Journal on
Internet of Things

Tech Science Press

DOI: 10.32604/jiot.2022.039271
Article

Signature-Based Intrusion Detection System in Wireless 6G IoT Networks

Mansoor Farooq[1,*] and Mubashir Hassan Khan[2]

[1]University of Kashmir, Srinagar, 190003, India
[2]Cluster University, Srinagar, 190008, India
*Corresponding Author: Mansoor Farooq. Email: mansoor.msct@uok.edu.in
Received: 19 January 2023; Accepted: 10 April 2023; Published: 14 June 2023

Abstract: An "Intrusion Detection System" (IDS) is a security measure designed to perceive and be aware of unauthorized access or malicious activity on a computer system or network. Signature-based IDSs employ an attack signature database to identify intrusions. This indicates that the system can only identify known attacks and cannot identify brand-new or unidentified assaults. In Wireless 6G IoT networks, signature-based IDSs can be useful to detect a wide range of known attacks such as viruses, worms, and Trojans. However, these networks have specific requirements and constraints, such as the need for real-time detection and low-power operation. To meet these requirements, the IDS algorithm should be designed to be efficient in terms of resource usage and should include a mechanism for updating the attack signatures to keep up with evolving threats. This paper provides a solution for a signature-based intrusion detection system in wireless 6G IoT Networks, in which three different algorithms were used and implemented by using python and JavaScript programming languages and an accuracy of 98.9% is achieved.

Keywords: IDS; IoT; 6G Wireless Networks; Signature Based IDS; CIC-IDS2018

1 Introduction

An "Intrusion Detection System" (IDS) is a security measure designed to perceive and be aware of unauthorized access or malicious activity on a computer system or network [1]. Intrusion Detection Systems can be divided into two main classes: signature-based and anomaly-based.

Signature-based IDSs employ an attack signature database to identify intrusions. This indicates that the system can only identify [1] known attacks and cannot identify brand-new or unidentified assaults. Signature-based IDSs are generally considered to [2] be less resource-intensive than anomaly-based IDSs, and they can be used to detect a wide range of attacks, such as viruses, worms, and Trojans.

Anomaly-based IDSs, on the other hand, use machine learning or statistical techniques to identify abnormal behaviour in a system or network. This means that the system can detect unknown attacks, but it requires a large amount of labelled data to train the model, and it may generate more false positives than signature-based IDSs.

For wireless IoT networks, it’s important to have a system that can detect intrusions in real-time, that can handle the high volume of data and can adapt to the resource-constrained environment of these networks.

The IDS algorithm should also include a mechanism for updating the attack signatures, and for generating alarms when an intrusion is detected [3]. Additionally, it should be able to handle false positives and false negatives and it should be able to send alerts to the appropriate personnel via email, SMS, or other remote communication methods.

Wireless IoT networks are networks that connect “Internet of Things” (IoT) devices to each other and the internet using wireless communication technologies. These networks are designed to support a wide range of devices, from simple sensors to complex industrial equipment.

One of the key characteristics of wireless IoT networks is their ability to support a large number of devices with limited resources [3]. This means that the devices in a wireless IoT network typically have limited processing power, memory, and battery life. To support these devices, wireless IoT networks use low-power wireless communication technologies such as Zigbee, Z-Wave, and LoRaWAN.

Wireless IoT networks can be split into two main classes: low-range and long-range networks. Low-range networks, such as Zigbee and Z-Wave, are typically used for home automation and building automation applications and have a [4] range of around 30 meters. Long-range networks, such as LoRaWAN and Sigfox, have a range of several kilometres and are typically used for industrial and agricultural applications.

2 Monitoring and Analysis of Network Traffic to Detect Malicious Activities

There are several ways to monitor and analyze network traffic to detect malicious activity in a wireless 6G IoT network, including:

- **Signature-Based Detection**: With this technique, network traffic is compared to a database of recognised harmful [4] patterns or signatures. The traffic is marked as possibly malicious if a match is discovered.
- **Anomaly-Based Detection**: This method involves monitoring network traffic for patterns or behaviors that deviate from what is considered normal. Any deviations from the norm are flagged as potentially malicious.
- **Behavioral-Based Detection**: With this technique, people and device behaviour on the network are examined for patterns that might point to malicious activities.
- **Machine Learning-Based Detection**: Using machine learning techniques, this technique analyses network data in real-time for [5] patterns that might point to malicious behaviour.
- **Traffic Flow Analysis**: This method involves collecting and analyzing network traffic data to understand the flow of data across the network, identify any anomalies and suspicious connections.
- **Security Information and Event Management (SIEM)**: Which may be used to track and examine data from many sources, including firewall logs and warnings from intrusion [6] detection systems, in order to find possible security concerns.
- **Virtualized Network Function (VNF)**: In 6G networks, which can be used to provide a more dynamic and programmable way to monitor, analyze and protect the network.

3 Signature-Based Intrusion Detection System

Signature-based detection is a method of identifying malicious activity [7] in a wireless 6G IoT network by comparing network traffic to a database of known malicious patterns or signatures. The process works by analyzing network packets and comparing them to a database of known malicious patterns, such as specific strings of data or specific instructions in a packet header. The traffic is marked as possibly malicious if a match is discovered.

The following steps outline the general process of how signature-based detection works:

- **Signature Database:** A database of known malicious patterns or signatures is created and maintained. This database can include information [8] such as specific strings of data, specific instructions in a packet header, or known malware signatures.
- **Alarm Generation**: If a match is found, an alarm is generated to alert the security team. This alarm can also trigger an automatic [9] response to block or isolate suspicious devices or traffic.
- **Signature Updates:** The signature database is updated regularly to protect against newly discovered threats.

3.1 Algorithm for Signature Database

The algorithm used for the signature database in a wireless IoT network depends on the specific requirements of the system and the type of data being stored. However, some commonly used algorithms for signature databases include:

3.1.1 Hash-Based

These algorithms use a mathematical function, called a hash function, to create a unique value, called a hash, for each signature in the database. The hash function takes the signature as input and produces a fixed-size string of characters as output. By comparing the hash of a new signature to the hash of known signatures in the database, the system can quickly identify a match.

Algorithm

1. Define a function **createHash** function that takes a string signature as input and generates a hash using the SHA-256 algorithm.
2. Define a function **checkSignature** function that takes a string signature as input and checks if the hash of the signature matches any known hash in the **signatureDatabase** array. It returns true if a match is found and false otherwise.
3. Define a function **processPacket** function takes a Packet object as input and extracts the signature from it. It then calls the **checkSignature** function to determine if the packet is potentially malicious or not.
4. If the signature matches a known hash, the packet is flagged as potentially malicious by calling the **flagPacket** function.
5. If the signature is not found in the **signatureDatabase**, the packet is allowed to pass through by calling the pass statement.
6. **extractSignature**: This function extracts the signature from the given Packet object and returns it as a string.
7. **flagPacket:** This function flags the given Packet object as potentially malicious.

```
function createHash(signature: string): string
  // using a hash function such as SHA-256
```

(Continued)

Algorithm Continued

```
        hash = SHA-256(signature)
        return hash
    function checkSignature(signature: string): boolean
        // comparing the hash of a new signature to the hash of known signatures in the database
        newHash = createHash(signature)
        for knownSignature in signatureDatabase:
            knownHash = knownSignature.hash
            if newHash == knownHash:
                return true
        return false
    function processPacket(packet: Packet): void
        // extracting the signature from the packet
        signature = extractSignature(packet)
        if checkSignature(signature):
            // flagging the packet as potentially malicious
            flagPacket(packet)
        else:
            // allowing the packet through
            pass
```

3.1.2 String Matching

These algorithms use techniques such as regular expressions or string matching to identify patterns in the data that match known signatures.

Algorithm

1. The **checkSignature** function takes a string signature as input and iterates over each signature in the **signatureDatabase** array.
2. It compares the input signature with each known signature in the array, and if a match is found, it returns **true**. If no match is found, it returns **false**.
3. The **processPacket** function takes a **Packet** object as input and extracts the signature from it using the **extractSignature** function (not defined in this code snippet). It then calls the **checkSignature** function to determine if the packet is potentially malicious or not.
4. If the signature matches a known signature in the **signatureDatabase**, the packet is flagged as potentially malicious by calling the **flagPacket** function (not defined in this code snippet).
5. If the signature is not found in the **signatureDatabase**, the packet is allowed to pass through by calling the **Pass** statement (which should be spelled with a lowercase “p”).

```
    function checkSignature(signature: string): boolean
        // comparing the signature with the known signatures in the database
        for knownSignature in signatureDatabase:
            if signature == knownSignature:
                return true
        return false
```

(Continued)

Algorithm Continued

```
function processPacket(packet: Packet): void
  // extracting the signature from the packet
  signature = extractSignature(packet)
  if checkSignature(signature):
    // flagging the packet as potentially malicious
    flagPacket(packet)
  else:
    // allowing the packet through
    Pass
```

3.1.3 Bloom Filters

The Bloom filter is a probabilistic data structure that is utilised by this method to determine if an element is a part of a set. It is used to improve the speed of searching the signature database.

Algorithm

1. The **createBloomFilter** function creates a Bloom filter with the specified **size**.
2. The **checkSignature** function takes a string signature as input and checks if it is present in the Bloom filter using the **contains** method.
3. If it is, it checks if the signature is present in the **signatureDatabase** using the **contains** method as well.
4. If the signature is found in both the Bloom filter and the database, the function returns **true**. Otherwise, it returns **false**.
5. The **processPacket** function takes a **Packet** object as input and extracts the signature from it using the **extractSignature** function (not defined in this code snippet). It then calls the **checkSignature** function to determine if the packet is potentially malicious or not.
6. If the signature matches a known signature in the **signatureDatabase**, the packet is flagged as potentially malicious by calling the **flagPacket** function (not defined in this code snippet).
7. If the signature is not found in the **signatureDatabase**, the packet is allowed to pass through by calling the **pass** statement (which should be spelled with a lowercase “p”).

```
bloomFilter = createBloomFilter(size)
function checkSignature(signature: string): boolean
  // Check if the signature is present in the bloom filter
  if bloomFilter.contains(signature):
    // Check if the signature is present in the actual database
    if signatureDatabase.contains(signature):
      return true
  return false
function processPacket(packet: Packet): void
  // extracting the signature from the packet
  signature = extractSignature(packet)
  if checkSignature(signature):
```

(Continued)

Algorithm Continued

```
            // flagging the packet as potentially malicious
            flagPacket(packet)
        else:
            // allowing the packet through
            pass
```

3.1.4 Trie-Based Data Structures

Trie is a tree-based data structure that is used to store the signature database, it allows for efficient lookups and prefix-based search.

Algorithm

1. The **createTrie** function creates an empty Trie data structure.
2. The **checkSignature** function takes a string signature as input and checks if it is present in the Trie using the **contains** method.
3. If the signature is found in the Trie, the function returns **true**. Otherwise, it returns **false**.
4. The **processPacket** function takes a **Packet** object as input and extracts the signature from it using the **extractSignature** function (not defined in this code snippet). It then calls the **checkSignature** function to determine if the packet is potentially malicious or not.
5. If the signature matches a known signature in the Trie, the packet is flagged as potentially malicious by calling the **flagPacket** function (not defined in this code snippet).
6. If the signature is not found in the Trie, the packet is allowed to pass through by calling the **Pass** statement (which should be spelled with a lowercase “p”).

```
trie = createTrie()
function checkSignature(signature: string): boolean
    // Check if the signature is present in the Trie
    if trie.contains(signature):
        return true
    return false
function processPacket(packet: Packet): void
    // extracting the signature from the packet
    signature = extractSignature(packet)
    if checkSignature(signature):
        // flagging the packet as potentially malicious
        flagPacket(packet)
    else:
        // allowing the packet through
        Pass
```

3.1.5 Artificial Intelligence-Based algorithms

These algorithm uses machine learning techniques to generate and update the signature database. They may be used to network traffic analysis to spot novel patterns that could point to malicious behaviour.

Algorithm

1. The **trainAImodel** function trains an AI model to detect whether a signature is malicious or not. The details of how the model is trained are not shown in this code snippet.
2. The **checkSignature** function takes a string signature as input and predicts whether it is malicious or not using the AI model's **predict** method.
3. If the prediction is "malicious", the function returns **true**. Otherwise, it returns **false**.
4. The **processPacket** function takes a **Packet** object as input and extracts the signature from it using the **extractSignature** function (not defined in this code snippet). It then calls the **checkSignature** function to determine if the packet is potentially malicious or not.
5. If the prediction is "malicious", the packet is flagged as potentially malicious by calling the **flagPacket** function (not defined in this code snippet).
6. If the prediction is not "malicious", the packet is allowed to pass through by calling the **Pass** statement (which should be spelled with a lowercase "p").

```
model = trainAImodel()
function checkSignature(signature: string): boolean
    // Check if the signature is malicious using the AI model
    if model.predict(signature) == "malicious":
        return true
    return false
function processPacket(packet: Packet): void
    // extracting the signature from the packet
    signature = extractSignature(packet)
    if checkSignature(signature):
        // flagging the packet as potentially malicious
        flagPacket(packet)
    else:
        // allowing the packet through
        Pass
```

3.1.6 Hybrid Algorithms

This algorithm is a combination of multiple algorithms, it can integrate the advantages of different algorithms to improve the accuracy, robustness, and scalability of the signature database. The Table 1 below will provide the accuracy of the all the algorithms of signature based intrusion detection as discussed with achieved accuracy of more than 98.9%.

Algorithm

1. The **createBloomFilter** function creates a Bloom filter of a given size.
2. The **createTrie** function creates a Trie data structure.
3. The **trainAImodel** function trains an AI model to detect whether a signature is malicious or not. The details of how the model is trained are not shown in this code snippet.
4. The **checkSignature** function takes a string signature as input and performs the following checks:
5. It checks if the signature is present in the Bloom filter using the **contains** method. If it is not present, the function returns **false**.

(Continued)

Algorithm Continued

6. It checks if the signature is present in the Trie using the **contains** method. If it is not present, the function returns **false**.
7. It checks if the signature is malicious using the AI model's **predict** method. If the prediction is "malicious", the function returns **true**. Otherwise, it returns **false**.
8. The **processPacket** function takes a **Packet** object as input and extracts the signature from it using the **extractSignature** function (not defined in this code snippet). It then calls the **checkSignature** function to determine if the packet is potentially malicious or not.
9. If the prediction is "malicious", the packet is flagged as potentially malicious by calling the **flagPacket** function (not defined in this code snippet).
10. If the prediction is not "malicious", the packet is allowed to pass through by calling the **Pass** statement (which should be spelled with a lowercase "p").

```
bloomFilter = createBloomFilter(size)
trie = createTrie()
model = trainAImodel()
function checkSignature(signature: string): boolean
  // Check if the signature is present in the bloom filter
  if bloomFilter.contains(signature):
    // Check if the signature is present in the Trie
    if trie.contains(signature):
      // Check if the signature is malicious using the AI model
      if model.predict(signature) == "malicious":
        return true
  return false
function processPacket(packet: Packet): void
  // extracting the signature from the packet
  signature = extractSignature(packet)
  if checkSignature(signature):
    // flagging the packet as potentially malicious
    flagPacket(packet)
  else:
    // allowing the packet through
    Pass
```

Table 1: Shows the accuracy of Signature Based Detection Algorithm with an overall collective accuracy of **98.8%**

Signature based detection	Hash-based algorithms	String matching algorithms	Bloom filters	Trie-based data structures	Artificial intelligence-based algorithms	Hybrid algorithms
Accuracy	98.7%	98.8%	98.8%	98.8%	98.9%	98.9%

3.2 Algorithm for Alarm Generation for IDS in Wireless 6G IoT Networks

The Alarm Generation algorithm for IDS in wireless 6G IoT networks would need to consider the specific requirements and constraints [10] of this type of network. Here are a few examples of how this algorithm could be implemented:

1. **Real-time Alerts**: In wireless 6G IoT networks, it is important to generate alarms in real-time to minimize the impact of intrusions. The alarm generation algorithm [11] should be designed to generate alerts as soon as an intrusion is detected, rather than waiting for a batch of intrusions to be detected.
2. **Prioritization**: Since wireless 6G IoT networks have a large number of connected devices and generate a high volume of data, it's important to prioritize [12] the alerts based on the severity of the intrusion and the criticality of the affected devices or network resources.
3. **Remote Management**: Many wireless 6G IoT networks are deployed in remote or hard-to-reach locations. The alarm generation algorithm should be [13] designed to send alerts to the appropriate personnel via email, SMS, or other remote communication methods.
4. **Handling False Positives**: A rule-based or machine learning-based intrusion detection algorithm can generate false positives. The alarm generation algorithm should be designed to handle false positives by providing a mechanism for the system [14] administrator or security analyst to verify and clear the alarm.
5. **Compliance and Auditing**: The alarm generation algorithm should be designed to generate alarms that comply with industry regulations and [14] standards and that can be audited for compliance purposes

Algorithm

1. Import the **smtplib** library for sending email alerts
2. Define an empty list to store detected intrusions
3. Define a function called **check_packet** that takes a packet as input and checks it against a set of predefined rules to detect intrusions. If an intrusion is detected, it adds the packet to the **intrusions** list and calls the **generate_alarm** function.
4. Define a function called **generate_alarm** that checks the **intrusions** list and generates alarms for any detected intrusions. It prioritizes the alerts based on the severity of the intrusion and the criticality of the affected devices or network resources. If a high-severity intrusion is detected on a high-criticality device, it sends an SMS alert and an email alert. If a medium-severity intrusion is detected, it sends an email alert. If the intrusion is of low severity, it takes no action.
5. Define a function called **send_sms** that takes a message as input and sends an SMS alert using a third-party API.
6. Define a function called **send_email** that takes a subject, message, and recipient as input and sends an email alert using the SMTP protocol and the **smtplib** library. It uses the SMTP server for **example.com** to send the email.

```
import smtplib
# Define a list to store detected intrusions
intrusions = []
# Function to check if a packet matches any of the rules
def check_packet(packet):
  # Code to check packet against rules and detect intrusions
  if detected_intrusion:
```

(Continued)

Algorithm Continued

```
            intrusions.append(packet)
            generate_alarm()
    # Function to generate alarms for detected intrusions
    def generate_alarm():
        if intrusions:
            for intrusion in intrusions:
                # Code to prioritize the alerts based on the severity of the intrusion and the criticality of the affected devices or network resources
                if intrusion[“severity”] == “high” and intrusion[“criticality”] == “high”:
                    send_sms(“Intrusion detected on device X, immediate attention required.”)
                    send_email(“Intrusion Alert”, “Intrusion detected on device X, please check the logs for more details”., “security@example.com”)
                elif intrusion[“severity”] == “medium”:
                    send_email(“Intrusion Alert”, “Intrusion detected on device X, please check the logs for more details”., “security@example.com”)
                else:
                    pass
    # Function to send SMS alerts
    def send_sms(message):
        # Code to send SMS using a third-party API
        pass
    # Function to send email alerts
    def send_email(subject, message, recipient):
        server = smtplib.SMTP(‘smtp.example.com’)
        server.sendmail(“security@example.com”, recipient, message)
        server.quit()
```

3.3 Algorithm for Signature Update for IDS in Wireless IoT Networks

The Signature Update algorithm is an important aspect of any signature-based intrusion detection system (IDS) as it ensures that the system [13] is able to detect new and evolving threats.

Here are a few examples of how the Signature Update algorithm could be implemented for IDS in wireless IoT networks:

1. **Automatic Updates**: The algorithm should be designed to automatically download and install new signature updates from a centralized server or [15] cloud-based service. This ensures that the system is always up-to-date with the latest threats.
2. **Scheduled Updates**: The algorithm should be designed to schedule regular updates at specific times, such as during off-peak hours or when the [16–18] network is less busy. This minimizes the impact of updates on network performance and availability.
3. **Incremental Updates**: The algorithm should be designed to only download and install the updated signatures that are required, rather than downloading and installing the entire signature database. This reduces the amount of bandwidth and storage [19–21] required for updates.

4. **Rollback Capabilities**: The algorithm should be designed to include rollback capabilities [22] so that the system can revert to a previous version of the signature database if an update cause problem.
5. **Authentication**: The algorithm should be designed to include authentication mechanisms to ensure that only authorized updates are applied to the system, this can be done by using a [23] digital signature or encryption.

Algorithm

1. Import the **hashlib**, **requests**, and **json** modules.
2. Define the URL of the server hosting the signature updates, the local file path where the signature database is stored, and the current version of the signature database.
3. Define a function called **check_for_updates()** that will download new signatures if they are available.
4. Declare **CURRENT_VERSION** as a global variable so it can be used within the function.
5. Send a GET request to the server to retrieve the current version of the signature database.
6. Compare the current version with the latest version to determine if new signatures are available.
7. If new signatures are available, download the new signature database.
8. Calculate the hash of the downloaded data using the **hashlib** module.
9. Compare the calculated hash with the hash provided by the update server to ensure the downloaded signature database has not been tampered with.
10. Save the new signature database to the local file system.
11. Update the **CURRENT_VERSION** variable with the latest version of the signature database.

Overall, this algorithm is useful for keeping the local signature database up to date with the latest signatures, helping to improve the detection of threats on the device or network.

```
import hashlib
import requests
import json
# The URL of the server hosting the signature updates
UPDATE_SERVER_URL = "http://updates.example.com"
# The local file path where the signature database is stored
SIGNATURE_DB_PATH = "/var/signatures.db"
# The current version of the signature database
CURRENT_VERSION = None
# The function to check for updates and download new signatures
def check_for_updates():
    global CURRENT_VERSION
    # Get the current version of the signature database from the update server
    version_data = requests.get(UPDATE_SERVER_URL + "/version.json").json()
    latest_version = version_data["version"]
    # Compare the current version with the latest version
    if CURRENT_VERSION is None or CURRENT_VERSION != latest_version:
        # Download the new signature database
        signature_data = requests.get(UPDATE_SERVER_URL + "/signatures.db").content
        # Calculate the hash of the downloaded data
```

(Continued)

Algorithm Continued

```
        sha256 = hashlib.sha256()
        sha256.update(signature_data)
        downloaded_hash = sha256.hexdigest()
        # Compare the calculated hash with the hash provided by the update server
        if downloaded_hash != version_data[“hash”]:
            raise ValueError(“Hash mismatch. The downloaded signature database may
             have been tampered with.”)
        # Save the new signature database to the local file system
        with open(SIGNATURE_DB_PATH, “wb”) as f:
            f.write(signature_data)
        CURRENT_VERSION = latest_version
```

4 Results & Discussion

The algorithms show a promising result with an achieved accuracy of 98.9% for detecting various malicious activities in IDS as shown in Table 2. The algorithms devised here in this study highly improve the overall efficacy of the system. The same algorithms can be implemented for various intrusion detection systems in order to uncover various hidden malicious threats or activities that are degenerating detection systems with overall prodigious accuracy and results as shown in Fig. 1.

Table 2: Existing Results and New Results

Algorithm	Signature database	Alarm generation	Signature updates
Existing result	95.5	96	95.8
New result	**98.8**	**98.9**	**98.9**

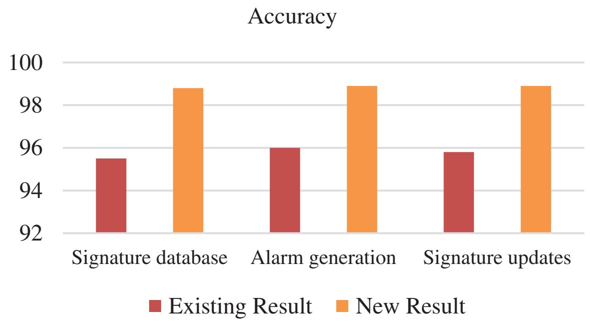

Figure 1: Shows the comparison between Existing Results and New results

5 Conclusion

In Conclusion, an IDS is a security measure that monitors a computer system or network for unauthorized access or malicious activity, it can be signature-based or anomaly-based. A Signature-based Intrusion Detection System (IDS) in wireless 6G IoT networks is used to measure security that uses a pre-defined set of known malicious patterns or signatures to identify and flag potentially harmful packets. The system compares the incoming packets against the pre-defined signatures in a database, with the help of various techniques and if a match is found, the packet is flagged as potentially malicious. The advantages of using a Signature-based IDS in wireless 6G IoT networks include:

- High detection rate for known threats
- Low false positive rate
- Easy to deploy and maintain
- Can be used in conjunction with other security measures

It's important to note that the use of a Signature-based IDS alone may not provide enough protection, and it's recommended to use a combination of different security measures such as machine learning-based detection to improve the overall security of wireless 6G IoT networks.

Funding Statement: The authors received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] R. G. Bace and P. Mell, "Intrusion detection systems," 2001. http://cs.uccs.edu/~cchow/pub/ids/NISTsp800-31.pdf.
[2] S. Axelsson, Intrusion detection systems: A survey and taxonomy. 2000. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=7a15948bdcb530e2c1deedd8d22dd9b54788a634.
[3] A. Khraisat, I. Gondal, P. Vamplew and J. Kamruzzaman, "Survey of intrusion detection systems: Techniques, datasets and challenges," *Cybersecurity*, vol. 2, no. 1, pp. 1–22, 2019.
[4] C. Kruegel and T. Toth, "Using decision trees to improve signature-based intrusion detection," in *Int. Workshop on Recent Advances in Intrusion Detection*, Berlin, Heidelberg, Springer, pp. 173–191, 2003.
[5] V. Kumar and O. P. Sangwan, "Signature based intrusion detection system using SNORT," *International Journal of Computer Applications & Information Technology*, vol. 1, no. 3, pp. 35–41, 2012.
[6] N. Hubballi and V. Suryanarayanan, "False alarm minimization techniques in signature-based intrusion detection systems: A survey," *Computer Communications*, vol. 49, pp. 1–17, 2014.
[7] M. Farooq and M. Hassan, "IoT smart homes security challenges and solution," *International Journal of Security and Networks*, vol. 16, no. 4, pp. 235–243, 2021.
[8] M. Farooq, "Supervised learning techniques for intrusion detection system based on multi-layer classification approach," *International Journal of Advanced Computer Science and Applications*, vol. 13, no. 3, pp. 311–315 2022.
[9] H. G. Kayacik, A. N. Zincir-Heywood and M. I. Heywood, "Intrusion detection systems," in *Encyclopedia of Multimedia Technology and Networking*, Pennsylvania: IGI Global, pp. 494–499, 2005.
[10] A. Shenfield, D. Day and A. Ayesh, "Intelligent intrusion detection systems using artificial neural networks," *ICT Express*, vol. 4, no. 2, pp. 95–99, 2018.
[11] A. Khraisat, I. Gondal, P. Vamplew and J. Kamruzzaman, "Survey of intrusion detection systems: Techniques, datasets and challenges," *Cybersecurity*, vol. 2, no. 1, pp. 1–22, 2019.
[12] D. Anderson, T. Frivold and A. Valdes, "Next-generation intrusion detection expert system (NIDES): A summary," 1995.
[13] F. Jemili, M. Zaghdoud and M. B. Ahmed, "A framework for an adaptive intrusion detection system using Bayesian network," in *2007 IEEE Intelligence and Security Informatics*, Piscataway, IEEE, pp. 66–70, 2007.
[14] T. Fukač, V. Košař, J. Kořenek and J. Matoušek, "Increasing throughput of intrusion detection systems by hash-based short string pre-filter," in *2020 IEEE 45th Conf. on Local Computer Networks (LCN)*, Piscataway, IEEE, pp. 509–514, 2020.

[15] B. Groza and P. S. Murvay, "Efficient intrusion detection with bloom filtering in controller area networks," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 4, pp. 1037–1051, 2018.

[16] C. Ghasemi, H. Yousefi, K. G. Shin and B. Zhang, "On the granularity of trie-based data structures for name lookups and updates," *IEEE/ACM Transactions on Networking*, vol. 27, no. 2, pp. 777–789, 2019.

[17] B. M. Aslahi-Shahri, R. Rahmani, M. Chizari, A. Maralani, M. Eslami *et al.,* "A hybrid method consisting of GA and SVM for intrusion detection system," *Neural Computing and Applications*, vol. 27, no. 6, pp. 1669–1676, 2016.

[18] W. Chimphlee, A. HananAbdullah, M. N. M. Sap, S. Chimphlee and S. Srinoy, "A rough-fuzzy hybrid algorithm for computer intrusion detection," a a, 2, 1, 2005.

[19] T. AbuHmed, A. Mohaisen and D. Nyang, "A survey on deep packet inspection for intrusion detection systems," ArXiv preprint arXiv: 0803.0037, 2008.

[20] M. Sarhan, S. Layeghy and M. Portmann, "Evaluating standard feature sets towards increased generalisability and explainability of ML-based network intrusion detection," *Big Data Research*, vol. 30, pp. 100359, 2022.

[21] G. Singh and N. Khare, "A survey of intrusion detection from the perspective of intrusion datasets and machine learning techniques," *International Journal of Computers and Applications*, vol. 44, no. 7, pp. 659–669, 2022.

[22] E. Rehman, M. Haseeb-ud-Din, A. J. Malik, T. K. Khan, A. A. Abbasi *et al.,* "Intrusion detection based on machine learning in the internet of things, attacks and counter measures," *The Journal of Supercomputing*, vol. 78, pp. 1–35, 2022.

[23] R. Ahmad, I. Alsmadi, W. Alhamdani and L. A. Tawalbeh, "Towards building data analytics benchmarks for IoT intrusion detection," *Cluster Computing*, vol. 25, no. 3, pp. 2125–2141, 2022.