2021 International Conference on COMmunication Systems & NETworkS (COMSNETS) | 978-1-7281-9127-0/20/$31.00 ©2021 IEEE | DOI: 10.1109/COMSNETS51098.2021.9352943

Improving ML Detection of IoT Botnets using Comprehensive Data and Feature Sets

Misha Mehra[§]
Computer Science and Engineering
Indian Institute of Technology Delhi
Delhi, India
mishamehra@gmail.com

Jay N. Paranjape[§]
Computer Science and Engineering
Indian Institute of Technology Delhi
Delhi, India
jay.edutech@gmail.com

Vinay J. Ribeiro
Computer Science and Engineering
Indian Institute of Technology Bombay
Mumbai, India
vinayr@iitb.ac.in

***Abstract*—In recent times, the world has seen a tremendous increase in the number of attacks on IoT devices. A majority of these attacks have been botnet attacks, where an army of compromised IoT devices is used to launch DDoS attacks on targeted systems. In this paper, we study how the choice of a dataset and the extracted features determine the performance of a Machine Learning model, given the task of classifying Linux Binaries (ELFs) as being benign or malicious. Our work focuses on Linux systems since embedded Linux is the more popular choice for building today's IoT devices and systems. We propose using 4 different types of files as the dataset for any ML model. These include system files, IoT application files, IoT botnet files and general malware files. Further, we propose using static, dynamic as well as network features to do the classification task. We show that existing methods leave out one or the other features, or file types and hence, our model outperforms them in terms of accuracy in detecting these files. While enhancing the dataset adds to the robustness of a model, utilizing all 3 types of features decreases the false positive and false negative rates non-trivially. We employ an exhaustive scenario based method for evaluating a ML model and show the importance of including each of the proposed files in a dataset. We also analyze the features and try to explain their importance for a model, using observed trends in different benign and malicious files. We perform feature extraction using the open source Limon sandbox, which prior to this work has been tested only on Ubuntu 14. We installed and configured it for Ubuntu 18, the documentation of which has been shared on Github.**

***Index Terms*—IoT Botnet, IoT Security, Machine Learning, Malware Analysis, Sandboxing**

I. Introduction

In recent years, Internet of Things (IoT) devices have proliferated. According to one estimate, by the year 2025 there will be around 74.44 billion IoT smart devices running with the use of 5G technology [1]. But what is inevitable with such a rise in the IoT technology is the ever-increasing attacks on these devices. The world has seen infamous attacks [2]–[4] where IoT devices were used to launch DDoS attacks on web servers of companies like Krebs, OVH, Twitter, Github, Netflix, Reddit among others. IoT devices like webcams, routers, DVRs, modems, IP cameras are easy targets because they are always connected to the Internet and are often poorly secured. It is very common for these devices to have no anti-virus installed [5] or to be configured with the default username and password with many opened ports. There is almost no provision of maintenance from vendor or manufacturer for these devices once they are installed at different locations [6].

A. IoT and IoT devices

According to [7], IoT is a network that connects uniquely identifiable "Things" to the Internet. "Things" represent physical objects which are always connected to the Internet and capable of sensing/actuating and performing various tasks without requiring manual intervention. These objects are often referred to as IoT devices.
The list of such devices is endless, ranging from industrial equipment and medical appliances to smart phones and watches, smart refrigerators, fitness trackers, smart TVs and so on. In our paper, we have considered malware capable of executing on IoT devices like IP cameras, Wi-Fi access points, internet-connected televisions, cable set-top boxes, DVRs, VoIP devices and media/video servers. These IoT devices are typically embedded systems running some customised version of the Linux operating systems with x86 architecture and are capable of transmitting and receiving information from other devices. [8].

B. IoT Botnet

A typical botnet is an army of compromised machines which works together to launch a targeted attack. These bots, or compromised machines are controlled by a Command & Control (C&C) server and execute commands at its behest. A widely reported form of botnet attack [2], [3], [9] on an IoT device is the DDoS attack where an army of infected IoT devices or IoT bots work in unison to target a victim system, thereby consuming its resources exhaustively and disrupting its services. But nowadays, botnet attacks on IoT devices are getting more sophisticated and diversified. An IoT botnet attack is no longer restricted to DDoS, but ranges from exploiting IoT devices for crypto mining [10], ransomware and SSH brute force [11], to denial of service and fraud [12]. We are likely to see more notorious forms of such attacks in the near future. A majority of these attacks in the past few years have been focused on Linux based IoT systems. According to a report by NSFOCUS [13], in 2019, 60% of the IoT botnets have been aimed at Linux based devices.

[§]Joint first-authors

C. Life Cycle of an IoT Botnet

A typical IoT botnet life cycle is primarily governed by the following stages [2], [4], [9]:

- Stage 1 - *Scanning and System Exploitation* - This is the initial stage where a bot tries to scan multiple IP addresses and look for open ports. It tries to connect to multiple devices via SSH or Telnet using default credentials. If found successful, the device is ready for infection.
- Stage 2 - *Malware Executable Download* - A loader server checks for infected devices and accordingly, drops and installs appropriate malware executables onto the infected machine.
- Stage 3 - *Communication with C&C Server* - The infected device now connects to its C&C server and receives commands for further action.
- Stage 4 - *Persistence and Covering up the Tracks* - A malware binary typically tries to delete the downloaded files and renames itself to some legitimate program to avoid evasion. It also tries to make itself persistent by adding itself to crontab so as to remain alive even when the system reboots. It tries to delete other infected files, if present and disables ports so as to avoid repetitive scanning.
- Stage 5 - *Lateral Movement* - The infected device tries to further scan more IP addresses so as to create a swarm of more infected devices, while receiving commands from its C&C server.

D. Motivation

Present day IoT devices are no longer low-end. They are equipped with high speed processors, RAMs in GBs and even powerful GPUs [14]. With such a setup, it becomes possible to run on IoT devices, executables which could formerly be run on high-end devices only. Hence, it is highly probable that malware meant for high end Linux systems can be modified to attack IoT devices. Also, most of the IoT devices are configured with default username/password, opened ports and services. Such improper controls can easily be exploited to execute botnet attacks.

Furthermore, a botnet is characterised by its system and network behaviour which in turn is captured using static, dynamic and network analysis. In this study, we answer questions like whether a model trained on a combined featureset i.e static, dynamic and network features taken together performs better. In addition, we also explore why and how training models on different types of files, affects the performance of a model. For example, does there exist a possibility of system files getting marked as malware if an ML based security system is not trained on a similar type of files? These questions have not been studied adequately in earlier works and so we address them in this paper.

E. Contributions

1) *Impact of Dataset on Performance:* We build a Random Forest Classifier for the classification task considering all possible types of Linux binaries (ELFs) as our dataset. It comprises of system files, IoT specific application files like drivers, IoT botnet files and general Linux malware files. We employ 3 types of features, namely static, dynamic and network. We show that with this combination of data and features, our model outperforms existing models since they leave out one or more of the data or the features we use.
2) *Feature Set Analysis:* We show non-trivial reductions in the rates of false positives and false negatives by using all the three types of features. We try to explain this behaviour of the model by analyzing the different features and observing how they vary with different types of files. We also propose an exhaustive scenario based approach that should be used instead of simply looking at the F1 score, while testing Machine Learning models for IoT security.
3) *Limon Upgrade:* Finally, as one of our contributions, we have also installed and configured the open source Limon sandbox [14] to run on Ubuntu 18 from Ubuntu 14. The necessary documentation is available on Github [15].

II. Related Work

Yin et al. explore IoT attacks on various architectures like MIPS, ARM and PowerPC [5]. They create a novel honeypot named IoTPOT to showcase different attacks existing on Telnet based IoT devices. These attacks are mainly based on DDoS attacks. The dataset created by the honeypot is available for usage as the IoTPOT dataset. A report from Infosec institute [16] analyzes the static features of a firmware to give insight into the behaviour of malicious Executable and Linkable Format (ELF) files. Cozzi et al. create a novel sandbox called Padawan for extracting detailed reports for ELF files by extracting static, dynamic and some network features for a variety of architectures [17]. They further document the features for different types of malware files, giving insights about the behaviours of Linux malware. Phu et al. also create a novel sandbox called F-sandbox to extract static and dynamic features of MIPS based ELF files [18]. This paper further takes malicious samples of LightAidra, Mirai, Gafgyt and Dofloo and about 230 benign files from embedded Linux Files and MIPS basic files. Then, they use machine learning algorithms like SVM, Naive Bayes and Random Forests to classify, thus getting an F1 score of 97%. However, they do not focus on the network features which may be useful for accuracy as well as robustness. Further, there is no analysis provided on whether using only four types of botnet families in the dataset is sufficient for the model to correctly classify a malware from a different family.

K. A. Asmitha and P. Vinod use open source tools like *strace* to extract static features of ELF files and use them for classification [19], while Koroniotis et al. use network features to classify traffic into malicious or benign [20]. However, both these approaches completely ignore the possibility of creating a combined feature set for better results. Moreover, these do not consider Linux malware or IoT specific application

files in their dataset, which are instrumental for decreasing the rate of false positives and false negatives. Nguyen et al. introduce a new feature for classification and showcase a performance improvement on a big dataset of benign and malicious files [21]. However, this paper does not take into account the modern system files of Ubuntu like systems files. Further, it does not focus on any analysis for zero day attacks. Nguyen et al. apply a CNN based classifier without showing considerable improvement in performance on the PSI graph dataset [22]. Phu et al. propose a new feature selection method based on the intermediate representation of malware called CFDVex [23]. They train a SVM classifier on their dataset and showcase a high accuracy for single architectures. Further, they are able to detect MIPS based malware with a model trained on Intel 80386 samples. However, they do not explore different Machine Learning Models and they do not provide explainability of their models or features. Su et al. propose a lightweight architecture where they first generate grayscale images from the file binaries and train a Convolutional Neural Net on them [24]. Their dataset consists of malware files from IoTPOT [5] and benign files from Ubuntu 16.04 system files, but is lacking in IoT specific files like drivers and application files. As far as we know, the present literature does not include a combination of the static, dynamic and network features associated with an executable for classification problems. Our system considers all the three aspects together - static,dynamic and network analysis. Also, benign files [24] [23] are generally considered to be system files of a typical Linux system only. However, we show that this is not sufficient to classify an actual IoT application. We create a new benign dataset comprising of system files as well as files gathered from IoT applications (like functions and drivers) from Github and demonstrate how previous approaches fail to identify them as benign.

III. Proposed System Architecture

The system architecture for our pipeline is shown in figure 1. The system takes an ELF file as input (benign and malicious), which is given to the Limon Sandbox for static, dynamic and network analysis. It generates analysis reports from which we extract our features. A machine learning classifier which is trained offline using supervised algorithms with the training feature set created using the same process, is used to classify the file as benign or malicious.

A. Sample Collection

We collect 4 types of files in our dataset. We term them as System Files, IoT benign files, IoT Botnet Files and Linux Malware files. The number of the samples for each type are shown in Table I. For system files, we collect benign ELFs from system directories of Ubuntu 18.04 operating system like /bin,/sbin and /usr/bin. We gather another set of benign ELFs representing applications specific to IoT devices for the second group. This set represents the IoT Benign Files in Table I. These are device drivers, functions and test files taken from various Github repositories targeting deployment on actual IoT devices. These include:

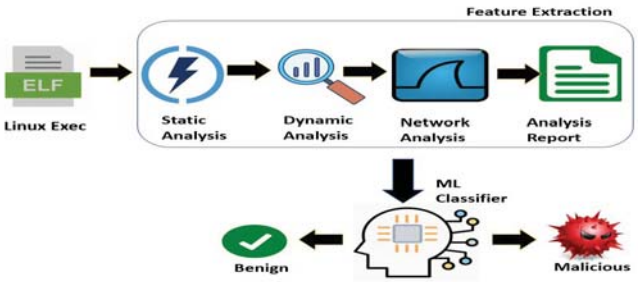

Fig. 1. Proposed System Architecture

- Test functions on a home based IoT device with IR-RC function and environment sensors [25]
- Library functions and device drivers [26]
- Modules related to audio, LED, sensors [27]
- Cloud IoT device Software Development Kit by Google for connecting low-end IoT devices to Google Cloud IoT Core [28]
- Amazon Web Services Client Library for Python for IoT devices [29]

For IoT botnet files, we consider prominent IoT botnets like Aidra, Amnesia, Bashlite, Dofloo, Gafgyt, Mirai and Tsunami. Mirai [3] had infected hundreds of thousands of IoT devices within few hours of its propagation and launched DDoS attacks on high profile targets [2]. Aidra [30] had launched telnet based attacks from IoT devices like WiFi access points, internet-connected televisions, cable set-top boxes, DVRs, VoIP devices, IP cameras, and media centers. Amnesia, which is a variant of the Tsunami botnet, targets an unpatched remote code execution vulnerability in DVR devices [31]. Bashlite, also known as Gafgyt, Lizkebab, Qbot, Torlus and LizardStresser, exploited vulnerability in the bash shell, popularly known as shell shock, and targeted IoT devices and linux servers. Dofloo, also known as AESDDoS is a popular botnet malware capable of launching large scale DDoS attacks and loading cryptocurrency miners to the infected machines [32].

We collect samples of such malicious ELFs from three online virus repositories namely VirusTotal [33], Virusshare [34] and Contagio Malware Dump [35]. This consists of 232 files which are found to be belonging to IoT botnet families by at least four antivirus engines available on VirusTotal. We call this group as IoT botnet Files. In addition, we also use files from the IoTPOT dataset [5], since it is used by many research papers and hence it becomes easier to compare with existing models.

For the purpose of testing on general Linux Malware and candidate zero day attacks, we collect another set of ELFs which we call as Linux Malware Files. These are the newer

TABLE I
SAMPLE COLLECTION

Type	Category	Total number
Malicious	IoT Botnet Files	797(VirusTotal,Virusshare, IoTPOT)
	Linux Malware Files	195
Benign	System Files	999
	IoT Benign Files	128

Linux malware collected between the year 2017, 2018 and 2019 from online malware repositories [33]. These Linux ELFs are not classified as belonging to any of the above mentioned families by the anti-virus engines on VirusTotal. However, they are capable of working as botnets by launching large scale DDoS attacks, ssh brute force attacks, performing crypto-mining operations and so on. We observe that these files have similar behaviour as those of the newer IoT botnets that are surfacing but are not yet as popular. Malware attackers use different techniques to launch more sophisticated attacks on IoT devices and a common technique is using different variants of existing Linux malware to exploit vulnerabilities in IoT devices [36] [37]. Hence, we consider these Linux ELF files as potential sources of zero day attacks (attacks which are not classified by present anti virus engines as IoT malware) and include them in our dataset as a separate group. We also empirically show through our experiments that IoT botnets have similar characteristics with Linux malware and a classifier trained on both(IoT botnet and Linux malware) is capable of detecting IoT botnet attacks more efficiently. Further details are covered in section IV

B. Sandbox for Analysing Malware Behaviour

Execution of malware in an isolated environment, better known as sandboxing [38] is a very popular technique for analysing malware behaviour. Sandboxes provide a virtual environment for a file to run without affecting the host machine. Hence, they are used to capture system calls, processes spawned, file modifications and network connections for the better understanding of malware behaviour. For analysing Windows malware, many commercial [39] [40] [41] [42] [43] and open-source sandboxes [44] are available. But analysis of Linux malware is still a challenging area for researchers, mainly because of the scarcity of sandboxes on latest versions of Linux. There are online sandboxes such as Padawan [17] and F-sandbox [18] for different architectures. However, Padawan does not provide pcaps to its users for network analysis, and F-sandbox only provides analysis for the MIPS architecture.

For our experiments, we have used an open-source sandbox named Limon Sandbox [14] for analysing Linux binaries. Limon uses open-source tools to perform static, dynamic and memory analysis of malware. It determines process activity, file system interaction, network connections, memory analysis in a controlled environment. It uses the Linux binary utility *readelf* [45] to perform static analysis of a file. *Strace* is used for capturing dynamic behaviour of malware like process system calls and file system information. Internet Services Simulation Suite (*InetSim*) [46] and *SysDig* [47] are used for monitoring network behaviour.

Limon Sandbox was tested successfully for Ubuntu 14.04 at the time of its development. We have installed and reconfigured it on Ubuntu 18.04 LTS with all the latest supporting open source tools required for complete execution of Linux binaries and with support for memory forensics using the volatility framework. For this, we have created a profile for Ubuntu 18.04 for Linux kernel version 4.15.0-96-generic. The documentation for installation and configuration is available on Github [15]. In our setup, Linux binaries are executed for 60 seconds after which the analysis reports are generated.

C. Proposed Feature Set

Prominent features have been extracted from system (static and dynamic) and network analysis of Limon sandbox reports. We have extracted 63 numerical features from Limon reports, useful in the classification of Linux binaries as benign or malicious. These features have been broadly divided into three categories namely:

- Static Features based on the ELF file format and information stored in symbol table.
- Dynamic Features based on the system calls used by the binary on execution.
- Network Features based on network communication observed during analysis.

Our entire featureset has been listed in table II.

D. Machine Learning Models

We use the feature set as mentioned above to train Machine Learning Algorithms. We use the Logistic Regression Model, the Support Vector Machine (SVM), the K-Nearest Neighbour (KNN) Classifier and the Random Forest (RF) for our experiments. We divide the dataset into various sections to conduct experiments as explained in section IV and come up with number of conclusions as explained in Section V. We do not use Deep Learning Models since the size of the dataset is not suitable for a deep neural network.

IV. EXPERIMENT AND RESULTS

We follow a scenario-based approach for training and testing our model. There are 4 different kinds of data items in the dataset as discussed above. To show the importance of each set for a machine learning model, we first divide our dataset into the following six categories:

- T1 - 100% samples from the system files and 100% samples from the IoT botnet files
- T2 - 20% samples from system files and all 100% samples from the Linux malware files
- T3 - 100% samples from the IoT benign files and 15% samples from the IoT botnet files
- T4 - 100% samples from the IoT benign files and 100% samples from the Linux malware files

TABLE II
FEATURE SET

Feature Type (Number)	List of Features
Static Features (36)	Number of Program Headers, Number of Section Headers, Section header string table index, Number and Size of Functions, Number and Size of Files, Number and Size of Local Functions, Number and Size of Global Functions, Number and Size of Weak Functions, Number and Size of Hidden Functions, Number and Size of Local Objects, Number and Size of Global Objects, Number and Size of Weak Objects, Number and Size of Hidden Objects, Number and Size of Local Files, Number and Size of Global Files, Number and Size of Weak Files, Number and Size of Hidden Files, File Size, Size of .text Section, Size of .data Section, Size of .rodata Section, Size of .bss Section
Dynamic Features (13)	Number of Calls to 'EXECVE', 'OPEN', 'CLOSE', 'READ', 'WRITE', 'CONNECT', 'SOCKET', 'FORK', 'CHDIR', 'CLONE', 'READONLY', 'READWRITE', Number of Re-Transmissions
Network Features (14)	Total Number of Destination IPs, Total Number of Packets, Ratio of Total Number of Packets and Time of Last Packet, Fraction of 'ARP', 'DNS', 'ICMP', 'MDNS', 'NBNS', 'TCP', 'BROWSER' Packets Exchanged, Mean and Median of Inter-Arrival Time of Packets, Mean and Median of Total Size of Packets Exchanged

- T5 - 70% samples from system files, 80% samples from the IoT benign files, 80% samples from IoT botnet files and 80% samples from the Linux malware files.
- T6 - The remaining files from the dataset which were not included in T5. Note that T5 and T6 are mutually exclusive sets. This set is used exclusively for testing.

Notice that each of the sets from T1 to T4 contains only 2 types of files from the total 4 types mentioned in Table II. Out of these, one type represents benign data(positive class) and the other type represents malicious data(negative class). For example, T1 contains only system files and IoT Botnet files. It does not contain IoT benign files nor Linux Malware files. System files are given the positive label and IoT Botnet files are given the negative label. Hence, when we train a model on Ti (where i is in 1,2,3,4), it represents that the model has only seen the two types of files in Ti and when tested on all 4 types of files, gives valuable observations on those files which are not in Ti. T5 and T6 represent a 80-20 split on the entire dataset. Thus, when a model is trained on T5(with benign files having positive labels and malicious files having negative labels), it has seen all the 4 types of files and hence, it performs well on all of them during testing. Note that T5 and T6 are mutually exclusive and so T6 acts as a test set for a model trained on T5. Thus, splitting of the dataset into 6 sets is done to empirically show the importance of each type of file during the training of a Machine Learning Model.

TABLE III
BEST ACCURACY FOR EACH DATASET FOR EACH SCENARIO

Dataset	S1	S2	S3	S4	S5
System Files	100%	99.89%	**0.2%**	**0%**	100%
IoT Benign Files	**5%**	**0%**	100%	100%	100%
IoT Botnet Files	100%	98.27%	100%	87%	100%
Linux Malware Files	**94%**	100%	**98%**	100%	99%

We use the term *Scenario* i, or Si for denoting the experiment when we train a Machine Learning model on Ti. Thus, we have a total of 5 different scenarios, represented by S1, S2, S3, S4 and S5, when the model is trained on T1, T2, T3, T4 and T5 respectively. Out of the Machine Learning Models including SVM, Logistic Regression, KNN and Random Forest, we observe the best performance by Random Forest and so we only present results for it. The accuracy on each of the 4 types of files, for each Scenario, are shown in Table III. We use the Random Forest model from Scenario 5 for comparing with existing models in Table IV and show a better performance over all types of files. Each of the existing approaches leave out one or the other type of files in their training, leading to a poor performance for those files. Finally, we show the confusion matrix for Scenario 5 in table VI, when the model is tested on T6. This gives us an F1 score of 0.99. Further analysis on the results from each scenario is done in Section V.

We also examine the results of the model when trained individually on the system, dynamic and network features, to confirm that we indeed get a better performance by combining all the categories, by examining the tuple of percentage false positives(when malicious files are treated as benign) and percentage false negatives (when benign files are classified as malicious). We observe a significant decrease in the false positives, by at least 4% (from the static-only case) and at most 26% (from the network-only case) in the test set. We do not observe significant reduction in the amount of false negatives in the static-only case, since it itself is close to 0 in the first place. However, we observe a maximum decrease of 19% in the network-only case, in the test set. These observations are tabulated in Table V.

We provide a detailed analysis about the importance and results of each scenario, trends followed by major features and insights to the malicious behaviour of IoT botnets observed during analysis in the next section.

V. ANALYSIS

A. Analysis of Scenarios

We make various inferences from our experiments as explained in section IV. They are as follows:

*1) **System Files V.S. IoT Benign Files**:* Training on Ubuntu system files is not sufficient for building a robust classifier since such a model is likely to misclassify IoT benign files. This can be inferred from the first and second columns

TABLE IV
COMPARING % ACCURACY SCORES FOR DIFFERENT APPROACHES ON ALL FILE TYPES

Model	System	IoT Benign	IoT Botnet	Linux Malware
K.A. Asmitha and P. Vinod [19]	97	3.4	97	95.2
Koroniotis et al. [20]	97.2	6.8	91	56.78
Nguyen et al. [21]	0.5	100	100	97.4
Our model	**99.89**	**100**	**100**	**99.50**

TABLE V
TUPLE OF % FALSE POSITIVES, %FALSE NEGATIVES FOR INDIVIDUAL FEATURE TYPES

Features	T5(Training Set)	T6(Testing Set)
Static features only	(0.6,0)	(3.75,0.12)
Dynamic features only	(0.8,0.33)	(4,1.3)
Network Features only	(13.8,4.3)	(26.25,16.2)
Combined Featureset	**(0.3,0)**	**(0,0.12)**

of Table III, where the observed accuracy for a model trained on system files performs poorly (with an accuracy of only 5% and 0%) on IoT benign files. Recall that Scenario 1 and Scenario 2 only used the system files for training the model as benign samples. On the other hand, training only on IoT benign files is not sufficient as well since such a model is highly likely to misclassify Ubuntu system files. As seen in the columns 3 and 4 of Table III, the model classifies almost all system files as malicious (accuracy of 0.2% and 0% only). Recall that Scenario 3 and Scenario 4 use only IoT benign files as the benign dataset.

*2) **Performance on Linux Malware Files**:* A model trained on IoT Malware files as the malicious dataset performs well on linux malware files as seen in columns 1 and 3 in Table III. Note that an important factor of IoT botnets is persistent network attacks given their constant connectivity to the internet. This includes attacks like DDoS. The linux malware dataset that we have taken consists of malware which is not classified among the major IoT botnets by the many antivirus engines on Virustotal and at the same time, their functionality of carrying out such large scale network attacks is quite similar to the known IoT botnets today. Hence, it is highly probable that new attacks in the future will be based on such types of files. However, we show that our model, trained on IoT botnet files is able to classify such files with a high accuracy. One more implication of this observation is that our model gives a more generalized system which is capable of detecting not only IoT botnets, but also other Linux malware. We try to explain this phenomenon through the feature space, in the next subsection.

*3) **Combining All Types of Datasets**:* In the previous two subsections, we show that system files as well as IoT benign files are needed for proper classification. Similarly, we show that IoT botnets are good representatives of Linux malware files. Hence, we train our model with these three as the training set and obtain an accuracy of **96.41%** on the Linux malware files. However, we get a better result(99% as seen in TableIII) when a model is trained on some examples from each of the datasets. This final model achieves a high F1 score on the test data T6 as well, which means that there are very few false positives and false negatives given by the model. The confusion matrix for this model on T6 is tabulated in table VI. The corresponding F1 score is **0.998**.

*4) **Improvement over Existing Models**:* As seen in table IV, our model gives close to 100 perecnt accuracy in all 4 types of files. The work done by K.A. Asmitha and P. Vinod [19] does not include IoT specific application files and hence, performs poorly on these files from our dataset. Similarly, Nguyen et al. [21] do not consider system files and hence perform poorly from samples collected from /bin and /sbin in our dataset. Koroniotis et al. [20] use an SVM over certain IoT application files and IoT Botnet files while not using system files, again showing a poorer performance on these files in our dataset. Further, the SVM model also shows poor generalization when tested on general Linux malware. This is not observed for our model or the other models as well, showing that Random Forests have a greater tendency to correctly generalize to Linux malware after getting trained on IoT Botnet Files.

*5) **Improvement by Combining Feature-Set**:* As seen in table V, using a combined feature-set of static, dynamic and network features gives a non-trivial decrease in the rate of false positives and false negatives. Static features capture the information about the actual code of the file whereas dynamic features capture information about different system calls made during the execution of the file. Both of these are important for describing the behaviour of a file. Adding network features also gives us indirect, valuable insights to file behaviour. For example, a high number of destination IPs is an indicator of the file having a higher chance of being malicious, since it tries to scans multiple IPs for finding vulnerable machines. Further, the C&C servers of botnets have been observed to change their IP addresses to avoid getting caught. It is also due this reason that the number of packets being exchanged is comparatively more. Number of Domain Name Server (DNS) queries also plays an important role as botnets connect to DNS servers for resolving IP addresses.

*6) **Robustness**:* Lastly, we analyze the robustness of our model for Scenario 5. For this, we manually change the feature values of malicious files to match the benign files. This is done for 1 feature and 2 features at a time. This fails to confuse the model as there is almost no change in the metric values. This shows that the model is not making decisions based on only a few features, proving their collective usefulness as opposed to a single dominating feature.

B. Analysis of Feature Behaviour

We use the Random Forest algorithm to list down the prominent features from our dataset. We find out the what

TABLE VI
CONFUSION MATRIX FOR T6, F1 SCORE=0.998

RF	Predicted positives	Predicted negatives
True Positives	226	0
True Negatives	1	211

are the significant features and the reasons for their trends in malicious and benign files. This is explained as follows:

- *Number of Program Headers/Section Headers* - We observe reduced numbers of program headers and section headers in the binaries of malicious ELFs as shown in Figure 2.The primary reason is that malware authors use assembly language for writing malware and this assembly code is converted into a malware executable. During this process, segments and sections information are stripped, resulting in a smaller number of program segments and sections. Malware authors also use the *strip* utility, which is available as a part of GNU Binary Utilities suite of tools in most Unix/Linux based systems, to remove symbols and sections from object files. This is primarily done to reduce file size and makes reverse engineering difficult for a security analyst [48].
- *Values from Symbol Table* - Symbol Table values like number of functions, size of functions, number of objects and size of objects are found to be on the higher side in malicious files as shown in Figure 3. The reason for this can be argued based on the fact that malware authors create polymorphic malware by changing the program's appearance in order to evade detection from Anti-Virus Engine. To achieve polymorphism, they copy instructions, linking them with each other with *jump* instructions and filling the remaining space with dead or redundant code [49], obfuscating variable names, register reassignments, equivalent code replacements [50] and so on. While disassembling the mirai code, we noticed large number of instructions with no-operations and jump statements.
- *System Calls* - Number of system calls like connect, socket, read are found to be higher in malicious binaries as shown in Figure 4. We observed read system calls made to access various files during execution. The details of these files are explained in the next subsection. Connect and socket system calls were made for connecting to multiple IP Addresses across network, for querying DNS servers and for contacting malicious URLs and C&C servers.
- *Destination IPs* - Number of IP addresses in the destination address of data packets generated during network analysis is found to be higher in malicious binaries as shown in Figure 5. The reason for this is that IoT malware tries to connect to multiple IPs for the purpose of scanning, making DNS queries and for connecting to C&C servers. Also, the total number of packets sent and received are found to be higher in case of IoT botnets.

*1) **Benign IoT Files Versus System Files**:* - We observe non-trivial differences in some of the feature values of system files and IoT benign files. For example, the number of READS in system files are found to be much higher than IoT benign files as shown in Figure 2. Hence, a model trained with only the IoT files as benign data may classify the system files as malicious. Similarly, the number of objects and functions in IoT files are much higher than those of system files (Fig. 3) and a model trained with only system files as benign data may misclassify IoT benign files as malicious. Thus, both types of files are necessary for a Machine Learning model. On the other hand, the feature values of IoT malware and proposed zero-day malware show no exceptional differences in the feature values, which makes the model still classify them as malware. Yet, including these types of files in the training does improve performance by about 2-3%.

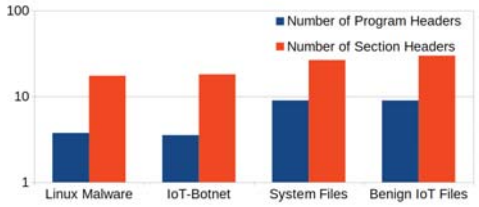

Fig. 2. Static Features based on ELF Header

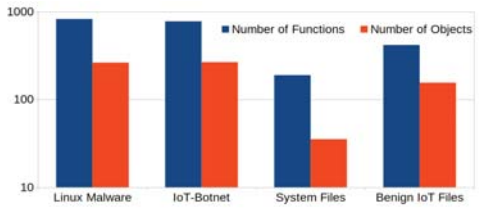

Fig. 3. Static Features based on Symbol Table

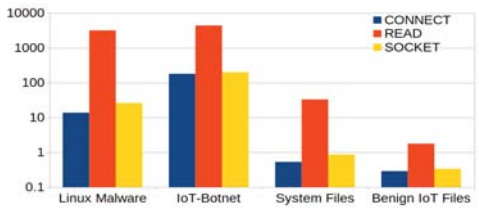

Fig. 4. Dynamic Features based on System Calls

C. Observations of Malicious Behaviour

In the malware analysis report of IoT botnets generated by Limon sandbox we observe following things which we find worth mentioning:

- *DNS Queries* - A lot of malware files read */etc/resolv.conf* - the resolver configuration file that is read by the resolver

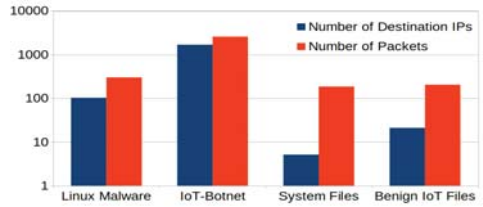

Fig. 5. Network Features

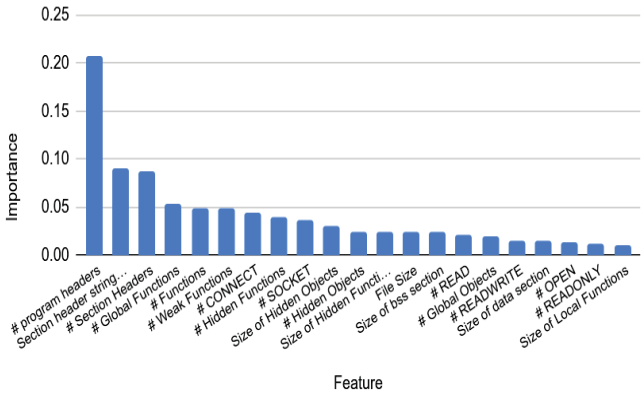

Fig. 6. Importance of features for Random Forest

routines, which in turn makes queries and interprets responses from the Internet Domain Name System (DNS).

- *Use of Packers* - Some of the samples of malware such as mirai and tsumani were obfuscated using Ultimate Packer for eXecutables (UPX). Section header table and symbol table information were missing in such samples. Such malware samples were unpacked using UPX unpacker option before analysis.
- *Frequently Accessed Files* - Lot of open and read system calls were made to files like */proc/net/tcp* (for reading active TCP connections), */proc/net/route* (for viewing kernel's IP routing table), */usr/dict/words* (for guessing passwords), *dev/watchdog*, *usr/lib/systemd/systemd-logind* (for managing user login information), */etc/passwd* (for reading usernames and other information) and */proc/net/dev* (containing network interface information).
- *Persistence* - In order to maintain persistence, IoT botnets try to copy their paths in */.bashrc*, */etc/init.d/boot.local*, */etc/inittab* and */usr/bin/crontab*. This is done so that the malware file is executed even if the system reboots or a user logs in again.
- *Network Connections* - We observed various telnet connections made to malicious IPs on port 23 and 2323. *HTTP GET* and *WGET* commands were used to connect to malicious URLs.
- *Frequently used Commands* - Linux commands for checking system information, changing directories, removing/copying malicious files, changing file permissions, finding Internet Service Provider (ISP) name and changing NVRAM variables were widely seen.
- *Files and Functions Name* - A lot of local files and function names starting with 'bot', 'bot ports', 'bot killer', 'malware', 'kill', 'udp attack' etc. were found in the symbol table.

D. Feature Importance for Random Forest

A random forest is a group of decision trees. Each of the decision trees is a classifier which at each step, selects a feature and a value randomly and splits the data points into two groups based on whether the feature value for a data point is greater or smaller than this selected value. The features which are used by more decision trees and which make purer splits, are given more importance than others. Pure splits are when a feature-value pair splits data points into two groups where one group contains majority of the data points of one class and the other group has a majority of data points of the other class.

Figure 6 shows some features in the decreasing order of their importance. We observe that when we consider only the top 28 features in the Random Forest model, the F1 score of the model does not reduce more than 1%.

VI. Conclusion

In this paper, we showed significant improvements in the robustness and the performance of Machine Learning techniques by employing a combined dataset of Linux system files, IoT application files, IoT Botnet files and Linux Malware files. We built a system trained on these files using a feature set including static, dynamic and network features, which shows an improvement over existing methods. We provide an exhaustive scenario-based method of analysis of machine learning models in this field. Using this method, we show that training only on system files or IoT application files is not enough today whereas training on IoT botnets can really help in the detection of zero day attacks, assuming they utilize existing Linux malware implementations, which is the historically observed case. Finally, we try to interpret our model by analysing feature values for each of the files and provide insights to the malicious behaviour of IoT botnet observed during the the analysis.

References

[1] Statista, "Internet of things -number of connected devices worldwide 2015-2025," 2020, accessed: 2020-06-01. [Online]. Available: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

[2] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, and M. Kallitsis, "Understanding the mirai botnet," in *26th {USENIX} Security Symposium ({USENIX} Security 17)*, Vancouver, BC, Canada, 2017, pp. 1093–1110.

[3] J. Margolis, T. T. Oh, S. Jadhav, Y. H. Kim, and J. N. Kim, "An in-depth analysis of the mirai botnet," in *2017 International Conference on Software Security and Assurance (ICSSA)*, Altoona, Pennsylvania, USA, 2017, pp. 6–12.

[4] A. Marzano, D. Alexander, O. L. H. M. Fonseca, E. C. Fazzion, C. Hoepers, K. Steding-Jessen, M. H. P. Chaves, Í. S. Cunha, D. O. Guedes, and W. Meira, "The evolution of bashlite and mirai iot botnets," in *2018 IEEE Symposium on Computers and Communications (ISCC)*, Natal, Brazil, 2018, pp. 00 813–00 818.

[5] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, "Iotpot: Analysing the rise of iot compromises," in *9th USENIX Workshop on Offensive Technologies (WOOT 15)*. Washington, D.C.: USENIX Association, Aug. 2015. [Online]. Available: https://www.usenix.org/conference/woot15/workshop-program/presentation/pa

[6] TECHWATCH, "Top 10 iot vulnerabilities," https://www.networkworld.com/article/3332032/top-10-iot-vulnerabilities.html, 2019, accessed: 2020-06-01.

[7] IEEE, "Ieee internet of things," https://IoT.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Revision1_27MAY15.pdf, 2015, accessed: 2020-06-01.

[8] R. Cohen and T. Wang, *Overview of Embedded Application Development for Intel Architecture*. Berkeley, CA: Apress, 2014, pp. 1–18. [Online]. Available: https://doi.org/10.1007/978-1-4842-0100-8_1

[9] C. Dietz, R. L. Castro, J. Steinberger, C. Wilczak, M. Antzek, A. Sperotto, and A. Pras, "Iot-botnet detection and isolation by access routers," in *2018 9th International Conference on the Network of the Future (NOF)*. Poznań, Poland: IEEE, 2018, pp. 88–95.

[10] CNBC, "Thousands of iot devices can be hacked to mine cryptocurrency," https://www.cnbc.com/2018/03/01/thousands-of-IoT-devices-can-be-hacked-to-mine-cryptocurrency-avast.html, 2018, accessed: 2020-06-01.

[11] Zdnet, "New kaiji malware targets iot devices via ssh brute force attacks," https://www.zdnet.com/article/new-kaiji-malware-targets-iot-devices-via-ssh-brute-force-attacks/, 2020, accessed: 2020-06-01.

[12] Radware, "Iot botnets on the rise," https://blog.radware.com/security/2018/10/iot-botnets-on-the-rise/, 2018, accessed: 2020-06-01.

[13] M. Hao, "Botnet trend report 2019-6," https://nsfocusglobal.com/botnet-trend-report-2019-6/, 2020, accessed: 2020-10-01.

[14] Monappa, "Setting up limon sandbox for analysing linux binaries," https://cysinfo.com/setting-up-limon-sandbox-for-analyzing-linux-malwares/, 2016, accessed: 2020-06-01.

[15] J. N. Paranjape, "Iot botnet analysis," 2020. [Online]. Available: https://github.com/JayParanjape/iot-Botnet-Analysis

[16] InfosecInstitute, "Firmware analysis for iot devices," https://resources.infosecinstitute.com/firmware-analysis-for-iot-devices/#gref, 2016, accessed: 2020-06-01.

[17] E. Cozzi, M. Graziano, Y. Fratantonio, and D. Balzarotti, "Understanding linux malware," in *2018 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, 2018, pp. 161–175.

[18] T. N. Phu, K. H. Dang, D. N. Quoc, N. T. Dai, and N. N. Binh, "A novel framework to classify malware in mips architecture-based iot devices," *Security and Communication Networks*, 2019. [Online]. Available: https://doi.org/10.1155/2019/4073940

[19] K. A. Asmitha and P. Vinod, "A machine learning approach for linux malware detection," in *2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*. Ghaziabad, India: IEEE, 2014, pp. 825–830.

[20] N. Koroniotis, N. Moustafa, E. Sitnikova, and J. Slay, "Towards developing network forensic mechanism for botnet activities in the iot based on machine learning techniques," in *International Conferenicce on Mobile Networks and Management*. Melbourne, VIC, Australia: Springer, 2017, pp. 30–44.

[21] N. Huy-Trung, N. Quoc-Dung, N. Doan-Hieu, and L. Van-Hoang, "Psi-rooted subgraph: A novel feature for iot botnet detection using classifier algorithms," *ICT Express*, vol. 6, no. 2, pp. 128 – 138, 2020.

[22] H. Nguyen, Q. Ngo, and V. Le, "Iot botnet detection approach based on psi graph and dgcnn classifier," in *2018 IEEE International Conference on Information Communication and Signal Processing (ICICSP)*. Singapore: IEEE, 2018, pp. 118–122.

[23] T. Phu, L. Hoang, N. Toan, N. Tho, and N. Binh, "Cfdvex: A novel feature extraction method for detecting cross-architecture iot malware," *SoICT 2019: Proceedings of the Tenth International Symposium on Information and Communication Technology*, pp. 248–254, 12 2019.

[24] J. Su, V. D. Vasconcellos, S. Prasad, S. Daniele, Y. Feng, and K. Sakurai, "Lightweight classification of iot malware based on image recognition," in *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2. Tokyo, Japan: IEEE, 2018.

[25] Opiopan, "Elflet," https://github.com/opiopan/elflet, 2020, accessed: 2020-06-01.

[26] Espressif, "Esp-iot-solution," https://github.com/espressif/esp-IoT-solution, 2019, accessed: 2020-06-01.

[27] RT-Thread, "Iot board," https://github.com/RT-Thread/IoT_Board, 2019, accessed: 2020-06-01.

[28] Google, "Iot device sdk embedded c," https://github.com/GoogleCloudPlatform/IoT-device-sdk-embedded-c, 2020, accessed: 2020-06-01.

[29] Amazon, "Aws iot elf," https://github.com/aws-samples/aws-IoT-elf, 2017, accessed: 2020-06-01.

[30] NJCCIC, "Aidra botnet njccic threat profile," 2016, accessed: 2020-06-01. [Online]. Available: https://www.cyber.nj.gov/threat-center/threat-profiles/botnet-variants/aidra-botnet

[31] NJCCIC, "Botnets njccic threat profile," https://www.cyber.nj.gov/threat-center/threat-profiles/botnet-variants, 2016, accessed: 2020-06-01.

[32] NJCCIC, "Dofloo botnet njccic threat profile," https://www.cyber.nj.gov/threat-center/threat-profiles/botnet-variants/dofloo, 2016, accessed: 2020-06-01.

[33] VirusTotal, "Virustotal," https://www.virustotal.com, 2004, accessed: 2020-06-01.

[34] VirusShare, "virusshare," https://www.virusshare.com, 2013, accessed: 2020-06-01.

[35] Contagiodump, "Contagio malware dump," https://contagiodump.blogspot.com, 2009, accessed: 2020-06-01.

[36] T. Micro, "Cryptocurrency mining malware targets linux systems uses rootkit for stealth," https://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/cryptocurrency-mining-malware-targets-linux-systems-uses-rootkit-for-stealth, 2018, accessed: 2020-06-01.

[37] Securityweek, "Ddos malware linux distributed ssh brute force attacks," https://www.securityweek.com/ddos-malware-linux-distributed-ssh-brute-force-attacks, 2015, accessed: 2020-06-01.

[38] M. Mehra and D. Pandey, "Event triggered malware: A new challenge to sandboxing," in *2015 Annual IEEE India Conference (INDICON)*. New Delhi, India: IEEE, 2015, pp. 1–6.

[39] T. Micro, "Advanced threat protection," https://www.trendmicro.com/en_in/business/products/network/advanced-threat-protection.html, 2020, accessed: 2020-06-01.

[40] F. Eye, "Malware analysis," https://www.fireeye.com/solutions/malware-analysis.html, 2020, accessed: 2020-06-01.

[41] Rarst.net, "Threatexpert.com – behavioral file analysis," https://www.rarst.net/web/threatexpert/, 2009, accessed: 2020-06-01.

[42] Comodo, "Comodo internet security," https://help.comodo.com/topic-72-1-451-4737-.html, 2020, accessed: 2020-06-01.

[43] J. Security, "Analyse malware in a depth previously not possible," https://www.joesecurity.org/, 2020, accessed: 2020-06-01.

[44] C. Guarnieri, "Cuckoo sandbox," https://cuckoosandbox.org/, 2019, accessed: 2020-06-01.

[45] Linux, "Linux readelf," https://linux.die.net/man/1/readelf, 2009, accessed: 2020-06-01.

[46] InetSim, "Inetsim," https://www.inetsim.org/packages.html, 2020, accessed: 2020-06-01.

[47] Tecmint, "Sysdig – a powerful system monitoring and troubleshooting tool for linux," https://www.tecmint.com/sysdig-system-monitoring-and-troubleshooting-tool-for-linux/, 2020, accessed: 2020-06-01.

[48] C. H. Malin, E. Casey, and J. M. Aquilina, *Malware Forensics: Investigating and Analyzing Malicious Code*. Syngress, 2008.

[49] P. Beaucamps, "Advanced polymorphic techniques," *International Journal of Computer Science*, vol. 2, no. 3, pp. 194–205, 2007.

[50] J. Drew, M. Hahsler, and T. Moore, "Polymorphic malware detection using sequence classification methods and ensembles," *EURASIP J. Inf. Secur.*, vol. 2017, no. 1, Dec. 2017. [Online]. Available: https://doi.org/10.1186/s13635-017-0055-6