Computer Communications 193 (2022) 53–62

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

# BotStop : Packet-based efficient and explainable IoT botnet detection using machine learning

Mohammed M. Alani *

*Computer Science Department, Toronto Metropolitan University, Toronto, Canada*
*School of IT Administration and Security, Seneca College of Applied Arts and Technology, Toronto, Canada*

ARTICLE INFO

*Keywords:*
IoT
Botnet
Intrusion detection
Explainable machine learning

ABSTRACT

The rapid increase in the adoption of the Internet of Things has increased the attack surface of these devices, encouraging malicious actors to target these devices. Vulnerable Internet of Things devices are susceptible to botnet infections that give attackers control over these devices from where they can launch attacks on other targets. In this paper, we present an efficient packet-based botnet detection system based on explainable machine learning. Our proposed approach also focuses on feature selection to produce a data set with only seven features to train a machine learning classifier that achieves very high accuracy. Testing the proposed system demonstrates an accuracy exceeding 99% relying on these seven selected characteristics extracted from the network packets. The proposed model is explained using Shapley additive explanation to provide transparency to the classifier prediction process.

## 1. Introduction

The broad adoption of the Internet of Things (IoT) has witnessed unprecedented growth in the past few years, with an even higher growth projected in the coming few years. Fig. 1 shows the past and projected growth in connected IoT devices in comparison to non-IoT devices.

The number of connected IoT devices is expected to nearly double in three years. Such tremendous technological growth comes with a significant expense: malicious attacks. Due to the lack of commitment of commercial IoT vendors, many IoT devices do not receive security patches promptly when vulnerabilities are discovered. In addition, consumer hesitation also contributes to this lack of routine patching. The following factors drive the current weak security posture of the IoT in most application areas:

- Many IoT vendors focus on fast production instead of strong security. Devices are often manufactured with old open-source software that contains known vulnerabilities [2].
- Even when vendors issue patches, many are difficult to install on IoT devices. Either the firmware does not support Over-The-Air (OTA) updates, or the update process requires a tech-savvy user or administrator, preventing non-technical IoT users from completing the patching process.
- The limited processing power and memory available in IoT devices increases the difficulty of defending against attacks at the device level.
- Many users of IoT do not change the default settings, so devices may still use the default username and password that are easily guessed or identified with brute force methods, as in the case of the Mirai botnet [3]. In addition, poor security practices common with IoT users make them susceptible to many security threats.

A specific threat to IoT devices is the botnet attack. A botnet is a network of breached devices controlled by a malicious actor. Fig. 2 illustrates the operation of a typical botnet.

The term botnet means a network of bots or robots. An attack begins with a malicious actor gaining unauthorized access to a single device and implanting botnet malware to take over control of the device without alerting its legitimate users. The robot, sometimes referred to as a zombie, then establishes a connection with a Command and Control (C&C) center owned by the attacker. The bot remains ready to receive commands from the C&C to launch another attack.

IoT devices, in the current general state of having many unpatched vulnerabilities, enable an accessible recruitment pool for botnets. An infamous IoT botnet is Mirai malware that appeared in 2016 and soon climbed to the top news stories following a massive orchestrated Distributed Denial of Service (DDoS) attack. In September 2016, Mirai temporarily crippled several high-profile online services such as OVH, Dyn, and Krebs on Security with the DDoS that exceeded a rate of 1 Tbps, the highest in history at that time. Mirai attacks were carried out via small IoT devices, such as home routers, air quality monitors, and personal surveillance cameras. At its peak, Mirai infected more than 600,000 vulnerable IoT devices connected to the Internet [4].

* Correspondence to: School of IT Administration and Security, Seneca College of Applied Arts and Technology, Toronto, Canada.
*E-mail address:* m@alani.me.

https://doi.org/10.1016/j.comcom.2022.06.039
Received 18 April 2022; Received in revised form 12 June 2022; Accepted 26 June 2022
Available online 30 June 2022

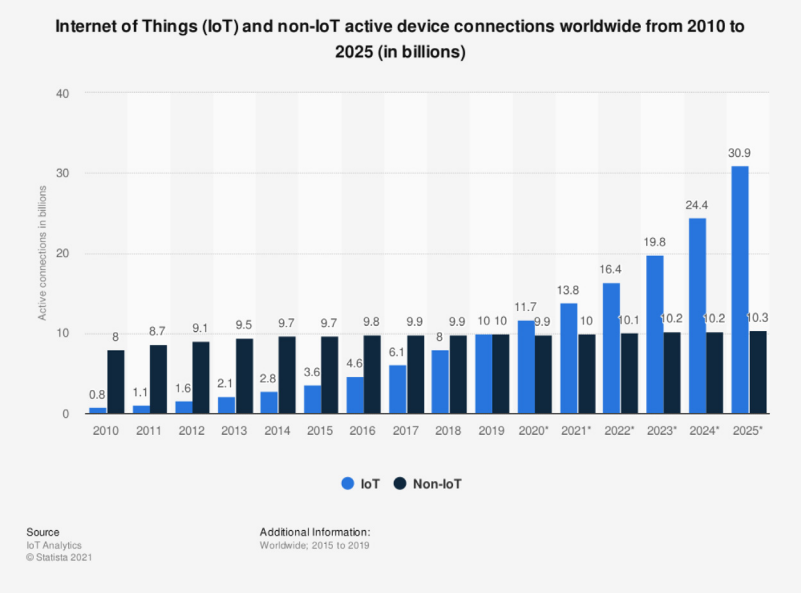


**Fig. 1.** IoT and non-IoT active device connections worldwide from 2010 to 2025 [1].

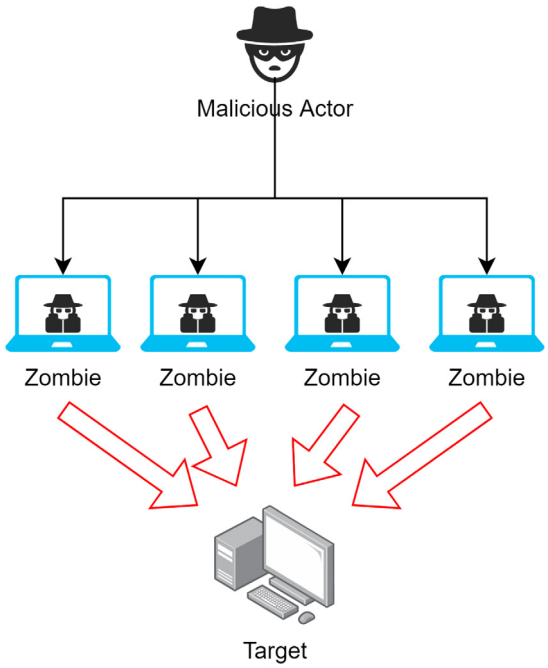


**Fig. 2.** The operation of botnets.

The creator of Mirai released its source code in 2017 on the dark web for other malicious actors to replicate, which led to a surge in variant botnets developed using Mirai's codebase. Mirai-variant botnets, such as Katana, Boonet Botnet, Mukashi, and over 60 variants, have been used in large-scale attacks [5–7].

In September 2020, Cloudflare received a large-scale UDP-based DDoS attack that peaked at 654 Gbps and lasted two minutes [8]. This DDoS was initiated by the Moobot botnet that infects IoT devices and originated from 18,705 unique IP addresses located across 100 countries. This attack exploited two known vulnerabilities for a long time but were never patched in the most vulnerable IoT devices.

Even recently, according to [9], the Gafgyt and Mirai IoT botnets comprise half of all DDoS attacks within the first half of 2021, amounting to 2.8 million DDoS attacks.

Once an IoT device is infected with Mirai or another family of botnets, it begins scanning the connected network searching for other IoT devices [10]. Once found, the infected device attempts to identify the specific type of IoT device found and launches attacks, such as port scanning, operating system (OS) fingerprinting, or telnet brute force, among other types of attacks, to infect the other devices and force it to become a bot. These infected bots can then be later coordinated to conduct large-scale attacks, such as DDoS [11].

Reports by Paloalto Networks in February 2021 indicated that a Mirai variant was detected while exploiting CVE-2021-27561 and CVE-2021-27562, only hours after these vulnerability details were published [12]. Both were considered critical vulnerabilities that could be exploited to provide an attacker with root access to execute remote code.

Recent reports, published in [13], suggested that a newer version of the Mirai botnet was exploiting CVE-2022-22965 [14], named Spring4Shell. With a CVS score of 9.8, this critical vulnerability is considered one of the most dangerous vulnerabilities discovered in 2022.

Earlier in 2022, Mirai variants were detected exploiting CVE-2021-44228, which is an unauthenticated remote code execution (RCE) vulnerability in the Log4j logging server that is popularly used on Java servers [15].

### 1.1. Research contribution

In this paper, we present BotStop, a packet-based botnet detection system that examines incoming and outgoing network traffic in an IoT device to prevent infections from botnets. The proposed system is founded on explainable machine learning (ML) algorithms with features extracted from network packets. Once an attack is detected, the source is blocked. Much of the previous intrusion and botnet detection research was focused on examining network flow instead of operating at the packet level, as in our proposal. The prior approach causes a noticeable delay in detection as the network flow must end or timeout before the attack can be identified. On the other hand, with our proposed system, detection is based on a few features extracted from network packets to provide high efficiency in detection capability. The contributions of this research are summarized in the following.

- Design and develop a high accuracy botnet detection system based on machine learning (Section 5).

**Table 1**
Annotations used in the paper.

| Symbol | Description |
|---|---|
| Accuracy | The ratio of correct predictions over all predictions |
| Precision | The ratio of positive instances correctly detected by the classifier |
| Recall | The ratio of positive instances correctly detected by the classifier |
| $F_1$ Score | The harmonic mean of precision and recall |
| $\mu$ | Statistical mean value |
| $\sigma$ | Standard deviation |

- Minimize the number and type of features required for detection by a machine learning classifier while maintaining a high accuracy (Section 5.5).
- Provide detailed explainability of the detection classifier using Shapley additive explanation (SHAP) values (Section 6).
- Curate a lightweight version of the data set that can be utilized in research for botnet detection (Section 5.5).

### 1.2. Paper layout

In the next section, previous works addressing the research problem are reviewed, followed by an overview of the proposed system and its operation in Section 3. Details on the data set used in training and testing, along with detailed preprocessing steps, are presented in Section 4, and Section 5 describes the conducted experiments, the results, and validation. Section 5 discusses the explainability of the machine learning model using SHAP values, followed by a performance comparison with previous work. Finally, Section 8 provides conclusions with future research directions. Table 1 shows a list of all annotations used in the paper.

## 2. Related works

A significant number of research papers have covered botnet detection as part of general intrusion detection solutions. Although reasonably successful, this approach still does not adequately address the rapidly expanding botnet problem. In our review of related works, we explore a variety of relevant data sets that provide in-depth information on botnet network behavior, such as N-BaIoT, IoTBot-IDS, and BoT-IoT. We explore machine learning-based solutions that utilized flow-based features and others that utilized packet-based features to compare the performance of both approaches. Within our research, we could not find an explainable machine learning approach that addresses the botnet problem to the best of our knowledge.

In 2018, McDermott et al. presented an IoT botnet detection system based on deep learning [16]. The system utilized a bidirectional long–short-term memory-based recurrent neural network (BLSTM-RNN) and compared its performance with the long–short-term memory-based recurrent neural network (LSTM-RNN). This research also produced a labeled data set that could be used for botnet detection, and their testing demonstrated high accuracy in most attack vectors, but not all.

Al Shorman et al. presented in 2020 a botnet detection mechanism based on a one-class support vector machine (OCSVM) [17]. The research incorporated an unsupervised evolutionary IoT botnet detection method using a Grey Wolf Optimization (GWO) algorithm to optimize the hyperparameters of the OCSVM and, simultaneously, identify features that best describe the IoT botnet problem. The system was tested using the N-BaIoT data set and showed an average g-mean metric of 0.968, with an average detection time of 5 s. Such a detection time is considered very high compared to other state-of-the-art detection algorithms. However, this research ignored the imbalance problem of the N-BaIoT data set, which limits generalization. In addition, the proposed method employed data normalization before training and testing, which can be computationally intensive for an IoT device to perform, making practical deployment unrealistic.

In 2020, Nguyen et al. presented a graph-based IoT botnet detection approach [18] that claimed to be lightweight while extracting high-level features from function-call graphs, called PSI-Graph, for each executable file. This method does not examine traffic but considers the functional-call graphs. The accuracy achieved 0.987, with a data set of 11,200 ELF files consisting of 7,199 IoT botnet samples and 4,001 benign samples.

Sriram et al. published in 2020 a network flow-based IoT botnet detection system based on deep learning [19] that utilized the N-BaIoT data set. Test results with a Deep Neural Network (DNN) classifier achieved an accuracy of 1.000 with zero false positives and false negatives and a testing time of 5.553 s. This paper also ignored the imbalance problem in the data set and overlooked that the DNN training took more than ten days. DNNs are expected to be computationally intensive and could o overload deployed IoT devices compared to other machine learning algorithms [20].

Vinayakumar et al. presented in 2020 another IoT botnet detection DNN method based on the analysis of domain name system (DNS) traffic [21]. The proposed framework consists of two levels. The first estimates similarity measures of DNS queries using Siamese networks based on a predefined threshold for selecting the most frequent DNS information across ethernet connections. The second level features a domain generation algorithm based on a DNN to categorize normal and abnormal domain names. This framework utilizes various visualization methods to characterize the data set and the embedding features, with testing results demonstrating accuracy of 0.992 and an $F_1$ Score of 0.915.

Abu Khurma et al. featured in 2021 an IoT botnet detection method based on swarm intelligence [22]. A wrapper feature selection model (SSA-ALO) was designed by hybridizing the Salp Swarm Algorithm (SSA) and Ant Lion Optimization (ALO). Experiments were performed using the N-BaIoT data set, with testing resulting in an average accuracy of 0.984 and an average testing time of 4.673 s.

In 2021, Ashraf et al. presented an IoT botnet detection framework based on a statistical learning-based method [23] utilizing the Beta Mixture Model (BMM) and a Corr-entropy model to capture normal behavior. Any deviation from the determined normal behavior is flagged as an anomalous event. While demonstrating accuracy of 0.992, the network flow features selected in this approach included many host-specific features, such as the source and destination IP and media access control (MAC) addresses, which undermines the framework's capability to generalize beyond its training data set. Additional research explored alternative methods of detecting botnet attacks, such as [26–29].

Table 2 summarizes this review of previous works, and further information about the other works can be found in [30,31], and [32].

## 3. Proposed system overview

The proposed system, BotStop, is an explainable machine learning-based bot detection system designed to achieve high accuracy and efficiency. The detection process relies on features extracted from incoming and outgoing network packets. Fig. 3 shows an overview of this proposed system.

The system is separated into development and deployment stages. At the development stage, the raw data set undergoes feature extraction. The feature extraction is focused on packet-level features, unlike other research directions focusing on network-flow features. Further details about the selected features are explained in Section 4. The tool used to extract the features from pcap files into comma separated values (CSV) format is called tshark [33]. This tool reads the captured raw files and extracts one instance per packet. Each instance contains the features extracted from that particular packet.

Once the selected features are extracted, the data set undergoes preprocessing. The purpose of the preprocessor is to prepare the data for training and testing. Within this stage, the data anomalies, such as text based data, and missing features, are handled. This step produced

**Table 2**
Summary of previous work reviewed.

| Reference | Algorithm used | Data set | Flow-based | Packet-based | Explainable model |
|---|---|---|---|---|---|
| McDermott et al. [16] | BLSTM-RNN | [16] | – | ✓ | – |
| Al Shorman et al. [17] | OCSVM | N-BaIoT | ✓ | – | – |
| Nguyen et al. [18] | Graph-based | [18] | – | – | – |
| Sriram et al. [19] | DNN | N-BaIoT | ✓ | – | – |
| Vinayakumar et al. [21] | DNN | N-BaIoT | ✓ | – | – |
| Abu Khurma et al. [22] | SSA-ALO | N-BaIoT | ✓ | – | – |
| Ashraf et al. [23] | BMM | IoTBoT-IDS | ✓ | – | – |
| Pokhrel et al. [24] | kNN | BoT-IoT | ✓ | – | – |
| Proposed work | XGB | [25] | – | ✓ | ✓ |

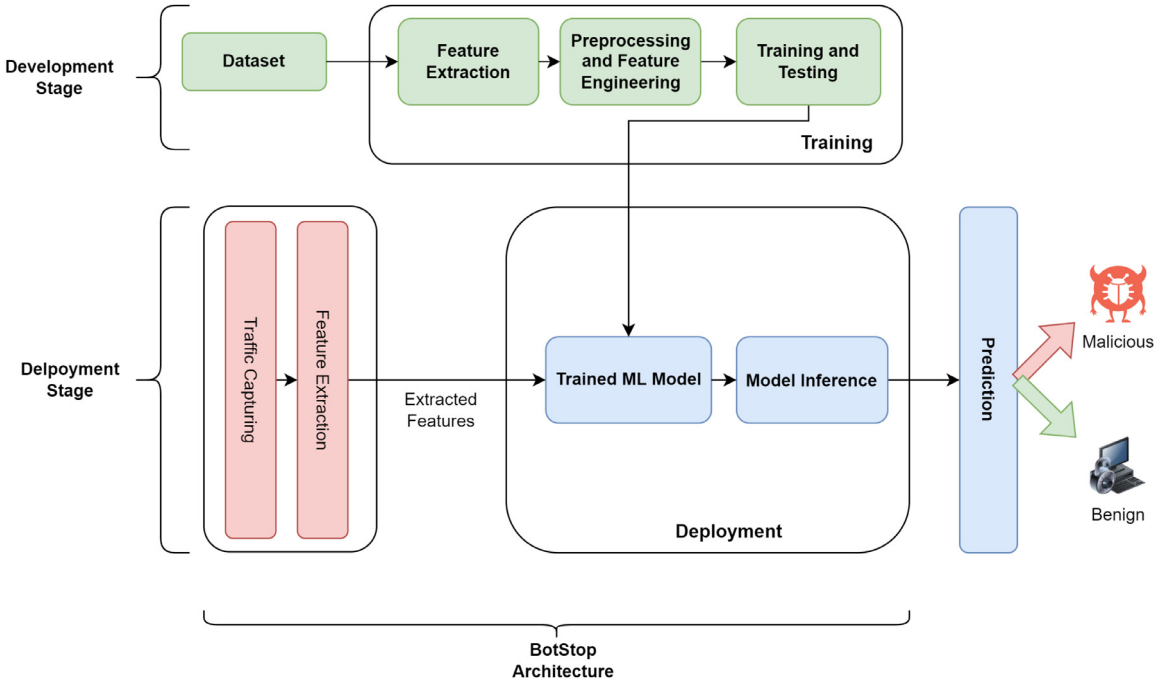


**Fig. 3.** An overview of the BotStop.

a data set with 23 features. The details of preprocessing are also explained in Section 4. The next step is to perform feature engineering to select the most effective features from the data set using RFE technique. This step eliminates features based on feature importance, as explained in Section 5.5. This step produces a smaller version of the data set with seven features only. This reduction in features improves the system's efficiency significantly, and makes it a lightweight solution that is suitable for the limited resources available in IoT devices.

The resulting data set of seven features undergoes training, testing, and validation, as described further in Section 5.3. These steps also present one of the trained classifiers as the best performing in terms of accuracy and timing metrics. Following the selection of the best-performing classifier, the model is stored for later use during the deployment stage.

At the deployment stage, the network traffic is collected in the "Traffic Capturing" unit. The tool used for packet capturing is called tcpdump [34]. This tool can be customized to be lightweight to operate with minimum footprint on the IoT device. The captured traffic is then passed to the feature extractor unit to extract features from raw traffic. The tool utilized for this purpose is called tshark [33]. This tool is leveraged during this phase to extract data directly from the raw packets and produce a comma-separated file containing the feature values. Once the features are extracted and passed to the pre-trained classifier, the classifier performs inference on this received data to return a prediction as to if this packet is malicious or benign.

If the packet is deemed "benign", nothing is done. If the examined packet is predicted to be malicious, the source IP address of the packet is blocked through a lightweight firewall named iptables [35]. This firewall was included with Linux kernels starting at version 3.13 in 2014, and all subsequent versions [36], and available in most IoT device Linux firmware. The units highlighted in the figure in bold were implemented in our experiments.

## 4. The data set and preprocessing

The data set for the initial training and testing was introduced in [25] and created by capturing raw packets of various types of network attacks within an IoT environment. Two smart home devices were used, SKT NUGU (NU 100) and the EZVIZ Wi-Fi Camera (C2C Mini O Plus 1080P). Other devices, including laptops and smartphones, were connected via the same wireless network, and packets were captured on a computer with a network adapter in monitor mode. The wireless headers were removed by Aircrack-ng .

Consisting of 42 raw network packet files (pcap) at different times, the data files capture multiple types of attacks. Table 3 lists these different attacks and the number of packets in each.

As the scope of this research is focused on botnets, we only include the raw data of the last four categories in Table 3, in addition to benign traffic. This subset includes 2,339,621 packets divided into 17 files, which contain packet captures of benign IoT traffic and multiple types of Mirai botnet traffic. The UDP Flooding attack is represented by 949,284 packets, which is approximately 40% of the entire set and 91% of the attack class. Therefore, we decided to reduce this type of attack in the data set to ensure the training process does not negatively impact

**Table 3**
Attack types and packet sizes included in the training data set.

| Type | Total packets | Attack packets |
|---|---|---|
| Normal | 137, 396 | – |
| ARP spoofing | 194, 184 | 101, 885 |
| DoS-SYN flooding | 141, 709 | 64, 646 |
| Scanning | 310, 480 | 25, 210 |
| Mirai-UDP flooding | 1, 187, 114 | 949, 284 |
| Mirai-ACK flooding | 313, 462 | 75, 632 |
| Mirai-HTTP flooding | 248, 294 | 10, 464 |
| Mirai-Host discovery and Telnet brute force | 453, 355 | 2, 597 |

other attack classes with such an imbalance. We eliminated redundant files that contained the UDP flooding attack, resulting in a final raw data set that includes 1,503,895 packets, of which 228,251 are botnet attack packets.

After defining the raw data set for training, we selected the features to extract. Because we focus on creating a packet-based detector, we limit the feature extraction at the packet level instead of at the network flow level, as discussed in Section 2. We performed a detailed analysis of the Mirai botnet operation by examining its source code. We also reviewed the captured packets thoroughly to consider the similarities and differences between benign and malicious traffic. Based on this data study, we selected 35 packet-level features that demonstrated some distinction between the benign and malicious classes, which are listed in Appendix. The following describes the selection criteria for these features:

- Features needed for labeling were identified by the creator of the data set as those necessary to clearly identify the packets as benign or malicious packets, including ip.src, ip.dst, and eth.addr. Most of these features are removed later to ensure generalization.
- Features that identified distinct characteristics between benign and malicious traffic were selected by comparatively examining the traffic and identifying fundamental differences in specific fields of packets, such as tcp.flags.syn, ip.flags, and icmp.

The resulting data set is characterized by the following:

1. Includes 1,503,895 instances, each containing data extracted from a single packet. These instances are divided into 1,275,644 benign packets and 228,251 malicious botnet packets with 35 features extracted from that packet.
2. Host-specific features exist, such as source and destination IP addresses and MAC addresses.
3. The fields ip.flags and tcp.flags are in hexadecimal format.
4. The fields tcp, icmp, and arp are in text format.

Based on these characteristics, we designed custom preprocessing steps to prepare the data for use in training and testing. The Algorithm 1 shows the steps in the preprocessing of the data set.

**Algorithm 1:** Data Set Preprocessing

**Input:** Extracted data set (1,503,895 instances, 35 features)
**Output:** Preprocessed data set (919,920 instance, 23 features)

$Array \leftarrow RawDataset$
In $(Array)$ remove $host\_specific\_features$
In $(Array)$ remove $ip.tos, frame.number$ features
label-encode $tcp, icmp, arp$ features
In $(Array)$ convert $tcp.flags, ip.flags$ features to Decimal
$Array \leftarrow RandomUnderSampling\_majority(Array)$ $Dataset \leftarrow Array$

The first preprocessing step removes the host-specific features, which, if kept in the data set, can lead to poor generalization as the classifier would be trained on a specific set of addresses and could not generalize well beyond the training set. These features include eth.src, eth.dst, ip.src,ip.dst,arp.dst.proto_ipv4, arp.src.proto_ipv4, arp.src.hw_mac, arp.dst.hw_mac, ip.addr, and eth.addr.

The next step removes the frame.number, because it is irrelevant to the actual packet captured, and removes ip.tos, because it was found to be empty throughout all instances. Only 23 features remain after these removals.

The third step encodes the text features of tcp, icmp, and arp, which indicate if the packet contains TCP, ICMP, or ARP data, respectively. We encode these features as 1 when the protocol is used and 0 otherwise.

The fourth preprocessing step converts the flag fields from hexadecimal to decimal. As the above characterizations show, a noticeable imbalance exists between the malicious and benign packets . Therefore, the final step in the preprocessing performs a random undersampling for the majority class. Here, data instances are removed randomly from the majority class (benign) to produce a more balanced version of the complete data set, leaving 919,920 instances, out of which 691,669 are benign and 228,251 malicious.

**Table 4**
Hardware specifications of the implementation environment.

| | |
|---|---|
| Processor | AMD Ryzen 5 3600 |
| Clock speed | 4.2 GHz |
| RAM | 128 GB |
| GPU | NVidia GeForce RTX 3060Ti |

**Table 5**
Software version specifications of the implementation environment.

| | |
|---|---|
| Operating system | Windows 10 professional |
| Python [37] | 3.8.5 |
| TensorFlow [38] | 2.3.0 |
| Keras [39] | 2.4.3 |
| SciKit Learn [40] | 1.0.1 |

## 5. Experiments and results

### 5.1. Implementation environment

Tables 4 and 5 list the hardware and software specifications of the implementation environment used during preprocessing, training, testing, and storage of the trained model.

### 5.2. Performance metrics

Four traditional performance metrics considered for a binary ML-based classifier are the following:

1. True Positive (TP): the number of test instances with a true value and predicted value of 1, divided by the number of test instances with a true value of 1.
2. True Negative (TN): the number of test instances with a true value and predicted value of 0, divided by the number of test instances with a true value of 0.
3. False Positive (FP): the number of test instances with a true value of 0 and predicted value of 1, divided by the number of test instances with a true value of 0.
4. False Negative (FN): the number of test instances with a true value of 1 and predicted value of 0, divided by the number of test instances with a true value of 1.

Combining these four metrics generates a confusion matrix, and our results consider the following six performance parameters:

1. Accuracy: measures the ratio of correct predictions over all predictions with:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

2. Precision: measures the ratio of the accuracy of positive predictions with:

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

3. Recall: measures the ratio of positive instances correctly detected by the classifier with:

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

4. $F_1$ Score: measures the harmonic mean of precision and recall with:

$$F_1 Score = 2 * \frac{\frac{TP}{TP+FN} * \frac{TP}{TP+FP}}{\frac{TP}{TP+FN} + \frac{TP}{TP+FP}} \tag{4}$$

5. Training time: the time required to train the classifier, measured in seconds.
6. Testing time: the time required to obtain a single prediction from a single data instance, measured in microseconds.

### 5.3. Experimental design

The training and testing processes were split into multiple phases, as described below:

1. Initial training and testing: the data set is divided 66.66% for training and 33.33% for testing, and a pipeline of five classifiers is configured, trained, and tested.
2. Feature Selection: the data set is filtered through Recursive Feature Elimination (RFE) based on feature importance, shown in Algorithm 2. The best performing classifier from phase one is selected to be passed through RFE. This recursive elimination of the least important features enables the classifier to perform with higher efficiency and reduces the number of features to be acquired during the later deployment stage.
3. Post Selection Testing: a second round of training and testing is performed on the feature-reduced data set to measure the impact of feature selection on the prediction performance.
4. Validation: a 10-fold cross-validation is performed to validate the selected features and ensure the feature-reduced classifier can generalize beyond its training data, as outlined in Algorithm 3. The data is split into ten random subsets and trained and tested in ten cycles. From each cycle, one subset is excluded from the training and used for testing. If the resulting performance parameters have a low variance, then the classifier is expected to generalize beyond its training data set.
5. Second Data Set Testing: testing is performed with another data set to further ensure the classifier can generalize. We selected the TON_IoT data set for validation with selected features extracted from 100,000 random instances and fed into the trained classifier.

### 5.4. Initial classifier training results

For the initial classifier training and testing phase, we designed a pipeline containing the following five classifiers:

- Random Forest (RF)
- Logistic Regression (LR)
- Decision Tree (DT)
- Gaussian Naive Bayes (GNB)
- eXtreme Gradient Boost (XGB)

Table 6 presents the performance metrics for each classifier calculated from the initial testing with the preprocessed data set using the filtered 23 features.

The XGB classifier presented the highest accuracy with a minimal difference from DT and RF. However, LR resulted in only slightly lower accuracy, while GNB performed poorly in comparison. For the test time, LR performed the fastest at 0.033μs. Considering the combined performance across all metrics, XGB appears to feature the best overall accuracy and timing.

**Algorithm 2:** Successive Feature Elimination Using Feature Importance

**Input:** Data set with m features.
**Output:** Data set with n features.

```
Array ← Dataset
model ← XGBClassifier
TargetFeatures ← n
while Features(Dataset) > TargetFeatures do
    RandomSplit(Array) → Train_Array, Test_Array
    train model with Train_Array
    importance ← FeatureImportance(model)
    i ← index of feature with lowest importance
    Array.DeleteFeature(i)
end
Dataset ← Array
```

**Algorithm 3:** 10-fold Cross-validation Algorithm

**Input:** Balanced data set (*data*).
**Output:** Classifier performance parameters.

```
create(classifer)
split_data(1to10) ← split_to_10_folds(data)
perf_m ← null
for i = 1to10 do
    train_data ← null
    j ← 1
    while j =< 10 do
        if j ≠ i then
            train_data ← append(train_data, split_data(j))
        end
    end
    train(classifier, train_data))
    test(classifier, data(i))
    perf_m ← performance_metrics(classifier)
end
print perf_m
```

### 5.5. Feature selection results

Based on the performance of XGB during the initial testing phase, we decided to apply it in the feature elimination algorithm to calculate feature importance. Fig. 4 shows the change in the $F_1$ Score during feature elimination where the number of features is reduced.

The classifier maintained a very high $F_1$ Score while the features with the lowest importance were recursively eliminated. However, the classifier's performance begins to degrade when the number of features drops below seven. Therefore, we selected this threshold to be the final number of features.

This process produced a reduced version of the data set with 919,920 instances, each containing seven features of frame.len, dp.dstport, ip.flags, tcp.dstport, ip.ttl, udp.srcport, and ip.len.

### 5.6. Post feature selection results

After the feature selection process, the resulting data set was again split 66.66% for a training subset and 33.33% for testing to be used in re-training and re-testing of the five classifiers during the next phase.

**Table 6**
Initial testing results with the filtered 23 features.

| Classifier | Accuracy | Precision | Recall | $F_1$ Score | Train time (s) | Test time (µs) |
|---|---|---|---|---|---|---|
| RF | 0.999041 | 0.999238 | 0.998187 | 0.998711 | 27.194731 | 5.116724 |
| LR | 0.977132 | 0.972857 | 0.965368 | 0.969039 | 17.225258 | 0.033045 |
| DT | 0.998982 | 0.999199 | 0.998067 | 0.998631 | 1.438189 | 0.117475 |
| GNB | 0.828108 | 0.793325 | 0.882372 | 0.805832 | 0.203297 | 0.321857 |
| XGB | 0.999048 | 0.999148 | 0.998294 | 0.998720 | 9.297277 | 0.330669 |

**Table 7**
Testing results with the filtered seven features.

| Classifier | Accuracy | Precision | Recall | $F_1$ Score | Train time (s) | Test time (µs) |
|---|---|---|---|---|---|---|
| RF | 0.997681 | 0.996344 | 0.997435 | 0.996888 | 17.138714 | 4.647947 |
| LR | 0.970093 | 0.959103 | 0.960697 | 0.959896 | 3.269533 | 0.026393 |
| DT | 0.997674 | 0.996331 | 0.997431 | 0.996880 | 0.555178 | 0.059637 |
| GNB | 0.947071 | 0.912983 | 0.960857 | 0.932985 | 0.110710 | 0.158622 |
| XGB | 0.997681 | 0.996344 | 0.997435 | 0.996888 | 4.985767 | 0.225059 |

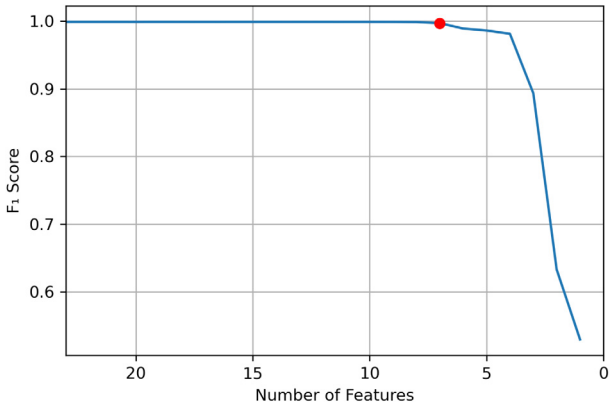


**Fig. 4.** Change in the $F_1$ Score following a reduction of features.

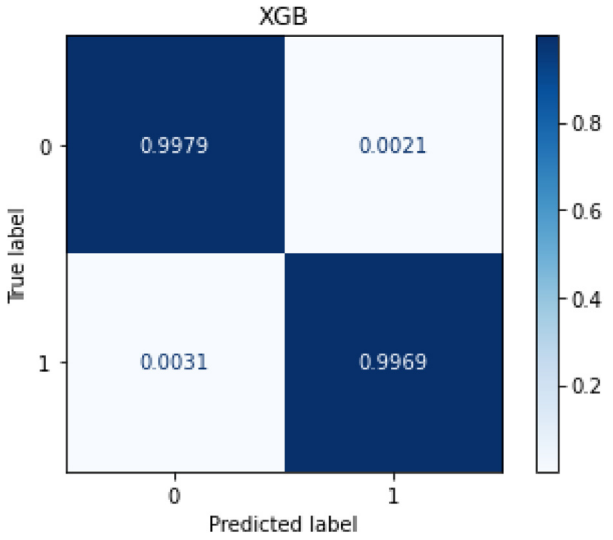


**Fig. 5.** Confusion matrix for the XGB Classifier trained with the seven-feature data set.

Table 7 lists the measured performance metrics of the classifiers after training and testing with the seven-feature version of the data set.

Using the reduced feature list, RF, LR, DT, and XGB classifiers maintained a similarly high performance. However, GNB experienced a significant improvement in accuracy from 0.828 to 0.947. XGB maintained a slightly better performance when considering accuracy and testing time. Fig. 5 presents the confusion matrix for the XGB classifier when tested with the reduced seven-feature data set.

**Table 8**
Results of the 10-fold cross-validation for XGB classifier with the reduced seven-feature data set.

| Fold | Accuracy | Precision | Recall | $F_1$ Score |
|---|---|---|---|---|
| 1 | 0.99742 | 0.99297 | 0.99660 | 0.99478 |
| 2 | 0.99773 | 0.99362 | 0.99728 | 0.99545 |
| 3 | 0.99791 | 0.99462 | 0.99691 | 0.99576 |
| 4 | 0.99780 | 0.99376 | 0.99747 | 0.99561 |
| 5 | 0.99780 | 0.99405 | 0.99710 | 0.99558 |
| 6 | 0.99748 | 0.99338 | 0.99648 | 0.99492 |
| 7 | 0.99761 | 0.99293 | 0.99756 | 0.99524 |
| 8 | 0.99788 | 0.99443 | 0.99694 | 0.99568 |
| 9 | 0.99765 | 0.99351 | 0.99713 | 0.99532 |
| 10 | 0.99781 | 0.99435 | 0.99690 | 0.99562 |
| $\mu$ | 0.99771 | 0.99376 | 0.99704 | 0.99540 |
| $\sigma$ | 0.00015 | 0.00056 | 0.00032 | 0.00031 |

The XGB classifier achieved very low FP and FN rates of 0.21% and 0.31%, respectively, which address significant problems in attack detection. High FN rates cause a high cost to companies due to the resulting wasted time of security personnel who must examine the false alarms. Comparatively, the high FN rate also suggests that many attacks could pass unnoticed.

In terms of the time metrics, all classifiers experienced a noticeable reduction in training and testing times, and this improvement in testing time indicates an improvement in efficiency.

An accuracy score of 0.997681 after feature selection, with an $F_1$ Score of 0.996888, indicates the feature selection process was successful in meeting the research goals of achieving significant efficiency improvement while maintaining high accuracy.

### 5.7. 10-fold cross-validation results

As another validation step, a 10-fold cross-validation was again performed on the reduced seven-feature data set for the XGB classifier. Table 8 shows the results of this validation.

### 5.8. Second data set testing results

To ensure the trained classifier can generalize well beyond its training data set, we performed another test using a second data set, TON_IoT [41]. Collected from multiple IoT and Industrial IoT (IIoT) devices, the data set represents a realistic and large-scale network designed at the Cyber Range and IoT Labs, the School of Engineering and Information Technology (SEIT), UNSW Canberra, and the Australian Defense Force Academy (ADFA).

As this raw data set includes multiple types of attacks, we randomly extracted 100,000 packets of botnet and benign traffic for our repeated experiments. These packets were passed through the same

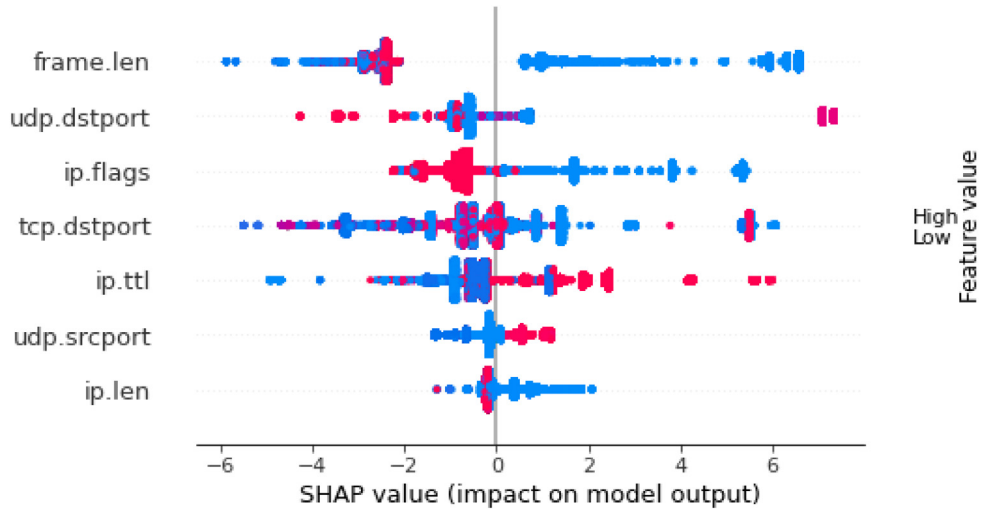


**Fig. 6.** SHAP values summary plot for the seven features.

**Table 9**
Results from testing the XGB classifier with the TON_IoT data set.

| Metric | Value |
|---|---|
| Accuracy | 0.997648 |
| Precision | 0.996309 |
| Recall | 0.997380 |
| $F_1$ Score | 0.996843 |
| Test Time (µs) | 0.218333 |

feature extraction process resulting in the same seven features selected previously. Then, all 100,000 instances were used to test the pre-trained XGB classifier. Table 9 lists the performance metrics resulting from testing the classifier with the 100,000 instances extracted from TON_IoT.

These performance metrics are comparable to the metrics achieved using the original data set, indicating that the trained classifier generalizes well beyond its training data set. This observation also suggests that the selected features are universal and not localized to the original data set.

## 6. Model explainability

To establish greater trust in our proposed solution, we explain the predictions of the trained model. This practice of model transparency builds trust in the ML technique and ensures that the achieved high accuracy originates from explainable conditions and not from a black-box operation.

We employ SHAP to explain our proposed model. First introduced in 2017 [42] as a model-agnostic method to explain machine learning models, SHAP provides explanations based on game theory. The process involves calculating the impact of each feature by calculating the difference between the model's performance with and without the feature present in the data. This approach offers an insight into the contribution that feature provides to the prediction outcome. The explainer type used in our experiment was the TreeExplainer.

Fig. 6 includes the summary plot of SHAP values of the seven selected features in descending order from the highest impact on the prediction decision to the lowest.

The values shown on the left side of the SHAP summary plot axis are those that drag the prediction value down, making it closer to a "benign" classification. The values on the right side of the axis push the prediction value up, making it closer to a "malicious" classification. The red dots represent a high value of the feature, and the blue dots represent a low value. For binary features, red dots represent 1, and blue dots represent 0.

Also shown in Fig. 6, the frame.len feature that indicates the number of bytes within a datalink frame presents the greatest impact on the classifier prediction. Higher values of frame.len push the prediction closer to benign, while lower values push the prediction closer to malicious. The feature ip.len that represents the length of the IP packet behaves similarly to frame.len. Higher values of ip.len push the prediction to benign, while lower values push it toward malicious. These two explanations are consistent with the observation that, during botnet attacks, the amount of information passed within a frame or an IP packet is relatively small. Most of these packets are reconnaissance, brute-force, or DoS attacks, which do not require much data to be carried throughout the attack process.

The feature with the second-highest impact on the classifier's decision is udp.dstport. However, the SHAP values for this feature do not provide a clear impact on the decision. As seen in Fig. 6, a lower destination port value would mostly push the prediction toward benign, but not always. On the other hand, very high port number values can push the prediction toward benign until a certain threshold. Beyond this, higher port numbers push the prediction to malicious. This result is consistent with UDP-based botnet attacks are mostly DoS attacks that target non-standard port numbers. In addition, we allocated all UDP features a value of −1 for all instances extracted from TCP segments, which explains why lower values of udp.dstport are closer to attacks, as most botnet attacks (with the exception of UDP flooding attacks) are conducted using TCP.

Examining the SHAP values of udp.srcport suggests this feature has a very clear indication that high values would push the prediction toward malicious, while lower values push the prediction to benign. This result is consistent with UDP-based botnet attacks using higher port numbers to perform the specific type of attack.

The SHAP values of ip.ttl, which represents the Time-To-Live value within an IP packet, associate a higher TTL value with malicious packets. Alternatively, lower TTL values push the prediction closer to benign. This result is consistent with the fact that most botnet attack packets are sent searching for adjacent IoT devices or try to apply brute force to gain access to these devices, making the TTL value high within a local network.

For the feature of ip.flags, the SHAP values suggest that higher values of the IP flags field push the prediction closer to benign, while lower values push it closer to malicious. This suggests that botnet attack packets have a lower value of IP flags because the value is high when the "Do Not Fragment" bit is set to 1. This is a common practice in legitimate packets that are sent from legitimate applications to inform

**Table 10**
Comparing our proposed system performance metrics to previously published work.

| Related work | Classifier | Data set | Accuracy | $F_1$ Score | Testing time |
|---|---|---|---|---|---|
| [16] | BLSTM-RNN | [16] | 0.9607 | – | – |
| [17] | OCSVM | Na-BaIoT | 0.9680 | – | 5 s |
| [19] | DNN | Na-BaIoT | 1.0000 | 1.0000 | 5.553 s |
| [21] | DNN | Na-BaIoT | 0.9920 | 0.9150 | – |
| [22] | SSA-ALO | Na-BaIoT | 0.9840 | – | 4.673 s |
| [23] | BMM | IoTBoT-IDS | 0.9920 | – | – |
| [24] | kNN | BoT-IoT | 0.9220 | – | – |
| BotStop | XGB | [25] | 0.9976 | 0.9968 | 0.2250 μs |

routers not to fragment this IP packet so that it can be delivered in one piece. On the other hand, attack packets are left to the default setting of 0 for this "Do Not Fragment" bit. Additional information about IP flags can be found in the original IP RFC [43].

Similar to udp.dstport, the tcp.dstport SHAP values are difficult to explain. As seen in Fig. 6, higher TCP ports are more associated with malicious prediction than lower ports. However, this distinction is not clear, which is consistent with the fact that botnet attacks use specific port numbers that are sometimes high and other times low, depending on which stage of the infection the device is currently experiencing. For example, the botnet malware scans many destination ports at the port scanning stage, while in a device brute force stage, the focus is on lower ports, such as 23 for Telnet and 22 for SSH.

In general, the SHAP values featured in Fig. 6 provide reasonable explanations for the classifier predictions that are aligned with a real-world botnet attack anatomy. This observed correspondence increases our trust in the modeled classifier and provides sufficient transparency for it not to be viewed as a black-box system.

## 7. Discussion

Table 10 compares the performance metrics of our proposed system with other related works.

The BotStop system proposed here outperforms all other systems in accuracy, with the exception of [19]. However, this other approach is based on deep neural networks, so its timing metrics are significantly worse than BotStop by 24 million. Another factor contributing to the high accuracy of our technique is the data set used for training. Most of the competing works in Table 10 used the Na-BaIoT data set, which suffers from a severe imbalance that was not addressed in much of the cited research. In addition, our proposed system applied RFE to remove the least important features to reduce noise that did not contribute positively to the classifier's performance.

BotStop also outperforms related works in terms of timing metrics. First, our proposed system utilizes an XGB classifier, which is considered noticeably less computationally intensive when compared to DNNs. Second, our proposed system uses only seven packet-level features to make predictions, while most competing systems incorporate many more features extracted from network flows.

A final key advantage of our proposed system is its explainability. In Section 6, we considered the explainability of the model and demonstrated how the selected features impact the classifier's decision in a transparent way that suggests good alignment with the structure and process of actual IoT attacks.

## 8. Conclusions and future work

In this paper, we implemented an explainable packet-based IoT botnet detection system named BotStop that was intensively tested using multiple stages of testing and validation. The trained classifier achieved an accuracy of 0.9976, with an $F_1$ Score of 0.9968, and a testing time of 0.2250μs that outperformed all techniques from related work. Our proposed system achieved very low FN and FP rates of 0.21% and 0.31%, respectively. The BotStop classifier is explainable using SHAP values, and the model's transparency is aligned with a realistic attack anatomy. Along with multiple validation steps, these results support increasing trust in the high accuracy achieved by BotStop compared to other works.

A future research direction will focus on deployment issues. In addition, we will explore the possibility of creating a distributed model to reduce the computational load on IoT devices further.

**Table A.11**
List of the extracted features with brief descriptions.

| Feature name | Description |
|---|---|
| frame.number | Number of the frame within the captured pcap file |
| frame.len | Number of bytes within a frame |
| eth.src | Source MAC address |
| eth.dst | Destination MAC address |
| ip.src | Source IP Address |
| ip.dst | Destination IP Address |
| ip.proto | Contents of the protocol field within the IP packet |
| ip.len | Number of bytes in an IP packet |
| ip.ttl | Value of the time-to-live field in an IP packet |
| ip.flags | Flags field of an IP packet |
| ip.hdr_len | Length of the IP packet header |
| ip.tos | Contents of the type-of-service field in an IP packet |
| arp | A feature identifying if this is an ARP packet and the type of the ARP packet |
| arp.dst.proto_ipv4 | Destination IP address as defined in the ARP packet |
| arp.src.proto_ipv4 | Source IP address as defined in the ARP packet |
| tcp.flags.syn | Value of the SYN flag in the TCP header |
| tcp.flags.reset | Value of the RESET flag in the TCP header |
| icmp | A field identifying if this is an ICMP packet |
| tcp.checksum.status | TCP header checksum |
| tcp.connection.syn | TCP connection SYN field |
| tcp.connection.synack | TCP connection SYNACK field |
| tcp.dstport | Destination TCP port number |
| tcp.srcport | Source TCP port number |
| tcp.flags | Contents of all of the TCP flags field in the TCP header |
| tcp.len | Length of the TCP segment, as mentioned in the TCP header |
| tcp.time_delta | Time between this packet and the packet received before it |
| tcp.window_size | TCP window size |
| tcp.urgent_pointer | TCP urgent pointer |
| udp.srcport | UDP source port number |
| udp.dstport | UDP destination port number |
| tcp | Field identifying if this packet is using the TCP protocol |
| arp.src.hw_mac | Source MAC address as defined in the ARP packet |
| arp.dst.hw_mac | Destination MAC address as defined in the ARP packet |
| eth.addr | Field defining the MAC address |
| ip.addr | The IP address of the packet (either source or destination address) |

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Appendix. Selected features

See Table A.11.

## References

[1] Global IoT and non-IoT connections 2010–2025 | statista, 2022, Statista, URL https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide. (Online; Accessed 3 April 2022).

[2] D. Palmer, Critical IoT security camera vulnerability allows attackers to remotely watch live video - and gain access, 2021, ZDNet, URL https://www.zdnet.com/article/critical-iot-security-camera-vulnerability-allows-attackers-to-remotely-watch-live-video-and-gain-access-to-networks.

[3] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J.A. Halderman, L. Invernizzi, M. Kallitsis, et al., Understanding the Mirai botnet, in: 26th {USENIX} Security Symposium ({USENIX} Security 17), 2017, pp. 1093–1110.

[4] G. Author, Inside the infamous Mirai IoT botnet: A retrospective analysis, 2020, The Cloudflare Blog, URL https://blog.cloudflare.com/inside-mirai-the-infamous-iot-botnet-a-retrospective-analysis.

[5] L. O'Donnell, Latest Mirai variant targets SonicWall, D-link and IoT devices, 2021, Threatpost, URL https://threatpost.com/mirai-variant-sonicwall-d-link-iot/164811.

[6] E. Montalbano, New Mirai variant 'mukashi' targets Zyxel NAS devices, 2020, Threatpost, URL https://threatpost.com/new-mirai-variant-mukashi-targets-zyxel-nas-devices/153982.

[7] Mirai variant targeting new IoT vulnerabilities, network security devices, 2021, Unit42, URL https://unit42.paloaltonetworks.com/mirai-variant-iot-vulnerabilities. (Online Accessed 1 May 2021).

[8] OT/IoT Security Report February 2021 | Nozomi networks, 2021, Nozomi Networks, URL https://www.nozominetworks.com/ot-iot-security-report. (Online Accessed 1 May 2021).

[9] 15+ Shocking botnet statistics and facts for 2022, 2022, Comparitech, URL https://www.comparitech.com/blog/information-security/botnet-statistics. (Online; Accessed 3 April 2022).

[10] M.M. Alani, Detection of reconnaissance attacks on IoT devices using deep neural networks, in: S.K. Shandilya, N. Wagner, V. Gupta, A.K. Nagar (Eds.), Advances in Nature-Inspired Cyber Security and Resilience, Springer International Publishing, Cham, 2022, pp. 9–27, http://dx.doi.org/10.1007/978-3-030-90708-2_2.

[11] M.M. Alani, IoT Lotto: Utilizing IoT devices in brute-force attacks, in: Proceedings of the 6th International Conference on Information Technology: IoT and Smart City, in: ICIT 2018, Association for Computing Machinery, New York, NY, USA, 2018, pp. 140–144, http://dx.doi.org/10.1145/3301551.3301606.

[12] Mirai variant targeting new IoT vulnerabilities, network security devices, 2021, Unit42, URL https://unit42.paloaltonetworks.com/mirai-variant-iot-vulnerabilities. (Online; Accessed 11 May 2022).

[13] Spring4Shell vulnerability exploited by Mirai botnet | SecurityWeek.com, 2022, URL https://www.securityweek.com/spring4shell-vulnerability-exploited-mirai-botnet. (Online; Accessed 11 May 2022).

[14] NVD - CVE-2022-22965, 2022, URL https://nvd.nist.gov/vuln/detail/CVE-2022-22965. (Online; Accessed 11 May 2022).

[15] L. Cashdollar, Mirai botnet abusing Log4j vulnerability, 2022, Akamai Technologies, URL https://www.akamai.com/blog/security/mirai-botnet-abusing-log4j-vulnerability.

[16] C.D. McDermott, F. Majdani, A.V. Petrovski, Botnet detection in the Internet of Things using deep learning approaches, in: 2018 International Joint Conference on Neural Networks, IJCNN, IEEE, 2018, pp. 1–8.

[17] A. Al Shorman, H. Faris, I. Aljarah, Unsupervised intelligent system based on one class support vector machine and grey wolf optimization for IoT botnet detection, J. Ambient Intell. Humaniz. Comput. 11 (7) (2020) 2809–2825.

[18] H.-T. Nguyen, Q.-D. Ngo, V.-H. Le, A novel graph-based approach for IoT botnet detection, Int. J. Inf. Secur. 19 (5) (2020) 567–577.

[19] S. Sriram, R. Vinayakumar, M. Alazab, K. Soman, Network flow based IoT botnet attack detection using deep learning, in: IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops, INFOCOM WKSHPS, IEEE, 2020, pp. 189–194.

[20] A. Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems, " O'Reilly Media, Inc.", 2019.

[21] R. Vinayakumar, M. Alazab, S. Srinivasan, Q.-V. Pham, S.K. Padannayil, K. Simran, A visualized botnet detection system based deep learning for the Internet of Things networks of smart cities, IEEE Trans. Ind. Appl. 56 (4) (2020) 4436–4456.

[22] R. Abu Khurma, I. Almomani, I. Aljarah, IoT botnet detection using salp swarm and ant lion hybrid optimization model, Symmetry 13 (8) (2021) 1377.

[23] J. Ashraf, M. Keshk, N. Moustafa, M. Abdel-Basset, H. Khurshid, A.D. Bakhshi, R.R. Mostafa, IoTBoT-IDS: A novel statistical learning-enabled botnet detection framework for protecting networks of smart cities, Sustainable Cities Soc. 72 (2021) 103041.

[24] S. Pokhrel, R. Abbas, B. Aryal, IoT security: Botnet detection in IoT using machine learning, 2021, arXiv preprint arXiv:2104.02231.

[25] H. Kang, D.H. Ahn, G.M. Lee, J.D. Yoo, K.H. Park, H.K. Kim, IoT network intrusion dataset, 2019, http://dx.doi.org/10.21227/q70p-q449, IEEE Dataport.

[26] N.M. Yungaicela-Naula, C. Vargas-Rosales, J.A. Pérez-Díaz, M. Zareei, Towards security automation in software defined networks, Comput. Commun. 183 (2022) 64–82, http://dx.doi.org/10.1016/j.comcom.2021.11.014, URL https://www.sciencedirect.com/science/article/pii/S0140366421004436.

[27] R. Panigrahi, S. Borah, M. Pramanik, A.K. Bhoi, P. Barsocchi, S.R. Nayak, W. Alnumay, Intrusion detection in cyber–physical environment using hybrid naïve Bayes—Decision table and multi-objective evolutionary feature selection, Comput. Commun. 188 (2022) 133–144, http://dx.doi.org/10.1016/j.comcom.2022.03.009, URL https://www.sciencedirect.com/science/article/pii/S0140366422000846.

[28] F. Hussain, S.G. Abbas, I.M. Pires, S. Tanveer, U.U. Fayyaz, N.M. Garcia, G.A. Shah, F. Shahzad, A two-fold machine learning approach to prevent and detect IoT botnet attacks, IEEE Access 9 (2021) 163412–163430, http://dx.doi.org/10.1109/ACCESS.2021.3131014.

[29] Y. Chen, B. Pang, G. Shao, G. Wen, X. Chen, DGA-based botnet detection toward imbalanced multiclass learning, Tsinghua Sci. Technol. 26 (4) (2021) 387–402, http://dx.doi.org/10.26599/TST.2020.9010021.

[30] M. Wazzan, D. Algazzawi, O. Bamasaq, A. Albeshri, L. Cheng, Internet of Things botnet detection approaches: Analysis and recommendations for future research, Appl. Sci. 11 (12) (2021) 5713.

[31] Y. Xing, H. Shu, H. Zhao, D. Li, L. Guo, Survey on botnet detection techniques: Classification, methods, and evaluation, Math. Probl. Eng. 2021 (2021).

[32] Z. Al-Othman, M. Alkasassbeh, S.A.-H. Baddar, A state-of-the-art review on IoT botnet attack detection, 2020, arXiv preprint arXiv:2010.13852.

[33] tshark(1), 2022, URL https://www.wireshark.org/docs/man-pages/tshark.html. (Online; Accessed 11 May 2022).

[34] Home | TCPDUMP & LIBPCAP, 2021, URL https://www.tcpdump.org. (Online; Accessed 12 June 2022).

[35] Iptables(8) - Linux man page, 2022, URL https://linux.die.net/man/8/iptables. (Online; Accessed 12 June 2022).

[36] Linux_3.13 - Linux kernel newbies, 2022, URL https://kernelnewbies.org/Linux_3.13#head-f628a9c41d7ec091f7a62db6a49b8da50659ec88. (Online; Accessed 12 June 2022).

[37] Welcome to Python.org, 2021, Python, URL https://www.python.org. (Online; Accessed 29 April 2021).

[38] TensorFlow, 2021, TensorFlow, URL https://www.tensorflow.org. (Online; Accessed 29 April 2021).

[39] K. Team, Keras: the python deep learning API, 2021, URL https://keras.io. (Online; Accessed 29 April 2021).

[40] Scikit-learn: machine learning in Python–scikit-learn 1.0.2 documentation, 2022, URL https://scikit-learn.org/stable. (Online; Accessed 21. Feb. 2022).

[41] A. Alsaedi, N. Moustafa, Z. Tari, A. Mahmood, A. Anwar, TON_IoT telemetry dataset: A new generation dataset of IoT and IIoT for data-driven intrusion detection systems, IEEE Access 8 (2020) 165130–165150.

[42] S.M. Lundberg, S.-I. Lee, A unified approach to interpreting model predictions, Adv. Neural Inf. Process. Syst. 30 (2017).

[43] J. Postel, Internet protocol– DARPA internet program protocol specification, rfc 791, 1981, University of Southern California/Information Sciences Institute.