A Framework for Generating Evasion Attacks for Machine Learning based Network Intrusion Detection Systems

This paper was downloaded from TechRxiv (<https://www.techrxiv.org>).

LICENSE

CC BY 4.0

SUBMISSION DATE / POSTED DATE

13-08-2021 / 19-08-2021

CITATION

Mogg, Raymond; Enoch, Simon; Kim, Dong Seong (2021): A Framework for Generating Evasion Attacks for Machine Learning based Network Intrusion Detection Systems. TechRxiv. Preprint.
<https://doi.org/10.36227/techrxiv.15164463.v1>

DOI

[10.36227/techrxiv.15164463.v1](https://doi.org/10.36227/techrxiv.15164463.v1)

A Framework for Generating Evasion Attacks for Machine Learning based Network Intrusion Detection Systems

Raymond Mogg¹, Simon Yusuf Enoch^{1,2}[0000–0002–0970–3621], and Dong Seong Kim¹[0000–0003–2605–187X]

¹ The University of Queensland, St Lucia QLD 4072, Australia
E-mail: raymondmogg@gmail.com, dan.kim@uq.edu.au

² Federal University, Kashere, Gombe State, Nigeria
E-mail: simonpuks@gmail.com

Abstract. Intrusion Detection System (IDS) plays a vital role in detecting anomalies and cyber-attacks in networked systems. However, sophisticated attackers can manipulate the IDS' attacks samples to evade possible detection. In this paper, we present a network-based IDS and investigate the viability of generating interpretable evasion attacks against the IDS through the application of a machine learning technique and an evolutionary algorithm. We employ a genetic algorithm to generate optimal attack features for certain attack categories, which are evaluated against a decision tree-based IDS in terms of their fitness measurements. To demonstrate the feasibility of our approach, we perform experiments based on the NSL-KDD dataset and analyze the algorithm performance.

Keywords: Adversarial Machine Learning · Evasion attacks · Genetic Algorithms · Intrusion Detection

1 Introduction

In the past few years, cyber-criminals have become more skilled and organized, as attackers use sophisticated means to frequently evade state-of-the-art security defenses on networked systems [8]. Consequently, the attackers can gain authorized access, exploit security vulnerabilities, and control a victim machine without being detected. An *evasion* is any technique used by the cyber-criminals that modifies a detectable attack to avoid possible detection. Intrusion Detection Systems (IDSes) play a critical role in the security of networked systems by monitoring and detecting malicious attacks against systems [3]. Machine Learning (ML) algorithms can assist an IDS to continuously learn and adapt to changes based on known attacks and to improve detection accuracy [9, 10, 13, 19]. However, despite the benefits offered by the ML approaches, attackers with knowledge about the type of the model or the working/design of the system, can exploit weaknesses in the algorithm and evade the defense mechanism put in place. In this paper, we focus on the practicability of generating interpretable

evasive attacks based on IDSes' benign samples. In particular, we investigate the feasibility of generating interpretable teardrop attack and probe attack against IDSes, and also we develop an approach to generate the new attacks from benign samples (which were considered normal but have characteristics of real attacks). We achieved this by generating possible samples that are similar to known seed attacks while still ensuring that the IDS classified them as benign. Moreover, we developed a decision tree (DT) based IDS which can be trained based on a dataset using behavior-based detection and network-based audit. We design a Genetic Algorithm (GA) [4,21] that uses the output of the decision tree-based IDS as a feedback loop to compute the fitness measurement that produces samples that are similar in structure (to a known attack), but were incorrectly classified as benign. Furthermore, in order to demonstrate the feasibility of our approach, we utilize the NSL-KDD dataset to conduct experiments.

- We investigate the possibility of generating interpretable evasion attacks against IDSes from benign samples;
- We develop a decision tree-based IDS with behavior-based detection and a network-based audit source;
- We propose a technique that generalized attack pipeline to allow the generation of interpretable evasion attacks against any black box IDS using Genetic Algorithms;
- We perform a comparative analysis of attack performance via experiments on the NSL-KDD dataset.

The rest of the paper is organized as follows. Section 2 summarizes the related work on attacks and machine learning-based IDS. Section 3 introduces our proposed approach. In Section 4, we present the experiments including evaluation, numerical results, and discussions. Finally, Section 5 concludes the paper.

2 Related Work

Pawlicki *et al.* [7] proposed an approach for evasion attack detection for IDS based on Neural Networks. In their work, they developed a four-phase IDS training/testing process which is capable of attacks binary classification (i.e., attacks or benign) based on a dataset. Vigneswaran *et al.* [20], proposed an approach to predict attacks on network IDS using Deep Neural Networks based on KDDCup-‘99’ dataset. Furthermore, they compared the results of the deep learning methods with other classical ML algorithms (e.g., Linear Regression, Random Forest, Linear Regression, Linear Regression), and then showed that the deep learning methods are more promising for cybersecurity tasks.

Roopak *et al.* [11] presented a deep learning IDS model for attack detection and security of Internet of Things networks, where they compared the deep learning models, machine learning models, and a hybrid model. Their results showed that the hybrid model performed better than the other models. Karatas *et al.* [6] used the Neural Network approach to identify new attacks for dynamic IDS and

to improve attack detection. Chapaneri *et al.* [2] presented an approach to detect malicious network traffic based on deep convolution Neural Networks using UNSW-NB15 dataset. Sabeel *et al.* [12] compared the performance of two techniques; deep neural network and Long Short Term Memory, in terms of their binary prediction of unknown Denial of Service (DoS) and Distributed Dos attacks using CICIDS2017 dataset. Their results showed that both models failed to accurately detect unknown (new) attacks as a result of the attacker's action of varying his profile slightly. In [15], the author presented a methodology for the automatic generations of rules for classifying network connections for IDS using genetic algorithms and DTs. Sarker *et al.* [14] presented a machine learning-based IDS model named IntruDTree for detecting cyber-attacks and anomalies in a network. In this work, they consider the ranking of security features according to their importance and based on the ranking a tree-based IDS is constructed. In [1], Bayesian Network, Naive Bayes classifier, DT, Random Decision Forest, Random Tree, Decision Table, and Artificial Neural Network was used to detect inconsistency and attacks in a computer network based on datasets. In the work, they showed that Random Decision Forest and DT outperformed their counterpart, respectively in terms of accuracy in their classifiers. Motivated by this work, we adopt the DT approach for our work. Sindhu *et al.* [16] proposed a lightweight IDS to detect anomalies in networks using on DT, where they removed redundant instance that they think may influence the algorithm decision. Stein *et al.* [17] presented a technique using a genetic algorithm and DT to increase the detection rate and decrease false alarm for IDS by selecting features based on DT classifiers. They used the KDDCUP 99 dataset to train and evaluate the DT classifier. Their results showed that the GA and DT combined were able to outperform the DT algorithm without feature selection. Ingre *et al.* [5] proposed a DT-based IDS that uses the CART (Classification and Regression Tree) algorithm with Gini index as splitting criteria for pattern classification and correlation-based feature selection for dimensionality reduction in order to improve the performance of IDS with respect to time and space.

3 Proposed Approach

Figure 1 summarizes the model, and we describe it as follows. We design the model based on a network traffics dataset. First, the dataset is extracted and analyzed based on relevant fields or features (such as protocol type, service, flag, src.bytes). After that, the fields' information that is extracted is pre-processed and transformed by encoding and normalization, then storing the results. We use both the training data traffic and testing traffic which contains labels indicating malicious or normal, where the labels in the testing traffic are used to check accuracy. We consider only evasion attacks; probe and teardrop attack. Afterward, the training data are pass to the training algorithm for training, which is used to construct decision tree IDS.

The attack ML pipeline is shown in Figure 1. It contains the individual components required to generate sample evasion attacks.

Fig. 1. The proposed approach and the stage of the training pipeline

Intrusion Detection System: We choose to use a DT-based IDS, the choice of using a DT for the IDS is because they are highly interpretable, and as such, results can easily be analyzed. In the model, the DT-based IDS provides the core function of classifying the traffic samples. Moreover, it is used in the attack generation pipeline as a feedback loop to gauge the fitness of individual samples. To train and test the model, we used a regular train test split. In particular, we test and load the datasets into Pandas Dataframe. We create an attack label based on a binary column with regards to whether the sample is an attack that is being trained on. So, if the sample is an attack other than the attack being investigated, the column's value will be 0. We drop the protocol type, service, and flag columns in both the training and testing sets and also the column for attacks from the training dataset. We use only the attack label column for the training. Furthermore, we take the processed data from the training traffic into the DT model for training.

Genetic Algorithm: The GA is the second key component used in the attack generation pipeline. It is responsible for producing the final attack samples. Based on GA, we design chromosome representation, a decoding procedure, genetic operators, constraints, and genetic representation of the solution space. In particular, in order to represent the solutions, we consider each feature variable within a dataset as a single gene.

In order to ensure that a sample is not changed in excess, causing it to no longer be a valid attack of the type being investigated, we added restrictions to limit which parameters can be mutated during the mutation phase. This ensures that the algorithm does not change fields that should not be changed for an attack type. The fields constrained were protocol type, service, and flag. These will ensure that the attack remains consistent with the attack type being generated. We use a fitness function to evaluate the quality of individual samples based on a specific goal. Specifically, our goal is to produce samples that were classified as benign by the DT-based IDS, but the characteristics of their data show that they are attacks. Hence, we achieve this by generating benign samples that are as similar to a known attack as possible, while still ensuring that they are classified as benign by the IDS.

$$f = \begin{cases} 2 * (1/d), & \text{If the sample is classified as benign} \\ (1/d), & \text{If the sample is classified as an attack} \end{cases}, \quad (1)$$

The deviation is defined as the sum of the difference in each feature variable from the given sample to the attack sample that seeded the algorithm. It is calculated by Equation (2), where s represents the attack sample that seeded the attack, g is the given sample that deviation is being calculated for, and n is index of the feature variable.

$$d = \sum_{n=1}^{41} abs(s_n - g_n) \quad (2)$$

The scalar value used as part of the benign sample fitness function is present to ensure that benign samples are more favored by the algorithm than attack samples. The inverse of deviation is used to ensure that deviation is minimized. Furthermore, we produce the initial population of the algorithm by taking the seed attack and breeding it with itself to produce the initial population of the required population size of 120. In order to breed each generation, we use the breeding function. This function takes in two samples, chosen randomly from the current population, and produces a new sample based on the following steps:

- With equal probability, produce a new sample by picking each feature variable from either parent 1, or parent 2 and using it as part of the new sample.
- Add genetic mutation on a gene by gene bases by sampling a number between 0 and 100 for each gene (feature variable) of the sample. If this number is less than the genetic mutation percentage, then mutate the gene.
- Mutate each gene as required by picking a new value for that gene that is within the maximum and minimum value for that feature variable. This new value is chosen randomly with equal distribution across the maximum and minimum value.

4 Experiments & Analysis

Dataset and data processing: We choose to use the NSL-KDD dataset [18] which is publicly available for researchers (and, it is also well labeled).

Attack generation: We summarize the steps taken by the attack pipeline as follows and then discuss them afterward.

- Train a DT based IDS on the provided attack types.
- From the dataset, select a seed attack to be used by the algorithm. The seed attack is a random sample from the dataset which is the specific attack type being investigated.
- Start the genetic algorithm using the produced IDS and seed attack.
- Run the genetic algorithm for 20 generations. Each generation contains 120 samples, with the fittest 30 of those samples (As evaluated by the algorithm's fitness function) moving to the next generation.
- Final 30 fittest samples are collected and classified. Here, the fittest sample is considered the best candidate for an evasion attack sample as it is most similar to the seed attack.

4.1 Analysis of results

To generate quality results on the NSL-KDD dataset, we conduct hyper-parameter tuning to find the optimal operating conditions for the algorithm. We explain each of them as follows:

In order to evaluate the sensitivity and effect of the genetic mutation variable on the produced attacks, we vary the genetic mutation variable from values 0% to 50%, but kept other parameter fixed. For each mutation value used in the experiment, we record the maximum and minimum fitness function values, as well as the number of attack samples and benign samples produced when we run the algorithm. The results are shown in Table 1.

Table 1. Genetic Mutation Testing Results

Mutation Percentage	Max fitness value	Min fitness value	Samples classified as an attack	Samples classified as benign
0	79.985	79.985	10	0
5	38.366	37.579	10	0
10	44.574	3.160	9	1
15	3.869	0.947	7	3
20	1.013	0.528	4	6
25	0.608	0.406	1	9
30	0.454	0.330	0	10
35	0.427	0.285	2	8
40	0.348	0.240	0	10
45	0.373	0.217	1	9
50	0.301	0.195	0	10

As can be seen from the results, the genetic mutation percent variable has a linear effect on the number of benign samples produced as well as the minimum and maximum value for fitness. However, we see that the fitness value later converges (with minimum deviation) for any given seed attack, where there is a maximum fitness value which depends on the attack seed used in the algorithm. Furthermore, the results show that higher mutation percentages produces more benign samples (which we consider a form of over-fitting). This is due to the fact that a high mutation percentage means each sample mutates very frequently, leading which lead it to differ from the original attack sample vastly. Therefore, it is best to keep the mutation values low. Thus, we reasonable choose to use 18% percent, as it sits between the 15% and 20% values which is where the optimal performance of the algorithm appears to occur.

The iterations parameter defines how many generations the GA runs for. In this section, we increase the number of generations/iterations from from 10 to 100 in steps of 10, to observe their effect on the final output generated. To do this, we kept the genetic mutation percent, samples per iteration, and offspring number fixed at 20%, 20, and 10 respectively.

We observed that it is better to use a low number of generations. And we choose to use 20 generations to maximize time when running the algorithm in the performance analysis.

The number of offspring and fittest offspring parameters define how many offspring to breed in each generation and how many of those bred move onto the next generation for breeding, respectively. To verify the effect of these two parameters and their interaction with each other, we conduct two different experiments. First, we fix the ratio between the two parameters, and then incrementally change the values. we set and use the ratio using Equation (3). Where n is the total number of offspring produced in each round, and f is the fittest offspring that moved through to the next round. In summary, in each generation, half of the offspring are killed off and half move through to be breed.

$$n = 2 * f \quad (3)$$

Furthermore, we verify the effect of the ratio between the two values. To do this, we fix the number of fittest offspring as 30, and then increase the total number of offspring per generation incrementally using Equation (4). From the equation, n is the total number of offspring produced, while the value i ranges from 2 to 10. We show the results in Table 2.

$$n = i * 30 \quad (4)$$

Table 2. Ratio Between Offspring Produced and Fittest Offspring Testing Results

Ratio (i)	Max fitness value	Min fitness value	Samples classified as an attack	Samples classified as benign
2	1.715	0.567	12	18
3	1.470	0.641	12	18
4	50.0	0.908	20	10
5	50.0	0.938	21	9
6	8.377	0.987	18	12
7	6.538	1.037	18	12
8	2.174	1.0	21	9
9	5.009	1.218	20	10

Table 2 shows that the ratio in terms of the number of generated offspring and fittest offspring vs samples classified as attacks. The results show that samples classified as attacks slightly increase with the ratios, but the sample classified as benign did not show any effect in terms of the ratio. Similarly, the maximum fitness value did not show any relationship with changes in the ratios used. However, the minimum fitness value increases with an increase in the ratio used.

4.2 Algorithm Performance

Based on the experiments conducted from the previous section, we use the following hyperparameter values to analyze the algorithm performance.

8 Mogg et al.

- Mutation Percentage - the percent chance that an individual feature variable mutates in any given offspring: 18%
- Iterations / Generations - The number of iterations the algorithm is run: 20
- Offspring - The total number of offspring to produce at each iteration: 120
- Fittest Offspring - the total number of offspring to live and breed the next generation: 30

The results in Figure 2 provide a high-level view of the performance of the GA for different attacks in terms of the number of attack samples, the number of benign samples, and the evasion rate. It is noted that the evasion rate is still a potential evasion rate as the benign samples have not been validated in real attacks, and only the fittest sample has been analyzed in a later section for the individual analyses.

Fig. 2. Algorithm Performance

In Figure 2, we included the analysis of other attack types as well to provide a general view on the performance of the algorithm with respect to the other attack types, which the hyperparameters were not tuned for. From the results, we may see that there is a lower evasion rate (i.e., 49.4% and 43.3%) for the Teardrop and Nmap attacks respectively, compared to 87.3% and 85.3% for Neptune and Loadmodule. However, upon analysis of the generated results for the other attacks (i.e., Neptune and Loadmodule), we observed that the higher evasion rate occurs as a result of the overfitting of the produced samples. This means that while the samples produced are benign, the algorithm has produced samples that differ significantly from the actual seed attack used. Whereas, in the teardrop and Nmap attacks, we observe that the samples that are generated are strong candidates for evasion attacks (this is shown in the next section), though they have a less number of benign samples produced. However, this can be reduced by tuning the hyperparameters for the specific attack sample being considered.

4.3 Attack Samples

In order to generate a more accurate result for the teardrop and Nmap attack, we perform experiments based on only one attack per time (i.e., only a single attack generation pipeline run is being considered for each experiment). This will allow more detailed analysis as only a single generated sample with a single seed sample and IDS is being analyzed. In each experiment, we run the algorithm many times to select the single pipeline run data that was to be used as part of the analysis. Then the average sample is computed and selected. This *average sample* is not a statistical average but it is based purely on the observation of the types of samples seen over the testing of the algorithm, and by picking a suitable average sample. Another approach that could have been taken would have been to run the algorithm N times and select the best result of the N algorithm runs - with best being defined as picking the run that produced the overall fittest sample, however, this does not accurately reflect the algorithm's usual output and as such, this method was avoided for this analysis.

Teardrop attack Figure 3 shows the DT for the teardrop attack. From the decision tree classifier, it can be seen that the algorithm has successfully altered the values that the decision tree uses as part of its boundary decisions in order to produce a sample similar to the seed attack, but instead of being classified as an attack, it is classified as benign. This process can be seen from the following differences between the seed sample and the generated sample:

- The wrong fragments flag in the produced sample has a value of 2, which is equal to the value at that decision boundary, and as such the left node is selected.
- The wrong fragments value is still greater than the 0.5 value at this decision boundary and as such the right node is selected.
- At the final node, the number of source bytes used in the generated sample is analyzed. The cutoff value at this node is 754, and as such the right node is again selected resulting in a benign classification.

One of the advantages of having a highly interpretable IDS is the fact that analysis along the decision boundaries of the IDS can be observed and inspected. This interpretability also allows a comparison of the generated sample to the ideal sample for a single path taken along the IDS. The ideal sample is one that has values exactly corresponding to the decision boundary of the IDS. This is considered an ideal sample as it will give a value closest to where the IDS considers the cutoff for an attack, which means that the sample may still be an attack itself as it will be very similar to a sample classified as an attack, but simply deviates enough to avoid attack classification.

For this produced sample, it can be seen that it correctly alters the wrong fragment feature variable to be in line with the ideal sample value of 2. In the context of a teardrop attack, this value is quite sensible as the aim of the teardrop attack is to send lots of wrong fragments to a host system. Any less than 2 wrong fragments and by definition the attack is no longer a teardrop attack as only a single wrong fragment is sent. The other value used to classify this generated

Fig. 3. Decision Tree for teardrop attack classification

simple is the source bytes feature variable. The source bytes feature variable is not in line with the ideal sample value of 745 and deviates substantially from this value with a value of 155268078 bytes. In the context of a teardrop attack, this could make logical sense as sending lots of bytes leads to many packets being sent. The validation of these generated samples and their values in the context of specific attacks is however out of scope for this paper.

NMAP attack Using the same process as the teardrop attack, we analyze the benign samples. We discuss the decision steps that classify the samples as benign as follows.

- dst host srv diff host rate error for the produced sample is equal to zero, so move left at the first decision boundary as this is less than 0.245
- dst host same src port rate is equal to 1 which is greater than the 0.785 value for the current decision node, so move right.
- dst host error rate is equal to 1 which is greater than 0.73, so again move right.
- Finally, dst host same srv rate is equal to 1 which is again greater than the decision value of 0.525, leading to a classification of benign.

On the DT, we observed that the first feature variable is altered (the dst host srv diff host rate error) and its optimal value for benign classification was 0.245, while the sample is having a value of 0, giving a difference of 0.245. Applying this same logic to all other variables, we get a total deviation sum of 1.25 out of a max possible deviation sum of 2.795, giving approx a difference of 44% from the optimal value. This is only a deviation percentage for the feature variables that were used to classify the sample, and by looking at the produced sample across the rest of the feature variables, it can be seen it is quite similar to the seed sample. In the context of the attack itself, it can be seen that the produced sample is still viable as an attack. The feature variable in the classification of the sample is the fact that the dst host same srv rate value of the produced sample is 1, compared to the seed attacks value of 0.06. This leads to the sample being

classified as benign. Since the dst host same srv rate feature variable identifies the percent of connections that were too different services, this means that all the connections to the dest server, identified by the feature variable dst host count, were too different services. This is logically sound in a probe where an attacker is attempting to identify what services are running on a given server, and as such the sample seems viable.

Also, the measure of deviation and use of a single seed sample can be considered a limitation of the GA pipeline. Since a single seed sample is used as part of the fitness function, all samples are limited to attempting to minimize the deviation from this sample. However, this may not be the most optimal way of producing samples as attacks in themselves can differ substantially across a single attack type. By using some aggregated method to seed the algorithm, improved attack samples may be able to be produced.

In the future, we will address these limitations and also implement real-time online IDS detection. We will also compare the DT and other machine learning approaches based on the GA.

5 Conclusion

In this paper, we have developed a DT-based-IDS. Based on the IDS and NSL-KDD dataset, we have investigated the practicability of generating interpretable evasion attacks against IDSes using genetic algorithms. In addition, we have proposed a generalized attack pipeline that allows the generation of evasion samples from a dataset. We demonstrated the feasibility of the proposed scheme and the results showed that the new genetic-based feature selection algorithm proposed is helpful in identifying important features needed to classify attacks from incorrectly classified benign samples. Besides, our experimental results showed attacks that similar to a given seed attack that has been classified as benign for both the teardrop and Nmap attack types.

References

1. H. Alqahtani, I. H. Sarker, A. Kalim, S. M. M. Hossain, S. Ikhlaq, and S. Hossain. Cyber intrusion detection using machine learning classification techniques. In *COMS2*, pages 121–131. Springer, 2020.
2. R. Chapaneri and S. Shah. Detection of malicious network traffic using convolutional neural networks. In *2019 ICCCNT*, pages 1–6. IEEE, 2019.
3. H. Debar, M. Dacier, and A. Wespi. Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31(8):805–822, 1999.
4. G. R. Harik, F. G. Lobo, and D. E. Goldberg. The compact genetic algorithm. *IEEE transactions on evolutionary computation*, 3(4):287–297, 1999.
5. B. Ingre, A. Yadav, and A. K. Soni. Decision tree based intrusion detection system for nsl-kdd dataset. In *International Conference on Information and Communication Technology for Intelligent Systems*, pages 207–218. Springer, 2017.
6. G. Karatas, O. Demir, and O. K. Sahingoz. A deep learning based intrusion detection system on gpus. In *2019 11th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, pages 1–6. IEEE, 2019.

12 Mogg et al.

7. M. Pawlicki, M. Choraś, and R. Kozik. Defending network intrusion detection systems against adversarial evasion attacks. *Future Generation Computer Systems*, 2020.
8. P. Porras. Directions in network-based security monitoring. *IEEE Security & Privacy*, 7(1):82–85, 2009.
9. E. Quiring, D. Klein, D. Arp, M. Johns, and K. Rieck. Adversarial preprocessing: Understanding and preventing image-scaling attacks in machine learning. In *USENIX Security 20*, 2020.
10. Y. Ren, Q. Zhou, Z. Wang, T. Wu, G. Wu, and K.-K. R. Choo. Query-efficient label-only attacks against black-box machine learning models. *Computers & Security*, 90:101698, 2020.
11. M. Roopak, G. Y. Tian, and J. Chambers. Deep learning models for cyber security in iot networks. In *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0452–0457. IEEE, 2019.
12. U. Sabeel, S. S. Heydari, H. Mohanka, Y. Bendhaou, K. Elgazzar, and K. El-Khatib. Evaluation of deep learning in detecting unknown network attacks. In *2019 SmartNets*, pages 1–6. IEEE, 2019.
13. T. Saranya, S. Sridevi, C. Deisy, T. D. Chung, and M. A. Khan. Performance analysis of machine learning algorithms in intrusion detection system: A review. *Procedia Computer Science*, 171:1251–1260, 2020.
14. I. H. Sarker, Y. B. Abushark, F. Alsolami, and A. I. Khan. Intrudtree: A machine learning based cyber security intrusion detection model. *Symmetry*, 12(5):754, 2020.
15. C. Sinclair, L. Pierce, and S. Matzner. An application of machine learning to network intrusion detection. In *Proceedings 15th Annual Computer Security Applications Conference (ACSAC’99)*, pages 371–377. IEEE, 1999.
16. S. S. S. Sindhu, S. Geetha, and A. Kannan. Decision tree based light weight intrusion detection using a wrapper approach. *Expert Systems with applications*, 39(1):129–141, 2012.
17. G. Stein, B. Chen, A. S. Wu, and K. A. Hua. Decision tree classifier for network intrusion detection with ga-based feature selection. In *Proceedings of the 43rd annual Southeast regional conference-Volume 2*, pages 136–141, 2005.
18. M. Tavallaei, E. Bagheri, W. Lu., and A. A. Ghorbani. A detailed analysis of the kdd cup 99 data set. *IEEE CISDA*, 2009.
19. K. Ullah, I. Rashid, H. Afzal, M. M. W. Iqbal, Y. A. Bangash, and H. Abbas. Ss7 vulnerabilities—a survey and implementation of machine learning vs rule based filtering for detection of ss7 network attacks. *IEEE Communications Surveys & Tutorials*, 22(2):1337–1371, 2020.
20. K. R. Vigneswaran, R. Vinayakumar, K. Soman, and P. Poornachandran. Evaluating shallow and deep neural networks for network intrusion detection systems in cyber security. In *ICCCNT*, pages 1–6. IEEE, 2018.
21. J. Yang and V. Honavar. Feature subset selection using a genetic algorithm. In *Feature extraction, construction and selection*, pages 117–136. Springer, 1998.