Edge Computing for Real Time Botnet Propagation Detection

Mikhail Gromov, David Arnold, and Jafar Saniie

Embedded Computing and Signal Processing (ECAP) Research Laboratory (http://ecasp.ece.iit.edu/)
Department of Electrical and Computer Engineering
Illinois Institute of Technology, Chicago IL, U.S.A.

***Abstract*- Continued growth and adoption of the Internet of Things (IoT) has greatly increased the number of dispersed resources within both corporate and private networks. IoT devices benefit the user by providing more local access to computation and observation compared to dedicated servers within a centralized data center. However, years of lax or nonexistent cybersecurity standards leave IoT devices as easy prey for hackers looking for easy targets. Further, IoT devices normally operate at the edge of the network, far from sophisticated cyberattack detection and network monitoring tools. When hacked, IoT can be used as a launching point to attack more sensitive targets or can be collected into a larger botnet. These botnets are frequently utilized for targeted Distributed Denial of Service (DDoS) attacks against service providers and servers, decreasing response time or overwhelming the system. In order to protect these vulnerable resources, we propose an edge computing system for detecting active threats against local IoT devices. Our system will utilize deep learning, specifically a Convolutional Neural Network (CNN) for detecting attacks. Incoming network traffic will be converted into an image before beings supplied to the CNN for classification. The network will be trained using the N-BaIoT dataset. Since the system is designed to operate at the edge of the network, it will run on the Jetson Nano for real-time attack detection.**

I. Introduction

Due to their emphasis on low cost-per-unit while providing capabilities at the edge of the network, the Internet of Things (IoT) has seen immense growth in recent years. However, these devices are frequently more exposed to threats and the large number in circulation means that single vulnerabilities can reward a hacker with hundreds, if not thousands, of zombie devices [1-2]. Zombie devices can be used to gain a foothold into the overall network or as a launching point at more sensitive servers. Frequently, zombie devices are accumulated into a larger botnet network. Common functions of botnets are to increase their ranks by propagating the attack, organizing into a distributed crypto-mining operation, and launching Distributed Denial of Service (DDoS) attacks [3-5]. In the case of crypto-mining, there can be noticeable impacts on the performance of the host, potentially degrading operations underneath acceptable limits. On the other hand, executing a DDoS attack disrupts operations of external devices by overwhelming them with network traffic until they appear as inoperable by the intended user. This tactic is more commonly used by the well-known botnets, such as the Mirai, Hajime, Reaper, and Gafgyt botnets. For instance, the Mirai botnet was capable of achieving an attack bit rate of around 1 terabit per second when performing a DDoS attack. In order to address this challenge, our project is focused on applying computer vision and deep learning towards detecting when a botnet is attempting to add a device to its ranks.

When applied together, computer vision and deep learning are powerful tools for classification problems. However, the extent of their application within cybersecurity is limited to detecting hidden malware within files [6, 7]. Nevertheless, they can be useful for tasks where a more robust and tested algorithm cannot be developed in a limited time frame. One such case would be botnet detection, where botnet detection and mitigation needs to occur in real time, and it is difficult to algorithmically detect where a botnet is present, since it typically exploits a security flaw to propagate.

For our implementation, we will convert the features of a network traffic packet (a .pcap file) into an image for consumption by a Convolutional Neural Network (CNN). Each image will be composed of multiple packets related by their time-of-arrival. In order to afford stronger protection to the IoT devices, detection will also be completed at the edge of the network. As presented in Figure 1, network traffic will be collected and forwarded to a local Jetson Nano for analysis. The Jetson Nano is a powerful embedded device that is equipped with a 128-core GPU and a Quad-core ARM A57 processor to bring Artificial Intelligence (AI) to the edge of the network [8]. This architecture will allow us to detect whether a botnet is attempting to propagate to nearby devices in real-time.

Through the remainder of this paper will discuss our deep learning architecture for detecting botnet attacks. First, we will discuss previous works and how our current work

differs from them. Next, we will discuss our procedure for converting network traffic into an image for our CNN. Then we will briefly discuss our results when applying the architecture in real-time. The model will be run on both the Jetson Nano an a Raspberry Pi in order to show the difference in capabilities between the devices.

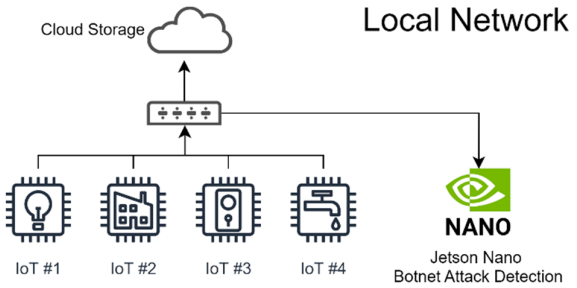

Fig. 1. Simple network architecture with our proposed edge botnet attack detector. Multiple IoT devices with various uses are present within the network and are potentially vulnerable to being incorporated into a botnet. Network traffic is captured by a local Jetson Nano for analysis.

II. Previous Works

Utilizing machine learning for cyberattack detection is not a novel idea. Computer vision algorithms have been used to detect malware hidden inside files [6], however this is not applicable in real-time. Another similar work used autoencoder neural networks to detect the Mirai botnet in real-time [9]. However, each autoencoder had to be trained on only one device and would detect any other device as being malicious. Both of these approaches were interesting an proved that machine learning could be applied to cybersecurity issues in real-time while computer vision could be used for issues where time was not a significant factor. However, they did not consider whether a computer vision approach could be used for malware detection in real-time. In this work, a computer vision approach using images and CNNs is utilized to detect botnet propagation in real-time ton a variety of different IoT applications. Using images allows a significant amount of data to be stored, including data about multiple parameters, as well as historical data. Using a CNN allows for complex processing to be implemented efficiently, however it still requires a significant amount of computing power. Therefore, any way we can pre-process that dataset to reduce the amount of data that needs to be processed will significantly increase the performance of our model.

II. Methodology

For this project, the N-BaIoT dataset was used [9]. This dataset includes data from nine IoT devices. Each of these devices include benign data, and data collected after a Gafgyt and Mirai botnet infection, if the devices can be infected with those malwares. It also contains data involving each of the attack vectors for the Mirai botnet. While it does not include data about the packet captures, it does include data from each packet. For each packet within the network, a record is made of a few statistics, including the average length and standard deviation of the average length, using a dampened window function. In the dataset there are 5 different windows, and 115 parameters total, however we will only be using the shortest window, since our neural network would store temporal data about packets in an image format. This would leave us with 23 parameters to work with. Related work in detecting malicious activity in the N-BaIoT dataset relied on the application of autoencoders [1, 9]. Generally, these implementations saw a decrease in accuracy and an increase in processing time when compared to other deep learning applications. We will compete with these implementations by first converting the network traffic into an image to decrease the size of our network while maintaining performance and increasing performance time.

The first step of our program involved gathering packet data. Initially, we used the pyshark library, which terrible to work with since each type of packet had different variable names. However since it was based on wireshark, a popular packet capture tool, it was assumed to be reliable. However, we encountered problems of the library only being able to capture packets when we were not processing data, so it could miss a large amount of packet data. Ultimately, we ended up using the scapy library, which is more efficient, and also able to continuously detect packets. It is also simpler to work with, with each of the important details, including source and destination ports were more standardized, so they could be extracted with less processing.

The second step involved extracting the same features as were available in the N-BaIoT database. For us to be able to do this, we needed to first group the packets into groups by the parameters extracted from the packet. We did that by simply storing a list of object, which was more convenient than storing raw data with independent processing functions operating on the data. Each of the objects would be able to detect if it matched the connection it was asked, or the reverse connection, which was stored with the forwards connection to reduce the lookup time for the radius and magnitude parameters. This entire program was developed to extract most of the features explained in [9], however a few features like the covariance were excluded, since they were insignificant for our detection. After we extracted our features, we used the OpenCV application to create our images. For each packet we allocated a 2x2 pixel region and included 6 packets per image for an image size of 12x12. The range for each pixel was between 0 and 255.

To start off with, we generated greyscale image for each of the parameters in the smallest window of the dataset for both malicious and benign traffic. While some of the images were similar in nature, in some of them there were some differences for malicious and benign data. For example, one of the parameters was the weight for the connection with matching source IP and MAC addresses, which measured how many packets fell into the window, with damping making earlier packets less important. In Figure 2 below, we can see the difference between a benign image(left), and two malicious images (center and right). It can be seen that malicious images contain significantly more lines, which we can detect. It should be noted that the color is also slightly different for most of the image, which should allow use to also detect this color change on smaller images.

(a) (b) (c)

Fig. 2. Comparison of benign (a) and malicious (b and c) images when considering the source IP and MAC address.

Another parameter that was determined to be significant was the mean of the length for each socket, which was determined by source and destination IP and port numbers. In this case, we observed more random noise for the benign traffic, however the malicious traffic often had the same lines present, and less noticeable noise. A comparison between a benign image and two malicious images is presented in Figure 3.

(a) (b) (c)

Fig. 3. Comparison of benign (a) and malicious (b and c) images when considering the mean of the length for each socket.

We also considered the socket magnitude, which was defined to be the square root of the sum of the squares of the means for each socket pair (reversing source and destination). This had a similar pattern as the previous parameter, however another interesting thing to note was that the magnitude and mean would match for the sockets with malicious data present, which makes sense since most of the malicious connections only send data, instead of receiving it. Again, three sample images are presented below in Figure 4.

(a) (b) (c)

Fig. 4. Comparison of benign (a) and malicious (b and c) images when considering the socket magnitude.

To train our neural network, we used the tensorflow framework, which allowed us to design our neural network as a series of layers, and to train it using backpropogation. It also allowed us to evaluate the neural network on the Jetson Nano after we trained it, and to achieve a high enough throughput by using the GPU provided on the Jetson Nano. For our Convolutional Neural Network, we experimented with different combinations and found the following network to be most effective for our application. The input to our network was 12x12 images and the network starts with a convolutional layer with a kernel of 7x7 and 9 filters and the ReLU activation function, which resulted in an output of size 6x6x9. Next, we included a maxpooling layer with a 2x2 kernel, which resulted in an output of size 3x3x9. After another convolutional layer with a kernel size of 2x2x9 and 6 filters we ended with a 2x2x6 image. We then used a dense layer with 8 outputs and using the ReLU activation function and ended with a dense layer that had 2 outputs. Figure 5 presents the overall model of our CNN. While this structure does utilize a classification function at the last layer, for each of the classes, the detection of malicious/benign was determined by which value was larger.

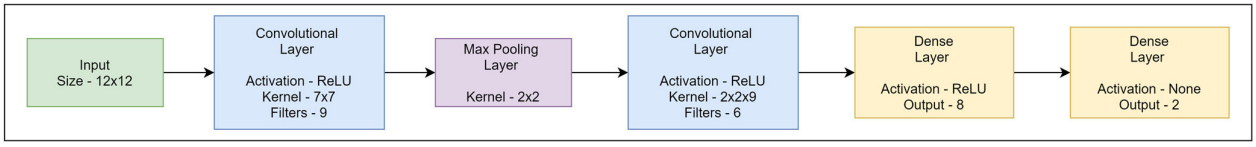

Fig. 5. Convolutional Neural Network for our Botnet Attack Detection Application. The network accepts images with size 12x12 and returns two outputs, one indicated whether malicious activity was present while other indicated none was present.

III. Results and Analysis

After finalizing the design of our CNN model, we then proceeded to train the model using the N-BaIoT dataset. The dataset captured data on seven IoT devices, ranging from a baby monitor to a security camera. We decided to split the dataset into testing, validation, and training devices in order to observe the effects of transferring the model between devices. For our training dataset we included the doorbell, thermostat, the baby monitor. The two Provision security cameras were used as a validation dataset. For the testing dataset we included the two Simple Home security cameras.

To tune the hyperparameters, the smaller training dataset was used to reduce the evaluation time. Since the testing dataset contained three devices, one device was used for the training dataset, one in the validation dataset, and the third in the testing dataset. The training dataset was used to train our network, while the validation dataset was used to terminate training when accuracy started to decline. The testing accuracy after optimizing hyperparameters was found to be 99.87%, which showed that training on one device and testing on another yielded good results. When using the full dataset, we used the same hyperparameters that were found to work well on the training dataset. When the trained model was applied to the testing data, we achieve an accuracy of 99.82% over the two Simple Home Security cameras.

Another part of this project involved simulating the performance of our neural network on a physical device, so we can both verify that the neural network works, while at the same time being able to measure the performance. This simulation had both a malicious and benign traffic source, with a VNC server being used as the benign traffic source. A VNC server would be one of the most extreme malicious cases, where the high packet rate could be detected as malicious, as well as the command and control potential of it. Additionally, even if the VNC server was not detected as a malicious system, it would have a high packet rate, which may make the malicious data less detectable. For the malicious part of the traffic, we implemented a SYN scan, which would scan random IPs on port 23 for a TCP server. We set a low timeout to ensure that the scan could be performed on a single thread, with a high scan rate. While we would not be able to actually identify whether there was a TCP server present, that was not one of the goals for us, so we would be okay with ignoring this data.

In addition to assessing the accuracy of our model, we also wanted to compare its performance on both the Jetson Nano and the Raspberry Pi. During the simulation, we observed a performance of 50 milliseconds per evaluation for both devices. Since each image contains 36 packets, we achieve a performance of 720 packets per second. We believe that this is sufficient for maintaining real-time detection of botnet threats against the IoT devices.

IV. Conclusion

Overall, we were successful at implementing a new solution to detecting attempts by the Mirai botnet at incorporating new devices into its collective. Our implementation converted incoming packets into an image and then inserted them into our Convolutional Neural Network model. This model was able to achieve 99.82% accuracy, when applied to the two security cameras from the dataset. Further, we were able to achieve a performance of 720 packets per second, which we believe is sufficient for real time applications.

Future work would attempt to increase the performance of the network while implemented on the Jetson Nano. We had trouble taking full advantage of the 128-core GPU, which could allow us to process multiple batches simultaneously. Potentially, this could allow us to increase the number of devices we are monitoring simultaneously. Further, we could decrease the time it took to pre-process the data into an image. Finally, our project currently only detects whether a botnet is attempting to take control over present IoT devices. This can be expanded to detect the specific type of attack or vulnerability that the botnet may be attempting to use.

References

[1] E. Bertino and N. Islam, "Botnets and Internet of Things Security," in Computer, vol. 50, no. 2, pp. 76-79, 2017

[2] M. Gromov, D. Arnold and J. Saniie, “Tackling Multiple Security Threats in an IoT Environment,” in *2022 IEEE International Conference on Electro Information Technology*, 2022.

[3] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher and Y. Elovici, “N-BaIoT - Network-Based Detectino of IoT Botnet Attacks Using Deep Autoencoder,” *IEEE Pervasive Computing,* vol. 17, no. 3, pp. 12-22, 2018.

[4] C. Kolias, G. Kambourakis, A. Stavrou and J. Voas, "DDoS in the IoT: Mirai and Other Botnets," in Computer, vol. 50, no. 7, pp. 80-84, 2017

[5] J. Margolis, T. T. Oh, S. Jadhav, Y. H. Kim and J. N. Kim, "An In-Depth Analysis of the Mirai Botnet," 2017 International Conference on Software Security and Assurance (ICSSA), 2017

[6] I. Baptista, S. Shiaeles and N. Kolokotronis, “A Novel Malware Detection System Based on Machine Learning and Binary Visualization,” in *2019 IEEE International Conference on Communications Workshops* , 2019.

[7] L. Barlow, G. Bendiab, S. Shiaeles and N. Savage, "A Novel Approach to Detect Phishing Attacks using Binary Visualisation and Machine Learning," 2020 IEEE World Congress on Services (SERVICES), 2020, pp. 177-182

[8] NVIDIA , “Jetson Nano Developer Kit,” [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit.

[9] Mirsky, Y., Doitshman, T., Elovici, Y., & Shabtai, A. (2018). Kitsune: An Ensemble of Auto-encoders for Online Network Intrusion Detection. Proceedings 2018 Network and Distributed System Security Symposium.