of architectures (centralised, decentralised and hybrid) greatly increase the difficulty to use network-based defence tools against botnets since network patterns generated by botnets may differ a lot. Likewise, network-based approaches might not be adequate for mobile botnets due to the inherent portability of mobile devices, which will constantly move among different networks. This implicates in need of protecting the host itself, even while in transit.

To neutralise the threat imposed by botnets, one of the approaches is to first use an efficient technique to detect the bot malware. This approach must combine high detection rates, with a low false positive rate, while requiring minimal time to identify the malware. Additionally, it must be considered whether the approach will be host or network-based, or combine aspects of both. Host-based approaches are capable of monitoring particular actions of a single device. On the other hand, network-based approaches are often restricted to monitoring the network, which only contains, when considering data concerning botnets, traces of the communication between the bot and the C&C infrastructure and from attacks performed by the bots.

Machine learning (ML) algorithms have been consistently employed for pattern recognition tasks, which consist of automatically discovering regularities in data (Bishop, 2013). Botnet and mobile malware detection applications are no exceptions. Proposals, such as Stevanovic and Pedersen (2014), Ariyapala et al. (2016), Singh et al. (2014) and Chen et al. (2017) have a superior performance by employing ML techniques than signature-based solutions. Stevanovic and Pedersen (2014) used multiple ML algorithms to analyse statistical features extracted from network flows, obtaining a f-measure of 95.96% when using a random forest. Similarly, in Singh et al. (2014) presented a distributed solution to pre-process packets, extract features based on them and, by using a random forest, to detect malicious traffic, achieving high detection rates. Likewise, Chen et al. (2017) propose an algorithm to deal with high imbalance rates (IRs) between malware and legitimate network flows, obtaining an area under the curve (AUC) always greater than 90%. In this sense, ML has proved to be a great tool when identifying botnets. However, the existing solutions may lack generalisation power when considering multiple botnets with diverse behaviours. To address this, we used features with very low abstraction (directly from system call invocations) to capture actions that botnets usually do, i.e., steal sensitive data and communicate with the C&C infrastructure. By using features extracted from system calls, it is possible to analyse the botnet actions as a collection of multiple sequential low level operations, which may likely reflect botnets data gathering and communication behaviours (core actions of those malwares).

This work proposes an approach for mobile botnet detection based on features extracted from system calls. The features are analysed by supervised ML algorithms with the intent of identifying patterns generated by bot malware. The main contributions of this paper are the following:

- Eight ML algorithms from different theoretical perspectives were used and evaluated. Additionally, all hyperparameters were tuned for a fairer comparison and to achieve better results.
- Using χ^2 and information gain (IG), the dimensionality of the problem was reduced from 133 to 19 features without significant impact to predictive performance.
- An analysis of the features importance was performed. The resulting selected features represent a group of basic indicators of botnet behaviours that can be used to detect them in many scenarios.

This paper is an extension of a previous work published by the same authors (da Costa et al., 2017). The previous work focused on the possibility of detecting mobile botnets using system calls, presenting initial results for only two ML algorithms and a ranking of feature importance. The current work explores more algorithms and provides an in-depth analysis of the features used. The experiments carried out here were designed to evaluate the impact of selecting the best time-window, performing feature selection and tuning the ML algorithms. Lastly, this work also provides a reduced set of features that can be used without compromising predictive performance.

The rest of the paper is organised as follows: in Section 2, related work is presented, while a comparative relation between each of them and this work is also established. In Section 3, the proposed approach and the features used are presented. Section 4 presents the data gathering process and details the dataset creation. The performance metrics are presented in Section 5. Section 6 shows the experiments performed to evaluate the approach. In Section 7, a discussion about the features per botnet family is presented. Section 8 provides a general discussion. Section 9 contains the conclusion and future work.

2 Related work

The related work is divided into two main groups:

1 PC-based botnets

2 mobile-based botnets.

Given the multiple similarities between them, the contributions and insights of papers from both groups can be taken into account when developing a mobile botnet detection approach. The latter group can be further divided into two kinds of approach: one that aims to detect mobile malware, regardless of they are botnets or not; and the other that focus on detecting only malware related to mobile botnets.

On the PC-based botnets domain, Saad et al. (2011) merged different honeynet-generated datasets, containing traffic from P2P botnets, with a general traffic dataset, which ranged from web browsing to P2P gaming. The features used by them were partially host-based but mainly network-based. One of the network-based features was byte-related and counted the number of bytes per network flow, which was an inspiration to part of our features. They obtained almost 98% of recall by using a support vector machine (SVM) with linear kernel, while also maintaining a false discovery rate (FDR) around 6%. Nevertheless, their dataset contained only a small sample of P2P botnets, resulting in a very narrow range of behaviours across the botnets.

Stevanovic and Pedersen (2014) proposed a network-based botnet detection system using features extracted from network flows. Using the dataset generated by Saad et al. (2011), they compared different ML algorithms. Their best result was achieved by using a random forest, obtaining around 95% of precision and recall. While their results are very significant, they are also limited by having only two different botnets in their testing dataset.

With the goal of detecting C&C domains from HTTP-based botnets, Sakib and Huang (2016) proposed a framework that combined unsupervised and semi-supervised ML techniques. They used features based on HTTP and responses from DNS servers since they claimed that, by evaluating those features, it would be difficult for a botnet to mimic a legitimate application. They combined multiple HTTP and DNS responses datasets and later evaluated the amount of detected C&C servers. They achieved a high recall, detecting around 93% of the C&C domains but with precision around 27%.

Singh et al. (2014) developed a framework which uses the random forest algorithm to detect P2P botnets. The proposed framework consists of three parts: one responsible for pre-processing network packets, the second is responsible for extracting statistical features from network flows formed by those packets, and the last one is a distributed version of the random forest algorithm. This work had a high prediction performance but the solution proposed was not employed to mobile botnet detection.

On the mobile-based domain, Burguera et al. (2011) research focus on using system calls to detect mobile malware. Their approach applied the k-means algorithm using as features the number of occurrences of each different system call. Their experiments revolved around clustering the system call logs and checking whether the algorithm was capable of grouping the logs into two separate groups, one containing malware logs and the other legitimate application logs. The malicious application dataset used contained only two real-world malware and one self-written malware, resulting in a very narrow scope of possible botnet actions.

To detect HTTP-based mobile botnets, Geiri and Shah (2016) proposed the analysis of mobile applications with Logcat files. Logcat is an android-specific logging application that collects system messages, which consist of stack traces when the device throws an error, messages written by the developer and information sent through the network (Android Studio Development Team, 2017). In their experiments, by filtering those log files, they were able to observe leaked information about the device, which means that the malware was sending information about the device to the C&C infrastructure, leaving as future work the automation of the identification process.

Ariyapala et al. (2016) proposed a hybrid (host and network-based) model to detect mobile botnet applications. The features employed by them were derived from logs collected with Android Logger application. This application collects host-based data from the processes (applications) on the device, network data from them, e.g., the number of bytes sent through the interfaces and other statistical data, and general system information (CPU, battery and memory statistics). Also, they employed the Wireshark tool to collect network-based data. After a discretisation process, a Markov-Chain-based method was used to train a ML algorithm, leaving the proposed model validation as future work.

Handling as a binary classification problem, Karim et al. (2016) focused on recognising a mobile application between a general malware or a bot malware, ignoring legitimate applications in the scenario. The applications were evaluated in a sandbox using host-based features, ranging from file activity to network operations. Their problem proved to be linearly separable, since the best performing algorithm was the logistic regression, achieving close to 100% performance across the metrics. Although the results are good, the proposed approach was employed to a relatively small number of malware samples with similar behaviours, being closely connected to SMS messages.

Yuan et al. (2016) combined various legitimate android applications and malware to validate an approach to detect mobile malware with host-based features by deep learning algorithms. They combined static-analysis and dynamic features collected by using a sandbox environment. Static-analysis features were extracted directly from applications' source code, while dynamic features were created based on their actions. By using those two types of features, they were able to achieve high performance using a deep belief network (DBN) and a SVM, but, when using only dynamic features, both algorithms performed poorly.

Another novel solution was proposed by Chen et al. (2017) by observing the pattern of the network flows generated by malware. Considering the real life, the authors stated that the number of malware to legitimate applications network flows would have an approximated IR of 1 to 4,565, resulting in a highly imbalanced problem. When traditional ML algorithms were employed, e.g., random forest, and even including algorithms for imbalanced problems, their performance was low. However, the authors proposed a modified version of the imbalanced data gravitation-based classification (IDGC) algorithm, replacing its optimisation mechanism for a more lightweight one. This modified approach was able to achieve AUC greater than 90%, even when the IR reached 1 to 7,000.

Some works focused on comparing the performance of different supervised classification algorithms on mobile malware detection. Narudin et al. (2016) proposed a network-based approach to detect mobile malware. The proposed approach selected six out of eleven TCP packet features using the ClassifierSubsetEval method. Then, the features were extracted and the following classification algorithms were evaluated: J48, Bayes network, multilayer perceptron (MLP), k-nearest neighbours (k-NN), and random forest. The results pointed out that random forest and Bayes network reached the best performances. Our work has some differences to the work by Narudin et al. (2016). We evaluated some classification algorithms that are not present on their work such as SVM, gradient boosting and extreme gradient boosting. Besides, our work aims to detect only botnet malware and relies on host-based data, while Narudin et al.'s (2016) is based on network data analysis to detect any type of malware.

Mas'ud et al. (2014) used five different groups of features and evaluated five classification algorithms over them. Four groups of features were based on previous work. The fifth group was generated using χ^2 and IG. The five classification algorithms evaluated were the following: naive Bayes (NB), k-NN, J48, MLP and random forest. According to their results, the best accuracy was reached with the combination of IG, χ^2, and MLP. The work by Mas'ud et al. (2014) has similarities to our work. It also uses system call features and employs IG and χ^2. However, it is not focused on botnet malware and did not evaluate some algorithms such as SVM and extreme gradient boosting.

In Table 1 a summary of all the related work is presented.

Despite many similarities between PC-based and mobile-based botnets, there is still a need for studies focusing on the latter, since the devices from these two types of botnets operate differently and have different computational power and capabilities. Also, mobile botnets steal different kinds of information, which also tend to be far more available for them (Ariyapala et al., 2016).

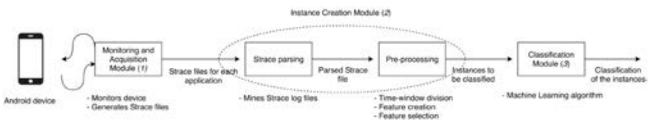
The lack of studies focusing on mobile botnet detection, with the few ones validating their solutions in relatively simple scenarios, is another key motivating factor. Another point is that the majority of works proposed a network-based approach. While this kind of approach scales better and is more lightweight, it is not feasible to assume that user devices would be connected all the time to a network monitoring system, such as an intrusion detection system (IDS). Likewise, those mobile devices are, as the name implies, mobile, and will constantly be connecting to different networks, creating the need for protecting themselves without relying on network methods. Lastly, when considering the ML algorithms employed on those works, some state-of-the-art algorithms were not used, e.g., the extreme gradient boosting, which tends to outperform other algorithms (Chen and Guestrin, 2016).

Table 1 Summary of related work

Work	*Detection objective*	*Features type*	*Modelling*
Saad et al. (2011)	P2P PC-based botnets	Network and host-based from flows	Used a SVM with RBF and one with linear kernel
Stevanovic and Pedersen (2014)	P2P PC-based botnets	Network-based from flows	Used multiple supervised ML algorithms, achieving the best performance using a RF
Singh et al. (2014)	P2P PC-based botnets	Network-based from flows	Proposed a large scale distribute system to identify P2P PC botnets using a RF
Sakib and Huang (2016)	C&C domains from HTTP PC-based botnets	Network-based on HTTP and DNS servers responses	Proposed a framework that combined unsupervised and semi-supervised ML techniques
Burguera et al. (2011)	Mobile malware	Host-based revolving around system calls	Clustered instances and checked if the resulting clusters contained only legitimate or only malicious applications
Mas'ud et al. (2014)	Mobile malware	Host-based revolving around system calls	Compared five traditional supervised ML algorithms over five groups of features
Geiri and Shah (2016)	HTTP-based mobile botnets	Extracted from Logcat files	Manual observation of information leak from the device to the C&C infrastructure
Ariyapala et al. (2016)	Mobile botnets	Host and network-based	Created a theoretical model based on the Markov-chain algorithm to train a ML algorithm in the future
Karim et al. (2016)	Mobile botnets from multiple types of malicious applications	Host-based	Multiple supervised ML algorithms, achieving the best performance with a logistic regression
Yuan et al. (2016)	Mobile botnets	Static and dynamic analysis of the applications	Used traditional supervised ML and deep learning
Narudin et al. (2016)	Mobile malware	Network-based from TCP packets	Compared traditional supervised ML algorithms
Chen et al. (2017)	Mobile malware	Network-based from flows	Proposed a new ML algorithm to deal with high imbalanced problems

This work proposes a host-based approach to detect mobile bot malware from legitimate applications that only use host-based features, which better addresses the mobile aspect of those devices. Also, the features used here were selected by using heuristic metrics, such as the IG, resulting in a small number of features. Multiple ML algorithms with different theoretical foundations were employed, and their performances were evaluated on a close-to-reality scenario containing applications with a broad range of behaviours. Furthermore, experiments were thoroughly created to evaluate different aspects of our proposed approach.

Figure 1 Diagram of the proposed approach


Instance Creation Module (2)
Android device
Monitoring and Acquisition Module (1)
- Monitors device
- Generates Strace files
Strace files for each application
Strace parsing
- Mines Strace log files
Parsed Strace file
Pre-processing
- Time-window division
- Feature creation
- Feature selection
Instances to be classified
Classification Module (3)
- Machine Learning algorithm
Classification of the instances

3 Proposed approach

In this section, the proposed approach is discussed. Figure 1 is a general representation of the approach to identify mobile android botnets by using a host-based approach. The method combines three modules: the monitoring and acquisition module (1); the instance creation module (2); and the classification module (3). Module 1 is responsible for collecting the data from the mobile android device. Module 2 parses the data and pre-processes it with the intent to generate the instances for the last module. This pre-processing consists of grouping the system calls by time-windows, creating the features for each window, and performing the feature selection step. Therefore, each instance is composed of the features corresponding to a time-window of a given application. The last module is a ML algorithm responsible for classifying the instances generated by module 2.

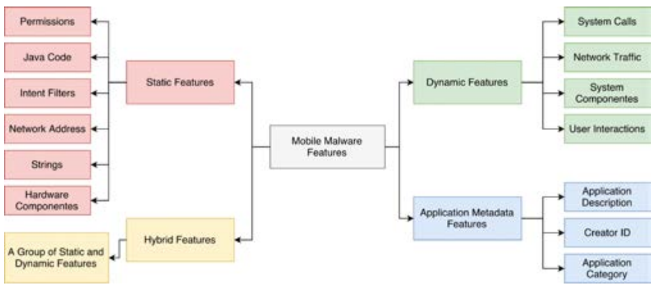
3.1 Botnet detection descriptors

Detecting mobile malware in a host-based approach involves features from multiple domains. Feizollah et al. (2015) presented a deep research about features to be used in mobile malware detection. Figure 2 presents the taxonomy of those possible features. In the mobile domain, the most used features from the dynamic branch are related to system calls, which are defined as a fundamental interface between applications and the kernel, and the interaction with the lowest level of abstraction an application can have with the kernel (Kerrisk, 2016). Our features are directly linked to system calls, taking advantage of analysing applications using one of the most generalising host-based features.

In the proposed approach, at first, it is necessary to collect data from system calls invoked by applications on an android device. The device needs to be rooted, so it is possible to install a tool for tracing the system calls invoked by the running applications. In an Unix environment, the most known tool for this purpose is the Strace tool. This tool is used for application debugging and analysis, being able to collect the invocations of system calls of a given process (application, in the mobile domain). It generates a log file where each line corresponds to a system call invocation, containing its parameters, return value and a timestamp. Module 1 continuously monitors the device, starting Strace calls on a newly created process with user-level permissions and redirecting its output to a text file.

To create each instance to be used in the classification module, it is needed to group all system calls invoked during a time-window period by an application. After that, module 2 creates features for each window that can be divided into three types, one being count-based, another byte-based and the last one is generalising, which represent actions with a higher abstraction level, e.g., observe the total number of actions performed. The count-based features are a mere counting of how many times a system call was invoked during that period. The final amount of count-based features depends on how many system calls are invoked throughout the system monitoring. In our tests, for example, 100 system calls were invoked and 100 count-based features were created. The byte-based features are more complex. In network-based approaches (Livadas et al., 2006; Garg et al., 2013; Stevanovic and Pedersen, 2014), some features are related to bytes (number of bytes in a network flow, for example), which granted good performance results on their experiments. By extending this idea that the volume of bytes involved in the operations performed by bot malware are important, byte-based features were generated for the system calls which the number of bytes would matter to. These features are presented in Table 2. In these calls, the return values of the system call invocation represent the number of bytes handled by the operation.

Figure 2 Taxonomy of features for mobile malware detection (see online version for colours)



Source: Adapted from Feizollah et al. (2015)

Based on the system calls presented in Table 2, different byte-based features were generated. There are 31 byte-based features, which are classified in the following seven categories:

1 A count of the total number of bytes involved in all invocations of a system call.

2 The average of bytes per system call.

3 The standard deviation of bytes per system call.

4 The number of bytes involved in all invocations of all write-type operations.

5 The number of bytes involved in all invocations of all read-type operations.

6 A ratio between the number of bytes wrote and read, where values close to
0 represent more bytes involved in read operations, while high values represent the opposite. Values close to 1 represent the balance between the application writing and reading bytes.

7 A ratio between network operations to send and receive data, working the same way as the write-read ratio.

The third type of features are the most generalising and contains two features. One that counts the number of distinct system calls invoked during the time-window, and the other that counts the total number of system calls invoked. The combination of the three types of features proposed resulted in a total of 133 features.

Table 2 System calls involving bytes

Type	*System call*	*Action*
Write	write	Writes on a file descriptor
	writev	Writes on multiple file descriptors
	pwrite	Performs the same action as the write system call, but with an offset
Read	read	Reads a file descriptor
	pread	Performs the same action as the read system call, but with an offset
Network operations	sendto	Sends bytes through a socket
	sendmsg	Sends action as the sendto system call
	recvfrom	Receives bytes through a socket
Directory operations	geetdents64	Reads entries in a directory

The generated instances are then fed to the module 3, which is responsible for analysing the data and identifying whether an instance belongs to a bot malware process. The classification module is a supervised ML algorithm and, in this sense, it needs to be previously induced using pre-labelled instances.

3.2 ML algorithms

Next, a brief overview of the algorithms used here are provided in this subsection, along with their strong and weak points and the reason they were chosen.

3.2.1 Decision tree

Decision tree is an algorithm that creates a model (in the format of a tree) by using a top-down approach. Each inner node of the tree is a split node representing a feature and a threshold value chosen as split condition. At the leaves of the tree, it is possible to do predictions on new instances based on previously labelled elements that fall on that leaf (Breiman et al., 1984; Loh, 2008).

To create a tree, at each node of the tree, the best features are selected according to a heuristic metric. IG and Gini impurity are the most commonly used, with the first being more broadly used in recent implementations of decision trees. After choosing a feature, the node is branched according to the data type of the feature. If the feature is categorical, there will be *m* new branches, where *m* is the number of possible values for that feature. Otherwise, if the feature is numerical, the split factor is two, and the split condition is in the form of $x_i \leq c$, where x_i is the value of the chosen feature *i* in a given instance, and *c* is the chosen split point (Breiman et al., 1984; Loh, 2008).

The main advantages of using a decision tree algorithm are the ease of interpreting trees with low depth and that decision trees handle nonlinear problems while being a fairly lightweight algorithm (Loh, 2008).

3.2.2 Gradient boosting machine

The gradient boosting machine is a framework for function approximation proposed by Friedman (2001). When used as a ML algorithm, the idea behind it is to learn by consecutively fitting new models with the goal of providing a more accurate estimation. This is done by making the models to be maximally correlated to the negative gradient of the loss function chosen by analysing the problem, i.e., each model will try to address the error of the previous models. The models normally used are weak learners, given the idea that, by combining weak learners, it is possible to create a strong learner (Friedman, 2001; Natekin and Knoll, 2013). In theory, any ML algorithm can be used with the framework, although in practice weak learners are used in almost all cases.

The GBM is highly flexible since any function can be used as the loss function, giving freedom to the researchers to choose, or even create, a function that best fits any data-driven task. There is also the possibility of using any weak learner, such as, decision trees. This algorithm has had success in real world problems and in ML and data-mining competitions (Natekin and Knoll, 2013).

3.2.3 K-nearest neighbours

The k-NN is a classical algorithm for classification and regression problems. For classification, the basic implementation of the algorithm firstly loads all labelled data in memory. After that, the distances among all labelled data points and the unknown instances are computed. Any distance metric can be used, with the most common one being the Euclidean distance. The *k*, an empirical parameter, closest labelled instances to the unknown instance are chosen and used to perform the classification, selecting the mode of the classes of the *k* elements (Jiang et al., 2007; Cover and Hart, 2006).

This algorithm does not perform feature selection and does not handle problems with very high dimensions. Another problem of the algorithm is the need to properly scale the features since features with high values would dominate the computation of the distance values. Being a pretty straightforward and simple algorithm, it was chosen as to validate the quality of the features in the proposed framework.

3.2.4 *Multilayer perceptron*

MLP is a general purpose feed-forward network (a class of artificial neural networks). This algorithm is flexible, handles nonlinear problems, and, given enough hidden layers and neurons, can approximate virtually any function to any desired accuracy, making it a universal approximator (Ruck et al., 1990; Sarle, 1994).

The general architecture is composed of an input layer, one or more hidden layers, and one output layer. The input layer receives the real feature values and feeds the first layer of hidden neurons. Then, those neurons do successive computations with their input values and weights producing output values that will be fed to the next layer. After all the hidden layers, the output layer deals with computing the prediction of the network. The training part of a MLP consists of computing the gradients and adjusting the weights at each layer according it. A common and very successful algorithm for this purpose is the back-propagation algorithm (Ruck et al., 1990; Sarle, 1994).

The idea behind MLP is to simulate brain connections to learn models that deal with linear and nonlinear data. One downside is the wide range of possible parameters configurations and tuning, making the good parametrisation of the network difficult and a problem-specific task (Sarle, 1994).

3.2.5 *Naive Bayes*

The NB algorithm is based on the Bayes's theorem and it works by making the naive assumption that all attributes are independent, given the value of a class label. It computes the frequency of occurrences of each feature for each class, and then combines all the frequencies with the frequency of the classes themselves, resulting in a set of probabilities (Lewis, 1998; Patil, 2013).

Although it has a naive view of real world problems, since the independence of attributes is rarely true, it tends to perform well and fast in these problems. It is also capable of dealing with missing values. Lastly, this is an on-line algorithm, where one can feed more instances to an already induced NB predictor (Lewis, 1998; Patil, 2013).

3.2.6 *Random forest*

Random forest, which was proposed by Breiman (2001), is an ensemble algorithm that works by growing a forest of decision trees and using the trees in conjunction to perform classification and regression operations. It was also introduced by Breiman the idea of bagging, a great part of random forest's success.

At first, the algorithm samples with replacement from the training dataset *n* instances, *n* being the size of the dataset. After that, the algorithm samples a number of features, defined by the user, and builds a decision tree using the sampled instances and features. This process is repeated to create the number of trees desired. After building the forest, it is used to perform classification through a voting process or perform regression by averaging the response values of each tree. Since it uses an ensemble of decision trees, the feature selection capabilities of them are inherited (Breiman, 2001).

The main advantages of this algorithm are its robustness to deal with outliers and noise and that it does not overfit to data (Breiman, 2001). Another interesting point is the ability to use the examples left out of the sampling process, called out-of-bag samples, to measure the performance of the built trees.

3.2.7 *Support vector machine*

The SVM, as described in Hearst et al. (1998) and Cortes and Vapnik (1995), is a traditional algorithm for binary problems (later adapted for multi-class problems). The objective of the SVM is to create the optimal separation hyperplane that divides the two classes. The points used at the boundaries of this hyperplane are called support vectors, thus naming the algorithm.

When data is not linearly separable, the algorithm can apply an operation called kernel trick. In that operation, the input features are mapped to higher dimensional spaces, based on a kernel function. This mapping has the intent to make the problem simpler by creating a higher dimensional space (Hearst et al., 1998; Cortes and Vapnik, 1995).

This algorithm handles high dimensional data, having guaranteed performance, as it is based on the statistical learning theory, and the training is very robust and efficient, being a good generalising algorithm (Hearst et al., 1998; Cortes and Vapnik, 1995).

3.2.8 *Extreme gradient boosting*

The extreme gradient boosting (XGBoost) is a variation of the GBM algorithm that uses only trees. The two main differences between them rely on modelling details and speed-focus of the XGBoost. While modelling, the XGBoost algorithm works with a regularisation parameter, which makes it perform better on real-world problems than GBM. When this regularisation parameter is zero, the optimisation objective falls back to a traditional GBM using trees (Chen and Guestrin, 2016).

Outperforming the traditional GBM algorithm, while also being easily parallelisable, the XGBoost algorithm is a state-of-the-art ML algorithm capable of dealing with multiple real-world problems and winning multiple competitions (Chen and Guestrin, 2016).

4 Data gathering and dataset creation

A Samsung tablet running android was used to carry out the experiments. A total of 31 mobile botnet applications selected across 13 different families were installed in this device. Table 3 presents the number of applications per botnet family and Table 4 presents some characteristics of each botnet family. A total of 88 legitimate applications were executed during the experiment, divided into core android apps, e-mail services (Gmail), games, music streaming applications (Spotify), navigation services (Google Maps and Waze) and YouTube. The device also included some preinstalled legitimate applications. Botnet applications were downloaded from the ISCX android botnet dataset generated by Abdul Kadir et al. (2015). As described in da Costa et al. (2017), applications were installed individually and also collectively, to create a scenario where they would compete for resources. The device was monitored during a non-sequential period of seven days, in which the applications were installed and ran on the device under different circumstances, i.e., without any other CPU-intensive application or with many applications competing for resources, and at different periods of the day. After collecting the needed data, the processes related to bot malware were manually labelled, and the parsed Strace files were created. Manual labelling was possible since we knew which processes corresponded to the bot malwares. These parsed Strace files are in the CSV format where each line corresponds to an invocation of a system call, containing the timestamp, the system call invoked and its returned value. After that, we created the instances used for training and testing in the manner described before, in Section 3, i.e., grouping system calls invocations by time windows (1s, 5s and 10s) and extracting the features. More details about these datasets can be found in da Costa et al. (2017), as the same datasets created on this previous work was used. Also, the datasets are publicly available on the web (http://www.uel.br/grupopesquisa/secmq/dataset-mobile-botnet.html).

Table 3 Applications per botnet family

Botnet family	*# of applications*
Anserverbot	4
BMaster	2
DroidDream	3
Geinimi	5
MisoSMS	3
NickiSpy	2
NotCompatible	1
PJapps	2
Pletor	1
Rootsmart	1
Sandroid	4
Tigerbot	1
Zitmo	2

Table 4 Characteristics of botnet families

		Propagation and attack types									
Botnet family	*C&C infra*	*Backdoor*	*Download*	*Exploit technique*	*Infected SMS*	*Repackaged app*	*Social engineering*	*Trojanised app*	*Data theft*	*Mobile banking attack*	*Ransomware*
Anserverbot	HTTP				X		X		X		
BMaster	HTTP			X					X		X
DroidDream	HTTP		X	X		X		X	X		
Geinimi	HTTP		X			X			X		
MisoSMS	Email			X				X	X		
NickiSpy	SMS					X			X		
NotCompatible	HTTP		X	X							
PJapps	HTTP					X		X	X		
Pletor	SMS/HTTP							X			X
Rootsmart	HTTP			X		X			X		
Sandroid	SMS							X		X	X
Tigerbot	SMS	X						X	X		
Zitmo	SMS				X	X	X		X	X	

Source: Adapted from Abdul Kadir et al. (2015)

Table 5 Resulting datasets after performing stratified sampling

Dataset	*Training*		*Testing*		*Percentage of botnet instances*
	Legitimate	*Botnet*	*Legitimate*	*Botnet*	
1s	16,667	2,999	7,143	1,286	0.180
5s	8,105	743	3,475	319	0.092
10s	5,943	392	2,547	168	0.066

5 Performance metrics

To numerically measure the performance of our approach, the following metrics were employed, as described in Sokolova and Lapalme (2009):

1 *accuracy (ACC):* $ACC = \frac{TP + TN}{TP + FP + TN + FN}$, which represents the percentage of instances correctly classified

2 *precision (PREC):* $PREC = \frac{TP}{TP + FP}$, which represents the percentage of instances correctly classified as botnet

3 *recall (REC):* $REC = \frac{TP}{TP + FN}$, which represents the effectiveness of the classifier to identify botnet instances

4 *specificity (SPEC):* $SPEC = \frac{TN}{TN + FP}$, which represents the effectiveness of the classifier to identify legitimate instances

5 *area under the curve (AUC):* $AUC = \frac{1}{2} * (recall + specificity)$ which represents the capability of the classifier, in percentage, to avoid false classification

6 *false positive rate (FPR):* $FPR = 1 - specificity$.

6 Experiments

To perform the experiments, we first divided the three datasets (one for each time-window) into training and testing by randomly selecting instances while maintaining the same botnet-legitimate ratio in the training and testing datasets. The class distributions of the resulting datasets as well as the amount of instances are presented in Table 5.

6.1 Experiment 1

The goal of the first experiment was to select the best time-window, favouring a smaller and better performing window. To do so, a RF with default hyperparameters (1,000 trees) was induced on the training dataset and evaluated on the test dataset. We only used the RF as it is not so highly dependable on hyperparameter tuning as a MLP and yields a high predictive performance.

In Table 6 the performance metrics for each time-window are presented and Tables 7, 8 and 9 present their respective confusion matrices.

At first, it is possible to see an increase in accuracy proportionally to the time-window. However, this came at the cost of reducing recall. When designing solutions to detect malware applications, a balance between detecting the majority of malwares while not classifying many legitimate instances as botnet is needed. In this sense, it is better to maximise both metrics. Although there is an increase in FPR, it is minimal, when compared to the high decrease in recall, of around 10%. So, while maximising performance and striving to select the smallest possible time-window, the best value is 1s.

Table 6 Performance metrics achieved by the RF for each time-window

Dataset	*ACC*	*PREC*	*REC*	*SPEC*	*AUC*	*FPR*
1s	0.962	0.964	0.784	0.995	0.889	0.005
5s	0.976	0.945	0.755	0.996	0.875	0.004
10s	0.979	0.982	0.678	0.999	0.839	0.001

Table 7 Confusion matrix using a time-window of 1 second

Real/predicted	*Legitimate*	*Botnet*
Legitimate	7,106	38
Botnet	278	1,008

Table 8 Confusion matrix using a time-window of 5 seconds

Real/predicted	*Legitimate*	*Botnet*
Legitimate	3,461	14
Botnet	78	241

Table 9 Confusion matrix using a time-window of 10 seconds

Real/predicted	*Legitimate*	*Botnet*
Legitimate	2,545	2
Botnet	54	114

6.2 Experiment 2

After selecting the best performing time-window, a feature selection process was employed considering only datasets for the 1 second window. Using the training dataset, the features were ranked using two heuristic metrics that measure the importance of a feature in our classification problem: IG and $\chi(chi)^2$.

IG is a metric that computes the amount of knowledge gained about a problem related to a given feature. At first, to compute the IG of a feature *c*, it is necessary to find all possible split values for that feature. After that, for each possible split value *v*, the original dataset *T* is divided into two, one containing the instances where, for the feature c, it has a value less or equal to v, and the other where the value is greater than v. The entropy of each sub-dataset S is computed using:

$$H(S, c) = -\sum_{i=1}^{J} fi \log_2 fi \tag{1}$$

where fi is the probability of randomly picking an element with the class i in the sub-dataset S and J is the total number of classes in the problem, in our case, two (botnet or legitimate instance).

Then, the weighted average entropy of the entropy of all sub-datasets resulting from the split, where the datasets are numbered from 1 to D, is computed by applying the following:

$$H(T, c) = \frac{1}{\mu(T)} \sum_{i=1}^{D} H(S_i, c)\mu(S_i) \tag{2}$$

where $\mu(X)$ is the size of a X dataset.

The IG is calculated by the difference between the entropy of the dataset T and the entropy of T related to the feature c:

$$IG(T, c) = H(T) - H(T, c) \tag{3}$$

After testing all the possible split values, the highest IG will be assigned as the IG of that feature c.

The χ^2 is a statistical metric that measures the dependence between a given feature f and whether an instance was generated from a botnet application or a legitimate application. By verifying this dependence, it is possible to rank the more dependent and statistically better features.

To select the features, the values of IG and χ^2 were first scaled from 0 to 1 separately. This is done by applying the following function to both separated ranks:

$$X_{scaled} = \frac{X - \min(X)}{\max(X) - \min(X)} \tag{4}$$

where X is the set of all IG or χ^2 values.

Then, an empirical threshold was applied to each metric, removing features with the importance value lower than the threshold. Two sets of selected features were generated, one per metric. After that, a full-join operation was performed on the sets, meaning that features remaining in the IG, χ^2 or both sets were selected. The threshold value was varied between 0.1 and 0.9 in steps of 0.1.

Considering each threshold value separately, a RF was induced in the training dataset and evaluated in the test dataset using only the remaining features according to each threshold value. The performance metrics obtained are presented in Figure 3.

Table 10 Rank of selected features using a 0.5 threshold ordered by average χ^2 and IG

#	*Feature*	*Details*	*Scaled* χ^2	*Scaled IG*	*Average*
1	send_receive_ratio[b]	Ratio between bytes sent and received.	0.545	0.144	0.344
2	recvfrom_bytes_avg[b]	Average bytes received by recvfrom system call.	0.669	0.114	0.391
3	ioctl[a]	Manipulates underlying device for input/output operations.	0.553	0.353	0.453
4	gettid[a]	Get the threads ID.	0.603	0.406	0.504
5	access[a]	Check if the caller can access a file.	0.819	0.320	0.569
6	fstat64[a]	Get information about a file.	0.860	0.360	0.610
7	close[a]	Closes a file descriptor.	0.864	0.367	0.615
8	gettimeofday[a]	Get the current time.	0.815	0.655	0.735
9	open[a]	Opens a file descriptor.	1.000	0.472	0.736
10	getpid[a]	Get process ID.	0.828	0.700	0.764
11	getuid32[a]	Get user ID.	0.845	0.757	0.801
12	read_bytes_sd[b]	Standard deviation of bytes in read system call.	0.886	0.803	0.844
13	epoll_wait[a]	Waits for an event of the event poll.	0.883	0.823	0.853
14	read_bytes_avg[b]	Average of bytes in read system call.	0.898	0.828	0.863
15	read[a]	Read bytes from a file descriptor.	0.943	0.911	0.927
16	total_syscalls[c]	Number of different system calls invoked.	0.930	0.939	0.934
17	clock_gettime[a]	Get time of a specific clock.	0.958	0.945	0.951
18	total_read[b]	Amount of bytes read by all read-type system calls.	0.978	0.996	0.987
19	read_bytes[b]	Amount of bytes read by read system call.	0.979	1.000	0.989

Note: [a]Count-based features. [b]Byte-based features. [c]Generalising features.

It is possible to see that there was no impact in removing the majority of features contained in the dataset (from 133 to 19 features). In this sense, we chose the 0.5 threshold (denoted by the black dotted line) as the best cutting point, since there was no reduction in predictive performance.

The 19 features selected when using a threshold of 0.5 are presented in Table 10.

From the 19 features:

1 Two features (1 and 2) are related to network communication, which is particularly relevant for botnets: this is either related to the communication between the botnet malware and its C&C infrastructure or an attack being performed by the bots. The malware is sending data gathered from the device or receiving new instructions.

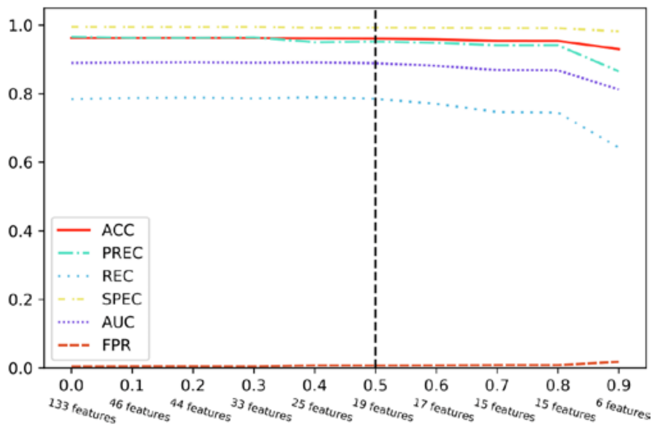
2 Two features (8 and 17) are linked to gathering information about the current time. First, this can be associated to botnet malwares going into sleep for an amount of time to avoid being detected. Likewise, some botnets only work in specific times of day, mostly during the night, when the device will likely be on standby.

3 Ten features are related to gathering information about the device. An important part of botnets is to identify vulnerabilities and to steal sensitive information. This is reflected by the presence of ten features (3, 5, 6, 7, 9, 12, 14, 15, 18 and 19) related to these actions.

4 Four features regarding asynchronous code execution. The features 4, 10, 11, 13 are related to the botnet malware performing asynchronous actions.

5 Lastly, one feature (16) that captures the diversity of actions performed by different applications.

Figure 3 Performance according to threshold value used for feature selection (see online version for colours)



6.3 Experiment 3

After performing the feature selection step, the ML algorithms presented previously (in Section 3.2) were evaluated in the training and testing datasets. A total of eight algorithms was used, those being: Scikitlearn's CART, a kind of decision tree; GBM; k-NN; MLP; NB; RF; SVM; and XGBoost. The algorithm implementations from the scikit-learn library (Pedregosa et al., 2011) were used, except for the XGBoost algorithm. For that, the library developed by Chen and Guestrin (2016) was used.

Grid search was used to optimise k-NN and CART due to their low amount of hyperparameters. For NB, the default hyperparameters were used. For the other algorithms, a random search was employed. Bergstra and Bengio (2012) provided a theoretical and empirical result that random search performs the same or outperforms grid or manual search taking less time to run than both the other strategies. Note that the algorithms were tuned by performing a ten-fold cross validation considering only the training dataset focusing on maximising AUC. In this sense, the training dataset is divided into ten parts. At the first iteration, the parts from 1 to 9 are used for training, while part 10 is used to evaluate the model. Then, at the second iteration, parts 1 to 8 and 10 are used for training, and the model is evaluated on part 9. This process is repeated until all parts were used for testing. The hyperparameters selected are the ones that resulted in the highest mean AUC value after the ten-fold cross validation. Table 11 shows the hyperparameters optimised for each algorithm.

Table 11 Hyperparameters optimised for each ML algorithm

Algorithm	*Hyperparameters*	*Optimisation strategy*
CART	Split criterion (Gini or IG); max depth	Grid search
GBM	Max depth; max features when search for the best split; learning rate	Random search
k-NN	Number of neighbours	Grid search
MLP	Number layers and neurons per layer	Random search
NB	No hyperparameters	Nothing
RF	Max depth; max features when search for the best split; split criterion	Random search
SVM	Kernel (linear or rbf); C (penalty parameter); class weight (adjust C according to class distribution); γ (kernel coefficient for RBF)	Random search
XGBoost	Max depth; learning rate	Random search

Table 12 contains the performance metrics (ACC, PREC, REC, SPEC, AUC and FPR) obtained by the tuned algorithms when trained on the training dataset and evaluated on the testing dataset. Considering PREC and SPEC, the best values were obtained by the RF algorithm. For REC, the SVM outperforms the other algorithms, followed close by the XGBoost. AUC values are the highest for GBM, SVM and XGBoost. In general, ensemble algorithms (GBM, RF and XGBoost) and SVM outperforms the other algorithms across different metrics.

Table 12 Performance metrics for each ML algorithm

Algorithm	*ACC*	*PREC*	*REC*	*SPEC*	*AUC*	*FPR*
CART	0.945	*0.927*	0.696	*0.990*	0.843	*0.009*
GBM	*0.959*	*0.927*	*0.793*	0.988	*0.891*	0.012
k-NN	0.948	0.919	0.722	0.988	0.855	0.012
MLP	0.945	0.916	0.708	0.988	0.848	0.012
NB	0.795	0.369	0.481	0.851	0.666	0.149
RF	*0.959*	*0.964*	0.760	*0.994*	0.877	*0.006*
SVM	0.927	0.733	*0.821*	0.946	*0.883*	0.054
XGBoost	*0.961*	*0.938*	*0.801*	*0.990*	*0.896*	*0.010*

Lastly, the algorithms provide the probability of each instance being legitimate or botnet. However, by default, algorithms consider a 0.5 threshold, meaning that, when an instance has more than 0.5 of probability of being botnet it is classified as such. Since the class distributions are imbalanced for the dataset used, the threshold applied to the probabilities was also tuned for each algorithm using a random search with the same ten-fold cross validation strategy considering only the training dataset and striking for a balance between precision and recall.

Table 13 presents the performance metrics for the algorithms after tuning the probability threshold. Although the ACC values for the algorithms decreased, PREC and REC are more balanced. In this sense, although the majority algorithms classify more legitimate instances as being botnet (i.e., higher FPR), more botnet instances are being detected, which is highly desired. Generaly, algorithms traded a similar amount of PREC for an increase in REC. However, considering that high values for both metrics are desirable, the trade-off is beneficial.

Table 13 Performance metrics for each ML algorithm after tuning the probability threshold

Algorithm	*ACC*	*PREC*	*REC*	*SPEC*	*AUC*	*FPR*
CART	0.936	0.795	0.782	0.963	0.873	0.037
GBM	*0.958*	*0.880*	*0.842*	*0.979*	*0.910*	*0.020*
k-NN	0.946	0.850	0.791	0.974	0.883	0.026
MLP	0.943	0.831	0.789	0.971	0.880	0.029
NB	0.775	0.339	0.497	0.825	0.661	0.175
RF	*0.960*	*0.889*	*0.842*	*0.981*	*0.911*	*0.019*
SVM	0.945	0.869	0.752	*0.979*	0.866	*0.021*
XGBoost	*0.958*	*0.877*	*0.849*	0.978	*0.913*	0.022

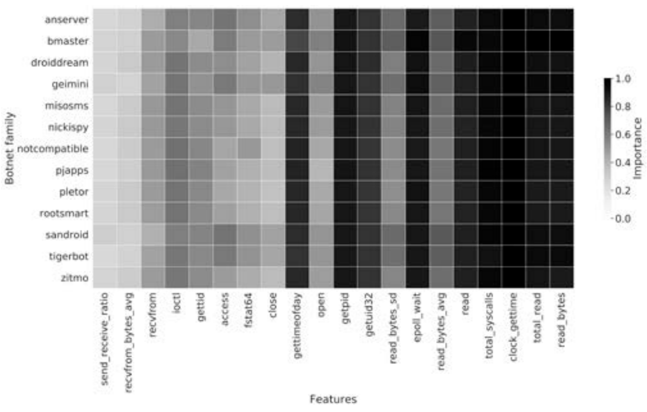
7 Features by botnet family

Our original problem was to detect whether an instance was legitimate or botnet. However, to perform a deeper feature analysis of which features are more important to identify an specific botnet family, we changed our classification problem. For each botnet family, a dataset was created, where the label of instances belonging to that botnet family is 1 and all the other instances (legitimate instances or from a different botnet) have a label 0.

For each dataset, we calculated the χ^2 and *IG* of all features. After that, two heat maps were generated using the importance factor metric. The y-axis of the heat map represents the botnet families, and the x-axis represents the features that have higher importance value than the threshold. The colour of each block is related to the importance of the feature and botnet family, where darker colours represent higher importance values, in contrast to brighter colours, which represent less important features.

Two heatmaps were created, one for *IG* (Figure 5) and another for χ^2 (Figure 4) considering the 19 features selected by the 0.5 threshold. The y-axis corresponds to the botnet family while the x-axis to the feature (ranked by worst to best when trying to identify whether or not an application is a botnet). First, let us consider only χ^2. When trying to identify botnet applications from an specific family, the importance of the features *gettimeofday*, *getpid*, *getuid32*, *epoll_wait*, *read*, *total_syscalls*, *clock_gettime*, *total_read* and *read_bytes* is very high and the same. *Read_bytes_sd* and *read_bytes_avg* are more important when trying to identify some botnet families, e.g., Anserver, Bmaster, Geinimi, Sandroid and Tigerbot. However, both features as well as *open* have low importance factor. This means that, when trying to classify an application as legitimate or botnet, these features are important, but when trying to identify a specific type of botnet, they do not help since they reflect a behaviour shared by all botnets. To some scale, this also happens for the remaining features, since they were selected when the classification problem was to detect botnet applications, but had importance values around 0.2–0.4 when identifying a specific botnet family. Considering *IG*, the patterns in the importance factors are very similar, with the only difference being that *IG* assigns features with lower importance values.

Figure 4 Heatmap of importance factor using χ^2 of the features selected using a threshold of 0.5 by botnet family

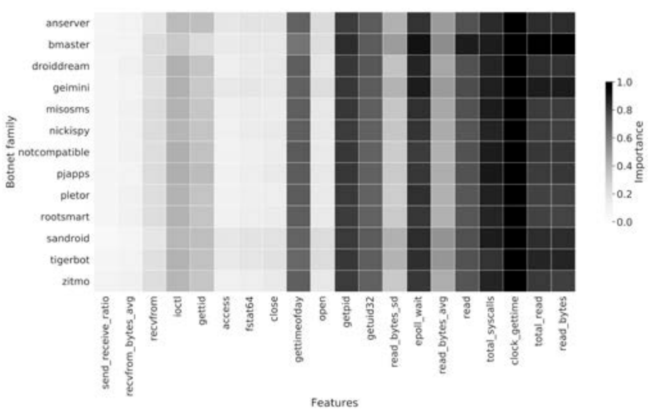


8 General discussion

This work focused on addressing the main weak points of the host-based approaches in Burguera et al. (2011), Geiri and Shah (2016), Ariyapala et al. (2016) and Karim et al. (2016), while also proposing the use of byte-based features, similarly to network-based approaches. Burguera et al. (2011) used the same count-based features as ours, but they were used in conjunction with an unsupervised ML approach and their work needed a post analysis by a specialist. Both Geiri and Shah (2016) and Ariyapala et al. (2016) suggested a modelling approach, leaving the detection of mobile botnets for future work. Karim et al. (2016) did not include any legitimate applications in their tests. Thus their solution was not evaluated in a close-to-reality scenario. We addressed this by creating a dataset composed of multiple botnet and legitimate applications.

This work showed that the RF algorithm had the best performance among eight algorithms, even considering the presence of newer algorithms, such as GBM and XGBoost. It also reported for the first time a set of 19 features extracted from system calls that when used could keep a close performance to the original set with 133 features. It is noteworthy that this analysis took into account algorithms with optimised hyperparameters. Furthermore the decision threshold was also tuned to strive for a balance between high precision and high recall, i.e., detecting most botnet applications without a high FP value.

Figure 5 Heatmap of importance factor using *IG* of the features selected using a threshold of 0.5 by botnet family



The proposed approach is robust to different types of botnet families, but to maintain performance in the real world, the classifier will need to be rebuilt when a significant decrease in performance is noted. This may occur with the creation of very different botnets, which, for example, might use new network protocols for communication or exploit new services. To remove this need, it is also possible to use an algorithm capable of being updated incrementally.

Finally, the used features have a very low abstraction level and can capture core behaviours shared by all different kinds of botnets, having a high generalisation capability. These behaviours include: gathering data about the device and its owner, sending data, receiving commands from the C&C infrastructure, and monitoring time to decide when to perform malicious actions without the owner of the device noticing.

9 Conclusions and future work

In this work, a host-based approach to detect mobile botnets was presented. By analysing the system calls the mobile applications invoked during a 1s time-window and using induced ML models, the approach achieved high performance across different metrics. Another important point to highlight is that reducing the dimensionality of the problem, from 133 to 19 features, did not have a significant negative impact on performance. Also, this granted greater interpretability of the problem and higher abstraction. Lastly, an insight of the best performing features was given in detail, which can be used by future mobile botnet detection works.

As future work, we will apply data stream mining techniques, in which there are new challenges, e.g., memory and computational time limitations, concept drifts, and novel situations, to identify mobile botnets in real-time. Also, in short to medium term, we intend to generate an even more diverse scenario, containing more legitimate and mobile botnet applications and using multiple mobile devices.

Acknowledgements

This work was partially supported by the National Funding from the FCT – Fundação para a Ciência e a Tecnologia through the UID/EEA/50008/2019 project; by Finep, with resources from Funttel, Grant No. 01.14.0231.00, under the Centro de Referênda em Radiocomunicações – CRR project of the Instituto Nacional de Telecomunicações (Inatel), Brazil; by Finatel through the Inatel Smart Campus project; by Brazilian National Council for Research and Development (CNPq) via Grant No. 309335/2017-5; and by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

References

Abdul Kadir, A.F., Stakhanova, N. and Ghorbani, A.A. (2015) *Android Botnets: What URLs are Telling Us*, pp.78–91, Springer International Publishing, Cham [online] http://dx.doi.org/ 10.1007/ 978-3-319-25645-0{_}6.

Alparslan, E., Karahoca, A. and Karahoc, D. (2012) 'BotNet detection: enhancing analysis by using data mining techniques', in *Advances in Data Mining Knowledge Discovery and Applications*, InTech, Chapter 17, [online] https://doi.org/10.5772/48804.

Android Studio Development Team (2017) *Logcat Command-Line Tool – Android Studio* [online] https://developer.android. com/studio/commandline/logcat.html?hl=en (accessed 8 February 2017).

Ariyapala, K., Do, H.G., Anh, H.N., Ng, W.K. and Conti, M. (2016) 'A host and network based intrusion detection for android smartphones', *Proceedings – IEEE 30th International Conference on Advanced Information Networking and Applications Workshops*, WAINA, pp.849–854.

Bergstra, J. and Bengio, Y. (2012) 'Random search for hyper-parameter optimization', *J. Mach. Learn. Res.*, Vol. 13, No. 1, pp.281–305 [online] http://dl.acm.org/citation.cfm?id=2503308.2188395.

Bishop, C.M. (2013) *Pattern Recognition and Machine Learning*, Springer, New York.

Breiman, L. (2001) 'Random forests', *Machine Learning*, Vol. 45, No. 1, pp.5–32.

Breiman, L., Friedman, J., Stone, C. and Olshen, R. (1984) *Classification and Regression Trees*, Wadsworth, Monterey.

Burguera, I., Zurutuza, U. and Nadjm-Tehrani, S. (2011) '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*, p.15.

Chen, T. and Guestrin, C. (2016) 'XGBoost: a scalable tree boosting system', *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

Chen, Z., Yan, Q., Han, H., Wang, S., Peng, L., Wang, L. and Yang, B. (2017) 'Machine learning based mobile malware detection using highly imbalanced network traffic', *Information Sciences*, Vol.433–434, pp.346–364.

Cortes, C. and Vapnik, V. (1995) 'Support-vector networks', *Machine Learning*, Vol. 20, No. 3, pp.273–297.

Cover, T. and Hart, P. (2006) 'Nearest neighbor pattern classification', *IEEE Trans. Inf. Theor.*, Vol. 13, No. 1, pp.21–27 [online] http://dx.doi.org/10.1109/TIT.1967.1053964.

da Costa, V.G.T., Barbon, S., Miani, R.S., Rodrigues, J.J.P.C. and Zarpelao, B.B. (2017) 'Detecting mobile botnets through machine learning and system calls analysis', in *2017 IEEE International Conference on Communications (ICC)*, pp.1–6.

Elzen, I.v.d.E. and Heugten, J.v.H. (2017) *MSc System and Network Engineering Techniques for Detecting Compromised IoT Devices*, Tech. Rep., University of Amsterdam.

Feizollah, A., Anuar, N.B., Salleh, R. and Wahab, A.W.A. (2015) 'A review on feature selection in mobile malware detection', *Digital Investigation*, Vol. 13, pp.22–37.

Friedman, J.H. (2001) 'Greedy function approximation: a gradient boosting machine', *The Annals of Statistics*, Vol. 29, No. 5, pp.1189–1232.

Garg, S., Singh, A.K., Sarje, A.K. and Peddoju, S.K. (2013) 'Behaviour analysis of machine learning algorithms for detecting P2P botnets', *2013 15th International Conference on Advanced Computing Technologies (ICACT)*, pp.1–4.

Geiri, D. and Shah, M.A. (2016) 'An enhanced botnet detection technique for mobile devices using log analysis', *The 22nd International Conference on Automation and Computing*.

Hearst, M.A., Dumais, S.T., Osuna, E., Platt, J. and Scholkopf, B. (1998) 'Support vector machines', *IEEE Intelligent Systems and their Applications*, Vol. 13, No. 4, pp.18–28.

Jiang, L., Cai, Z., Wang, D. and Jiang, S. (2007) 'Survey of improving K-nearest-neighbor for classification', *Proceedings – Fourth International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2007*, Vol. 1, pp.679–683.

Karim, A., Salleh, R., Khan, M.K., Siddiqa, A. and Choo, K-K.R. (2016) 'On the analysis and detection of mobile botnet', *Journal of Universal Computer Science*, Vol. 22, No. 4, pp.567–588.

Kerrisk, M. (2016) *The Linux Man-Pages Project* [online] https://www.kernel.org/doc/man-pages/ (accessed 18 January 2017).

Lewis, D.D. (1998) *Naive (Bayes) at Forty: the Independence Assumption in Information Retrieval*, pp.4–15, Springer Berlin Heidelberg, Berlin, Heidelberg [online] https://doi.org/10.1007/BFb0026666.

Lin, K-C., Chen, S-Y. and Hung, J.C. (2014) 'Botnet detection using support vector machines with artificial fish swarm algorithm', *Journal of Applied Mathematics*, No. 1.

Livadas, C., Walsh, R., Lapsley, D. and Strayer, W.T. (2006) 'Using machine learning techniques to identify botnet traffic', *Proceedings – Conference on Local Computer Networks, LCN*,
No. 1, pp.967–974.

Loh, W. (2008) 'Classification and regression tree methods', *Encyclopedia of Statistics in Quality and Reliability*, Vol. 1, pp.315–323 [online] http://onlinelibrary.wiley.com/doi/10.1002/9780470061572.eqr492/full.

Mahmoud, M., Nir, M. and Matrawy, A. (2015) 'A Survey on botnet architectures, detection and defences', *International Journal of Network Security*, Vol. 17, No. 3, pp.272–289.

Mas'ud, M.Z., Sahib, S., Abdollah, M.F., Selamat, S.R. and Yusof, R. (2014) 'Analysis of features selection and machine learning classifier in Android malware detection', in *2014 International Conference on Information Science Applications (ICISA)*, pp.1–5.

Narudin, F.A., Feizollah, A., Anuar, N.B. and Gani, A. (2016) 'Evaluation of machine learning classifiers for mobile malware detection', *Soft Computing*, Vol. 20, No. 1, pp.343–357 [online] https://doi.org/10.1007/ s00500-014-1511-6.

Natekin, A. and Knoll, A. (2013) 'Gradient boosting machines, a tutorial', *Frontiers in Neurorobotics*, Vol. 7, No. 1.

Patil, T.R. (2013) 'Performance analysis of Naive Bayes and J48 classification algorithm for data classification', *International Journal of Computer Science and Applications*, Vol. 6, No. 2, pp.256–261, ISSN: 0974-1011.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E. (2011) 'Scikit-learn: machine learning in Python', *Journal of Machine Learning Research*, Vol. 12, No. 1, pp.2825–2830.

Ruck, D.W., Rogers, S.K., Kabrisky, M., Oxley, M.E. and Suter, B.W. (1990) 'Letters: the multilayer perceptron as an approximation to a Bayes optimal discriminant function', *IEEE Transactions on Neural Networks*, Vol. 1, No. 4, pp.296–298.

Saad, S., Traore, I., Ghorbani, A., Sayed, B., Zhao, D., Lu, W., Felix, J. and Hakimian, P. (2011) 'Detecting P2P botnets through network behavior analysis and machine learning', *2011 9th Annual International Conference on Privacy, Security and Trust, PST*, pp.174–180.

Sakib, M.N. and Huang, C.T. (2016) 'Using anomaly detection based techniques to detect HTTP-based botnet C&C traffic', in *2016 IEEE International Conference on Communications (ICC)*, pp.1–6.

Sarle, W.S. (1994) 'Neural networks and statistical models', *Proceedings – Fourth International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2007*, pp.1–13.

Silva, S.S.C., Silva, R.M.P., Pinto, R.C.G. and Salles, R.M. (2013) 'Botnets: a survey', *Computer Networks*, Vol. 57, No. 2, pp.378–403.

Singh, K., Guntuku, S.C., Thakur, A. and Hota, C. (2014) 'Big data analytics framework for peer-to-peer botnet detection using random forests', *Information Sciences*, Vol. 278, pp.488–497 [online] http://dx.doi.org/10.1016/j.ins.2014.03.066.

Sokolova, M. and Lapalme, G. (2009) 'A systematic analysis of performance measures for classification tasks', *Information Processing and Management*, Vol. 45, No. 4, pp.427–437 [online] http://dx.doi.org/10.1016/j-ipm.2009.03.002.

Stevanovic, M. and Pedersen, J.M. (2014) 'An efficient flow-based botnet detection using supervised machine learning', *2014 International Conference on Computing, Networking and Communications (ICNC)*, pp.797–801.

Yuan, Z., Lu, Y. and Xue, Y. (2016) 'DroidDetector: android malware characterization and detection using deep learning', *Tsinghua Science and Technology*, Vol. 21, No. 1, pp.114–123.